From Generalized Linear Models to Neural Networks, and Back

Mario V. Wüthrich*

Version of March 2, 2020

Abstract

We present how to enhance classical generalized linear models by neural network features. On the way there, we highlight the traps and pitfalls that need to be avoided to get good statistical models. This includes the non-uniqueness of sufficiently good regression models, the balance property, and representation learning, which brings us back to the concept of the good old generalized linear models.

Keywords. Generalized linear model, GLM, neural network, regression modeling, exponential dispersion family, deviance loss, balance property, canonical link, representation learning, regularization, LASSO, claims frequency modeling.

1 Introduction

In recent years, neural networks have become increasingly popular in all sorts of regression and classification problems. In this manuscript, we present neural networks as an extension of generalized linear models (GLMs). GLMs are the state-of-the-art regression models in insurance problems. In fact, GLMs have a lot of good statistical properties which are lacking in neural network regression models. Therefore, we start here from GLMs and we carry these properties over to neural network regression models which, as a consequence, brings us back to GLMs. This is in the spirit of the book of Efron–Hastie [13], who describe statistical inference in the computer age, and it highlights the two modeling cultures discussed by Breiman [5].

Organization. As we move along this manuscript, we will provide the relevant literature. The next section discusses GLMs based on a rigorous introduction of the exponential dispersion family. This family of models builds our basis of regression modeling. In Section 3 we extend GLMs to neural network regression models, and we discuss important points that should be considered when using these models for insurance pricing. Interpretation and explanation of calibrated neural networks follows in Section 4, in particular, we discuss representation learning. In Section 5 we combine the framework of GLMs and neural networks. The benefit of this is that we receive better run times in model calibration, and that we can explicitly identify deficiencies in GLMs which, in turn, allows us to enhance these GLMs by missing model structure. Finally, in Section 6 we conclude by expanding Breiman's [5] two modeling cultures by another paradigm.

*RiskLab, Department of Mathematics, ETH Zurich, mario.wuethrich@math.ethz.ch

2 Generalized linear models

GLMs have been introduced in the seminal work of Nelder–Wedderburn [35] in 1972. Since then, they have gained great popularity within the statistical community because they provide very flexible regression and classification models. Representative in actuarial science we mention the textbooks of Frees [15], Ohlsson–Johansson [37] and Denuit et al. [10]. GLMs are based on distribution functions belonging to the exponential dispersion family (EDF), and they can be seen as a starting point of neural network regression and classification models. In the present section we recall the EDF and its properties, and we introduce GLMs for regression and classification modeling.

2.1 Exponential dispersion family

The exponential family (EF) and the EDF have been used in the work of Nelder–Wedderburn [35], and they have been studied in detail by Jørgensen [27, 28] and Barndorff-Nielsen [3]. We use Jørgensen [27, 28] as a main reference here.

2.1.1 1-dimensional exponential family

We start by choosing a σ-finite measure ν_1 on $\mathbb{R}$ (equipped with the Borel σ-field); examples include the Lebesgue measure on $\mathbb{R}$ and the counting measure on $\mathbb{N}_0$. Next, we choose a measurable function $y \mapsto \exp\{c_1(y)\}$ on the same domain as ν_1. The Laplace transform in the *canonical parameter* $\theta \in \mathbb{R}$ is given by

$$\theta \;\mapsto\; \mathcal{E}_1(\theta) = \int_{\mathbb{R}} \exp\{\theta y + c_1(y)\} d\nu_1(y). \tag{2.1}$$

The *effective domain* of the canonical parameter θ is defined by the set

$$\boldsymbol{\Theta} = \{\theta \in \mathbb{R};\; \mathcal{E}_1(\theta) < \infty\}\,. \tag{2.2}$$

There are two cases:[1] either (i) we have $\boldsymbol{\Theta} \subset \{\theta_0\}$ for some $\theta_0 \in \mathbb{R}$, or (ii) the effective domain $\boldsymbol{\Theta}$ is a possibly infinite interval with non-empty interior $\boldsymbol{\Theta}^\circ$. The first case (i) is not of interest because either $\boldsymbol{\Theta} = \emptyset$ is the empty set or the canonical parameter $\theta = \theta_0$ is perfectly known. Therefore, in the sequel, we focus on models satisfying the second case (ii) which provides us with a non-empty interior $\boldsymbol{\Theta}^\circ$ of the effective domain $\boldsymbol{\Theta}$. In this second case, we define the *1-dimensional EF* by the densities w.r.t. ν_1

$$f(y;\theta) = \exp\{\theta y - b(\theta) + c_1(y)\}, \tag{2.3}$$

for $\theta \in \boldsymbol{\Theta}^\circ$ and where we define the *cumulant function* $b(\theta) = \log \mathcal{E}_1(\theta)$, which provides the right normalization to receive a density w.r.t. ν_1.
The functions $\mathcal{E}_1(\cdot)$ and $b(\cdot)$ are infinitely often differentiable in $\boldsymbol{\Theta}^\circ$. Another property of the densities $f(\cdot;\theta)$ is that by construction their domain does not depend on the explicit choice of $\theta \in \boldsymbol{\Theta}^\circ$. This is used in the following lemma.

[1] The proof of this statement is deferred to Lemma 2.5, below.

Lemma 2.1 *Choose $\theta \in \mathbf{\Theta}^\circ$ and $r \in \mathbb{R}$ sufficiently close to zero such that $\theta + r \in \mathbf{\Theta}^\circ$. The moment generating function of $Y \sim f(\cdot;\theta)$ in r is given by*

$$\mathbb{E}\left[\exp\{rY\}\right] = \exp\left\{b(\theta + r) - b(\theta)\right\}.$$

Proof. The proof is straightforward, we refer to Lemma 7.20 in Wüthrich [51]. □

An immediate consequence of the previous lemma is that for any $\theta \in \mathbf{\Theta}^\circ$ we have

$$\mu = \mathbb{E}[Y] = b'(\theta) \qquad \text{and} \qquad \mathrm{Var}(Y) = b''(\theta).$$

The latter statement implies $b'' > 0$ (if ν_1 is not a single point measure) and, therefore, b' is invertible on $\mathbf{\Theta}^\circ$. In fact, the cumulant function b is an infinitely often differentiable and convex function on $\mathbf{\Theta}^\circ$. This allows us to construct the *canonical link function* defined by

$$g(\mu) = (b')^{-1}(\mu), \tag{2.4}$$

which links the mean $\mu = \mathbb{E}[Y]$ to the canonical parameter θ. Table 1 gives common examples of distribution functions belonging to the 1-dimensional EF.

distribution	support	$b(\theta)$	$\mathbf{\Theta}$	$g(\mu)$	link name
normal $\mathcal{N}(\mu, 1)$	$\mathbb{R}$	$\theta^2/2$	$\mathbb{R}$	μ	linear
exponential	$\mathbb{R}_+$	$-\log(-\theta)$	$\mathbb{R}_-$	$-1/\mu$	negative inverse
Bernoulli	$\{0, 1\}$	$\log(1 + e^\theta)$	$\mathbb{R}$	$\log(\mu/(1-\mu))$	logit
binomial	$\{0, \ldots, n\}$	$n \log(1 + e^\theta)$	$\mathbb{R}$	$\log(\mu/(n-\mu))$	logit
Poisson	$\mathbb{N}_0$	$\exp(\theta)$	$\mathbb{R}$	$\log(\mu)$	log

Table 1: Selected examples of distributions in the 1-dimensional EF.

2.1.2 1-dimensional exponential dispersion family

The 1-dimensional EDF is an extension of the 1-dimensional EF. We choose a family $\mathcal{C} = (\exp\{c_\kappa(\cdot)\}, \nu_\kappa)_{\kappa \in \mathcal{K}}$ of measurable functions and σ-finite measures on $\mathbb{R}$ with index set $\{1\} \subset \mathcal{K} \subset \mathbb{R}_+$. We define the effective domain $\mathbf{\Theta}$ according to (2.2) for index $\kappa = 1 \in \mathcal{K}$, and we assume that it has non-empty interior $\mathbf{\Theta}^\circ$. We say that the family $\mathcal{C}$ generates a 1-dimensional EDF if the cumulant function

$$\theta \in \mathbf{\Theta} \;\mapsto\; b(\theta) = \frac{1}{\kappa} \log \mathcal{E}_\kappa(\theta) = \frac{1}{\kappa} \log \left(\int \exp\{\theta y + c_\kappa(y)\} d\nu_\kappa(y) \right), \tag{2.5}$$

does not depend on $\kappa \in \mathcal{K}$. This motivates the densities w.r.t. ν_κ

$$f_\kappa(y;\theta) = \exp\{\theta y - \kappa b(\theta) + c_\kappa(y)\},$$

for $(\kappa, \theta) \in \mathcal{K} \times \mathbf{\Theta}$. A change of variable $y \mapsto \kappa y$ provides us with the densities

$$f(y;\theta,\kappa) = \exp\left\{\kappa(\theta y - b(\theta)) + c(y;\kappa)\right\}, \tag{2.6}$$

and log-likelihoods, respectively,

$$\log f(y;\theta,\kappa) = \kappa(\theta y - b(\theta)) + c(y;\kappa), \tag{2.7}$$

where $c(y;\kappa)$ is obtained from $\exp\{c_\kappa(\kappa y)\}d\nu_\kappa(\kappa y)$ which may have a different form depending on the explicit choice of the σ-finite measure ν_κ. Thus, the 1-dimensional EDF is defined on $\mathcal{K}\times\mathbf{\Theta}$, the domain of the random variable $Y\sim f(\cdot;\theta,\kappa)$ does *not* depend on θ, but may depend on κ. In any interior point $\theta\in\mathbf{\Theta}^\circ$ the cumulant function $b(\cdot)$ possesses all derivatives and is convex. Its moment generating function is given by the following corollary.

Lemma 2.2 *Choose $(\kappa,\theta)\in\mathcal{K}\times\mathbf{\Theta}^\circ$ and $r\in\mathbb{R}$ sufficiently close to zero such that $\theta+r/\kappa\in\mathbf{\Theta}^\circ$. The moment generating function of $Y\sim f(\cdot;\theta,\kappa)$ in r is given by*

$$\mathbb{E}\left[\exp\{rY\}\right]=\exp\left\{\kappa\left(b(\theta+r/\kappa)-b(\theta)\right)\right\}.$$

Proof. The proof follows completely analogously to the one of Lemma 2.1. □

An immediate consequence of the previous lemma is that for any $\theta\in\mathbf{\Theta}^\circ$ we have

$$\mu=\mathbb{E}[Y]=b'(\theta)\qquad\text{and}\qquad\mathrm{Var}(Y)=\frac{1}{\kappa}b''(\theta).$$

Thus, the 1-dimensional EDF extends the 1-dimensional EF in that it allows us to model an individual *dispersion parameter* $1/\kappa>0$. Table 2 provides selected examples.

distribution	support	$b(\theta)$	$\mathbf{\Theta}$	$g(\mu)$	κ
normal $\mathcal{N}(\mu,\sigma^2)$	$\mathbb{R}$	$\theta^2/2$	$\mathbb{R}$	μ	$1/\sigma^2$
gamma	$\mathbb{R}_+$	$-\log(-\theta)$	$\mathbb{R}_-$	$-1/\mu$	γ
binomial	$\{0,1/n,\dots,1\}$	$\log(1+e^\theta)$	$\mathbb{R}$	$\log(\mu/(1-\mu))$	n

Table 2: Selected examples of distributions in the 1-dimensional EDF.

From Tables 1 and 2 we observe that the binomial distribution can either be chosen as a 1-dimensional EF example or as an over-dispersed version of the Bernoulli distribution. The dispersion parameter $1/\kappa$ is treated as a known hyper-parameter (also called nuisance parameter) here, we also refer to Section 2.1.3, below.

Lemma 2.3 *Choose $(\kappa,\theta)\in\mathcal{K}\times\mathbf{\Theta}^\circ$. Assume $Y_i\overset{\text{ind.}}{\sim}f(\cdot;\theta,\kappa_i)$ for $i=1,\dots,n$. The MLE of θ is given by*

$$\widehat{\theta}^{\text{MLE}}=g\left(\frac{1}{\sum_{i=1}^n\kappa_i}\sum_{i=1}^n\kappa_iY_i\right),$$

with canonical link function $g(\cdot)=(b')^{-1}(\cdot)$.

Proof. The log-likelihood function of the observations $\boldsymbol{Y}=(Y_1,\dots,Y_n)$ is given by, see (2.7),

$$\theta\ \mapsto\ \ell_{\boldsymbol{Y}}(\theta)=\sum_{i=1}^n\kappa_i(\theta Y_i-b(\theta))+c(Y_i;\kappa_i).$$

Since b is infinitely often differentiable and convex the MLE optimization problem has the unique solution $\widehat{\theta}^{\text{MLE}}$ as stated above (this solution might lie at the boundary of $\mathbf{\Theta}$ in which case we have a degenerate model). □

Corollary 2.4 (balance property) *We make the same assumptions as in Lemma 2.3. The MLE $\widehat{\theta}^{\mathrm{MLE}}$ fulfills the balance properties for $j = 1, \ldots, n$*

$$\widehat{\mathbb{E}}\left[Y_j\right] = \frac{1}{\sum_{i=1}^n \kappa_i} \sum_{i=1}^n \kappa_i Y_i \qquad \textit{or, equivalently,} \qquad \sum_{i=1}^n \kappa_i \widehat{\mathbb{E}}\left[Y_i\right] = \sum_{i=1}^n \kappa_i Y_i,$$

where $\widehat{\mathbb{E}}$ denotes the means under the estimated distributions $Y_i \sim f(\cdot; \widehat{\theta}^{\mathrm{MLE}}, \kappa_i)$.

Proof. Using the properties of the EDF we obtain

$$\widehat{\mathbb{E}}\left[Y_j\right] = b'\left(\widehat{\theta}^{\mathrm{MLE}}\right) = b'\left(g\left(\frac{1}{\sum_{i=1}^n \kappa_i} \sum_{i=1}^n \kappa_i Y_i\right)\right) = \frac{1}{\sum_{i=1}^n \kappa_i} \sum_{i=1}^n \kappa_i Y_i.$$

This finishes the proof. □

Consequences.

- An important consequence in insurance pricing of Corollary 2.4 is that the MLE calibrated model (with canonical link) is unbiased

$$\mathbb{E}\left[\widehat{\mathbb{E}}\left[Y_j\right]\right] = \frac{1}{\sum_{i=1}^n \kappa_i} \sum_{i=1}^n \kappa_i \mathbb{E}\left[Y_i\right] = b'(\theta) = \mu, \tag{2.8}$$

  and the estimation uncertainty is given by

$$\mathrm{Var}\left(\widehat{\mathbb{E}}\left[Y_j\right]\right) = \frac{1}{(\sum_{i=1}^n \kappa_i)^2} \sum_{i=1}^n \kappa_i^2 \mathrm{Var}\left(Y_i\right) = \frac{\sum_{i=1}^n \kappa_i^2}{(\sum_{i=1}^n \kappa_i)^2} b''(\theta).$$

- Note that the unbiasedness (2.8) even holds if the model is misspecified! That is, if the data Y_i has been generated by a (true) model, say, having a cumulant function $b^\star$ different from the cumulant function b, Corollary 2.4 still provides the balance property and, henceforth, unbiasedness over the entire estimation portfolio. Thus, the balance property is a very strong and useful property because it tells us that the price level is correct even under a potentially wrong model specification.

- As can be seen from the proof of Corollary 2.4, the balance property holds true under the canonical link choice (2.4). Below, for GLMs, we are going to use other link functions and, in general, these other models fitted with MLE are not unbiased (meet the right price level), for instance, the gamma GLM with log-link function does not meet the balance property.

- Lemma 2.3 also shows that $S(\boldsymbol{Y}) = \sum_{i=1}^n \kappa_i Y_i$ is a sufficient statistics for θ, see Chapter 2 in Lehmann [30]. This justifies the functional form and parametrization chosen in the EDF (2.6). Moreover, one can prove that the estimator meets the Cramér–Rao (lower) bound if and only if it is of EF type, and the Fisher information matrix is given by

$$\mathcal{I}(\theta) = \mathbb{E}\left[\left(\frac{\partial}{\partial \theta} \log f(Y; \theta, \kappa)\right)^2\right] = -\mathbb{E}\left[\frac{\partial^2}{\partial \theta^2} \log f(Y; \theta, \kappa)\right] = \kappa b''(\theta),$$

  we refer to the last paragraph of Section 2.7 in Lehmann [30].

2.1.3 k-dimensional exponential family

The above results carry over to multi-dimensional cases. We briefly summarize these results for the EF. Choose a σ-finite measure ν_1 on $\mathbb{R}^m$ and let $c_1 : \mathbb{R}^m \to \mathbb{R}$ and $T : \mathbb{R}^m \to \mathbb{R}^k$ be measurable functions. Consider the Laplace transform in the canonical parameter $\theta \in \mathbb{R}^k$

$$\mathcal{E}_1(\theta) = \int_{\mathbb{R}^m} \exp\left\{\theta^\top T(y) + c_1(y)\right\} d\nu_1(y).$$

This allows us to define the effective domain

$$\mathbf{\Theta} = \left\{\theta \in \mathbb{R}^k;\ \mathcal{E}_1(\theta) < \infty\right\}.$$

Lemma 2.5 *The effective domain $\mathbf{\Theta}$ is convex.*

Proof. Assume that $\mathcal{E}_1(\theta_i) < \infty$ for $\theta_i \in \mathbb{R}^k$, $i = 1, 2$. Consider $\theta = \alpha\theta_1 + (1-\alpha)\theta_2$ for $\alpha \in [0, 1]$. The claim follows from Hölder's inequality

$$\mathcal{E}_1(\theta) = \int_{\mathbb{R}^m} \exp\left\{(\alpha\theta_1 + (1-\alpha)\theta_2)^\top T(y) + c_1(y)\right\} d\nu_1(y) \leq \mathcal{E}_1(\theta_1)^\alpha \mathcal{E}_1(\theta_2)^{1-\alpha}.$$

□

For $\theta \in \mathbf{\Theta}^\circ$ we define the cumulant function

$$b(\theta) = \log \mathcal{E}_1(\theta).$$

The following defines the *k-dimensional EF* density on $\mathbb{R}^m$ w.r.t. ν_1 (and the choice $c = c_1$)

$$f(y; \theta) = \exp\left\{\theta^\top T(y) - b(\theta) + c(y)\right\}. \tag{2.9}$$

The support of this exponential family does *not* depend on the explicit choice of θ.

Lemma 2.6 *Assume that the interior $\mathbf{\Theta}^\circ$ of the effective domain $\mathbf{\Theta}$ is non-empty. The moment generating function of $T(Y)$, with $Y \sim f(\cdot; \theta)$ for $\theta \in \mathbf{\Theta}^\circ$, and $r \in \mathbb{R}^k$, with $\|r\|$ sufficiently small, satisfies*

$$\mathbb{E}\left[\exp\left\{r^\top T(Y)\right\}\right] = \exp\left\{b(\theta + r) - b(\theta)\right\}.$$

Proof. Exercise. □

Corollary 2.7 *Assume that ν_1 is not concentrated in a single point and that the interior $\mathbf{\Theta}^\circ$ of the effective domain $\mathbf{\Theta}$ is non-empty. Then, $\theta \mapsto b(\theta)$ is convex, and*

$$\mathbb{E}\left[T(Y)\right] = \nabla_\theta b(\theta) \qquad \text{and} \qquad \mathrm{Var}\left(T(Y)\right) = \mathbf{H}_\theta b(\theta),$$

where ∇_θ is the gradient and $\mathbf{H}_\theta$ is the Hessian w.r.t. θ. Moreover, all moments of $T(Y)$ exist.

Proof. The existence of the moment generating function for all sufficiently small $r \in \mathbb{R}^k$ (around the origin) implies that we have a power series expansion of the moment generating function around the origin, that all moments exists, and that we can exchange the limits of the power series expansion and the moment calculations. □

Remark. If the components of $T(Y)$ are linearly independent for all $\theta \in \mathbf{\Theta}^\circ$, then $\mathbf{H}_\theta b(\theta)$ has full rank and b is strictly convex. In this case the representation is called minimal. From now on we assume that we always work with a minimal representation, see Lemma 8.1 and Corollary 8.1 in Barndorff-Nielsen [3].

Definition 2.8 (canonical link) *Assume ν_1 is not concentrated in a single point and that we have a minimal representation with an effective domain $\mathbf{\Theta}$ that has a non-empty interior $\mathbf{\Theta}^\circ$. The canonical link is defined by $g = (\nabla_\theta b)^{-1}$, thus,*

$$g\left(\mathbb{E}\left[T(Y)\right]\right) = \theta.$$

Remark. Definition 2.8 says that we can either work with the canonical parameter $\theta \in \mathbf{\Theta}$ or with the mean parameter $\mu \in \{\nabla_\theta b(\theta);\ \theta \in \mathbf{\Theta}\}$, and there is a one-to-one correspondence between these two parametrizations.

For parameter estimation we choose i.i.d. observations $Y_i \sim f(\cdot;\theta)$ where we assume that the underlying σ-finite measure ν_1 is not concentrated in a single point and where we have chosen a minimal representation with an effective domain $\mathbf{\Theta}$ that has a non-empty interior. The log-likelihood of the data $\boldsymbol{Y} = (Y_1, \ldots, Y_n)$ reads as

$$\theta \ \mapsto \ \ell_{\boldsymbol{Y}}(\theta) = \sum_{i=1}^n \theta^\top T(Y_i) - b(\theta) + c(Y_i),$$

which provides MLE

$$\widehat{\theta}^{\mathrm{MLE}} = g\left(\frac{1}{n}\sum_{i=1}^n T(Y_i)\right). \tag{2.10}$$

Corollary 2.9 (balance property) *The estimated distribution fulfills the balance property on portfolio level, i.e.*

$$\widehat{\mathbb{E}}\left[T(Y_1)\right] = \frac{1}{n}\sum_{i=1}^n T(Y_i).$$

Remarks. For $Y \sim f(\cdot;\theta)$ we define the score

$$s(\theta, Y) = \nabla_\theta \log f(Y;\theta) = \nabla_\theta \left(\theta^\top T(Y) - b(\theta) + c(Y)\right) = T(Y) - \nabla_\theta b(\theta).$$

We have $\mathbb{E}[s(\theta, Y)] = 0$ and we receive Fisher information

$$\mathcal{I}(\theta) = \mathbb{E}\left[s(\theta,Y)s(\theta,Y)^\top\right] = -\mathbb{E}\left[\mathbf{H}_\theta \log f(Y;\theta)\right] = \mathbf{H}_\theta b(\theta) = \mathrm{Var}\left(T(Y)\right).$$

In particular, the sufficient statistics $S(\boldsymbol{Y}) = \sum_{i=1}^n T(Y_i) \in \mathbb{R}^k$ for $\theta \in \mathbb{R}^k$ meets the Cramér–Rao bound.

We provide examples which expand the list of Table 1; they revisit some of the examples of Table 2 modeling the over-dispersion parameter $1/\kappa$ in the canonical parameter space $\mathbf{\Theta}$, and not as nuisance parameter.

Categorical distribution. We choose as ν_1 the counting measure on $\{1, \ldots, k\}$. Set $T(y) = (\mathbb{1}_{\{y=1\}}, \ldots, \mathbb{1}_{\{y=k-1\}})^\top$, $\theta = (\theta_1, \ldots, \theta_{k-1})^\top$, and

$$c(y) = 0, \qquad b(\theta) = \log\left(1 + \sum_{i=1}^{k-1} e^{\theta_i}\right), \qquad \mu = \nabla_\theta b(\theta) = \frac{e^\theta}{1 + \sum_{i=1}^{k-1} e^{\theta_i}},$$

with effective domain $\mathbf{\Theta} = \mathbb{R}^{k-1}$. With these choices we have (set $\theta_k = 0$)

$$f(y;\theta) = \exp\left\{\theta^\top T(y) - \log\left(1 + \sum_{i=1}^{k-1} e^{\theta_i}\right)\right\} = \prod_{j=1}^{k}\left(\frac{e^{\theta_j}}{\sum_{i=1}^{k} e^{\theta_i}}\right)^{\mathbb{1}_{\{y=j\}}}.$$

This is a $(k-1)$-dimensional EF. The canonical link is slightly more complicated. Set vectors $v = \exp\{\theta\} \in \mathbb{R}^{k-1}$ and $w = (1,\ldots,1)^\top \in \mathbb{R}^{k-1}$. This provides $\mu = \nabla_\theta b(\theta) = \frac{1}{1+w^\top v} v \in \mathbb{R}^{k-1}$. Set matrix $A_\mu = \mathbb{1} - \mu w^\top \in \mathbb{R}^{(k-1)\times(k-1)}$. The latter can be rewritten as $\mu = A_\mu v$, and since A_μ has full rank we receive canonical link

$$\theta = g(\mu) = \log(A_\mu^{-1}\mu).$$

Normal distribution $\mathcal{N}(\mu,\sigma^2)$. Choose ν_1 to be the Lebesgue measure on $\mathbb{R}$. Set $T(y) = (y, y^2)^\top$, and

$$\begin{aligned} c(y) &= -\frac{1}{2}\log(2\pi), \qquad b(\theta) = -\frac{\theta_1^2}{4\theta_2} - \frac{1}{2}\log(-2\theta_2), \\ (\mu, \sigma^2+\mu^2) &= \nabla_\theta b(\theta) = \left(\frac{\theta_1}{-2\theta_2}, (-2\theta_2)^{-1} + \frac{\theta_1^2}{4\theta_2^2}\right), \end{aligned}$$

with effective domain $\mathbf{\Theta} = \mathbb{R} \times (-\mathbb{R}_+)$. With these choices we have

$$f(y;\theta) = \frac{1}{\sqrt{2\pi}(-2\theta_2)^{-1/2}} \exp\left\{-\frac{1}{2}\frac{1}{(-2\theta_2)^{-1}}\left(y - \frac{\theta_1}{-2\theta_2}\right)^2\right\}.$$

This is a 2-dimensional EF, modeling simultaneously the mean and variance of $\mathcal{N}(\mu,\sigma^2)$.

Gamma distribution $\Gamma(\gamma, c)$. Choose ν_1 to be the Lebesgue measure on $\mathbb{R}_+$. Set $T(y) = (y, \log y)^\top$, and

$$\begin{aligned} c(y) &= -\log y, \qquad b(\theta) = \log\Gamma(\theta_2) - \theta_2\log(-\theta_1), \\ \left(\frac{\gamma}{c}, \frac{\Gamma'(\gamma)}{\Gamma(\gamma)} - \log(c)\right) &= \nabla_\theta b(\theta) = \left(\frac{\theta_2}{-\theta_1}, \frac{\Gamma'(\theta_2)}{\Gamma(\theta_2)} - \log(-\theta_1)\right), \end{aligned}$$

with effective domain $\mathbf{\Theta} = (-\mathbb{R}_+) \times \mathbb{R}_+$. With these choices we have

$$f(y;\theta) = \frac{(-\theta_1)^{\theta_2}}{\Gamma(\theta_2)} y^{\theta_2 - 1} \exp\{-(-\theta_1)y\}.$$

This is a 2-dimensional EF modeling simultaneously shape and scale parameters of $\Gamma(\gamma, c)$.

2.2 Generalized linear models

2.2.1 Linear predictors

The EF and EDF, respectively, provide homogeneous settings by assuming that the claims Y_i, $i = 1,\ldots,n$, use the same canonical parameter θ. GLMs allow us to relax this homogeneity assumption by letting the canonical parameter depend on *features* (*covariates*, *explanatory variables*, *predictor variables*), i.e. $\boldsymbol{x} \mapsto \theta = \theta(\boldsymbol{x})$ for features $\boldsymbol{x} \in \mathcal{X} \subset \{1\} \times \mathbb{R}^q$.

For simplicity, we restrict ourselves to the 1-dimensional EDF in this section. Assume we have data $\{(Y_i, \boldsymbol{x}_i, o_i)\}_{i=1}^n$, where Y_i describes the claim (is modeled by a random variable), $\boldsymbol{x}_i \in \mathcal{X} \subset \{1\} \times \mathbb{R}^q$ describes the feature information and $o_i \in \mathbb{R}$ is a given *offset* of claim $i = 1, \ldots, n$. The resulting *design matrix* $\mathfrak{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^\top \in \mathbb{R}^{n \times (q+1)}$ is assumed to have full rank $q + 1 \leq n$. In a GLM context we assume that all observations Y_i are independent and belong to the *same* 1-dimensional EDF w.r.t. $\mathcal{C} = (\exp\{c_\kappa(\cdot)\}, \nu_\kappa)_{\kappa \in \mathcal{K}}$ and, henceforth, w.r.t. the cumulant function b, see (2.5). Their means are assumed to use the *linear predictors*

$$(o_i, \boldsymbol{x}_i) \;\mapsto\; h\left(\mathbb{E}[Y_i]\right) = o_i + \boldsymbol{\beta}^\top \boldsymbol{x}_i = o_i + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle, \tag{2.11}$$

for a differentiable and strictly increasing *link function* h and *regression parameter* $\boldsymbol{\beta} \in \mathbb{R}^{q+1}$.

- At the moment the link function h is not necessarily the canonical one.
- The domain and the range of the link function may be subsets of $\mathbb{R}$, for instance, the canonical link in the gamma model $h(\mu) = g(\mu) = -1/\mu \in \mathbb{R}_- = \boldsymbol{\Theta}$ is restricted to the negative real line for $\mu \in \mathbb{R}_+$, see Table 2. This may impose restrictions for the domains of offsets, parameters and feature values on the right-hand side of (2.11).
- The first component of the feature $\boldsymbol{x}_i \in \mathcal{X} \subset \{1\} \times \mathbb{R}^q$ is identically equal to one for all claims $i = 1, \ldots, n$, and it serves the purpose of modeling the intercept component.
- The offsets $o_i \in \mathbb{R}$ are a priori differences between the claims, hence known intercepts which need not to be estimated. To simplify notation we could rewrite the right-hand side of (2.11) as
$$h\left(\mathbb{E}[Y_i]\right) = o_i + \boldsymbol{\beta}^\top \boldsymbol{x}_i = \widetilde{\boldsymbol{\beta}}^\top \widetilde{\boldsymbol{x}}_i = \langle \widetilde{\boldsymbol{\beta}}, \widetilde{\boldsymbol{x}}_i \rangle,$$
with $\widetilde{\boldsymbol{\beta}}^\top = (\boldsymbol{\beta}^\top, 1)$ and $\widetilde{\boldsymbol{x}}_i^\top = (\boldsymbol{x}_i^\top, o_i)$. We refrain from doing so.

The above assumptions imply the following distributional property

$$Y_i \overset{\text{ind.}}{\sim} f(\cdot; \theta_i, \kappa_i) = \exp\left\{\kappa_i(\theta_i y - b(\theta_i)) + c(y; \kappa_i)\right\}, \tag{2.12}$$

with canonical parameter

$$\theta_i = \theta(o_i, \boldsymbol{x}_i) = g\left(\mathbb{E}[Y_i]\right) = g \circ h^{-1}\left(o_i + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle\right) \;\overset{!}{\in}\; \boldsymbol{\Theta}. \tag{2.13}$$

Note that we require $(\overset{!}{\in})$ that the canonical parameter θ_i is in the effective domain $\boldsymbol{\Theta}$, otherwise we do not have a well-defined model. In the special case of the canonical link $h = g$, the canonical parameter is equal to the linear predictor

$$\theta_i = \theta(o_i, \boldsymbol{x}_i) = o_i + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \;\overset{!}{\in}\; \boldsymbol{\Theta}. \tag{2.14}$$

Remark. Often, one assumes $\kappa_i = w_i/\phi > 0$ for given (known) weights $w_i > 0$ and (unknown) dispersion parameter $\phi > 0$. The unknown dispersion parameter ϕ will not harm all MLE arguments below, because it cancels in the maximization problems.

2.2.2 Maximum likelihood estimation

Model (2.12)-(2.13) has a $(q+1)$-dimensional parameter $\boldsymbol{\beta}$ that we have to calibrate to the data $\{(Y_i, \boldsymbol{x}_i, o_i)\}_{i=1}^n$. The log-likelihood function is given by

$$\boldsymbol{\beta} \;\mapsto\; \ell_{\boldsymbol{Y}}(\boldsymbol{\beta}) = \sum_{i=1}^n \kappa_i(\theta_i Y_i - b(\theta_i)) + c(Y_i;\kappa_i).$$

MLE of $\boldsymbol{\beta}$ requires to solve the score equations

$$0 \;\stackrel{!}{=}\; \frac{\partial}{\partial \boldsymbol{\beta}} \ell_{\boldsymbol{Y}}(\boldsymbol{\beta}) = \sum_{i=1}^n \kappa_i (Y_i - b'(\theta_i)) \frac{\partial \theta_i}{\partial \boldsymbol{\beta}} = \sum_{i=1}^n \kappa_i (Y_i - \mu_i) \frac{\boldsymbol{x}_i}{V(\mu_i) h'(\mu_i)},$$

with variance function $\mu \mapsto V(\mu) = b''(g(\mu))$ and individual means $\mu_i = b'(\theta_i)$. This is solved by the iterated re-weighted least squares (IRLS) algorithm, we refer to McCullagh–Nelder [33] and Dobson [11].
In the case of the canonical link function $h = g$ things simplify to solving

$$0 \;\stackrel{!}{=}\; \frac{\partial}{\partial \boldsymbol{\beta}} \ell_{\boldsymbol{Y}}(\boldsymbol{\beta}) = \sum_{i=1}^n \kappa_i (Y_i - b'(\theta_i)) \frac{\partial \theta_i}{\partial \boldsymbol{\beta}} = \sum_{i=1}^n \kappa_i (Y_i - b'(\theta_i)) \boldsymbol{x}_i. \tag{2.15}$$

Corollary 2.10 (balance property) *Assume that the observations $\{(Y_i, \boldsymbol{x}_i, o_i)\}_{i=1}^n$ are independent, come from the EDF $\mathcal{C} = (\exp\{c_\kappa(\cdot)\}, \nu_\kappa)_{\kappa \in \mathcal{K}}$, and have linear predictors (2.11) under the canonical link choice $h = g = (b')^{-1}$, see (2.14). The MLE $\widehat{\boldsymbol{\beta}}^{\mathrm{MLE}}$ of $\boldsymbol{\beta} \in \mathbb{R}^{q+1}$ fulfills the balance property*

$$\sum_{i=1}^n \kappa_i \widehat{\mathbb{E}}\left[Y_i\right] = \sum_{i=1}^n \kappa_i Y_i,$$

where $\widehat{\mathbb{E}}$ denotes the mean under the estimated distribution having parameter $\widehat{\boldsymbol{\beta}}^{\mathrm{MLE}}$.

Proof. Using the properties of the EDF we obtain

$$\sum_{i=1}^n \kappa_i \widehat{\mathbb{E}}\left[Y_i\right] = \sum_{i=1}^n \kappa_i b'\left(o_i + \langle \widehat{\boldsymbol{\beta}}^{\mathrm{MLE}}, \boldsymbol{x}_i \rangle\right) = \sum_{i=1}^n \kappa_i b'\left(\widehat{\theta}_i\right).$$

The parameters $\widehat{\theta}_i = o_i + \langle \widehat{\boldsymbol{\beta}}^{\mathrm{MLE}}, \boldsymbol{x}_i \rangle$ have the property that they solve (2.15) in each equation (component) $j = 0, \ldots, q$. The zeroth component of (2.15) models the intercept and has the property that $x_{i,0} = 1$ for all $i = 1, \ldots, n$. This implies the claim. □

2.2.3 Deviance losses and predictive performance

In the previous two subsections we have described model choices and model fitting within GLMs. In this subsection we briefly like to touch upon model validation and parameter selection. Starting from classical statistics, we choose a model for the data $\{(Y_i, \boldsymbol{x}_i, o_i)\}_{i=1}^n$ which is fitted to that data using MLE. Now, we can use the whole statistical theory for model validation and parameter selection, we can perform hypothesis testing using, say, likelihood ratio tests, etc. (see for instance Lehmann [30] and Chapter 3 of Ohlsson–Johansson [37]). Breiman [5] calls this the *data modeling culture*, distinguishing it from the *algorithmic modeling culture*.
However, ultimately, we are mostly interested in the prediction accuracy of our model to forecast claims of insurance policies that we have not seen, yet. Or in other words, we are interested in

knowing how our model *generalizes to unseen data*, and we are "not directly" interested in the model itself as long as predictive performance is good. Breiman [5] calls this the *algorithmic modeling culture*, because the main focus of the modeling lies on predictive performance and less on explainable models.

To measure predictive performance one typically partitions the available data $\{(Y_i, \boldsymbol{x}_i, o_i)\}_{i=1}^n$ into a *learning data set* $\mathcal{D}$ and a *test data set* $\mathcal{T}$. The entire model selection and parameter estimation (model fitting) is performed solely on the learning data set $\mathcal{D}$, and predictive performance is evaluated on the test data set $\mathcal{T}$, which serves as the unseen data in this experimental setup. In the language of the previous subsection, we derive the MLE $\widehat{\boldsymbol{\beta}}^{\text{MLE}}$ of $\boldsymbol{\beta}$ on the learning data set $\mathcal{D}$ by studying the corresponding log-likelihood function

$$\boldsymbol{\beta} \;\mapsto\; \ell_{\mathcal{D}}(\boldsymbol{\beta}) = \sum_{i \in \mathcal{D}} \kappa_i (\theta_i Y_i - b(\theta_i)) + c(Y_i; \kappa_i), \tag{2.16}$$

and we use this estimated model to see how it performs on $\mathcal{T}$. This concept is going to be used throughout this manuscript. Remark that the balance property of Corollary 2.10 only holds on the portfolio $\mathcal{D}$ that has been used for model fitting.

The MLE of $\boldsymbol{\beta}$ is found by maximizing the log-likelihood function (2.16). This is equivalent to minimizing the corresponding *deviance loss function* which is defined by

$$\begin{aligned} \boldsymbol{\beta} \;\mapsto\; \mathcal{L}(\mathcal{D}, \boldsymbol{\beta}) &= \frac{2}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \log f(Y_i; g(Y_i), \kappa_i) - \log f(Y_i; \theta_i, \kappa_i) \\ &= \frac{2}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \kappa_i \left(g(Y_i) Y_i - b(g(Y_i)) - \theta_i Y_i + b(\theta_i) \right) \;\geq\; 0, \end{aligned}$$

where $g(\cdot) = (b')^{-1}(\cdot)$ is the canonical link. The deviance loss function compares the log-likelihood of the saturated model[2] to the log-likelihood of the selected model $\theta_i = \theta(o_i, \boldsymbol{x}_i)$, see (2.13). Since the former model has maximal degrees of freedom, the deviance loss function is always non-negative, and we have

$$\widehat{\boldsymbol{\beta}}^{\text{MLE}} = \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\text{MLE}} = \arg\max_{\boldsymbol{\beta}} \ell_{\mathcal{D}}(\boldsymbol{\beta}) = \arg\min_{\boldsymbol{\beta}} \mathcal{L}(\mathcal{D}, \boldsymbol{\beta}).$$

This provides us with the *in-sample loss*

$$\mathcal{L}(\mathcal{D}, \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\text{MLE}}) = \frac{2}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \kappa_i \left(g(Y_i) Y_i - b(g(Y_i)) - \widehat{\theta}_i^{\text{MLE}} Y_i + b(\widehat{\theta}_i^{\text{MLE}}) \right) \;\geq\; 0, \tag{2.17}$$

for canonical parameter estimate (based on $\mathcal{D}$)

$$\widehat{\theta}_i^{\text{MLE}} = g\left(\widehat{\mathbb{E}}[Y_i] \right) = g \circ h^{-1} \left(o_i + \left\langle \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\text{MLE}}, \boldsymbol{x}_i \right\rangle \right).$$

Predictive performance is measured by the *out-of-sample loss* on $\mathcal{T}$ w.r.t. MLE $\widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\text{MLE}}$

$$\mathcal{L}(\mathcal{T}, \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\text{MLE}}) = \frac{2}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \kappa_i \left(g(Y_i) Y_i - b(g(Y_i)) - \widehat{\theta}_i^{\text{MLE}} Y_i + b(\widehat{\theta}_i^{\text{MLE}}) \right) \;\geq\; 0. \tag{2.18}$$

[2]The saturated model is obtained by estimating each canonical parameter θ_i individually by the MLE $g(Y_i)$, $i \in \mathcal{D}$, see Lemma 2.3.

This out-of-sample loss $\mathcal{L}(\mathcal{T}, \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\text{MLE}})$ will be our main quantity of interest to measure predictive performance. In general, we assume that $\mathcal{D}$ and $\mathcal{T}$ are sufficiently similar because otherwise the $\mathcal{D}$-estimated model cannot generalize to $\mathcal{T}$. Moreover, we are not directly interested into the in-sample loss $\mathcal{L}(\mathcal{D}, \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\text{MLE}})$ because over-parametrized models have the tendency to over-fit to the data $\mathcal{D}$, i.e. such models do not only learn model structure (the regression function) but they also mimic the noisy (random) part of the data.
Remark that the split of the data $\{(Y_i, \boldsymbol{x}_i, o_i)\}_{i=1}^n$ into a learning data set $\mathcal{D}$ and a test data set $\mathcal{T}$ requires that we have a sufficiently large sample size to ensure that $\mathcal{D}$ is sufficiently rich. If this is not the case we cannot "waste" the test data set $\mathcal{T}$ during model fitting and, typically, we would use K-fold cross-validation as an alternative model validation method, we refer to Section 1.4.4 in [53].

2.3 Example: claims frequency modeling

2.3.1 Data

We use the data `freMTPL2freq` which is included in the R package `CASdatasets`,[3] see Charpentier [8]. This data comprises a French motor third party liability (MTPL) insurance portfolio with corresponding claim counts observed within one accounting year. For an extended empirical analysis of this French MTPL data we refer to the tutorials [14, 36, 43] of the Swiss Association of Actuaries (SAA).[4] Therefore, we refrain from copying the corresponding plots here.

Listing 1: French MTPL claims frequency data; version CASdatasets_1.0-8.

```
1  > str(freMTPL2freq)
2  'data.frame':   678013 obs. of  12 variables:
3   $ IDpol     : num  1 3 5 10 11 13 15 17 18 21 ...
4   $ ClaimNb   : int  1 1 1 1 1 1 1 1 1 1 ...
5   $ Exposure  : num  0.1 0.77 0.75 0.09 0.84 0.52 0.45 0.27 0.71 0.15 ...
6   $ Area      : Factor w/ 6 levels "A","B","C","D",..: 4 4 2 2 2 5 5 3 3 2 ...
7   $ VehPower  : int  5 5 6 7 7 6 6 7 7 7 ...
8   $ VehAge    : int  0 0 2 0 0 2 2 0 0 0 ...
9   $ DrivAge   : int  55 55 52 46 46 38 38 33 33 41 ...
10  $ BonusMalus: int  50 50 50 50 50 50 50 68 68 50 ...
11  $ VehBrand  : Factor w/ 11 levels "B1","B10","B11",..: 4 4 4 4 4 4 4 4 4 4 ...
12  $ VehGas    : Factor w/ 2 levels "Diesel","Regular": 2 2 1 1 1 2 2 1 1 1 ...
13  $ Density   : int  1217 1217 54 76 76 3003 3003 137 137 60 ...
14  $ Region    : Factor w/ 22 levels "R11","R21","R22",..: 18 18 3 15 15 8 8 20 20 12 ...
```

Listing 1 provides a short excerpt of the data. We have $n = 678'013$ insurance policies, each having a unique identifier `IDpol`. The variable `ClaimNb` describes the number of claims that each policy suffers, `Exposure` gives the time exposures, and the lines 6 to 14 of Listing 1 describe the available feature information. Before we start with regression modeling of this data we perform the same (small) data cleaning part described in the SAA tutorials [14, 36, 43].[5]
A peculiarity worth mentioning is that for roughly 75% of the policies we have an `Exposure` of less than 1 (accounting year), and the average `Exposure` is 0.5285. This seems to be a rather

[3] CASdatasets website `http://cas.uqam.ca`; the data is described on page 55 of the reference manual [7]; we use version CASdatasets_1.0-8.

[4] See `https://www.actuarialdatascience.org/`

[5] The R code is available from `https://github.com/JSchelldorfer/ActuarialDataScience`

atypical situation in general insurance. Based on further analysis of the data we believe that this high ratio of partial annual exposures is caused by mutations and the fact that policies which expire and are renewed during the accounting year are registered as two policies. For instance, a policy that is renewed at the end of March has two entries in that data set, the first one having an exposure of 0.25 and the second one an exposure of 0.75. We believe that this is unfortunate because such policies should be merged to one policy being active during the whole accounting year. However, the information in Listing 1 is not sufficient to perform this merger, and we have to work with these segregated policies. Theoretically, in a Poisson regression model segregation is not a problem because the sum of two independent Poisson random variables is again Poisson distributed, see Theorems 2.12 and 2.14 in [51]. However, the data of Listing 1 may also contain early termination of policies because of insurance claims. In that case, our model assumptions may not be fulfilled.

2.3.2 Poisson claims frequency modeling

The canonical model for claims count `ClaimNb` modeling is the Poisson model. From Table 1 we know that the Poisson model belongs to the 1-dimensional EF with canonical link function given by the log link, i.e. $\mu \mapsto \theta = g(\mu) = \log(\mu)$ for $\mu > 0$. Typically, different car drivers cannot be described by a homogeneous model because they have different skills, different driving experience, drive different cars on different roads at different day times, etc. We try to capture this heterogeneity by using a GLM which is based on individual policy information $\boldsymbol{x}_i \in \mathcal{X}$, where $\mathcal{X} \subset \{1\} \times \mathbb{R}^q$ is the $(q+1)$-dimensional feature space that describes all (potential) insurance policies.
We use the canonical link function and assume that the canonical parameter of policy i can be described by the linear predictor, see (2.14),

$$\theta_i = \theta(o_i, \boldsymbol{x}_i) = o_i + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle \ \in \ \boldsymbol{\Theta} = \mathbb{R},$$

where the offset o_i is chosen to be the logarithm of the `Exposure` of policy i, see line 5 of Listing 1, and where $\boldsymbol{\beta} \in \mathbb{R}^{q+1}$ is the unknown regression parameter to be determined.
Using the Poisson assumption with canonical link, we assume that the claims counts `ClaimNb` of all policies i are independent and distributed as

$$Y_i \overset{\text{ind.}}{\sim} \text{Poi}\left(v_i \lambda(\boldsymbol{x}_i)\right), \tag{2.19}$$

where we set for the exposure $v_i = \exp\{o_i\} = \texttt{Exposure}_i \in (0, 1]$, and

$$\mu_i = \mu(\theta_i) = b'(\theta_i) = \exp\left\{o_i + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle\right\} = v_i \exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle = v_i \lambda(\boldsymbol{x}_i);$$

the last identity defines the expected frequency $\boldsymbol{x} \mapsto \lambda(\boldsymbol{x})$ which is the volume v_i adjusted expected number of claims.

Task. The main problem to be solved is to find the regression function $\lambda(\cdot)$ such that it appropriately describes the data, and such that it generalizes to similar data which has not been seen, yet. The task of finding an appropriate regression function $\lambda : \mathcal{X} \to \mathbb{R}_+$ also includes the choice of the feature space $\mathcal{X}$ which typically varies over different modeling approaches.

We first describe this latter problem of choosing the feature space $\mathcal{X}$. Assumption (2.19) implies that we have expected frequencies

$$\frac{1}{v_i}\mathbb{E}[Y_i] = \lambda(\boldsymbol{x}_i) = \exp\langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle = \exp\{\beta_0\} \prod_{j=1}^{q} \exp\{\beta_j x_{i,j}\}. \tag{2.20}$$

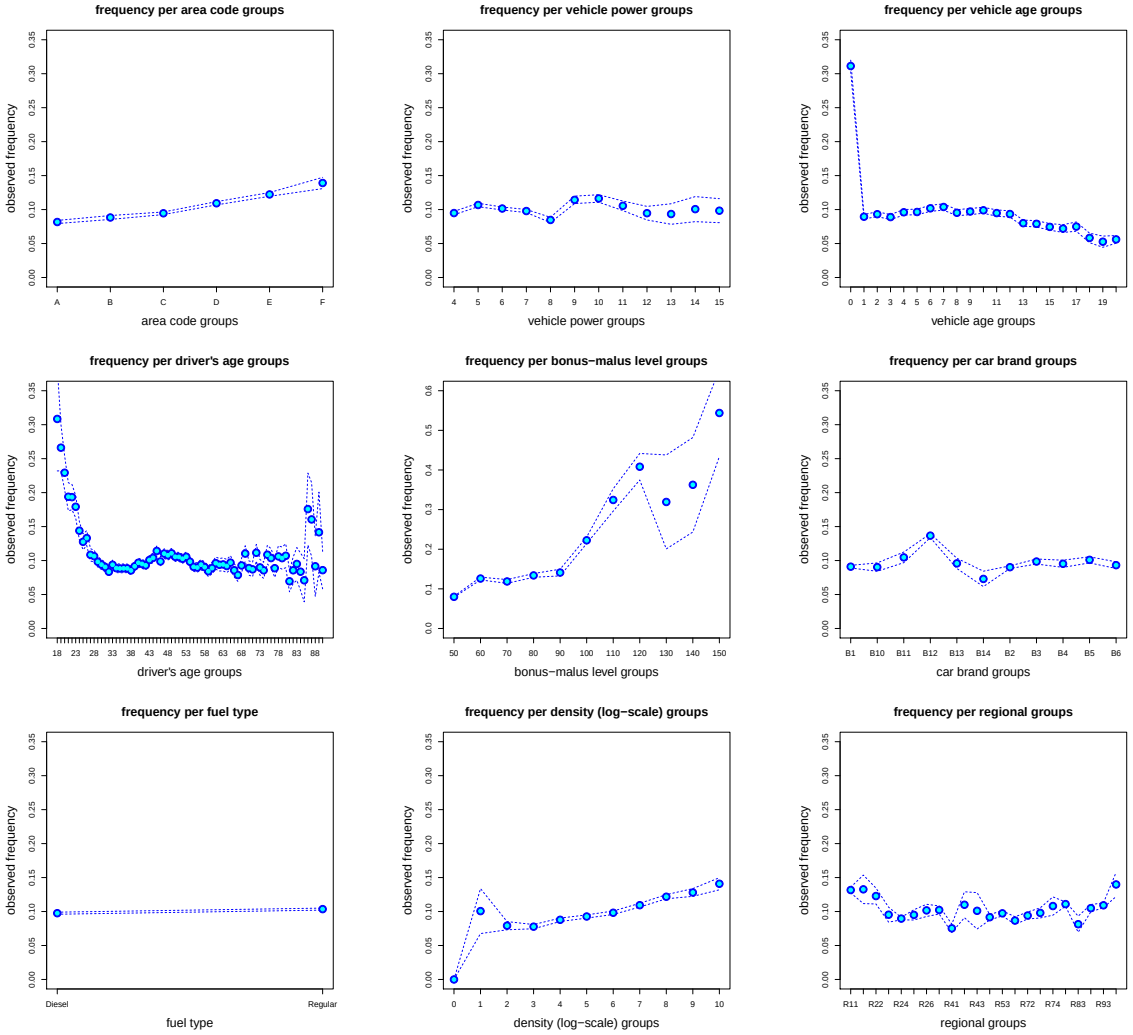

Figure 1: Observed marginal frequencies for the features on lines 6 to 14 of Listing 1; the dotted lines give empirical two standard deviations confidence bounds (based on the Poisson model assumption).

Thus, the intercept $\beta_0 \in \mathbb{R}$ provides the base frequency $\exp\{\beta_0\}$, and the feature components $x_{i,j}$ describe how the frequency of policy i deviates from this base frequency. Approach (2.20) provides a *log-linear functional form* for each of these feature components, and the feature components interact in a *multiplicative way*. This multiplicative structure is often a desired property in insurance pricing because it leads to high explainability in the regression models, we may refer to the 1960 paper of Bailey–Simon [2]. Therefore, often, the log-link function $h(\cdot) = \log(\cdot)$ is preferred over the canonical link function $g(\cdot)$ in non-Poissonian EDF models. In

general, this leads to the *failure of the balance property*, and one should in a downstream step adjust the intercept β_0 for this bias.
The log-linear structure $x_{i,j} \mapsto \exp\{\beta_j x_{i,j}\}$ might be problematic because in most of the cases this is not the straightforward functional form in which features enter regression functions. In Figure 1 we provide all marginal frequencies of our example for the different features given on lines 6 to 14 of Listing 1. The dotted lines illustrate empirical confidence bounds of two standard deviations (estimated under Poisson model assumptions). From these plots it is obvious that the functional forms are more sophisticated than log-linear; we conclude this even though these plots of Figure 1 neglect potential interactions of the features.

Conclusion. Features $\boldsymbol{x}_i$ need appropriate pre-processing so that they match the GLMs with regression functions described by (2.20).

2.3.3 Feature pre-processing

As described above, features $\boldsymbol{x}_i$ need pre-processing so that they can be used in GLMs. There are two (different) difficulties:

- Categorical features, like `VehBrand`, need to be mapped to numerical values so that they can be used in a regression model.
- Continuous and ordered features, like `DrivAge`, need be transformed so that they fit to the pre-specified functional form (2.20).

Categorical features.
For categorical features we use *dummy coding*. Assume that x_j is categorical taking k different labels $\{a_1, \ldots, a_k\}$. Dummy coding is obtained by declaring one label, say a_k, to be the *reference level* and describing all other labels relative to that reference level. This motivates the map

$$x_j \;\mapsto\; (\mathbb{1}_{\{x_j=a_1\}}, \ldots, \mathbb{1}_{\{x_j=a_{k-1}\}})^\top \in \mathbb{R}^{k-1}, \tag{2.21}$$

we also refer to the categorical distribution in Section 2.1.3, and it provides regression function

$$x_j \;\mapsto\; \exp\{\beta_0\} \exp\left\{\sum_{l=1}^{k-1} \beta_l \mathbb{1}_{\{x_j=a_l\}}\right\}. \tag{2.22}$$

Importantly, dummy coding leads to full rank design matrices $\mathfrak{X}$ and, henceforth, to unique MLEs. There are other full rank codings like Helmert's coding, such alternatives lead to the same predictive models, but they give different interpretations to the parameters. From (2.22) we see that every label a_l different from the reference level has its own regression parameter β_l. In machine learning methods, one is less concerned with full rank design matrices. Therefore, typically *one-hot encoding* is preferred which is described by the map

$$x_j \;\mapsto\; (\mathbb{1}_{\{x_j=a_1\}}, \ldots, \mathbb{1}_{\{x_j=a_k\}})^\top \in \mathbb{R}^{k}. \tag{2.23}$$

We will use dummy coding in this section.[6]

[6] In R, `factor` type variables are treated as categorical, and to those dummy coding is automatically applied.

Continuous and ordered features.
For continuous features one may try to find maps $x_j \mapsto z(x_j)$ that transform the original features x_j to features $z_j = z(x_j)$ satisfying model structure (2.20). We will present such an example below. More common in a first approach is to *categorize* continuous and ordered features x_j by choosing appropriate classes (levels). These classes are then treated as categorical features, and using dummy coding.

We are going to consider two different feature pre-processings and we call the resulting models Model GLM1 and Model GLM2. These two models coincide with the ones in Table 1 of [43].

Model GLM1. We use the following feature pre-processing, motivated by Figure 1:

- `Area`: we choose a continuous log-linear feature component for the `Area` code, therefore we map $\{\mathrm{A}, \ldots, \mathrm{F}\} \mapsto \{1, \ldots, 6\} \subset \mathbb{R}$ to real values;
- `VehPower`: we build 6 categorical classes by merging vehicle power groups bigger and equal to 9, this results in $k = 6$ labels for which we use dummy coding (2.21);
- `VehAge`: we build 3 categorical classes $[0, 1)$, $[1, 10]$, $(10, \infty)$ and use dummy coding (2.21) with $k = 3$;
- `DrivAge`: we build 7 categorical classes $[18, 21)$, $[21, 26)$, $[26, 31)$, $[31, 41)$, $[41, 51)$, $[51, 71)$, $[71, \infty)$ and use dummy coding (2.21) with $k = 7$;
- `BonusMalus`: continuous log-linear feature component, and we cap at value 150;
- `VehBrand`: categorical feature component with totally $k = 11$ labels;
- `VehGas`: binary feature component;
- `Density`: log-density is chosen as continuous log-linear feature component, i.e. we pre-process $\texttt{Density} \mapsto \log(\texttt{Density}) \in \mathbb{R}$;
- `Region`: categorical feature component with totally $k = 22$ labels.

Thus, we consider 3 continuous feature components (`Area`, `BonusMalus`, log-`Density`), 1 binary feature component (`VehGas`) and 5 categorical feature components (`VehPower`, `VehAge`, `DrivAge`, `VehBrand`, `Region`). The latter two are categorical by nature; the former three are continuous but their functional forms are far from being log-linear, therefore, we group them categorically. The resulting feature space $\mathcal{X}$ is given by

$$\mathcal{X} \subset \{1\} \times \mathbb{R} \times \{0,1\}^5 \times \{0,1\}^2 \times \{0,1\}^6 \times \mathbb{R} \times \{0,1\}^{10} \times \{0,1\} \times \mathbb{R} \times \{0,1\}^{21}. \tag{2.24}$$

That is, we have a $q+1 = 1+1+5+2+6+1+10+1+1+21 = 49$ dimensional feature space $\mathcal{X} \subset \{1\} \times \mathbb{R}^q$, the feature components in $\{0,1\}^{k-1}$ add up either to 0 or 1 (dummy coding), and the continuous feature components live on a finite interval, these side constraints are the reason for using the symbol " $\subset$ " in formula (2.24). In general, we may and will assume that the feature space $\mathcal{X}$ is compact.

Model GLM2. We use the same feature pre-processing as in Model GLM1 except for the driver age variable `DrivAge`. We replace the categorical `DrivAge` variable of the previous model by the following 5-dimensional feature vector

$$\texttt{DrivAge} \;\mapsto\; \left(\texttt{DrivAge}, \log(\texttt{DrivAge}), \texttt{DrivAge}^2, \texttt{DrivAge}^3, \texttt{DrivAge}^4\right)^\top \in \mathbb{R}^5, \tag{2.25}$$

thus, we have polynomial terms of up to order 4 and an additional logarithmic term. This choice will allow us to obtain fairly flexible age profiles in the regression function, i.e. the log-linear function is extended by a log-square, log-cubic and log-quartic term, as well as a linear term (under the log-link function). This is an easy way to obtain more general functional forms in (2.20). The resulting feature space reads as

$$\mathcal{X} \subset \{1\} \times \mathbb{R} \times \{0,1\}^5 \times \{0,1\}^2 \times \mathbb{R}^5 \times \mathbb{R} \times \{0,1\}^{10} \times \{0,1\} \times \mathbb{R} \times \{0,1\}^{21}, \tag{2.26}$$

having dimension $q+1 = 1+1+5+2+5+1+10+1+1+21 = 48$.

Conclusion. We need to pre-process feature information to bring it into a functional form compatible with the regression function assumption (2.20). This can be done in different ways, and it will result in different GLMs, compare (2.24) and (2.26).

2.3.4 GLM Results

We assume that all policies $\{(Y_i, \boldsymbol{x}_i, o_i)\}_{i=1}^n$ are independent and Poisson distributed according to (2.19), with canonical link function, linear predictors (2.14), and features $\boldsymbol{x}_i \in \mathcal{X}$ pre-processed either according to Model GLM1, see (2.24), or Model GLM2, see (2.26).
In this section we fit these two models to the data and analyze their predictive performance. Therefore, we first partition the data $\{(Y_i, \boldsymbol{x}_i, o_i)\}_{i=1}^n$ into the learning data set $\mathcal{D}$ and the test data set $\mathcal{T}$. We use exactly the same partition as in Listing 2 of [36].[7] This provides us with the sets described in Table 3. At the first sight $\mathcal{D}$ and $\mathcal{T}$ seem very similar, with a slightly higher empirical frequency on the test data set $\mathcal{T}$.

	numbers of observed claims					empirical	total
	0	1	2	3	4	frequency	volume
empirical probability on $\mathcal{D}$	94.99%	4.74%	0.36%	0.01%	0.002%	10.02%	610'212
empirical probability on $\mathcal{T}$	94.83%	4.85%	0.31%	0.01%	0.003%	10.41%	67'801

Table 3: Comparison of $\mathcal{D}$ and $\mathcal{T}$: the last columns provide the empirical frequency on these two data sets and the underlying number of policies, the other columns show the split of the policies according to the numbers of observed claims.

Next we provide the in-sample and the out-of-sample deviance losses (2.17) and (2.18) under the Poisson model assumption and the canonical link choice

$$\mathcal{L}(\mathcal{D}, \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\mathrm{MLE}}) = \frac{2}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} Y_i \left[\frac{v_i \exp\left\langle \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\mathrm{MLE}}, \boldsymbol{x}_i \right\rangle}{Y_i} - 1 - \log\left(\frac{v_i \exp\left\langle \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\mathrm{MLE}}, \boldsymbol{x}_i \right\rangle}{Y_i} \right) \right],$$

[7] The partition received from Listing 2 in [36] leading to our Table 3 has been done under R version 3.5.0 (2018-04-23). In the upgrade from R version 3.5.x to R version 3.6.x the function `sample` which is used to generate this partition has been changed. For reproducability under the new version one should set `RNGversion(''3.5.0'')`.

and

$$\mathcal{L}(\mathcal{T}, \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\mathrm{MLE}}) = \frac{2}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} Y_i \left[\frac{v_i \exp\left\langle \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\mathrm{MLE}}, \boldsymbol{x}_i \right\rangle}{Y_i} - 1 - \log\left(\frac{v_i \exp\left\langle \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\mathrm{MLE}}, \boldsymbol{x}_i \right\rangle}{Y_i} \right) \right],$$

where the summands on the right-hand sides are set equal to $2v_i \exp\langle \widehat{\boldsymbol{\beta}}_{\mathcal{D}}^{\mathrm{MLE}}, \boldsymbol{x}_i \rangle$ for $Y_i = 0$.

Listing 2: R code to fit Model GLM2.

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

To fit Model GLM1 and Model GLM2 to the learning data set $\mathcal{D}$, we use the R command `glm`; the corresponding R code for Model GLM2 is illustrated in Listing 2. The results are presented in Table 4 and they will be commented in detail below.

	run time	# param.	in-sample loss	out-of-sample loss	average frequency
(a) homogeneous model	–	1	32.935	33.861	10.02%
(b1) Model GLM1	23s	49	31.267	32.171	10.02%
(b2) Model GLM2	24s	48	31.257	32.149	10.02%
(b*) Model GAM	624s	–	31.207	32.186	10.02%

Table 4: Run time, number of model parameters, in-sample and out-of-samples losses (units are in 10^{-2}), average estimated frequency on $\mathcal{D}$ (balance property).

The first column of Table 4 provides the run times.[8] The second column provides the number of parameters involved, see (2.24) and (2.26). The remaining columns give the in-sample and out-of-sample losses as well as the balance property, see Corollary 2.4.
The first row (a) provides the results for the homogeneous model (also called intercept model or null model). This homogeneous model does not consider any feature information, but only uses an intercept β_0 and, henceforth, assumes that the underlying portfolio $\{(Y_i, \boldsymbol{x}_i, o_i)\}_{i=1}^n$ has a homogeneous expected claims frequency $\lambda(\boldsymbol{x}_i) = \exp\{\beta_0\}$, and only differs in the offsets $o_i = \log(v_i) = \log(\texttt{Exposure}_i)$. Lines (b1)-(b2) consider Models GLM1 and GLM2. We obtain (clearly) lower loss figures and, thus, are in favor of the GLMs over the homogeneous model. This is also supported by a χ^2-test for the null hypothesis of having the homogeneous model against the alternatives of either having Model GLM1 or GLM2, we also refer to Lemma 3.1 in Ohlsson–Johansson [37]. In both cases we receive a p-value of roughly 0 which gives a high support to the GLMs over the homogeneous model. In view of the out-of-sample losses we prefer Model GLM2 over Model GLM1, this is also supported by the fact that Model GLM2 has a smaller AIC value[9] of 252'995 than Model GLM1 (253'062).

[8] The run times have been measured on a personal laptop Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99GHz with 16GB RAM.

[9] Akaike Information Criterion AIC, see Akaike [1].

Listing 3: ANOVA analysis of variables in Model GLM2.

```
Analysis of Deviance Table

Model: poisson, link: log

Response: ClaimNb

Terms added sequentially (first to last)


              Df Deviance Resid. Df Resid. Dev
NULL                          610211     200974
AreaGLM        1    669.4    610210     200305
VehPowerGLM    5    154.2    610205     200151
VehAgeGLM      2   4585.2    610203     195566
BonusMalusGLM  1   3854.5    610202     191711
VehBrand      10     50.9    610192     191660
VehGas         1     59.2    610191     191601
DensityGLM     1      8.4    610190     191593
Region        21    178.1    610169     191415
DrivAge        1    256.2    610168     191158
log(DrivAge)   1     27.2    610167     191131
I(DrivAge^2)   1     95.4    610166     191036
I(DrivAge^3)   1    242.0    610165     190794
I(DrivAge^4)   1     61.4    610164     190732
```

Next we can perform variable selection. Listing 3 provides the ANOVA[10] of Model GLM2. This analysis *sequentially adds* feature components to the GLM exactly in the order used in the `glm` command of Listing 2. It shows the reductions in in-sample deviance losses by adding one feature component after the other. From this ANOVA we see that the variable `DensityGLM` leads to a comparably small reduction, and `VehBrand` leads to a comparably small reduction in relation to the number of parameters used. The latter tells us that probably some car brands can be merged (clustered) leading to less parameters and a comparably good model (indeed, this will be one of the findings, below). The interpretation of `DensityGLM` is a bit more difficult. Further analysis shows that `AreaGLM` and `DensityGLM` are highly correlated and, as a consequence, it may be sufficient to only keep one of these two variables in the model. The ANOVA of Listing 3 tells us that once `AreaGLM` is in the GLM we do no longer need `DensityGLM`, and similarly if we exchange the order of these two variables we find that once `DensityGLM` is in the model we no longer need `AreaGLM`. For the time being we keep both variables in the model (but we give a slight preference to `DensityGLM` because it is slightly more significant). A more thorough approach would be to use an elastic net for regularization which at the same time tracks overfitting and performs parameter selection, we refer to Zou–Hastie [57] and Section 4.2 in Hastie et al. [24]. Finally, the functional form (2.25) for `DrivAge` seems to work very well because all 5 components should be kept in the model.

Conclusion and question. We conclude that we prefer Model GLM2, and we are going to use this as a benchmark for all following regression models. But is this model "good"? Or is it just the best option among a set of bad options?

[10] analysis of variance

The latter question is quite hard to answer, and we would like to conclude this section with a few points that should be considered.

- Are the in-sample and the out-of-sample losses of $31.257 \cdot 10^{-2}$ and $32.149 \cdot 10^{-2}$ small or large? If we run a simulation analysis and simulate new observations Y_i^* from Model GLM2 (parametric bootstrap using the Poisson assumption), then we can calculate the deviance losses of these simulations against the true model (which is known here because we have been simulating from it). On $\mathcal{D}$ we receive $28.048 \cdot 10^{-2}$ and on $\mathcal{T}$ we get $28.228 \cdot 10^{-2}$. These numbers are clearly smaller than the ones in Table 4 which indicates that we may still have some room for model improvements. On the other hand, we should also keep in mind that these bootstrapped figures might be too small which may happen due to model errors that do not fully reflect the complexity of the real data (in fact, we have some indication that this the case here, but we do not want to further exploit this, here).

- There are two options to improve our model:

  – At the moment, our model only captures multiplicative interactions, see (2.20). We should explore other forms of feature interactions. This is going to be done and discussed in the neural network sections below.

  – Improve the functional forms of the marginals. A natural way to do so is to replace GLMs by generalized additive models (GAMs) using natural cubic splines for improving the linear predictors. That is, basically we try to find optimal transformations $x_j \mapsto z(x_j)$ for feature components x_j within the family of natural cubic splines for $z(\cdot)$. For more background we refer to Hastie–Tibshirani [21, 22], Wood [50], Ohlsson–Johansson [37] and Chapter 3 in Wüthrich–Buser [53]. In our situation GAMs do not provide any improvement, and this at the price of much longer run times, see line (b*) of Table 4. In fact, under the calibration used we observe an over-fitting.[11] For this reason, we do not use GAMs.

- Comparing the in-sample loss of $28.048 \cdot 10^{-2}$ (bootstrap simulation) to the in-sample loss of $31.257 \cdot 10^{-2}$ (Model GLM2) illustrates that the reduction of in-sample loss from Model GLM2 to an optimal model can be at most 10%. The remainder of at least $28.048 \cdot 10^{-2}$ has to be allocated to pure randomness from simulation. Thus, we try to find model structure on a scale that is much smaller than the scale of the noise in the data. This explains why these regression problems are so hard to be solved because data is heavily dominated by the noise terms. This is also related to the class imbalance problem which is a notoriously difficult problem in model calibration.

3 Feed-forward neural network regression models

We present feed-forward neural networks (short *networks*) in this section. In regression theory, networks are seen as a generalization of GLMs (when choosing deviance losses as objective

[11] We use the R command `gam` from the package `mgcv` under the standard parametrization; over-fitting could be improved by choosing less knots for some of the natural cubic splines. It seems that the generalized cross-validation (GCV) criterion used is not fully competitive, here. The GCV criterion is used instead of proper cross-validation due to computational reasons.

functions). The main difference to GLMs is that the linear predictor (2.11) is replaced by a non-linear one. For a general introduction and discussion of networks we refer to Goodfellow et al. [19], the SAA tutorials [14, 36, 43] and the lecture notes of Wüthrich–Buser [53].

3.1 Generic definition of feed-forward neural networks

Choose hyper-parameter $d \in \mathbb{N}$ for the *depth* of the network and replace the linear predictor (2.11) by the *network predictor*, for $i = 1, \ldots, n$,

$$(o_i, \boldsymbol{x}_i) \;\mapsto\; h\left(\mathbb{E}[Y_i]\right) = o_i + \left\langle \boldsymbol{\beta}^{(d+1)}, \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}_i) \right\rangle, \tag{3.1}$$

where $\boldsymbol{z}^{(m)}$, $1 \leq m \leq d$, describes the *m-th hidden network layer* of dimension $q_m + 1 \in \mathbb{N}$, and where $\boldsymbol{\beta}^{(d+1)} \in \mathbb{R}^{q_d+1}$ is the output parameter providing one neuron in the *output layer*. The output is also called *readout* with *readout parameter* $\boldsymbol{\beta}^{(d+1)} \in \mathbb{R}^{q_d+1}$. For a given *activation function* $\psi : \mathbb{R} \to \mathbb{R}$, the m-th hidden network layer with activation function ψ is a map

$$\boldsymbol{z}^{(m)} : \{1\} \times \mathbb{R}^{q_{m-1}} \to \{1\} \times \mathbb{R}^{q_m}, \qquad \boldsymbol{z} \mapsto \boldsymbol{z}^{(m)}(\boldsymbol{z}) = \left(1, z_1^{(m)}(\boldsymbol{z}), \ldots, z_{q_m}^{(m)}(\boldsymbol{z})\right)^\top, \tag{3.2}$$

with *hidden neurons* $z_j^{(m)}$, $1 \leq j \leq q_m$, being described by

$$z_j^{(m)}(\boldsymbol{z}) = \psi\left\langle \boldsymbol{\beta}_j^{(m)}, \boldsymbol{z} \right\rangle,$$

for given network parameters $\boldsymbol{\beta}_j^{(m)} \in \mathbb{R}^{q_{m-1}+1}$. Moreover, we initialize $q_0 + 1 = q + 1$ being the dimension of the feature space $\boldsymbol{x}_i \in \mathcal{X} \subset \{1\} \times \mathbb{R}^q$ (again including the intercept component). Examples of commonly used activation functions ψ are given in Table 5, below. The resulting network parameter $\boldsymbol{\beta} = (\boldsymbol{\beta}_1^{(1)}, \ldots, \boldsymbol{\beta}_{q_d}^{(q)}, \boldsymbol{\beta}^{(d+1)})^\top \in \mathbb{R}^r$ has dimension, set $q_{d+1} = 1$,

$$r = \sum_{m=1}^{d+1} (q_{m-1} + 1) q_m.$$

An example is provided in Figure 2. This example considers the following variables as continuous `Area`, `VehPower`, `VehAge`, `DrivAge`, `BonusMalus`, `VehGas` and log-`Density` (in blue color), and the following variables by one-hot encoding `VehBrand` and `Region` (in green and magenta color, respectively). This gives $q = 40$.

3.2 Interpretation and properties of networks

3.2.1 Generalization of generalized linear models

We first explain two ways of interpreting the network being an extension of GLMs.

(i) The obvious way is to choose a network with $d = 0$ hidden layers, i.e. we directly link the input layer to the output layer. Naturally, we obtain a GLM with link function h, see (3.1).

(ii) A second way of receiving a GLM from a network is by choosing the identity function for the activation function, i.e. $\psi(x) = x$ for $x \in \mathbb{R}$. In this case, all neurons are linear functions $\boldsymbol{z} \mapsto z_j^{(m)}(\boldsymbol{z}) = \langle \boldsymbol{\beta}_j^{(m)}, \boldsymbol{z} \rangle$. As a consequence, the network predictor (3.1) is

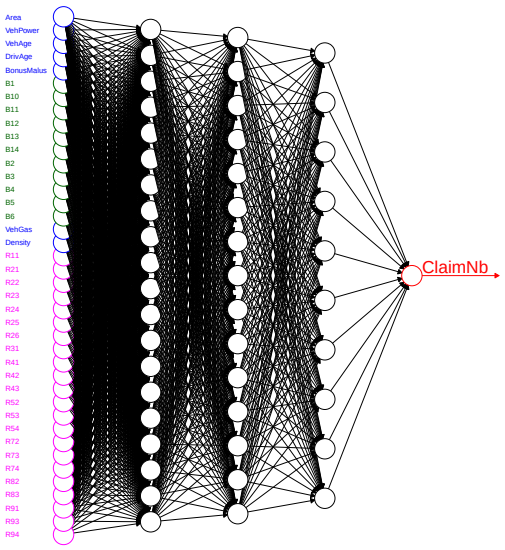

Figure 2: Network of depth $d = 3$ having $(q_1, q_2, q_3) = (20, 15, 10)$ hidden neurons, feature space $\mathcal{X} \subset \{1\} \times \mathbb{R}^q$ of dimension $q_0 + 1 = q + 1 = 41$ and with network parameter $\boldsymbol{\beta} \in \mathbb{R}^r$ of dimension $r = 1'306$.

a composition of linear maps, thus, it is linear too. Therefore, we are back in a GLM under the linear activation function choice. The maximal dimension (in the sense of a linear principal components analysis (PCA)) between input layer and last hidden layer is determined by the smallest layer (including the input layer), and this network acts like a linear autoencoder (bottleneck network), we refer Sections 2 and 3 in Rentzmann–Wüthrich [38] and the references therein. In particular, this provides us with a type of PCA regression if a hidden layer is smaller than the input layer.

3.2.2 Representation learning

We compare the GLM to the mapping from the last hidden layer to the output layer (readout) of the network. The GLM is given by, see (2.11),

$$(o_i, \boldsymbol{x}_i) \;\mapsto\; \mathbb{E}[Y_i] = h^{-1}\left(o_i + \langle \boldsymbol{\beta}, \boldsymbol{x}_i \rangle\right).$$

For the network we have, see (3.1),

$$(o_i, \boldsymbol{z}_i) \;\mapsto\; \mathbb{E}[Y_i] = h^{-1}\left(o_i + \langle \boldsymbol{\beta}^{(d+1)}, \boldsymbol{z}_i \rangle\right), \tag{3.3}$$

where the transformed features are given by

$$\boldsymbol{x}_i \;\mapsto\; \boldsymbol{z}_i = \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}_i) \;\in\; \mathcal{Z} = \{1\} \times \mathbb{R}^{q_d}. \tag{3.4}$$

Thus, we have exactly the same functional form in the two approaches w.r.t. $\boldsymbol{x}_i \in \mathcal{X}$ and $\boldsymbol{z}_i \in \mathcal{Z}$, respectively, the only difference is that networks pre-process the features $\boldsymbol{x}_i \mapsto \boldsymbol{z}_i$ according to (3.4). In GLMs the actuary (or statistician) performs feature pre-processing so that they can be used in a GLM on a suitable feature space $\mathcal{X}$, see Section 2.3.3, whereas in networks the feature pre-processing is done by the network, and the modeler only needs to specify the network

architecture and, in particular, the dimension $q_d + 1$ of the (hidden) feature space $\mathcal{Z}$, see (3.4). In network jargon the mapping (3.4) is called *representation learning*, see Richman [39], because during network calibration we learn a representation $\boldsymbol{z}_i$ for $\boldsymbol{x}_i$ that is suitable for a GLM on the feature space $\mathcal{Z}$.

3.2.3 Universal approximation property

The reason for the functioning of representation learning lies in the *universal approximation property* of appropriately specified networks. A crucial ingredient of a network is the choice of the activation function ψ. Table 5 provides the most common choices of continuous activation functions.

sigmoid/logistic activation function	$\psi(x) = (1 + e^{-x})^{-1}$	$\psi' = \psi(1 - \psi)$
hyperbolic tangent activation function	$\psi(x) = \tanh(x)$	$\psi' = 1 - \psi^2$
exponential activation function	$\psi(x) = \exp(x)$	$\psi' = \psi$
rectified linear unit (ReLU) activation function	$\psi(x) = x\mathbb{1}_{\{x \geq 0\}}$	

Table 5: Typical choices of continuous activation functions.

The hyperbolic tangent can be received as a simple transformation of the sigmoid function

$$x \;\mapsto\; \tanh(x) \;=\; \frac{e^x - e^{-x}}{e^x + e^{-x}} \;=\; \frac{2e^{2x}}{1 + e^{2x}} - 1 \;=\; 2\left(1 + e^{-2x}\right)^{-1} - 1.$$

Therefore, these two activation functions do not really differ in their properties. In deep networks, i.e. of large depth d, we prefer the hyperbolic tangent activation function because its range is centered around the origin which is an advantage in model fitting (because outputs are more likely centered around zero). The sigmoid activation function is *continuous* and so-called *discriminatory*, see Cybenko [9]. These two properties are important for the next statement.

Theorem 3.1 (Cybenko's universality theorem [9]) *Choose an activation function ψ that is continuous and discriminatory. Define the set of all network functions of depth $d = 1$ on the unit cube $[0,1]^{q_0}$ by, recall (3.1),*

$$\mathcal{G}_1 = \left\{ \boldsymbol{x} \in [0,1]^{q_0} \mapsto \left\langle \boldsymbol{\beta}^{(2)}, \boldsymbol{z}^{(1)}(1, \boldsymbol{x}) \right\rangle ;\; q_1 \in \mathbb{N} \text{ and } \boldsymbol{\beta} \in \mathbb{R}^r \text{ with } r = (q_0 + 1)q_1 + (q_1 + 1) \right\}.$$

$\mathcal{G}_1$ is dense in the set $C([0,1]^{q_0}, \mathbb{R})$ of continuous functions on $[0,1]^{q_0}$ (w.r.t. supremum norm).

For the proof we refer to Cybenko [9] and Hornik et al. [26]. Basically, Theorem 3.1 states that networks allow us to approximate any continuous function on a compact support arbitrarily well if we allow for arbitrarily many neurons $q_1 \in \mathbb{N}$ in the hidden layer. This explains that general networks with sigmoid (or hyperbolic tangent) activation function can learn any representation (so that ultimately it fits to a GLM structure in the sense of (3.3)). Leshno et al. [31] have proved that the universal approximation property holds in networks of depth $d = 1$ if and only if the chosen activation function is non-polynomial. This also includes the popular ReLU activation function.

Thus, in principle, we can work with networks $\mathcal{G}_1$ of depth $d = 1$. However, such networks may not be very efficient in approximating interactions in regression functions, and we may need a

huge number q_1 of hidden neurons to receive the required precision in approximation. Therefore, one should at the same time go deep $d \geq 2$ to get an efficient representation learning. For a simple illustrative example we refer to Examples 6.1 and 6.2 in [14], and we mention the work of Zaslavsky [55] and Montúfar et al. [34] who study this question from a combinatorial point of view.

In a nutshell, adding more neurons to a network of depth $d = 1$ means that we *superimpose* more basis functions, see (3.2), similar to Fourier series approximations of functions. Going deep $d \geq 2$ in the network architecture means that we *compose* functions which may lead to more complexity in functional forms with less parameters, see (3.1). In fact, under quite general assumptions one can show that deep networks converge exponentially to a true regression function where classical approximation theory based on superposition may only provide polynomial convergence with an increasing number of parameters, see Grohs et al. [20].

3.2.4 Embedding learning

In GLMs we have applied dummy coding (2.21) to categorical variables, and in networks one typically uses one-hot encoding (2.23) for categorical variables. The latter means that the k different categorical labels $\{a_1, \ldots, a_k\}$ are mapped to the k basis vectors in $\mathbb{R}^k$. This is illustrated in Figure 2 for our example of Listing 1. Blue color in the input layer illustrates the 7 ordered and binary feature components, green color illustrates the labels of the 11 car brands, and magenta color gives the 22 labels of the different French regions. Altogether one-hot encoding results in a feature space $\mathcal{X}$ of dimension $q_0 + 1 = q + 1 = 41$ (including the intercept variable). This does not seem to be a very efficient representation of categorical variables. Moreover, one-hot encoding does not give us a proximity for categorical variables that are more similar w.r.t. the regression modeling task (because the Euclidean distance between all basis vectors is the same). Borrowing ideas from natural language processing (NLP) we promote to embed categorical variables into lower dimensional spaces; see Bengio et al. [4] and Sahlgren [41] for NLP, and recently in actuarial science Richman [39] for embedding learning.

Embedding learning means that we put in front of the network a so-called *embedding layer*. This requires to choose for each categorical variable an embedding dimension $b < k$ (hyperparameter). The embedding is then defined by n mapping

$$\boldsymbol{e} : \{a_1, \ldots, a_k\} \to \mathbb{R}^b, \qquad a \ \mapsto \ \boldsymbol{e}(a). \tag{3.5}$$

Thus, we allocate to every label $a_1, \ldots, a_k$ a b-dimensional vector $\boldsymbol{e}(a_1), \ldots, \boldsymbol{e}(a_k)$. This is called an *embedding* of $\{a_1, \ldots, a_k\}$ into $\mathbb{R}^b$, and the embedding weights $(\boldsymbol{e}(a_l))_{l=1,\ldots,k}$ are learned during the model calibration which is called *embedding learning*.

An embedding layer of dimension b adds kb embedding weights to the network parameter $\boldsymbol{\beta}$, but it reduces the number of network parameters $(\boldsymbol{\beta}_j^{(1)})_j$ from kq_1 (using one-hot encoding for k labels) to bq_1 (using embedding dimension b). If we choose embedding layers of dimension $b = 2$ for both car brand and French regions, we receive a network parameter $\boldsymbol{\beta} \in \mathbb{R}^r$ of dimension (including the embedding weights)

$$r = b(11 + 22) + (7 + b + b + 1)q_1 + (q_1 + 1)q_2 + (q_2 + 1)q_3 + q_3 + 1 = 792,$$

for a network of depth $d = 3$ with $(q_1, q_2, q_3) = (20, 15, 10)$ hidden neurons. This is illustrated in Figure 3 (rhs). If we choose 1-dimensional embedding layers $b = 1$ we receive $r = 719$.

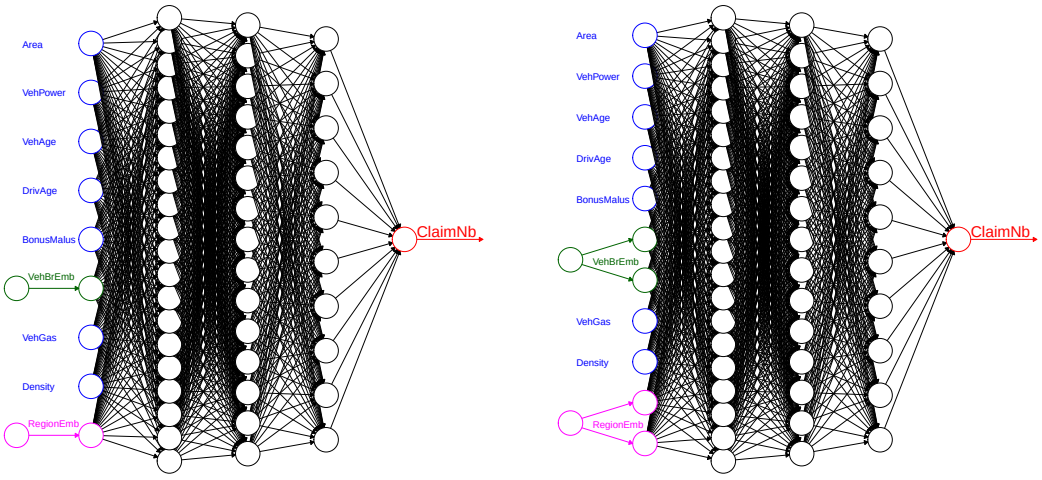

Figure 3: Network of depth $d = 3$ having $(q_1, q_2, q_3) = (20, 15, 10)$ hidden neurons, the feature space $\mathcal{X}$ uses embedding layers of dimensions (lhs) $b = 1$ and (rhs) $b = 2$ which results in network parameters $\boldsymbol{\beta} \in \mathbb{R}^r$ of dimensions (lhs) $r = 719$ and (rhs) $r = 792$.

These embedding weights $\boldsymbol{e}(a_l) \in \mathbb{R}^b$, $l = 1, \ldots, k$, are also learned (in an optimal way) during model calibration. Analysis of these embedding weights will allow us to study proximity of categorical labels w.r.t. the regression task. For $b = 1$ the embedding weights play a similar role as the GLM regression parameters in (2.22). But in general, we will choose higher embedding dimensions b so that more complex embeddings can be learned, in the spirit of Cybenko's universal approximation Theorem 3.1.

3.3 Example: French MTPL data, revisited

3.3.1 Model calibration using gradient descent methods

Summarizing the previous development, we try to calibrate the network parameter $\boldsymbol{\beta} \in \mathbb{R}^r$ to the learning data set $\mathcal{D}$ by minimizing the in-sample loss (objective function)

$$\boldsymbol{\beta} \;\mapsto\; \mathcal{L}(\mathcal{D}, \boldsymbol{\beta}) = \frac{2}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \kappa_i \left(g(Y_i) Y_i - b(g(Y_i)) - \theta_i Y_i + b(\theta_i) \right) \;\geq\; 0, \tag{3.6}$$

with network predictors for $i = 1, \ldots, n$

$$\theta_i = g\left(\mathbb{E}[Y_i]\right) = g \circ h^{-1} \left(o_i + \left\langle \boldsymbol{\beta}^{(d+1)}, \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)} \right) (\boldsymbol{x}_i) \right\rangle \right), \tag{3.7}$$

and where $\boldsymbol{\beta} \in \mathbb{R}^r$, and $\boldsymbol{x}_i$ may include embedding weights for categorical variables. h is the chosen link function and g is the canonical link of the corresponding EDF.
In analogy to GLMs, one is tempted to estimate the network parameter $\boldsymbol{\beta} \in \mathbb{R}^r$ by maximum likelihood. However, this will not be possible in general. The objective function (3.6) is non-convex in $\boldsymbol{\beta}$ and there is no hope of finding the global minimum of this objective function. At first sight this seems very unfortunate, but actually it is not at all. Recalling Cybenko's universal approximation Theorem 3.1 suggests that for sufficiently complex network the MLE calibrated model would highly over-fit to the learning data $\mathcal{D}$ and it would very poorly generalize to the

test data $\mathcal{T}$. For this reason, we are not interested into the MLE of $\boldsymbol{\beta}$, but a regularized version thereof will be good enough.
State-of-the-art network calibration uses stochastic gradient descent (SGD) algorithms. These algorithms step-wise locally improve the parameter choice $\boldsymbol{\beta}_{[t]} \mapsto \boldsymbol{\beta}_{[t+1]}$, the lower index $_{[t]}$ indicating algorithmic time. These SGD algorithms are *early stopped* as soon as there is a sign of over-fitting, say at algorithmic time $t^\star$, and the early stopped parameter $\widehat{\boldsymbol{\beta}}^{\mathrm{NN}} = \boldsymbol{\beta}_{[t^\star]}$ is chosen as network calibration. Let us give some remarks before describing the SGD algorithm in more detail.

- Early stopping requires that we track over-fitting. This should *not* be done on the test data set $\mathcal{T}$ (which is hold out for studying generalization to unseen data), but we should further partition the learning data set $\mathcal{D}$ into a training data set $\mathcal{U}$ and a validation data set $\mathcal{V}$. The former will be used in the SGD algorithm to step-wise locally improve $\boldsymbol{\beta}_{[t]}$ and the latter one is used to decide on early stopping.
- The chosen network parameter $\widehat{\boldsymbol{\beta}}^{\mathrm{NN}}$ will depend on (a) the partition of $\mathcal{D}$ into $\mathcal{U}$ and $\mathcal{V}$; (b) the choice of the stopping criterion; (c) the choice of the initial value $\boldsymbol{\beta}_{[0]}$ of the SGD algorithm, and (d) on further parameters involved in the SGD method. As a consequence, there does not exist any *best* solution $\widehat{\boldsymbol{\beta}}^{\mathrm{NN}}$, but there are infinitely many *sufficiently good* solutions $\{\widehat{\boldsymbol{\beta}}^{\mathrm{NN}}\}$.
- The non-uniqueness of the solutions $\{\widehat{\boldsymbol{\beta}}^{\mathrm{NN}}\}$ is troublesome in particular in insurance pricing, because different seeds $\boldsymbol{\beta}_{[0]}$ will lead to different solutions. Note that there is not any objective criterion to prefer one seed over the other! This is schematically illustrated in Figure 4. The surface illustrates the objective function $\boldsymbol{\beta} \mapsto \mathcal{L}(\mathcal{D}, \boldsymbol{\beta})$ with blue color indicating the area of under-fitting, green color the early stopping region, and the red color indicating the global minimas reflecting over-fitting.

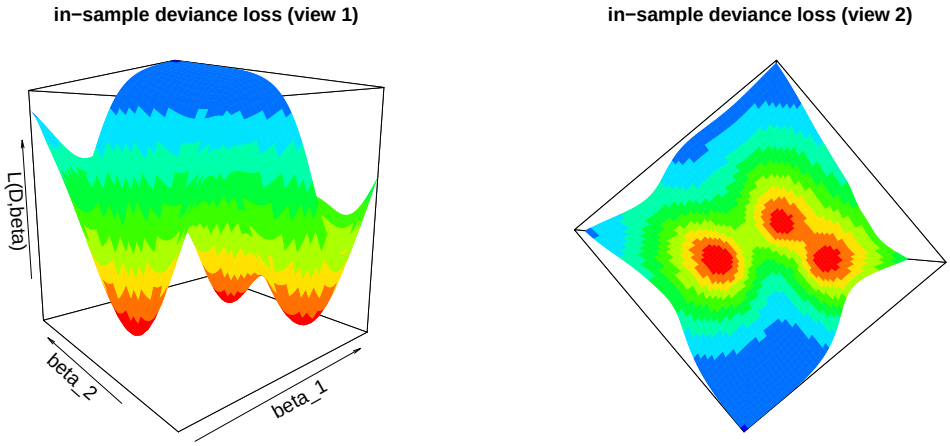

Figure 4: Objective function $\boldsymbol{\beta} \mapsto \mathcal{L}(\mathcal{D}, \boldsymbol{\beta})$ with blue area illustrating under-fitting, green area giving the early stopping region, and the red area shows the global minimas reflecting over-fitting (the surface is the same in both plots but illustrated from different angles).

Typically, the SGD algorithm starts in a seed $\boldsymbol{\beta}_{[0]}$ lying in the blue area of the loss surface

of Figure 4. Step-wise local improvement $\boldsymbol{\beta}_{[t]} \mapsto \boldsymbol{\beta}_{[t+1]}$ means that we approach the green early stopping region, and as soon as we are hitting it we terminate the algorithm. From this plot it is obvious that there are infinitely many sufficiently good early stopped solutions $\widehat{\boldsymbol{\beta}}^{\mathrm{NN}} = \boldsymbol{\beta}_{[t^\star]}$ in the green rings around the local minimas, and the selected choice typically depends on where the algorithm has been starting from.

- Early stopping is a method of parameter regularization, we refer to Section 7.8 in Goodfellow et al. [19].

We briefly describe SGD methods, for a detailed description we refer to Goodfellow et al. [19]. First, we randomly partition the learning data set $\mathcal{D}$ into the training data set $\mathcal{U}$ and the validation data set $\mathcal{V}$, for instance, at ratio 9:1. The validation data set $\mathcal{V}$ is used for tracking over-fitting to the training data $\mathcal{U}$. Gradient descent step calculations may be time-consuming for big training data sets $\mathcal{U}$ because each step involves matrix multiplications of a size directly linked to the size of $\mathcal{U}$. Therefore, typically, $\mathcal{U}$ is (randomly) sub-divided into smaller sets $\mathcal{U}_1, \ldots, \mathcal{U}_S$, so-called *batches* having a given *batch size*. In our example below we will choose a batch size of 10'000 policies. Running through all the batches $\mathcal{U}_1, \ldots, \mathcal{U}_S$ once in the SGD algorithm is called a *training epoch*.
SGD then step-wise iteratively improves a current parameter value $\boldsymbol{\beta}_{[t]}$ at algorithmic time t by

$$\boldsymbol{\beta}_{[t]} \;\mapsto\; \boldsymbol{\beta}_{[t+1]} = \boldsymbol{\beta}_{[t]} - \varrho_t \nabla_{\boldsymbol{\beta}} \mathcal{L}(\mathcal{U}_s, \boldsymbol{\beta}_{[t]}), \tag{3.8}$$

for a *tempered learning rate* $\varrho_t > 0$ such that the first order condition is still the dominant one in the neighborhood of $\boldsymbol{\beta}_{[t]}$ and, henceforth, the value of the objective function decreases by step (3.8); and where $s = s(t)$ runs over all batch indexes $\{1, \ldots, S\}$ in each training epoch. At the same time we evaluate the validation loss $\mathcal{L}(\mathcal{V}, \boldsymbol{\beta}_{[t]})$, and as soon as this validation loss starts to increase[12] we early stop the SGD algorithm because this is a sign of over-fitting to the training data $\mathcal{U}$.
We illustrate SGD training in Figure 5. The blue line shows the in-sample losses $\mathcal{L}(\mathcal{U}_s, \boldsymbol{\beta}_{[t]})$, $t = 0, \ldots, 1'000$, on the training batches $\mathcal{U}_s$. They are constantly decreasing in a slightly wiggly manner because we perform (3.8) over random batches (and not over the entire training data $\mathcal{U}$). The validation losses $\mathcal{L}(\mathcal{V}, \boldsymbol{\beta}_{[t]})$, $t = 0, \ldots, 1'000$, in green color have a minimum after 400 to 500 training epochs, which determines the early stopping time $t^\star$.
We have just explained the plain-vanilla SGD algorithm. Its applications involve an efficient calculation of the gradients in (3.8), for instance, using the back-propagation method of Rumelhart et al. [40], and acceleration methods such as momentum-based methods, we refer to Goodfellow et al. [19]. In our applications we use the `adam` version of the SGD algorithm, we refer to Sections 8.3 and 8.5 in Goodfellow et al. [19].

3.3.2 Revisiting the MTPL example

We revisit the French MTPL example of Section 2.3.1. As network regression model we exactly use the architecture depicted in Figure 3 (rhs). That is, we choose a network of depth $d = 3$ having hidden neurons $(q_1, q_2, q_3) = (20, 15, 10)$. The feature space contains 7 ordered and binary variables, and for the two categorical variables we choose $b = 2$ dimensional embedding

[12] "Starts to increase" can also be replaced by "stops to decrease".

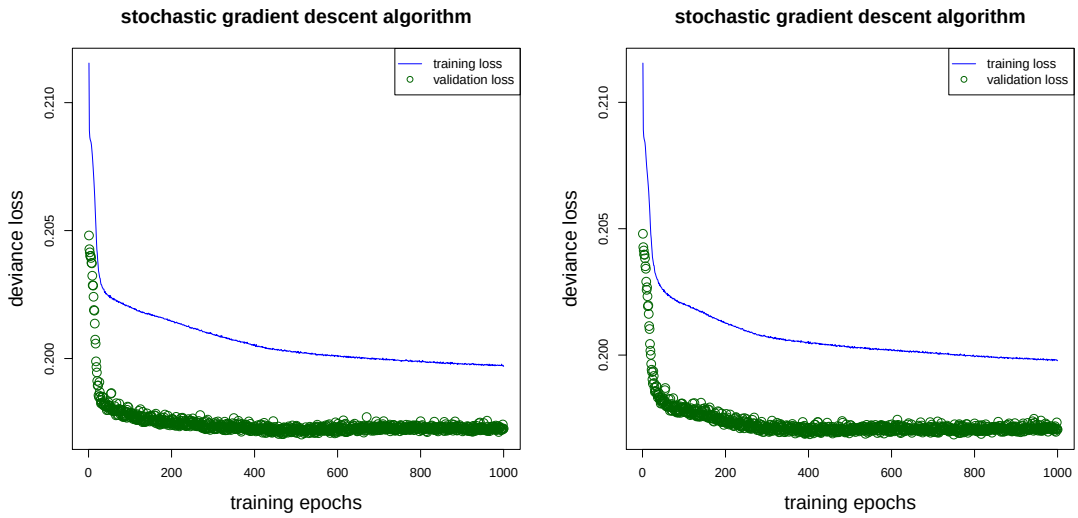

Figure 5: SGD algorithm over 1'000 training epochs on random batches $\mathcal{U}_s$ of size 10'000 for two different initial values $\boldsymbol{\beta}_{[0]}$.

layers. This results in a network having $r = 792$ network parameters. As activation function ψ we choose the hyperbolic tangent, and as link function h we choose the canonical one of the Poisson model. Moreover, we make the Poisson model assumption (2.19) with exposures $v_i = \exp\{o_i\} = \texttt{Exposure}_i \in (0, 1]$. Thus, the only difference to the GLM in Section 2.3 is that we replace the linear predictor by the network predictor (3.1).
In order to get an efficient SGD calibration we have to normalize the continuous feature components because every feature component should live on a similar scale. Note that we model all blue feature components in Figure 3 as continuous variables. We use the *MinMaxScaler* that transforms feature component $x_l \in \mathbb{R}$ as

$$x_l \;\mapsto\; 2\,\frac{x_l - \min x_l}{\max x_l - \min x_l} - 1 \;\in\; [-1, 1].$$

We are now ready for the network calibration. We use the R package `keras` which is a user-friendly API to TensorFlow. The corresponding code is provided in Listing 4.

	run time	# param.	in-sample loss	out-of-sample loss	average frequency
(a) homogeneous model	–	1	32.935	33.861	10.02%
(b2) Model GLM2	24s	48	31.257	32.149	10.02%
(c1) Model NN	390s	792	30.411	31.503	9.90%

Table 6: Run time, number of model parameters, in-sample and out-of-samples losses (units are in 10^{-2}), average estimated frequency on $\mathcal{D}$ (balance property).

We run the code of Listing 4 over 1'000 training epochs and we use a callback for receiving the model with the smallest validation loss $\mathcal{L}(\mathcal{V}, \boldsymbol{\beta}_{[t]})$. The results are presented on line (c1) of Table 6, the run time corresponds to the time elapsed to the callback. We see a clear decay

Listing 4: R code for a network of depth $d = 3$ and using embedding layers.

```
Design    <- layer_input(shape = c(7), dtype = 'float32', name = 'Design')
VehBrand <- layer_input(shape = c(1), dtype = 'int32',   name = 'VehBrand')
Region    <- layer_input(shape = c(1), dtype = 'int32',   name = 'Region')
LogVol    <- layer_input(shape = c(1), dtype = 'float32', name = 'LogVol')

BrEmb = VehBrand %>%
    layer_embedding(input_dim = 11, output_dim = 2, input_length = 1, name = 'BrEmb') %>%
    layer_flatten(name='Br_flat')

ReEmb = Region %>%
    layer_embedding(input_dim = 22, output_dim = 2, input_length = 1, name = 'ReEmb') %>%
    layer_flatten(name='Re_flat')

Network = list(Design, BrEmb, ReEmb) %>% layer_concatenate(name='concate') %>%
    layer_dense(units=20, activation='tanh',   name='hidden1') %>%
    layer_dense(units=15, activation='tanh',   name='hidden2') %>%
    layer_dense(units=10, activation='tanh',   name='hidden3') %>%
    layer_dense(units=1,  activation='linear', name='Network')

Response = list(Network, LogVol) %>% layer_add(name='Add') %>%
    layer_dense(units=1, activation='exponential', name = 'Response', trainable=FALSE,
                    weights=list(array(1, dim=c(1,1)), array(0, dim=c(1))))

model <- keras_model(inputs = c(Design, VehBrand, Region, LogVol), outputs = c(Response))
model %>% compile(loss = 'poisson', optimizer = 'adam')
```

of out-of-sample loss on the test data set $\mathcal{T}$ which illustrates that the network model has a clearly better performance w.r.t. our objective function (deviance loss). The in-sample loss is also clearly lower than the GLM and GAM ones, this illustrates that the improvement comes from missing interactions in the GLM and in the GAM, but not from a better marginal modeling of the individual feature components.

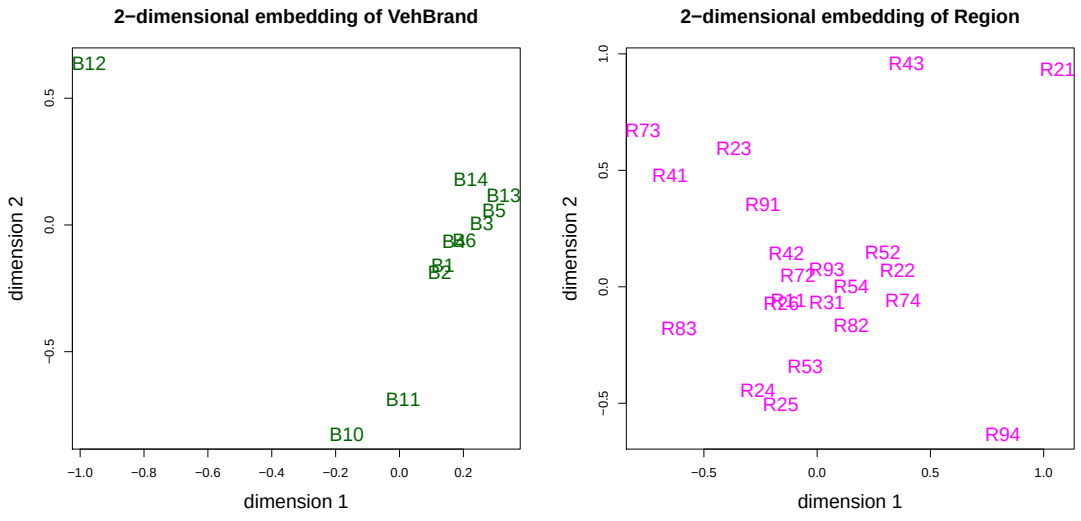

Figure 6: Learned 2-dimensional embeddings for the categorical variables `VehBrand` (lhs) and `Region` (rhs).

Since we use $b = 2$ dimensional embedding layers for the categorical features `VehBrand` and `Regions` we can illustrate the embeddings learned by the network. These are illustrated in Figure 6. In the car brands plot we see one outlier (brand B12), and quite some car brands seem to cluster, i.e. are similar for regression modeling. These similarities reflect that probably the car brand variable is over-parametrized if one uses dummy coding, this has exactly been observed in the ANOVA of Listing 3, and suggests to merge some of the car brands.

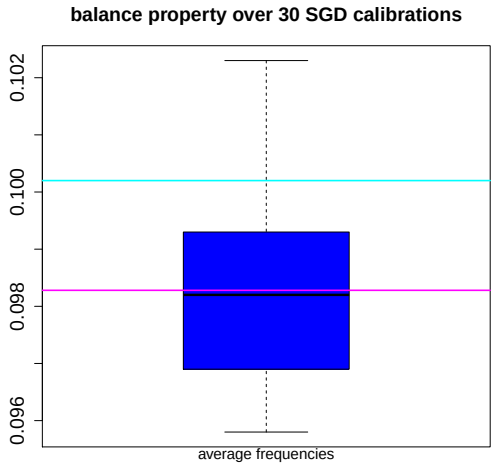

Figure 7: 30 SGD calibrations with different initial values $\boldsymbol{\beta}_{[0]}$.

More worrying is that the balance property fails to hold in the network regression model, see last column of Table 6. If we run 30 SGD calibrations with different initial values $\boldsymbol{\beta}_{[0]}$ (we control this by setting different seeds in `keras`) we receive quite some variability in the resulting models. In Figure 7 we analyze the balance property over the 30 different model calibrations which ranges from 9.59% to 10.23%, the average over these 30 runs is roughly 9.8% (in magenta color) and the balance condition would be at 10.02% (in cyan color). Thus, we receive a negative bias over the entire learning portfolio $\mathcal{D}$, here.

	run time	# param.	in-sample loss	out-of-sample loss	average frequency
(c1) Model NN seed 1	390s	792	30.411	31.503	9.90%
(c2) Model NN seed 2	390s	792	30.352	31.418	10.23%
(c3) Model NN seed 3	390s	792	30.315	31.500	9.61%

Table 7: Run time, number of model parameters, in-sample and out-of-samples losses (units are in 10^{-2}), average estimated frequency on $\mathcal{D}$ (balance property).

In Table 7 we provide the first three calibrations (having seeds 1,2 and 3). We do not only see considerable variations in the average frequencies but also in the corresponding loss figures. Of course, this is very undesirable and needs improvements.

3.4 Bias regularization and the balance property

3.4.1 Returning to generalized linear models

In Figure 7 and Table 7 we have observed that the balance property fails to hold, and misspecification of the overall level may be substantial. Following Wüthrich [52], it is comparably simple to correct for the failure of the balance property (if we have fully understood the connection between GLMs and network regression models). In the case where the link function h is not the canonical link, one should adjust the corresponding intercept variable $\beta_0^{(d+1)}$ of the output layer (readout).[13]

In the case of the canonical link function $h = g$ we can do even better to correct for the bias in a GLM/MLE way. This is going to be explained next. There are two important ingredients we would like to recall:

1. A network has the same structure w.r.t. the last hidden layer as a GLM, i.e. all previous network layers (only) serve the purpose of pre-processing $\boldsymbol{x}_i$ to network features $\boldsymbol{z}_i$, see (3.4) for representation learning.

2. A GLM has the balance property under the canonical link choice, see Corollary 2.10. Essentially, the balance property is guaranteed by the fact that we have full flexibility of choosing the intercept parameter appropriately.

The reason why networks typically do not have the balance property is that we exercise early stopping in network calibrations. In particular, this implies that we are not in a critical point of the deviance loss function w.r.t. to the readout parameter $\boldsymbol{\beta}^{(d+1)}$.

The proposed solution works as follows: Assume we have a network calibration which provides an optimal network parameter $\widehat{\boldsymbol{\beta}}^{\mathrm{NN}}$, see Section 3.3.1. This optimal network parameter gives us estimated network layers $\widehat{\boldsymbol{z}}^{(m)}(\cdot)$, $1 \leq m \leq d$, by inserting the estimate $\widehat{\boldsymbol{\beta}}^{\mathrm{NN}}$ for the network parameter $\boldsymbol{\beta} \in \mathbb{R}^r$. That is, we have *learned representations*, see also (3.4),

$$\boldsymbol{x}_i \;\mapsto\; \widehat{\boldsymbol{z}}_i = \left(\widehat{\boldsymbol{z}}^{(d)} \circ \cdots \circ \widehat{\boldsymbol{z}}^{(1)}\right)(\boldsymbol{x}_i) \;\in\; \mathcal{Z} = \{1\} \times \mathbb{R}^{q_d}. \tag{3.9}$$

Thus, we receive new (working) data $\{(Y_i, \widehat{\boldsymbol{z}}_i, o_i)\}_{i=1}^n$ with pre-processed feature information. Based on this pre-process data we then simply apply another GLM/MLE step on the feature space $\mathcal{Z}$. This step improves (in terms of in-sample deviance loss) the readout parameter estimate for $\boldsymbol{\beta}^{(d+1)}$ from the early stopped SGD estimation to a proper MLE based on the working data $\{(Y_i, \widehat{\boldsymbol{z}}_i, o_i)\}_{i=1}^n$.

Remarks.

- Bias regularized parameter estimation is done in two steps. In the first step we use the SGD algorithm to fit the entire parameter vector $\boldsymbol{\beta} \in \mathbb{R}^r$. The second step, uses MLE on the working data $\{(Y_i, \widehat{\boldsymbol{z}}_i, o_i)\}_{i=1}^n$, only estimating the readout parameter $\boldsymbol{\beta}^{(d+1)} \in \mathbb{R}^{q_d+1}$.

  The same could also be achieved in gradient descent methods, if we freeze the values of all parameters that do not belong to the output layer after a certain number of training epochs, and only train the readout parameter.

[13] Remark that also GLMs will be biased if one does not use the canonical link function, e.g. the log-link function in the gamma GLM typically leads to a bias.

- This additional GLM/MLE step may lead to over-fitting. If this is the case, one should either decrease the number of training epochs in the first step of the fitting process, or one should decrease the number of hidden neurons in the last hidden layer $\boldsymbol{z}^{(d)}$.
- An alternative to an additional GLM/MLE step is to introduce a regularization term that controls the bias, this has been proposed in formula (4.7) of Wüthrich [52].

3.4.2 MTPL example, revisited

We apply the two step calibration described in the previous subsection. We first apply the SGD algorithm with early stopping which provides the network models of Table 7. Afterwards, we apply the additional GLM/MLE step to the feature activations in the last hidden layer, see (3.9), the code is provided in Listing 5.

Listing 5: R code for bias regularization.

```
z.layer  <- keras_model(inputs=model$input, outputs=get_layer(model, 'hidden3')$output)
learn[,c("z1","z2","z3","z4","z5","z6","z7","z8","z9","z10")] <-
               data.frame(z.layer %>% predict(list(Design, VehBrand, Region, LogVol)))

glm(ClaimNb ~ z1 + z2 + z3 + z4 + z5 + z6 + z7 + z8 + z9 + z10,
              data=learn, offset=log(Exposure), family=poisson())
```

We perform this regularization step to the 3 solutions obtained in Table 7. The bias regularized versions are provided on lines (d1)-(d3) of Table 8.

	run time	# param.	in-sample loss	out-of-sample loss	average frequency
(a) homogeneous model	–	1	32.935	33.861	10.02%
(b2) Model GLM2	24s	48	31.257	32.149	10.02%
(c1) Model NN seed 1	390s	792	30.411	31.503	9.90%
(c2) Model NN seed 2	390s	792	30.352	31.418	10.23%
(c3) Model NN seed 3	390s	792	30.315	31.500	9.61%
(d1) Reg. Model NN seed 1	+7s	792	30.408	31.488	10.02%
(d2) Reg. Model NN seed 2	+7s	792	30.346	31.418	10.02%
(d3) Reg. Model NN seed 3	+7s	792	30.303	31.462	10.02%

Table 8: Run time, number of model parameters, in-sample and out-of-samples losses (units are in 10^{-2}), average estimated frequency on $\mathcal{D}$ (balance property).

We observe rather fast run times for this last calibration step. As expected, the in-sample losses decrease because we apply an additional MLE step to the output layer (readout). Interestingly, also the out-of-sample losses decrease which illustrates that the early stopped networks have not been over-fitting, yet. Finally, the last column verifies that the additional GLM/MLE step has been successful in achieving the balance property. Figure 8 confirms that we get consistently improved loss figures over the 30 runs of the SGD algorithm illustrated in Figure 7.

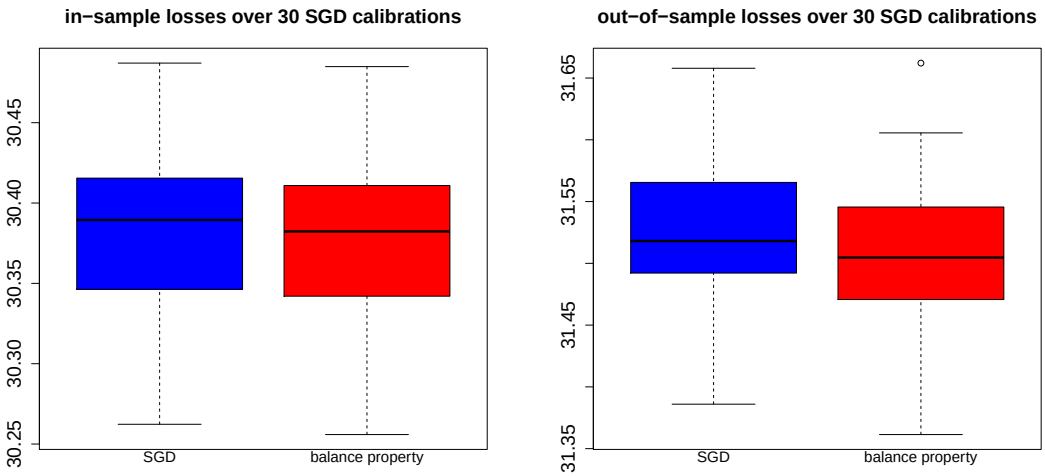

Figure 8: Comparison of the network calibrations and the corresponding bias regularized versions over 30 SGD algorithms with different seeds (lhs) in-sample losses and (rhs) out-of-sample losses.

3.5 Short summary of open points

We briefly summarize what is left to be done. This should be considered to be challenges to be solved in current state-of-the-art network approaches.

- We have improved the failure of the balance property, but we are still facing the problem of the non-uniqueness of the "optimal" network model. Frankly speaking, we do not have a good solution for this problem. Gradient descent methods focus on decreasing a given loss function which means that a complex problem is mapped to the real line. This mapping results in a (big) loss of information and structure (similar to risk measures in risk measurement). One way to tackle the question of optimal network models could be to characterize more broadly sets of solutions (parameters) having good properties, this may lead to more geometric approaches to the problem. Interesting papers on properties of loss surfaces are Draxler et al. [12] and Garipov et al. [18].

- Network solutions are often criticized for their non-explainability of models and results. In the language of Breiman [5], this is not the main focus of the algorithmic modeling culture, but instead we try to get the best possible prediction accuracy. In the following section we try to better understand the representations learned by the networks. This will give us some insight.

- So far, we have not been giving any advice for choosing the network architecture. Similar to the nonexistent best network parameter, there does not exist a best network architecture, but there are suitably good architectures.

- The network has a better predictive performance than our GLM, at least according to Table 8. These findings are non-constructive in the sense that we do not get any insight, yet, what particular modeling features our GLMs are lacking to make them as competitive as the networks. We are going to explore this point in more detail, below.

4 Learned network representations

4.1 Choice of network architecture

In this section we are going to choose a bigger network which will allow for a better interpretation of the results. From Figure 5 we observe that an architecture of depth $d = 3$ with $(q_1, q_2, q_3) = (20, 15, 10)$ hidden neurons does not severely over-fit to the training data $\mathcal{U}$ within the first 1'000 training epochs. As a rule of thumb, an architecture of depth $d = 3$ actively promotes interactions of 3 or 4 feature components which seems reasonable. Therefore, we keep this network depth of size 3. However, we are going to increase the number of hidden neurons in these hidden layers. We choose $(q_1, q_2, q_3) = (30, 30, 10)$ hidden neurons, i.e. for the illustrations below we keep the size of the last hidden layer, but we increase the first two hidden layers to 30 neurons each, which allows us for more complexity in the network regression function and, henceforth, for a bigger potential to over-fit to the training data $\mathcal{U}$.

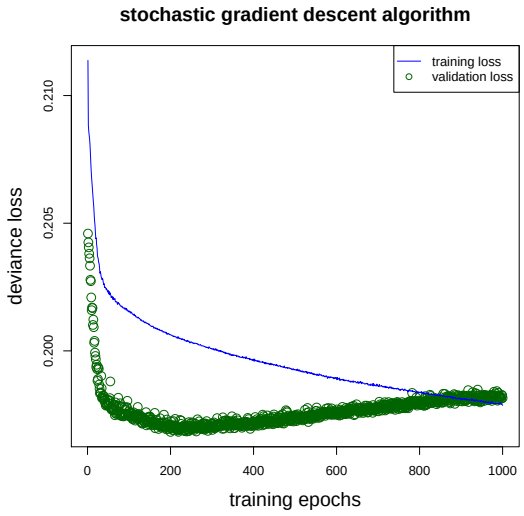

Figure 9: SGD algorithm over 1'000 training epochs on random batches $\mathcal{U}_s$ of size 10'000 on a network of depth $d = 3$ with $(q_1, q_2, q_3) = (30, 30, 10)$ hidden neurons.

We fit this bigger network to our French MTPL data. Figure 9 illustrates the decrease in in-sample training loss on $\mathcal{U}$ and the corresponding validation loss on $\mathcal{V}$. From this graph we conclude that we need about 250 training epochs (on batches $\mathcal{U}_s$ of size 10'000) before the SGD algorithm starts to over-fit to the training data, see Figure 9.
Table 9 provides the results on lines (e1)-(e3). This bigger network has $r = 1'667$ network parameters $\boldsymbol{\beta} \in \mathbb{R}^r$ and an optimal solution uses roughly 250 training epochs. The in-sample loss in the over-fitted network is clearly smaller than in the early stopped network, as can be seen from line (e1) versus line (e2) of Table 9. The bias regularized network on line (e3) is of similar out-of-sample predictive accuracy as the previously studied networks. The latter illustrates that, typically, many different network architectures are comparably good in the sense that they generalize comparably good to unseen data. The network architecture should have a sufficient complexity so that it is capable to reflect non-linearities in the predictor variables as well as interactions between these. This complexity is related to the number of knots needed

	run time	# param.	in-sample loss	out-of-sample loss	average frequency
(a) homogeneous model	–	1	32.935	33.861	10.02%
(b2) Model GLM2	24s	48	31.257	32.149	10.02%
(c1) Model NN (20-15-10) seed 1	390s	792	30.411	31.503	9.90%
(d1) Reg. Model NN (20-15-10) seed 1	+7s	792	30.408	31.488	10.02%
(e1) Model NN 30-30-10 over-fitting	1'280s	1'667	29.794	31.657	9.44%
(e2) Model NN 30-30-10 early stopped	320s	1'667	30.252	31.524	9.75%
(e3) Reg. Model NN 30-30-10 early stopped	+7s	1'667	30.246	31.506	10.02%

Table 9: Run time, number of model parameters, in-sample and out-of-samples losses (units are in 10^{-2}), average estimated frequency on $\mathcal{D}$ (balance property).

in GAMs (called effective degrees of freedom), as well as to the complexity of the interactions, see (3.1). In general, slightly more complex networks need less SGD descent steps (because of more flexibility), however, calculations of gradients are more expensive. Thus, typically there is a trade-off between flexibility in SGD steps and computational time of these steps.

Remark (reservoir computing). Recently, a method called reservoir computing has been developed. Though reservoir computing has been introduced for recurrent neural networks, it is closely related to (feed-forward) network regression modeling as introduced above. In reservoir computing one chooses a very deep network, i.e. a very large depth d of the network, with many neurons in these hidden layers. The network parameters of these layers are chosen at random and are not trained. The idea of this large randomized network is that we do not want to learn a controlled representation $(\widehat{\boldsymbol{z}}_i)_i$ of the features $(\boldsymbol{x}_i)_i$, but we would rather like to decompose the features into all its bits and pieces. This has some similarity to categorizing and dummy coding of continuous feature variables that do not fit to the functional forms prescribed by the linear predictor in GLMs. This huge decomposition of information is called reservoir and it contains all information of the features in a very segmented manner. In the second steps one trains the readout parameter, similar to bias regularization. One tries to capture all relevant information of the segmented features. In reservoir computing this is called to train the readout that maps $\widehat{\boldsymbol{z}}_i \mapsto \langle \boldsymbol{\beta}^{(d+1)}, \widehat{\boldsymbol{z}}_i \rangle$. In the case of the identity link function with quadratic loss function this can be done very efficiently, even if the dimension $q_d + 1$ of the last hidden layer is huge. To prevent from over-fitting this readout is trained using ridge regression regularization, introduced by Tikhonov [46]. For more on reservoir computing we refer to Verstraeten et al. [48].

4.2 Learned representation

In this section we would like to understand the learned representations $\boldsymbol{x}_i \mapsto \widehat{\boldsymbol{z}}_i$ of policies $i \in \mathcal{D}$, see (3.9). We analyze this for the network with $(q_1, q_2, q_3) = (30, 30, 10)$ hidden neurons, using exactly the calibration presented on lines (e1) and (e2) of Table 9.

The first 10 plots of Figure 10 (red and blue colors) illustrate the learned marginals of the representations in the last hidden layer $\widehat{\boldsymbol{z}}_i \in \mathbb{R}^{q_3+1}$ over all insurance policies $i \in \mathcal{D}$. This last hidden layer has $q_3 = 10$ hidden neurons plus an additional intercept component. Figure 10 gives the density plots for each of the $q_3 = 10$ hidden neurons over all policies $i \in \mathcal{D}$. The red graphs

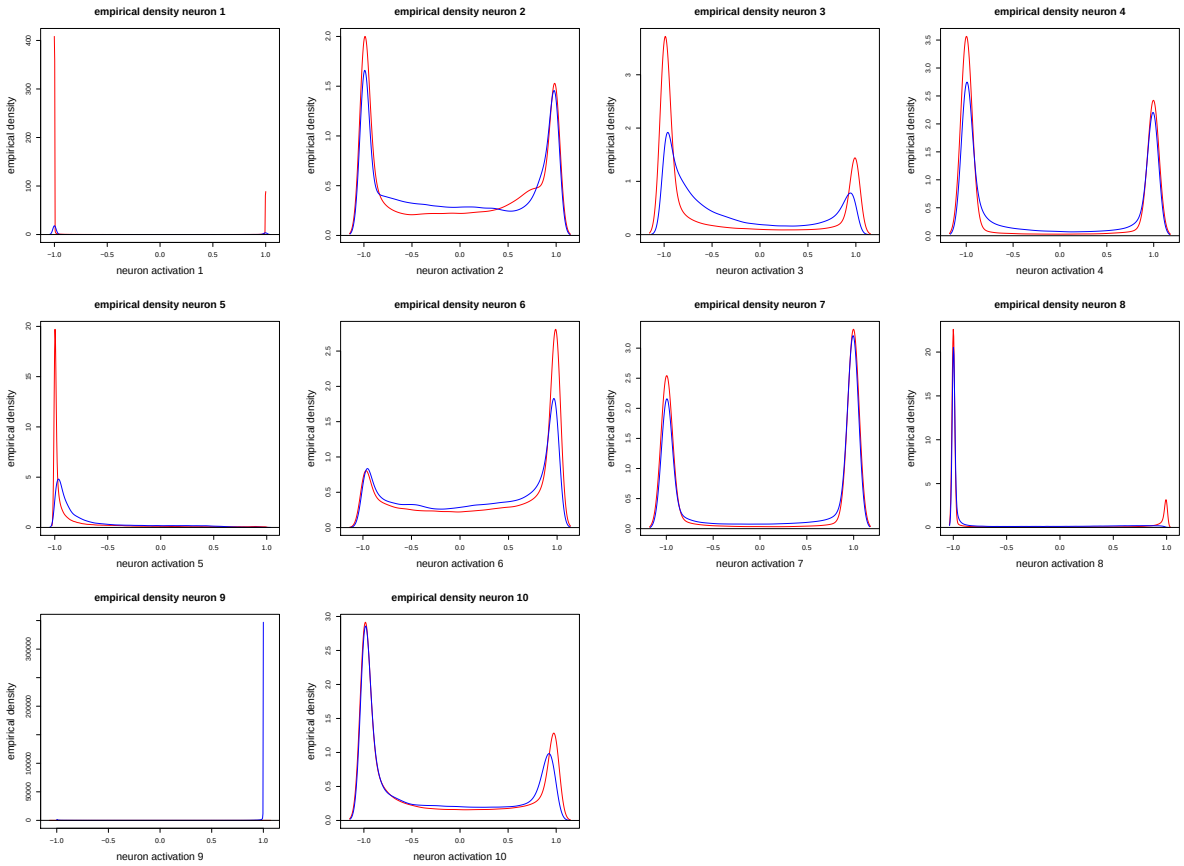

Figure 10: Density plots of the learned representations $\widehat{\boldsymbol{z}}_i \in \mathbb{R}^{10+1}$, $i \in \mathcal{D}$, in the last hidden layer: each plot shows one neuron, red color provides the over-fitted model, blue color the early stopped model, see lines (e1) and (e2) of Table 9.

show the densities of the over-fitted network of line (e1) in Table 9, and the blue graphs the early stopped ones of line (e2). In this example, we observe that the graphs are bi-modal, illustrating that the network tries to separate policies according to "good" or "bad", and in the over-fitted network this separation is becoming more pronounced. For instance, the 1st neuron (plot top-left) looks like a Bernoulli distribution providing two different groups of policies. Neurons $2, 3, 4, 6, 7, 10$ show a less extreme picture, whereas neuron 9 basically separates (probably) one value of one feature component from all other ones, in view of Figure 1, we suspect that this neuron takes care of the variable `VehAge`, see Figure 1 (top-right); this will be confirmed by Figure 11, below.

We believe that the plots in Figure 10 are also interesting in view of other machine learning methods such as regression trees and gradient boosting machines. We refer to Breiman et al. [6], Schapire [42] and Freund [16] for these methods. These methods try to partition the feature space $\mathcal{X}$ into homogeneous subsets such that the regression function can be described by a constant value on each subset. That is, the subsets are homogeneous w.r.t. the regression modeling task. This is quite similar to the bi-modal plots of Figure 10 because the two modes also describe a partitioning of the original feature space, but rather in a soft than a hard boundary

manner, because we do not have a pure black and white picture if we use continuous (or even differentiable) activation functions ψ. For recent developments of combining networks with trees and boosting we refer to Ke et al. [29].
We remark that in other examples we have received multi-modal density plots, thus, the bi-modal property will depend on the data and network architecture considered.

4.3 Explainability of learned representation

Our next goal is to better understand the learned representations $\widehat{\boldsymbol{z}}_i \in \mathcal{Z}$ of Figure 10 w.r.t. to the original features $\boldsymbol{x}_i \in \mathcal{X}$. In particular, we try to explain these plots relative to the observable feature information.

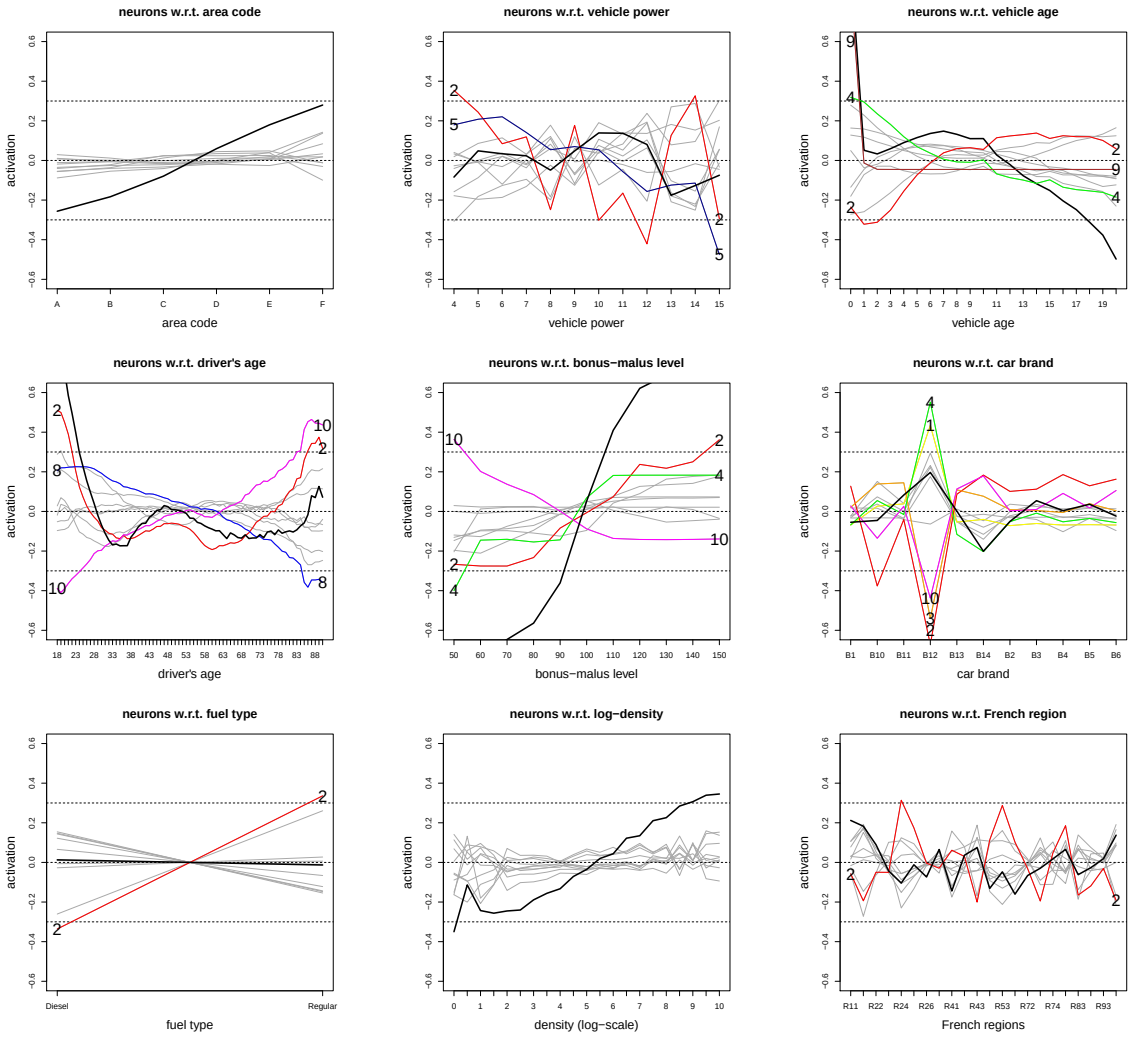

Figure 11: Colored and gray lines: observed average (scaled) neuron activations $a \mapsto \theta_j^*(a)$ for each neuron $j = 1, \ldots, q_3$ as a function of the observed feature values a; black solid lines: marginal network predictor $a \mapsto \theta^*(a)$ per feature value a (the y-scale is always the same).

We describe Figure 11 in detail; at first sight this figure might be a bit complicated. The aim is to analyze the learned feature activations $\widehat{\boldsymbol{z}}_i = (1, \widehat{z}_{1,i}, \ldots, \widehat{z}_{q_3,i})^\top \in \mathcal{Z}$ in each neuron

$j = 1, \ldots, q_3 = 10$ as a function of the original feature components $\boldsymbol{x}_i = (\texttt{Area}_i, \ldots, \texttt{Region}_i)^\top$, see Listing 1. For illustrative purposes, we choose the age of driver variable `DrivAge` to explain these plots. The driver's age variable is supported in $\mathcal{A} = \{18, \ldots, 90\}$ (these are the observed driver's ages in our portfolio). We can now study the average neuron activations in the last hidden layer for a given driver's age $a \in \mathcal{A}$ defined by

$$\overline{z}_j(a) = \frac{1}{|\{i : \texttt{DrivAge}_i = a\}|} \sum_{i:\, \texttt{DrivAge}_i = a} \widehat{z}_{j,i}, \qquad \text{for } j = 1, \ldots, q_3. \tag{4.1}$$

The function $a \mapsto \overline{z}_j(a)$ provides the average activation of hidden neuron j in the last hidden layer, as a function of driver's age $a \in \mathcal{A}$. Since these neurons have different readout parameter components $\widehat{\boldsymbol{\beta}}^{(d+1)} = (\widehat{\beta}_0^{(d+1)}, \ldots, \widehat{\beta}_{q_3}^{(d+1)})^\top \in \mathbb{R}^{q_3+1}$, we center and scale these average activations $\overline{z}_j(\cdot)$ (we use the bias regularized version of the readout parameter as described in Section 3.4). We define the average scaled neuron activation

$$a \;\mapsto\; \theta_j^*(a) = \widehat{\beta}_j^{(d+1)} \left(\overline{z}_j(a) - \frac{1}{|\mathcal{A}|} \sum_{b \in \mathcal{A}} \overline{z}_j(b) \right), \qquad \text{for } j = 1, \ldots, q_3. \tag{4.2}$$

These average scaled neuron activations $a \mapsto \theta_j^*(a)$ are plotted in colored or gray lines in Figure 11, each colored or gray line corresponds to one neuron $j = 1, \ldots, q_3$, and the different plots correspond to the different features (1st row) `Area`, `VehPower`, `VehAge`, (2nd row) `DrivAge`, `BonusMalus`, `VehBrand`, (3rd row) `VehGas`, `Density` and `Region`. We are going to explain the explicit choices of the colors, below. Note that we switch from the z-notation in (4.1) to the θ-notation in (4.2) because the latter is on the same scale as the network predictor (3.7).
The black solid lines illustrate the total activations for a given feature component, say, driver's age $a \in \mathcal{A}$, over all neurons, that is,

$$a \;\mapsto\; \theta^*(a) = \sum_{j=1}^{q_3} \theta_j^*(a).$$

This corresponds to the marginals of the network predictors θ_i, see (3.7), as a function of the corresponding feature values $a \in \mathcal{A}$. In particular, these black solid lines reflect the estimated marginals corresponding to the empirical marginals of Figure 1, but on the log-scale because we use the log-link function, here. We observe that, indeed, the shapes of the black lines in Figure 11 match the ones in Figure 1. This supports the network calibration.

Interpretations of Figure 11.

- The empirical plots in Figure 1 coincide with the black lines in Figure 11. Thus, we conjecture that the network has learned the right marginals.
- Each colored or gray line $a \mapsto \theta_j^*(a)$ in Figure 11 corresponds to one neuron $j = 1, \ldots, q_3$, and each plot considers a different feature component of the input variables $\boldsymbol{x} = (\texttt{Area}, \ldots, \texttt{Region})^\top$. These functions lie in the interval $(-0.7, 0.9)$. If the function $a \mapsto \theta_j^*(a)$ exits the interval $(-0.3, 0.3)$ it receives a color different from gray, because such large values indicate that the corresponding neuron j performs a particular task. Identical coloring will be used across all plots.

- For the feature components `Area` and `Density` we do not have extreme values in $a \mapsto \theta_j^*(a)$ and, therefore, all lines are in gray color. These feature components are more important for fine-tuning of frequency predictions.

- The feature component `VehPower` has two neurons that take extreme values, neuron $j = 2$ in red color and neuron $j = 5$ in darkblue color. Neuron $j = 2$ also has extreme values in several other feature components `VehAge`, `DrivAge`, `BonusMalus`, `VehBrand`, `VehGas`, `Region` (all in red color). This indicates that we have non-trivial interactions between these feature components; these interactions are captured by neuron $j = 2$. The darkblue neuron $j = 5$ does not have extreme values in other feature components, therefore, this neuron describes more the marginal shape of `VehPower`.

- Feature component `VehAge` has extreme values in neurons $j = 2, 4, 9$ (red, green, brown). The red one has already been described. The green one for $j = 4$, interacts more strongly with `BonusMalus` and `VehBrand` (also in green color). Of course, this makes perfect sense because all these feature components are related to the age of the car (and we expect non-trivial interactions).

  The brown neuron $j = 9$ is mainly used to describe the marginal of `VehAge`, and it supports our interpretation of neuron 9 in Figure 10.

- Feature component `DrivAge` has extreme values in neurons $j = 2, 8, 10$ (red, blue, magenta). The red one has already been described before. The magenta one $j = 10$ interacts with `BonusMalus` and `VehBrand` (also in magenta color). This makes perfect sense because all these variables are directly related to the age of driver.

  The blue neuron $j = 8$ is mainly used to describe the marginal of `DrivAge`.

- Similar considerations hold for the remaining feature components. We just mention that car brand B12 is special, which is not surprising in view of Figure 6.

Conclusions. The representations learned $\widehat{\boldsymbol{z}}_i \in \mathcal{Z}$ are explainable in terms of the original features $\boldsymbol{x}_i = (\texttt{Area}_i, \ldots, \texttt{Region}_i)^\top$. Some neurons take care of the marginal functions, and others explain strong interactions.

Remark. Alternatively to (4.1), we could also study partial dependence plots (PDPs) proposed by Friedman [17]. As described in Zhao–Hastie [56], the PDPs can be interpreted in terms of causality graphs. In particular, similar to the do-operation in causality analysis, they integrate out certain features (over their marginals) in order to analyze the sensitivity of the regression function in the remaining features. These PDPs will be different from our averaging (4.1) in cases where the features have dependence structures very different from the independent one.

4.4 Dimension reduction, principal components and LASSO

We have interpreted the learned representations $\widehat{\boldsymbol{z}}_i = (1, \widehat{z}_{1,i}, \ldots, \widehat{z}_{q_3,i})^\top \in \mathcal{Z}$ in the previous section. The next question in state-of-the-art statistical modeling is about dimension reduction. We do not aim to reduce the dimensions $(q_1, q_2) = (30, 30)$ of the first two hidden layers because we would like to have sufficient flexibility in representation learning, but we analyze

whether we need all $q_3 = 10$ neurons in the last hidden layer of the learned representations $\widehat{\boldsymbol{z}}_i = (1, \widehat{z}_{1,i}, \ldots, \widehat{z}_{q_3,i})^\top \in \mathcal{Z}$. In particular, this will give us information about the importance of each learned feature component.

4.4.1 Principal components analysis

We start by applying principal components analysis (PCA) which chooses a different orthonormal coordinate system of the Euclidean space $\mathbb{R}^{q_3}$ to represent $\widehat{\boldsymbol{z}}_i \in \mathcal{Z} \subset \{1\} \times \mathbb{R}^{q_3}$. For a reference to PCA, see Section 14.5.1 in Hastie et al. [23]. Looking at the marginal distributions in Figure 10 already raises some doubts whether this dimension reduction may work, because these plots do not (immediately) suggest that we are in a linear dependence situation (PCA is a linear dimension reduction technique).

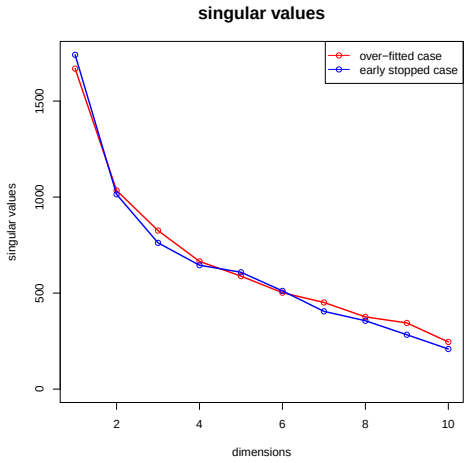

Figure 12: Singular values of the learned representations $\widehat{\boldsymbol{z}}_i \in \mathcal{Z} \subset \{1\} \times \mathbb{R}^{q_3}$.

The results are sketched in Figure 12. We study both the over-fitted and the early stopped models (e1) and (e2) of Table 9. Figure 12 gives the singular values of the PCA on the last hidden layer of the network (for the learned representations $\widehat{\boldsymbol{z}}_i \in \mathcal{Z} \subset \{1\} \times \mathbb{R}^{q_3}$) in the over-fitted model (red color) and in the early stopped model (blue color). The singular values look rather similar in both cases (which is not surprising in view of Figure 10). Interestingly, the graph has an almost linear shape (without a flattening of the slope). This indicates that there is not any rapidly vanishing importance into one particular direction, but the volatility in the learned representations is distributed across a $q_3 = 10$ dimensional space. This may suggest that we can still increase the size of the last hidden layer.

4.4.2 LASSO regularized generalized linear model

A popular way of variable selection is to employ LASSO (least absolute shrinkage and selection operator) regression. LASSO regression goes back to Tibshirani [45], see also Hastie et al. [24]. In the spirit of bias regularization in Section 3.4, we try to minimize the the following objective

function in $\boldsymbol{\beta}^{(d+1)}$

$$\mathcal{L}^{\eta}(\mathcal{D}, \boldsymbol{\beta}^{(d+1)}) = \frac{2}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} Y_i \left[\frac{v_i e^{\langle \boldsymbol{\beta}^{(d+1)}, \widehat{\boldsymbol{z}}_i \rangle}}{Y_i} - 1 - \log \left(\frac{v_i e^{\langle \boldsymbol{\beta}^{(d+1)}, \widehat{\boldsymbol{z}}_i \rangle}}{Y_i} \right) \right] + \eta \sum_{j=1}^{q_d} |\beta_j^{(d+1)}|, \qquad (4.3)$$

for a given *regularization parameter* $\eta \geq 0$ and learned representations $(\widehat{\boldsymbol{z}}_i)_{i \in \mathcal{D}}$. LASSO regression (4.3) considers an L^1-penalty term and the regularization parameter η controls its influence on the objective function. The L^1-penalty term has the property that for large regularization parameters η some components of the optimal estimate for $\boldsymbol{\beta}^{(d+1)}$ will be shrunk *exactly equal to zero*, this fact is nicely emphasized on the book cover of Hastie et al. [24]. Setting some components of $\boldsymbol{\beta}^{(d+1)}$ equal to zero implies that the corresponding neurons are "switched off" and, henceforth, are not used in the regression model. This is a way to rank the importance of the neurons in the last hidden layer. Note that the intercept parameter $\beta_0^{(d+1)}$ of $\boldsymbol{\beta}^{(d+1)}$ is excluded from regularization in (4.3), otherwise the estimated model will have a bias (similar as the early stopped SGD solution).

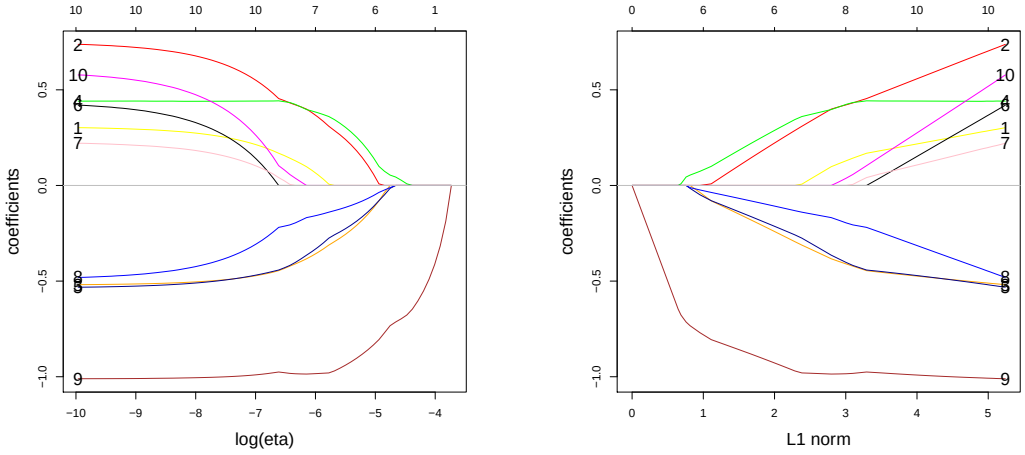

Figure 13: LASSO regression for different regularization parameters $\eta \in (0, 0.024]$ for feature selection: (lhs) x-axis is $\log(\eta)$ and (rhs) x-axis is $\sum_{j=1}^{q_d} |\beta_j^{(d+1)}|$; the numbers above the figures illustrate the number of non-zero regression parameters in $\beta_1^{(d+1)}, \ldots, \beta_{q_3}^{(d+1)}$; the numbers in the figures illustrate the labeling of the neurons $j = 1, \ldots, q_3 = 10$.

Figure 13 gives the magnitude of the regression parameters $\beta_1^{(d+1)}, \ldots, \beta_{q_3}^{(d+1)}$ as a function of the regularization parameter $\eta \in (0, 0.024]$ of the early stopped model on line (e3) of Table 9. The different colors illustrate the different neurons $j = 1, \ldots, q_3$ in the last hidden layer, the coloring of these neurons is identical to the one in Figure 11. Focusing on the figure on the left-hand side with $\log(\eta)$ on the x-axis, we see that one component $\beta_j^{(d+1)}$ after the other is shrunk to zero as η increases. The order is $j = 6$ (black), 7 (pink), 10 (magenta), 1 (yellow), 2 (red), 3 (orange), 5 (darkblue), 8 (blue), 4 (green) and 9 (brown), the latest being interpreted to be the most important one. This order reflects the magnitudes of the amplitudes in Figure 11. Note that neurons 6 and 7 are not colored in Figure 11 and, indeed, these are the two neurons that are dropped first in Figure 13.

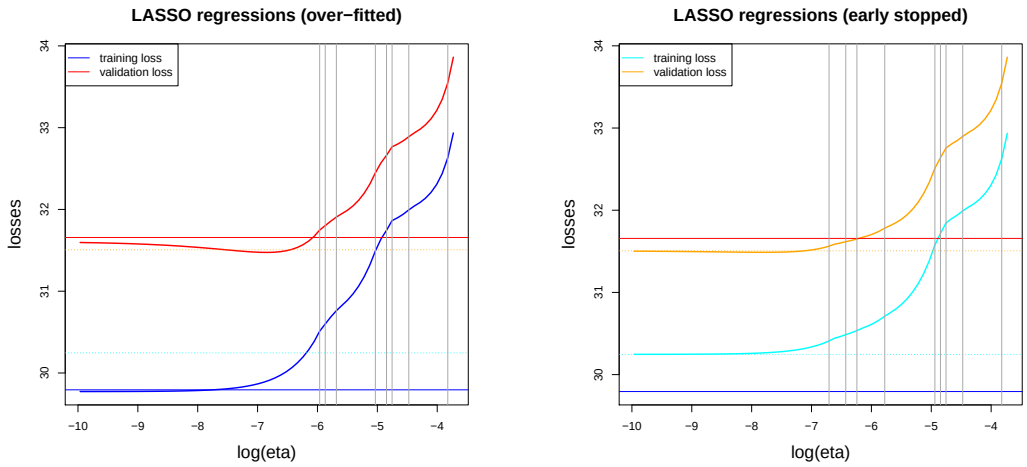

Figure 14: LASSO regression for different regularization parameters $\eta \in (0, 0.024]$: in-sample losses (blue and cyan) and out-of-sample losses (red and orange) in the over-fitted model (lhs) and the early stopped model (rhs) on lines (e1) and (e3) of Table 9; the horizontal lines give the values of lines (e1) and (e3) of Table 9 where no regularization is used; the y-scale is identical in both plots.

Figure 14 provides the resulting in-sample losses (blue and cyan) and out-of-sample losses (red and orange) of these models. The left-hand side gives the over-fitted model (line (e1) of Table 9) and the right-hand side the early stopped one (line (e3) of Table 9). The vertical gray lines show the regularization parameter values η's at which the components $\beta_1^{(d+1)}, \ldots, \beta_{q_3}^{(d+1)}$ are shrunk exactly to zero. We observe that in the early stopped model the first component $j = 6$ is set to zero at $\log(\eta) = \log(0.0012) = -6.707$. We also observe that the out-of-sample losses (red and orange) still decrease with decreasing η below 0.0012 which basically says that we should not drop any of the $q_3 = 10$ neurons. We also observe that the over-fitted version can be improved here by applying LASSO regularization. In our situation the LASSO regularization of the over-fitted model provides a similarly good predictive model as the early stopped one. However, the latter is not necessarily the case, it might be that the last hidden layer is already too over-fitted so that a readout with regularization is not able to remove this over-fitting. This is may happen, for instance, if we have very spiky bi-modal plots in Figure 10.

4.5 Drop-out layers

Finally, we consider network calibration using drop-out layers after each hidden layer. Drop-out layers have been introduced by Srivastava et al. [44] and Wager et al. [49]. Drop-out layers are auxiliary layers that prevent networks from over-fitting if applied properly. A drop-out layer temporarily removes neurons at random (and independently from each other) during the SGD steps. This way it can be achieved that a single neuron cannot be over-trained to a particular task, but the composite of all neurons has to perform sufficiently well. More intuitively speaking, drop-out layers may also have the interpretation of averaging over different network architectures (because some edges are removed at random). More rigorously, one can prove in a Gaussian situation that drop-out layers act as ridge regression, see Section 18.6 in Efron–Hastie [13]. For

the implementation in R we refer to Listing 6. This listing also provides the drop-out probabilities used in our example.

Listing 6: R code for the implementation of drop-out layers.

```
Network = list(Design, BrEmb, ReEmb) %>% layer_concatenate(name='concate') %>%
    layer_dense(units=30, activation='tanh',   name='hidden1') %>%
    layer_dropout(rate=0.2) %>%
    layer_dense(units=30, activation='tanh',   name='hidden2') %>%
    layer_dropout(rate=0.2) %>%
    layer_dense(units=10, activation='tanh',   name='hidden3') %>%
    layer_dropout(rate=0.1) %>%
    layer_dense(units=1,  activation='linear', name='Network')
```

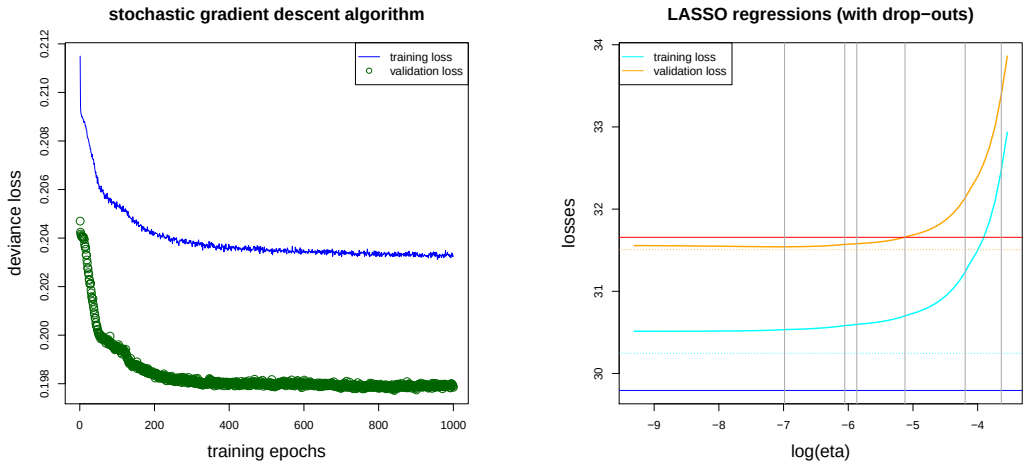

Figure 15: (lhs) SGD algorithm over 1'000 training epochs of random batches of size 10'000 on a network with $(q_1, q_2, q_3) = (30, 30, 10)$ and using drop-out layers; (rhs) LASSO regression on learned representations using drop-outs.

In Figure 15 (lhs) we plot the SGD training on this network with drop-outs, according to the architecture of Listing 6. Obviously, the algorithm does not over-fit during 1'000 training epochs, and the resulting models are provided on lines (f1) and (f2) of Table 10. The resulting models are slightly worse than our previous models, which indicates that either the drop-out rates are too high or the network architecture is too small for the given drop-out probabilities to cope with the other networks.

Model (f1) from Table 10 again provides representations $\widehat{\boldsymbol{z}}_i \in \mathcal{Z} \subset \{1\} \times \mathbb{R}^{q_3}$ in the last hidden layer which we could analyze in more detail, as before. The common finding is that this learned representation has less extreme values, because a neuron could drop-out and its job has to be taken over by the remaining neurons. As a consequence, we also expect that there are more redundancies in the learned representations $\widehat{\boldsymbol{z}}_i$. Indeed, this is the case. If, for instance, we perform LASSO regression (4.3), we receive Figure 15 (rhs). This figure indicates that one neuron in the last hidden layer is superfluous because after the vertical gray line at $\log(\eta) = \log(0.0091) = -7$ (which gives the removal of the first neuron in the last hidden layer) the out-of-sample loss does not decrease any further.

	run time	# param.	in-sample loss	out-of-sample loss	average frequency
(a) homogeneous model	–	1	32.935	33.861	10.02%
(b2) Model GLM2	24s	48	31.257	32.149	10.02%
(c1) Model NN (20-15-10) seed 1	390s	792	30.411	31.503	9.90%
(d1) Reg. Model NN (20-15-10) seed 1	+7s	792	30.408	31.488	10.02%
(e1) Model NN 30-30-10 over-fitting	1'280s	1'667	29.794	31.657	9.44%
(e2) Model NN 30-30-10 early stopped	320s	1'667	30.252	31.524	9.75%
(e3) Reg. Model NN 30-30-10 early stopped	+7s	1'667	30.246	31.506	10.02%
(f1) Model NN 30-30-10 drop-out (0.2,0.2,0.1)	2'040s	1'667	30.526	31.572	9.86%
(f2) Reg. Model NN 30-30-10	+7s	1'667	30.513	31.561	10.02%
(f3) LASSO NN 30-30-10 ($\eta = 0.0091$)	12s	1'667	30.533	31.541	10.02%

Table 10: Run time, number of model parameters, in-sample and out-of-samples losses (units are in 10^{-2}), average estimated frequency on $\mathcal{D}$ (balance property).

5 Improving generalized linear model predictions

In this section we further explore the question of how to improve the original GLM, or how network modeling can support us to detect weaknesses in GLMs. The idea is to let the GLM and the network compete, which can also be seen as boosting the GLM with (non-linear) network features. Interestingly, already Tukey [47] has been considering similar ideas in the 1970s in a different context (called "twicing"), see Hoaglin [25].

5.1 Nesting generalized linear models into networks

The crucial observation is that the linear predictor (2.11) of the GLM and the network predictor (3.1) have exactly the same structural form. Therefore, we can easily combine them into a joint predictor

$$(o_i, \boldsymbol{x}_i) \;\mapsto\; h\left(\mathbb{E}[Y_i]\right) = o_i + \left\langle \boldsymbol{\beta}^{\mathrm{GLM}}, \boldsymbol{x}_i \right\rangle + \left\langle \boldsymbol{\beta}^{(d+1)}, \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}_i) \right\rangle, \tag{5.1}$$

with parameter vector $(\boldsymbol{\beta}^{\mathrm{GLM}}, \boldsymbol{\beta})^\top \in \mathbb{R}^{q_0+1+r}$. This proposal has been studied in Schelldorfer–Wüthrich [43] under the name combined actuarial and neural network (CANN) model, see also Wüthrich–Merz [54]. Ansatz (5.1) allows us to let the linear predictor of the GLM compete with the non-linear one of the network. This architecture is illustrated in Figure 16. Basically, it adds a so-called *skip connection* to the network that directly connects the input with the output layer, and the GLM is packed into this skip connection.

A naïve way of model calibration of (5.1) would be to just use the machinery introduced in Section 3: the SGD algorithm would provide an early stopped fitted model, and prediction could be done. But, actually, we can do better. We can reduce the run time of the SGD calibration substantially by choosing a smart initial value $(\boldsymbol{\beta}_{[0]}^{\mathrm{GLM}}, \boldsymbol{\beta}_{[0]})^\top \in \mathbb{R}^{q_0+1+r}$ for the SGD algorithm. Note that if we initialize the SGD algorithm by $\boldsymbol{\beta}_{[0]}^{\mathrm{GLM}} = \widehat{\boldsymbol{\beta}}^{\mathrm{MLE}}$ (this is the GLM part in the skip connection, see (5.1)) and the network readout parameter by $\boldsymbol{\beta}_{[0]}^{(d+1)} = \mathbf{0}$, then the SGD algorithm exactly starts in the MLE fitted GLM, according to Section 2.2.2. Thus, the starting value of the SGD algorithm exactly provides the in-sample loss received from the

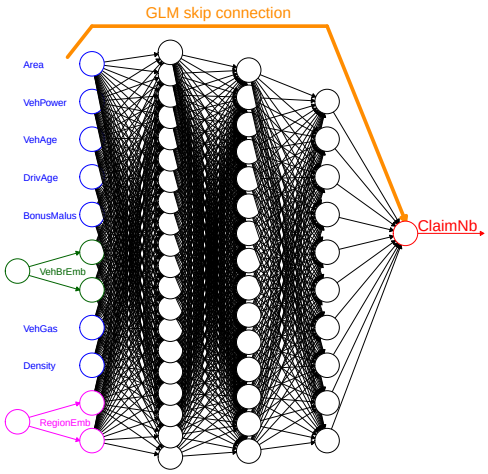

Figure 16: CANN approach: combination of linear GLM predictor and non-linear network predictor into a single network using a skip connection for the GLM.

MLE fitted GLM. Running the SGD algorithm may now result either in a major decrease of this loss or this loss may remain more or less constant. The former indicates that the GLM can be substantially improved, the latter means that the GLM is competitive because the network features do not really improve the model.

In the examples below we explore a variant of this combined approach. Namely, we declare the initial value $\boldsymbol{\beta}_{[0]}^{\text{GLM}} = \widehat{\boldsymbol{\beta}}^{\text{MLE}}$ in the skip connection to be *non-trainable*. This means that we treat the GLM model as known a priori differences in an offset, and we build the network around this offset model. Define

$$\widehat{o}_i^{\text{GLM}} = o_i + \left\langle \widehat{\boldsymbol{\beta}}^{\text{MLE}}, \boldsymbol{x}_i \right\rangle, \tag{5.2}$$

and consider the network predictor, see (5.1),

$$(\widehat{o}_i^{\text{GLM}}, \boldsymbol{x}_i) \;\mapsto\; h\left(\mathbb{E}[Y_i]\right) = \widehat{o}_i^{\text{GLM}} + \left\langle \boldsymbol{\beta}^{(d+1)}, \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}_i) \right\rangle. \tag{5.3}$$

This network can be implemented and fitted exactly as in Listing 4, just by correspondingly modifying the offsets. Of course, the bias regularization step, as described in Listing 5, should be exploited during model calibration.

5.2 French MTPL example, revisited

We return to our GLM example of Section 2.3.4, in particular, we choose Model GLM2 to define the GLM calibrated offsets $\widehat{o}_i^{\text{GLM}}$ as defined in (5.2). Using these modified offsets, we can boost the GLM using the CANN architecture (5.3), see also Figure 16.

Figure 17 illustrates the SGD training on the training data $\mathcal{U}$ (blue color) and the validation loss on $\mathcal{V}$ (green color). The set-up is identical to Figure 5 except that we start the training of the network in the MLE fitted GLM. Indeed, as claimed above, we receive a much faster convergence, and this network starts to over-fit after roughly 150 SGD steps. Thus, the CANN approach leads to better run times due to the fact that we already start the algorithm in a reasonable predictive model. We then apply the bias regularization step to the early stopped

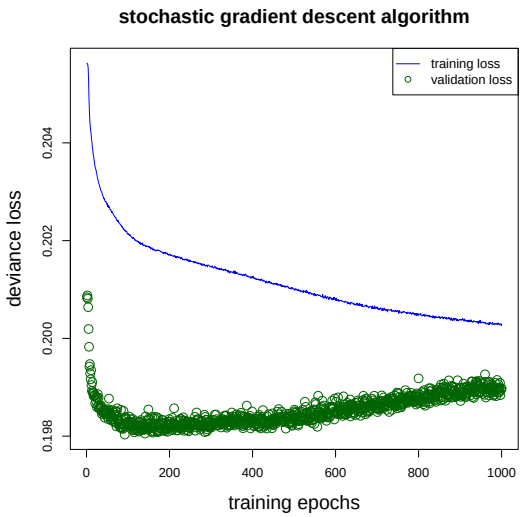

Figure 17: SGD algorithm over 1'000 training epochs on random batches $\mathcal{U}_s$ of size 10'000 on a CANN architecture of depth $d = 3$ with $(q_1, q_2, q_3) = (20, 15, 10)$ hidden neurons and modified offsets (5.2).

SGD parametrization, as suggested in Section 3.4. The resulting model is summarized on line (g1) of Table 11.

	run time	# param.	in-sample loss	out-of-sample loss	average frequency
(a) homogeneous model	–	1	32.935	33.861	10.02%
(b2) Model GLM2	24s	48	31.257	32.149	10.02%
(c1) Model NN (20-15-10) seed 1	390s	792	30.411	31.503	9.90%
(d1) Reg. Model NN (20-15-10) seed 1	+7s	792	30.408	31.488	10.02%
(g1) Reg. CANN (20-15-10) seed 1	100s	48+792	30.564	31.639	10.02%

Table 11: Run time, number of model parameters, in-sample and out-of-samples losses (units are in 10^{-2}), average estimated frequency on $\mathcal{D}$ (balance property).

We observe that the CANN boosting step of Model GLM2 leads to an out-of-sample loss decrease from 32.149 to 31.639. This indicates that important regression structure is missing in Model GLM2 that could be found by the CANN approach (5.3).
Another observation is that the CANN approach on line (g1) of Table 11 has a slightly worse performance than the original regularized network on line (d1). This is a common observation over several runs, and probably it is caused by the fact that the network is build around the GLM. That is, the network can improve weaknesses of the GLM, but it does not start "on greenfield" which might be a small disadvantage for the resulting predictive model. Unfreezing of the GLM parameters may help in further training, but only if the current GLM parametrization is not stuck too much in a (locally) non-optimal point of the loss function.

5.3 Finding missing interactions in generalized linear models

In this section we explore network predictor (5.3) to find missing *pair-wise* interactions and other weaknesses in our benchmark Model GLM2. In view of (5.3), this can easily be achieved. Our features $\boldsymbol{x}_i = (1, x_{i,1}, \ldots, x_{i,q})^\top$ have $q = q_0$ explanatory variables (excluding the intercept variable). We may now explore whether our Model GLM2 is missing a bivariate interaction, say, between $x_{i,j}$ and $x_{i,k}$. For this we modify (5.3) as follows

$$(\widehat{o}_i^{\text{GLM}}, (x_{i,j}, x_{i,k})^\top) \;\mapsto\; h\left(\mathbb{E}[Y_i]\right) = \widehat{o}_i^{\text{GLM}} + \left\langle \boldsymbol{\beta}^{(d+1)}, \left(\boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(x_{i,j}, x_{i,k})^\top \right\rangle, \qquad (5.4)$$

i.e. we only send the two selected feature components through the network architecture. We can do this for every pair in $\boldsymbol{x}_i$, and the corresponding results are presented in Figure 18.
In Figure 18 we explore all pair-wise interactions of the features given in Listing 1, the order of the components is exactly the same as on lines 6-14 in that listing. The losses of the pairs plotted in black color do not undergo a major decrease, whereas the blue and cyan pairs indicate model improvements relative to Model GLM2. It is conspicuous that all bivariate plots including the variable `BonusMalus` are affected (cyan color). This is a sign that the variable `BonusMalus` has not been modeled sufficiently well in Model GLM2. The corresponding enhancement is illustrated in Listing 7, we call this Model GLM3.

Listing 7: R code to fit Model GLM3.

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

This enhanced GLM replaces the log-linear modeling of `BonusMalus` by additional log-square, log-cubic and log-quartic terms, as well as a linear term (under the log-link choice), similarly to the age of driver modeling, see (2.25). We find that these terms improve the benchmark model, and the resulting losses are presented on line (b3) of Table 12.

	run time	# param.	in-sample loss	out-of-sample loss	average frequency
(a) homogeneous model	–	1	32.935	33.861	10.02%
(b2) Model GLM2	24s	48	31.257	32.149	10.02%
(b3) Model GLM3	22s	52	31.173	32.105	10.02%
(c1) Model NN (20-15-10) seed 1	390s	792	30.411	31.503	9.90%
(d1) Reg. Model NN (20-15-10) seed 1	+7s	792	30.408	31.488	10.02%
(g1) Reg. CANN (20-15-10) seed 1	100s	48+792	30.564	31.639	10.02%

Table 12: Run time, number of model parameters, in-sample and out-of-samples losses (units are in 10^{-2}), average estimated frequency on $\mathcal{D}$ (balance property).

We conclude that Listing 7 provides an improved benchmark Model GLM3. We perform the pair-wise interaction analysis (5.4) again on the improved benchmark for all pairs that involve

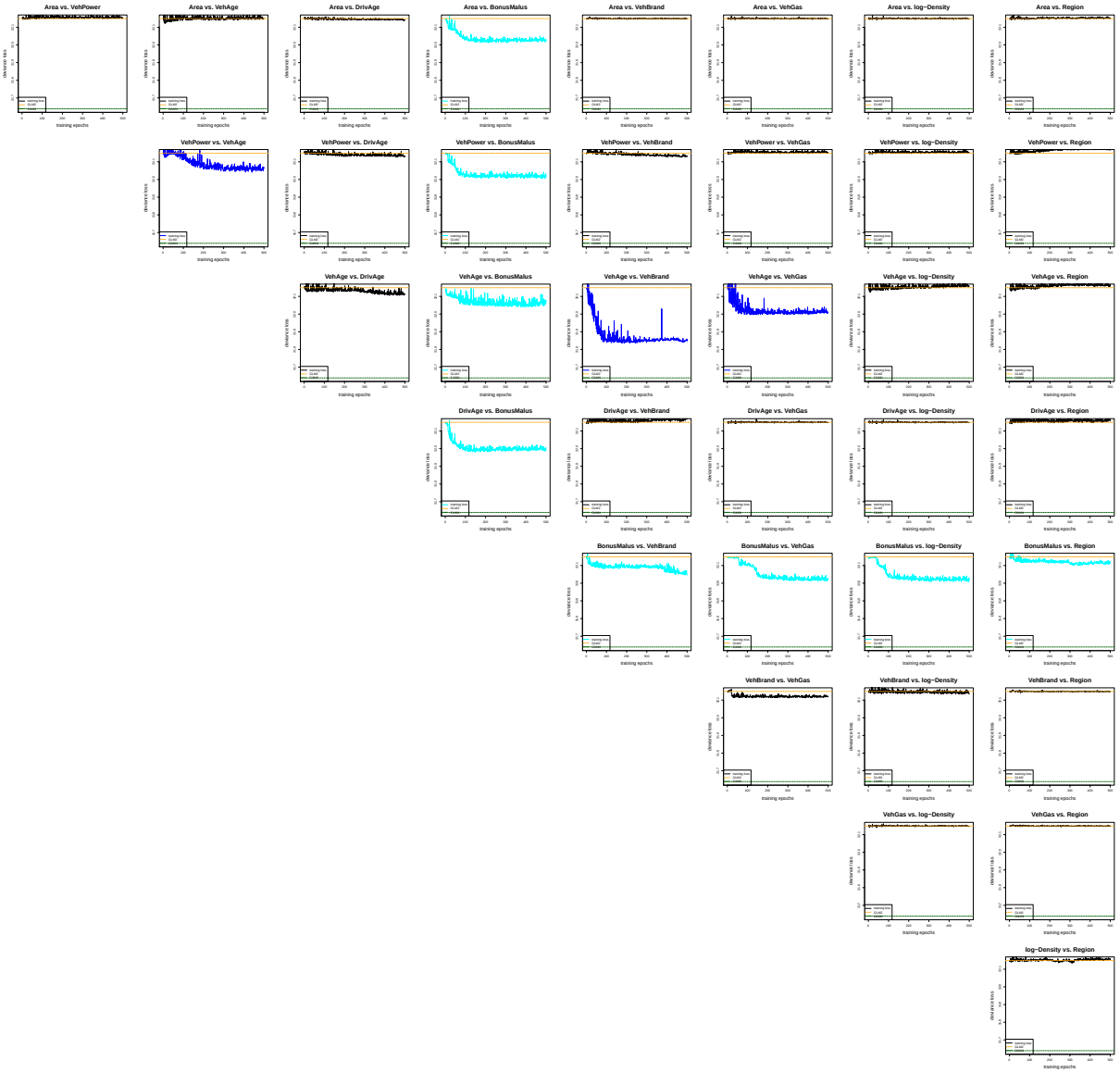

Figure 18: Exploring all pair-wise interactions: out-of-sample losses over 500 gradient descent epochs for all pairs of feature components, the orange dotted line shows Model GLM2 (the scale on the y-axis is identical in all plots).

the variable `BonusMalus`, and we modify the offsets $\widehat{o}_i^{\text{GLM}}$ according to Model GLM3. The results are presented in Figure 19.

In the step from Figure 18 to Figure 19 we replace all pair-wise plots that include the variable `BonusMalus`: the former figure uses Model GLM2 as modified offsets and the latter Model GLM3. In particular, we replace in Figure 19 to orange GLM2 losses by the red improved GLM3 losses. In this enhanced version we observe that there are 4 pairs left for which the losses substantially decrease w.r.t. their starting values (blue plots in Figure 19). These are the pairs `VehPower-VehAge`, `VehAge-VehBrand`, `VehAge-VehGas` and `DrivAge-BonusMalus`. In particular, this analysis tells us that these interactions cannot be modeled satisfactorily by a multiplicative

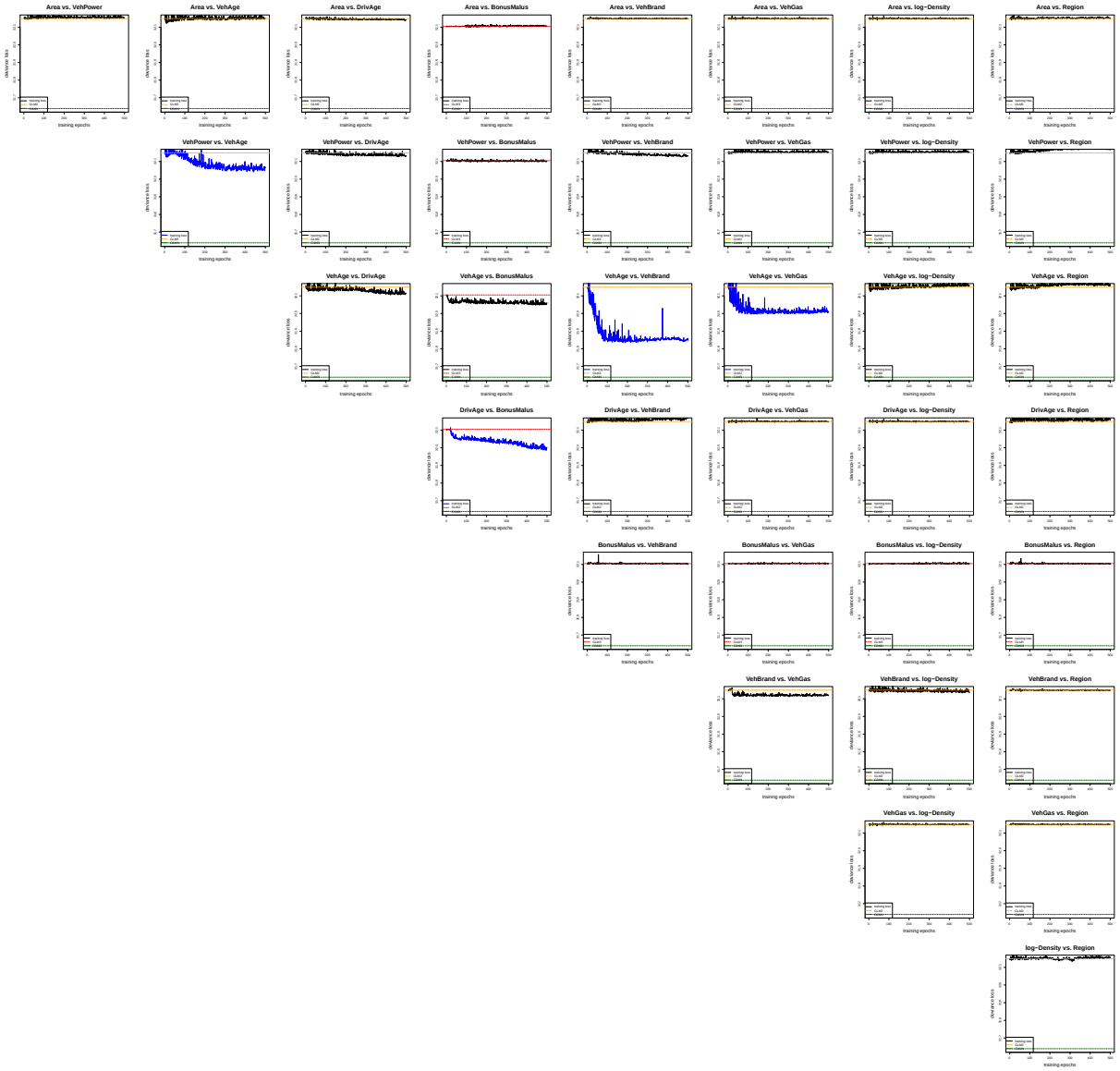

Figure 19: Exploring all pair-wise interactions: out-of-sample losses over 500 gradient descent epochs for all pairs of feature components, the orange dotted line shows Model GLM2 and the red dotted line Model GLM3 (the scale on the y-axis is identical in all plots).

approach (recall that we use the log-link). Intuitively, these missing interactions found make good sense because, for instance, a driver aged 18 cannot be on the lowest bonus-malus level because this level can only be reached after several years of driving without an accident. Remark that these interactions have also been found in Figure 11, however, in a less transparent way.
Our next aim is to enhance the GLM by implementing the missing pair-wise interactions. The implementation of `VehPower-VehAge`, `VehAge-VehBrand`, `VehAge-VehGas` is simply done by a pair-wise modeling of the corresponding variables, see lines 1 and 2 in Listing 8. The interaction modeling of `DrivAge-BonusMalus` is less clear because we can let multiple terms interact. The chosen interaction is illustrated on line 3 of Listing 8.

Listing 8: R code to fit Model GLM4.

```
glm(ClaimNb ~ AreaGLM + VehPowerGLM*VehAgeGLM + VehAgeGLM*VehBrand
                + VehGas*VehAgeGLM + DensityGLM + Region +
                + BonusMalusGLM*log(DrivAge) + log(BonusMalusGLM) +  I(BonusMalusGLM^2)
                + I(BonusMalusGLM^3) + I(BonusMalusGLM^4),
                + DrivAge + I(DrivAge^2) + I(DrivAge^3) + I(DrivAge^4),
              data=learn, offset=log(Exposure), family=poisson())
```

	run time	# param.	in-sample loss	out-of-sample loss	average frequency
(a) homogeneous model	–	1	32.935	33.861	10.02%
(b2) Model GLM2	24s	48	31.257	32.149	10.02%
(b3) Model GLM3	22s	52	31.173	32.105	10.02%
(b4) Model GLM4	36s	85	30.772	31.672	10.02%
(c1) Model NN (20-15-10) seed 1	390s	792	30.411	31.503	9.90%
(d1) Reg. Model NN (20-15-10) seed 1	+7s	792	30.408	31.488	10.02%
(g1) Reg. CANN (20-15-10) seed 1	100s	48+792	30.564	31.639	10.02%

Table 13: Run time, number of model parameters, in-sample and out-of-samples losses (units are in 10^{-2}), average estimated frequency on $\mathcal{D}$ (balance property).

It turns out that the resulting model (called Model GLM4) gets close to the predictive power of the networks, see line (b4) in Table 13. Thus, the CANN framework helps us to substantially improve our benchmark GLM, and using Model GLM4 we are back in the GLM framework, having a model that is competitive on the one hand, gained insights on the other hand and, moreover, we can now explore all statistical techniques of model and parameter selection. At this point we should emphasize that we start to change the philosophy of the two modeling cultures introduced by Breiman [5], namely, we introduce the paradigm of *model learning* in the sense that using the algorithmic modeling culture we learn the data generating model.
Next we provide the ANOVA of Model GLM4 in Listing 9. This ANOVA illustrates that parameter reduction should be explored. Lines 20 and 25-28 give the function `DrivAge` $\mapsto F($`DrivAge`$)$ for driver's age with function $F(\cdot)$ given in (2.25), and lines 19 and 21-24 give the corresponding function `BonusMalusGLM` $\mapsto F($`BonusMalusGLM`$)$ for the bonus-malus level. Interestingly, lines 29-32 of Listing 9 clearly illustrate the added value of considering the corresponding interactions. We analyze these interactions in more detail in Figure 20.
In Figure 20 we illustrate the observed marginal frequencies of `VehAge` (top row and bottom left) and `BonusMalus` (bottom right) split w.r.t. different cohorts of `VehGas`, `VehBrand` and `VehPower` (for `VehAge`) and `DrivAge` (for `BonusMalus`). These are the (missing) pair-wise interactions in Model GLM3 which have been identified by Figure 19. If the multiplicative interaction structure in Model GLM3 would be correct, then the cohorts in these plots should be roughly ordered (modulo non-uniformities in portfolio structures and other interactions). That is, the empirical marginal plots per cohort provide us with another tool of identifying missing pair-wise interactions, though not as easily visible as in Figure 19. The interpretation of the `VehAge` plots is that mainly vehicle age 0 causes difficulties in using a multiplicative structure, which may suggest that we model a joint interaction between all four variables `VehAge`, `VehGas`, `VehBrand` and `VehPower`. The interpretation of `BonusMalus` versus `DrivAge` is more difficult because the

Listing 9: ANOVA analysis of variables in Model GLM4.

```
Analysis of Deviance Table

Model: poisson, link: log

Response: ClaimNb

Terms added sequentially (first to last)


                           Df Deviance Resid. Df Resid. Dev
NULL                                       610211     200974
AreaGLM                     1    669.4    610210     200305
VehPowerGLM                 5    154.2    610205     200151
VehAgeGLM                   2   4585.2    610203     195566
VehBrand                   10     85.3    610193     195480
VehGas                      1     51.6    610192     195429
DensityGLM                  1     27.6    610191     195401
Region                     21    172.9    610170     195228
BonusMalusGLM               1   3813.8    610169     191415
log(DrivAge)                1    279.9    610168     191135
log(BonusMalusGLM)          1     13.7    610167     191121
I(BonusMalusGLM^2)          1     61.8    610166     191059
I(BonusMalusGLM^3)          1    282.7    610165     190776
I(BonusMalusGLM^4)          1     48.2    610164     190728
DrivAge                     1      0.9    610163     190727
I(DrivAge^2)                1    153.0    610162     190574
I(DrivAge^3)                1    287.6    610161     190287
I(DrivAge^4)                1     64.6    610160     190222
VehPowerGLM:VehAgeGLM      10    393.7    610150     189828
VehAgeGLM:VehBrand         20   1582.9    610130     188246
VehAgeGLM:VehGas            2    393.0    610128     187853
BonusMalusGLM:log(DrivAge)  1     77.7    610127     187775
```

plots are very volatile, but obviously there is not a fixed order in the lines w.r.t. driver age classes 1 to 7 (these are identical to driver age classes in Model GLM1).

	run time	# param.	in-sample loss	out-of-sample loss	average frequency
(a) homogeneous model	–	1	32.935	33.861	10.02%
(b2) Model GLM2	24s	48	31.257	32.149	10.02%
(b3) Model GLM3	22s	52	31.173	32.105	10.02%
(b4) Model GLM4	36s	85	30.772	31.672	10.02%
(b5) Model GLM5	1'336s	405	30.612	31.587	10.02%
(c1) Model NN (20-15-10) seed 1	390s	792	30.411	31.503	9.90%
(d1) Reg. Model NN (20-15-10) seed 1	+7s	792	30.408	31.488	10.02%
(g1) Reg. CANN (20-15-10) seed 1	100s	48+792	30.564	31.639	10.02%

Table 14: Run time, number of model parameters, in-sample and out-of-samples losses (units are in 10^{-2}), average estimated frequency on $\mathcal{D}$ (balance property).

If we let the four variables `VehAge`, `VehGas`, `VehBrand` and `VehPower` interact in a GLM and choose all other terms as in Listing 8, we receive the results on line (b5) of Table 14. This quadruple interaction implementation is very much brute-force, it results in an improvement in

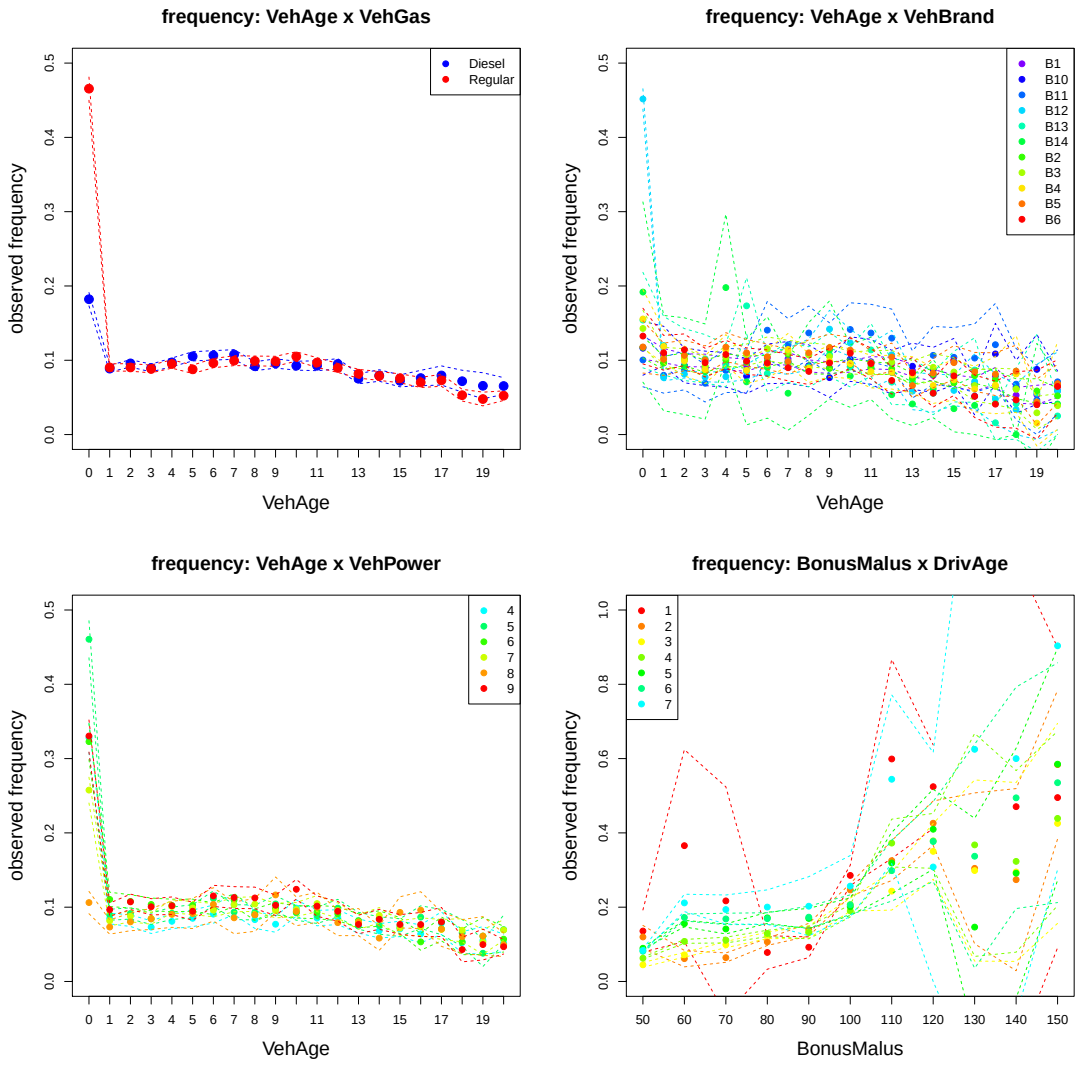

Figure 20: Observed marginal frequencies of `VehAge` and `BonusMalus` similar to Figure 1, but split to different cohorts: (top left) `VehGas`, (top right) `VehBrand`, (bottom left) `VehPower` classes 4-9 as in Model GLM1, (bottom right) `DrivAge` classes 1-7 as in Model GLM1.

terms of predictive performance, however, run times are non-competitive and the 405 regression parameters involved require much more exploration.

Summarizing, in our case we prefer Model GLM4 over the network regression model and Model GLM5 because of more transparency, more interpretability and better run times.

6 Conclusions and outlook

This manuscript has described the connection between GLMs and neural network regression models. We have discussed how both modeling approaches can benefit from each other in guaranteeing the balance property (for the right price level) and in identifying (missing) regression

structure. The latter has allowed us to interpret data and model structure which is a crucial point in explaining insurance products to management and customers. Going one step further, we have introduced a third paradigm regarding the two modeling cultures mentioned by Breiman [5]. Breiman's 1st culture is the *Data Modeling Culture* that defines a model which is *fitted (estimated, calibrated)* to the data. Breiman's 2nd culture is the *Algorithmic Modeling Culture* which *learns* from data to optimally predict unseen data. Our paradigm is to use the algorithmic culture to *Learn the Data Model*. This paradigm is not completely new, already Karl Pearson and Sir Ronald Aylmer Fisher have been inspired by similar ideas, and we interpret the work of Zhao–Hastie [56] in a similar way.

We conclude by mentioning a few open points that we have not been considering. Probably the most delicate one is the non-uniqueness of sufficiently good networks. In fact, we have received infinitely many sufficiently good networks (according to our objective function) which may still result in rather different model structures on a micro level. Usually, it is proposed to average over such models, however, in doing so one loses more interpretability as well as blended models are more difficult to manage. Another open point is claim size modeling. In our example, we have focused on claims frequency modeling within the Poisson distribution. This is by far more simple than modeling claims sizes with multi-parameter models. In particular, one can typically not rely on a single two parametric distribution for appropriately modeling the whole range of potential claim sizes.
Finally, we did not enter any legal discussion about differentiation and discrimination. Regulation may define some discriminatory features w.r.t. which price sensitivity is forbidden. It should be discussed how such discriminatory features can be flattened in pricing. For instance, if gender is declared to be a discriminatory feature that it is not allowed to be used in pricing, how can we guarantee that resulting prices do not discriminate w.r.t. gender, also not implicitly through learning the gender from other feature information. For an extended discussion we refer to Lindholm et al. [32].

Acknowledgment. I would like to kindly thank the following people who have carefully read previous versions of this manuscript: Daniel Balint (ETH Zurich), Andrea Gabrielli (ETH Zurich), Simon Rentzmann (AXA-Winterthur), Ronald Richman (QED Actuaries and Consultants).

References

[1] Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control* **19/6**, 716-723.

[2] Bailey, R.A., Simon, L.J. (1960). Two studies on automobile insurance ratemaking. *ASTIN Bulletin* **1**, 192-217.

[3] Barndorff-Nielsen, O. (2014). *Information and Exponential Families: In Statistical Theory*. John Wiley & Sons.

[4] Bengio, Y., Schwenk, H., Senécal, J.-S., Morin, F., Gauvain, J.-L. (2006). Neural probabilistic language models. In: *Innovations in Machine Learning*. Studies in Fuzziness and Soft Computing, Vol. 194. Springer, 137-186.

[5] Breiman, L. (2001). Statistical modeling: the two cultures. *Statistical Science* **16/3**, 199-215.

[6] Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). *Classification and Regression Trees.* Wadsworth Statistics/Probability Series.

[7] CASdatasets Package Vignette (2016). Reference Manual, May 28, 2016. Version 1.0-6. Available from `http://cas.uqam.ca`.

[8] Charpentier, A. (2015). *Computational Actuarial Science with R.* CRC Press.

[9] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems* **2**, 303-314.

[10] Denuit, M., Hainaut, D., Trufin, J. (2019). *Effective Statistical Learning Methods for Actuaries I: GLMs and Extensions.* Springer Actuarial Lecture Notes.

[11] Dobson, A.J. (2002). *An Introduction to Generalized Linear Models.* 2nd edition. Chapman & Hall, CRC.

[12] Draxler, F., Veschgini, K., Salmhofer, M., Hamprecht, F.A. (2019). Essentially no barriers in neural network energy landscape. *arXiv*:1803.00885v5.

[13] Efron, B., Hastie, T. (2016). *Computer Age Statistical Inference.* Cambridge University Press.

[14] Ferrario, A., Noll, A., Wüthrich, M.V. (2018). Insights from inside neural networks. *SSRN Manuscript* ID 3226852. Version November 14, 2018.

[15] Frees, E.W. (2010). *Regression Modeling with Actuarial and Financial Applications.* Cambridge University Press.

[16] Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation* **121/2**, 256-285.

[17] Friedman, J.H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics* **29/5**, 1189-1232.

[18] Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D., Wilson, A.G. (2018). Loss surfaces, mode connectivity, and fast ensembling of DNNs. *arXiv*:1802.10026v4.

[19] Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning.* MIT Press.

[20] Grohs, P., Perekrestenko,D., Elbrächter, D., Bölcskei, H. (2019). Deep neural network approximation theory. Submitted to *IEEE Transactions on Information Theory* (invited paper).

[21] Hastie, T., Tibshirani, R. (1986). Generalized additive models (with discussion). *Statistical Science* **1**, 297–318.

[22] Hastie, T., Tibshirani, R. (1990). *Generalized Linear Models.* Chapman & Hall.

[23] Hastie, T., Tibshirani, R., Friedman, J. (2009). *The Elements of Statistical Learning. Data Mining, Inference, and Prediction.* 2nd edition. Springer Series in Statistics.

[24] Hastie, T., Tibshirani, R., Wainwright, M. (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations.* CRC Press.

[25] Hoaglin, D.C. (2003). John W. Tukey and data analysis. *Statistical Science* **18/3**, 311-318.

[26] Hornik, K., Stinchcombe, M., White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks* **2**, 359-366.

[27] Jørgensen, B. (1986). Some properties of exponential dispersion models. *Scandinavian Journal of Statistics* **13/3**, 187-197.

[28] Jørgensen, B. (1987). Exponential dispersion models. *Journal of the Royal Statistical Society. Series B (Methodological)* **49/2**, 127-145.

[29] Ke, G., Xu, Z., Zhang, J., Bian, J., Liu, T.-Y. (2019). DeepGBM: a deep learning framework distilled by GBDT for online prediction tasks. *Proceeding KDD2019*, Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 384-394.

[30] Lehmann, E.L. (1983). *Theory of Point Estimation*. Wiley.

[31] Leshno, M., Lin, V.Y., Pinkus, A., Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks* **6/6**, 861-867.

[32] Lindholm, M., Richman, R., Tsanakas, A., Wüthrich, M.V. (2020). Discrimination-free insurance pricing. *SSRN Manuscript* ID 3520676.

[33] McCullagh, P., Nelder, J.A. (1983). *Generalized Linear Models*. Chapman & Hall.

[34] Montúfar, G., Pascanu, R., Cho, K., Bengio, Y. (2014). On the number of linear regions of deep neural networks. *Neural Information Processing Systems Proceedings*$^{\beta}$ **27**, 2924-2932.

[35] Nelder, J.A., Wedderburn, R.W.M. (1972). Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)* **135/3**, 370-384.

[36] Noll, A., Salzmann, R., Wüthrich, M.V. (2018). Case study: French motor third-party liability claims. *SSRN Manuscript* ID 3164764. Version November 8, 2018.

[37] Ohlsson, E., Johansson, B. (2010). *Non-Life Insurance Pricing with Generalized Linear Models*. Springer.

[38] Rentzmann, S., Wüthrich, M.V. (2019). Unsupervised learning: what is a sports car? *SSRN Manuscript* ID 3439358.

[39] Richman, R. (2018). AI in actuarial science. *SSRN Manuscript* ID 3218082.

[40] Rumelhart, D.E., Hinton, G.E., Williams, R.J. (1986). Learning representations by back-propagating errors. *Nature* **323/6088**, 533-536.

[41] Sahlgren, M. (2015). A brief history of word embeddings. `https://www.linkedin.com/pulse/brief-history-word-embeddings-some-clarifications-magnus-sahlgren/`

[42] Schapire, R.E. (1990). The strength of weak learnability. *Machine Learning* **5/2**, 197-227.

[43] Schelldorfer, J., Wüthrich, M.V. (2019). Nesting classical actuarial models into neural networks. *SSRN Manuscript* ID 3320525.

[44] Srivastava, N., Hinton, G., Krizhevsky, A. Sutskever, I., Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning* **15**, 1929-1958.

[45] Tibshirani, R. (1996). Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society. Series B (Methodological)* **58/1**, 267-288.

[46] Tikhonov, A.N. (1943). On the stability of inverse problems. *Doklady Akademii Nauk SSSR* **39/5**, 195-198.

[47] Tukey, J.W. (1977). *Explanatory Data Analysis*. Addison-Wesley.

[48] Verstraeten, D., Schrauwen, B., D'Haene, M., Stroobrandt, D. (2007). An experimental unification of reservoir computing methods. *Neural Networks* **20**, 391-403.

[49] Wager, S., Wang, S., Liang, P.S. (2013). Dropout training as adaptive regularization. In: *Advances in Neural Information Processing Systems* **26**, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Weinberger (eds.), Curran Associates, Inc, 351-359.

[50] Wood, S.N. (2017). *Generalized Additive Models: an Introduction with* R. 2nd edition. CRC Press.

[51] Wüthrich, M.V. (2013). Non-Life Insurance: Mathematics & Statistics. *SSRN Manuscript* ID 2319328. Version March 20, 2019.

[52] Wüthrich, M.V. (2019). Bias regularization in neural network models for general insurance pricing. To appear in *European Actuarial Journal*.

[53] Wüthrich, M.V., Buser, C. (2016). Data analytics for non-life insurance pricing. *SSRN Manuscript* ID 2870308, Version of June 4, 2019.

[54] Wüthrich, M.V., Merz, M. (2019). Editorial: Yes, we CANN! *ASTIN Bulletin* **49/1**.

[55] Zaslavsky, T. (1975). Facing up to arrangements: face-count formulas for partitions of space by hyperplanes. *Memoirs of the American Mathematical Society* **154**.

[56] Zhao, Q., Hastie, T. (2019). Causal interpretations of black-box models. To appear in *Journal of Business & Economic Statistics*.

[57] Zou, H., Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B* **67/2**, 301-320.