Astin Bulletin (2023), **53**, pp. 213–232
doi:10.1017/asb.2023.15

RESEARCH ARTICLE

The use of autoencoders for training neural networks with mixed categorical and numerical features

Łukasz Delong[1] and Anna Kozak[2]

[1]SGH Warsaw School of Economics, Institute of Econometrics, Warsaw, Poland and [2]Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland
Corresponding author: Łukasz Delong; Email: lukasz.delong@sgh.waw.pl

Received: 21 December 2022; **Revised:** 15 March 2023; **Accepted:** 20 March 2023; **First published online:** 24 April 2023

Keywords: Autoencoders; representation learning; categorical and numerical features; embeddings; initialization of neural networks

Abstract
We focus on modelling categorical features and improving predictive power of neural networks with mixed categorical and numerical features in supervised learning tasks. The goal of this paper is to challenge the current dominant approach in actuarial data science with a new architecture of a neural network and a new training algorithm. The key proposal is to use a joint embedding for all categorical features, instead of separate entity embeddings, to determine the numerical representation of the categorical features which is fed, together with all other numerical features, into hidden layers of a neural network with a target response. In addition, we postulate that we should initialize the numerical representation of the categorical features and other parameters of the hidden layers of the neural network with parameters trained with (denoising) autoencoders in unsupervised learning tasks, instead of using random initialization of parameters. Since autoencoders for categorical data play an important role in this research, they are investigated in more depth in the paper. We illustrate our ideas with experiments on a real data set with claim numbers, and we demonstrate that we can achieve a higher predictive power of the network.

1. Introduction
In this paper, we deal with the following three important problems of training neural networks with mixed categorical and numerical features in supervised learning tasks:

- How to construct a numerical representation of categorical features which is fed, together with the numerical features, into the hidden layers of a neural network,
- Which architecture of a neural network we should build, in particular, how we should treat (concatenate) features of different types (in our case categorical and numerical features),
- How we should initialize the weights and the bias terms of a neural network to guide the network towards a point in the surface of parameters where the network has a high predictive power and good generalization properties.

There are many possible approaches to these problems. We present an approach inspired by autoencoders.

Neural networks have recently gained a lot of attention in actuarial science. In particular, Noll *et al*. (2019) and Ferrario *et al*. (2020) were among the first to discuss applications of neural networks to actuarial non-life insurance pricing problems and compared neural networks with generalized linear models. The current approach to supervised learning tasks in actuarial data science is to build a neural network where *entity embeddings* for categorical features are used. Entity embeddings were introduced by Guo and Berkhahn (2016) in the machine learning literature to help neural networks to deal with sparse categorical data of high dimension. They were first promoted by Richman (2021), Noll *et al*. (2019) and Ferrario *et al*. (2020) in actuarial data science and they were further developed by Blier-Wong *et al*. (2021), Kuo and Richman (2021) and Shi and shi (2022). An entity embedding, as part of a neural network for a supervised learning task, is learned separately for each categorical feature and allows us to derive a real-value representation of a categorical feature in a low-dimensional space. The numerical representations of the categorical features are concatenated with the numerical features and they are fed together as input into hidden layers of a neural network. The weights of the entity embeddings are learned together with all other parameters of the neural network, and the objective is to minimize a loss appropriate for a target response. All weights and bias terms of the network are initialized with random values from uniform distributions, and the back propagation algorithm is used to learn the parameters of the network. To the best of our knowledge, the architecture of a neural network described above is the only architecture of a neural network investigated to date for actuarial applications. In particular, other numerical representations of categorical features have not been tested as inputs to neural networks. In addition, advances in training algorithms for neural networks have not been discussed in actuarial data science. The only distinct training algorithm was proposed by Merz and Wüthrich (2019) and Schelldorfer and Wüthrich (2019) who promoted the *Combined Actuarial Neural Network* (CANN). The goal of this paper is to challenge the current dominant approach to supervised learning tasks in actuarial data science with a new architecture of a neural network, in particular, with a new numerical representation of categorical features and with a new training algorithm.

It is known that in order to achieve a high predictive power of a neural network, the input to the network should contain the most important information for the supervised learning task under consideration. A highly informative input can be effectively pre-processed in hidden layers of the network to provide good predictions of the response. It has been shown in the machine learning literature that non-linear *autoencoders*, in particular, *denoising autoencoders*, built with neural networks, can capture the main factors of variation in the input and detect key characteristics of the multivariate and high-dimensional distribution of the input. As a result, representations of features derived using (denoising) autoencoders, learned in unsupervised learning tasks, can improve the predictive power of regression models if these representations are used as inputs to neural networks to predict the response, see for example Vincent *et al*. (2008) and (2010).

Autoencoders without noise for numerical data have been investigated in actuarial data science. The benefits of autoencoders without noise in supervised and unsupervised learning tasks have been demonstrated by Gao and Wüthrich (2018), Hainaut (2018), Rentzmann and Wüthrich (2019), Blier-Wong *et al*. (2021, 2022), Miyata and Matsuyama (2022) and Grari *et al*. (2022). To the best of our knowledge, autoencoders for categorical features are less common in actuarial data science. The only exception we are aware of is the paper by Lee *et al*. (2019) in which the authors discuss how to build word embeddings, which are similar to but are not exactly autoencoders for categorical data in the meaning investigated in this paper.

The first contribution of this paper is that we investigate different types of autoencoders for categorical data. We demonstrate that we can benefit from non-linear autoencoders built with neural networks when the purpose is to derive informative representations of categorical features. We show that an autoencoder for categorical features should be of a different type than an autoencoder for numerical features. Most importantly, we deduce that the best autoencoder for categorical features, which extracts the most important information from the vector of categorical features, implies a different numerical representation of categorical features than the representation from entity embeddings currently used in supervised learning tasks in actuarial data science. From our experiments, we conclude that we should learn one numerical representation for all categorical features, rather than multiple representations for each separate feature, to build a more robust and informative representation of the categorical data. The other, and our main contribution, is that we use a joint numerical representation of categorical features, together with all other numerical features, as the input to the hidden layers of a neural network trained to predict a target response. In other words, we introduce a new architecture of a neural network with mixed categorical and numerical features for the supervised learning tasks. The change in the architecture compared to the current approach is that all one-hot encoded categorical features are transformed with one embedding into a real-value representation in a low-dimensional space and the joint representation of the categorical features is then concatenated with the numerical features. Finally, we fine-tune the numerical representation of the categorical features from a proper autoencoder by training its weights together with all other parameters of the neural network. Hence, the autoencoder for the categorical data is only used to derive an initial representation of the categorical features. This approach is known in the machine learning literature as *pre-training of layers with autoencodeders*, see Vincent *et al*. (2008), Erhan *et al*. (2010, 2009) and Vincent *et al*. (2010). Hence, we pre-train the joint embedding for categorical features in a neural network for a supervised learning task with our autoencoder for categorical data. From Erhan *et al*. (2009) and (2010), we know that pre-training other layers of the neural network is also beneficial, and this can be achieved with an autoencoder for numerical data. We use both autoencoders, without noise and denosing autoencoders, in this research.

The benefits of using (denosing) autoencoders for pre-training layers of neural networks have been demonstrated in the literature (see the papers referred to above). In comparison to these papers:

- We perform our experiments with categorical data, instead of binary data, which means that we use a different autoencoder and a different type of a corruption process for the denoising autoencoder,
- We perform our experiments with Poisson distributed data and the Poisson deviance loss, which is the most common loss function in actuarial data science, instead of the mean square loss and the cross-entropy loss commonly used in the machine learning literature,
- We propose and validate a new architecture of a neural networks, with a joint embedding for all categorical features, for the supervised learning tasks. This has not been considered to date in actuarial data science,
- We propose to scale appropriately the representation of the categorical features from an autoencoder for categorical data before an autoencoder for numerical data is built to pre-train the first hidden layer of a neural network. This improves significantly the approach to supervised learning tasks with our new architecture,
- We compare various initialization techniques and we show that pre-training of layers of a neural network with non-linear and over-complete/denoising autoencoders produces much better results than applications of classical linear and under-complete autoencoders without noise (MCA, PCA),
- We investigate the balance property, the bias and the stability of the predictions which are crucial for actuarial pricing.

The main conclusion of this paper is that we can improve the current approach to modelling categorical features in supervised learning tasks, which uses separate entity embeddings, and the training algorithm, which randomly initializes the parameters of the neural network. The proposal is to change the architecture of the network by using a different numerical representation of the categorical features, learned with a joint embedding, and initialize the layers of the network, in particular, the joint embedding and the first hidden layer, with representations learned with (denosing) autoencoders in unsupervised learning tasks.

This paper is structured as follows. In Section 2, we present the general setup for neural networks and our numerical experiments. In Section 3, we discuss autoencoders for categorical and numerical features. In Section 4, we focus on training neural networks with mixed categorical and numerical features. Details of our experiments and some additional results are presented in the Online Supplement. The R codes for training our categorical autoencoders are available on https://github.com/LukaszDelong/Autoencoders.

2. General setup

We assume that we have a data set consisting of $(y_i, \boldsymbol{x}_i)_{i=1}^n$ where y_i describes the one-dimensional response for observation i and $\boldsymbol{x}_i = (x_{1,i}, .., x_{j,i}, ..., x_{d,i})'$ is a d-dimensional vector of features which characterizes the observation. We may omit the index i, which indicates the observation, and simply use $(y, \boldsymbol{x})$. The vector $\boldsymbol{x}$ consists of mixed categorical and numerical features. We assume that we have c categorical features and $d - c$ numerical features.

The categorical features are first one-hot encoded. Let x_j denote a categorical feature with m_j different labels $\left\{a_1^j, ..., a_{m_j}^j\right\}$. This categorical feature is transformed into a m_j-dimensional vector of zeros and one:

$$x_j \mapsto \boldsymbol{x}_j^{cat} = \left(x_{j_1}, ..., x_{j_{m_j}}\right)' = \left(\mathbf{1}\{x_j = a_1^j\}, ..., \mathbf{1}\{x_j = a_{m_j}^j\}\right)' \in \mathbb{R}^{m_j}.$$

The dimension of the vector of features $\boldsymbol{x} = \left((\boldsymbol{x}^{cat})', (\boldsymbol{x}^{num})'\right)' = \left((\boldsymbol{x}_1^{cat})', ..., (\boldsymbol{x}_c^{cat})', x_{c+1}, ..., x_d\right)'$ becomes $\sum_{j=1}^c m_j + d - c$. As far as the numerical features are concerned, we assume that each numerical feature takes its values from $[-1, 1]$, that is min–max scaler transformations are applied to the numerical features on the original scale.

In general, we use neural networks with $M \in \mathbb{N}$ hidden layers and $q_m \in \mathbb{N}$ neurons in each hidden layer $m = 1, \ldots, M$. The network layers are defined with the mappings:

$$z \in \mathbb{R}^{q_{m-1}} \mapsto \theta^m(z) = \left(\theta_1^m(z), \ldots, \theta_{q_m}^m(z)\right)' \in \mathbb{R}^{q_m}, \quad m = 1, \ldots, M, \tag{2.1}$$

$$z \in \mathbb{R}^{q_{m-1}} \mapsto \theta_r^m(z) = \chi^m\left(b_r^m + \langle \boldsymbol{w}_r^m, z\rangle\right), \quad r = 1, \ldots, q_m, \tag{2.2}$$

where $\chi^m : \mathbb{R} \to \mathbb{R}$ denotes an activation function, $\boldsymbol{w}_r^m \in \mathbb{R}^{q_{m-1}}$ denotes the network weights, $b_r^m \in \mathbb{R}$ denotes the bias term, and $\langle\cdot, \cdot\rangle$ denotes the scalar product in $\mathbb{R}^{q_{m-1}}$. By q_0 we denote the dimension of the input vector to the network. The mapping:

$$z \in \mathbb{R}^{q_0} \mapsto \Theta^{M+1}(z) = \left(\Theta_1^{M+1}(z), \ldots, \Theta_{q_{M+1}}^{M+1}(z)\right)' \in \mathbb{R}^{q_{M+1}}, \tag{2.3}$$

with a composition of the network layers $\theta^1, \ldots, \theta^M$, and the components:

$$z \mapsto \Theta_r^{M+1}(z) = b_r^{M+1} + \left\langle \boldsymbol{w}_r^{M+1}, \left(\theta^M \circ \cdots \circ \theta^1\right)(z)\right\rangle, \quad r = 1, \ldots, q_{M+1},$$

gives us the prediction from the network in the output layer $M + 1$ of dimension q_{M+1} based on the input vector z. The output (2.3) returns the prediction with the linear activation function, and this prediction can be transformed with appropriate non-trainable and non-linear mapping if this is required for an application. If we set $M = 0$ in (2.1)–(2.2), then we assume that the input vector is just linearly transformed to give the prediction in the output layer of dimension q_1 and, in this case, the components in (2.3) are given by

$$z \mapsto \Theta_r^1(z) = b_r^1 + \left\langle \boldsymbol{w}_r^1, z\right\rangle, \quad r = 1, \ldots, q_1.$$

In our numerical experiments, we use the data set freMTPL2freq, which is included in the R package CASdatasets. The data set has 678,013 observations from insurance policies. The response Y describes the number of claims per policy. Each policy has nine features and an exposure: $(\boldsymbol{x}, Exp)$. This data set is extensively studied by Noll *et al*. (2019), Schelldorfer and Wüthrich (2019) and Ferrario *et al*. (2020) in the context of applications of generalized linear models and neural networks to modelling the number of claims. We perform the same data cleaning and feature pre-processing as in these papers. For the purpose of our experiments, we work with the features presented in Table 1.

We consider a supervised learning task where the goal is to predict the number of claims for a policyholder characterized by $(\boldsymbol{x}, Exp)$ by estimating the regression function $\mathbb{E}[Y|\boldsymbol{x}, Exp]$. The prediction is constructed with the neural network described above and the one-dimensional output from the network (2.3) is transformed with the non-trainable and the non-linear exponential transformation:

$$\mathbb{E}[Y|\boldsymbol{x}, Exp] = e^{\log(Exp) + \Theta_1^{M+1}(\boldsymbol{x})}. \tag{2.4}$$

Table 1. *Features used in our experiments.*

6 categorical features	2 numerical features	1 binary feature
Area — 6 levels	BonusMalus (caped at 150)	VehGas
VehPower — 6 levels	log-Density	
VehAge — 3 levels		
DrivAge — 7 levels		
VehBrand — 11 levels		
Region — 22 levels		

The parameters of the network are trained by minimizing the Poisson deviance loss function, see for example Noll *et al*. (2019), Schelldorfer and Wüthrich (2019) and Ferrario *et al*. (2020). Our supervised learning task is solved with the help of unsupervised learning tasks where autoencoders are used.

3. Autoencoders

Let $\boldsymbol{x}$ denote a vector of (categorical, numerical, mixed) features of dimension p. An *autoencoder* consists of two functions:

$$\varphi : \mathbb{R}^p \mapsto \mathbb{R}^l, \quad \text{and} \quad \psi : \mathbb{R}^l \mapsto \mathbb{R}^p.$$

The mapping φ is called the *encoder*, and ψ is called the *decoder*. The mapping $\boldsymbol{x} \mapsto \varphi(\boldsymbol{x})$ from the encoder gives an l-dimensional representation of the p-dimensional vector $\boldsymbol{x}$. The mapping $\boldsymbol{z} \mapsto \psi(\boldsymbol{z})$ from the decoder tries to reconstruct the p-dimensional vector $\boldsymbol{x}$ from its l-dimensional representation $\boldsymbol{z} = \varphi(\boldsymbol{x})$. We define the reconstruction function as

$$\pi = \psi \circ \varphi : \mathbb{R}^p \mapsto \mathbb{R}^p.$$

For a data set with observations $(\boldsymbol{x}_i)_{i=1}^n$, the goal is to find the functions φ and ψ such that the reconstruction error as

$$\frac{1}{n}\sum_{i=1}^{n} L(\pi(\boldsymbol{x}_i), \boldsymbol{x}_i),$$

measured with a loss function L, is minimized. If we can find an autoencoder for which the reconstruction error is small, then we can claim that the encoder extracts the most important information from a multi-dimensional vector of features. Consequently, we can use the representation $\varphi(\boldsymbol{x})$, instead of $\boldsymbol{x}$, as input to predict the response in our supervised learning task. The observed response y is not used in this approach when we train an autoencoder. We train autoencoders in a fully unsupervised fashion, but we will improve the representation based on the target y when we solve the supervised learning task.

Linear autoencoders are well-known in statistics. By a linear autoencoder, we mean an autoencoder where both the functions φ and ψ are linear. Classical examples of linear autoencoders include autoencoders built with *Principal Component Analysis* for numerical data and *Multiple Correspondence Analysis* for categorical data. We refer for example to Chapter 6.2 in Dixon *et al*. (2020) for the equivalence between the linear autoencoder built by minimizing the mean square reconstruction loss function and the representation built with the PCA algorithm. For MCA and its relation to PCA, we refer for example to Pagès (2015) and Chavent *et al*. (2017).

In this paper, we are interested in non-linear autoencoders where at least one of the functions φ or ψ is non-linear. We use the notation (2.1)–(2.2) from the previous section. To build an autoencoder for the input $\boldsymbol{x}$, we use a neural network with one hidden layer, that is $M = 1$. The dimension of the single hidden layer is set to $q_1 = l$, and the dimensions of the input and the output are set to $q_0 = q_2 = p$. The activation function for the hidden layer depends on the type of data and is discussed in the sequel. The vector $(\theta_1^1(\boldsymbol{x}), ..., \theta_l^1(\boldsymbol{x}))'$ gives us the representation of the input $\boldsymbol{x}$ from the encoder. The vector

$(\Theta_1^2(\boldsymbol{x}), ..., \Theta_p^2(\boldsymbol{x}))'$, transformed with a non-trainable and non-linear function if required for the data, gives us the reconstruction of the input $\boldsymbol{x}$ predicted with the decoder. This means that ψ also includes a non-trainable and non-linear transformation of the output (2.3) from the network if such a transformation is required for application. Clearly, we could also build deep autoencoders with more hidden layers, but shallow autoencoders with one hidden layer are sufficient for our main application in Section 4.

If $l < p$, we construct *under-complete autoencoders* and we reduce the dimension of the input $\boldsymbol{x}$. Linear autoencoders built with the PCA and MCA algorithms are examples of under-complete autoencoders. If we choose $l = p$, then we can achieve a zero reconstruction error by learning the identity mapping. Interestingly, we can also learn *over-complete autoencoders* with $l > p$, and *denoising autoencoders* are examples of over-complete autoencoders. In order to construct a denoising autoencoder with $l > p$, we corrupt the input for the network. The objective for training a denoising autoencoder is to find the functions φ and ψ such that the reconstruction error:

$$\frac{1}{n}\sum_{i=1}^{n} L(\pi(\tilde{\boldsymbol{x}}_i), \boldsymbol{x}_i),$$

measured with a loss function L, is minimized. This time, the input $\tilde{\boldsymbol{x}}$ is corrupted input $\boldsymbol{x}$ which is constructed by *adding* a noise to $\boldsymbol{x}$. It has been demonstrated in the machine-learning literature that denosing autoencoders are very good at extracting the most important information from a multi-dimensional vector of features, see for example Vincent *et al*. (2008) and (2010). We can also construct over-complete autoencoders using data without noise if a low number of epochs is used for training the autoencoder built with a neural network.

In the next two sections, we discuss autoencoders for numerical and categorical features.

3.1. *Autoencoders for numerical features*

As discussed in Introduction, autoencoders without noise for numerical features have been investigated in various actuarial applications. In this paper, we adopt the approach from Rentzmann and Wüthrich (2019). We use the hyperbolic tangent activation function in the hidden layer (χ^1), reconstruct the input using the linear prediction:

$$\boldsymbol{x} \in \mathbb{R}^p \mapsto \hat{\boldsymbol{x}} = \pi(\boldsymbol{x}) = \left(\Theta_1^2(\boldsymbol{x}), ..., \Theta_p^2(\boldsymbol{x})\right)' \in \mathbb{R}^p, \tag{3.1}$$

and use the mean square error loss function L to measure the reconstruction error between the prediction $\hat{\boldsymbol{x}} = \pi(\boldsymbol{x})$ and the input $\boldsymbol{x}$. We build a non-linear autoencoder since we use a non-linear activation function (the hyperbolic tangent) in the hidden layer. In contrast to Rentzmann and Wüthrich (2019), we allow for bias terms in the network since we use the min–max scaler transformation of the numerical features instead of zero mean and unit variance standardization. An example of the architecture of a neural network used in this paper to build an autoencoder for numerical features is presented in Figure 1.

As far as denoising autoencoders are concerned, we apply two types of corruption processes to distort the input, see for example Vincent *et al*. (2008), (2010):

- Gaussian disturbance (*gaussian*): For each observation, $i = 1, ..., n$, and each numerical feature in vector $\boldsymbol{x}_i$, the original input is corrupted with the transformation $x_{j,i} \mapsto \tilde{x}_{j,i} \sim N(x_{j,i}, \sigma^2)$.
- Masking to zero (*zero*): For each observation, $i = 1, ..., n$, and the fraction ν of numerical features in vector $\boldsymbol{x}_i$ chosen at random, the original input is corrupted with the transformation $x_{j,i} \mapsto \tilde{x}_{j,i} = 0$.

3.2. *Autoencoders for categorical features*

We consider two types of architecture of autoencoders for categorical features. Neither has been explored in the actuarial literature, although they appear, and versions of them appear, in many applications of machine learning methods in various fields.

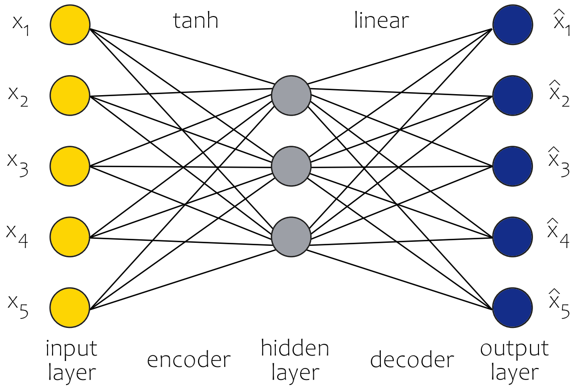

Figure 1. *Architecture of the autoencoder for numerical features used in the paper.*

1. Separate autencoders for each feature (*Separate AEs*): For categorical feature x_j with m_j different labels and its one-hot representation $\boldsymbol{x}_j^{cat} = (x_{j_1}, ..., x_{j_{m_j}})'$, we build a neural network (2.1)–(2.2) with $M = 1, q_0 = m_j, q_1 = l_j, q_2 = m_j$, where l_j is the required dimension of the representation of the categorical feature. Since we use the one-hot representation of x_j as the input to the network, there is no need to train bias terms in the hidden layer, so we set $b_r^1 = 0$ for $r = 1, ..., l_j$. However, it is still beneficial to train bias terms in the output layer in order to match the output expressed with probabilities (see below). The linear activation function for χ^1 in the hidden layer is a natural choice here since the linear mappings $\langle \boldsymbol{w}_r^1, \boldsymbol{x}_j^{cat} \rangle$, for neurons $r = 1, ..., l_j$, yield unique constants for each label of the categorical feature, so there is no need to apply non-linear transformations to these constants. We reconstruct the input using the prediction:
$$\boldsymbol{x}_j^{cat} \in \mathbb{R}^{m_j} \mapsto \hat{\boldsymbol{x}}_j^{cat} = \pi(\boldsymbol{x}_j^{cat}) = \left(\pi_1(\boldsymbol{x}_j^{cat}), ..., \pi_{m_j}(\boldsymbol{x}_j^{cat})\right)' \in \mathbb{R}^{m_j}, \tag{3.2}$$
where
$$\pi_r(\boldsymbol{x}_j^{cat}) = \frac{e^{\Theta_r^2(\boldsymbol{x}_j^{cat})}}{\sum_{u=1}^{m_j} e^{\Theta_u^2(\boldsymbol{x}_j^{cat})}}, \qquad r = 1, ..., m_j.$$
The soft-max activation function is applied to the output from the network (2.3) to derive the reconstructed input. The reconstruction function returns the probabilities that the reconstructed feature takes a particular label. The label with the highest predicted probability is the label predicted for the reconstructed feature. Since we now deal with a classification problem for the single categorical feature x_j, it is natural to use the cross entropy loss function L to measure the reconstruction error between the prediction $\hat{\boldsymbol{x}} = \pi(\boldsymbol{x})$ and the input $\boldsymbol{x}$:
$$L(\pi(\boldsymbol{x}_{j,i}^{cat}), \boldsymbol{x}_{j,i}^{cat}) = -\sum_{r=1}^{m_j} x_{j_r,i}^{cat} \log\left(\pi_r(\boldsymbol{x}_{j,i}^{cat})\right), \qquad i = 1, ..., n. \tag{3.3}$$
We build a non-linear autoencoder since we use a non-linear activation function (the soft-max function) in the output layer. The approach described above is applied to all categorical features in the data set. An example of the architecture of a neural network used in this paper to build the autoencoder of type *Separate AEs* for categorical features (with 2 and 3 labels) is presented in Figure 2.
2. Joint autoencoder all features (*Joint AE*): We consider a vector of categorical features $(x_1, ..., x_c)$ with $(m_1, ..., m_c)$ different labels and their one-hot representations $\boldsymbol{x}^{cat} = \left((\boldsymbol{x}_1^{cat})', ..., (\boldsymbol{x}_c^{cat})'\right)'$. Let $\bar{m}_0 = 0$ and $\bar{m}_j = \sum_{u=1}^{j} m_u$ for $j = 1, ..., c$. This time we build a neural network (2.1)–(2.2) with $M = 1, q_0 = \bar{m}_c, q_1 = l, q_2 = \bar{m}_c$, where l is the dimension of the required joint representation of all categorical features. We still set $b_r^1 = 0$ for $r = 1, ..., l$, train bias terms in the output layer

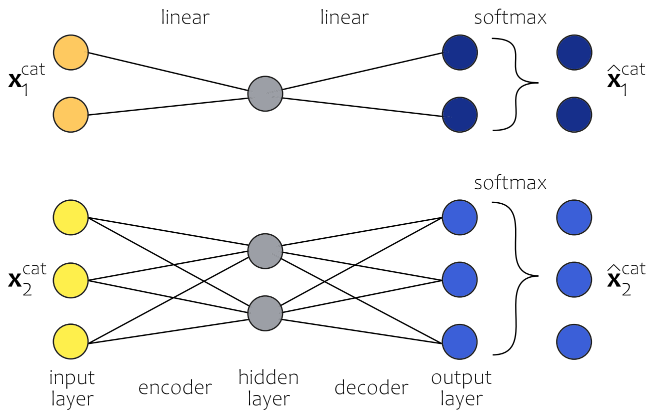

Figure 2. *Architecture of the autoencoder of type* Separate AEs *for categorical features.*

and apply the linear activation function in χ^1. We reconstruct the input using the prediction:

$$\begin{aligned}\boldsymbol{x}^{cat} \in \mathbb{R}^{\bar{m}_c} \mapsto \hat{\boldsymbol{x}}^{cat} = \pi(\boldsymbol{x}^{cat}) = \big(&\pi_1(\boldsymbol{x}^{cat}), ..., \pi_{\bar{m}_1}(\boldsymbol{x}^{cat}), ...,\\ &\pi_{\bar{m}_{j-1}+1}(\boldsymbol{x}^{cat}), ..., \pi_{\bar{m}_j}(\boldsymbol{x}^{cat}), ...,\\ &\pi_{\bar{m}_{c-1}+1}(\boldsymbol{x}^{cat}), ..., \pi_{\bar{m}_c}(\boldsymbol{x}^{cat})\big)' \in \mathbb{R}^{\bar{m}_c},\end{aligned} \tag{3.4}$$

where

$$\pi_r(\boldsymbol{x}^{cat}) = \frac{e^{\Theta_r^2(\boldsymbol{x}^{cat})}}{\sum_{u=\bar{m}_{j-1}+1}^{\bar{m}_j} e^{\Theta_u^2(\boldsymbol{x}^{cat})}}, \quad r = \bar{m}_{j-1}+1, ..., \bar{m}_j, \quad j = 1, ..., c,$$

and $\pi_r(\boldsymbol{x}^{cat})$, for $r = \bar{m}_{j-1}+1, ..., \bar{m}_j$, return probabilities that the categorical feature x_j takes a particular label among its m_j labels. The prediction of the label for x_j is the label with the highest predicted probability among $\pi_r(\boldsymbol{x}^{cat})$. Clearly, we build a non-linear autoencoder. We remark that the soft-max activations functions are now applied to groups of neurons in the output layer from the network (2.3) which correspond to the labels of the categorical features. Hence, the decoder here returns probabilities in classification problems for all categorical features. This time all neurons in the layers of the autoencoder (before the soft-max transformations are applied) share the parameters of one neural network. By applying the *Separate AEs*, we independently solve multiple classification problems for our categorical features with separate autoencoders, whereas by applying the *Joint AE*, we jointly solve multiple classification problems for our categorical features with one autoencoder. Such an approach is called *multi-task learning* in machine learning, see for example Caruana (1997) and Ruder (2017). We use the cross entropy loss function L to measure the reconstruction error between prediction $\hat{\boldsymbol{x}} = \pi(\boldsymbol{x})$ and input $\boldsymbol{x}$:

$$L(\pi(\boldsymbol{x}_i^{cat}), \boldsymbol{x}_i^{cat}) = -\sum_{j=1}^{c}\sum_{r=1}^{m_j} x_{jr,i}^{cat}\log\big(\pi_r(\boldsymbol{x}_i^{cat})\big), \quad i = 1, ..., n. \tag{3.5}$$

An example of the architecture of type *Joint AE* is presented in Figure 3.

In order to build denosing autoencoders, we apply the following corruption processes for categorical features:

- For each observation, $i = 1, ..., n$, and a fraction v of categorical features in vector $\boldsymbol{x}_i$ chosen at random, the original input is corrupted with the transformations:

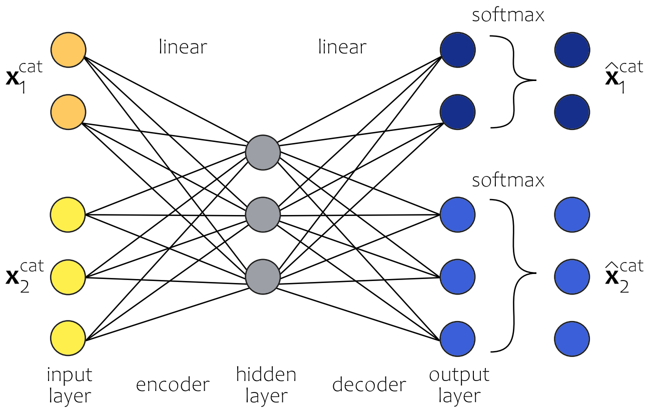

Figure 3. *Architecture of the autoencoder of type* Joint AE *for categorical features.*

- Sampling a new label (*sample*): The original input is corrupted with the transformation $x_{j,i} \mapsto \tilde{x}_{j,i} \sim \hat{F}_{x_j}$ and one-hot encoded with $\tilde{x}_{j,i} \mapsto \tilde{\boldsymbol{x}}_{j,i}^{cat}$, where $\hat{F}_{x_j}$ is the empirical distribution of the feature x_j in the data set. This corruption process can be seen as an extension of the *salt-and-pepper* noise for binary data to categorical data, see for example Vincent *et al*. (2008, 2010) for the *salt-and-pepper* noise for binary data.
- Masking to zero (*zero*): The original input and its one-hot encoding are corrupted with the transformation $\boldsymbol{x}_{j,i}^{cat} \mapsto \tilde{\boldsymbol{x}}_{j,i}^{cat} = \mathbf{0}'$, where $\mathbf{0}$ is a vector of zeros. This corruption process is an analogue to the technique of masking applied to numerical features from Section 3.1.

We conclude this section with some remarks on the types of architecture of our autoencoders for categorical data:

(a) The approach with the *Separate AEs* has at least two disadvantages compared to the *Joint AE*. First, we have to train a number of autoencoders equal to the number of categorical features, which may be time-consuming. Secondly, and more importantly, we neglect possible dependencies between different categorical features when creating representations with separate and independent autoencoders. The second disadvantage is explored in Experiment 1 below. We consider the approach with the *Separate AEs* as a benchmark since it gives us a representation of categorical data which matches the representation of categorical data learned with entity embeddings in supervised learning tasks.

(b) If the categorical features are binary features, then our approach with the *Joint AE* is aligned with the approach for binary data used by Vincent *et al*. (2008, 2010) in their experiments. For binary data, autoencoders which coincide with our *Joint AE* are also used in Generative Adversarial Imputation Nets, see for example Yoon *et al*. (2018).

(c) Hespe (2020) recommends a multi-task learning autoencoder for categorical data which agrees with our *Joint AE*. He also describes single-task learning autoencoders learned with loss functions different from the cross-entropy.

3.3. Experiment 1: the reconstruction ability of autoencoders

We compare the following four autoencoders for categorical data:

- *Separate AEs*,
- *Joint AE*,
- *MCA* — we build a linear autoencoder with the classical MCA algorithm, that is instead of training a neural network, we apply Generalized Singular Value Decomposition (GSVD) to a

matrix with centered one-hot encoded categorical features, see Pagès (2015) and Chavent *et al*. (2017),

- *MCA as non-linear PCA* — we build a non-linear autoencoder for numerical data, the one described in Section 3.1, on linearly transformed one-hot encoded categorial features. From Pagès (2015) and Chavent *et al*. (2017), MCA is PCA on centered one-hot encoded categorical data transformed with linear mappings (GSVD). Instead of building a linear autoencoder, which is equivalent to the PCA algorithm, on linearly transformed centered one-hot encoded categorial features, we build a non-linear autoencoder with the hyperbolic tangent activation function in the single hidden layer by minimizing the mean square reconstruction error.

From the data set freMTPL2freq with 678,013 observations, we sample 100,000 observations. We work with a smaller data set to speed up the calculations. We limit our attention to categorical features and we consider the six categorical features from Table 1. Our data set with 100,000 observations is next split randomly into five data sets with 20,000 observations. We build our autoencoders on each of these five sets and report the average metric for these five sets evaluated at the training set. As the metric, we use the *cosine similarity* measure, but the findings also hold for example for the number of correct predictions. In this experiment, we only build under-complete autoencoders without noise, as this is sufficient to derive the key conclusions. We train our autoencoders with 15, 100 and 500 epochs. We do not differentiate between a training, a validation and a test set (we do not discuss possible overfitting) since we are only interested in evaluating the reconstruction errors of the autoencoders. Details are presented in Section 1 in the Online Supplement.

The dimension of the data matrix with the one-hot encoded categorical features is 54. We consider a range of dimensions of the representation of the categorical features: $q_1 = l = 6, 8, 10, 12, 15, 20, 30$. For the *Separate AEs*, we have to specify the number of neurons l_j (the dimension of representation) for each feature j. We assume that the number of neurons l, which defines the global dimension of the representation for all categorical features, is split across the individual categorical features evenly, if possible, and if not possible, a larger number of neurons is allocated to a feature with a larger number of labels. For example if we choose $l = 6$, then we build representations of dimension 1 for each feature, if we choose $l = 12$, then we build representations of dimension 2 for each feature, but if we choose $l = 8$, then we build representations of dimension 2 for Region and VehBrand (these two features have the two largest number of labels in the data set) and representations of dimension 1 for the remaining features.

We present the results in Figure 4. It is obvious that the cosine similarity increases with the number of neurons and the number of epochs. For the large number of epochs (500), for which we achieve the smallest reconstruction errors for all our autoencoders in terms of the loss functions minimized in the training process, the autoencoders *Separate AEs* and *Joint AE* are very similar in terms of their reconstruction power measured with the cosine similarity and they are much better than the remaining two autoencoders. The first conclusion confirms that categorical data have different intrinsic properties than numerical data, which are explored when a low-dimensional representation is built with an autoencoder, and categorical data should not be compressed with algorithms derived for numerical data (MCA is just PCA on linearly transformed data). The second conclusion is that for the low and the medium number of epochs (15, 100), especially for the low number of epochs, the performance of *Joint AE* is superior in terms of its ability to reconstruct the input from a low dimensional representation. In particular, our experiment shows that there are dependencies between the categorical features in the data set which are efficiently captured by the *Joint AE* at initial epochs (15 epochs) of the learning process of the autoencoder, and which cannot be captured by learning independent *Separate AEs*. Intuitively, dependencies between categorical features should allow the *Joint AE* to learn more robust and informative representations of categorical features and the *Joint AE* should lead to better reconstruction errors compared to the *Separate AEs*. For the low number of epochs (15) and a low dimension of the representation (6, 8, 10), the *Joint AE* is very similar to the *MCA*, but the performance of the *Joint AE* improves quickly when we increase the number of epochs. Clearly, we can benefit from non-linear autoencoders when the purpose is to derive informative representations of categorical features.

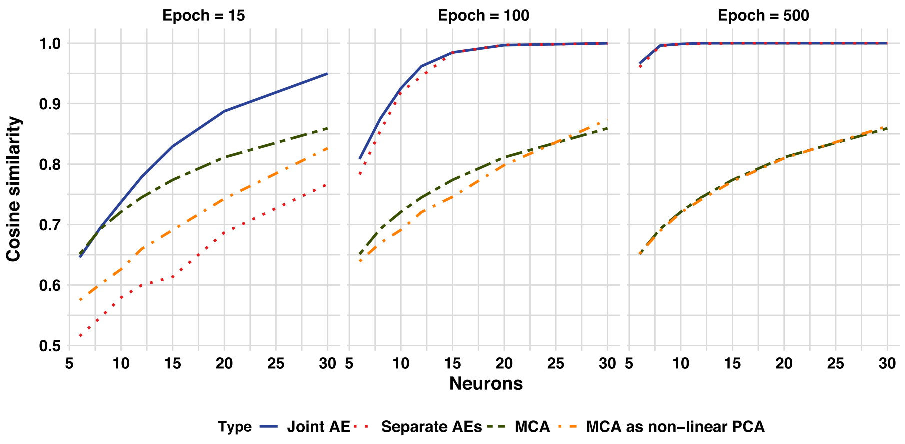

Figure 4. *Cosine similarity measures for autoencoders for categorical data.*

As discussed in Section 3, if we can find an autoencoder for which the reconstruction error is small, then we can say that the encoder extracts the most important information from a multi-dimensional vector of features. Our example points out that representations of categorical features built with the *Separate AEs* may not be optimal in terms of their robustness and informativeness, especially if we do not want to spend much time on training autoencoders with a large number of epochs. It is known that the predictive power of neural networks and their generalization properties in supervised learning tasks depend on providing a good representation of the available information for its efficient pre-processing in hidden layers before the final prediction of the response is constructed with the output. Since the *Joint AE* performs better than the *Separate AEs* in terms of providing a more robust and informative representation of categorical features, we may prefer to use the numerical representation of categorical features implied by the *Joint AE*, rather than the *Separate AEs*, as the input to neural networks built for supervised learning tasks. However, in all practical examples in actuarial data science to date, the numerical representation of categorical features which is fed into hidden layers of a neural network matches the representation from the *Separate AEs*. We have to use a different type of architecture of a neural network to use the representation from the *Joint AE*. This experiment may serve as a motivating example for what we present in the sequel.

4. Training neural networks with mixed categorical and numerical features

We now move to the main topic of this paper. Below, we discuss different approaches to training neural networks with mixed categorical and numerical features in supervised learning tasks. These approaches differ in the architecture of the neural network and initialization of the parameters of the neural network.

4.1. Architecture A1 with separate entity embeddings

Let us start by recalling the concept of an entity embedding developed by Guo and Berkhahn (2016). An entity embedding for categorical feature x_j is a neural network which maps the categorical feature x_j, with its one-hot representation $\boldsymbol{x}_j^{cat}$, into a vector of dimension l_j:

$$\boldsymbol{x}_j^{cat} \in \mathbb{R}^{m_j} \mapsto \boldsymbol{x}_j^{ee} = (x_{j1}^{ee}, \ldots, x_{jl_j}^{ee})' \in \mathbb{R}^{l_j},$$

where

$$x_{jr}^{ee} = \langle \boldsymbol{w}_r^{ee}, \boldsymbol{x}_j^{cat} \rangle, \qquad r = 1, \ldots, l_j.$$

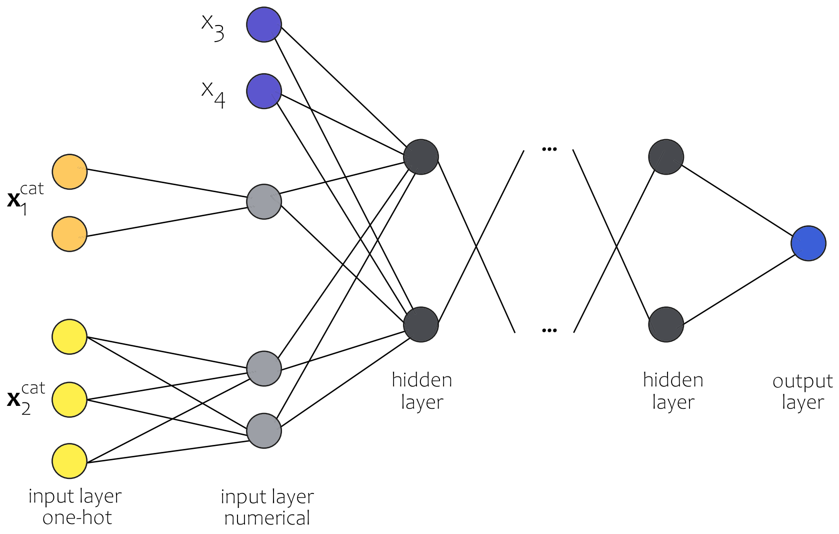

Figure 5. *Architecture of type A1 with separate entity embeddings.*

With an entity embedding, each label, from the set of m_j possible labels $\{a_1, ..., a_{m_j}\}$ of the categorical feature x_j, can be represented with a vector in the space $\mathbb{R}^{l_j}$. The parameter l_j is the dimension of the embedding for the categorical feature x_j.

In Figure 5, we provide an example of the architecture of a neural network with mixed categorical and numerical features used in supervised learning tasks in actuarial data science. This architecture uses entity embeddings for categorial features and has been promoted by Richman (2021), Noll *et al*. (2019) and Ferrario *et al*. (2020). We present a simple example with two categorical features $\boldsymbol{x}_1^{cat}, \boldsymbol{x}_2^{cat}$, with 3 and 2 levels, and two numerical features x_3 and x_4. For $\boldsymbol{x}_1^{cat}$, we implement the entity embedding of dimension 2, and for $\boldsymbol{x}_2^{cat}$ — the entity embedding of dimension 1. More generally, within (2.1)–(2.2), we define a neural network with Architecture 1 (A1):

- For each categorical feature x_j, $j = 1, .., c$, we build an entity embedding — a sub-network without hidden layers, that is $M = 0$, where the input $\boldsymbol{z} = \boldsymbol{x}_j^{cat}$, $q_0 = m_j$ and the output $q_1 = l_j$,
- Once all one-hot encoded categorical features are transformed with linear mappings of the entity embeddings, the outputs from the entity embeddings, that is the numerical representations of the categorical features, are concatenated with the numerical features to yield a new numerical vector of all features. This new vector is fed as the input into another sub-network with M hidden layers,
- We build a sub-network with M hidden layers with neurons $q_1, ..., q_M$ and the hyperbolic tangent activation functions $\chi^1, ..., \chi^M$ in the hidden layers, where the input $\boldsymbol{z} = \big((\boldsymbol{x}_1^{ee})', ..., (\boldsymbol{x}_c^{ee})', x_{c+1}, ..., x_d\big)'$ and $q_0 = \sum_{j=1}^{c} l_j + d - c$,
- All weights of the network (including the weights of the entity embeddings) are initialized with values sampled from uniform distributions with the Xavier initialization, see Glorot and Bengio (2010), and the bias terms are initialized with zero.

The goal of this paper is to challenge A1 with a new architecture and a new training process of a neural network. The results from Experiment 1 provide us with arguments regarding how we could change A1. We can now clearly observe that the numerical representations of the categorical features learned with the entity embeddings in A1 matches, in their architectures, the numerical representations learned with the *Separate AEs*. From Section 3.3, we conclude that we could replace the numerical representations of the categorical features in A1 with the representation learned with the *Joint AE*. This leads us to introduce Architecture 2.

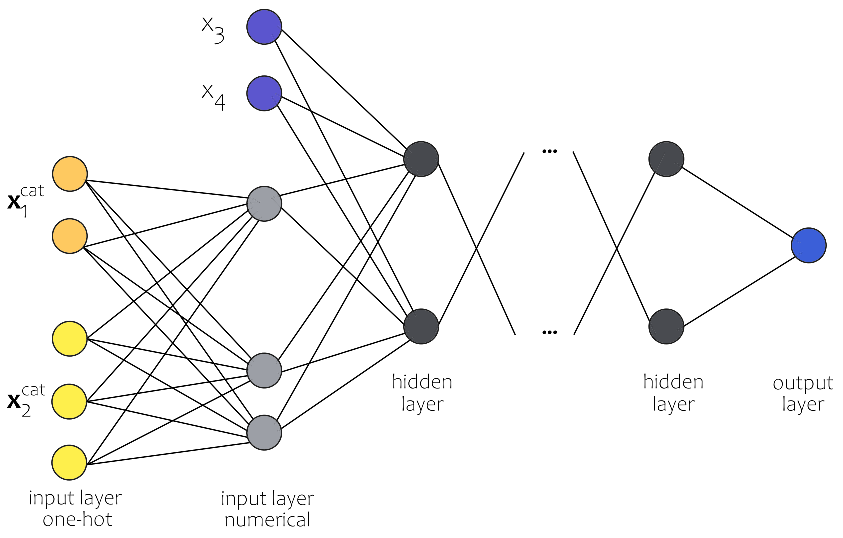

Figure 6. *Architecture of type A2 with joint embedding.*

4.2. Architecture A2 with joint embedding

Instead of applying separate entity embeddings to each categorical feature, we now use a joint embedding for all categorical features. A joint embedding is understood here as a neural network with the following mapping:

$$\boldsymbol{x}^{cat} = ((\boldsymbol{x}_1^{cat})', ..., (\boldsymbol{x}_c^{cat})')' \in \mathbb{R}^{\bar{m}_c} \mapsto \boldsymbol{x}^{\tilde{ee}} = (x_1^{\tilde{ee}}, ..., x_l^{\tilde{ee}})' \in \mathbb{R}^l,$$

where

$$x_r^{\tilde{ee}} = \langle \boldsymbol{w}_r^{\tilde{ee}}, \boldsymbol{x}^{cat} \rangle, \quad r = 1, ..., l.$$

Parameter l is the dimension of the embedding for all categorical features $(x_1, ..., x_c)$. We expect that $l < l_1 + ... + l_c$.

Our new architecture of a neural network with mixed categorical and numerical features where the categorical features are modelled with a joint embedding is presented in Figure 6. For $\boldsymbol{x}_1^{cat}, \boldsymbol{x}_2^{cat}$, we implement a joint embedding of dimension 3—this is the only, but also a significant difference between the architectures in Figures 5 and 6. Within (2.1) and (2.2), we define a neural network with Architecture 2 (A2):

- For all categorical features $(x_1, ..., x_c)$, we build a joint embedding — a neural sub-network without hidden layers, that is $M = 0$, where the input $\boldsymbol{z} = \boldsymbol{x}^{cat}$, $q_0 = \bar{m}_c$ and the output $q_1 = l$,
- The next steps of building the network for predicting the response are the same as for A1.
- We initialize all weights with the Xavier initialization and set the bias terms equal to zero, as for A1.

We can observe that the numerical representation of the categorical features learned with the joint embedding in A2 matches, in its architecture, the representation learned with the *Joint AE*. We have already discussed the advantages of this representation in unsupervised learning tasks, which should also hold in supervised learning tasks. In addition, we can expect that by learning a joint embedding for all categorical features, we allow all categorical features, not only labels for a single categorical feature, to share the information about their impact on the response. As a result, we should be able to improve predictions of the response based on the experience collected from similar categorical features and their similar labels. Hence, the switch from A1 to A2 has intuitive foundations. To the best of our knowledge, A2 has not been considered to date in any actuarial data science problem.

4.3. Initialization of A1 and A2

The issue of initialization of parameters of neural networks has been noticed in actuarial data science. Under A1, Merz and Wüthrich (2019) and Schelldorfer and Wüthrich (2019) propose the Combined Actuarial Neural Network (CANN) approach to initialize a neural network with predictions from a GLM — we call this architecture and the training process A1_CANN. The idea is to add a skip connection to the output from the network with architecture A1. In mathematical terms, in A1 we use the prediction:

$$\lambda_i = e^{\log(Exp_i)+\Theta_1^{M+1}\left(((\boldsymbol{x}_{1,i}^{ee})',\ldots,(\boldsymbol{x}_{c,i}^{ee})',x_{c+1,i},\ldots,x_{d,i})'\right)}, \quad i=1,\ldots,n, \tag{4.1}$$

whereas in A1_CANN we use the prediction:

$$\lambda_i = e^{\log(Exp_i)+\eta_i^{GLM}+\Theta_1^{M+1}\left(((\boldsymbol{x}_{1,i}^{ee})',\ldots,(\boldsymbol{x}_{c,i}^{ee})',x_{c+1,i},\ldots,x_{d,i})'\right)}, \quad i=1,\ldots,n, \tag{4.2}$$

where η_i^{GLM} denotes the prediction, on the linear scale, of the unit intensity (for exposure equal to one) from a Poisson GLM with a log link for observation i.

In A1 and A2, we initialize the weights of the embeddings for the categorical features with the Xavier initialization. However, since autoencoders extract important information about features, we could initialize the weights of the embeddings with the weights from the encoder of the appropriate autoencoder and define the weights of the embeddings as non-trainable in the training process. This is reasonable, but may be sub-optimal for a supervised learning task since the representation of the categorical features learned with an autoencoder without the information about the response would be kept fixed. To improve the representation from an autoencoder, we should fine-tune it in a supervised learning task with a target response. In the machine learning literature, Erhan *et al*. (2010, 2009) propose to pre-train layers of neural networks with denoising autoencoders, that is initialize neurons in layers of a neural network for a supervised learning task with representations of the neurons from denoising autoencoders built in unsupervised learning tasks for the input to the layers. We recover and modify their approach in this paper.

Apart from changing the architecture from A1 to A2, we initialize the weights and the bias terms in the joint embedding for the categorical feature and the first hidden layer in A2 with the weights and the bias terms from the representations of the neurons in the layers learned with autoencoders. From Erhan *et al*. (2010, 2009), we know that the initialization procedure with autoencoders gives the largest gains in predictive power of a neural network when it is applied to initial layers of the network. We proceed in the following way:

- We build an autoencoder of type *Joint AE* (denoted as the 1st AE) for the categorical input $(x_1, \ldots, x_c)$ using its one-hot representation $\boldsymbol{x}^{cat} = \left((\boldsymbol{x}_1^{cat})', \ldots, (\boldsymbol{x}_c^{cat})'\right)'$. To build a denoising autoencoder, we corrupt the categorical input with the *sample* or the *zero* transformation, see Section 3.2,
- We take the weights from the encoder of the 1st AE, denoted by $\boldsymbol{w}_r^{enc} = (w_{r,1}^{enc}, \ldots, w_{r,\bar{m}_c}^{enc})$, $r = 1, \ldots, l$, and initialize the weights $\boldsymbol{w}_r^{\bar{e}e}$, $r = 1, \ldots, l$, of the joint embedding in A2 with these weights,
- We take the representation of the categorical features predicted by the 1st AE: $\boldsymbol{x}^{enc} = (x_1^{enc}, \ldots, x_l^{enc})'$ with $x_r^{enc} = \langle \boldsymbol{w}_r^{enc}, \boldsymbol{x}^{cat} \rangle$, $r = 1, \ldots, l$, concatenate this vector with the vector of the numerical features $(x_{c+1}, \ldots, x_d)'$ and create a new vector of numerical features $\boldsymbol{z} = ((\boldsymbol{x}^{enc})', x_{c+1}, \ldots, x_d)'$,
- We build an autoencoder from Section 3.1 (denoted as the 2nd AE) for the numerical input $\boldsymbol{z}$. The dimension of the representation to be learned for the $(l + d - c)$-dimensional vector $\boldsymbol{z}$ is equal to q_1, where q_1 denotes the number of neurons used in the first hidden layer in the sub-network with M hidden layers, which is built for the input constructed by concatenating the representation from the joint embedding with the numerical features. To build a denoising autoencoder, we corrupt the numerical input with the *gaussian* or the *zero* transformation, see Section 3.1,

- We take the weights and the bias terms from the encoder of the 2nd AE and initialize the weights and the bias terms $b_r^1, \boldsymbol{w}_r^1, r = 1, ..., q_1$, in the first hidden layer in the sub-network with M hidden layers with these weights and the bias terms.
- All other weights are initialized with the Xavier initialization and the bias terms are initialized with zero.

The initialization procedure applied here also clarifies why we were only interested in building autoencoders with one hidden layer in Section 3. For A2, and any initialization of layers, we use the predictions:

$$\lambda_i = e^{\log(Exp_i) + \Theta_1^{M+1}\left(((\boldsymbol{x}_i^{\tilde{e}e})', x_{c+1,i}, ..., x_{d,i})'\right)}, \quad i = 1, ..., n. \tag{4.3}$$

The autoencoders, which are trained without the information about the response, are only used to derive initial values of the neurons in the two layers of A2. These initial values are next fine-tuned by training the whole neural network to predict the target response. When training an autoencoder, we are only interested in extracting the most important discriminatory factors in the multi-dimensional input vector, which are next improved and optimally transformed by taking into account the target response. Since in this application autoencoders are trained for a low number of epochs, in Experiment 1, we should only look at the results for epochs 15 and 100, which show clear advantages of the representation of categorical features learned with the *Joint AE* compared to the *Separate AEs*.

The third step above where we concatenate the numerical representation of the categorical features from the 1st AE with the other numerical features deserves attention. We propose a modification of the pre-training strategy of layers with autoencoders which has not been considered by Erhan *et al*. (2010, 2009). It is known that the features, fed into a neural network, should live on the same scale in order to perform effective training of the network. We can easily control the numerical features and scale them to $[-1, 1]$, which is done before the training process is started. However, we cannot expect the numerical representation of the categorical features learned with the 1st AE, that is the values given by $x_r^{enc} = \langle \boldsymbol{w}_r^{enc}, \boldsymbol{x}^{cat} \rangle$, $r = 1, ..., l$, to yield predictions in $[-1, 1]$. If the predictions from the encoder of the 1st AE live on a scale different from $[-1, 1]$, which is the scale where the numerical features live, then the input to the 2nd AE and the input to the first hidden layer of the sub-network with M hidden layers will have features on different scales, and the training process of the neural networks may suffer from this inconsistency in scales. Fortunately, we can modify the weights and the bias terms of the encoder and the decoder of the 1st AE to keep the reconstruction error unchanged and have the representation of the categorical features in the desired scale. This is possible due to the linear activations functions assumed and the bias terms chosen to be trained in the output layer in the autoencoder of type *Joint AE*, before the soft-max functions are applied. In the encoder part of the *Joint AE*, we re-define the weights:

$$w_{r,k}^{enc} \mapsto w_{r,k}^{enc,*} = \frac{2}{\max_i\{x_{r,i}^{enc}\} - \min_i\{x_{r,i}^{enc}\}} w_{r,k}^{enc} - \frac{1}{c}\left(\frac{2\min_i\{x_{r,i}^{enc}\}}{\max_i\{x_{r,i}^{enc}\} - \min_i\{x_{r,i}^{enc}\}} + 1\right),$$

for $r = 1, ..., l$ and $k = 1, ..., \bar{m}_c$. We can deduce that for these new representations we have $\langle \boldsymbol{w}_r^{enc,*}, \boldsymbol{x}_i^{cat} \rangle \in [-1, 1]$ for all $r = 1, ..., l$ and all observations $i = 1, ..., n$. Since $\boldsymbol{x}_i^{cat}$ is always a vector with c elements equal to 1 and the remaining elements are equal to zero, the constant term $-2\min_i\{x_{r,i}^{enc}\}/\left(\max_i\{x_{r,i}^{enc}\} - \min_i\{x_{r,i}^{enc}\}\right) - 1$ from the min–max scaler transformation of the original predictions from the encoder $\langle \boldsymbol{w}_r^{enc}, \boldsymbol{x}_i^{cat} \rangle$, for each neuron r, can be absorbed by the new weights of the encoder by dividing the constant by c. Let $(b_r^{dec}, \boldsymbol{w}_r^{dec})_{r=1}^{\bar{m}_c}$ denote the weights and the bias terms from the decoder. In the decoder part of the *Joint AE*, we now re-define:

$$w_{r,k}^{dec} \mapsto w_{r,k}^{dec,*} = \frac{\max_i\{x_{k,i}^{enc}\} - \min_i\{x_{k,i}^{enc}\}}{2} w_{r,k}^{dec},$$

$$b_r^{dec} \mapsto b_r^{dec,*} = b_r^{dec} + \sum_{k=1}^{l}\left(w_{r,k}^{dec,*} + \min_i\{x_{k,i}^{enc}\} w_{r,k}^{dec}\right),$$

for $r = 1, ..., \bar{m}_c$ and $k = 1, ..., l$. We can conclude that the predictions in the output layer from the autoencoder with the modified weights and bias terms remain exactly the same as in the original autoencoder, hence the reconstruction error remains unchanged. Since the bias terms are needed in the decoder to adjust the representation, in Section 3.2, we decided to train the bias terms in the decoder of the autoencoder for categorical data.

Let us conclude with remarks on our architectures A1–A2:

(a) We could initialize the representations of the categorical features in A1 with the weights from the encoders from the *Separate AEs*. Based on the results from Experiments 1, we expect that this type of initialization of A1 would not be an efficient solution for improving predictive power of neural networks, and we decided not to proceed with this approach in this paper. Moreover, training multiple autoencoders in unsupervised learning tasks for initialization of a neural network for a supervised learning task would be time-consuming and would be unlikely to gain popularity in practical applications.

(b) Other architectures of neural networks are also possible. For example, we could consider Architecture 3. First, the one-hot encoded categorical features are centered and linearly transformed with non-trainable mappings defined as in the MCA algorithm before the PCA algorithm is applied. Then, they could be treated as numerical data together with the other numerical features. Such an approach is proposed in the Factor Analysis of Mixed Data, see Pagès (2015) and Chavent *et al*. (2017). In other words, we could define neurons in the first hidden layer of a neural network as linear transformations of linearly transformed one-hot encoded categorical features and numerical features. In Figure 6, we would remove the intermediate layer with grey neurons. We would only need the autoencoder for numerical data to pre-train the first hidden layer of the network and the autoencoder for the categorical data would not be needed at all. Based on the results from Experiment 1, we reject such architecture because we believe that categorical data should be treated differently from numerical data. This view is also supported with experiments presented by Brouwer (2004) and Yuan *et al*. (2020).

4.4. Experiment 2 — the predictive power of A1 and A2

We study architectures and training processes of neural networks denoted by A1, A1_CANN, A2, A2_MCA, A2_1AE, A2_2AEs, where A1, A1_CANN, A2 are defined above and we introduce:

- A2_MCA — we only pre-train the joint embedding of A2 with a linear autoencoder, that is we initialize the weights of the joint embedding for the categorical features with the weights from the encoder from a linear autoencoder built with the MCA algorithm. In our experiment, and also in general, we cannot apply a linear autoencoder built with the PCA algorithm as the 2nd AE since PCA only allows us to build under-complete autoencoders, whereas the number of neurons in the first hidden layer of a sub-network with M hidden layers is usually much larger than the dimension of the input to the layer — this remark can serve as an additional argument for using over-complete autoencoders for pre-training layers of neural networks rather than classical under-complete autoencoders,
- A2_1AE — we only pre-train the joint embedding of A2 with a non-linear autoencoder, that is we initialize the weights of the joint embedding for the categorical features with the weights from the encoder from an autoencoder of type *Joint AE*. We use only one non-linear autoencoder since we want to directly validate MCA with a non-linear autoencoder,
- A2_2AEs — our main approach, in which we pre-train the joint embedding and the first hidden layer of A2 with two non-linear autoencoders, that is we initialize the weights of the joint embedding for the categorical features with parameters from the encoder from an autoencoder for categorical data of type *Joint AE* and we initialize the weights and bias terms of the first hidden layer in the sub-network with M hidden layers with parameters from the encoder from an autoencoder for numerical data from Section 3.1.

A1_CANN is initialized with GLM1 from Schelldorfer and Wüthrich (2019), which is a Poisson GLM with log link function where the features in Table 1 are used as regressors and the categorical features are coded with dummy variables.

The dimension of the categorical input, which consists of the one-hot encoded categorical features, is equal to 54. We set the dimension of the representation of the categorical features to 8. For A1, we build separate representations of dimension 2 for Region and VehBrand and separate representations of dimension 1 for all other features—Area, VehPower, VehAge and DrivAge. This choice is compatible with Experiment 1 and the choice made by Noll *et al*. (2019), Schelldorfer and Wüthrich (2019) and Ferrario *et al*. (2020). For A2, we build a joint representation of dimension 8 for all categorical features. The dimension of the input to the first hidden layer, which consists of the numerical features and the numerical representation of the categorical features, is equal to 11, since we concatenate the representation of the categorical features learned with the embeddings with the three numerical features — BonusMalus, Density and VehGas. The number of neurons in the single hidden layer in the 1st AE is 8, as this number must coincide with the dimension of the representation of the categorical features for our supervised learning task. The number of neurons in the single hidden layer in the 2nd AE is equal to the number of neurons in the first hidden layer of the sub-network with M hidden layers. We consider sub-networks with $M = 3$ hidden layers in our experiments below. We consider three possible choices for the numbers of neurons in the hidden layers in A2, similar to Noll *et al*. (2019), Schelldorfer and Wüthrich (2019) and Ferrario *et al*. (2020), and we define the numbers of neurons for A1 so that the number of trainable parameters in A1 and A2 are equal, see Table 2.1 in Section 2 in the Online Supplement for the numbers of neurons.

Experiment 2 is conducted on the same 100,000 observations as Experiment 1. Since the predictive power of neural networks depends on their hyperparameters, in the first step of this experiment, we perform hyperparameter optimization. With hyperparameter optimization, we also control over-fitting of the networks. We try to identify the best hyperparameters for the two autoencoders (the 1st AE and the 2nd AE) trained in an unsupervised process and the best hyperparameters for the neural networks with Architectures 1 and 2 trained in a supervised process. The hyperparameters optimized in the experiment, together with their best values, are presented in Section 2 in the Online Supplement, where the hyperparameter optimization process is described in detail. As a part of the hyperparameter optimization, we choose between denoising autoencoders and autoencoders without noise. We point out that we prefer a denoising autoencoder for pre-training the joint embedding of A2, see Table 2.3 in Section 2 in the Online Supplement.

In the second step of this experiment, we study in more detail the predictive power of the best neural networks identified in the first step of the experiment for each architecture and training process. The set of 100,000 observations is split into a training, a validation and a test set to the proportions 3:1:1. We perform 100 calibrations for each best approach for A1, A1_CANN, A2, A2_MCA, A2_1AE, A2_2AEs. In each calibration, we train the autoencoders in unsupervised learning tasks, if required, and the neural network for the supervised learning task. The training process is the same as in the hyperparameter optimization. We train the networks on the training set by minimizing the Poisson loss, early stop the algorithm on the validation set and evaluate the predictive power of the trained networks by calculating the Poisson loss on the test set.

The box plots of the Poisson loss values on the test set in 100 calibrations are presented in Figure 7, and their key characteristics in Table 2. By initializing A1 with GLM1, we gain on average a small predictive power of 0.0052 and we increase the standard deviation of the loss from 0.0384 to 0.0708. In general, the predictive power of A1_CANN depends on the GLM used for initialization of A1 and here we use one of the simplest GLMs investigated by Noll *et al*. (2019). As discussed by Schelldorfer and Wüthrich (2019), A1_CANN could only benefit from a very good initial GLM. If we switch from A1 to A2, then the predictive power of the network increases slightly on average by 0.0095, but at the same time, A2 has standard deviation of the loss twice as high as A1. A1, A1_CANN and A2 are all close in terms of their predictive power and we do not find strong evidence that A1_CANN and A2 are better than

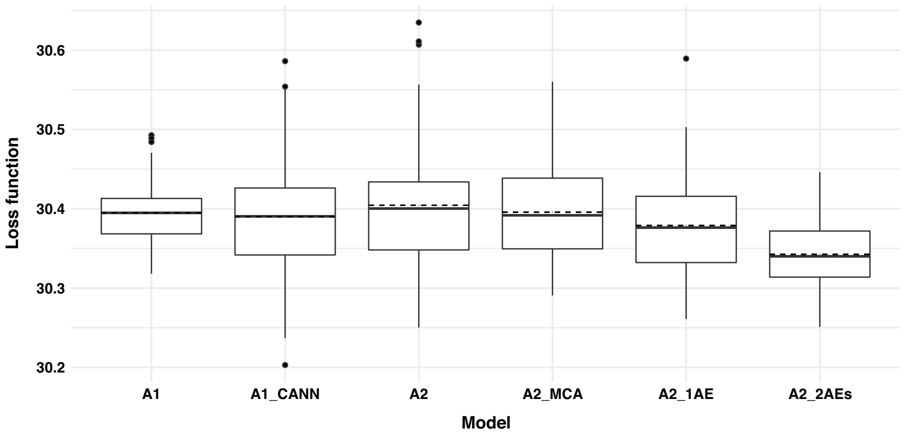

Figure 7. *Distributions of the Poisson loss on the test set (for each network the dotted line represents the average loss in 100 calibrations).*

A1. In fact, we observe that the Poisson loss values achieved in calibrations are dispersed more under A1_CANN and A2 than A1. If we improve the training process of A2 by initializing its parameters with the parameters from the autoencoders, then the performance of A2 improves in terms of the predictive power, the standard deviation and quantiles of the Poisson loss. By initializing the joint embedding of A2 with the linear autoencoder, we gain on average a small amount of predictive power of 0.0087. If we replace the linear autoencoder of type *MCA* with the non-linear denoising autoencoder of type *Joint AE*, then we can observe on average a significant gain in the predictive power of 0.0256 (A2_1AE vs. A2). This shows that linear autoencoders are not sufficient for pre-training layers of neural networks for supervised learning tasks and we have to rely on non-linear denoising autoencoders to initialize neural networks (it is not possible to use PCA for pre-training the first hidden layer of the sub-network with three hidden layers and only the joint embedding can be pre-trained with MCA). If we pre-train A2 with the two autoencoders for the categorical and the numerical input, then we reduce the Poisson loss on average by 0.0620 (A2_2AEs vs. A2). When we move from A2_1AE to A2_2AEs, the improvement in the predictive power on average of 0.0364 is possible only if we re-scale the weights from the autoencoder for the categorical input before we train the autonencoder for the numerical input (see Section 4.3 for details on this step). Without this step, A2_2AEs would fail to provide superior results. The size of the improvement in the predictive power when we switch from A2_1AE to A2_2AEs also depends on the choice of the autoencoder for the categorical input. Hence, the choice of the 1st AE (recall that we choose a denoising autoencoder) is important even though the 2nd AE leads to a larger decrease in the average value of the Poisson loss. By pre-training A2 with our two autoencoders, we also decrease the standard deviation of the loss. Most importantly, we finally compare A2_2AEs with A1. We achieve an improvement in the Poisson loss on average of 0.0525 for A2_2AEs compared to A1. All reported quantiles are lower for A2_2AEs than for A1, and the distribution of the loss from A2_2AEs is shifted to the left compared to A1, but the standard deviation of the loss from A2_2AEs is slightly larger than the standard deviation of the loss from A1. We can conclude that our new architecture with a joint embedding for all categorical features and initialized with parameters from (denosing) autoencoders is better, in terms of its predictive power, than the classical architecture nowadays commonly used in actuarial data science with separate entity embeddings for categorical features and random initialization of parameters. The improvement of the predictive power from 30.3950 to 30.3425 can indeed be interpreted as significant for this data set—Schelldorfer and Wüthrich (2019) demonstrate for example that the Poisson loss can decrease from 31.5064 to 31.4532 by optimizing the dimensions of the entity embedding, or the Poisson loss can decrease from 32.1490 to 32.10286 by boosting a GLM with one regressor transformed with a neural network (for the BonusMalus which achieves the largest improvement).

Table 2. *Distributions of the poisson loss on the test set, their quantiles (q), average values (avg) and standard deviations (SD).*

Statistics	A1	A1_CANN	A2	A2_MCA	A2_1AE	A2_2AEs
q0.05	30.3362	30.2875	30.3133	30.3142	30.2863	30.2767
q0.25	30.3684	30.3417	30.3481	30.3495	30.3322	30.3138
avg	30.3950	30.3898	30.4045	30.3958	30.3789	30.3425
q0.75	30.4130	30.4261	30.4338	30.4386	30.4157	30.3719
q0.95	30.4688	30.5101	30.5444	30.5104	30.4833	30.4126
sd	0.0384	0.0708	0.0758	0.0597	0.0624	0.0433

The bias of the predictors and the performance of auto-calibrated predictors is investigated in Section 3 in the Online Supplement.

5. Conclusion

We have presented a new approach to training neural networks with mixed categorical and numerical features for supervised learning tasks. We have illustrated that our new architecture of a network with a joint embedding for all categorical features and network parameters properly initialized with parameters from (denosing) autoencoders, learned in an unsupervised manner, performs better, in terms of the predictive power and the stability of the predictions, than the classical architecture, used nowadays in actuarial data science, with separate entity embeddings for categorical features and random initialization of parameters. We hope that the results described in this paper will draw attention in actuarial data science to a new possible architecture of a neural network for supervised learning tasks and benefits of autoencoders in deriving representations of features for supervised learning tasks. In fact, we are already aware of new (unpublished) experiments with autoencoders used for initialization of neural networks for actuarial pricing, see Holvoet *et al*. (2022).

Despite the results presented in the paper, there is one more advantage of our new architecture. As far as hyperparameter optimization is concerned for the classical architecture, we should optimize the loss function with respect to multiple hyperparameters which describe the dimensions of the entity embeddings for categorical features. In our new architecture, we only search for the optimal value of only one hyperparameter which specifies the dimension of the joint embedding for all categorical features. Consequently, our new architecture enables faster and more convenient optimization of the dimension of the representation of categorical features, see Section 4 in the Online Supplement. There is also a disadvantage of our approach. We lose simple graphical interpretation of the joint embedding of categorical features due to a higher dimension of the joint representation compared to one- or two-dimensional representations of entity embeddings. Hopefully, we should be able to use rich methods of explainable AI to interpret the impact of categorical features, modelled with a joint embedding, on the response.

Finally, we could modify our approach by learning autoencoders, used for pre-training layers of a network, jointly with a network with a target response, which uses the representations from the autoencoders as the input (at the additional cost of fine-tuning the weight between the unsupervised and the supervised loss). Such an approach is also postulated in the machine learning literature, see for example Ranzato and Szummer (2008), Lei *et al*. (2018). This last remark reinforces the conclusion stated above that autoencoders should be included in the toolbox of data science actuaries who build predictive models.

Acknowledgement. The authors would like to thank the referees and the Editor-in-Chief for very useful remarks which helped to improve the first version of this paper. At its initial stage, this research was partially funded with The European Regional Development Fund, Smart Growth Programme, as a part of a grant from The National Centre for Research and Development.

Supplementary material. To view supplementary material for this article, please visit https://doi.org/10.1017/asb.2023.15.

References

Blier-Wong, C., Baillargeon, J.-T., Cossette, H., Lamontagne, L. and Marceau, E. (2021) Rethinking representations in P&C actuarial science with deep neural networks. https://arxiv.org/abs/2102.05784.

Blier-Wong, C., Cossette, H., Lamontagne, L. and Marceau, E. (2022) Geographical ratemaking with spatial embeddings. *ASTIN Bulletin*, **52**, 1–31.

Brouwer, R. (2004) A hybrid neural network for input that is both categorical and quantitative. *International Journal of Intelligent Systems*, **19**, 979–1001.

Caruana, R. (1997) Multitask learning. *Machine Learning*, **28**, 41–75.

Chavent, M., Kuentz-Simonet, V., Labenne, A. and Saracco, J. (2017) Multivariate analysis of mixed data: The R package pcamixdata. https://arxiv.org/abs/1411.4911.

Dixon, M., Halperin, I. and Bilokon, P. (2020) *Machine Learning in Finance: From Theory to Practice*. Nature Switzerland, AG: Springer.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P. and Bengio, S. (2010) Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, **11**, 625–660.

Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S. and Vincent, P. (2009) The difficulty of training deep architectures and the effect of unsupervised pre-training. *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pp. 153–160.

Ferrario, A., Noll, A. and Wüthrich, M.V. (2020) Insights from inside neural networks. https://ssrn.com/abstract=3226852.

Gao, G. and Wüthrich, M. (2018) Feature extraction from telematics car driving heatmaps. *European Actuarial Journal*, **8**, 383–406.

Glorot, X. and Bengio, Y. (2010) Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pp. 249–256.

Grari, V., Charpentier, A., Lamprier, S. and Detyniecki, M. (2022) A fair pricing model via adversarial learning. https://arxiv.org/abs/2202.12008.

Guo, C. and Berkhahn, F. (2016) Entity embeddings of categorical variables. https://arxiv.org/abs/1604.06737.

Hainaut, D. (2018) A neural-network analyzer for mortality forecast. *ASTIN Bulletin*, **48**, 481–508.

Hespe, N. (2020) Building autoencoders on sparse, one-hot encoded data. https://towardsdatascience.com/building-autoencoders-on-sparse-one-hot-encoded-data-53eefdfdbcc7.

Holvoet, F., Antonio, K. and Henckaerts, R. (2022) Neural networks for frequency-severity modelling: A benchmark study from data preprocessing steps to technical tarif. Talk Presented in European Actuarial Journal Conference in Tatru.

Kuo, K. and Richman, R. (2021) Embeddings and attention in predictive modeling. https://arxiv.org/abs/2104.03545.

Lee, G., Manski, S. and Maiti, T. (2019) Actuarial applications of word embedding models. *ASTIN Bulletin*, **50**, 1–24.

Lei, L., Petterson, A. and White, M. (2018) Supervised autoencoders: Improving generalization performance with unsupervised regularizers. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 107–117.

Merz, M. and Wüthrich, M. (2019) Editorial: Yes, we CANN! *ASTIN Bulletin*, **49**, 1–3.

Miyata, A. and Matsuyama, N. (2022) Extending the Lee Carter model with variational autoencoder: A fusion of neural network and bayesian approach. *ASTIN Bulletin*, **53**, 798–812.

Noll, A., Salzmann, R. and Wüthrich, M. (2019) Case study: French Motor Third-Party Liability claims. https://ssrn.com/abstract=3164764.

Pagès, J. (2015) *Multiple Factor Analysis by Example Using R*. New York: CRC Press.

Ranzato, M. and Szummer, M. (2008) Semi-supervised learning of compact document representations with deep networks. *Proceedings of the 25th International Conference on Machine Learning*, pp. 792–799.

Rentzmann, S. and Wüthrich, M.V. (2019) Unsupervised learning: what is a sports car? https://ssrn.com/abstract=3439358.

Richman, R. (2021) AI in actuarial science - a review of recent advances - part 1. *The Annals of Actuarial Science*, **15**, 207–229.

Ruder, S. (2017) An overview of multi-task learning in deep neural networks. https://arxiv.org/pdf/1706.05098.

Schelldorfer, J. and Wüthrich, M. (2019) Nesting classical actuarial models into neural networks. https://ssrn.com/abstract=3320525.

Shi, P. and Shi, K. (2022) Nonlife insurance risk classification using categorical embedding. *North Americal Actuarial Journal.*

Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.-A. (2008) Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th International Conference on Machine Learning*, pp. 1096–1103.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.-A. (2010) Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, **11**, 3371–3408.

Yoon, J., Jordon, J. and Van der Schaar, M. (2018) GAIN: Missing data imputation using Generative Adversarial Nets. https://arxiv.org/abs/1806.02920.

Yuan, Z., Jiang, Y., Li, J. and Huang, H. (2020) Hybrid—DNNs: Hybrid Deep Neural Networks for mixed inputs. https://arxiv.org/abs/2005.08419.