An Explainable Attention Network for Fraud Detection in Claims Management

Helmut Farbmacher
Munich Center for the Economics of Aging, Max Planck Society, Germany

Leander Löw
Hamburg Business School, University of Hamburg, Germany

Martin Spindler*
Hamburg Business School, University of Hamburg, Germany

Insurance companies must manage millions of claims per year. While most of these are not fraudulent, those that are nevertheless cost insurance companies and those they insure vast amounts of money. The ultimate goal is to develop a predictive model that can single out fraudulent claims and pay out non-fraudulent ones automatically. Health care claims have a peculiar data structure, comprising inputs of varying length and variables with a large number of categories. Both issues are challenging for traditional econometric methods. We develop a deep learning model that can handle these challenges by adapting methods from text classification. Using a large dataset from a private health insurer in Germany, we show that the model we propose outperforms a conventional machine learning model. With the rise of digitalization, unstructured data with characteristics similar to ours will become increasingly common in applied research, and methods to deal with such data will be needed.

Key words: Claims management, health insurance, fraud detection, machine learning, deep learning, categorical data, embeddings

1. Introduction

Health insurers receive millions of claims per year. Given that information asymmetries between the principal (insurer) and the agents (health care providers and the insured) can lead to moral hazard, insurance companies face the choice of either paying out insurance claims immediately without any adjustments or reviewing claims that are suspicious. The most common method for undertaking the latter involves manually auditing claims data, which is a time-consuming and expensive process (see, for instance, Townsend 1979, Bond and Crocker 1997). Machine learning

* Corresponding author: martin.spindler@uni-hamburg.de Helpful comments were provided by Philipp Bach, Sven Klaaßen, Heinrich Kögel, Jannis Kück, the editors and two anonymous reviewers. We thank Max Hartmann and Tim Matheis for excellent research assistance.

models can greatly cut auditing costs by automatically screening incoming claims and flagging up those that are deemed to be suspicious – i.e. potentially incorrect – for subsequent manual auditing.

Health care claims have an unusual input data structure. During each doctor consultation or hospital visit, several tasks are performed and may be billed as separate items. Medical coders identify all services, prescriptions, and supplies received by the insured during his or her appointment and create an insurance claim. Each claim consists of multiple items, including the dates of treatment, a short description of the task or tasks performed, and the associated costs. The number of items varies from patient to patient, resulting in claims of variable length. Because ordinary machine learning methods, such as gradient boosting or random forests, require a vector of fixed size as input, the standard approach so far has been to find a fixed-size vector by manually engineering features based on domain knowledge. Doing so, however, requires costly domain experts and does not scale well to other problems, even if they are similar. Moreover, good features can be difficult to build with manual feature engineering. In search of a better solution and, in particular, a way to automate this process, we exploit the similarities between the data structure of a claim and the data structure of a text.

Our main contribution is to describe deep learning architectures that can be used to handle datasets with a varying number of variables per observation and/or variables with very many categories. Datasets like these share some similarities with text data because sentences in a text can be of varying length, and words in a sentence can be interpreted as a variable with a large number of categories (each word in a dictionary is a category). One difference to text data is that the order of words in a text is crucial to understanding the meaning of a sentence, whereas it is uninformative in the applications we consider here. There are many other applications in which the deep learning architectures we describe might play an important role. For instance, incorporating occupational classifications in labor economics, industry codes in industrial organization, or International Classification of Diseases (ICD) codes in health economics, entails dealing with variables with very many categories. Researchers in these and other fields will increasingly have to grapple with unstructured and massive datasets as these become more readily available due to the rise of digitalization.

In recent years, deep learning models based on artificial neural networks have begun to show human-level performance in text processing (see, for example, Kim 2014, Lai et al. 2015, Vaswani et al. 2017, Devlin et al. 2019). Because methods like these can directly handle unusual data structures and perform feature engineering as part of the learning process, they may represent an automated alternative to manual feature engineering. We pursue this approach for an important step in the claims management process: auditing and managing the risk of fraud and error. Additionally, we investigate the extent to which such models deliver meaningful explanations for

their predictions. These explanations can be used to shed light on suspicious claim items, greatly simplifying the subsequent auditing process and thus transforming the model from a tool that is merely predictive to one that is prescriptive.

In our empirical application, we train standard and non-standard machine learning models on real data containing past claims and their associated labels, which classify each claim as either correct or suspicious. The latter may refer to any kind of error, ranging from typographical errors and unintentional upcoding to fraud. For a comprehensive overview of various fraud behaviors, see Li et al. (2008). For the sake of simplicity, we refer in the following to the detection of suspicious errors as (insurance) fraud detection. For each claim, we want to predict the probability of fraud and pay out all non-fraudulent claims automatically without any further manual auditing. Fraudulent or suspicious claims, however, should be flagged up so that they can be audited. Supervised machine learning models are tailored to this task. Like all supervised models, they should be updated regularly to capture new types of fraud and changes in regulations.

There is a vast literature on the economics of auditing and optimal auditing (see, for instance, Townsend 1979, Mookherjee and Png 1989, Picard 1996, Schiller 2006, Dionne et al. 2009, and, for an overview, Picard 2013). Schiller (2006) provided evidence that auditing becomes more effective when insurers condition their audits on the information provided by fraud detection systems. Dionne et al. (2009) showed that an optimal auditing strategy takes the form of a red-flags approach, which entails referring a claim to the auditing unit once certain fraud signals have been observed. We, in turn, demonstrate that our deep learning model is a particularly useful tool in the claims management process because it not only adds another red flag but gives further insights into the fraud mechanisms themselves.

Detecting fraud or systemic abuse is a major challenge for health insurers (see, for instance, Becker et al. 2005, Heese 2018, Bastani et al. 2019), and there is also a vast literature on data-driven methods for fraud detection. For instance, Liou et al. (2008) compared the accuracy of logistic regression, neural networks, and classification trees in identifying fraud and claim abuse in diabetic outpatient services. Viaene et al. (2002) compared the performance of several classification techniques in detecting car insurance fraud. Van Vlasselaer et al. (2017) proposed a model that employs information about firm networks to detect social security fraud. Classification methods have also been used in other settings, for example to detect management fraud (Cecchini et al. 2010) or identify high-risk disability claims (Urbanovich et al. 2003). An overview of data-driven methods for fraud assessment is given by Bolton and Hand (2002) and Ekin et al. (2018), the latter of whom focus on the assessment of medical fraud.

In the next section, we discuss an important trade-off that a cost-minimizing insurer faces in the claims management process. Our economic evaluation criterion, which we use to assess the

performance of our machine learning model, reflects this trade-off. In Section 3, we describe the unusual data structure of our health care claims in detail. Section 4 gives a brief introduction to neural networks and deep learning. In Sections 5 and 6, we describe how deep learning can handle categorical variables with high cardinality, flexibly model nonlinearities, and aggregate the claims with variable length in a data-driven fashion—both generally and with regard to our specific application. Section 7 discusses the results of the models when applied to claims data from a private health insurer in Germany. Section 8 concludes.

2. The Economics of Auditing

A cost-minimizing insurance company faces a trade-off between the expected benefits of auditing (above all, the expected value of the adjustments to upcoded or fraudulent claims) and the auditing costs. This trade-off is an important component in models of optimal auditing (see, e.g., Mookherjee and Png 1989). Our economic evaluation criterion, which we use to evaluate our machine learning model, reflects this trade-off. The main goal of our machine learning model is to identify suspicious claims automatically. A crucial aspect in this regard is its ability to predict the probability of fraud (P) for new, unseen claims. In doing so, two types of error may occur: the model may classify a non-fraudulent claim as suspicious, which would be a false positive, or it might fail to identify a fraudulent claim, which would be a false negative. Our evaluation criterion takes both errors into account.

Let N denote the number of claims (both fraudulent and non-fraudulent), c the fixed costs of manual auditing, and a_i the adjustment associated with a particular claim. The last of these represents the cash benefit for that claim if correctly identified as suspicious. We assume that manual auditing of the suspicious claims detected by our models correctly identifies truly fraudulent claims with a probability equal to one. In this manner, the auditing process conditional on any machine learning model achieves a cost reduction of π as follows:

$$\pi = \sum_{i=1}^{N} I(P_i > \tau) \cdot (a_i - c)\,. \tag{1}$$

$I()$ is the indicator function, which equals 1 if the statement in parentheses is true. τ is a threshold chosen by the insurer. It can be any value between 0 and 1 depending on free capacities in the auditing unit, risk preferences of the principal, or (fraud) signals from agents. Such a signal could, for instance, be based on the total number of claims a health care provider submits to the insurer (Li et al. 2008) or on hours worked by the provider (Fang and Gong 2017). Splitting the claims into fraudulent and non-fraudulent cases, we have

$$\pi = \sum_{i \in \text{fraud}} I(P_i > \tau) \cdot (a_i - c) \; - \sum_{i \in \text{no fraud}} I(P_i > \tau) \cdot c \; . \tag{2}$$

The first term captures the net benefit of the true positives, and the second term the costs of the false positives. Any false positive causes a monetary loss c due to the manual auditing costs. Let $FP_\tau = \sum_{i \in \text{no fraud}} I(P_i > \tau)$ denote the number of false positives and $FN_\tau = \sum_{i \in \text{fraud}} I(P_i \leq \tau)$ the number of false negatives. Generally, a lower value of τ increases the number of suspicious claims at the cost of a (potentially) larger FP_τ, while a higher value of τ increases FN_τ .

To incorporate the cost reductions missed due to false negatives, we compare π with the maximum reduction in cost that can be achieved in the auditing process. This maximum is defined as

$$\pi^{\max} = \sum_{i \in \text{fraud}} (a_i - c) \, . \tag{3}$$

Note that the missing indicator function implies that there are no false negatives. Moreover, there are also no false positives in $\pi^{\max}$. That is, it reflects the maximum cost reduction that can be achieved if we know the fraudulent claims a priori from an oracle model. Finally, our evaluation criterion is defined as

$$\gamma = \pi^{\max} - \pi \, , \tag{4}$$

which measures the foregone cost savings of any screening model. A cost-minimizing insurer aims to set $\pi = \pi^{\max}$.

For simplification, our evaluation criterion abstracts from several issues. First, we assume that the cost of manual auditing is a fixed amount per suspicious claim. However, one can imagine that these costs do not scale to all sizes of auditing unit. A unit that processes only pre-selected claims, for example, may be less expensive per claim than a larger auditing unit that has to verify all of the claims an insurer receives. As a result, our evaluation criterion may underestimate the potential benefits of a fraud detection system. Second, the benefits of auditing claims are not limited to identifying and adjusting fraudulent claims but rather extend to the deterrent effect that auditing can have on potential defrauders. Tennyson and Salsas-Forn (2002) and Dionne et al. (2009) are two of many authors who discuss the importance of claims auditing as a deterrent. Effective machine learning models could strengthen this effect. Third, meaningful explanations of suspicious claims may considerably shorten the subsequent (manual) auditing process. In summary, our evaluation criterion does not represent a general measure of the benefits of auditing but rather focuses on the direct net gains of our machine learning model.

3. Data Structure and Descriptives

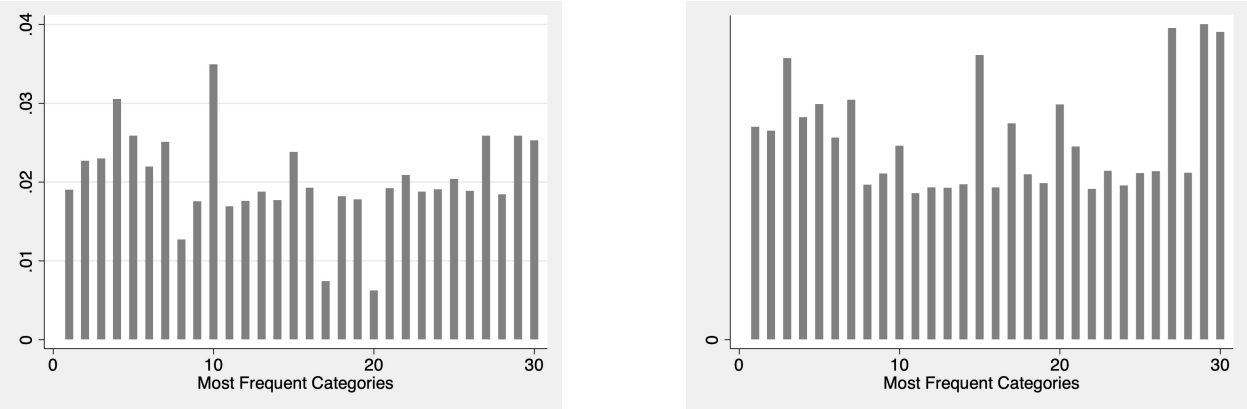
Generally, items in health care claims consist of several variables that indicate the quantity and type of medical treatment provided, potential complications, and the price of the treatment. Depending on the type of health insurance and the country in question, prices vary for the same service and may thus contain additional information about the potential for settling suspicious claims. This is, for instance, the case for private health insurance in the US. Alternatively, there may be a physician fee schedule, which determines a fixed price for any medical service (as, for instance, in Medicare in the US or generally in private and public health insurance in Germany). In our empirical application, we employ health care claims from a private health insurer in Germany. These contain three variables for every claim item: a procedure code specifying which medical service was provided, a multiplier, which measures the patient-specific severity of the task, and the price of the medical service in question (as a numerical value). Each procedure code is assigned a base price, which, after the multiplier has been applied, yields the price of this medical service.

The dataset we use is a copy of all claims submitted to the private health insurer during a certain period. The structure of the dataset is representative of the German private health insurance market as a whole. It contains 381,013 claims comprising more than three million claim items. On average, a claim in our dataset contains 8.6 items (range: 1-100), and 50% of the claims contain no more than 5 items. Each claim item consists of three variables: (1) the procedure code, which is a categorical variable with $2,205$ categories; (2) the multiplier with six categories; and (3) the price, which we scale between zero and one after log transformation. The input vectors of a claim can be set up as a two-dimensional matrix, where each row is one claim item consisting of these three variables. Every claim has an associated label indicating whether an adjustment was made to it. For example, when the original charges in a claim were 100€ but ultimately only 80€ were paid, then we define the difference as the adjustment and label the claim as fraudulent. 1.5% of the claims in our data are fraudulent. The labels, which originate from manual auditing, refer to any kind of error, ranging from typos and unintentional upcoding to fraud. They might miss truly fraudulent claims and hence mislabelling might be prevalent among the non-fraud labels. This is a typical challenge of supervised learning.

Of course, insurance companies have additional variables at their disposal to detect fraud, such as characteristics of providers and the insured. While these can easily be included in our deep learning model, we refrain from doing so for confidentiality reasons and to focus on the peculiar data structure of the claims, which was the scope of our cooperation with the insurer.

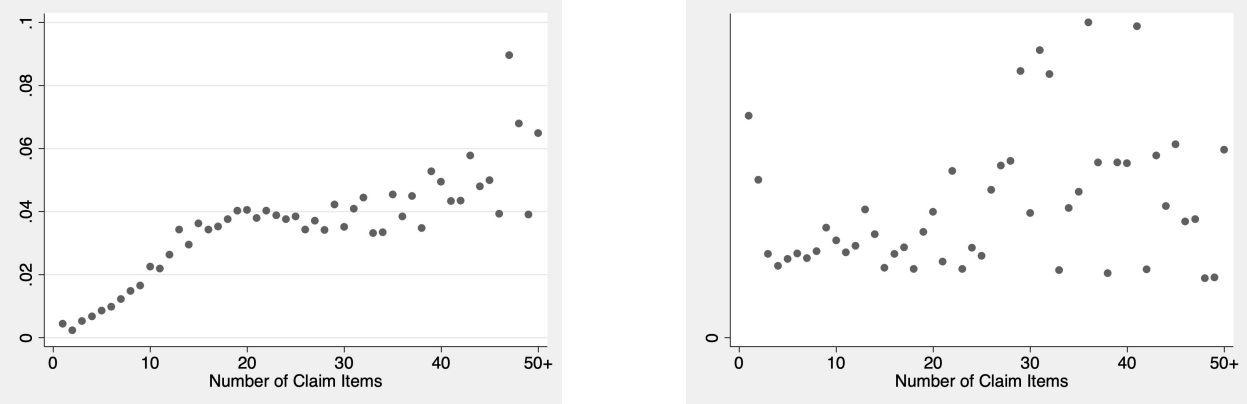
We briefly examine the descriptive relationship between a claim being fraudulent and some of the important information in our data.[1] Figure 1 shows the probability of fraud and the average adjustments conditional on fraudulent claims separately for the most frequent categories of the procedure code. The considerable heterogeneity that we can observe across these suggests that there is also considerable heterogeneity across the many other categories that we will use later in the model. Figure 2 relates the fraud probability and the average adjustments to the length of the claims, i.e. the number of claim items. Claims with more than 50 items are aggregated to a single scatter dot. The probability of fraud tends to increase with the claim length, particularly for relatively short claims. The average adjustments show a highly nonlinear relationship with claim length.

Figure 1 Heterogeneity over Most Frequent Categories of Procedure Code



(a) Fraud Probability (b) Average Adjustments (cond. on fraud)

Figure 2 Descriptive Relationship with Claim Length



(a) Fraud Probability (b) Average Adjustments (cond. on fraud)

[1] For confidentiality reasons, we are not allowed to publish the absolute amounts of the adjustments and have therefore deleted the labels of the axes. However, the value zero is included in all graphs to allow for a relative assessment of the figures.

4. A Short Introduction to Neural Networks

In this section, we give a brief introduction to neural networks. We focus on the basic concepts required to understand how our deep learning model can automatically classify claims, and we connect these concepts to the econometric terminology. Machine learning methods like deep learning or boosted trees can be applied to both regression problems (continuous outcome variables) or classification problems (discrete outcome variables). In the regression case, we want to approximate the unknown relation $f(x)$ between a dependent variable y and a p-dimensional set of input variables $x = (x_1, \ldots, x_p)'$:

$$y = f(x) + \epsilon$$

with ϵ denoting the error term. In the classification task, we model the probability of belonging to a certain class instead.

Deep learning, which in essence is an approach that uses neural networks with many layers, is a powerful way to estimate complex nonlinear relationships. All standard feedforward neural networks have a very similar structure, which consists of an input layer and an output layer, as well as several hidden layers. Each layer can consist of many units, also called neurons, which can be interpreted as constructed regressors. Every layer can have a different number of units. The input layer contains the input variables (also called features) in our model. The units within a hidden layer receive inputs from the previous layer, execute a sequence of calculations, and then pass the result on to the subsequent layer. Finally, the output layer combines all results to predict the outcome variable. In the binary classification task considered here, the output layer has just one unit, i.e. it predicts only a single value, which can be interpreted as the (classification) probability.

Every unit $k\,(=1,\ldots,K_j)$ in layer $j\,(=1,\ldots,J)$ applies a series of calculations. First, every unit k computes a linear index ($z_k^{[j]}$) of the weights ($w_k^{[j]}$) and the inputs from the previous layer ($a^{[j-1]}$), and adds an intercept term ($b_k^{[j]}$),

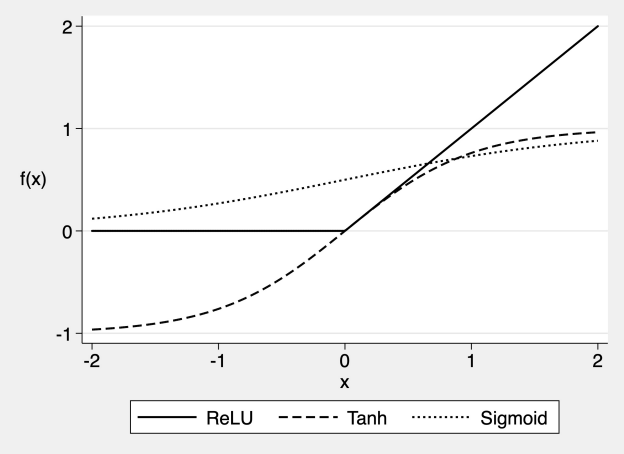
$$z_k^{[j]} = a^{[j-1]\prime} w_k^{[j]} + b_k^{[j]} \tag{5}$$

where $a^{[j-1]}$ and $w_k^{[j]}$ are vectors and $b_k^{[j]}$ is a scalar. The output of the initial (input) layer can be interpreted as $a^{[0]} = x$. In the neural network literature, the coefficients are usually called weights and the intercept is called bias. The biases and weights are usually allowed to be different for each unit. Second, the linear index is passed through a so-called activation function. To achieve non-linearity, the activation functions are key. Each unit calculates

$$a_k^{[j]} = g(z_k^{[j]}) \tag{6}$$

with $g(\cdot)$ being a non-linear activation function and $a^{[j]} = \left(a_1^{[j]}, \ldots, a_{K_j}^{[j]}\right)'$ forming the inputs of the subsequent layer. Usually, the activation function is the same for all units within a layer. The non-linearity of the activation functions enables the neural net to approximate a potentially complex relation between outcome and predictors. There are several activation functions in the literature. The most common ones are illustrated in Figure 3: rectified linear units, $\text{relu}(x) = max(0, x)$; the logistic sigmoid, $\sigma(x) = (1 + exp(-x))^{-1}$; and the hyperbolic tangent, $\tanh(x) = 2\,\sigma(2x) - 1$. The default choice is usually the rectified linear units (ReLU). Glorot et al. (2011) identify sparsity as a key element of the success of ReLU activations. The exact zeros that can arise from this function allow a sparse representation, which is appealing from a computational point of view despite its non-differentiability at zero.

Figure 3 Common Activation Functions in Neural Networks



Finally, in the output layer, the features are then used to construct an estimator $\widehat{y}$ for the outcome variable y. In the case of a regression, the continuous output is directly constructed by a linear combination of the previous units

$$a^{[J]} = a^{[J-1]\prime} w_k^{[J]} + b_k^{[J]} \tag{7}$$

$$y = id(a^{[J]}) \tag{8}$$

with id denoting the identity function. In the case of a binary classification (0/1), instead of the identity, a logistic transformation is applied

$$P(y = 1|x) = \sigma\left(a^{[J]}\right). \tag{9}$$

This can easily be generalized to cases with more than two classes, say K_J classes. The output-layer then consists of K_J units, each describing the probability that the outcome falls into this category.

This type of output layer is usually modelled by a multinomial distribution.[2] To simplify notation, we will switch to matrix notation when we describe our model in the following sections. For every layer j, we then have

$$W_j = \begin{pmatrix} w_1^{[j]\prime} \\ \vdots \\ w_k^{[j]\prime} \\ \vdots \\ w_{K_j}^{[j]\prime} \end{pmatrix} \quad , \quad b_j = \begin{pmatrix} b_1^{[j]} \\ \vdots \\ b_k^{[j]} \\ \vdots \\ b_{K_j}^{[j]} \end{pmatrix} \tag{10}$$

with $W_j \in \mathbb{R}^{K_j \times K_{j-1}}$ and $b_j \in \mathbb{R}^{K_j}$.

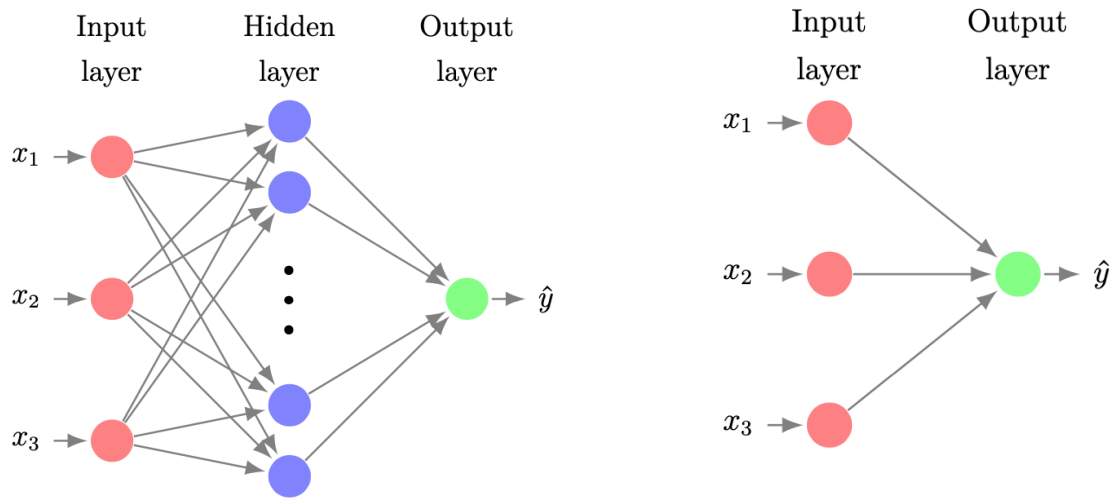
A standard neural network consists of a series of layers. For illustration, Figure 4a depicts a shallow (as opposed to deep) neural network. The units are connected to each other across layers but not within a layer. Early applications of such single hidden layer networks in econometrics are Kuan and White (1994), Giacomini et al. (2008), McAleer et al. (2008) and Medeiros et al. (2008). By using additional hidden layers and more units, the network can approximate more complex functional relationships (Hornik et al. 1989, Hornik 1991, Montúfar et al. 2014, Eldan and Shamir 2016, Telgarsky 2016). The additional hidden layers are created in the same way: each unit in a given layer is a nonlinear transformation of a linear combination of the units from the preceding layer. The depth of a neural network refers to the number of hidden layers. Figure 4b illustrates the structure of a logistic regression, which can be interpreted as a neural network without hidden layers. The unit in the output layer calculates a single linear index ($z = x'w + b$) and uses a sigmoid activation function, $\sigma(z) = \widehat{y}$.

Here we would like to mention briefly several general issues related to implementing deep neural networks. First, a neural network partly depends on a priori decisions of the researcher, which often cannot be theoretically motivated. For instance, prior to optimization, we have to choose the number of hidden layers, the number of units within a hidden layer, and the type of activation functions.[3] These decisions are the hyperparameters of a neural net. Usually, we consider a grid of values for the hyperparameters and pick the model that performs best. Second, a neural network often contains many parameters relative to the available sample size. As one can easily see, the number of parameters grows quickly as the number of units per layer and the number of layers increase. While a neural network with more units and more layers allows a better approximation of the unknown functional form, it is prone to overfit unique characteristics of the data that are

[2] Also known as softmax function in the machine learning literature. Strictly speaking $K_J - 1$ units are sufficient because the probabilities are supposed to sum up to one.

[3] The activation function in the output layer can generally be justified by theoretical considerations. In the binary classification task, the sigmoid function is a good choice because its output can be interpreted as probability.

Figure 4 Graphical Illustration of Neural Networks



(a) 2-layer neural network

(b) 1-layer NN ((Logit) regression)

not present in an additional, independent dataset. To avoid overfitting, we randomly split the data into three parts: a training sample, a validation sample for tuning hyperparameters, and a hold-out dataset for the final testing of our model. The hold-out sample should not be used to train the parameters or tune the hyperparameters. The latter could lead to indirect overfitting. Third, a challenge is how to estimate the parameters of a neural network. Measuring the discrepancy between observed values y and predicted values $\widehat{y}$ with a loss function, we must find the parameter vectors that minimize this loss function. The standard optimization algorithm is stochastic gradient descent (Goodfellow et al. 2016), which involves calculating the gradient of the loss function using only a subset of observations randomly chosen from the sample at each step. For this purpose, the training set is usually divided into mini-batches of 128 observations. The gradient is calculated via the backpropagation algorithm (Rumelhart et al. 1986). Additional hyperparameters, such as the learning rate, have to be set in the optimization. The learning rate determines how much the optimization adjusts the parameters in each iteration. The standard loss function for regressions or classifications is the mean squared error or the likelihood function, respectively. For a broader discussion of deep learning and neural networks, see Goodfellow et al. (2016).

5. Deep Learning for Categorical Data, Sequences and Highly Unbalanced Data

In the previous section, we discussed a standard feedforward neural network architecture. Our dataset has three characteristics that make it difficult to implement such networks: namely variables with very many categories, varying input length and highly unbalanced data. The methods we

present in this section to address these challenges may be of independent interest because these characteristics are common for many datasets in applied research.

5.1. Methods for Categorical Data

Generally, categorical variables cannot be used directly in neural networks. In this section, we describe methods for encoding variables that have a large number of categories. Many of these methods have been developed for text analysis because a word can be seen as a realization from a dictionary with very many entries. There are, however, many other applications in applied economics that must deal with high cardinality. The method we present here can, for instance, be used to embed ICD codes in health economics, occupational classifications in labor economics, or industry codes in industrial organization.[4]

The most common way to handle categorical variables is to expand them into dummy variables, an approach known as **one-hot encoding** in machine learning and as dummy encoding in the econometrics literature. If there are m categories (size of the dictionary), a unique integer index i is associated with every category and subsequently turned into a binary vector of length m. This vector has zero entries except for the ith entry, which is 1. Using the Kronecker delta, we can formalize one-hot encoding as a function u, which maps a categorical input variable x that has values $\{\alpha_1, \ldots, \alpha_m\}$ into a m-dimensional vector δ_α with entries $(\delta_{x,\alpha_k})_{k=1}^m$

$$u : x \mapsto \delta_\alpha \,, \tag{11}$$

The Kronecker delta is defined as 1 if $x = \alpha_k$ and zero otherwise. One-hot encoding provides a sparse representation. If there are many categories, one-hot vectors are high-dimensional and storing them requires a lot of memory. Moreover, the encoded vectors for different categories are orthogonal and hence cannot reflect similarities or other relations between them.

One-hot encoding can already be considered the first layer in a deep learning architecture. We, however, refrain from doing so because there are no weights in this "layer" so far. The idea of categorical embeddings is, essentially, to incorporate such weights, making it a complete layer that has a linear activation function.

Categorical embeddings are an alternative method to incorporate categorical variables in deep learning models (Guo and Berkhahn 2016). They take the concept behind word embeddings, which is to learn a real-valued representation in which words with similar meaning have a similar representation, and adapt this to general types of categorical variables. The embedding layer maps each category of a discrete variable (or each one-hot encoded vector) to a real-valued vector. Each categorical variable x_j is given its own embedding layer, which, in our application, is applied to

[4] For completeness, we additionally survey special methods to embed text data in Appendix A.1.

every item of a claim to transform the categorical variable into numerical variables. Denote the number of categories of x_j by m_j and the size of the embedding space by d_m. The dimension of the embedding space is usually smaller than the number of categories (or the length of the one-hot encoded vector), leading to a dimension reduction. The weights are tabulated in the embedding matrix ($W_E \in \mathbb{R}^{d_m \times m_j}$), which can be interpreted as a lookup table that maps every category k into a d_m-dimensional space. Starting with the one-hot encoded categorical vector δ_α of length m_j, the embedding function is given by

$$e : x_j \mapsto W_E \delta_\alpha \,, \tag{12}$$

which picks the kth column of W_E. One can either use pre-trained embedding matrices (Mikolov et al. 2013, for word embeddings) or learn the weights during the training process of the deep learning model. There are no task-specific embeddings available in our application and we, therefore, have to learn the $(d_m \cdot m_j)$ parameters of the embedding matrix as part of our task-specific model. This representation tends to put similar categories in the fraud process close to each other in the embedding space. Once the weights are learned, they can also be used as pre-trained embedding matrices with other datasets.

Categorical embeddings are especially useful for variables with high cardinality. When we one-hot encode such variables, the high cardinality leads to a huge set of parameters, which in turn can result in overfitting. Using an embedding layer rather than one-hot encoding reduces the number of parameters substantially and generally leads to better predictions, albeit at the expense of increased training time. If we suppose a variable has 1000 categories and the subsequent layer has 128 units, then one-hot encoding needs $1000 \times 128 = 128000$ parameters. In contrast, an embedding first maps the categories in a lower-dimensional space, say 30 dimensions. For this we need 30×1000 parameters and then just 128×30 parameters in the subsequent layer, i.e. overall 33840 parameters.

5.2. Methods for Sequences

Generally, machine learning methods require input data with fixed length. In this section, we describe methods to handle data with variable length. A very simple approach to make machine learning methods, such as boosted trees or random forests, work in situations with variable length is to fill up the shorter sequences with zeros so that all samples have the same length. This, however, may lead to suboptimal results. Another simple but widely used approach is the so-called bag-of-words. Let us assume we have a categorical variable and the number of times we observe this variable varies from observation to observation. Using a bag-of-words approach, one would determine which categories exist ("dictionary"), one-hot encode the input variables, and then aggregate the vectors by summing them up (i.e. counting how often a category/word is present

in an observation), for example, or taking the maximum over all entries (i.e. indicating whether a category or word is present at all). The result of this procedure is a fixed size vector (with length of the size of the dictionary) for each observation. Far more powerful procedures are available in the deep learning literature.[5]

Bahdanau et al. (2015) recently proposed the **attention mechanism** to find fixed-length and context-dependent representations. This approach allows dependencies between elements in a sequence to be taken into account independently of their distance. It has been successfully used to model context in machine translation (Bahdanau et al. 2015) or image captioning (Xu et al. 2015). The idea of the attention mechanism is to create a context vector that contains the most relevant information from a set of vectors by taking a weighted average, which is then used as input in the subsequent layer. The weights are determined in a data-driven way as described next.

The attention model has three ingredients (vectors): query and a key-value pair. The output is a weighted average of the value vectors, where the weights are determined by a similarity function (also called a compatibility or alignment function) between the query and the keys. Keys and values can be different vectors, but in many situations they are identical. More formally, we start with a query q, values $v = (v_1, \ldots, v_T)$ and keys $k = (k_1, \ldots, k_T)$. The output c is computed as

$$c = \sum_{t=1}^{T} \alpha_t v_t \,.$$

c is the context vector (also called an attention vector) and has a fixed size that is independent of the length of the input sequence. The weights are calculated using the softmax-function applied to a similarity function f between query and keys

$$\alpha_t = \frac{\exp f(k_t, q)}{\sum_{j=1}^{T} \exp f(k_j, q)} \,.$$

In the literature, different similiarity functions have been proposed, such as the (scaled) dot-product attention or additive attention (see, for instance, Bahdanau et al. 2015, Luong et al. 2015, Vaswani et al. 2017). The (scaled) dot-product attention is given by

$$f(k_t, q) = \frac{k_t' q}{\sqrt{d_k}} \,,$$

where d_k denotes the length of the key vectors. The scaling by the factor $\sqrt{d_k}$ is mainly done to achieve numerical stability, which is challenging for both attention and self-attention mechanisms and remains an important focus of ongoing research.[6]

[5] Most sequence classification models rely either on recurrent neural networks (Hochreiter and Schmidhuber 1997, Cho et al. 2014) or convolutional neural networks (LeCun 1989, Kim 2014). For a brief introduction to recurrent neural networks and convolutional neural networks, see Appendix A.2.

[6] In Appendix A.3 we also explain the self-attention mechanism, which is an extension of the attention mechanism and additionally allows a context-dependent interpretation of the features to be incorporated at the item-level. We observe, in our application, that self-attention does not perform distinctly better than the basic attention mechanism and, therefore, rely on the latter.

In our application, we use the attention mechanism to aggregate the varying claim length to a fixed-length vector by taking a weighted average over the items (see also Figure 5 in Section 6). For this, the keys and values are the features of the claim items and the query is a learned parameter vector. The output is a weighted average that represents all items of a claim, and the attention weights indicate the relevance of a particular claim item. They could potentially be used to understand and substantiate the predictions, which in turn would reveal valuable information for the subsequent claims management process.

5.3. Methods for Highly Unbalanced Data

A third characteristic of our fraud data is that they are highly unbalanced. This means that the number of fraudulent cases is usually much smaller than that of the non-fraudulent ones. In addition, there are a couple of large frauds, which are particularly worthwhile to catch. This poses problems for learning the fraud process. Undersampling and oversampling are typical approaches to working with highly unbalanced data, leading either to a decrease in the number of non-fraudulent cases or an increase in the number of fraudulent cases, respectively – and in this way to a more balanced proportion of fraudulent to non-fraudulent observations. An alternative approach to handling this problem is cost-sensitive training. The idea behind cost-sensitive training is to use tailor-made loss functions that incorporate the savings of detecting a fraudulent case and the costs of inspecting a non-fraudulent one. In our empirical application, we observe that cost-sensitive training considerably improves performance and leads to better out-of-sample decisions. A short introduction to cost-sensitive learning is given in Ling and Sheng (2010).

For a binary classification model, a logistic regression is usually used to estimate the probabilities. This leads to the following loss function, which is also called the binary cross entropy loss in machine learning:

$$-\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p(y_i)) + (1-y_i)\log(1-p(y_i))\right] .$$

y_i denotes the binary outcome in $\{0,1\}$, and $p(y_i)$ the estimated probability of the neural network for observation i given the learned parameters. In cost-sensitive learning, the loss function is scaled by the associated costs leading to:

$$-\frac{1}{N}\sum_{i=1}^{N}\left[(a_i-c)y_i \log(p(y_i)) + c(1-y_i)\log(1-p(y_i))\right] .$$

Here, $(a_i - c)$ measures the net benefits from identifying fraudulent claims, and c denotes the fixed costs of inspecting a claim. Using this weighted loss function leads to better results overall when dealing with unbalanced data because the fraudulent cases are weighted by their severity.

6. Deep Learning Model for Fraud Detection

In this section, we describe our model, which uses deep learning to process health care claims and identify suspicious claims automatically. We discuss the three main issues our deep learning model can address: the high cardinality of the categorical variables, the unknown patterns of the fraud process, and the relation between different items on a claim. Figure 5 depicts the general structure of our deep learning model for fraud detection.

In the **embedding layer**, we use the technique described in Section 5.1 to find a real-valued representation of the categorical variables. The output of the embeddings is then concatenated with the continuous variables. Let X denote the resulting $d_e \times T$ matrix, where T denotes the number of items of a particular claim (analogous to the number of words in a sentence) and d_e denotes the sum of the number of the continuous variables and the sizes of all embedding layers. X is now passed forward to the first feature extraction layer, which due to the real-valued representation of the categorical variables now allows us to model similarities flexibly between the different categories. This is the main strength of categorical embeddings.

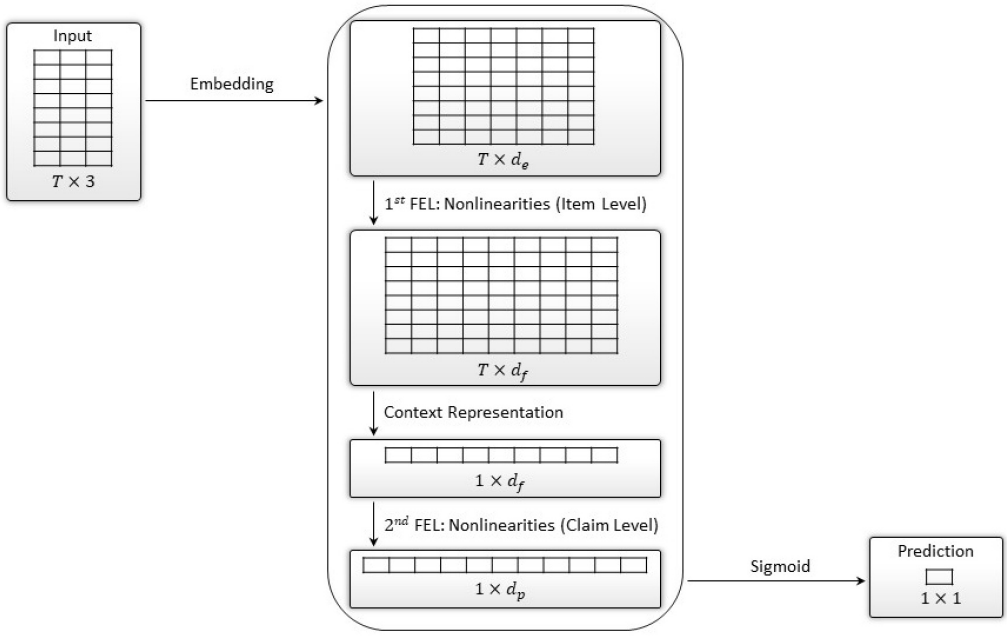
The **first feature extraction layer** (FEL) is responsible for capturing nonlinearities and interactions between parts of the inputs at the claim item level. We use a standard feedforward network here as described in Section 4. Every variable in the matrix X is turned into a set of features H_{J_1} by using J_1 sublayers with d_f units each and ReLU activation functions:

$$H_j = \text{relu}(W_{Hj} H_{j-1} + b_{Hj}) \quad \text{with } j = 1, \ldots, J_1 \text{ and } H_0 = X\,,$$
$$H_j \in \mathbb{R}^{d_f \times T},\ b_{Hj} \in \mathbb{R}^{d_f} \quad \forall\, j;\ W_{H1} \in \mathbb{R}^{d_f \times d_e};\ W_{Hj} \in \mathbb{R}^{d_f \times d_f} \text{ with } j = 2, \ldots J_1$$

It is important to observe that all weights of the feature extraction layer are independent of the varying claim length. This is a requisite for incorporating variable sequence lengths, and it even enables the model to handle claims of unseen lengths. However, to predict fraud probabilities, we must find one fixed-size vector for each claim.

This is the task of the **aggregation layer**, which takes as its input H the output of the feature extraction layer ($H = H_{J_1}$). It aims to combine the information contained in each item of a claim into a single (fixed-size) vector. Sum, mean, and max pooling for each feature are simple methods that either sum, take the average, or identify the maximum value over all items of a claim. These methods connect the claim items in a rather crude and inflexible way. However, the contribution of an item to identifying potential upcoding or fraud probably varies within a claim. The claim therefore represents a context that we aim to exploit in the aggregation. Many approaches have been proposed recently to find context-dependent representations in natural language interpretation. We use the attention mechanism described in Section 5.2 in the aggregation layer. Query, values and keys are derived as linear combinations of the input variables H, and $c = \sum_{t=1}^{T} \alpha_t v_t$ is the

Figure 5 Illustration of the Distinct Layers in our Deep Learning Model



Note: For better visibility, the matrix dimensions are transposed in this illustration.

resulting context vector of a particular claim. It has a fixed size $(d_f \times 1)$ that is independent of the length of this claim. In our application, we use sigmoid attention because we observe that it works better than softmax attention. In sigmoid attention, the weights need not sum up to one, which gives a slight nudge towards classifying longer sequences as suspicious cases. In the descriptive results, we have already seen that more comprehensive claims are fraudulent more often than less comprehensive ones (see Figure 2).

After collapsing the claim items into a single observation, we have one context vector of fixed dimension per claim and can link it to the corresponding label for this claim. This leaves us with what is essentially a "standard" machine learning problem. In this step, we could, for instance, use random forests (Breiman 2001), boosted trees (Breiman et al. 1984), or a feedforward network. We decide to set up a feedforward network in this last step to train the whole architecture, end to end. In this **second feature extraction layer** we capture nonlinearities and interactions at the claim level using J_2 sublayers with d_p units each and ReLU activations, and finally obtain the scalar predictions P from a sigmoid activation in the **output layer**:

$$
\begin{aligned}
c_j &= \text{relu}(W_{cj} c_{j-1} + b_{cj}) \quad \text{with } j = 1, \ldots, J_2 \text{ and } c_0 = c\,, \\
P &= \sigma(w' c_{J_2} + b)\,, \\
c_j \in \mathbb{R}^{d_p}\,,\ b_{cj} \in \mathbb{R}^{d_p} \ \ \forall\, j\,;\ W_{c1} \in \mathbb{R}^{d_p \times d_f}\,;\ W_{cj} \in \mathbb{R}^{d_p \times d_p} &\text{ with } j = 2, \ldots, J_2\,;\ w \in \mathbb{R}^{d_p}\,;\ b \in \mathbb{R}\,.
\end{aligned}
$$

We apply this model to the claims data described in Section 3. We use a validation dataset for model selection and an independent dataset for testing, each containing 20% of the claims. To

train our deep learning model, we use the remaining 60% of the sample. We use dropout, which artificially corrupts the training data, to control for overfitting (Wager et al. 2013). Additionally, weight decay prevents overfitting by adding an L_2-regularization to the training criterion. We select the dropouts and the weight decay parameters by random search. The number of hidden units, batch sizes and the number of sublayers for the nonlinearities are chosen from a grid of values. The grid values and the hyperparameters ultimately chosen are displayed in Table 1. The size of the embedding space for the categorical variable is set to the square root of the number of categories (rounded integer), i.e. $d_m = r(\sqrt{m})$. We implement the model in PyTorch and use the Adam optimizer (Kingma and Ba 2014) with learning rate $1e$-04 and early stopping.

Table 1 Hyperparameter Selection

Batch Size	32	64	128	**128**
Hidden Units (d_f & d_p)	64	128	256	**128**
# of Sublayers (1^{st} FEL)	1–5			**1**
# of Sublayers (2^{nd} FEL)	1–5			**4**
Weight Decay	$1e$-04–$1e$-05			**3e-05**
Dropouts	0–0.5			**0.164**

Note: Bold values are the chosen hyperparameters, which performed best in the validation sample. FEL stands for Feature Extraction Layer.

7. Empirical Evaluation

The success of automatically screening incoming claims crucially depends on how well our model can handle new claims. Based on the N unseen claims in the test dataset, we determine the extent to which our screening, $I(P_i > \tau)$, aligns with the true fraud labels. We compare three methods to model P_i. First, we consider a random assignment of claims to manual auditing as a weak benchmark.[7] Second, we use boosted trees (BT) with one-hot encoded categorical variables as a competing machine learning method. Finally, we use a classification derived from our deep learning model.

To make all three methods comparable, a constant number of claims are flagged up by each method and assigned to manual auditing. This analysis can be motivated with capacity constraints in the auditing unit. Let the number of claims, which the auditing unit can handle, be denoted by N_τ. Every model m then needs a specific threshold τ_m such that a constant number $N_\tau = \sum_{i=1}^{N} I(P_i^m > \tau_m)$ of suspicious claims are assigned to manual auditing. This way we interpret P_i as a ranking and assign the claims that are most suspicious according to the respective screening model first. The quality of the automatic screening at any N_τ depends on how well the model can classify the claims (i.e. separate fraudulent ones from non-fraudulent ones), and on how well it can prioritize or rank the suspicious claims (i.e. finding claims with high adjustments first). Figure 6

[7] For the random assignment, we draw P_i from a uniform distribution between 0 and 1.

shows the Lorenz curve for the cumulative adjustments of the fraudulent claims. Detecting the top 1% of fraud claims would already yield more than 20% of the total adjustments, which underscores that prioritization is of crucial importance in our application.

Figure 6 Lorenz Curve of Adjustments

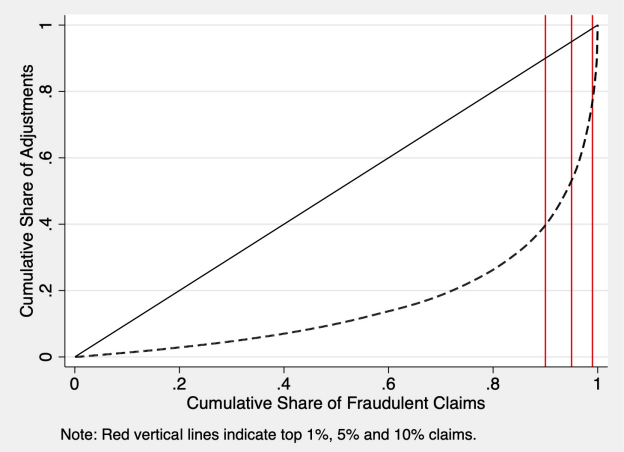
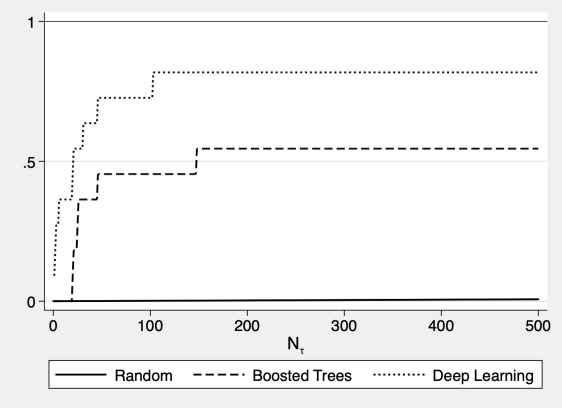


Figure 7 shows how often we detect the top 1, 5, or 10% of fraudulent claims for different values of N_τ. Prioritization based on our deep learning model performs better for all of these metrics. Figure 7 also shows that the identification of fraudulent claims levels out after $N_\tau \approx 300$. While this may indicate that we are missing some systematic frauds with our models, it could also be caused in part by random errors, such as unintentional upcoding, which we cannot separately identify in our labelled data and are generally hard to model due to their random nature. Randomly assigning claims to manual auditing is unable to detect a non-negligible fraction of the top frauds due to the low probability of fraudulent claims in our data in combination with the small set of top frauds we are looking for.

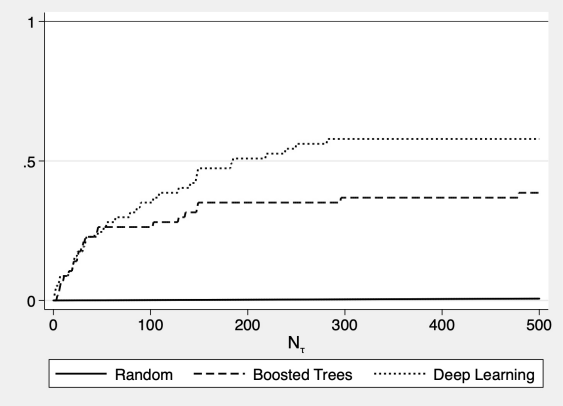
Suppose now that the claims are ranked based on their adjustment value such that $i = 1$ refers to the claim with the largest adjustment. The foregone cost savings of model m at every N_τ are then defined as

$$\begin{aligned}\gamma_{N_\tau} = \pi^{\max}_{N_\tau} - \pi^{m}_{N_\tau} &= \sum_{i=1}^{N_\tau} a_i - \sum_{i=1}^{N} I(P_i^m > \tau_m) a_i \\ &= \sum_{i=1}^{N_\tau} I(P_i^m \leq \tau_m) a_i + \sum_{i=1}^{N_\tau} I(P_i^m > \tau_m) a_i - \sum_{i=1}^{N_\tau} I(P_i^m > \tau_m) a_i - \sum_{i=N_\tau+1}^{N} I(P_i^m > \tau_m) a_i \\ &= \sum_{i=1}^{N_\tau} I(P_i^m \leq \tau_m) a_i - \sum_{i=N_\tau+1}^{N} I(P_i^m > \tau_m) a_i \,,\end{aligned}$$

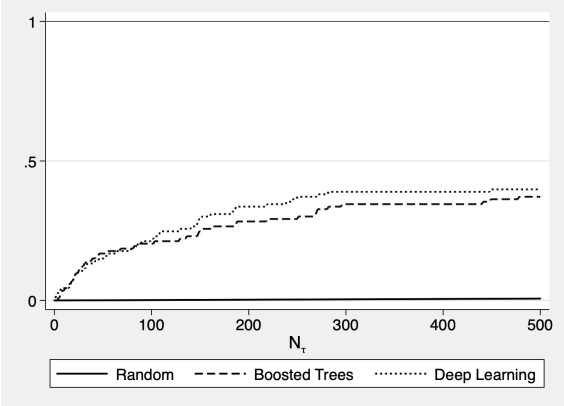
Figure 7 Fraction of Top Adjustments in Selected Set



(a) Top 1%, i.e. 11 fraudulent claims



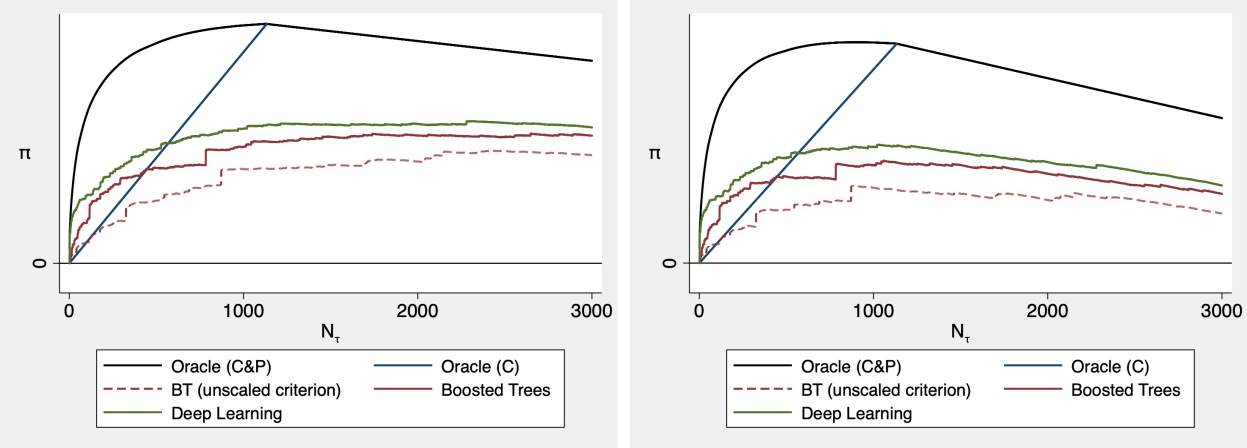
(b) Top 5%, i.e. 57 fraudulent claims



(c) Top 10%, i.e. 113 fraudulent claims

where we used the fact that constant auditing costs per claim cancel out. The foregone cost savings comprise the losses from missing some of the top N_τ frauds and the cost reductions we yield by auditing less attractive claims instead, which includes auditing false positives ($a_i = 0$). Figure 8 depicts the cost savings ($\pi^m_{N_\tau}$) if we use deep learning (green line) or boosted trees (red line) to identify suspicious claims automatically. In Figure 8(a), we use a constant cost parameter of $\bar{c}$. Only 0.4% of the fraudulent claims offer an adjustment below $\bar{c}$. Figure 8(b) shows the results for a cost parameter that is twice as large. 17.8% of the fraudulent claims have an adjustment below $2\bar{c}$. The expected cost savings from randomly assigning claims to manual auditing are always negative because doing so leads to many costly false positives. We therefore do not illustrate them in Figure 8. Instead we show the expected gains from an oracle that randomly draws from the subset of fraudulent claims (blue line). Assignments based on this oracle (C) benefit only from perfect classification of the fraudulent claims but completely fail with respect to the prioritization task. When we use the cost-sensitive loss, then both model-based assignments perform better than this oracle at the beginning of the path, which suggests that both models pay attention to the

Figure 8 Cost Savings for Boosted Trees and Deep Learning



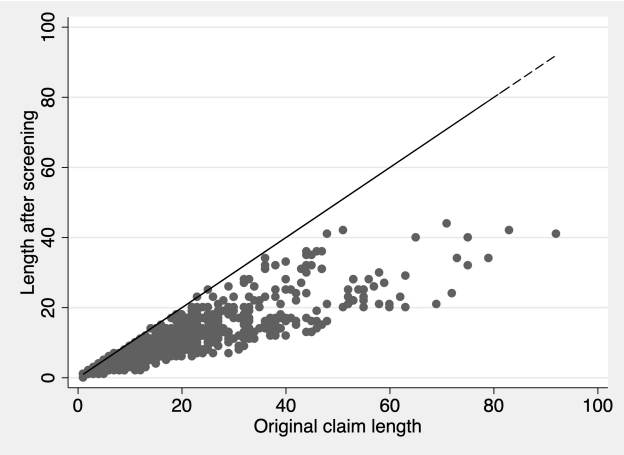
(a) Auditing costs $c = \bar{c}$ (b) Auditing costs $c = 2\bar{c}$

prioritization task. As reference we also add the results of boosted trees without cost-sensitive loss, which clearly fails in the prioritization. Oracle (C&P) benefits from both perfect classification and perfect prioritization of the fraudulent claims, which leads to $\pi_{N_\tau}^{\max}$ depicted as black line in Figure 8. The forgone savings are the differences in π between this oracle and the model-based assignments. Our deep learning model clearly reduces the foregone savings compared to boosted trees for any value of N_τ.

Neural attention has been a cornerstone in the development of many modern deep learning models. In our application, it not only increases the quality of predictions but also adds, with the estimation of the attention distribution, a potentially informative tool to the decision-support system. The attention weights shed light on the relevant claim items and may, additionally, provide meaningful explanations for the predictions. On average, a selection based on the attention weights being larger than 0.5 reduces the number of relevant claim items roughly by half. Figure 9 shows a scatter plot of the original claim length and the length after screening.

The auditing unit will particularly benefit from the attention weights if these provide meaningful explanations, because then the weights can help elucidate which claim items are especially relevant for classifying a claim as being suspicious. Clearly, such information would speed up the subsequent (manual) auditing process. Therefore, meaningful explanations would transform our model from a tool that is merely predictive to one that is prescriptive. An open research question, however, is the extent to which the obtained attention weights deliver meaningful insights into the relative importance of the inputs. Recent studies document mixed evidence on the reliability of the explanations provided by attention weights (Jain and Wallace 2019, Jain et al. 2019, Vig and Belinkov 2019, Vashishth et al. 2019). In a Monte Carlo simulation, we show that the attention weights can indeed recover the invalid claim items in a stylized setting (see Appendix B). Future

Figure 9 Claim Length Reduction using Attention Weights (fraudulent claims only)



research could explore whether it would be possible to use item-based labels in a real application to verify the selection based on the learned attention weights.

8. Conclusion

We outline basic similarities between the way that texts and health insurance claims are structured, and how these similarities allow deep learning techniques from text classification to be extended to the task of classifying claims. Claims management is a promising application for these methods outside of natural language processing or image analysis, and has real and quantifiable economic value. Methods based on neural networks can improve upon or even replace hand-designed features.

We develop a deep neural network that can be used to detect fraudulent claims automatically. It not only classifies claims but also delivers potentially meaningful explanations. Our model can handle data with inputs of varying length and variables with a large number of categories. As the digitalization of many aspects of our lives continues, very large amounts of unstructured data with characteristics similar to ours will become available, and it is therefore important for applied researchers to have such methods at their disposal.

Our proposed architecture can be used to mitigate informational asymmetries and accelerate subsequent auditing decisions. It outperforms default models on a large dataset from a private insurance company in Germany. A more comprehensive model can be built in a straightforward manner by incorporating additional input variables, such as information about health care providers or patients' medical histories. The empirical results substantiate the performance of deep learning as a data-driven tool in predictive analytics in claims management. Moreover, we demonstrate that our deep learning model not only adds another red flag but also gives additional insights into the fraud mechanisms themselves. We conduct a simulation study to assess their utility as a tool in prescriptive analytics. The simulations provide preliminary evidence that an attention model is able to deliver meaningful explanations of its predictions in claims management.

References

Bahdanau D, Cho K, Bengio Y (2015) Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015*.

Bastani H, Goh J, Bayati M (2019) Evidence of upcoding in pay-for-performance programs. *Management Science* 65(3):1042–1060.

Becker D, Kessler D, McClellan M (2005) Detecting medicare abuse. *Journal of Health Economics* 24(1):189–210.

Bolton RJ, Hand DJ (2002) Statistical fraud detection: A review. *Statistical Science* 17(3):235–249.

Bond EW, Crocker KJ (1997) Hardball and the soft touch: the economics of optimal insurance contracts with costly state verification and endogenous monitoring costs. *Journal of Public Economics* 63(2):239–264.

Breiman L (2001) Random forests. *Machine learning* 45(1):5–32.

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) *Classification and regression trees* (Wadsworth).

Cecchini M, Aytug H, Koehler GJ, Pathak P (2010) Detecting management fraud in public companies. *Management Science* 56(7):1146–1160.

Cho K, van Merriënboer B, Bahdanau D, Bengio Y (2014) On the properties of neural machine translation: Encoder–decoder approaches. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 103–111.

Devlin J, Chang MW, Lee K, Toutanova K (2019) Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT* 41714186.

Dionne G, Giuliano F, Picard P (2009) Optimal auditing with scoring: Theory and application to insurance fraud. *Management Science* 55(1):58–70.

Ekin T, Ieva F, Ruggeri F, Soyer R (2018) Statistical medical fraud assessment: Exposition to an emerging field. *International Statistical Review* 86(3):379–402.

Eldan R, Shamir O (2016) The power of depth for feedforward neural networks. *29^{th} Annual Conference on Learning Theory*, 907–940.

Fang H, Gong Q (2017) Detecting potential overbilling in medicare reimbursement via hours worked. *American Economic Review* 107(2):562–91.

Giacomini R, Gottschling A, Haefke C, White H (2008) Mixtures of t-distributions for finance and forecasting. *Journal of Econometrics* 144(1):175–192.

Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. *Proceedings of the 14^{th} International Conference on Artificial Intelligence and Statistics*, 315–323.

Goodfellow I, Bengio Y, Courville A (2016) *Deep learning* (MIT press).

Guo C, Berkhahn F (2016) Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737* .

Heese J (2018) The role of overbilling in hospitals earnings management decisions. *European Accounting Review* 27(5):875–900.

Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural Computation* 9(8):1735–1780.

Hornik K (1991) Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4(2):251–257.

Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* 2(5):359–366.

Jain S, Mohammadi R, Wallace BC (2019) An analysis of attention over clinical notes for predictive tasks. *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, 15–21.

Jain S, Wallace BC (2019) Attention is not explanation. *arXiv preprint arXiv:1902.10186* .

Kim Y (2014) Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1746–1751.

Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Kuan CM, White H (1994) Artificial neural networks: An econometric perspective. *Econometric Reviews* 13(1):1–91.

Lai S, Xu L, Liu K, Zhao J (2015) Recurrent convolutional neural networks for text classification. *Twenty-ninth AAAI conference on artificial intelligence*.

LeCun Y (1989) Generalization and network design strategies. Pfeifer R, Schreter Z, Fogelman-Soulié F, Steels L, eds., *Connectionism in Perspective*, 143–155 (Elsevier Amsterdam).

Li J, Huang KY, Jin J, Shi J (2008) A survey on statistical methods for health care fraud detection. *Health Care Management Science* 11(3):275–287.

Ling CX, Sheng VS (2010) Cost-sensitive learning. Sammut C, Webb GI, eds., *Encyclopedia of Machine Learning*, 231–235 (Boston, MA: Springer US).

Liou FM, Tang YC, Chen JY (2008) Detecting hospital fraud and claim abuse through diabetic outpatient services. *Health Care Management Science* 11(4):353–358.

Luong T, Pham H, Manning CD (2015) Effective approaches to attention-based neural machine translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1412–1421.

McAleer M, Medeiros MC, Slottje D (2008) A neural network demand system with heteroskedastic errors. *Journal of Econometrics* 147(2):359–371.

Medeiros MC, McAleer M, Slottje D, Ramos V, Rey-Maquieira J (2008) An alternative approach to estimating demand: Neural network regression with conditional volatility for high frequency air passenger arrivals. *Journal of Econometrics* 147(2):372–383.

Mikolov T, Chen K, Corrado GS, Dean J (2013) Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* .

Montúfar GF, Pascanu R, Cho K, Bengio Y (2014) On the number of linear regions of deep neural networks. *Advances in neural information processing systems 27*, 2924–2932.

Mookherjee D, Png I (1989) Optimal auditing, insurance, and redistribution. *Quarterly Journal of Economics* 104(2):399–415.

Picard P (1996) Auditing claims in the insurance market with fraud: The credibility issue. *Journal of Public Economics* 63(1):27–56.

Picard P (2013) Economic analysis of insurance fraud. *Handbook of Insurance*, 349–395 (Springer).

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. *Nature* 323(6088):533–536.

Schiller J (2006) The impact of insurance fraud detection systems. *Journal of Risk and Insurance* 73(3):421–438.

Telgarsky M (2016) Benefits of depth in neural networks. *29^{th} Annual Conference on Learning Theory*, 1517–1539.

Tennyson S, Salsas-Forn P (2002) Claims auditing in automobile insurance: Fraud detection and deterrence objectives. *Journal of Risk and Insurance* 69(3):289–308.

Townsend RM (1979) Optimal contracts and competitive markets with costly state verification. *Journal of Economic Theory* 21(2):265–293.

Urbanovich E, Young EE, Puterman ML, Fattedad SO (2003) Early detection of high-risk claims at the workers' compensation board of british columbia. *Interfaces* 33(4):15–26.

Van Vlasselaer V, Eliassi-Rad T, Akoglu L, Snoeck M, Baesens B (2017) Gotcha! Network-based fraud detection for social security fraud. *Management Science* 63(9):3090–3110.

Vashishth S, Upadhyay S, Tomar GS, Faruqui M (2019) Attention interpretability across NLP tasks. *arXiv preprint arXiv:1909.11218* .

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. *Advances in Neural Information Processing Systems*, volume 30, 5998–6008.

Viaene S, Derrig RA, Baesens B, Dedene G (2002) A comparison of state-of-the-art classification techniques for expert automobile insurance claim fraud detection. *Journal of Risk and Insurance* 69(3):373–421.

Vig J, Belinkov Y (2019) Analyzing the structure of attention in a transformer language model. *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 63–76.

Wager S, Wang S, Liang PS (2013) Dropout training as adaptive regularization. *Advances in neural information processing systems*, volume 26, 351–359.

Xu K, Ba J, Kiros R, Cho K, Courville A, Salakhudinov R, Zemel R, Bengio Y (2015) Show, attend and tell: Neural image caption generation with visual attention. *Proceedings of the 32^{nd} International Conference on Machine Learning*, 2048–2057.

A. Appendix

A.1. Methods for Categorical Variables and Text

In this Appendix, we additionally outline methods for processing text data or variables that have a large number of categories. Word embeddings represent a major breakthrough in analysing texts using machine learning methods.

Illustration of Embeddings: For illustration, we focus on words and describe so-called word embeddings. This principle can be applied, however, to any categorical variable. The idea of embeddings is to represent a certain category by a real-valued vector. The high-dimensional one-hot encoded vector is embedded in a low-dimensional space. A naive approach to construct such an embedding would be to associate a random dense vector with each word. But this embedding would be unstructured and make it hard to train deep neural networks. Indeed, it is desirable for the embedding space to maintain some structure. For instance, words with similar meaning (synonyms) should be represented by "similar" vectors. More generally, the geometric relationships between word vectors should reflect the semantic relationships between these words. Moreover, directions should be interpretable as well. For instance, the difference in the embedding space between the words man and woman should be the same as between king and queen so as to reflect the gender difference in both cases. Embeddings should incorporate the semantic information about the words they represent. Word embeddings are a powerful concept. A key contribution was the development of word2vec (Mikolov et al. 2013), which consists of continuous bag-of-words and skip-gram (both will be explained below). Although embeddings were originally developed for words and documents, these methods can also be used for categorical variables with a large number of categories.

Continuous Bag-of-Words Model: Text is partitioned into so-called bag-of-words (sets of words showing up jointly in texts), and the prediction for a certain word is based on its context words, i.e. the other words in this text. The simplest version of the continuous bag-of-words model (CBOW) is the "single context word". The idea is to predict one target word given one context word. In the general version, more than one context word is used to predict the target word. The basic model is given in Figure A.1.

The input vector $x = (x_1, \ldots, x_V)$ is one-hot encoded where V denotes the size of the dictionary. The hidden layer is created by a linear transformation with a weight matrix W of dimension $V \times N$. N will be the dimension of the constructed embedding space. The hidden values are constructed as $h = xW = W_{k,\cdot}$, where $W_{k,\cdot}$ denotes the kth row of W. From the hidden layer to the output layer, there is a different weight matrix W' of dimension $N \times V$. A score for each word in the dictionary is given by the corresponding entry in the score vector u:

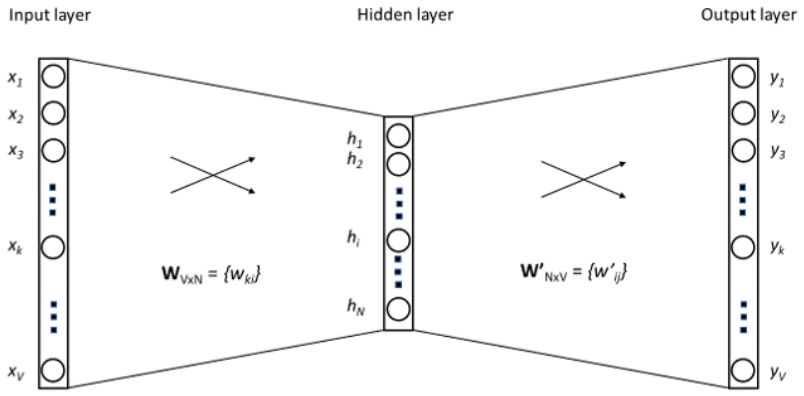
$$u = hW'.$$

Finally, we can use a softmax function to obtain a distribution of the words that tells us how likely each word in the dictionary is the corresponding target word:

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{k=1}^{V} \exp(u_k)} .$$

$p(w_j|w_I)$ is the probability that word w_j is associated with the input word w_I. The coefficient matrices W and W' are then estimated by (text) data. The rows of the matrix W represent the embedding for the corresponding word.

Figure A.1 Continuous Bag-of-Words Model

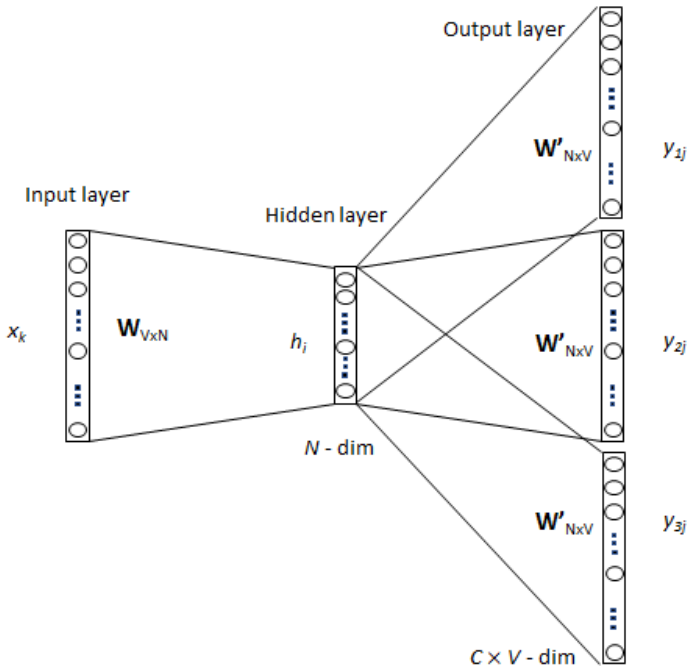


Skip-Gram Model: This can be regarded as the opposite of the CBOW. While context is used to predict a word when using the CBOW approach, in Skip-Gram the context words are predicted from an input word. The underlying structure is given in Figure A.2. The input layer is constructed exactly as in the CBOW model and given by

$$h = W_{k,\cdot} = v_{W_I} .$$

There is now not just one but C multinomial distributions at the output layer. Here each distribution is governed by the same weight matrix W'. The probability that the cth context word is $w_{O,c}$ given the input word w_I is given by $p(w_{c,j} = w_{O,c}|w_I) = y_{c,j} = \frac{\exp(u_{c,j})}{\sum_{k=1}^{V} \exp(u_{c,k})}$. The loss function is then given by the product of the probabilities of a certain context C.

Figure A.2 Skip-Gram Model



A.2. Methods for Sequential Data

In this Appendix, we will briefly discuss standard methods for handling sequence data. Text consists of sentences of different lengths. This is also the situation we have with claims data. Each claims consists of a different number of items. To handle inputs of different length in deep learning, sequence models have been developed.

Convolution Neural Networks (CNNs): These are very popular for image analysis (LeCun 1989), but can also be used to analyze sequences. CNNs usually combine a convolution layer and a pooling layer sequentially to extract useful feature information. The idea behind the convolution layer is to extract local information and form more complex structures in a hierarchical way from low-level features. The *convolution layer* is a linear operation followed by a nonlinear transformation. Let us assume that we have a sequence of variables of length s where each variable has many categories. Standard examples are sentences or time series. We assume that the variables have been embedded in a space of dimension d such that the input dimension is $d \times s$, leading to an input matrix $Y \in \mathbb{R}^{d \times s}$. Next we define a window size h and apply a convolution operation on a rolling window of this size. For this, we define a filter, a matrix of weights in $\mathbb{R}^{h \times s}$, which has to be learned during the training process. The ith output element is then given by

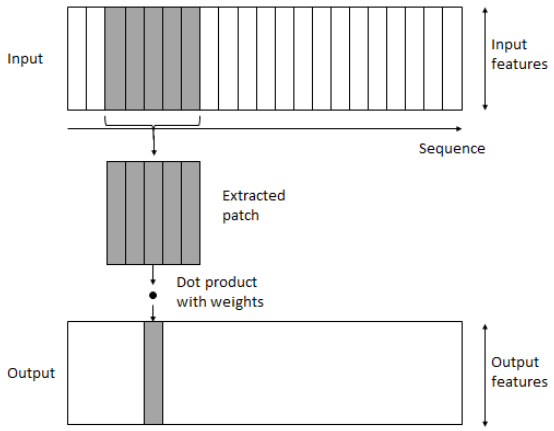
$$o_i = W * Y[i : i + h - 1,].$$

$Y[i:i+h-1,]$ is simply a submatrix of Y and $*$ returns the sum of the elementwise product. The principle is also summarized in Figure A.3. Finally, a nonlinear activation function is applied:

$$c_i = f(o_i) + b\,.$$

The output vector c is of length $s-h+1$, and b is an intercept. It is possible to apply not only one but several filters at the same time. The weights and intercepts have to be estimated by the data.

Figure A.3 Convolution Step



In many situations, the exact position of the features in the input document does not matter, but rather that they are present. To enable such behavior, global k-max pooling is employed in the *pooling layer*. This means that the k largest values from each feature map are extracted and concatenated. This gives a final vector whose size remains constant during training. Instead of maximum, other operations like the mean can be used.

After the convolutional and pooling layers, a feedforward network can be used to extract further information until the output is generated, which is done for a categorical variable via a softmax activation or for a regression via a linear combination of the features.

Recurrent Neural Networks (RNNs): While CNNs work well with regular sequences and are restricted to local patterns, they cannot capture long-term dependencies. To capture such behavior, RNNs have been developed. We present the basic idea behind RNNs in the following. In practice, it is mainly two extensions of RNNs that are used, namely long short-term memory (LSTM, see Hochreiter and Schmidhuber 1997) and gated recurrent units (GRU, see Cho et al. 2014).

Given an (ordered) input sequence $\{x_1, \ldots, x_T\}$, an ordered output sequence $\{y_1, \ldots, y_T\}$ is generated. Additionally, we have a sequence $\{h_0, h_1, \ldots, h_T\}$ of hidden states (vectors), where the first hidden state is initialized as a zero vector. Each output y_t depends now not only on the corresponding input x_t but also on the previous hidden state h_{t-1}. This is illustrated in Figure A.4. The hidden states can be regarded as a kind of "short-term" memory that summarizes the previous states. More formally, the hidden states evolve as

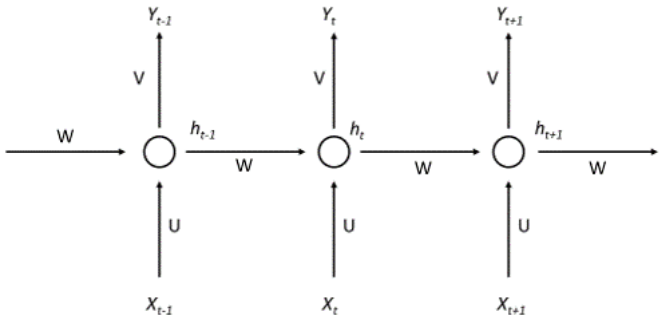
$$h_t = f(Ux_t + Wh_{t-1} + b)\,,$$

f is a nonlinear activation function applied elementwise. U, W are weight matrices and b the intercepts, which are estimated during the training process. x_t and h_t are vectors that can have different dimensions. The final output (say, categorical) is generated as

$$y_t = softmax(Vh_t)\,,$$

where V again denotes a weight matrix to be estimated. The matrices U, V and W are shared over all time steps.

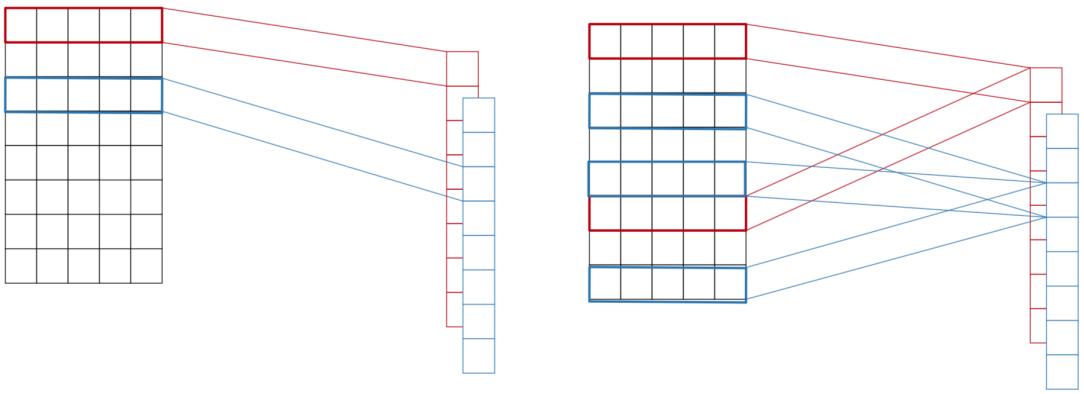
Figure A.4 Recurrent Neural Net



A.3. Attention Models

Attention and self-attention are important recent innovations in deep learning. Colloquially speaking, to pay attention to something means to direct ones focus at or take greater notice of it. As an action, the verbal phrase also describes how humans capture the content of images or the meaning of a text. While originally introduced for tasks such as machine translation, image recognition and natural language processing, attention and self-attention are now widely used in other domains, as well. In this section, we give an intuitive introduction to these mechanisms because they are useful for fraud detection and may be applied in economics and finance in the future. We use the attention mechanism to aggregate the information in the claims, which are of varying dimensions, to fixed sized vectors that can then be processed further. Other methods for aggregation are max-pooling or summation, but these operations connect the claim items in a rather crude and inflexible way. Additionally, extracting features at the level of individual items may be a promising application of the self-attention mechanism in our fraud detection task. In some situations, like in text analysis, it is useful to take into account the location and dependencies of the input features in the feature extraction process. For example, in our fraud application, the simultaneous presence of two claim items might be predictive of a fraud case, but where these items are listed on the bill might be arbitrary. Figure A.5 compares self-attention with a standard feedforward feature extractor. Whereas a feature of an input element depends, in feedforward networks, only on its input, it depends in self-attention networks both on itself and any other elements of the sequence.

Figure A.5 Comparison between standard feedforward and self-attention feature extractors.



(a) Standard feedforward (b) Self-attention

Self-attention models apply the attention mechanism to every position of an input sequence. First, for each position of the input vector, the three vectors query, key, and value are created. Next, the attention mechanism outlined above is applied to every position. Assume our input vector is

$x = (x_1, \ldots, x_p)$. Then an output vector $y = (y_1, \ldots, y_p)$ is constructed in the following way: For a position x_i (query), x serves both as a key and value.

Next, we show how self-attention might be used for feature extraction allowing for dependence between different variables. For each observation, the sequence of inputs represented by matrix X is turned into a sequence of features H e.g. by applying scaled dot-product self-attention. This was proposed in Vaswani et al. (2017), whom we follow here. Queries (Q), keys (K), and values (V) are derived from a linear transformations of our inputs – via matrix multiplication with parameters and adding a bias, as follows:

$$Q = W_q X + b_q$$
$$K = W_K X + b_k$$
$$V = W_V X + b_v$$
$$W_Q, W_K, W_V \in \mathbb{R}^{d_m \times T},\ b_q, b_k, b_v \in \mathbb{R}^{d_m},\ Q, K, V \in \mathbb{R}^{d_m \times T}.$$

Second, we use the scaled dot-product attention operation to turn the queries, keys, and values into a sequence of features H:

$$H = softmax(\tfrac{QK^T}{\sqrt{d_m}})V\,,$$
$$H \in \mathbb{R}^{d_m \times T}\,.$$

The features derived from self-attention are then context-dependent as well – that is, we can see which other elements in the input sequence are important for interpreting the features of a particular item.

We also used a self-attention network for feature extraction in our application but observed that it did not perform distinctly better. A challenge with self-attention are normalizations and numerical problems, both of which are the subject of ongoing research.

B. Interpretability of Attention Weights – Simulation Results

An open research question is the extent to which the attention weights obtained in the aggregation layer deliver meaningful insights into the relative importance of the inputs. For this purpose, we perform a simulation study that allows us to compare the estimated attention weights with the *known* causal reasons for a suspicious claim. Some recent studies have analyzed the interpretability of attention weights for particular applications (for instance, Jain and Wallace 2019, Jain et al. 2019, Vig and Belinkov 2019, Vashishth et al. 2019). With our simulations in the area of health insurance claims management, we contribute to this strand of the literature.

We generate artificial claims data as a sequence of two categorical variables per claim item. Each of the two variables contains 20 categories. We draw the categories from a uniform distribution. Each claim consists of up to 100 claim items. This results in a data structure similar to the claims data in our empirical application. Our simulated dataset contains $100,000$ artificial claims. We label a claim as suspicious following one or multiple invalid claim combinations. We let the invalidity comprise multiple claim items and depend on a combination of the categorical variables because this is how we expect suspicious cases to occur in the real world. To make the simulations more complex, we relabel 10% of the frauds as non-fraudulent despite having an invalid item combination. Because we artificially generate the claims data, we know the cause of each suspicious label and can therefore verify the extent to which our model is able to recover these sequence elements.

We use different sets of fraud conditions to generate invalid combinations of claim items. In the first setup, we generate a suspicious case if the following two conditions jointly hold: The claim contains an item with the constellation $(19, 1)$ and an item with the constellation $(17, 8)$. This way, the models must capture both the interactions of variables from a single item and the interactions between different items. In the other setups, we include multiples of the conditions discussed above with various combinations of categories – whereby the occurrence of one of these causes the claim to be suspicious in the simulated data.

We say that a prediction has been correctly explained if the model puts high (attention) weights on the correct invalid claim items. To evaluate the quality of the explanations in our simulations, we say that a claim item is the explanation of a suspicious prediction if the attention weight is larger than some threshold. We set the threshold at 0.5, but it turns out to be irrelevant as the attention weights become saturated at zero and one after some training time.

Obtaining meaningful explanations is a core task of prescriptive analytics. We use the following measures to evaluate the extent to which our deep learning models can explain their predictions:

1.) How many times did the model recover all relevant invalid items (variable screening)?

2.) And, subsequently, upon how many additional (valid) items did the model place weight? Here we take the average and maximum number of claim items flagged as suspicious.

Table B.1 Simulation results (Explainable attention network).

Conditions	Share	Screening	Average	Max
1	0.016	1	2.00	2
2	0.032	1	2.31	4
5	0.074	1	3.05	7
10	0.138	1	4.72	11

SHARE denotes the fraction of fraudulent claims.

After generating the data, we split it into training and test datasets. We train our deep learning model on the training data and assess the quality of the results using the test data. We first note that our model can easily achieve perfect prediction performance, meaning that it can perfectly separate between fraudulent and non-fraudulent cases. If the machine learning model cannot deliver meaningful attention weights in our simplified setting, there is little hope that it will deliver them in more realistic settings, where a multitude of invalid claim combinations exist and systematic fraud or upcoding may be covered by more noise.

In general, we find that the explanations of our model are very good (see Table B.1). This model places most of the attention mass on the claim items that are the reasons for the invalidity of the claim. Moreover, the model reduces the number of relevant claim items considerably. Employing the information in the attention weights, we can reduce the number of problematic items to 3.05 on average in the setup with five conditions and to 4.72 in the setup with ten conditions. The worst-case claims contain 7 and 11 potentially problematic items, respectively. This is a successful reduction of claim complexity considering that the average number of claim items is around 50 in our simulations. These results suggest that our model delivers meaningful explanations. Exploiting these in the subsequent management process has great potential to speed up the manual auditing decisions.

An interesting result is that the maximum number of marked items reflects the number of claim items that contribute to a potential fraud mechanism. Here we see a potential weakness of our model: The representations of the inputs are context-independent. As a result, we will always mark a claim item as relevant if it is part of a potential fraud combination (even if the other item in this combination is not part of the claim). To illustrate, imagine we have two fraud conditions, either [(19,1) and (17,8)] or [(15,2) and (13,7)] in the same claim. Now, the cases where the model marks too many inputs as relevant are cases where, for example (19,1), (17,8) and (15,2), are part of the same claim. This highlights a potential advantage of the self-attention model: Because the input representations are context-dependent, it is theoretically possible to solve this problem.