UiO : University of Oslo

Håvard Kvamme

Time-to-Event Prediction with Neural Networks

Thesis submitted for the degree of Philosophiae Doctor

Department of Mathematics
Faculty of Mathematics and Natural Sciences

2019

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 2346

ISSN 1501-7710

Cover: Hanne Baadsgaard Utigard.
Print production: Reprosentralen, University of Oslo.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of *Philosophiae Doctor* at the University of Oslo. The thesis is a collection of four papers with the common theme of time-to-event prediction with neural networks. Their chronological order is representative of my understanding of the field. Although I have a background in statistics from the Norwegian University of Science and Technology, I have always found the empirical mindset of machine learning to be closer to my way of thinking. It has, therefore, been interesting to work at the intersection of statistical survival analysis and a part of machine learning that originated in computer science.

I am deeply grateful to my three supervisors Ørnulf Borgan, Ida Scheel, and Kjersti Aas, as they allowed me the freedom to explore my ideas, but managed to always be available with knowledge, assistance, and encouragement. In particular, I want to thank my main supervisor and main collaborator Ørnulf. His incredible knowledge of survival analysis has been invaluable for our work, and his rigor and attention to detail has caught more of my mistakes than I care to admit. I want to thank Ida for always having time for my questions, no matter how little time she had available. I want to thank Kjersti for having multiple interesting projects available for me when I started in 2016, and for getting us through the revisions and publishing process of our first paper. I am also grateful to my other co-authors Nikolai Sellereite and Steffen Sjursen. Additionally, I want to thank Nikolai for showing me the KKBox churn prediction data set which ended up being an important part of two other papers he was not a part of.

During my time as a research fellow, I got to visit Trevor Hastie at Stanford University for six months, and I am very grateful to him for including me in his research group. His students, in addition to the rest of the Stanford statistics students and staff, made my stay very enjoyable. I am grateful to BigInsight for providing me with necessary funding for the stay, in addition to the funding for all other research-related trips. I want to thank all my colleagues at the Section of Statistics and Data Science at UiO and at the Norwegian Computing Center, in addition to Gudmund Horn Hermansen and Erlend Aune for inspiring conversations and conference trips.

Finally, I want to thank my friends and family for making my time outside the university so great. In particular, I am grateful to Amalie for her patience, understanding, and for never questioning the time I spent on my research, even when it came at the expense of our time together.

Håvard Kvamme
Oslo, December 2019

List of Papers

Paper I

Håvard Kvamme, Nikolai Sellereite, Kjersti Aas, and Steffen Sjursen. Predicting Mortgage Default Using Convolutional Neural Networks. *Expert Systems with Applications*, 102: 207–217, 2018.

Paper II

Håvard Kvamme, Ørnulf Borgan, and Ida Scheel. Time-to-Event Prediction with Neural Networks and Cox Regression. *Journal of Machine Learning Research*, 20(129): 1–30, 2019.

Paper III

Håvard Kvamme and Ørnulf Borgan. Continuous and Discrete-Time Survival Prediction with Neural Networks. *Submitted for publication.*

Paper IV

Håvard Kvamme and Ørnulf Borgan. The Brier Score under Administrative Censoring: Problems and Solutions. *Submitted for publication.*

Contents

Chapter 1

Introduction

As the title of this thesis suggests, we investigate how machine learning, and in particular neural networks, can be applied for time-to-event prediction. The topic of time-to-event prediction is a subfield of survival analysis predominately concerned with *when* in the future an event will occur. So, contrary to the rest of survival analysis research, we make little effort to understand *why* the event occurred. The event in question can be a mortgage default, as in Paper I; customers stopping their subscription to a service, as in Paper II; the failure time of a mechanical system; an actual death from a disease; or any other suitable event. In the context of this thesis, the term *neural networks* refers to a set of machine learning algorithms, also known as *artificial neural networks*, *connectionist systems*, or *deep learning*, and are not to be confused with the eponymous biological networks.

So, when are we willing to trade some of the understanding of the event-time process for improved predictive performance? Often, we are interested in both. Take customer relationship management as an example. One would, of course, like to know why customers are churning (leaving the service). But to retain current customers, a viable strategy can be to simply identify customers that are likely to churn and provide them some discount or offer. Even if we are primarily interested in understanding the effect of the variables, valuable insights might be obtained by studying methods purely focused on prediction. If such a "prediction-focused" model has much better predictive performance than our "understandable" model, the latter is clearly not using the available information to its full potential and might require some modifications. This way a "prediction-focused" model can be used to benchmark our "understandable" model.

The thesis essentially consists of three parts: the first is mainly about finding a good neural network model for a specific problem, the second part explores how methods from survival analysis can be combined with neural networks for improved predictive performance, and the third part is concerned with the evaluation of such predictions. The three topics represent the chronological progress of my research, as each of them was motivated by experiences encountered in the previous topic. In the first project, we created a neural network model for predicting mortgage defaults for Norway's largest commercial bank. We later understood that such time-to-event problems could benefit from combining methodology from survival analysis and machine learning. Then, while developing such methodology, we found that some of the evaluation criteria did not behave as expected and could, possibly, give misleading results. So, naturally, we continued by investigating the performance metrics.

To make one's research accessible to others, it is import to provide the necessary details in a usable format. Code has, therefore, been an essential addition to the papers. This has resulted in a python package that contains implementations of the proposed methods, data sets, simulations, and evaluation metrics in Papers II, III, and IV. The package is built on the popular deep learning framework PyTorch (Paszke et al., 2017) and is available at `github.com/havakv/pycox`.

The thesis is structured as follows: In Chapter 2, I will give an introduction to neural networks in the context of our papers, as all the predictive methodology is built on neural networks. This chapter is most relevant for Paper I, as it is concerned with finding a network model that is suitable for time-series modeling. The three remaining papers, on the other hand, consider the networks as general function approximators that minimize some objective function and is more concerned with the survival methodology.

In Chapter 3, I will give a brief introduction to the field of survival analysis while focusing on predictive modeling. I will introduce the statistical models that are the foundations of Papers II and III, in addition to similar methods in the literature. In short, most of the approaches are classical statistical models where a linear predictor function is replaced by a neural network. This chapter also addresses some of the common criteria for evaluating the predictive performance of time-to-event models. Here I will briefly discuss some of the problems that led us to the investigations in Paper IV.

A summary of the four papers is given in Chapter 4. In Chapter 5, I will discuss some natural extensions of the proposed methods. Also, I will use the opportunity to address some shortcomings of the papers and proposed methods.

1.1 Neural Networks and Statistical Models

A substantial part of this thesis is about improving the predictive performance of classical statistical models with methodology from machine learning. To illustrate the general approach, I will start by explaining how the Logistic Regression model for binary classification can be extended with neural networks. This should also provide the reader with the necessary terminology for understanding Chapter 2 on neural networks.

Consider an individual i with covariates $\mathbf{x}_i \in \mathbb{R}^p$ and response $y_i \in \{0, 1\}$. We assume that y_i is an observation of the random variable $Y_i \sim \text{Bernoulli}(\pi_i)$, where the parameter π_i depends on the covariates,

$$\pi_i = \mathrm{P}(Y_i = 1 \,|\, \mathbf{x}_i).$$

Logistic Regression assumes that the relationship between $\mathbf{x}_i$ and π_i is given by the logistic function

$$\pi_i = \sigma(\mathbf{x}_i) = \frac{1}{1 + \exp[-\phi(\mathbf{x}_i)]}, \tag{1.1}$$

where $\phi(\mathbf{x}_i)$, typically, is the linear predictor function $\phi(\mathbf{x}_i) = \mathbf{w}^T\mathbf{x}_i$ with parameters $\mathbf{w} \in \mathbb{R}^p$. For a data set of n individuals, we can estimate the parameter values $\mathbf{w}$ by minimizing the negative log-likelihood of Bernoulli data

$$\text{loss} = -\sum_{i=1}^{n} \Big(y_i \log[\sigma(\mathbf{x}_i)] + (1-y_i) \log\left[1-\sigma(\mathbf{x}_i)\right] \Big), \tag{1.2}$$

with respect to the parameters $\mathbf{w}$. For a new individual j, $\hat{\pi}_j = \hat{\sigma}(\mathbf{x}_j)$ is the estimated probability of $Y_j = 1$, and $\hat{\pi}_j$ can be used to make a prediction of the response y_j.

The terminology in machine learning differs slightly from that of classical statistics. The response y_i is often called a *label* or *target*, and the parameters $\mathbf{w}$ are called *weights*. The logistic function in (1.1) is referred to as the *sigmoid* function, and we rarely address its inverse; the link function that is so common in statistics. The data set is typically split into multiple subsets that serve different purposes, and the subset used to minimize the *loss function* (1.2) is called the *training set*.

A model *generalizes* well if it has accurate predictions and we often refer to the topic as *predictive performance*. Consequently, we are typically not interested in minimizing the loss (1.2) with respect to the training set, but instead, we want the loss to be small for data we have not yet seen. This can be approached by considering the loss of a held-out subset, called a *validation set*, which essentially serves the same purpose as in cross-validation. Finally, we use a held-out *test set* to quantify the predictive performance of the model.

If we now refer to the linear predictor $\phi(\mathbf{x}_i) = \mathbf{w}^T\mathbf{x}_i$ as a *neural network*, we have left the field of classical statistics and are instead doing machine learning. This is because a neural network is simply a parametric model such as $\phi(\mathbf{x}_i)$, though we are typically interested in more complex functions than $\mathbf{w}^T\mathbf{x}_i$. A neural network consists of a set of transformations of the covariates that are applied in sequence. Consider three such transformations denoted h_1, h_2, and h_3. A network can then be defined as $\phi(\mathbf{x}_i) = h_3(h_2(h_1(\mathbf{x}_i)))$. The transformations are typically referred to as *layers*, and each layer's *outputs* are the values obtained from the transformation. We refer to the number of layers as the *depth* of the network, and the term *deep learning* refers to deep networks. There is, however, no formal depth requirement for a network to be considered deep. The structure of the network is often called the *network architecture* and refers to the general design of the network.

Returning to the Logistic Regression, but with neural network terminology, we find that the linear predictor $\phi(\mathbf{x}_i)$ is a single-layer network (depth of one), consisting of a single *dense*, or *fully-connected*, layer. If we include the sigmoid function (1.1) as a part of the network, we can call $\sigma(\mathbf{x}_i)$ a two-layer network where $\phi(\mathbf{x}_i)$ is a hidden layer. It is, however, more common to refer to transformations without parameters, such as the sigmoid, as *activation functions*, and we typically consider them part of a parametric layer. One can also consider the loss function (1.2) a part of the network, as it is conceptually no different from the other layers. This interpretation is mostly useful for illustrative purposes.

The most standard neural network structure is the multilayer perceptron (MLP). It consists of multiple dense layers on the form $h(\mathbf{z}) = g(\mathbf{W}\mathbf{z})$, where g is an activation function, $\mathbf{W}$ is a matrix of parameters or weights, and $\mathbf{z}$ represents the output of the previous layer. For layers h_1, h_2, and h_3, with parameters $\mathbf{W}_1 \in \mathbb{R}^{q \times p}$, $\mathbf{W}_2 \in \mathbb{R}^{r \times q}$, $\mathbf{w}_3 \in \mathbb{R}^{r}$, and sigmoid activation functions (applied element-wise), we have the network

$$\phi(\mathbf{x}_i) = \mathbf{w}_3^T \sigma\Big(\mathbf{W}_2\, \sigma\big[\mathbf{W}_1 \mathbf{x}_i\big]\Big). \tag{1.3}$$

If we replace the linear predictor in the Logistic Regression with this network, we will have a much more flexible model. We do, however, also have many more parameters. To train the model, we apply some version of the gradient descent algorithm. The gradients are obtained with the *back-propagation algorithm* which is, essentially, application of the chain rule of differentiation.

It should now be clear that neural networks can be applied to many statistical models simply by replacing the linear predictor $\mathbf{w}^T\mathbf{x}$ of the classical statistical model with a network such as in (1.3).

The network architecture of modern MLP's differs slightly from the example in (1.3). This is one of the topics of Chapter 2, together with gradient descent, back-propagation, other layer types, and more.

Chapter 2

Neural Networks

The field of neural networks is a branch of machine learning concerned with gradient-based optimization of models composed of sequences of transformations. It is commonly referred to as *deep learning*, *artificial neural networks*, or even *artificial intelligence*. One might argue that these terms actually represent different research areas or subcategories, a distinction that will be ignored in this thesis as I will simply use the term *neural networks*.

Recently, neural networks have received much attention, both from academia and the industry. This has resulted in impressive results in areas such as image recognition (Krizhevsky et al., 2012; Szegedy et al., 2015; He et al., 2015a), image segmentation (Long et al., 2015; He et al., 2017), language modeling and language translation (Wu et al., 2016; Gehring et al., 2017; Vaswani et al., 2017; Radford et al., 2018, 2019), and image and audio generation (Goodfellow et al., 2014; Oord et al., 2016; Brock et al., 2018; Razavi et al., 2019). In part, the successes of neural networks are owed to the modularity of the back-propagation algorithm for automatic gradient computation (Kelley, 1960; Rumelhart et al., 1986). While back-propagation is essentially the chain rule of differentiation, in practice it allows for modular implementations of the desired transformations. This makes the networks very applicable, as it is easy to extend or change existing network structures.

In the following, I will first give a brief overview of the development of neural networks, starting from the late 1950s. I will then present multilayer perceptrons (MLP's), which are the most basic networks, and I will cover how they are trained and regularized. At the end of the chapter, I will describe convolutional neural networks, which are an important part of Paper I.

2.1 A Brief Historical Perspective

The development of neural networks dates back to *the perceptron* by Rosenblatt (1958) and ADALINE by Widrow and Hoff (1960), both of which are binary classifiers with linear decision boundaries. Both methods were heavily criticized, especially by Minsky and Papert (1969), for their limited ability to express richer functions, and received limited academic attention. More than a decade later, Rumelhart et al. (1985, 1986) reestablished interest in neural networks by presenting the benefits of applying multiple perceptrons sequentially with non-linear functions in between them. This directly addressed the concerns raised by Minsky and Papert (1969). Furthermore, Hornik et al. (1989) established the notion that these multilayer perceptrons (MLP) are universal approximators in the sense that they can "approximate any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available.” At the same time, LeCun (1989) introduced *convolutional neural networks*, or CNN’s, that are essentially a sparse version of the MLP with some parameters in the network forced to be identical (called *weight sharing*). This idea was tailored to the grid structure of images represented by pixels and was found to be quite successful in recognizing handwritten digits for the US Postal Service (LeCun et al., 1989). The networks were, however, very computationally expensive and somewhat unpractical to train.

This period also includes the development of the generative models: Boltzmann machines (Ackley et al., 1985), Belief networks (Neal, 1992), and the Helmholtz machine (Dayan et al., 1995), in addition to Recurrent neural networks (Rumelhart et al., 1986) and Long Short-Term Memory networks (Hochreiter and Schmidhuber, 1997) for processing sequential data. At the time, the methods were quite computationally expensive, and it was problematic to train deeper networks. As a result, other methods such as Random Forests and Support-Vector Machines received more attention.

Renewed interest in the field came with the breakthrough of Hinton et al. (2006) who were able to efficiently train deep Belief networks by greedy layer-wise pre-training. Shortly after, Bengio et al. (2007) and Poultney et al. (2007) showed that other deep network structures could be trained with the same strategy. Furthermore, they were able to show that both unsupervised pre-training and deeper networks improved generalization. With this new focus on depth, neural networks were rebranded as “deep learning” (Allen, 2015; Goodfellow et al., 2016), which marks the beginning of modern neural network research.

The following period was marked by three important discoveries. The first was that graphical processing units (GPUs) could massively speed up the training process of the networks (Mohamed et al., 2009; Raina et al., 2009), which enabled researchers to faster explore new networks and allowed for more parameters in the networks. The second discovery was that better non-linear *activation functions* and better parameter initialization made the optimization problem much nicer (Jarrett et al., 2009; Glorot and Bengio, 2010; Nair and Hinton, 2010; Glorot et al., 2011). This had a large impact on both training time and generalization error. The third discovery was the importance of large labeled data sets. While this might seem obvious, researchers had, to some extent, been focusing on marginal improvements on smaller data sets. Faster computers and much “nicer” networks, enabled networks to be fitted to much larger data sets.

The most prominent example of the importance of large data sets is likely the 2012 computer vision contest ILSVRC (Russakovsky et al., 2015). Here the convolutional network by Krizhevsky et al. (2012) achieved a top-5 error rate of 15.3% (compared to the second place of 26.1%). Their network architecture was actually quite similar to the architectures proposed a decade earlier by LeCun et al. (1998), but deeper and with the advancements discussed above. Although there had been increasing interest in neural network research since 2006, the entry by Krizhevsky et al. (2012) was, in fact, the only neural network entry to the competition that year. Their success, however, marks a turning point in the popularity of neural networks and the competition has since only been won by neural network entries. In the years after 2012, there have been numerous success stories of neural networks for a variety of applications, some of which were mentioned in the introduction to this chapter.

For a more detailed historical overview, see, e.g., the blog series by Andrey Kurenkov[1], or the book by Goodfellow et al. (2016).

2.2 The Multilayer Perceptron

Multilayer perceptrons, or MLP's, are the simplest and, arguably, the most fundamental network structures. MLP's were briefly covered in the context of Logistic Regression in Section 1.1, but we will here provide a more general overview of the topic.

To fit a standard MLP, we need input covariates $\mathbf{x} \in \mathbb{R}^p$, targets $\mathbf{y} \in \mathbb{R}^m$, and a differentiable loss function $\text{loss}(\phi(\mathbf{x}), \mathbf{y})$, where $\phi(\mathbf{x})$ denotes the output of the MLP. The loss function can, for instance, be the mean squared error, or the negative log-likelihood of Bernoulli data (binary cross-entropy). An MLP consists of one or more *dense*, or *fully-connected*, layers of the form $h(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{q \times p}$ and $\mathbf{b} \in \mathbb{R}^q$ are learnable parameters. It is common to refer to the parameters $\mathbf{W}$ as *weights* and $\mathbf{b}$ as *biases*. The biases $\mathbf{b}$ are sometimes excluded from the model, as we did in Section 1.1.

For $\phi(\mathbf{x})$ to be a non-linear function of $\mathbf{x}$, we need to add non-linear transformations to the network. These transformations are called *activation functions* and are commonly applied to the output of a dense layer. In fact, activation functions are so common that we typically consider them part of the layer, meaning we have the layer $h(\mathbf{x}) = g\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$. The most commonly used activation function is the rectified linear unit, or ReLU, defined as $g(z) = \max\{0, z\}$. There are, however, many proposed alternatives to this function, such as the ELU (Clevert et al., 2015), PReLU (He et al., 2015b), and SELU (Klambauer et al., 2017). Up till the late 2000s, it was more common to use the sigmoid function $g(z) = 1/(1 + \exp[-z])$ or $g(z) = \tanh(z)$, as these more closely mimic the on/off behavior of a biological neural network. Both were, however, found to work rather poorly as the resulting gradients become very small when z is not close to 0. Nevertheless, they are still useful in other settings; the sigmoid is, for instance, commonly used to scale the final output of a network to be in $[0, 1]$ for classification tasks.

This brings us to the final layer of the MLP, the output layer. This is often a dense layer, but with an output activation that is suited to the loss function. For instance, for binary classification we can use the sigmoid activation, for multi-class classification we can use the softmax $g_j(\mathbf{z}) = \exp(z_j)/\sum_{k=1}^{m} \exp(z_k)$, for $j = 1, \ldots, m$, and for regression one can simply use the identity $g(z) = z$. Conventions here vary, but sometimes the output activation is implicitly included in the loss function to ensure numerical stability. This is, however, most relevant for the implementation and usage of a method, and often less relevant for descriptive purposes.

[1]andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning

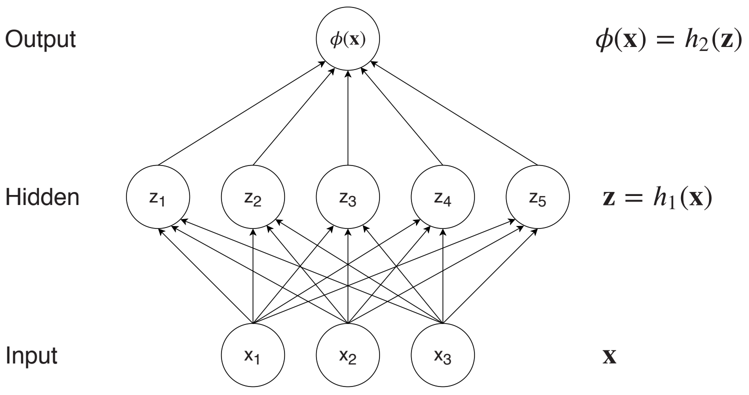

Figure 2.1: Simple MLP illustration with three covariates, a hidden layer with five units, and a single output.

Figure 2.1 shows the typical illustration of a one-hidden-layer MLP. Although the figure does not provide much additional insights compared to simply writing the model as $\phi(\mathbf{x}) = h_2(h_1(\mathbf{x}))$, it illustrates that it is common to think of a neural network as a computational graph. In particular, frameworks for working with neural networks, such as Theano, Tensorflow, and PyTorch, are typically built on this graph representation. The nodes represent values and the arrows represent the matrix multiplication with the parameters. As the activation functions are considered part of the layers, the hidden layer is actually $h_1(\mathbf{x}) = g_1(\mathbf{W}\mathbf{x} + \mathbf{b})$, and the output layer is $h_2(\mathbf{z}) = g_2(\mathbf{w}^T\mathbf{z})$.

It can be shown that with a sufficiently "wide" hidden layer $h(\mathbf{x})$, an MLP can approximate any function (Hornik et al., 1989). In practice, however, it is typically better to stack multiple hidden layers sequentially, giving networks such as $\phi(\mathbf{x}) = h_3(h_2(h_1(\mathbf{x})))$.

2.3 Training Networks

The parameter values of a neural network $\phi(\mathbf{x})$ are generally found by minimizing the loss function with some version of gradient descent. In the example of the Logistic Regression in Section 1.1, the loss function was the negative log-likelihood of Bernoulli data, and we will generally consider negative log-likelihoods as loss functions in this thesis.

Consider the network $\phi(\mathbf{x})$ with parameters denoted by $\mathbf{w}$ for simplicity, i.e., $\mathbf{w}$ contains all the parameters of all the layers. For a training set of n individuals, each with covariates $\mathbf{x}_i$, we want to obtain the minimizers

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \operatorname{loss}\left(\phi(\mathbf{x}_i),\, \mathbf{y}_i\right), \tag{2.1}$$

which, for a negative log-likelihood loss, corresponds to the maximum likelihood estimates. In practice, we find the parameter estimates by some version of gradient descent, meaning we calculate the partial derivatives of the loss with respect to the parameters, and move the parameter values in the negative direction of these partial derivatives. With α denoting a step size, a somewhat informal notation for this is

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \left(\sum_{i=1}^{n} \text{loss}\left(\phi(\mathbf{x}_i),\, \mathbf{y}_i\right) \right).$$

By repeating this *gradient step* multiple times, we will eventually reach a minimum of the loss function.

Usually, we perform each gradient step by only considering a small subset of the data set at a time. This subset is called a *batch* and is changed for every gradient update. In the neural networks literature, we refer to this procedure as *stochastic gradient descent* or just SGD, and it comes in many flavors such as Adagrad (Duchi et al., 2011), RMSprop (Hinton et al., 2012), and Adam (Kingma and Ba, 2014). Their distinction is not important for the understanding of this thesis.

If we have a model with many parameters compared to the size of the data set, the optimal parameters in (2.1) are probably overfitted to the training set, meaning that they generalize poorly. It is, therefore, common to monitor the loss on a held-out validation set, and stop the optimization procedure when this validation loss stops improving.

This is known as *early stopping* and is just one of many regularization techniques, a topic we will further investigate in Section 2.4.

2.3.1 Back-propagation

For all but the simplest neural networks, it can be cumbersome to derive the gradients of all the parameters. To this end, *back-propagation*, an application of the chain rule of differentiation, was developed to efficiently obtain the gradients of the parameters in neural networks.

Consider the neural network $\phi(\mathbf{x}) = h_m(h_{m-1}(\dots(h_1(\mathbf{x}))))$, consisting of the transformations $\mathbf{z}_1 = h_1(\mathbf{x})$, $\mathbf{z}_2 = h_2(\mathbf{z}_1), \dots, \mathbf{z}_m = h_m(\mathbf{z}_{m-1})$, each with parameters denoted $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m$. We require that each transformation in the network can calculate the partial derivatives of its output $\mathbf{z}_k$ with respect to its parameters $\mathbf{w}_k$ and its input $\mathbf{z}_{k-1}$. Informally written, this means that layer k needs to calculate $\partial \mathbf{z}_k / \partial \mathbf{w}_k$ and $\partial \mathbf{z}_k / \partial \mathbf{z}_{k-1}$. By the chain rule of differentiation, the gradients of the parameters with respect to the loss can be obtained by

$$\begin{aligned}\frac{\partial \text{loss}(\phi(\mathbf{x}),\, \mathbf{y})}{\partial \mathbf{w}_k} &= \frac{\partial \text{loss}(\phi(\mathbf{x}),\, \mathbf{y})}{\partial \mathbf{z}_k} \frac{\partial \mathbf{z}_k}{\partial \mathbf{w}_k} \\ &= \frac{\partial \text{loss}(\phi(\mathbf{x}), \mathbf{y})}{\partial \mathbf{z}_m} \frac{\partial \mathbf{z}_m}{\partial \mathbf{z}_{m-1}} \cdots \frac{\partial \mathbf{z}_{k+1}}{\partial \mathbf{z}_k} \frac{\partial \mathbf{z}_k}{\partial \mathbf{w}_k}.\end{aligned}$$

Obtaining the gradients of all parameters in the network is called a *backward pass* while calculating the loss is called a *forward pass*.

It should now be clear that modifying or replacing a layer $\mathbf{z}_k = h_k(\mathbf{z}_{k-1})$ only requires redefining $\partial\mathbf{z}_k/\partial\mathbf{w}_k$ and $\partial\mathbf{z}_k/\partial\mathbf{z}_{k-1}$, and the rest of the implementation is left untouched. This modularity is rather powerful as it greatly simplifies experimentation with new layers and network structures. In fact, in Paper II and III, we propose methods based on standard neural network structures, but with new loss functions, which can be viewed as the final layer of the computational graph.

The development of neural network methodology is further simplified by frameworks such as Tensorflow (Abadi et al., 2015), Keras (Chollet et al., 2015), PyTorch (Paszke et al., 2017), Torch (Collobert et al., 2002), Caffe (Jia et al., 2014), and Theano (Bergstra et al., 2010). Some of these also perform automatic differentiation, meaning one only needs to implement the transformations and not the partial derivatives. For a more in-depth discussion of back-propagation, see, e.g., Chapter 6.5 of the book by Goodfellow et al. (2016).

2.4 Overfitting and Regularization

Neural networks are generally overparameterized. Rumelhart et al. (1986) found that networks with just enough connections to perform a given task tend to get stuck in local minima that are far worse than the global minima, and that "adding a few more connections creates extra dimensions in weight-space and these dimensions provide paths around the barriers that create poor local minima in the lower dimensional subspaces". So, creating a neural network is not simply about finding the right number of parameters in the network. Typically, the best model, in terms of generalization error, is an appropriately regularized large model (Goodfellow et al., 2016). There is, of course, much literature on regularization, and in the following, we will investigate some of the most relevant.

2.4.1 Weight Decay

Neural networks aside, regularization in statistics and machine learning is typically associated with penalization of the loss with respect to the size of the parameter values. Consider a model $\phi(\mathbf{x})$ with parameters $\mathbf{w}$. The loss penalized by the L^2 norm is then

$$\text{loss}_\gamma = \text{loss}(\phi(\mathbf{x}), \mathbf{y}) + \gamma\frac{1}{2}\|\mathbf{w}\|_2^2,$$

where γ is a hyperparameter. The gradient updates with vanilla gradient descent will then take the form

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha\nabla\text{loss}_\gamma,$$

or, equivalently, as $\nabla\frac{1}{2}\|\mathbf{w}\|_2^2 = \mathbf{w}$,

$$\mathbf{w} \leftarrow (1 - \alpha\gamma)\mathbf{w} - \alpha\nabla\text{loss}(\phi(\mathbf{x}), \mathbf{y}). \tag{2.2}$$

For expression (2.2), we can interpret the gradient update as performing the regular non-penalized update $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \text{loss}(\phi(\mathbf{x}), \mathbf{y})$, but we shrink the weights $\mathbf{w}$ by a constant factor $(1 - \alpha\gamma)$ at every iteration. As a result, L^2-regularization is referred to as *weight decay* in the neural network literature. In fact, generally, we do not calculate the penalty $\frac{1}{2}\|\mathbf{w}\|_2^2$ directly but, instead, consider the weight decay a part of the optimization scheme.

2.4.2 Ensemble Learning and Dropout

A common approach to improving the generalization of a model is to train an ensemble of models and combine their predictions. Bagging (Breiman, 1996) and Random Forest (Breiman, 2001) are two well-known methods built on this principle, and both average the predictions of many tree models. Averaging of multiple predictions reduces the variance and, consequently, improves the generalization error of methods with high variance and low bias. As neural networks are high variance, such ensemble strategies are applicable.

There are many different approaches to ensemble learning for neural networks as one can bootstrap or subsample the training set (such as in bagging), use subsamples of the covariates, combine randomly initialized homogeneous network structures, etc. In Paper I, we found that averaging the predictions of networks fitted to subsets of the covariates (individual time series) substantially improved the performance over combining all covariates in a single network. We also averaged predictions from multiple randomly initialized networks for further improvements. Similar approaches are commonly used by the winning entries in prediction competitions such as the ILSVRC (Russakovsky et al., 2015).

Dropout (Srivastava et al., 2014) can be thought of as a computationally efficient alternative to training an ensemble of neural networks and is one of the most commonly applied regularization strategies for deep neural networks. The idea is to randomly "drop" units from the network (represented by nodes in Figure 2.1), and thus train an ensemble of all possible subnetworks. With $\phi_\gamma(\mathbf{x})$ denoting a subnetwork γ obtainable with dropout, we want to minimize the expected loss over all possible subnetworks $\mathbb{E}_\gamma\left[\text{loss}(\phi_\gamma(\mathbf{x}), \mathbf{y})\right]$. This is, however, typically intractable, so we instead minimize an unbiased estimate by sampling γ.

In practice, dropout is applied by multiplying the output of a layer with a mask of iid 0/1 variables drawn from a Bernoulli(p) distribution (at every batch iteration), where p is a hyperparameter. The ensemble prediction can then be obtained by computing the average over a sample of masks

$$\phi_{\text{ensemble}}(\mathbf{x}) = \frac{1}{m}\sum_{k=1}^{m} \phi_{\gamma_k}(\mathbf{x}).$$

A more common, less computationally expensive, alternative is to scale the output of each dropout layer by the dropout-probability p, meaning we compute an estimate of the expected output of that layer. This approach, called *weight scaling*, has limited theoretical justifications but works well in practice.

2.4.3 Data Augmentation

Possibly the simplest way to improve the generalization of a machine learning model is by increasing the size of the training set (assuming the quality of the data is reasonable). Obtaining more data if often hard or expensive, so data augmentation techniques instead attempt to artificially increase the size of the training set. The most basic form of data augmentation is to introduce noise to the covariates (Sietsma and Dow, 1991), in which case dropout can be viewed as an augmentation scheme.

For image recognition, data augmentation has become an important consideration when training networks. There has been great success in artificially augmenting data sets by subjecting images to label-preserving transformations such as rotation, translation, scaling, and mirroring. Krizhevsky et al. (2012) emphasize this approach as an important part of their winning entry to the LSVRC-2012 competition (Russakovsky et al., 2015).

In Paper I, we explore how data augmentation can be applied to time-series data by considering multiple overlapping windows of the same time series per individual. This is found to have a substantial impact on the performance of our models.

2.4.4 Other Approaches to Regularization

Three other common regularization techniques are batch normalization (Ioffe and Szegedy, 2015), though its main objective is to improve optimization; cyclical learning rates (Loshchilov and Hutter, 2016), to find better local minima; and early stopping of the optimization routine based on a held-out validation set.

Finally, an alternative approach to regularization is to change the structure of the neural networks. If one tailor the network architecture to a specific problem, one can potentially reduce the number of parameters without sacrificing the network's ability to model the objective. Two such approaches are the use of sparse layers, meaning that many weights are set to zero; and the use of shared parameter values, meaning multiple parameters are fixed to be identical. The most prominent application of these two techniques is the development of the *convolutional networks* by LeCun (1989).

2.5 Convolutional Networks

The name *neural networks* is from biology, as some of the first learning algorithms were modeled after the brain. It is, however, hard to argue that modern networks are more than loosely inspired by biological networks. Still, convolutional networks claim to capture some of the same properties as the primary visual cortex (LeCun et al., 2010; Goodfellow et al., 2016). Convolutional networks, convolutional neural networks, ConvNets, and CNN's are all terms we use to describe neural networks where convolutional layers are a crucial part of the network architecture. LeCun (1989) is considered the inventor of CNN's, though he was heavily inspired by the works of Fukushima (1980) and Lang (1988). The

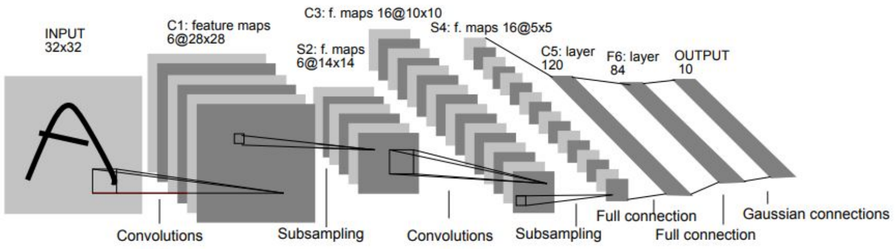

Figure 2.2: The LeNet-5 architecture from LeCun et al. (1998)

main motivation of CNN's was to reduce the number of free parameters in a network by tailoring its architecture to the grid structure of an image represented by pixels. By enforcing sparsity and weight sharing, LeCun could accomplish this reduction without necessarily reducing the size of the network.

In Figure 2.2, you can see the CNN proposed by LeCun et al. (1998) for recognizing digits (even though the figure use the character A as an example). In the figure, convolutions are represented by the small square on the input image. By applying this square to different parts of the input image one obtains the outputs in "C1: feature maps". The small squares represent convolutional kernels and are often referred to as *filters*. This is because each filter is "looking" for a pattern in the image, and its output (C1: feature maps) is a map describing which parts of the input image that contain the filter pattern. For example, a filter might look for horizontal lines in the image, while a second filter might look for vertical lines. Figure 2.2 also contains other layers than the convolutional layers, but for now, we only note that the last layers are an MLP as described in Section 2.2.

Images are not really part of this thesis, but in Paper I, we used CNN's for time-series classification. Therefore, when describing the convolutions in detail, I will phrase them in terms of time series. The reader should, however, note that the following generalizes to images.

Convolutional layers are, essentially, the discrete convolution operation known from mathematics (see, e.g., Goodfellow et al., 2016). We do, however, prefer to think of a convolution as a filter $\mathbf{w}$ applied to a patch $\tilde{\mathbf{x}}$ of the time series $\mathbf{x}$, where $\mathbf{w}$ is a vector of parameters. In this case, the convolution takes the form of the cross-correlation $c = \mathbf{w}^T\tilde{\mathbf{x}}$. The same filter $\mathbf{w}$ is then applied to numerous patches from the time series, denoted $\tilde{\mathbf{x}}_i$, which results in a vector of outputs $\mathbf{c}$. We can alternatively express this operation as $\mathbf{c} = \mathbf{A}\mathbf{x}$, where $\mathbf{x}$ is the full time-series and $\mathbf{A}$ is a sparse diagonal-constant matrix (Toeplitz matrix) with $\mathbf{w}$ descending along the diagonal. This means that the convolution can be viewed as a fully-connected layer, but with sparse weights and weight sharing.

The new feature series $\mathbf{c}$ tells us something about the correlation between the time series and the filter $\mathbf{w}$. In other words, $\mathbf{c}$ tells us where in the time series we find patterns close to that of the filter $\mathbf{w}$. As $\mathbf{w}$ can only express a single

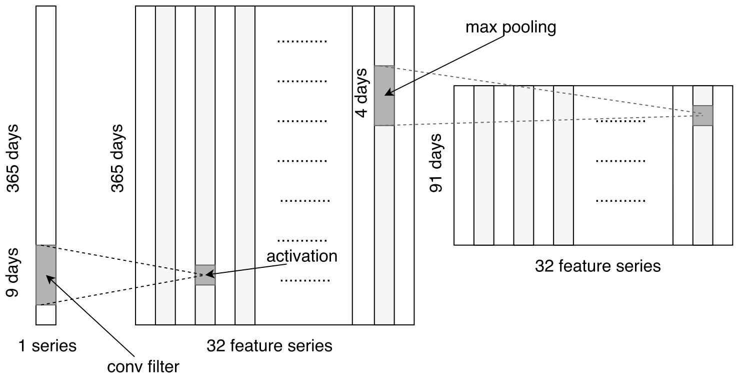

Figure 2.3: Illustration from Paper I of a convolutional layer applied to a time series. Only the first part of the full network is included. From left to right the figure shows a time series, a convolution operation with the resulting features, and pooling operation with the resulting features.

pattern, we repeat the operation with q different filters $\{\mathbf{w}_1, \mathbf{w}_2, \dots \mathbf{w}_q\}$, giving the corresponding feature series $\{\mathbf{c}_1, \mathbf{c}_2, \dots \mathbf{c}_q\}$. This is illustrated in Figure 2.3 where we have a time series of length 365 and apply 32 different filters of size 9.

Each filter $\mathbf{w}$ consists of randomly initialized parameters and is fitted in the same manner as the dense layers in the MLP's. After the convolutional layers, we typically add an MLP that performs the final transformations. When trained, the different filters can extract various information that is useful for minimizing our loss function. When we build a neural network consisting of multiple convolutional layers applied in sequence (with non-linear activation functions in between), the network can learn a hierarchy of representations. This is hard to illustrate for time series, but it is quite common to view for images (see, e.g., Simonyan et al., 2013; Zeiler and Fergus, 2014). For images, we generally find that the first layers only learn to recognize simple patterns such as edges with different orientations, while the subsequent layers are able to detect more complex patterns such as corners and textures. The last layers can find whole objects such as faces, text, and animals.

2.5.1 Controlling the Feature Space

Consider a time-series input of length m and a filter of length v. If we slide the filter across the time series one index at a time, the resulting output will be of length $m - v + 1$. For deeper networks (many convolutional layers), the output might become very short. *Zero padding* can be applied to prevent this shrinkage from occurring and works by adding zeros to the beginning and end of the time series, thus making the time series longer. The typical argument for applying zero padding is, however, that it preserves the information at the borders of the series.

On the other hand, it can be beneficial to reduce the length of the convolved features, as larger feature spaces require more computations and are more prone to overfitting. We can, therefore, use downsampled convolutions, or strided convolutions, where we move the filter by s indices at a time. This will reduce the size of the output by a factor $1/s$.

An alternative to downsampled convolutions is to use some version of *pooling*. Pooling summarizes an area of the input (i.e., input to the pooling operation) by computing a simple function such as the average or maximum. This creates a representation that is approximately invariant to small translations in the input and is, therefore, particularly useful when the existence of a pattern is more important than the exact location of that pattern. If we apply pooling to non-overlapping patches, we reduce the length of the output. This is illustrated in Figure 2.3, where we perform non-overlapping max pooling over 4 features (days) at a time and, consequently, reduce the feature length by a factor of 4. Correspondingly, an example of pooling for images is shown in Figure 2.2 where pooling is referred to as subsampling.

2.5.2 CNN's for Time Series

In Paper I, we applied CNN's to the time-series classification problem of mortgage default. At that time, there were limited literature on CNN's for time series, as recurrent neural networks (RNN's), such as the long short-term memory networks (Hochreiter and Schmidhuber, 1997), were the preferred architectures for processing sequential data. This is, in a sense, surprising as the predecessor of the CNN's, the *time-delay neural networks* (Lang, 1988) can be considered convolutions applied to time series. Although versions of RNN's continue to be the most popular architecture for time series, CNN's have the benefit of being more computationally efficient. Therefore, CNN's for time series is an active research area, and there have been multiple examples CNN's claiming to have better performance than RNN's (Zhang et al., 2015; van den Oord et al., 2016; Bai et al., 2018; Elbayad et al., 2018).

Chapter 3

Time-to-Event Prediction

Time-to-event prediction is a subfield of survival analysis concerned with prediction of future events. In classical statistics, survival analysis tends to focus on understanding how variables affect the event-time distribution. We here distinguish us from the majority of the statistical survival literature, as we will make no claim of understanding the survival process. Instead, we model the event-times as a “black box” that produces estimates of the event-time distribution.

In the following, we will start with a basic introduction to survival analysis to make it clear how we can model the event-time distributions. Next, we will look at some classical regression models and see how they can be extended with neural networks. At the end of the chapter, we will discuss some of the typical evaluation criteria used for evaluating survival predictions.

3.1 Event-Time Modeling

In this chapter, individuals are generally denoted by i and their covariates by $\mathbf{x}_i$. But in the following basic introduction to survival analysis, the covariates are disregarded for simplicity of notation and, instead, only a single individual is considered.

Let $f(t)$ be the probability density function (PDF) of the continuous-time event time T^*. In survival analysis, it is common to study the *survival function* $S(t)$, which gives the probability of survival beyond time t

$$S(t) = \mathrm{P}(T^* > t).$$

The *hazard rate* $h(t)$ is also quite commonly studied and, for continuous time, it is defined by the conditional probability

$$h(t) = \lim_{\Delta t \to 0} \frac{\mathrm{P}(t \leq T^* < t + \Delta t \,|\, T^* \geq t)}{\Delta t}.$$

This means that the hazard tells us something about the immediate likelihood of an event, given that the event has yet to occur,

$$h(t)\, \Delta t \approx \mathrm{P}(t \leq T^* < t + \Delta t \,|\, T^* \geq t).$$

Note that the density, the survival function, and the hazard rate are all representations of the same distribution, so knowing one is sufficient to obtain the other two. So, with $F(t)$ and $H(t)$ denoting the cumulative distribution

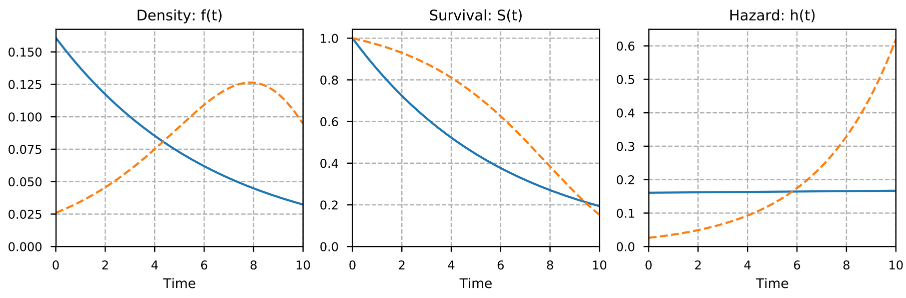

Figure 3.1: Illustration of the density, survival, and hazard for two individuals.

function and cumulative hazard rate

$$
\begin{aligned}
F(t) &= \int_0^t f(u)\, du, \\
H(t) &= \int_0^t h(u)\, du,
\end{aligned}
$$

we have that

$$
\begin{aligned}
f(t) &= -S'(t) = h(t)\, S(t), \\
S(t) &= 1 - F(t) = \exp[-H(t)], \\
h(t) &= \frac{f(t)}{S(t)} = -\frac{d \log [S(t)]}{dt}.
\end{aligned}
$$

An illustration of these three representations of the event-time distribution is shown in Figure 3.1 for two individuals (blue and orange). Clearly, the three functions give contrasting views of the survival distribution, and it is reasonable to study all three. In the four papers of this thesis, the objective was to obtain estimates of the survival function and we will, therefore, consider estimation of the survival function the objective of this thesis as well. The hazard and density are, however, very useful means to this end.

3.1.1 Censoring and Truncation

In Figure 3.1, one might notice that the survival functions $S(t)$ do not drop below approximately 0.2, but it looks like both would continue to decrease if we extend the time axis beyond 10. If we are only able to make observations in the interval between the start and 10, we say that individuals are *censored* at time 10. Censoring, and the related topic *truncation*, do, however, extend far beyond this illustrative case, and is a fundamental part of survival analysis.

Let C^* denote a right-censoring time. C^* is typically considered a random variable, though that may not always be the case. I will only use the notation

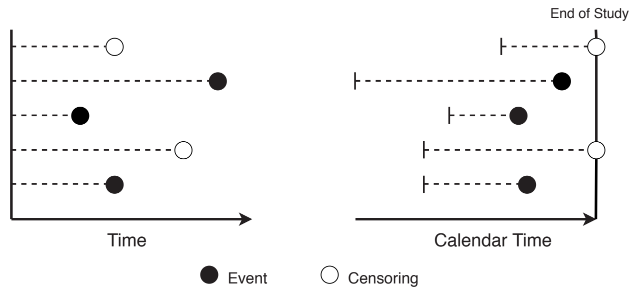

Figure 3.2: Example of administrative censoring (progressive type I) for five individuals (two censored and three observed events). The left figure represents the time scale in which we model the event times. The right figure represents the same individuals, but in the calendar time in which the data was collected.

C^* for the censoring time, and the reader is expected to understand from context whether or not it is a random variable.

Let us first consider random censoring, where C^* is a random variable with density $f_{C^*}(t)$ and survival function $S_{C^*}(t)$ defined in the same manner as for the event time T^*. A typical example of random censoring is when an individual i decides to no longer be part of a study and, consequently, we only know that the individual had still to experience the event at time C_i^*. As both C_i^* and T_i^* are random variables, we are only able to observe the shorter of the two. We, therefore, define the observed time T_i and event indicator D_i as follows,

$$T_i = \min\{T_i^*, C_i^*\} \tag{3.1}$$

$$D_i = \mathbb{1}\{T_i^* \leq C_i^*\}. \tag{3.2}$$

In Papers II and III, we consider observations on this form.

Administrative censoring, or *progressive type I* censoring, is an alternative where the right-censoring times C_i^* are known for *all* individuals. So we know C_i^* even when $T_i^* < C_i^*$. Administrative censoring often occurs in studies where individuals have different times of entry to the study (in calendar time) with a defined end date of the study. This is illustrated in Figure 3.2, where the left figure shows the event and censoring times in the study, while the right figure shows the calendar times at which the individuals were recorded. As an example, the figure could illustrate the time to mortgage defaults, such as in Paper I, where the mortgages are started on different dates and the end date of the study is today. The censoring time C_i^* will, in this case, be the difference between the entry date and the end-of-study date. The observations are still given by (3.1) and (3.2), but we now also know all C_i^*'s.

As the time of entry can be considered random, it is common to consider C_i^* random in these data sets too. Also, researchers tend to disregard the additional information provided by C_i^* and treat the problem as we did for non-administrative censoring.

Only right-censored observations were considered in the papers that constitute this thesis. Left-truncation is, however, also quite common in survival data. As an example, for a left-truncation time V_i, we will only observe individual i if $T_i^* > V_i$. This means that if $T_i^* < V_i$ we might not even know that the individual ever existing. Consequently, we are limited to model the distribution conditioned on the truncation time, or we can make sufficiently strong assumptions on the distribution of V_i to model the full event times. As an example of left-truncation, Klein and Moeschberger (2003) consider the age at death of residents at a retirement center. Individuals need to survive to a sufficient age to enter the retirement center, meaning all individuals that died before this time never entered the study.

The topic of left-truncation will be revisited in Chapter 5. For a more extensive review of other types of censoring and truncation, see, e.g., Chapter 3 of the book by Klein and Moeschberger (2003).

3.1.2 Discrete-Time Survival

Thus far, we have considered the time scale to be continuous. Sometimes, it can, however, be more appropriate with a discrete time scale. As an example of discrete-time survival data, we can consider the event of leaving a subscription service where each subscription payment lasts for a full month at a time. This means that the events can only occur after month number $1, 2, 3$, etc., even though the decision to leave might happen sometime between the discrete time points. It is also quite common to have events that occur in continuous-time, but we are only able to make observations at discrete points in time.

Let $0 = \tau_0 < \tau_1 < \ldots$ denote the discrete time scale. The probability mass function (PMF), the survival function, and the discrete hazard are defined as

$$\begin{aligned}
f(\tau_j) &= \mathrm{P}(T^* = \tau_j),\\
S(\tau_j) &= \mathrm{P}(T^* > \tau_j) = \sum_{k>j} f(\tau_k),\\
h(\tau_j) &= \mathrm{P}(T^* = \tau_j \,|\, T^* > \tau_{j-1}) = \frac{f(\tau_j)}{S(\tau_{j-1})}.
\end{aligned}$$

Consequently, we also have that

$$\begin{aligned}
f(\tau_j) &= S(\tau_{j-1}) - S(\tau_j),\\
S(\tau_j) &= [1 - h(\tau_j)]\, S(\tau_{j-1}) = \prod_{k=1}^{j} [1 - h(\tau_k)].
\end{aligned}$$

Assuming the same time scale for the censoring distribution, we denote the corresponding PMF and survival $f_{C^*}(\tau_j)$ and $S_{C^*}(\tau_j)$.

In Paper III, we are concerned with discrete-time models, and we also address how the discrete-time models can be applied to continuous-time data. Both discretization schemes for continuous-time data and interpolation schemes for obtaining continuous-time survival estimates are considered.

3.1.3 The Likelihood for Right-Censored Event Times

Continuing with the discrete time-scale and right-censored observations of the form

$$\begin{aligned} T_i &= \min\{T_i^*, C_i^*\}, \\ D_i &= \mathbb{1}\{T_i^* \leq C_i^*\}, \end{aligned}$$

we will derive the likelihood. For observations t and d, the probability of the observations is

$$\mathrm{P}(T = t, D = d) = \mathrm{P}(T^* = t, C^* \geq t)^d \, \mathrm{P}(T^* > t, C^* = t)^{1-d}.$$

If the observed censoring time c is deterministic, then $d = 1$ implies that $t \leq c$, and $d = 0$ implies that $t = c$. This gives us

$$\mathrm{P}(T = t, D = d) = \mathrm{P}(T^* = t)^d \, \mathrm{P}(T^* > t)^{1-d} = f(t)^d \, S(t)^{1-d}. \tag{3.3}$$

It is, however, more common to consider random censoring C^*. If we assume that T^* and C^* are independent, we obtain the probability

$$\begin{aligned} \mathrm{P}(T = t, D = d) &= \left[\mathrm{P}(T^* = t)\, \mathrm{P}(C^* \geq t)\right]^d \left[\mathrm{P}(T^* > t)\, \mathrm{P}(C^* = t)\right]^{1-d} \\ &= \left[f(t)\left(S_{C^*}(t) + f_{C^*}(t)\right)\right]^d \left[S(t) f_{C^*}(t)\right]^{1-d} \\ &= \left[f(t)^d \, S(t)^{1-d}\right] \left[f_{C^*}(t)^{1-d} \left(S_{C^*}(t) + f_{C^*}(t)\right)^d\right]. \end{aligned}$$

Now, assuming $f(t)$ does not depend on the parameters of $f_{C^*}(t)$, we can view $[f_{C^*}(t)^{1-d} \left(S_{C^*}(t) + f_{C^*}(t)\right)^d]$ as a constant and only consider the proportional expression

$$\mathrm{P}(T = t, D = d) \propto f(t)^d \, S(t)^{1-d},$$

which is identical to the expression with deterministic censoring in (3.3). So, in both cases, for individuals denoted by i, with covariates $\mathbf{x}_i$, observed times t_i, and event indicators d_i, we consider the likelihood

$$L = \prod_i f(t_i \,|\, \mathbf{x}_i)^{d_i} \, S(t_i \,|\, \mathbf{x}_i)^{1-d_i}. \tag{3.4}$$

If we substitute the PMF with the PDF in (3.4), we obtain the corresponding likelihood for continuous-time data. As both the PMF and PDF are denoted $f(t_i \,|\, \mathbf{x}_i)$, the expression is unchanged. The derivations are also very close to that of (3.4).

The likelihood for right-censored event times in (3.4) is the foundation for many survival methods, some of which will be addressed in the next sections. In fact, the survival methods presented Papers II and III are built on this likelihood. However, if we have, e.g., truncated or left-censored data, some alterations need to be made. This is somewhat outside the scope of this thesis, so I will refer the interested reader to Chapter 3.5 of the book by Klein and Moeschberger (2003).

3.2 Regression Models

The application of regression models in survival analysis typically involves maximization of the survival likelihood (3.4). It is, however, more common to phrase the likelihood in terms of the hazard rate $h(t_i \,|\, \mathbf{x}_i)$, rather than the PMF/PDF $f(t_i \,|\, \mathbf{x}_i)$. Considering continuous-time data, with individuals denoted by i with covariates $\mathbf{x}_i$, observed times t_i, and event indicators d_i, we can rewrite (3.4) to

$$L = \prod_i h(t_i \,|\, \mathbf{x}_i)^{d_i} \exp\left[-H(t_i \,|\, \mathbf{x}_i)\right]. \tag{3.5}$$

Correspondingly, for discrete-time we have

$$L = \prod_i \left(h(t_i \,|\, \mathbf{x}_i)^{d_i} [1 - h(t_i \,|\, \mathbf{x}_i)]^{1-d_i} \prod_{j:\, \tau_j < t_i} [1 - h(\tau_j \,|\, \mathbf{x}_i)] \right). \tag{3.6}$$

Many survival models have been proposed by defining a parameterization of $h(t \,|\, \mathbf{x})$, but we will here only consider the most relevant for the work in Papers II and III. A more general overview is given in the book by Klein and Moeschberger (2003), and Tutz and Schmid (2016) provide an overview of discrete-time regression models.

3.2.1 Cox Regression

The Cox proportional hazards model assumes that the continuous-time hazard rate is defined as

$$h(t \,|\, \mathbf{x}) = h_0(t) \exp[g(\mathbf{x})], \tag{3.7}$$

where $h_0(t)$ is a non-parametric function and $g(\mathbf{x})$ is a parametric function, typically $g(\mathbf{x}) = \boldsymbol{\beta}^T \mathbf{x}$. Note that the hazard ratio between two individuals i and j is constant with respect to time

$$\frac{h(t \,|\, \mathbf{x}_i)}{h(t \,|\, \mathbf{x}_j)} = \exp[g(\mathbf{x}_i) - g(\mathbf{x}_j)],$$

which explains why we call it a *proportional hazards* model.

We want to fit this model with the likelihood (3.5), but the likelihood can be made arbitrarily large by letting $h_0(t)$ be zero except for close to the observed event times $\{t_i : d_i = 1\}$ where we let it peek higher and higher. We, therefore, assume a cumulative baseline hazard $H_0(t)$ in the form of a step function with steps at the event times. With η_i denoting the increase in $H_0(t)$ at time t_i, we have

$$H_0(t) = \sum_{i:\, t_i \leq t} \eta_i.$$

Under this extended model, the likelihood (3.5) takes the form

$$L = \prod_i \left(\eta_i \exp[g(\mathbf{x}_i)]\right)^{d_i} \exp\left(-H_0(t_i)\exp[g(\mathbf{x}_i)]\right). \tag{3.8}$$

The maximizer of (3.8) with respect to η_i can be shown to be

$$\hat{\eta}_i = \frac{d_i}{\sum_{j \in \mathcal{R}_i} \exp[g(\mathbf{x}_j)]},$$

where $\mathcal{R}_i = \{j : t_j \geq t_i\}$ is the set of individuals still *at risk* at time t_i. Correspondingly, we have

$$\hat{H}_0(t) = \sum_{i:\, t_i \leq t} \frac{d_i}{\sum_{j \in \mathcal{R}_i} \exp[g(\mathbf{x}_j)]}, \tag{3.9}$$

which is known as the *Breslow estimator*. By inserting these maximizers into (3.8) we obtain the profile likelihood

$$\begin{aligned} L &= \left[\prod_i \left(\frac{\exp[g(\mathbf{x}_i)]}{\sum_{j \in \mathcal{R}_i} \exp[g(\mathbf{x}_j)]}\right)^{d_i}\right] \exp\left[-\sum_i d_i\right] \\ &\propto \prod_{i:\, d_i = 1} \frac{\exp[g(\mathbf{x}_i)]}{\sum_{j \in \mathcal{R}_i} \exp[g(\mathbf{x}_j)]}. \end{aligned} \tag{3.10}$$

The expression in (3.10) is known as the *Cox partial likelihood* and can be maximized to obtain parameter estimates of $g(\mathbf{x})$. With such estimates $\hat{g}(\mathbf{x})$, survival estimates can be obtained with

$$\hat{S}(t \,|\, \mathbf{x}) = \exp[-\hat{H}(t \,|\, \mathbf{x})] = \exp[-\hat{H}_0(t)\, \hat{g}(\mathbf{x})].$$

In summary, we have shown that Cox regression maximizes the survival likelihood for right-censored event-times (3.5), by considering a cumulative baseline hazard $H_0(t)$ in the form of a step function. Note that the estimates above do not directly provide us with estimates of the baseline hazard $h_0(t)$. However, we are rarely interested in these estimates anyway.

For a more rigorous derivation of the partial likelihood as a profile likelihood, see Johansen (1983) or the book by Aalen et al. (2008, p. 204). The connection between the partial likelihood and the survival likelihood emphasizes the relationship of Cox regression to the other methods discussed in this chapter. It is, however, an uncommon way to introduce Cox regression, and a more standard approach is given in Paper II.

The Cox model can be made more flexible by considering other versions of the parametric function $g(\mathbf{x})$ in (3.7). For instance, we can let $g(\mathbf{x})$ be parameterized by a neural network. Faraggi and Simon (1995) were the first to propose this approach, though with limited improvements over regular Cox regression. However, with modern deep neural networks, numerous papers have shown improved predictive performance of this approach (Katzman et al., 2018; Ching et al., 2018; Yousefi et al., 2017; Zhu et al., 2016; Zhu et al., 2017). In Paper II, we propose an even more flexible relative risk model without the proportionality constraint of the Cox model. In short, this proposed *Cox-Time* method is made possible by allowing the parametric function to depend on time, meaning we have $g(t, \mathbf{x})$. This time-dependence makes the partial log-likelihood (3.10) quite computationally expensive, so the partial log-likelihood is approximated with techniques from nested case-control studies. A proportional hazards version of this method is also proposed which is referred to as CoxCC and Cox-MLP (CC) in Papers II and III.

3.2.2 Discrete-Time Regression

What we today know as *Cox regression* was originally proposed in the paper by Cox (1972), one of the most cited statistical papers of all time. It is, however, less known that the same paper also proposed a discrete-time hazard parameterization by considering the sigmoid (inverse logit) of the linear predictor $\phi_j(\mathbf{x}) = \alpha_j + \boldsymbol{\beta}^T \mathbf{x}$,

$$h(\tau_j \,|\, \mathbf{x}) = \frac{1}{1 + \exp[-\phi_j(\mathbf{x})]}. \tag{3.11}$$

Cox considered this an *ad hoc* modification of his continuous-time model and, therefore, proposed an estimation procedure analogous to the partial likelihood (3.10). Brown (1975) later showed that we can estimate the parameters of $h(\tau_j \,|\, \mathbf{x})$ in the same manner as a regular Logistic Regression. By defining the labels $y_{ij} = \mathbb{1}\{\tau_j = t_i, d_i = 1\}$, the likelihood (3.6) can be written as

$$L = \prod_i \prod_{j:\, \tau_j \leq t_i} h(\tau_j \,|\, \mathbf{x}_i)^{y_{ij}} \, [1 - h(\tau_j \,|\, \mathbf{x}_i)]^{1 - y_{ij}},$$

which we recognize as the Bernoulli likelihood. Brown (1975), therefore, named this approach the *Logistic-Hazard* model, but it is now also referred to as *Logistic Discrete Hazard* and *Partial Logistic Regression.*

The logistic function (3.11) is still very commonly used, though there are alternatives such as the probit link, log link, and clog-log link (Tutz and Schmid, 2016, Chapter 3). There have also been proposed different versions of $\phi_j(\mathbf{x})$ and, to the best of my knowledge, Biganzoli et al. (1998) were the first to let $\phi_j(\mathbf{x})$ be parameterized by a neural network. Gensheimer and Narasimhan (2019) later parameterized $\phi_j(\mathbf{x})$ with modern deep neural networks.

An alternative to the Logistic-Hazard is to instead parameterize the PMF $f(\tau_j \,|\, \mathbf{x}_i)$ in (3.4). This was the approach of the DeepHit method proposed by Lee et al. (2018). However, Lee et al. (2018) assume all individuals experience an event within a finite time scale $\tau_1, \ldots, \tau_m$. This is an assumption we find unnecessary, so in Paper III we propose a similar version that allows for survival past time τ_m.

In Paper III, we compare the Logistic-Hazard and PMF approach to parameterizing the survival likelihood. Furthermore, we investigate how they can be applied to continuous-time data by discretization of the time scale and interpolation for continuous-time survival estimates.

3.2.3 The Piecewise Exponential Model

The piecewise exponential model was first proposed by Holford (1976) and later extended by Friedman (1982). For a defined set of times $0 = \tau_0 < \tau_1 < \cdots < \tau_m = \tau$, assume that the hazard is constant in each interval, meaning $h(t \,|\, \mathbf{x}) = \eta_j(\mathbf{x})$ for $t \in (\tau_{j-1}, \tau_j]$. Now, defining $y_{ij} = \mathbb{1}\{t_i \in (\tau_{j-1}, \tau_j],\, d_i = 1\}$ and

$$\Delta\tilde{t}_{ij} = \begin{cases} \tau_j - \tau_{j-1}, & \text{if } t_i > \tau_j \\ t_i - \tau_{j-1}, & \text{if } \tau_{j-1} < t_i \leq \tau_j \\ 0, & \text{if } t_i \leq \tau_{j-1}, \end{cases}$$

the likelihood in (3.5) can be written as

$$L = \prod_i \prod_{j:\, \tau_j \leq t_i} [\Delta\tilde{t}_{ij}\, \eta_j(\mathbf{x}_i)]^{y_{ij}} \exp\left[-\Delta\tilde{t}_{ij}\, \eta_j(\mathbf{x}_i)\right].$$

This is proportional to the likelihood of independent Poisson-distributed variables y_{ij} with expectations $\mu_{ij} = \Delta\tilde{t}_{ij}\, \eta_j(\mathbf{x}_i)$. So if we define $\eta_j(\mathbf{x}) = \exp[\phi_j(\mathbf{x})]$, we can use GLM software for the optimization procedure. Fornili et al. (2014) extended this methodology by parameterizing $\phi_j(\mathbf{x})$ with a neural network. In Paper III we revisit this method, but with modern frameworks for neural networks and some modifications to the link function.

3.2.4 Binary Classifiers

Outside the survival literature, a common approach to event-time prediction is to frame the problem as binary classification. For a time τ, we want to estimate the probability of experiencing an event before this time, meaning we want to estimate $\mathrm{P}(T_i^* \leq \tau)$. By disregarding individuals censored before time τ, meaning that we remove individuals with $C_i^* < \min\{T_i^*, \tau\}$, we can fit a binary classifier to the remaining individuals with labels $y_i = \mathbb{1}\{t_i > \tau\}$. This was our approach in Paper I for predicting mortgage defaults.

If we want to obtain predictions for more than a single point in time, we can either repeat the steps above for multiple τ_j's, or we can create a model that does this implicitly. The latter can be achieved with a neural network with an output for every τ_j. This approach was applied in Paper IV.

Clearly, estimates obtained from such methods are dependent on the censoring distribution. However, for a negligible amount of censoring, the estimates of the survival function will be close to those of a method that accounts for censoring. Furthermore, if we are only interested in discrimination, meaning well-calibrated estimates is not important, the binary classifier can be a reasonable approach.

There are of course some benefits to using a survival method in this case too, as the ranking can be affected by the censoring bias. But, at least in my experience, these differences are smaller than those of the evaluation metrics that also consider the calibration of the estimates.

3.3 Evaluation of Survival Estimates

A substantial part of the machine learning literature is based on empirically verifying that a new method performs better than competing methodology for some evaluation criteria. So, when we combine survival methodology with methods from machine learning, it is crucial that we have suitable evaluation criteria for quantifying the performance of the proposed methods.

A good evaluation metric for survival methodology needs to account for censored observations. This is quite problematic as popular metrics typically depend on the censoring distribution. If the scores are affected by the researcher's assumptions on the censoring distribution, it is not obvious to what degree we can trust the results. To stress this point, consider the analogous situation in classification where there are missing labels (responses) in our test set. An evaluation metric, such as accuracy, will need to make some assumptions on the distribution of the missing labels. For example, we can assume that the labels are missing at random and simply disregard the missing labels. If, in fact, the probability of a label being missing is correlated with the label itself, the scores would be biased as the class proportions would be wrong.

Returning to survival data, a seemingly obvious choice for evaluating survival predictions is to use the likelihood for right-censored event times in (3.4) and (3.5). The likelihood is, however, not very well suited for comparing estimates from different approaches, such as discrete and continuous time, and is, therefore, rarely used.

An alternative metric, that we use in Papers II, III, and IV, is the Brier score. If there is no censoring, the Brier score is essentially the mean squared prediction error between our survival estimates $\hat{S}(t \mid \mathbf{x}_i)$ and the current state of the individuals $\mathbb{1}\{T_i^* > t\}$,

$$\mathrm{BS}(t) = \frac{1}{n} \sum_{i=1}^{n} \left[\mathbb{1}\{T_i^* > t\} - \hat{S}(t \mid \mathbf{x}_i) \right]^2. \tag{3.12}$$

Therefore, the Brier score is a function of the time t. In practice, to obtain a single number for the score, we can calculate the integrated Brier score by numerically approximating the integral

$$\mathrm{IBS} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \mathrm{BS}(u)\, du,$$

for a suitable time interval $[t_1,\, t_2]$.

3.3.1 Inverse Probability of Censoring Weighting

For right-censored event times, the Brier score can be approximated by a weighting scheme that preserves the expectation of the score. Graf et al. (1999) proposed to weight the Brier score (3.12) by the inverse probability of censoring,

$$\mathrm{BS}_{\mathrm{IPCW}}(t) = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{\hat{S}(t \,|\, \mathbf{x}_i)^2 \mathbb{1}\{t_i \leq t,\, d_i = 1\}}{\hat{S}_{C^*}(t_i - \,|\, \mathbf{x}_i)} + \frac{[1 - \hat{S}(t \,|\, \mathbf{x}_i)]^2 \mathbb{1}\{t_i > t,\}}{\hat{S}_{C^*}(t \,|\, \mathbf{x}_i)},\right]$$

where t_i- denotes the time right before t_i. They proposed to use the Kaplan-Meier estimator to obtain censoring estimates $\hat{S}_{C^*}(t \,|\, \mathbf{x})$, meaning one assumes that the censoring distribution is independent of the covariates. Gerds and Schumacher (2006) extended the idea to covariate-dependent censoring estimates.

The Brier score is obtained directly from the survival estimates. This makes the Brier score attractive for comparing any methods that produce survival estimates, which is the case for most survival methods. A drawback is that the scores are dependent on our estimate of the censoring distributions, which can be as difficult to estimate as the event-time distributions. If the censoring distributions substantially affect the scores, it can be hard to make any strong claims based on the results. In Paper IV, we discuss some of the issues when this scheme is applied to administratively censored times, in particular when covariates are very informative of the censoring times.

The IPCW scheme is more general than the Brier score and can be applied to any suitable function. For example, in Paper II, we use the same weighting scheme for the binomial log-likelihood, as proposed by Graf et al. (1999).

3.3.2 Concordance

The Harrell Jr et al. (1982) concordance index, or C-index, is a commonly used evaluation metric in the predictive survival literature. For uncensored data, it can be interpreted as the relative frequency of concordant pairs, meaning, for a given time t, it provides an estimate of

$$C(t) = \mathrm{P}(\hat{S}(t \,|\, \mathbf{x}_i) < \hat{S}(t \,|\, \mathbf{x}_j) \,|\, T_i^* < T_j^*). \tag{3.13}$$

This is particularly useful for "one-to-one" corresponding models, defined by the property $S(t \,|\, \mathbf{x}_i) < S(t \,|\, \mathbf{x}_j) \iff S(s \,|\, \mathbf{x}_i) < S(s \,|\, \mathbf{x}_j)$ for all s and t, as the concordance is not dependent on the time, i.e., $C(t) = C(s)$. The Cox proportional hazards model is an example of a "one-to-one" corresponding model, as the survival functions are given by $S(t \,|\, \mathbf{x}) = S_0(t)^{\exp[g(\mathbf{x})]}$.

For models that are not "one-to-one", it is somewhat common to replace the survival functions in (3.13) with summary statistics $M(\mathbf{x})$, as this results in a single score that is not a function of the time. In this case, $M(\mathbf{x}_i)$ is meant to represent the ranking of individual i's "predicted event time", and can, for example, be the integral of the survival function, $M(\mathbf{x}) = \int_0^\tau S(t \,|\, \mathbf{x})\, dt$. This approach might suppress some drawbacks of model restrictions, such as proportional hazards, but the interpretability of a single number for the concordance might be worth it. Alternatively, one can, of course, integrate the concordance scores in (3.13) with respect to the time t by some numerical approximation.

For censored data, the concordance is dependent of the censoring distribution, making the scores harder to interpret. It is, however, still quite common to use it. To retain the original interpretation, Uno et al. (2011) and Gerds et al. (2013) proposed an IPCW concordance in a similar manner to that of the Brier score in Section 3.3.1.

Antolini et al. (2005) proposed a version of the concordance that instead estimates

$$C^{\text{td}} = \mathrm{P}(\hat{S}(T_i \,|\, \mathbf{x}_i) < \hat{S}(T_i \,|\, \mathbf{x}_j) \,|\, T_i < T_j,\, D_i = 1),$$

which they showed is equivalent to a weighted ROC AUC over time. This metric is attractive because it provides a single score (without integrating over time) while still considering the time dependence of the estimates. Furthermore, for "one-to-one" corresponding models it is equivalent to the Harrell Jr et al. (1982) concordance index. In Paper II, we show that by using the concordance of Antolini et al. (2005) for hyperparameter tuning of Random Survival Forests (Ishwaran et al., 2008), we obtain better predictive performance than that obtained by the Harrell Jr et al. (1982) concordance, both in terms of discrimination and calibration.

Chapter 4

Summary of Papers

4.1 Paper I

Håvard Kvamme, Nikolai Sellereite, Kjersti Aas, and Steffen Sjursen. Predicting Mortgage Default Using Convolutional Neural Networks. *Expert Systems With Applications*, 102: 207–217, 2018.

This paper is concerned with predicting defaults of private mortgages at Norway's largest bank DNB. In contrast to similar work in the literature, we only use the state of each customer's checking account, savings account, and credit card, while disregarding all other information about the customers. Hence, the aim of the paper is twofold: to show that the transaction data holds information useful for estimating the risk of a mortgage, and how to extract this information.

No survival methodology was considered in this paper. Instead, the problem was approached as time-series classification. As was argued in Section 3.2.4, it is reasonable to apply a binary classifier to survival data if there are relatively few censored observations. Although this was not addressed in the paper, the binary classifier only considers mortgage defaults in the span of one year, meaning there was very little censoring.

The data set consists of the daily balances of the checking accounts, the savings accounts, and the credit cards, in addition to the daily number of checking account transactions and the total amount transferred to the checking account. For each customer, a full year of data was used as covariates. Convolutional networks were applied to the times series and it was found better to fit individual networks for each of the time series, rather than fitting a single network with all the time series as input. At the time of this project, it was not very common to apply convolutional networks to time series, and most literature instead considered recurrent networks for this purpose. We found, however, that the convolutions performed better than such recurrent networks, and in the years since, we have seen an increase in the use of convolutions for these types of problems.

The training set only consisted of 13,000 individuals. Hence, an augmentation technique was proposed. The scheme uses multiple "windows" of a customer's time series, and was shown to improve predictive accuracy substantially. In relation to this, empirical studies found that increasing the size of the training set would likely improve the performance of the classifier. This suggests that we were not able to take advantage of all the information available in the time series.

In conclusion, the proposed methodology was able to extract valuable information from the transaction time series. We do, however, believe that by combining this research with other data sources, further improvements would be possible.

4.2 Paper II

Håvard Kvamme, Ørnulf Borgan, and Ida Scheel. Time-to-Event Prediction with Neural Networks and Cox Regression. *Journal of Machine Learning Research*, 20(129): 1–30, 2019

In this paper, we propose new ways to apply neural networks to survival data. The task is approached by extending the well-known Cox regression. There are numerous papers on parameterizing a Cox regression with neural networks, but we distinguish us from them by proposing a model with non-proportional hazards. This is achieved by allowing for interactions between the covariates and the time. In particular, the hazard model is $h(t \,|\, \mathbf{x}) = h_0(t) \exp\left[g(t, \mathbf{x})\right]$, where $h_0(t)$ is a non-parametric function and $g(t, \mathbf{x})$ is parameterized by a neural network. This model is actually a *relative risk* model rather than a Cox model. Regardless, we refer to the method as *Cox-Time*, as the name is likely more familiar to most readers.

Training the proposed Cox-Time method with the Cox partial log-likelihood would be very computationally expensive. By instead using an approximation of the partial log-likelihood from nested case-control studies, we are able to fit the model in reasonable time. Prediction with the Cox-Time method can, however, still be quite computationally expensive. Some suggestions were proposed to alleviate the cost, but further research is warranted.

A proportional Cox model, fitted with the same approximation, is also proposed. The method is referred to as CoxCC or Cox-MLP (CC), where CC refers to the case-control approximation of the partial log-likelihood. The validity of the approximation was verified through simulation studies, and we found that, in general, very few control samples were needed.

The proposed methodology was compared with other methods from the literature in a study of five data sets. Four of the data sets were obtained from the survival literature, each consisting of 1,000 to 10,000 individuals. The fifth was created with data made available by the Kaggle KKBox Churn Prediction Challenge. Although not originally intended for time-to-event prediction, we were able to create a data set of more than 2 million individuals. As neural networks typically excel with larger data sets, we wanted to have at least one large data set in our experiments. Our proposed Cox-Time method was found to perform well compared to competing methods, in particular in terms of the Brier score and binomial log-likelihood, both weighted by the inverse probability of censoring.

4.3 Paper III

Håvard Kvamme and Ørnulf Borgan. Continuous and Discrete-Time Survival Prediction with Neural Networks. *Submitted for publication*, 2019

This paper is, in some ways, a continuation of Paper II. However, the semi-parametric Cox model is abandoned, and we instead direct our attention to fully parametric approaches that directly optimize the likelihood of right-censored event-times. The explored methods either assume a discrete time-scale or consider the time-scale to be divided into intervals. Consequently, we need to address the issue of how discrete-time models can be applied to continuous-time data sets. Two discretization schemes are considered; one that splits the time scale into equidistant intervals, and one that uses the quantiles of the estimated event-time distribution to split the time scale. To obtain continuous-time predictions from the discrete-time models, we consider two interpolation schemes; one that assumes a constant density function in each interval (linear interpolation of survival estimates), and one that assumes constant hazard rates (exponential survival function in each interval).

For smaller data sets, a coarse discretization grid can be beneficial as it reduces the number of parameters in the neural networks. This was verified in a simulation study. It was also found that the discretization grid based on quantiles was especially useful for the small data sets. Larger data sets, on the other hand, allows for finer discretization grids, meaning the differences between the two discretization and interpolation schemes decrease. It was, generally, found that the granularity of the discretization grid had a much larger impact on the predictive performance than the discretization and interpolation schemes.

The paper compares the two approaches to discrete-time modeling discussed in Section 3.2.2, i.e., the Logistic-Hazard and the PMF method. In terms of predictive performance, the two methods are very close, but we find that the Logistic-Hazard generally gives more stable performance, while the PMF method is more sensitive to hyperparameter configurations.

In this paper, we also propose a continuous-time method that assumes constant hazards in each interval. This is essentially the piecewise exponential model described in Section 3.2.3, but with modern neural networks and a softplus function replacing the regular exponential function in $h(\tau_j \mid \mathbf{x}) = \exp[\phi_j(\mathbf{x})]$. We find that its performance is competitive with the two other methods discussed in this paper, in addition to other methods found in the literature. In general, the methods described in this paper seem to perform slightly better than the Cox-Time methods proposed in Paper II.

4.4 Paper VI

Håvard Kvamme and Ørnulf Borgan. The Brier Score under Administrative Censoring: Problems and Solutions. *Submitted for publication*, 2019

This paper is an investigation of the Brier score, which is commonly used to evaluate survival predictions. In short, the Brier score is the mean squared difference between the survival estimates and the state of the individuals at a given time denoted $\mathbb{1}\{T_i^* > t\}$. For right-censored event-times, the Brier score can be weighted by the inverse probability of censoring, called IPCW. The paper shows that if the censoring times can be identified from the covariates, then the IPCW Brier score might be biased. Furthermore, this bias is equivalent to the bias of the binary classifiers in Section 3.2.4 and, therefore, benefits the binary classifiers. This is also verified through a simulation study where we show that a binary classifier can obtain better IPCW Brier scores than the true survival function. The paper also addresses that estimation of the censoring distribution can be problematic, especially if the distribution is dependent on the covariates.

Under administrative censoring, meaning all censoring times are observed, we propose the administrative Brier score which does not require estimation of the censoring distribution and does not have the bias of IPCW scores when the censoring times can be identified from the covariates. This is verified through a simulation study.

The KKBox churn prediction data set from Paper II is administratively censored and is, therefore, used as a case study in the paper. We find that the data set exhibits the traits we produced with simulations, meaning the covariates are likely to hold information about the censoring times of a subset of the customers. We show that the IPCW Brier score is substantially affected by the estimated censoring distribution. For the highest times, meaning the highest proportion of censored customers, binary classifiers outperform the Logistic-Hazard method. However, for the administrative Brier score, we get that the Logistic-Hazard performs better than the binary classifiers, as we would expect.

Chapter 5

Discussion

In this thesis, we have investigated how neural networks can be used to improve the predictive performance of existing survival methods. As this topic is still somewhat unexplored, there are almost limitless extensions that could be addressed here. I have tried to choose a relevant subset and instead provide some more details. I will, also, use the opportunity to discuss some of the weaknesses of the discussed methodology, in addition to my experiences working with each of the proposed methods. Instead of addressing each of the four papers individually, the following is a general discussion of the machine learning methodology for time-to-event prediction.

5.1 The Survival Methods

In Papers II and III, the predictions by various survival methods are compared using metrics such as the concordance and the Brier score. After working with numerous approaches to survival prediction, I would like to take the opportunity to address some of my experiences that are not as easily quantifiable and share some of my thoughts on the methods.

Random Survival Forests (RSF) by Ishwaran et al. (2008) is by far the simplest method to use. As with other Random Forest methods, very little preprocessing of the data is required (if any at all), it is easy to find a good set of hyperparameters, and as it is fully non-parametric, there are no numerical issues. This means that it is easy to get good and stable performance. However, in my experience, RSF is very slow to train on larger data set and, in general, it took many times longer to train than the neural networks. I do not know if this is because the method is very computationally expensive or if it was just the implementation I used that was slow.

The Cox-Time method generally performs better than the Cox methods with proportional hazards, but obtaining predictions is much more computationally expensive. This was discussed in Paper II.

Comparing the proportional CoxPH method and CoxCC method, where the first minimizes the negative partial log-likelihood as described in Section 3.2.1 and the latter minimizes an approximation of this negative partial log-likelihood, we found that CoxCC has slightly less stable performance than CoxPH. It is also worth mentioning that the implementation of CoxCC is more complicated than that of CoxPH. However, the CoxCC method has a validation loss that is more comparable with the training loss which gives better insights to the training process.

In Paper III, we found that the Logistic-Hazard and PC-Hazard were generally the two best-performing methods. Compared to Cox-Time, predictions are much less computationally expensive and, compared to all the semi-parametric Cox methods, the implementation is much simpler. In fact, considering the implementation, the Logistic-Hazard method is probably the simplest, as it requires only a few lines of code to implement it in a deep learning framework. Also, it is likely one of the more intuitive methods for those without a background in survival analysis. The downside of the Logistic-Hazard and PC-Hazard is that they require discretization of the time scale, which introduces additional hyperparameters. RSF and the Cox-Methods, on the other hand, all handle the continuous time non-parametrically.

5.2 Can We Trust Individual Predictions?

The methods discussed in Papers II and III give predictions in the form of survival estimates. These estimates were evaluated with multiple performance metrics such as the concordance index and the Brier score. However, these scores only verify that the survival estimates work well on average and do not address the individual predictions. As none of the methods produce uncertainty estimates, can we really trust the survival estimate for a single person?

To illustrate this concern, I have created a simple simulation study with constant and covariate independent hazard, meaning that all individuals have the same survival function. I then include a single covariate that is a monotone function of the administrative censoring time. This is supposed to illustrate a scenario where we have covariates that contain a lot of information about the censoring times, just as in Paper IV. If we fit a Cox model with proportional hazards to this data set, it would simply learn to disregard this covariate and give survival estimates close to the true survival function. The non-proportional methods discussed in Papers II and III, on the other hand, have more difficulties with this estimation.

In Figure 5.1, I have plotted the survival estimates of various methods, obtained by training on 1,000 samples from the simulation study. The vertical dotted red line in each plot gives the censoring time. Note that this censoring time represents the covariate used to obtain the survival predictions in each plot. The blue line in each plot represents the true survival function we want to estimate and, as it is independent of the censoring time, it is the same in all the six plots.

As expected, for times smaller than the censoring time (left of the vertical red line), all methods are able to estimate the survival function reasonably well. However, the methods clearly struggle with understanding that all individuals have the same survival function. As a result, the methods are not concerned with the survival estimation after the censoring time, as this is never considered in the likelihood. At least this is my interpretation of the results.

In a sense, these results are perfectly reasonable, as we should not predict for longer time horizons than what is available in the training set for a given set of covariates. However, usage of such methods clearly requires the practitioner to be aware of what time horizon that is applicable for each individual. This could

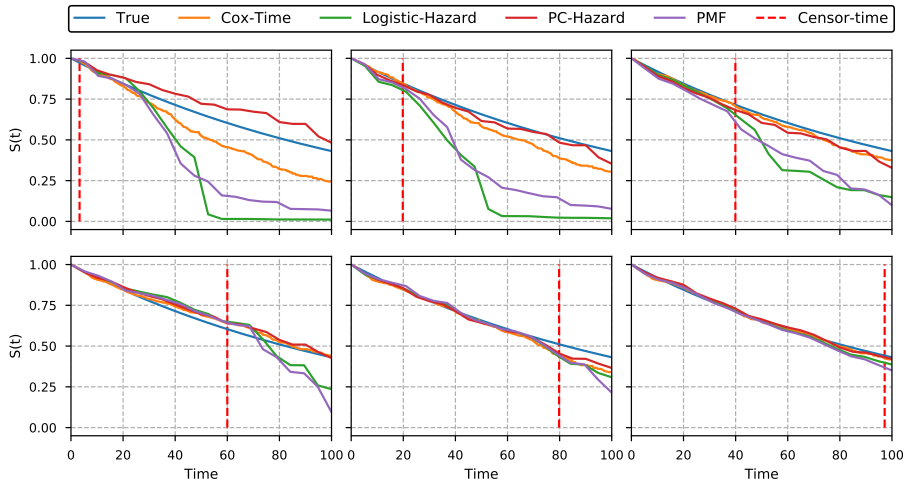

Figure 5.1: Comparison of survival estimates by various survival methods. The only covariate is a monotone function of the censoring time, which is represented by the vertical red dotted lines. The blue line is the true survival function, which is independent of the covariate. Hence, the true survival function is identical in all six plots.

be handled by also estimating the censoring distribution and only use survival estimates for times t that satisfies $\hat{S}_{C^*}(t \,|\, \mathbf{x}_i) > \xi$, where ξ is some positive threshold. Estimation of the censoring distribution is, in itself, a survival problem censored by the event times. This means that we need to know the time horizons that are reasonable for our censoring estimates.

The point I am trying to make is that one needs to be careful when using the individual survival estimates of these methods. Especially, as the methods can give survival predictions for much longer time horizons than what is reasonable for a particular individual. Uncertainty estimates would highly benefit these methods, though they may be hard to obtain. A reasonable starting point could be to consider estimates of the censoring distribution.

5.3 Extensions of the Methods

The methods discussed in Papers II and III are somewhat basic, in the sense that they only consider right-censored event times. DeepHit (Lee et al., 2018) is the exception as it also handles multiple event types. Natural extensions of the methods would be to account for other types of censoring and truncation (left, right, interval), competing risks, dynamic prediction (time-dependent covariates), recurrent events, and multistate models. These are all topics addressed in the statistical survival literature. In the following, we will investigate how a few of these extensions can be approached for the methods in Papers II and III. We will here use the same notation as presented in Chapter 3, meaning that, for individual i with covariates $\mathbf{x}_i$, the event time is T_i^*, the censoring time is C_i^*, and the potentially right-censored time is T_i with D_i denoting if it is an event or censored observation.

5.3.1 Left Truncation

Section 3.5 of the book by Klein and Moeschberger (2003) addresses how the survival likelihood changes with censoring and truncation of the types left, right, and interval. This is a good starting point for extending the methods that directly optimize the survival likelihood, such as Logistic-Hazard, the PMF method, and PC-Hazard. For the corresponding extension of Cox regression, see, e.g., Pan and Chappell (2002) and Rennert and Xie (2018). In the following, I will shortly address the case of left truncation, as it the most common of the truncation types.

If we have left-truncated data, our estimates will be conditioned on these left-truncation times. So, for ν denoting a left-truncation time, we can estimate $S(t \,|\, \nu) = \mathrm{P}(T^* > t \,|\, T^* > \nu)$. Note that this conditioning is similar to the hazard specification. This makes the hazard suitable for working with left-truncated data.

Consider a discrete time scale. In the same manner as in Section 3.1.3, we can derive the probability distribution of T and D, but now conditioned on the left-truncation time $\nu < t$,

$$\begin{aligned}
\mathrm{P}(T = t, D = d \,|\, T > \nu) &= \frac{\mathrm{P}(T = t, D = d, T > \nu)}{\mathrm{P}(T > \nu)} \\
&= \frac{\mathrm{P}(T = t, D = d)}{\mathrm{P}(T^* > \nu)\,\mathrm{P}(C^* > \nu)} \\
&= \left[\frac{f(t)^d\, S(t)^{1-d}}{S(\nu)}\right] \left[\frac{f_{C^*}(t)^{1-d}\,(S_{C^*}(t) + f_{C^*}(t))^d}{S_{C^*}(\nu)}\right] \\
&\propto \frac{f(t)^d\, S(t)^{1-d}}{S(\nu)}.
\end{aligned}$$

This means that the "regular" likelihood contribution of an individual (3.4), is scaled by the inverse of the survival probability at the left-truncation time. For an individual i with left truncation time ν_i, the discrete-time likelihood contribution is then

$$L_i = \frac{f(t_i \,|\, \mathbf{x}_i)^{d_i}\, S(t_i \,|\, \mathbf{x}_i)^{1-d_i}}{S(\nu_i \,|\, \mathbf{x}_i)}. \tag{5.1}$$

Written in terms of the hazard, this is equivalent to

$$L_i = h(t_i \,|\, \mathbf{x}_i)^{d_i}\, [1 - h(t_i \,|\, \mathbf{x}_i)]^{1-d_i} \prod_{j=\kappa(\nu_i)}^{\kappa(t_i)-1} [1 - h(\tau_j \,|\, \mathbf{x}_i)]\,,$$

where the τ_j's denote the discrete time scale, and $\kappa(t_i)$ is the index corresponding to time t_i, meaning $t_i = \tau_{\kappa(t_i)}$. These two ways of expressing the likelihood contribution can be used to extend the Logistic-Hazard and PMF method in Paper III to account for left-truncated event-times.

For continuous time, the likelihood contribution can still be expressed by (5.1), but with $f(t_i \,|\, \mathbf{x}_i)$ denoting the event-time density function instead of the probability mass function. Written in terms of the hazards, the continuous-time likelihood contribution is

$$L_i = h(t_i \,|\, \mathbf{x}_i)^{d_i} \, \exp\left[H(\nu_i \,|\, \mathbf{x}_i) - H(t_i \,|\, \mathbf{x}_i)\right].$$

This can be used to extend the PC-Hazard method in Paper III to account for left-truncated event times.

For Cox regression, the left-truncated event times can be handled by modifying the risk set in Section 3.2.1 to $\mathcal{R}_i = \{j : \nu_j < t_i \leq t_j\}$, as explained in the book by Aalen et al. (2008, p. 32). This should also work for the proposed Cox-Time and CoxCC in Paper II.

For all the methods discussed here, the survival functions are monotone. Hence, the estimates of $S(t \,|\, \mathbf{x})$ are dependent on the estimates up till time t. Left truncation can cause a data set to contain very few individuals that are followed from time zero, so if we estimate the survival from time zero, poor estimation at the lower times might ruin the later survival estimates. It is, therefore, more reasonable to estimate the conditional survival

$$S(t \,|\, \nu_0, \mathbf{x}) = \mathrm{P}(T^* > t \,|\, T^* > \nu_0, \mathbf{x}), \tag{5.2}$$

for a lower time ν_0 at which the set of individuals we follow, $\{i : \nu_i \leq \nu_0\}$, is of reasonable size.

5.3.2 Dynamic Prediction

The KKBox data set explored in Papers II and IV contains the subscription histories of KKBox's customers. We created a simple data set for predicting churn, with stationary covariates obtained from each customer's first subscription. This means that we considered the starting point $t = 0$ to be the time of the first subscription for each customer. In practice, however, it is probably more interesting to make dynamic predictions in *calendar time*, meaning that from a given calendar date t we want to predict the probability of churn for each time $t + v$. This is more in line with the original KKBox Kaggle competition, as they wanted churn predictions from a given date, and not from the time of each customer's first subscription. Explicitly, for time-dependent covariates $\mathbf{x}(t)$ and a given date t, we want to predict time v into the future, conditioned on all information up till time t. Using the notation of (5.2), we can express the conditional survival as

$$S(t + v \,|\, t, \, \{\mathbf{x}(u)\}_{u \leq t}) = \mathrm{P}(T^* > t + v \,|\, T^* > t, \, \{\mathbf{x}(u)\}_{u \leq t}).$$

One approach to this objective is to jointly model the event-time distribution and the distribution of the time-dependent covariates. While there is an extensive statistical literature on this topic (e.g., Ibrahim et al., 2010; Lawrence Gould et al., 2015), it requires estimation of the covariate process.

A more straightforward approach is that of *landmarking* (Van Houwelingen, 2007; van Houwelingen and Putter, 2011). In its simplest form, one creates K data sets, one for each landmarking time s_k, by truncating individuals with $T_i < s_k$, and fit individual models to each of the K data sets. This way, model k can predict

$$S(s_k + v \,|\, s_k,\, \mathbf{x}(s_k)) = \mathrm{P}(T^* > s_k + v \,|\, T^* > s_k,\, \mathbf{x}(s_k)),$$

where $\mathbf{x}(s_k)$ is the covariates at time s_k. This approach requires one model for each landmarking time s_k, which is problematic for a large K. It is, therefore, desirable to combine all these K models into a single model. This can be approached by conditioning on the landmarking time in the sense that we consider s_k a covariate. Also, it is probably better to let the covariates be some function of all previous information, ensuring that the covariate space is of constant dimensions p. We denote these covariates $\tilde{\mathbf{x}}(s_k) \in R^p$, as they are a function of the landmarking times and all previous information, giving

$$\tilde{\mathbf{x}}(s_k) = g\left(s_k, \{\mathbf{x}(u)\}_{u \le s_k}\right).$$

Partly conditional modeling (Zheng and Heagerty, 2005) closely resembles landmarking, but considers the residual time v instead of $t + v$. This means that we reset the time at every landmark s_k. For a set of landmarking times $\{s_k\}_{k=1}^{K}$, we substitute individual i by a set of "fake" individual i_k, one for each s_k, with covariates $\tilde{\mathbf{x}}_i(s_k)$ and observed time $v_{i_k} = t_i - s_k$. This means that we can fit the methods in Papers II and III with no alterations and their survival predictions will represent $S(v \,|\, s_k,\, \tilde{\mathbf{x}}(s_k))$. This approach was used by Martinsson (2016) who considered $v \,|\, s_k, \tilde{\mathbf{x}}(s_k)$ to be Weibull distributed, and constructed $\tilde{\mathbf{x}}(s_k)$ with a recurrent neural network over $\{\mathbf{x}(u)\}_{u \le s_k}$.

To my understanding, we are, technically, not restricted to use the same landmarking times for all individuals, and one could have a different set for each individual. However, I have not seen anyone address this in the literature.

By following the reasoning of landmarking or partially conditional modeling, it should be possible to extend the methods in Papers II and III to work for dynamic prediction. Note, however, that this section should be considered a starting point, as I have not tested any of the approaches, and there may be issues I have failed to address.

In Paper I, we proposed an augmentation scheme for artificially increasing the size of the mortgage default data set. In short, for each individual, we extracted data from multiple time periods in a similar manner to that of partial conditional modeling. However, the objective was to classify mortgage defaults within one year, and we only used covariates from one year of data. This means that, for each landmarking time s_k, we estimated $S(365 \,|\, s_k,\, \{\mathbf{x}(u)\}_{u=s_k-365}^{s_k})$, where the data $\{\mathbf{x}(u)\}_{u=s_k-365}^{s_k}$ was processed by a convolutional network. Paper I could benefit from a partial conditional modeling approach, as the specific time of each mortgage default holds more information than the binary labels, and building on an established approach is likely more reasonable than our ad hoc data augmentation scheme.

5.3.3 Competing Risks

In this thesis, I have only considered situations where there is only one event type for each data set. In some situations, there can, however, be multiple competing event types. For example, there are multiple diseases a patient can die from. We typically think of these types of events as competing processes where we only observe the event type with the shortest event time. The survival literature, therefore, refers to these situations as *competing risks.*

In the following, we will investigate possible extensions of the methods in Papers II and III to enable them to handle multiple event types. To this end, we need the random variable $R \in \{0, \dots, n_r\}$ for denoting the event type with n_r possible event types and $R = 0$ denoting a censored time. As before, the lower case letters represent the corresponding observations, meaning an observation consist of the tuple (t_i, r_i), and we let d_i be the event indicator $d_i = \mathbb{1}\{r_i > 0\}$.

The cumulative incidence functions

$$F_r(t \,|\, \mathbf{x}) = \mathrm{P}(T^* \leq t, R = r \,|\, \mathbf{x}).$$

are commonly reported in competing risk studies and we will consider the estimation of these functions our objective.

5.3.3.1 Logistic-Hazard

From the statistical literature on discrete-time survival analysis, we know that a natural extension of the Logistic-Hazard model is the Multinomial Logit Hazard model. The following is a derivation of the Multinomial Logit Hazard model, based on the presentation by Tutz and Schmid (2016).

The event-specific hazard at the discrete time τ_j is

$$h_r(\tau_j \,|\, \mathbf{x}) = \mathrm{P}(T^* = \tau_j,\, R = r \,|\, T^* \geq \tau_j, \mathbf{x}),$$

and the *overall* hazard is

$$h(\tau_j \,|\, \mathbf{x}) = \mathrm{P}(T^* = \tau_j \,|\, T^* \geq \tau_j, \mathbf{x}) = \sum_{r=1}^{n_r} h_r(\tau_j \,|\, \mathbf{x}).$$

This gives the survival function and event probability

$$S(\tau_j \,|\, x) = \mathrm{P}(T^* > \tau_j \,|\, \mathbf{x}) = \prod_{k=1}^{j} \left[1 - h(\tau_k \,|\, \mathbf{x})\right],$$
$$f_r(\tau_j \,|\, \mathbf{x}) = \mathrm{P}(T^* = \tau_j, R = r \,|\, \mathbf{x}) = h_r(\tau_j \,|\, \mathbf{x})\, S(\tau_{j-1} \,|\, \mathbf{x}).$$

The cumulative incidence functions can be expressed as

$$F_r(\tau_j \mid \mathbf{x}) = \sum_{k=1}^{j} f_r(\tau_k \mid \mathbf{x}). \tag{5.3}$$

For a function $\phi(\mathbf{x}) \in \mathbb{R}^{m \times n_r}$, such as a neural network with an output node for each combination of the time and the event type, we define the event-specific hazards as

$$h_r(\tau_j \mid \mathbf{x}) = \sigma_{jr}(\mathbf{x}) = \frac{\exp[\phi_{jr}(\mathbf{x})]}{1 + \sum_{k=1}^{n_r} \exp[\phi_{jk}(\mathbf{x})]},$$

and we let

$$\sigma_{j0}(\mathbf{x}) = \mathrm{P}(T^* > \tau_j \mid T^* \geq \tau_j, \mathbf{x}) = \frac{1}{1 + \sum_{k=1}^{n_r} \exp[\phi_{jk}(\mathbf{x})]},$$

denote the conditional survival. Note that $\sigma_{jr}(\mathbf{x})$ is the softmax function with the statistical convention of a reference class $\phi_{j0}(\mathbf{x}) = 0$.

Recall that $d_i = \mathbb{1}\{r_i > 0\}$. Disregarding $\mathbf{x}$ for simpler notation and assuming that the censoring time is independent of the event time (given covariates), the probability of T and R is

$$\begin{aligned}
&\mathrm{P}(T = t,\, R = r) \\
&= [\mathrm{P}(T^* = t, R = r)\,\mathrm{P}(C^* \geq t)]^{d}\,[\mathrm{P}(T^* > t)\,\mathrm{P}(C^* = t)]^{1-d} \\
&\propto \mathrm{P}(T^* = t, R = r)^{d}\,\mathrm{P}(T^* > t)^{1-d}.
\end{aligned}$$

This means the likelihood contribution of individual i is

$$\begin{aligned}
L_i &= f_{r_i}(t_i \mid \mathbf{x}_i)^{d_i}\, S(t_i \mid \mathbf{x}_i)^{1-d_i} \qquad (5.4)\\
&= h_{r_i}(t_i \mid \mathbf{x}_i)^{d_i}\,[1 - h(t_i \mid \mathbf{x}_i)]^{1-d_i} \prod_{j=1}^{\kappa(t_i)-1} [1 - h(\tau_j \mid \mathbf{x}_i)],
\end{aligned}$$

where $\kappa(t_i)$ denotes the index corresponding to time t_i, meaning that $t_i = \tau_{\kappa(t_i)}$. By introducing labels

$$y_{ijr} = \begin{cases} \mathbb{1}\{t_i = \tau_j, r_i = r\}, & \text{if } r > 0 \\ \mathbb{1}\{t_i > \tau_j\}^{d_i}, & \text{if } r = 0, \end{cases}$$

such that $\sum_{r=0}^{n_r} y_{ijr} = 1$, we can write the negative log-likelihood as

$$\text{loss} = -\sum_{i=1}^{n} \sum_{j=1}^{\kappa(t_i)} \sum_{r=0}^{n_r} y_{ijr} \log[\sigma_{jr}(\mathbf{x}_i)].$$

We recognizer this as the negative of the multinomial log-likelihood, also known as the *categorical cross-entropy*. This is one of the most commonly used loss functions for neural networks (e.g., for image classification) and should, therefore, be available in any deep learning framework.

5.3.3.2 PMF

For the discrete-time PMF method, Lee et al. (2018) have already proposed a competing risks version expressed by the event probabilities

$$f_r(\tau_j \,|\, \mathbf{x}) = \mathrm{P}(T^* = \tau_j, R = r \,|\, \mathbf{x}).$$

To my understanding, however, they assume $S(\tau_m \,|\, \mathbf{x}) = 0$. We will, therefore, investigate a version without this restriction. By considering observations on the discrete time scale $\tau_1, \ldots, \tau_m$, we simply let

$$\sum_{j=1}^{m} \sum_{r=1}^{n_r} f_r(\tau_j \,|\, \mathbf{x}) + S(\tau_m \,|\, \mathbf{x}) = 1,$$

with $S(\tau_m \,|\, \mathbf{x}) \in [0, 1]$. In the same manner as in Paper III, for $j \in \{1, \ldots, m\}$ and $r \in \{1, \ldots, n_r\}$, we let the risk-specific PMF be defined by the softmax

$$\begin{aligned} f_r(\tau_j \,|\, \mathbf{x}) &= \frac{\exp[\phi_{jr}(\mathbf{x})]}{1 + \sum_{k=1}^{m} \sum_{q=1}^{n_r} \exp[\phi_{kq}(\mathbf{x})]}, \\ S(\tau_m \,|\, \mathbf{x}) &= \frac{1}{1 + \sum_{k=1}^{m} \sum_{q=1}^{n_r} \exp[\phi_{kq}(\mathbf{x})]}. \end{aligned}$$

The survival function can then be written as

$$S(\tau_j \,|\, \mathbf{x}) = \sum_{k=j+1}^{m} \sum_{r=1}^{n_r} f_r(\tau_k \,|\, \mathbf{x}) + S(\tau_m \,|\, \mathbf{x}).$$

The negative log-likelihood corresponding to (5.4) can now be written as

$$\begin{aligned} \text{loss} = &- \sum_{i=1}^{n} d_i \log[f_{r_i}(t_i \,|\, \mathbf{x}_i)] \\ &- \sum_{i=1}^{n} (1 - d_i) \log\left[\sum_{k=\kappa(t_i)+1}^{m} \sum_{r=1}^{n_r} f_r(\tau_k \,|\, \mathbf{x}_i) + S(\tau_m \,|\, \mathbf{x}_i) \right]. \end{aligned}$$

This does not correspond to any commonly used loss function, meaning implementation requires some more work than for the competing risks version of the Logistic-Hazard. The cumulative incidence functions can be obtained from the risk-specific PMF $f_r(\tau_j \,|\, \mathbf{x})$ with (5.3).

5.3.3.3 PC-Hazard

For continuous time, the cumulative incidence functions take the form of the integral

$$F_r(t \,|\, \mathbf{x}) = \int_0^t f_r(u \,|\, \mathbf{x}) \, du = \int_0^t h_r(u \,|\, \mathbf{x}) \, S(u \,|\, \mathbf{x}) \, du. \tag{5.5}$$

The continuous-time PC-Hazard method in Paper III can be extended to competing risks by considering the cause-specific hazards. With $\kappa(t)$ now denoting the interval of time t, meaning that $t \in (\tau_{\kappa(t)-1}, \tau_{\kappa(t)}]$, the cause-specific hazard is constant in each interval

$$h_r(t \,|\, \mathbf{x}) = \eta_{\kappa(t)\, r}(\mathbf{x}).$$

The continuous-time survival function can be expressed as (Prentice et al., 1978)

$$S(t \,|\, \mathbf{x}) = \exp\left[- \sum_{r=1}^{n_r} \int_0^t h_r(u \,|\, \mathbf{x})\, du \right],$$

and, by following the same line of reasoning as in Paper III, we can write the survival function as

$$S(t \,|\, \mathbf{x}) = \prod_{r=1}^{n_r} \left(\exp[-\eta_{\kappa(t)\, r}(\mathbf{x})\, (t - \tau_{\kappa(t)-1})] \prod_{j=1}^{\kappa(t)-1} \exp[-\eta_{jr}(\mathbf{x}) \Delta\tau_j] \right),$$

where $\Delta\tau_j = \tau_j - \tau_{j-1}$. The continuous-time likelihood contribution, corresponding to (5.4), is then

$$\begin{aligned} L_i &= h_{r_i}(t_i \,|\, \mathbf{x}_i)^{d_i}\, S(t_i \,|\, \mathbf{x}_i) \\ &= \eta_{\kappa(t_i)\, r_i}(\mathbf{x}_i)^{d_i} \prod_{r=1}^{n_r} \left(\exp[-\eta_{\kappa(t_i)\, r}(\mathbf{x}_i)\, (t - \tau_{\kappa(t_i)-1})] \prod_{j=1}^{\kappa(t_i)-1} \exp[-\eta_{jr}(\mathbf{x}_i) \Delta\tau_j] \right), \end{aligned}$$

and the negative log-likelihood follows.

We can, alternatively, represent the likelihood as a Poisson-likelihood. This is achieved by considering independent Poisson-distributed variables with expectation $\mu_{ijr} = \Delta\tilde{t}_{ij} \eta_{jr}$, as described in Appendix C of Paper III.

5.3.3.4 Cox-Time

The Cox proportional hazards model can be extended to competing risks by considering the cause-specific hazard

$$h_r(t \,|\, \mathbf{x}) = h_{0r}(t) \exp[\phi_r(\mathbf{x})],$$

and the partial likelihood (Holt, 1978; Prentice et al., 1978)

$$\mathrm{PL} = \prod_{r=1}^{n_r} \prod_{i=1}^{n} \left(\frac{\exp[\phi_r(\mathbf{x}_i)]}{\sum_{j \in \mathcal{R}_i} \exp[\phi_r(\mathbf{x}_j)]} \right)^{\mathbb{1}\{r_i = r\}},$$

where $\mathcal{R}_i = \{j : t_j \geq t_i\}$ and $r_i = 0$ for censored times. Here $\phi(\mathbf{x}) \in \mathbb{R}^{n_r}$ can be parameterized by a neural network with an output node for each event type. The cause-specific baseline hazards can be obtained with the Breslow estimator (3.9), and the cumulative incidence functions can be expressed in terms of the risk specific hazard increments (Cheng et al., 1998). The Cox-Time method proposed in Paper II can be extended to competing risks in the same manner by considering

$$\mathrm{PL}_{\mathrm{CC}} = \prod_{r=1}^{n_r} \prod_{i=1}^{n} \left(\frac{\exp[\phi_r(t_i, \mathbf{x}_i)]}{\sum_{j \in \tilde{\mathcal{R}}_i} \exp[\phi_r(t_i, \mathbf{x}_j)]} \right)^{\mathbb{1}\{r_i = r\}},$$

where $\tilde{\mathcal{R}}_i \subset \mathcal{R}_i$.

In summary, it should be possible to make competing risks version of the methods in Papers II and III.

5.4 Evaluation of Survival Estimates

Neural networks can represent much more flexible models than classical statistical models, but this added flexibility can introduce new issues not really relevant for the simpler models. As an example, in Paper IV we discuss how the IPCW Brier score is biased for the KKBox churn data set. For simpler models than ours, this bias is likely not an issue, but by introducing neural networks, we find that these biases needed to be addressed.

A substantial part of machine learning research is concerned with empirical evaluation of methods on various data sets. To some degree, for prediction, it does not matter if your method is reasonable as long as it achieves good scores. Combining machine learning methodology with survival analysis is somewhat problematic, as censoring and truncation cause the test set to have incomplete observations. We typically account for censoring and truncation by making assumptions about their distributions and relationship to the event-time distribution. When these assumptions are essential for the evaluation metric, the choices made by the researcher may substantially affect the results. Phrased differently, we have missing data in our test set and the score accounts for the missing data by assuming something about its distribution. Different assumptions may lead to different results.

The IPCW Brier score in Paper IV can be biased if the covariates contain information that can identify some of the censoring times. Interestingly, the bias is in favor of the bias of a binary classifier that disregards individuals as they are censored. By choosing a method based on "best performance" in terms of the IPCW Brier score, we can end up choosing a binary classifier over a method that accounts for censored observations in a reasonable manner. For someone not familiar with survival analysis, the aforementioned binary classifier can seem to be a reasonable approach. To convince this audience to move to survival methodology, we need to be able to show the benefits of survival approaches. Rather than arguing that survival methodology is better suited for time-to-event prediction than binary classifiers, we need to provide evaluation metrics that verify this empirically.

5.4.1 The Concordance Index

Recall the that the concordance index is a metric for evaluating the discriminatory performance of survival estimates $\hat{S}(t \,|\, \mathbf{x})$. It aims at estimating the probability

$$C(t) = \mathrm{P}(\hat{S}(t \,|\, \mathbf{x}_i) < \hat{S}(t \,|\, \mathbf{x}_j) \,|\, T_i^* < T_j^*),$$

which can be somewhat problematic for right-censored observations. Uno et al. (2011) and Gerds et al. (2013) handle right censoring by inverse probability of censoring weighting (IPCW) of the concordance in a similar manner to that of the Brier score in Section 3.3.1. For administrative censoring with identifiable censoring times, this IPCW score might suffer from the same problems as the Brier score addressed in Paper IV. Therefore, a similar study to that of Paper IV is warranted and, depending on the findings, one could create an administrative concordance index by following the same line of reasoning as for the Brier score.

The concordance by Antolini et al. (2005), discussed in Section 3.3.2, estimates

$$C^{\mathrm{td}} = \mathrm{P}(\hat{S}(T_i \,|\, \mathbf{x}_i) < \hat{S}(T_i \,|\, \mathbf{x}_j) \,|\, T_i < T_j,\, D_i = 1),$$

which has the benefit of giving the score as a single number instead of a function of t. Unfortunately, the C^{td} is dependent on the censoring distribution, leaving the interpretation somewhat unintuitive. It would be desirable to find a scaling, such as IPCW, that accounts for the censoring distribution in a manner that lets us instead estimate $\mathrm{P}(\hat{S}(T_i^* \,|\, \mathbf{x}_i) < \hat{S}(T_i^* \,|\, \mathbf{x}_j) \,|\, T_i^* < T_j^*)$. If one could manage to create such a score, it would also be reasonable to investigate the impact of administrative censoring with identifiable censoring times.

5.4.2 On the Administrative Brier Score

The administrative Brier score, presented in Paper IV, is useful when the censoring times C_i^* can be identified from the covariates $\mathbf{x}_i$. This is because the IPCW Brier score requires a censoring distribution with $S_{C^*}(t \,|\, \mathbf{x}_i) > 0$, but with sufficient information about the censoring time, we approach $S_{C^*}(t \,|\, \mathbf{x}_i) = \mathbb{1}\{C_i^* > t\}$.

In short, the administrative Brier score $\mathrm{BS}_{\mathrm{A}}(t)$ disregards all individuals with a censoring time $C_i^* < t$. Note that we observe the censoring time for all individuals, meaning we remove both individuals that are censored and individuals that have experienced the event prior to time t.

For future survival prediction, it is problematic to use covariates that can predict the censoring time, as this essentially means that the covariates of future individuals are different from the covariates in our training set. Extrapolation is generally hard, and we want to avoid it if possible. It is, therefore, reasonable to remove covariates that are closely connected to the censoring times. If this substantially changes the predictions, one has at least learned something about the non-stationary behavior of the data set.

Note that if we are able to remove covariates closely related to the censoring time, we can use the IPCW Brier score instead, as we now have $S_{C^*}(t \,|\, \mathbf{x}_i) > 0$.

This will require estimation of the censoring distribution, but the interpretability of the IPCW score can be worth this disadvantage. If one prefers the IPCW scores, the administrative score can still be useful for understanding the data set and could be applied to verify that the IPCW scores are reasonable.

5.5 Time-to-Event Prediction and Machine Learning

I believe there is great potential in combining methodology from machine learning with that of survival analysis. This research field is relatively new, and the approaches discussed in this thesis are only concerned with a minor subfield of survival analysis. There are still parts of the survival literature that could benefit from machine learning methodology and at the point of writing there are plenty of "low-hanging fruits". For example, in this chapter, we have briefly addressed the topics of left-truncation, competing risks, dynamic prediction, and extensions of the concordance index.

Returning to the mortgage defaults in Paper I, this problem could benefit from a survival approach to the default modeling. Interestingly, there are actually multiple levels of delinquencies before a customer defaults on their mortgage. By considering a multistate model appropriate for the delinquencies, one could provide the model with more information than the simple binary responses of the defaults. Such multistate models have not been addressed in this thesis and would be an interesting direction for future research.

For the customer churn data sets in Papers II and IV, the customers can start and stop their subscription to KKBox multiple times. This means that individuals can move between two states (customer and non-customer) multiple times. This type of data is reasonable to consider with recurrent events models or multiple-spell modeling (Tutz and Schmid, 2016, Chapter 10). This is also a widely studied topic in survival analysis that has received limited attention with machine learning methodology.

A substantial part of machine learning research compares methods by empirical evaluation on a held-out test set. For event-time prediction, there are, however, very few large data sets and the current literature generally relies on somewhat small data sets. The resulting variance in the evaluation scores makes it hard to compare methods. In Paper II, we contribute to this issue by modifying the existing KKBox Kaggle data set, resulting in an event-time data set with more than 2 million individuals. We do, however, need more large data sets for the further advancement of the field.

I hope that the research I have been a part of can help to draw some attention to this field, and maybe inspire someone to further improve on the methodology for time-to-event prediction.

Bibliography

Odd Aalen, Ørnulf Borgan, and Håkon Gjessing. *Survival and Event History Analysis: A Process Point of View*. Springer Science & Business Media, 2008.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

Kate Allen. How a Toronto professor's research revolutionized artificial intelligence. *The Star*, Apr 2015.

Laura Antolini, Patrizia Boracchi, and Elia Biganzoli. A time-dependent discrimination index for survival data. *Statistics in Medicine*, 24(24):3927–3944, 2005.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems*, pages 153–160, 2007.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, volume 4. Austin, TX, 2010.

Elia Biganzoli, Patrizia Boracchi, Luigi Mariani, and Ettore Marubini. Feed forward neural networks for the analysis of censored survival data: a partial logistic regression approach. *Statistics in Medicine*, 17(10):1169–1186, 1998.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Charles C Brown. On the use of indicator variables for studying the time-dependence of parameters in a response-time model. *Biometrics*, 31(4):863–872, 1975.

S C Cheng, Jason P Fine, and L J Wei. Prediction of cumulative incidence function under the proportional hazards model. *Biometrics*, 54:219–228, 1998.

Travers Ching, Xun Zhu, and Lana X Garmire. Cox-nnet: An artificial neural network method for prognosis prediction of high-throughput omics data. *PLoS Computational Biology*, 14(4):e1006076, 2018.

François Chollet et al. Keras. https://keras.io, 2015.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Ronan Collobert, Samy Bengio, and Johnny Marithoz. Torch: A modular machine learning software library, 2002.

David R Cox. Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220, 1972.

Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The Helmholtz machine. *Neural Computation*, 7(5):889–904, 1995.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. Pervasive attention: 2d convolutional neural networks for sequence-to-sequence prediction. *arXiv preprint arXiv:1808.03867*, 2018.

David Faraggi and Richard Simon. A neural network model for survival data. *Statistics in Medicine*, 14(1):73–82, 1995.

Marco Fornili, Federico Ambrogi, Patrizia Boracchi, and Elia Biganzoli. Piecewise exponential artificial neural networks (PEANN) for modeling hazard function with right censored data. In *Computational Intelligence Methods for Bioinformatics and Biostatistics*, pages 125–136, 2014.

Michael Friedman. Piecewise exponential models for survival data with covariates. *The Annals of Statistics*, 10(1):101–113, 1982.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org, 2017.

Michael F Gensheimer and Balasubramanian Narasimhan. A scalable discrete-time survival model for neural networks. *PeerJ*, 7:e6257, 2019.

Thomas A Gerds and Martin Schumacher. Consistent estimation of the expected Brier score in general survival models with right-censored event times. *Biometrical Journal*, 48(6):1029–1040, 2006.

Thomas A Gerds, Michael W Kattan, Martin Schumacher, and Changhong Yu. Estimating a time-dependent concordance index for survival prediction models with covariate dependent censoring. *Statistics in Medicine*, 32(13):2173–2184, 2013.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323, 2011.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z Ghahramani, M Welling, C Cortes, N D Lawrence, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Erika Graf, Claudia Schmoor, Willi Sauerbrei, and Martin Schumacher. Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17-18):2529–2545, 1999.

Frank E Harrell Jr, Robert M Califf, David B Pryor, Kerry L Lee, and Robert A Rosati. Evaluating the yield of medical tests. *Journal of the American Medical Association*, 247(18):2543–2546, 1982.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015b.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

Geoffrey E Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning: Lecture 6a: Overview of mini-batch gradient descent, 2012.

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

Theodore R Holford. Life tables with concomitant information. *Biometrics*, 32 (3):587–597, 1976.

John D Holt. Competing risk analyses with special reference to matched pair experiments. *Biometrika*, 65(1):159–165, 1978.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Joseph G Ibrahim, Haitao Chu, and Liddy M Chen. Basic concepts and methods for joint models of longitudinal and survival data. *Journal of Clinical Oncology*, 28(16):2796, 2010.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

Hemant Ishwaran, Udaya B Kogalur, Eugene H Blackstone, and Michael S Lauer. Random survival forests. *Annals of Applied Statistics*, 2(3):841–860, 2008.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, Sep. 2009.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

Søren Johansen. An extension of Cox's regression model. *International Statistical Review*, 51(2):165–174, 1983.

Jared L Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. Deepsurv: personalized treatment recommender system using a Cox proportional hazards deep neural network. *BMC Medical Research Methodology*, 18(1):24, Feb 2018.

Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10): 947–954, 1960.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 971–980, 2017.

John P Klein and Melvin L Moeschberger. *Survival Analysis: Techniques for Censored and Truncated Data.* Springer, New York, 2. edition, 2003.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

Kevin J Lang. The development of the time-delay neural network architecture for speech recognition. *Technical Report CMU-CS-88-152*, 1988.

A Lawrence Gould, Mark Ernest Boye, Michael J Crowther, Joseph G Ibrahim, George Quartey, Sandrine Micallef, and Frederic Y Bois. Joint modeling of survival and longitudinal non-survival data: current methods and issues. report of the dia bayesian joint modeling working group. *Statistics in Medicine*, 34 (14):2181–2195, 2015.

Yann LeCun. Generalization and network design strategies. Technical Report CRG-TR-89-4, University of Toronto, 1989.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256. IEEE, 2010.

Changhee Lee, William R Zame, Jinsung Yoon, and Mihaela van der Schaar. Deephit: A deep learning approach to survival analysis with competing risks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Egil Martinsson. WTTE-RNN : Weibull Time To Event Recurrent Neural Network. Master's thesis, Chalmers University Of Technology, 2016.

Marvin Minsky and Seymour Papert. Perceptrons: An introduction to computational geometry. *MIT press*, 1969.

Abdelrahman Mohamed, George Dahl, and Geoffrey Hinton. Deep belief networks for phone recognition. In *Nips Workshop on Deep Learning for Speech Recognition and Related Applications*, volume 1, page 39. Vancouver, Canada, 2009.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

Radford M Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113, 1992.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Wei Pan and Rick Chappell. Estimation in the Cox proportional hazards model with left-truncated and interval-censored data. *Biometrics*, 58(1):64–70, 2002.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

Christopher Poultney, Sumit Chopra, Yann L Cun, et al. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems*, pages 1137–1144, 2007.

Ross L Prentice, John D Kalbfleisch, Arthur V Peterson Jr, Nancy Flournoy, Vern T Farewell, and Norman E Breslow. The analysis of failure times in the presence of competing risks. *Biometrics*, 34(4):541–554, 1978.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 873–880. ACM, 2009.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *arXiv preprint arXiv:1906.00446*, 2019.

Lior Rennert and Sharon X Xie. Cox regression model with doubly truncated data. *Biometrics*, 74(2):725–733, 2018.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

Jocelyn Sietsma and Robert JF Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–79, 1991.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

Gerhard Tutz and Matthias Schmid. *Modeling Discrete Time-to-Event Data*. Springer, 2016.

Hajime Uno, Tianxi Cai, Michael J Pencina, Ralph B D'Agostino, and L J Wei. On the c-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. *Statistics in Medicine*, 30(10):1105–1117, 2011.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Hans van Houwelingen and Hein Putter. *Dynamic Prediction in Clinical Survival Analysis*. CRC Press, 1st edition, 2011.

Hans C Van Houwelingen. Dynamic prediction by landmarking in event history analysis. *Scandinavian Journal of Statistics*, 34(1):70–85, 2007.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Safoora Yousefi, Fatemeh Amrollahi, Mohamed Amgad, Chengliang Dong, Joshua E Lewis, Congzheng Song, David A Gutman, Sameer H Halani, Jose Enrique Velazquez Vega, Daniel J Brat, et al. Predicting clinical outcomes from large scale cancer genomic profiles with deep survival models. *Scientific Reports*, 7(11707), 2017.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833, 2014.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657, 2015.

Yingye Zheng and Patrick J Heagerty. Partly conditional survival models for longitudinal data. *Biometrics*, 61(2):379–391, 2005.

Xinliang Zhu, Jiawen Yao, and Junzhou Huang. Deep convolutional neural network for survival analysis with pathological images. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 544–547, 2016.

Xinliang Zhu, Jiawen Yao, Feiyun Zhu, and Junzhou Huang. Wsisa: Making survival prediction from whole slide histopathological images. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6855–6863, 2017.

Papers

Expert Systems With Applications 102 (2018) 207–217

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Predicting mortgage default using convolutional neural networks

Håvard Kvamme[a,*], Nikolai Sellereite[b], Kjersti Aas[b], Steffen Sjursen[c]

[a] Department of Mathematics, University of Oslo, Niels Henrik Abels hus Moltke Moes vei 35, Oslo 0851, Norway
[b] Statistical Analysis, Machine Learning and Image Analysis, Norwegian Computing Center, Gaustadalleen 23a, Oslo 0373, Norway
[c] Group Risk Modelling, DNB ASA, Dronning Eufemias gate 30, Oslo 0191, Norway

A R T I C L E   I N F O

Article history:
Received 15 August 2017
Revised 17 February 2018
Accepted 18 February 2018
Available online 19 February 2018

Keywords:
Consumer credit risk
Machine learning
Deep learning
Mortgage default model
Time series

A B S T R A C T

We predict mortgage default by applying convolutional neural networks to consumer transaction data. For each consumer we have the balances of the checking account, savings account, and the credit card, in addition to the daily number of transactions on the checking account, and amount transferred into the checking account. With no other information about each consumer we are able to achieve a ROC AUC of 0.918 for the networks, and 0.926 for the networks in combination with a random forests classifier.

1. Introduction

The ability to discriminate bad customers from good ones is important for banks and other lending companies. A small improvement in prediction accuracy may result in a large gain in profitability. Early identification of high risk consumers may aid the prevention of loan defaults and help the consumers to better manage their personal economy.

In credit scoring, one builds a model for the correspondence between default and various loan obligor characteristics based on a relevant sample of people, and use this model to predict the probability that a person will repay his debts.

There is an extensive literature on credit scoring, both for assessing private loans (Butaru et al., 2016; Chi & Hsu, 2012; Khandani, Kim, & Lo, 2010; Sousa, Gama, & Brandão, 2016) and corporate loans (Jones, Johnstone, & Wilson, 2015; Ravi Kumar & Ravi, 2007). Some recent work include Abellán and Castellano (2017); Chen, Zhou, Wang, and Li (2017); Xia, Liu, Li, and Liu (2017), and Barboza, Kimura, and Altman (2017). For an overview and comparison of papers, see García, Marqués, and Sánchez (2014) and Lessmann, Baesens, Seow, and Thomas (2015).

All the papers above attempt to model delinquencies and defaults by applying machine learning algorithms to a set of explanatory variables. While there are variations in the information that is available to researchers (see e.g. Lessmann et al., 2015), the constructed explanatory variables tend to be quite similar. Papers typically use information from credit bureaus, such as number of outstanding accounts, delinquent accounts, and balance on other loans; individual account characteristics, such as current balance of the individuals accounts and monthly income; and demographic data, such as age and marital status. Butaru et al. (2016) also include macroeconomic variables, such as interest rates and unemployment statistics, as an attempt to make the delinquency model generalize better over longer periods of time.

As all these papers use similar explanatory variables, the researchers commonly explore differences between scoring models rather than the benefit of adding new explanatory variables. There are however some exceptions: Khandani et al. (2010) explore the benefit of adding information from detailed purchase volumes to their models. This includes travel expenses, gas station expenses, bar expenses, etc. Chi and Hsu (2012) also introduce consumer transaction data through an aggregated measure called average utilization ratio of credit.

In this paper we further investigate how transaction data can be used for credit scoring. In a joint research with Norway's largest financial service group, DNB, we use transaction data to predict mortgage defaults. In 2012, the average Norwegian made 323 card transactions, where 71% of the value transferred was through debit payments (Norges-Bank, 2012). Hence, transactional data may provide a useful description of user behavior, and subsequently consumer credit risk.

The transaction information consists of the daily balances on consumers' credit, checking, and savings accounts, in addition to

* Corresponding author.
E-mail addresses: haavakva@math.uio.no (H. Kvamme), nikolai.sellereite@nr.no (N. Sellereite), kjersti.aas@nr.no (K. Aas), sasjurse@gmail.com (S. Sjursen).

https://doi.org/10.1016/j.eswa.2018.02.029
0957-4174/© 2018 Elsevier Ltd. All rights reserved.

the daily number of transactions on the checking account, and the amount transferred into the checking account. Our dataset is thus a collection of time series for each customer. We do not use any of the common covariates mentioned above, as our goal is to investigate the value of the information available in the time series data. In a production setting, our model can be combined with existing risk models to improve the overall performance.

The idea behind this paper is to use deep learning, or deep neural networks (LeCun, Bengio, & Hinton, 2015), to predict mortgage defaults. Deep learning have had a dramatic impact in fields like image classification, text mining, and speech recognition. Common to these three fields is that the data is unstructured. Thus, the original data (pixels, letters, words, frequencies) needs to be transformed into meaningful covariates before they can be passed to a classical algorithm. Until recently, such tasks have been solved by the manual creation of informative covariates from the data. Deep neural nets, on the other hand, process the data in a sequential manner, learning multiple levels of abstraction. Hence, a deep net use data to "learn" how to create good explanatory variables for the problem at hand.

In this paper we apply a type of deep neural networks denoted convolutional neural networks (CNNs). To the extent of our knowledge, we are the first to apply CNNs to consumers' account balances to predict mortgage defaults. Transaction data has previously been used for credit scoring, see e.g. Khandani et al. (2010). However, most existing work relies on heuristic hand-crafted feature design. It is often hard for us humans to figure out appropriate features or covariates for credit scoring. Hence, the purpose of this study is to use a convolutional neural network to automate feature engineering from the raw time series, in a systematic way. The learned features resulting from such a model may be viewed as the higher level abstract representation of the low level raw time series signals.

There have been some successful attempts applying deep neural nets to time series data in other application areas. In the field of human activity recognition, Ordóñez and Roggen (2016); Yang, Nguyen, San, Li, and Krishnaswamy (2015), and Hammerla, Halloran, and Ploetz (2016) use sensor data to recognize activities improving the state-of-the-art. Cui, Chen, and Chen (2016); Prasad and Prasad (2014); Zheng, Liu, Chen, Ge, and Zhao (2014), and Le Guennec, Malinowski, and Tavenard (2016) present network structures for time series classification across multiple domains. Also, Sirignano, Sadhwani, and Giesecke (2016) assess mortgage risk using a deep net on 294 explanatory variables. However, on the contrary to our work, they use multilayer perceptrons instead of CNNs, and the majority of the variables are loan-level feature and performance variables. The deep net is mainly used to identify complex interactions between the input variables.

Much work in credit scoring is related to regulatory frameworks such as the Basel Accord. The Basel II regulations require transparency in the loan-granting process. Due to their non-linear structure, CNNs are usually considered as black boxes. That is, it is usually not obvious what exactly makes them arrive at their predictions. However, since this lack of transparency in many applications is considered a major drawback, the development of methods for explaining and interpreting deep learning models has recently attracted increased attention, see e.g. Lundberg and Lee (2016); Samek, Wiegand, and Müller (2017); Shrikumar, Greenside, Shcherbina, and Kundaje (2016), and Ribeiro, Singh, and Guestrin (2016). Hence, it is not obvious that CNNs will remain inappropriate for building regulatory models in the future. Meanwhile, a credit scoring system based on a CNN may be useful in many other applications. Financial institutions may e.g. use them in their own internal risk estimation or as part of a validation exercise (Pillar 2 of the Basel framework).

A lender commonly makes two types of credit decisions: first, whether to grant credit to a new applicant, and second, how to deal with existing applicants, including whether to increase their credit limits. The first problem is denoted application scoring and the latter behavioral scoring. The majority of previous work focus on application scoring, while prior work on behavioral scoring is much less developed, see e.g. Thomas (2000) and Kennedy, Mac Namee, Delany, O'Sullivan, and Watson (2013). The methodology proposed in this paper may be used both for application and behavioral scoring. Consumer transaction histories contain implicit repayment behavior, making them suitable for behavior scoring. However, as the new Revised Payment Service Directive (PSD2) becomes implemented across the EU and the European Economic Area during 2018, the banks (and other lending institutions) will have access to transaction data even for new customers. Hence, this kind of data may also be used for application scoring.

The remainder of the paper is organized as follows. In Section 2 we describe our dataset and how it was processed before fitting the neural networks. In Section 3 we introduce convolutional neural networks, and show how they were applied to our problem. In Section 4 we evaluate the performance of the proposed algorithms. Finally, in Section 5, we summarize our findings.

2. Data

This study is a collaboration with the largest Norwegian financial service group, DNB, using data from their banking services. The dataset consists of a sample of 20,989 customers who either previously had a mortgage, or got approved for a mortgage at some point between January 2012 and April 2016. For every customer we have the daily balances on their checking account, savings account, and credit card, in addition to the daily number of transactions on the checking account. We also know the daily amounts that are transferred into the checking account (e.g. from salary, payments from friends, etc.). Hence, we have a set of five time series for each customer. Finally, adding the sum of checking account, savings account, and credit card as a new time series, we end up with a total of six series. A summary of the six series can be found in Table 1.

For a customer to be granted a mortgage by DNB, he or she is required to open a checking account at the bank, given that this criterion is not already satisfied. There are however no requirements as far as savings accounts and credit cards are concerned. As a result, there are many missing series in the dataset.

Our definition of default is the one used in Basel II, i.e. that the obligor is past due for more than 90 days on the obligation to the bank. Housing prices in Norway have generally increased steadily since 2003, and thus, the mortgage market has seen few defaults (Finanstilsynet, 2016). The lack of defaults poses a problem due to the fact that our algorithms need large datasets to generalize well. In addition to this, the large class imbalance (ratio between non-defaults and defaults) is commonly regarded as a problem for classification algorithms. Both issues are addressed in later sections.

2.1. Training and test set

We divide our data into a training set and a test set. For training, we extract transaction histories from the time period December 31, 2012 to December 31, 2013 and use the outcome (default / non-default) in the period from January 1, 2014 to January 1, 2015 as response variables. A subset of the training set was held out from training and used as a validation set for tuning of our algorithms. For testing, we extract transaction histories from the time period February 28, 2014 to February 28, 2015, and response variables from the period March 1, 2015 to March 1, 2016. There are no customers who appear in both the training and test set, meaning that our study is both "out-of-time" and "out-of-sample". Cus-

Table 1
Time series.

Series	Abbrev.	Explanation
Checking account	ch	Balance on the checking account.
Savings account	sa	Balance on the savings account.
Credit card	cc	Balance on the bank issued credit card.
Checking transactions	trch	Number of transaction on the checking account.
Into checking	in	Sum of transactions into the checking account.
Sum	sum	Sum of series ch, sa, and cc.

The time series used in this paper. Each time series consists of values aggregated to a daily level.

Table 2
Data proportions for training, validation, and test sets.

Dataset	Defaults	Non-defaults	Total	Default type
Training	1298	11,398	12,696	90 or 60 days past
Training augmented	95,647	841,247	936,894	90 or 60 days past
Validation	329	6043	6372	90 or 60 days past
Test	96	1825	1921	90 days past

tomers for whom we did not have a full year of transaction data, were removed both from the training and the test set. However, we evaluate the performance of our methods on customers with missing data in Section 4.5. For a further specification of the datasets, see Table 2.

2.2. Preprocessing time series

Convolutional neural networks require the input to be scaled. Hence, for images it is common to normalize the pixels to lie in the range [0, 1] (Goodfellow, Bengio, & Courville, 2016). Pixels however have the benefit of all being in [0, 255] and somewhat evenly distributed in this interval. On the contrary, there are great differences in the magnitude of the accounts. If we scale all series based on the most extreme account values, most of the series will have very small variations, making it hard for our neural net to learn from the data. Therefore, the time series were scaled such that each individual series is in the range [0, 1] through

$$\mathbf{x} = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})}, \tag{1}$$

where $\mathbf{x}$ denotes *one* time series. With this scaling, the neural net receives inputs of similar magnitude. However, this comes at the cost of removing the magnitude of the individual series. Hence, the network can only find information in the relative patterns of the series, and has no knowledge about the actual size of the different accounts. Missing series were treated as accounts without any movements.

3. Methods

While there is an extensive literature on time series classification, we have chosen to focus only on the subset of algorithms that falls under deep learning. For a review of other state-of-the-art methods, see e.g. Bagnall, Lines, Bostrom, Large, and Keogh (2016).

In the following, we will first, in Section 3.1, introduce convolutional neural networks and how they were applied to our credit scoring problem. Then, in Section 3.2, we will discuss methods used to avoid overfitting and how we approached the class imbalance problem.

3.1. Convolutional neural networks

Deep learning arguably owes much of its success to its extent of modularity. The framework is based on combining differentiable transformations, where each such transformation is commonly referred to as a *layer*. The way one organizes the layers is called the *network architecture*. When fitting such a network to a dataset, or performing the “training” as it is denoted in machine learning, it is necessary to define a loss function that can be optimized. The loss function does not necessarily need to be the objective we actually want to optimize, but rather a function that indirectly optimizes our true goal (Goodfellow et al., 2016). For binary classification it is common to use the binary cross entropy

$$\text{Loss} = -\sum_i \{y_i \log p_i + (1 - y_i)\log(1 - p_i)\}, \tag{2}$$

where y_i denotes the true class label as {0, 1}, and p_i denotes our prediction for customer i in [0, 1]. Eq. (2) is the same as the negative log likelihood used in logistic regression. Note that the loss decreases as the p_i’s move closer to the y_i’s, hence reducing the loss should improve our objective.

During training the data is passed through the network, the loss is calculated, the gradients are computed with respect to all the parameters in the network, and the parameters are optimized through some version of gradient descent. The gradients can be obtained in a nice way using backpropagation (LeCun, Bottou, Orr, & Müller, 1998), which basically is application of the chain rule for differentiation. Backpropagation calculates the gradients in a sequential manner, such that the gradients of each transformation can be derived solely from the error passed back from the succeeding layer. The main takeaway here is that layers can be implemented independent of each other, simplifying the process of combining them in an architecture.

Convolutional neural networks (CNNs) constitute the subset of neural networks that contain convolutional layers. However, they typically contain other layers as well. The following sections describe the different types of layers that have been used in our neural networks.

3.1.1. Convolutional layers

The first two layers of our network are presented in Fig. 1. The left block shows an input time series that is 365 days long. For now, assume that the dataset only consist of a single time series per customer (instead of six).

A convolutional layer consists of multiple *filters* that are applied to the input series. A filter is simply a vector (or matrix) consisting of weights that need to be optimized. The first convolutional layer has filters of length 9, as shown in the figure. The application of such a filter to the input series is just a sum over the element-wise product of the filter weights and the time series, resulting in

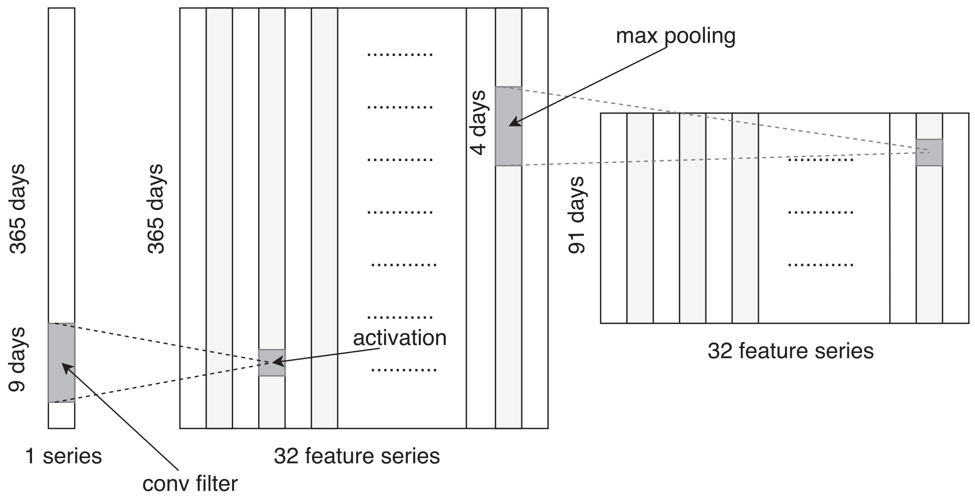

Fig. 1. The first convolutional layer and first pooling layer of our model. The left block is the input series. The gray area marks the part of the series for which a convolutional filter is currently being applied. The middle block shows the convolved features (or activations) resulting from the 32 different convolutional filters. The right block shows the result of applying max pooling to the middle block.

a convolved feature (or activation). A convolved feature is therefore a measure of the correlation between the filter and the relevant part of the input. The middle block of Fig. 1 displays all the convolved features from the input, using 32 different filters. Each filter is slided across the input to generate as many convolved features as the length of the time series[1]. Due to the fact that the 32 filters are randomly initialized, they end up extracting different information from the input.

After applying the filters to a time series, the convolved features are passed through a ReLU transform, which is simply $\max\{0, x\}$. Such *activation functions* are applied to make the transformation non-linear, and they are commonly considered a part of the convolutional layer rather than a separate layer. See Gu et al. (2017) for a discussion of other activation functions.

3.1.2. Max pooling layers

A convolutional layer preserves the resolution in the temporal dimension. There can however be some benefits of reducing the temporal resolution, such as reducing the number of parameters in the next layer, and introducing some shift-invariance (Gu et al., 2017). As a result, many CNNs include so-called max pooling layers, which simply replace a set of values with their maximum. Note that this layer does not introduce any additional parameters.

We use max pooling after each convolutional layer. An example is shown to the right in Fig. 1, where four values of each column of the convolved features are pooled into one. As shown in the figure, pooling is applied separately for each feature series. Thus, the pooling layer preserves the number of feature series, while reducing the temporal resolution. See Gu et al. (2017) for a discussion of other pooling layers.

3.1.3. Fully connected layers

To perform the actual classification, it is necessary to combine the convolved features into a binary classifier. This is typically done through one or more *fully connected* (or *dense*) layers, which form a multilayer perceptron (a classical neural net). A dense layer requires a one-dimensional input, so the columns of the last layer are concatenated into a single vector x. The transformation is then simply the inner product with a weight matrix, $z = x^T W$, with some non-linear activation function (typically ReLU). The number of columns in W determines the dimension of the output. Our final layer has only two outputs, one for each class, and therefore a softmax activation function is applied to ensure that the predictions are in the interval [0, 1].

$$\text{Softmax}(z)_c = \frac{e^{z_c}}{\sum_j e^{z_j}}, \tag{3}$$

where c and j refers to classes. Note that the softmax can be applied for an arbitrary number of classes, but for binary classification it it equivalent to the logit function. As a result, in combination with our choice of loss function, this final layer is equivalent to a logistic regression.

3.1.4. Network architecture and training

In our network architecture we alternate between a convolutional layer and a max pooling layer two times, followed by two fully connected layers. For an input of length 365 days, this results in 199,234 weights that need to be optimized. The whole structure is displayed in Fig. 2, were we also show the length of the filters and number of filters for the convolutions, the size of the pooling kernels, and the number of output nodes in the dense layers. This architecture was developed manually by evaluating on a validation set[2], and the hyper parameter search was therefore not as rigorous as with e.g. a grid search. However, it was found that a variety of model configurations resulted in quite similar performance. In particular, further increasing the number of nodes, filters, or layers did not really seem to affect the performance.

We train one network for each of our six time series, completely independently, using the Adam optimization algorithm (Kingma & Ba, 2014) with default parameters and batch size of 512. The resulting six predictions are averaged to give a final prediction

[1] Zero-padding is used to preserve the temporal dimension (Goodfellow et al., 2016).

[2] Note that the validation set is not the same as the test set.

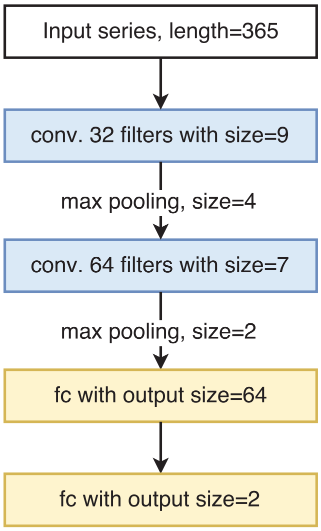

Fig. 2. Our convolutional neural net. The blue layers are convolutional layers and the yellow layers are fully connected layers. All activations are ReLU, with the exception of softmax in the final layer. Dropout, with rate 0.5, is performed between the last two layers. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

for each customer. When computing the averages, predictions corresponding to missing series are discarded. We also experimented with averaging the log-odds instead of the predictions, which resulted in a slight performance decrease.

We also tried an alternative model where we train a network on all six series simultaneously. Thus, the input is 365×6 rather than 365×1, meaning that the first convolutional filter will have size 9×6, rather than 9×1. Apart from this, the architecture is identical to the architecture for the individual series. By combining all series in the input, the network has the potential to learn interactions between a customer's time series. However, as this model is more complex, it will require more data than the previous model.

In addition to the two models presented here, we have also tried deeper and more shallow models, as we will get back to in Section 4.2. In early stages of the project, we even experimented with recurrent neural networks in the form of Long Short-Term Memory networks (LSTM). They were found to be outperformed by the CNNs, in addition to being much more computationally expensive. Hence, they were dropped from further investigation.

3.2. Overfitting and class imbalance

In the following sections we explain the techniques we have used to prevent the models from overfitting and how we handled the large class imbalance between defaulting and non-defaulting customers.

3.2.1. Regularization

Deep neural networks are commonly overparameterized, making them prone to overfitting (Gu et al., 2017). Subsequently, regularizing techniques have been introduced to cope with these problems. Among the more common is the use of a validation set for monitoring the training process. We used this set to stop the gradient descent iterations when the net starts to overfit. This technique is commonly referred to as *early stopping*.

Another common form of regularization is *dropout* (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014), which simply sets activations to zero with a given probability during training. This prevents units in the network from co-adapting too much. At test time, the dropout layer scales the activations according to the dropout rate. We use dropout with rate 0.5 between the two last layers, but it can in practice be applied between any two layers.

See Goodfellow et al. (2016) and Gu et al. (2017) for a further review of regularization.

3.2.2. Data augmentation

The problem of predicting mortgage defaults is highly imbalanced in that the number of defaults is very small compared to the number of non-defaults. As a classifier's performance commonly deteriorates under imbalance, we under-sample the non-defaulting customers to make the two classes more similar in size. Moreover, to increase the number of defaults in the training set, we use delinquencies of 60 days in addition to those of 90 days. Hence, we ended up with a training set consisting of 12,696 customers, 10% of whom where characterized with default.

Deep neural nets commonly require more data than classical machine learning algorithms to perform well. As a result, data augmentation schemes have been explored in the literature (Goodfellow et al., 2016; Krizhevsky, Sutskever, & Hinton, 2012). Most of this work has been on images, and does not necessarily generalize well to other data sources. However, recently some attempts of performing data augmentation on time series have been proposed. Le Guennec et al. (2016) explore the possibilities of training a CNN in a semi-supervised manner using time series from other datasets. Cui et al. (2016) propose a sliding window technique where they split the time series in many overlapping slices, and train a classifier using each slice as an individual series. This is in some sense similar to other time series prediction tasks where the dataset originally consists of many overlapping series of data (e.g. Ordóñez & Roggen (2016)).

In this paper, we use a slight variation of the sliding windows approach of Cui et al. (2016). Our approach can be described as follows. For each defaulting customer we first define a training period which ends one month before the default, and starts two years earlier. If the customer did not exist in the dataset at the start of this period, the start was set to the earliest date the customer existed. We then extract the time series corresponding to the first 365 days of the training period and use that as an observation. By jumping a fixed number of days, called *stride* in the machine learning literature, we can get a new observation that is partially overlapping with the previous observation. This process is repeated until the end of our defined training period is reached. In this way, we get a training set consisting of multiple overlapping series from each defaulting customer. This augmentation scheme is illustrated in Fig. 3. The gray area shows the defined training periods for three different customers, and the red dots mark the time of their defaults. For id: 2 we show three of the observations (windows) that are added to the training set. Note that the first of the corresponding periods ends one year before the default occurs, while the last ends one month before. Thus both these observations, and all in between, fit our objective of predicting a default in the following 365 days.

We also need to perform a similar augmentation for the non-defaulting individuals. Our first experiments showed that sampling non-defaulting and defaulting customers from different time periods made our net learn different characteristics of these periods instead of the actual objective. Hence, to avoid this sampling bias, we had to sample the start and end dates for the non-defaulting customers from the ones of the defaulting customers.

Ideally one would use strides of 1 as this would give most diversity in the training data. However, our preliminary analysis

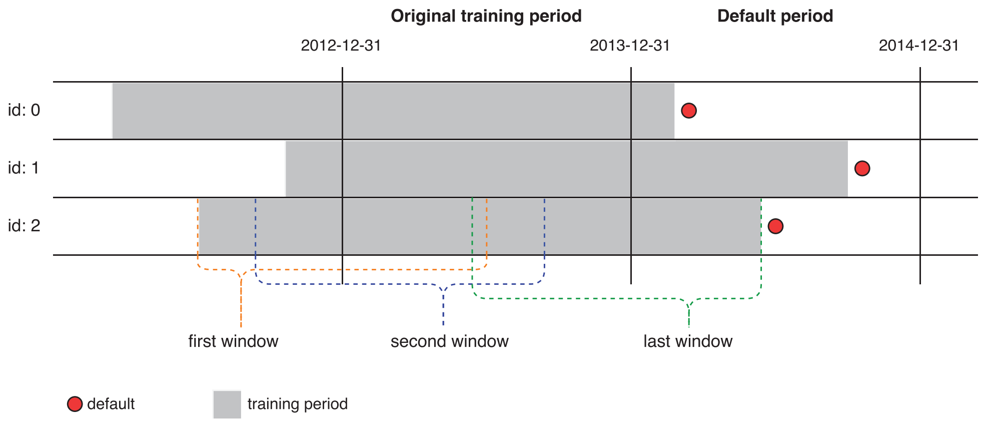

Fig. 3. Illustration of our data augmentation scheme. The gray area identifies the time period from which we extract data, and the red dots show the time of default. The sliding window approach is illustrated for id: 2. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

showed that strides of 1 did not give better results than strides of 5. Hence, since the memory usage of the latter is only a fifth of the first, we decided to use strides of 5 in our final analysis.

In Table 2 we have shown the size of our training set, the training set augmented with the sliding windows, the validation set, and the test set. The financial group DNB does not want to disclose its true default ratio. Hence, to generate the test set, 2000 customers were randomly sampled in such a way that the fraction of defaulting customers was 5%. After removing customers with missing history, we ended up with a test set of 1921 customers. The size of the test set is quite small, which results in larger variance in our results. Note however that we have experimented with different test sets from different time periods, and the results seem to be quite consistent.

Table 3
AUC for different CNN architectures.

Model	Mean AUC	Std AUC	AUC for average pred.
Full, 1 conv layer	0.8884	0.0055	0.8979
Full, 2 conv layers	0.8879	0.0041	0.8982
Full, 3 conv layers	0.8897	0.0048	0.9015
Indiv, 1 conv layer	0.9022	0.0022	0.9043
Indiv, 2 conv layers	**0.9146**	**0.0025**	**0.9184**
Indiv, 3 conv layers	0.9106	0.0033	0.9148

AUC mean and standard deviation for 10 experiments for each of six different CNNs. "Full" refers to the approach with all time series as input, and "Indiv" to the approach where we compute the average of the predictions from six individually trained models. The rightmost column gives the AUC of the predictions averaged over the 10 trained models.

4. Experiments

In the following, we first present the measures used to evaluate the performance of our models. We then describe our experiments and their results.

4.1. Evaluation measures

The Receiver Operating Characteristic (ROC) curve is a common measure for evaluating a classifier's ability to separate two classes. The ROC curve is a plot of the true positive rate (for default) against the false positive rate, for all thresholds. A threshold is the chosen cut-off in the estimated scores from the neural net.

Since the ROC curve is based on the true positive rates and the false positive rates, it is invariant to class proportions. Moreover, it is not dependent on the quality of the predictions (probability estimates), only on the classifier's ranking capabilities.

ROC curves are often summarized through the area under the curve, or AUC. With perfect classification, the AUC will be 1, while random predictions result in an AUC of 0.5 on average.

AUC is a general measure of binary classification, and is widely used across disciplines. There are however more specialized measures for evaluating a credit scoring model. One such measure is the Expected Maximum Profit (EMP) (Verbraken, Bravo, Weber, & Baesens, 2014), which is a profit based performance measure accounting for the benefits generated by healthy loans and the costs caused by loan defaults. We have chosen not to use EMP in this paper. The main reason is that this measure is heavily dependent on the class proportions. As stated earlier, we have used a fictive default rate, since the bank did not want to disclose its true ratio. Hence, it does not make any sense to use a measure that significantly changes when the class proportions are altered.

Our convolutional neural network is initialized with random weights. Moreover, we use dropout by multiplying random Bernoulli variables with the input to the last layer, and the net is trained with stochastic gradient descent. As a result, the network will be slightly different every time it is trained. Hence, to control how our results are influenced by randomness, we repeat all experiments 10 times and report the average and standard deviation of the 10 resulting AUC values.

4.2. Competing architectures

In the following section, the two CNNs introduced in Section 3.1.4 are compared and evaluated. The architectures of the two models are identical, except that the first takes individual time series as input (*Indiv model*), while all six time series are input to the latter (*Full model*). Note that all training sets are augmented as described in Section 3.2.2. Summary statistics for the AUCs are shown in Table 3 under the names *Indiv, 2 conv layes* and *Full, 2 conv layers*. We see that the individual model is clearly better than the full model. This might indicate that the full model is not able to take advantage of interactions between the series, or at least that the added benefit is smaller than the downside of the added complexity of the model.

Table 4
AUC for individual series.

Series	Mean AUC	Std	NM Mean AUC	NM Std	Prop Missing
ch	0.8632	0.0040	0.8646	0.0036	0.0068
in	0.7116	0.0108	0.7140	0.0107	0.0068
cc	0.7636	0.0080	0.8099	0.0101	0.2405
sa	0.7902	0.0056	0.8178	0.0039	0.1463
sum	0.8752	0.0078	0.8752	0.0078	0
trch	0.8472	0.0064	0.8480	0.0063	0.0068
All	0.9146	0.0025	0.9146	0.0025	0

In the NM columns (Not Missing) we have removed customers having missing series or series without any activity. "Prop missing" gives the proportion of series removed from the test set in NM. The AUC is averaged over 10 experiments. The "All" series gives the score of the averaged predictions (Indiv model).

Table 5
AUC for different subsets of the training data.

Proportion	Mean AUC	Std AUC
1.00	0.9146	0.0026
0.90	0.9109	0.0048
0.75	0.9086	0.0054
0.50	0.9028	0.0033
0.25	0.8954	0.0082
0.10	0.8827	0.0062
Non-augmented	0.8880	0.0042

For each subset, we sample the customers from the training set, and use sliding window for data augmentation. The leftmost column shows the proportion of customers sampled from the training set, while the remaining columns show the average and standard deviation of AUC for the 10 repetitions. 1.00 gives the results for the full training set (Indiv model), while "Non-augmented" refers to using all customers, but with no data augmentation.

Table 6
AUC for different lengths of time series.

Model	Mean AUC	Std AUC	AUC for average pred.
365 days	0.9146	0.0026	0.9184
180 days	0.8945	0.0051	0.8981
91 days	0.8744	0.0051	0.8791
31 days	0.8425	0.0043	0.8460

The "Mean AUC" is averaged over 10 repetitions. The "AUC for average pred." gives the AUCs for the averaged predictions of the 10 repetitions of the same model. All models have the same architecture as the 365 days CNN.

Table 3 also includes four other models. The only difference between the models is the number of convolutional layers (and pooling layers), and whether the input consist of individual series or all six series simultaneously. For the full specification of the models, see Fig. A.1 in the Appendix. From the table we see that all the individual models have higher average AUC than the full models. Additionally, it seems that there is no reason to have more than two convolutional layers in the networks.

Averaging the predictions from six individually trained models has a regularizing effect. To check whether the differences in performance between the full models and the individual models are caused by this averaging, we also, for each model, compute the AUC for the average of the 10 predictions for each customer. The results are shown in the rightmost column of Table 3 ("AUC for average pred.") We see that this improves the AUC for all models. The individual models are still better than the full models, but the difference seems to be smaller.

In the remainder of this section we concentrate on the *Indiv model, 2 conv layers*, since this is the one that shows the best performance in Table 3.

Table 7
AUC for different length of series.

Model	Mean AUC	Std AUC	AUC for average pred.
RF 365 days	0.9130	–	–
RF 31 days	0.8907	–	–
Combined 365 days	0.9254	0.0011	0.9260
Combined 31 days	0.8941	0.0006	0.8946

RF is a random forests classifier with covariates extracted from the unscaled time series. Combined gives the averaged prediction of the CNN and the random forests classifier. The results for the combined classifier are averaged over 10 repetitions of the CNN. The "AUC for average pred." gives the AUC for the averaged predictions of the 10 repetitions of the CNN.

4.3. Importance of different series

Our model creates predictions based on six different time series: checking account balance (ch), amount transferred into checking account (in), credit card balance (cc), savings account balance (sa), the sum of checking, saving and credit card balances (sum), and number of transactions on the checking account (trch). To investigate how much information there is in each series, we report their individual AUC values in Table 4. As before, we report the results averaged over 10 experiments. The checking account (ch and trch) and the sum seem to be most informative, while the amount transferred into the checking account (in) provides less information.

In Section 2 it was stated that missing savings accounts and credit card accounts are regarded as accounts without any movements. This means that the comparison between the different series in the two leftmost columns of Table 4 might not be completely fair. Hence, we did a new experiment in which we removed all customers missing the relevant time series from the test set. The results are shown in columns 3 and 4 of the table (NM Mean and NM Std), while the proportions of removed customers is shown in column 5. From the table we see that the AUC for credit card accounts (cc) and savings accounts (sa) significantly increase, while there are small differences for the other series. This is reasonable, as the proportion of missing data is largest for the credit cards and savings accounts.

4.4. Training data

In this section we investigate how the size of the training set affects the performance of our classifier. This can help us understand to what extent more training data would improve our discriminator. We also evaluate the effect of our augmentation scheme.

Table 5 shows AUC for models trained on different subsets of the training data. The subsets were created by randomly sampling customers. For a given proportion of the full training set we train 10 models in the same way as previously for the full training set.

Investigating the table, the performance seems to be improved with the size of the training data. This suggests that with more data we might be able to obtain even better results than those presented in this paper.

The last row of the leftmost column in Table 5 shows the results of fitting the same model to the original (not augmented) training set. We see that using 10% to 25% of the original customers with augmented data results in the same performance as using all customers with non-augmented data.

4.5. Length of time series

To investigate to what extent lack of information affects the performance of our model, we have also fitted our CNN to series of different lengths, More specifically, we used the last month (31

Table 8
Evaluation metrics.

Model	Threshold	Accuracy	Sensitivity	Specificity	AUC	Brier Score	H-measure
CNN	0.3	0.954	0.374	0.985	0.915	0.0381	0.564
Combined	0.3	0.947	0.527	0.970	0.925	0.0364	0.592
LR	0.3	0.910	0.490	0.935	0.864	0.0458	0.455
MLP (512)	0.3	0.899	0.590	0.918	0.875	0.0550	0.484
RF	0.3	0.920	0.602	0.940	0.913	0.0386	0.557
CNN	0.5	0.953	0.064	0.999	0.915	0.0381	0.564
Combined	0.5	0.956	0.177	0.997	0.925	0.0364	0.592
LR	0.5	0.943	0.219	0.981	0.864	0.0458	0.455
MLP (512)	0.5	0.927	0.455	0.953	0.875	0.0550	0.484
RF	0.5	0.957	0.332	0.991	0.913	0.0386	0.557

Different evaluation metrics for two different threshold values. All results are based on the time series with length 365 days. For each model we have averaged the metrics over 10 trained models. CNN is the best-performing CNN from Table 3, LR is a logistic regression model, MLP is a Multilayer perceptron with one hidden layer and 512 nodes, RF is the random forest model, and Combined is the combined model in which we averaged the predictions from the CNN and the RF.

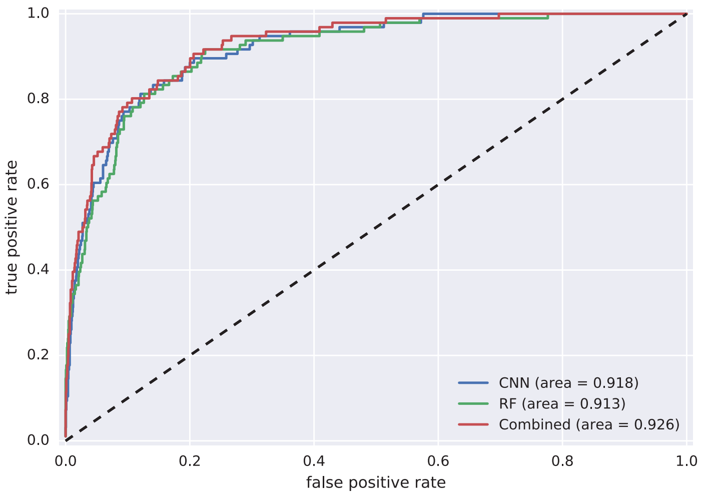

Fig. 4. ROC curves for the CNN, the RF, and their averaged predictions model. The time series are the full 365 day long series. The CNN comes from the averaged predictions over the 10 experiments.

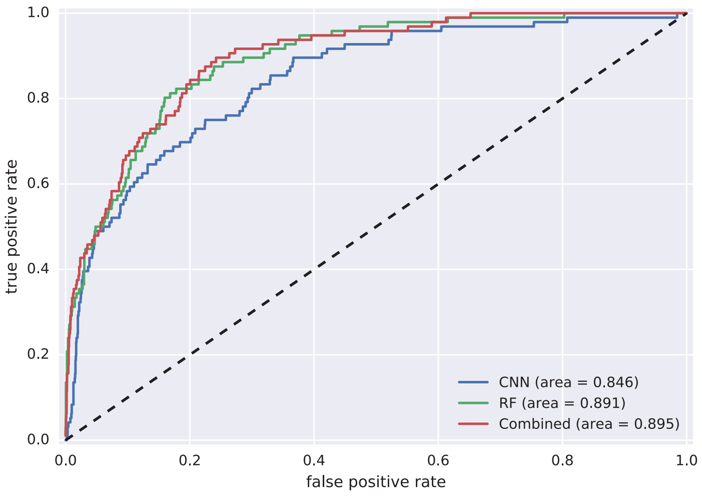

Fig. 5. ROC curves for the CNN, the RF, and their averaged predictions model. The time series used are contain only the last 31 days of data. The CNN comes from the averaged predictions over the 10 experiments.

days), the last quarter (91 days), and the two last quarters (182 days) of the original dataset to train the same model as previously.

The results are shown in Table 6. As can be seen, there is a clear drop in performance when the time series get shorter. On the downside, this makes it harder to evaluate customers with a short relationship to the bank. However, the results also suggest that it might be possible to increase the performance of our models by applying them to time series longer than 365 days.

4.6. Classical covariates

As described in Section 2.2, all time series are scaled to be in [0, 1] before fitting the CNN. This kind of scaling means that information about the magnitude of the series is lost. To assess the potential decrease in results connected to this removal of relevant information, we fit a random forests classifer (RF), consisting of 800 trees, to explanatory variables created from the original, unscaled time series. RF is often credited as a very strong classifier outperforming several alternative methods (including Support Vector Machines and Neural networks) when applied to credit scoring (Brown & Mues, 2012; Kruppa, Schwarz, Arminger, & Ziegler, 2013; Lessmann et al., 2015). For each of the six time series, we computed the mean, max, minimum, standard deviation, and the standard deviation scaled by the mean. These covariates were computed both for the full series and for the last month (31 days). In addition, we divided each of the covariates for the full series by the equivalent covariate calculated for the last month, producing a total of 15 features for each series. The resulting AUC is shown in the upper row of Table 7. We also fitted a logistic regression model and a Multilayer perceptron (with one hidden layer and 512 nodes) using the same explanatory variables. The AUC values for these models were significantly worse (0.86 and 0.88 see Table 8) than that for the RF, confirming the observations from previous studies. Hence, we decided not to include these models in the further comparisons.

Comparing the numbers in the upper rows of Tables 6 and 7 we see that for the 365 days period, the AUC for the CNN is slightly better than that for the RF. We also used a combined model in which we averaged the predictions from the CNN and the RF. From the third row of Table 6 we see that there is a small increase in performance compared to that obtained using one of the models only. The ROC curves for the CNN, the RF and the combined model are shown in Fig. 4. Running the diagnostic test for comparing ROC curves described in DeLong, DeLong, and Clarke-Pearson (1988) verifies that there are no significant differences between the AUCs for any pair of ROC-curves.

Table 8 gives the values of 6 evaluation metrics for each of the 5 models. In addition to Accuray, Sensitivity, Specificity and AUC, the Brier Score (Brier, 1950) and the H-measure (Hand, 2009; 2010) are reported. The AUC is sometimes criticised for treating the relative severities of misclassifications differently when different classifiers are used. The H-measure, on the other hand, may accommodate expert knowledge regarding misclassification costs, whenever that is available. We want to treat misclassifications of the smaller class as more serious than those of the larger class, since otherwise very little loss would be made by assigning everything to the larger class. Hence, we set the cost of misclassifying a bad customer as healthy to 0.95 and the cost of misclassifying a good customer as defaulter to 0.05.

We have also compared the three methods using data only from the last 31 days. The AUC-values are given in the last row of Table 6 and rows two and four in Table 7, respectively, while the corresponding ROC-curves are shown in Fig. 5. As can be seen from the tables and the figure, the performance of the RF does not degrade as much as that of the CNN when decreasing the length of the time series to 31 days. The *p*-value of the test for comparing the two ROC-curves is 0.005, meaning that the AUC for the RF is significantly higher than that for the CNN in this case.

5. Discussion

The ability to discriminate bad customers from good ones is very important for banks and other lending companies. There is a large literature on methods used to predict defaults of consumer loans. Status quo in both the industry and the academic literature is to use models with handcrafted explanatory variables. In contrast, we use a highly nonlinear convolutional neural network (CNN), where the input is raw account transactional data. More specifically, we have used daily balances of customers' checking accounts, savings accounts, and credit card accounts, in addition to daily number of transaction on the checking accounts, and amount transferred into the checking accounts to predict mortgage delinquency.

A CNN has a number of hyperparameters which need to be chosen, such as the number of layers, type of layers, and the type of regularization. Our best performing CNN with two convolutional and two fully connected layers achieved an AUC of 0.915. Considering the fact that our training set only consists of 12,696 customers, and that we do not use any other information than the account data, the results are quite promising.

In our analysis, we found that the AUC for our CNN was increasing with the size of the training set, suggesting that a larger dataset might result in even higher performance. In addition, the performance increased with the length of the time series, indicating that using a time series that is longer than 365 days may give better results. Comparing the predictions from a single CNN with ensemble predictions from 10 randomly initialized models showed that the increase in AUC using several model was limited.

By using transaction data only, one throws away a lot of other informative data that is typically used for credit scoring, e.g. the loan balance, socioeconomic data, payment history, and credit bureau data. Further, all time series are scaled to be in the interval [0,1] before fitting the CNN, meaning that information about the magnitude of the time series is lost. To account for the scaling, a random forests classifier was fitted to covariates extracted from the unscaled series, resulting in an AUC of 0.913. By combining the predictions of the CNN and the random forests, we achieved an AUC of 0.925.

Future research directions could include the combination of our model with other credit scoring models that use more classical credit information; applying our model on a larger dataset, as CNNs tend to perform better with more data; and using more granular data in the form of labeled transactions (e.g. groceries, gas, rent, vacation).

Acknowledgments

This work was supported by The Norwegian Research Council 237718 through the Big Insight Center for research-driven innovation. We thank Ørnulf Borgan for comments and suggestions on the paper. We also thank Sven Haadem for his help on creating a dataset from DNB's mortgage portfolio.

Appendix A

Fig. A.1.

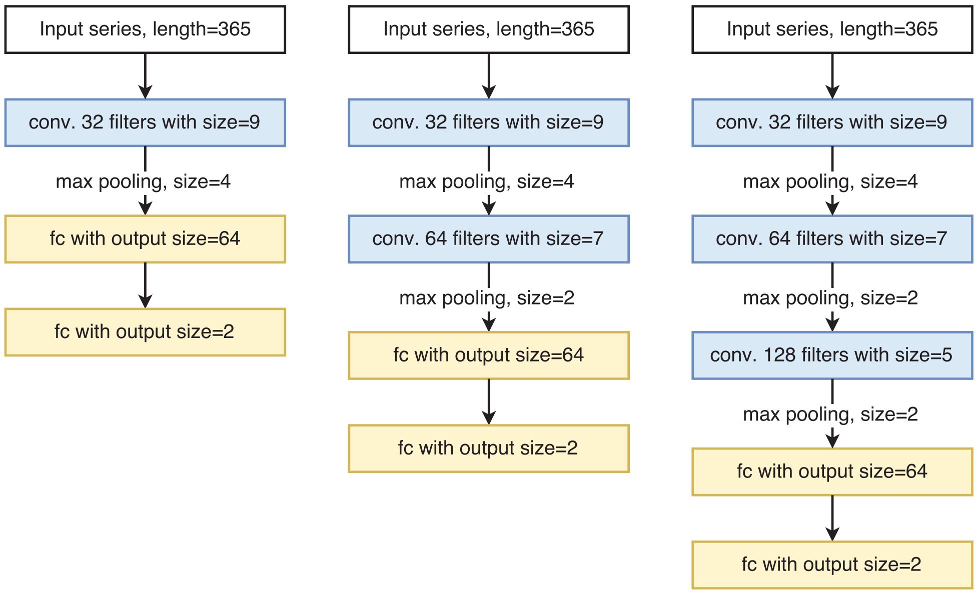

Fig. A.1. Our three convolutional neural net architectures for single time series. We also have a version of each model with 365 × 6 input. The blue layers are convolutional layers and the yellow layers are fully connected layers. All activations are ReLU, with the exception of softmax in the final layer. Dropout, with rate 0.5, is performed between the last two layers. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

References

Abellán, J., & Castellano, J. G. (2017). A comparative study on base classifiers in ensemble methods for credit scoring. *Expert Systems with Applications, 73*, 1–10. doi:10.1016/j.eswa.2016.12.020.

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2016). The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*. doi:10.1007/s10618-016-0483-9.

Barboza, F., Kimura, H., & Altman, E. (2017). Machine learning models and bankruptcy prediction. *Expert Systems with Applications, 83*, 405–417. doi:10.1016/j.eswa.2017.04.006.

Brier, G. W. (1950). Verification of forecasts expressed in terms of probability. *Monthly Weather Review, 78*(1), 1–3.

Brown, I., & Mues, C. (2012). An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications, 39*(3), 3446–3453.

Butaru, F., Chen, Q., Clark, B., Das, S., Lo, A. W., & Siddique, A. (2016). Risk and risk management in the credit card industry. *Journal of Banking & Finance, 72*, 218–239. doi:10.1016/j.jbankfin.2016.07.015.

Chen, X., Zhou, C., Wang, X., & Li, Y. (2017). The credit scoring model based on logistic-bp-adaboost algorithm and its application in p2p credit platform. In X. Li, & X. Xu (Eds.), *Proceedings of the fourth international forum on decision sciences* (pp. 119–130). Singapore: Springer Singapore.

Chi, B.-W., & Hsu, C.-C. (2012). A hybrid approach to integrate genetic algorithm into dual scoring model in enhancing the performance of credit scoring model. *Expert Systems with Applications, 39*(3), 2650–2661. doi:10.1016/j.eswa.2011.08.120.

Cui, Z., Chen, W., & Chen, Y. (2016). Multi-Scale Convolutional Neural Networks for Time Series Classification. *CoRR* http://arxiv.org/abs/1603.06995.

DeLong, E. R., DeLong, D. M., & Clarke-Pearson, D. L. (1988). Comparing the areas under two or more correlated receiver operating characteristic curves: A nonparametric approach. *Biometrics, 44*(3), 837–845.

Finanstilsynet (2016). Finansielt utsyn 2016. http://www.finanstilsynet.no/Global/Venstremeny/Rapport/2016/Finansielt_utsyn_2016.pdf. Accessed: 2017-02-06.

García, V., Marqués, A. I., & Sánchez, J. S. (2014). An insight into the experimental design for credit risk and corporate bankruptcy prediction systems. *Journal of Intelligent Information Systems, 44*(1), 159–189. doi:10.1007/s10844-014-0333-4.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. http://www.deeplearningbook.org.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., & Wang, G. (2017). Recent Advances in Convolutional Neural Networks. *CoRR* http://arxiv.org/abs/1512.07108.

Hammerla, N. Y., Halloran, S., & Ploetz, T. (2016). Deep, Convolutional, and Recurrent Models for Human Activity Recognition Using Wearables. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (pp. 1533–1540). New York, New York, USA: AAAI Press. http://dl.acm.org/citation.cfm?id=3060832.3060835.

Hand, D. J. (2009). Measuring classifier performance: A coherent alternative to the area under the ROC curve. *Machine Learning, 77*(1), 103–123. doi:10.1007/s10994-009-5119-5.

Hand, D. J. (2010). Evaluating diagnostic tests: The area under the ROC curve and the balance of errors. *Statistics in Medicine, 29*(14), 1502–1510. doi:10.1002/sim.3859.

Jones, S., Johnstone, D., & Wilson, R. (2015). An empirical evaluation of the performance of binary classifiers in the prediction of credit ratings changes. *Journal of Banking & Finance, 56*, 72–85. doi:10.1016/j.jbankfin.2015.02.006.

Kennedy, K., Mac Namee, B., Delany, S. J., O'Sullivan, M., & Watson, N. (2013). A window of opportunity: Assessing behavioural scoring. *Expert Systems with Applications, 40*(4), 1372–1380.

Khandani, A. E., Kim, A. J., & Lo, A. W. (2010). Consumer credit-risk models via machine-learning algorithms. *Journal of Banking & Finance, 34*(11), 2767–2787. doi:10.1016/j.jbankfin.2010.06.001.

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *CoRR* http://arxiv.org/abs/1412.6980.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 25* (pp. 1097–1105). Curran Associates, Inc. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Kruppa, J., Schwarz, A., Arminger, G., & Ziegler, A. (2013). Consumer credit risk: Individual probability estimates using machine learning. *Expert Systems with Applications, 40*(13), 5125–5131.

Le Guennec, A., Malinowski, S., & Tavenard, R. (2016). Data augmentation for time series classification using convolutional neural networks. In *Proceedings of the ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data ,Riva Del Garda, Italy*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. doi:10.1038/nature14539.

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (1998). Efficient backprop. *Neural Networks: Tricks of the Trade*, 9–50. doi:10.1007/3-540-49430-8_2.

Lessmann, S., Baesens, B., Seow, H.-V., & Thomas, L. C. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research, 247*(1), 124–136. doi:10.1016/j.ejor.2015.05.030.

Lundberg, S., & Lee, S. (2016). An unexpected unity among methods for interpreting model predictions CoRR abs/1611.07478.

Norges-Bank (2012). Årsrapport om betalingssystem 2012. http://static.norges-bank.no/pages/94894/Betalingssystem_2012_o.pdf?v=27052013101201&ft=.pdf. Accessed: 2017-02-09.

Ordóñez, F., & Roggen, D. (2016). Deep convolutional and LStm recurrent neural networks for multimodal wearable activity recognition. *Sensors, 16*(1), 115. doi:10.3390/s16010115.

Prasad, S. C., & Prasad, P. (2014). Deep Recurrent Neural Networks for Time Series Prediction. *CoRR* http://arxiv.org/abs/1407.5949.

Ravi Kumar, P., & Ravi, V. (2007). Bankruptcy prediction in banks and firms via statistical and intelligent techniques – A review. *European Journal of Operational Research, 180*(1), 1–28. doi:10.1016/j.ejor.2006.08.043.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why should i trust you?: Explaining the predictions of any classifier CoRR abs/1602.04938.

Samek, W., Wiegand, T., & Müller, K. (2017). Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models CoRR abs/1708.08296.

Shrikumar, A., Greenside, P., Shcherbina, A., & Kundaje, A. (2016). Not just a black box: Learning important features through propagating activation differences CoRR abs/1605.01713.

Sirignano, J., Sadhwani, A., & Giesecke, K. (2016). Deep Learning for Mortgage Risk. *ArXiv e-prints* http://adsabs.harvard.edu/abs/2016arXiv160702470S.

Sousa, M. R., Gama, J., & Brandão, E. (2016). A new dynamic modeling framework for credit risk assessment. *Expert Systems with Applications, 45*, 341–351. doi:10.1016/j.eswa.2015.09.055.

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

Thomas, L. C. (2000). A survey of credit and behavioural scoring: forecasting financial risk of lending to consumers. *International journal of forecasting, 16*(2), 149–172.

Verbraken, T., Bravo, C., Weber, R., & Baesens, B. (2014). Development and application of consumer credit scoring models using profit-based classification measures. *European Journal of Operational Research, 238*(2), 505–513. doi:10.1016/j.ejor.2014.04.001.

Xia, Y., Liu, C., Li, Y., & Liu, N. (2017). A boosted decision tree approach using Bayesian hyper-parameter optimization for credit scoring. *Expert Systems with Applications, 78*, 225–241. doi:10.1016/j.eswa.2017.02.017.

Yang, J. B., Nguyen, M. N., San, P. P., Li, X. L., & Krishnaswamy, S. (2015). Deep convolutional neural networks on multichannel time series for human activity recognition. In *Proceedings of the twenty-fourth international conference on artificial intelligence*. In *IJCAI'15* (pp. 3995–4001). AAAI Press.

Zheng, Y., Liu, Q., Chen, E., Ge, Y., & Zhao, J. L. (2014). Time series classification using multi-channels deep convolutional neural networks. *Lecture Notes in Computer Science*, 298–310. doi:10.1007/978-3-319-08010-9_33.

II

Journal of Machine Learning Research 20 (2019) 1-30 Submitted 6/18; Revised 7/19; Published 8/19

Time-to-Event Prediction with Neural Networks and Cox Regression

Håvard Kvamme haavakva@math.uio.no
Ørnulf Borgan borgan@math.uio.no
Ida Scheel idasch@math.uio.no
Department of Mathematics
University of Oslo
P.O. Box 1053 Blindern
0316 Oslo, Norway

Editor: Jon McAuliffe

Abstract

New methods for time-to-event prediction are proposed by extending the Cox proportional hazards model with neural networks. Building on methodology from nested case-control studies, we propose a loss function that scales well to large data sets and enables fitting of both proportional and non-proportional extensions of the Cox model. Through simulation studies, the proposed loss function is verified to be a good approximation for the Cox partial log-likelihood. The proposed methodology is compared to existing methodologies on real-world data sets and is found to be highly competitive, typically yielding the best performance in terms of Brier score and binomial log-likelihood. A python package for the proposed methods is available at `https://github.com/havakv/pycox`.

Keywords: Cox regression, customer churn, neural networks, non-proportional hazards, survival prediction

1. Introduction

In this paper, we consider methodology for time-to-event prediction, a part of survival analysis that reasons about when a future event will occur. Applications of time-to-event predictions can be found in a variety of settings such as survival prediction of cancer patients (e.g., Vigan et al., 2000), customer churn (e.g., Van den Poel and Lariviere, 2004), credit scoring (e.g., Dirick et al., 2017), and failure times of mechanical systems (e.g., Susto et al., 2015). Arguably, the field of survival analysis has predominantly focused on interpretability, potentially at some cost of predictive accuracy. This is perhaps the reason why binary classifiers from machine learning are commonly used in industrial applications where survival methodology is applicable. However, while the binary classifiers can provide predictions for *one* predetermined duration, one loses the interpretability and flexibility provided by modeling the event probabilities as a function of time. Furthermore, in time-to-event data, it is common that some individuals are not followed all the way to their event time, resulting in *censored* times rather than event times. While binary classifiers typically ignore these observations, one of the main objectives in survival analysis is to account for them. Hence, in applications with a substantial amount of censoring, the use of survival models tends to be advantageous.

In our work, we propose an approach for combining machine learning methodology with survival models. We do this by extending the Cox proportional hazards model with neural networks, and further remove the proportionality constraint of the Cox model. Building on methodology from nested case-control studies (e.g., Langholz and Goldstein, 1996) we are able to do this in a scalable manner. The resulting methods have the flexibility of neural networks while modeling event times continuously. Building on the PyTorch framework (Paszke et al., 2017), we provide a python package for our methodology, along with all the simulations and data sets presented in this paper.[1]

The paper is organized as follows. Section 2 contains a summary of related work. In Section 3, we review some basic concepts from survival analysis and introduce the Cox proportional hazards model with our extensions. In Section 4 we discuss some evaluation criteria of methods for time-to-event prediction. In Section 5, we conduct a simulation study, verifying that the methods we propose behave as expected. In Section 6 we evaluate our methods on five real-world data sets and compare their performances with existing methodology. We conclude in Section 7.

2. Related Work

The extension of Cox regression with neural networks was first proposed by Faraggi and Simon (1995), who replaced the linear predictor of the Cox regression model, cf. formula (3) below, by a one hidden layer multilayer perceptron (MLP). It was, however, found that the model generally failed to outperform regular Cox models (Xiang et al., 2000; Sargent, 2001). Katzman et al. (2018) revisited these models in the framework of deep learning and showed that novel networks were able to outperform classical Cox models in terms of the C-index (Harrell Jr et al., 1982). Our work distinguishes itself from this in the following way: The method by Katzman et al. (2018), denoted DeepSurv, is constrained by the proportionality assumption of the Cox model while we propose an extension of the Cox model where proportionality is no longer a restriction. In this regard, we propose an alternative loss function that scales well for both the proportional and the non-proportional cases.

Similar works based on Cox regression include SurvivalNet (Yousefi et al., 2017), a framework for fitting proportional Cox models with neural networks and Bayesian optimization of the hyperparameters, and Zhu et al. (2016) and Zhu et al. (2017) which extended the Cox methodology to images. Both Zhu et al. (2016) and Zhu et al. (2017) replace the MLP of DeepSurv with a convolutional neural network and applied these methods to pathological images of lung cancer and to whole slide histopathological images.

An alternative approach to time-to-event prediction is to discretize the duration and compute the hazard or survival function on this predetermined time grid. Luck et al. (2017) proposed methods similar to DeepSurv, but with an additional set of discrete outputs for survival predictions and computed an isotonic regression loss over this time grid. Fotso (2018) parameterized a multi-task logistic regression with a neural net that directly computes the survival probabilities on the time grid. Lee et al. (2018) proposed a method,

1. Implementations of methods and the data sets are available at `https://github.com/havakv/pycox`.

denoted DeepHit, that estimates the probability mass function with a neural net and combine the log-likelihood with a ranking loss; see Appendix D for details. Furthermore, the method has the added benefit of being applicable for competing risks.

The majority of the papers mentioned benchmark their methods against the random survival forests (RSF) by Ishwaran et al. (2008). RSF computes a random forest using the log-rank test as the splitting criterion. It computes the cumulative hazards of the leaf nodes and averages them over the ensemble. Hence, RSF is a very flexible continuous-time method that is not constrained by the proportionality assumption.

3. Methodology

In the following, we give a brief review of some concepts in survival analysis. For a more in-depth introduction to the field, see, for example, Klein and Moeschberger (2003).

Our objective is to model the event distribution as a continuous function of time. So with $f(t)$ and $F(t)$ denoting the probability density function and the cumulative distribution function of an event time T^*, we want to model

$$\mathrm{P}(T^* \leq t) = \int_0^t f(s)\, ds = F(t).$$

As alternatives to $F(t)$, it is common to study the *survival function* $S(t)$ and the *hazard rate* $h(t)$. The survival function is defined as

$$S(t) = \mathrm{P}(T^* > t) = 1 - F(t),$$

and is commonly used for visualizing event probabilities over time. For specifying models, however, it is rather common to use the hazard rate

$$h(t) = \frac{f(t)}{S(t)} = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \mathrm{P}(t \leq T^* < t + \Delta t \mid T^* \geq t).$$

If we have the hazard rate, the survival function can be retrieved through the cumulative hazard, $H(t) = \int_0^t h(s) ds$, by

$$S(t) = \exp[-H(t)]. \tag{1}$$

The survival function and the hazard rate therefore provide contrasting views of the same quantities, and it may be useful to study both.

Working with real data, the true event times are typically not known for all individuals. This can occur when the follow-up time for an individual is not long enough for the event to happen, or an individual may leave the study before its termination. Instead of observing the true event time T^*, we then observe a possibly right-censored event time $T = \min\{T^*, C^*\}$, where C^* is the censoring time. In addition, we observe the indicator $D = \mathbb{1}\{T = T^*\}$ labeling the observed event time T as an event or a censored observation. Now, denoting individuals by i, with covariates $\mathbf{x}_i$ and observed duration T_i, the likelihood for censored survival times is given by

$$L = \prod_i f(T_i \,|\, \mathbf{x}_i)^{D_i} S(T_i \,|\, \mathbf{x}_i)^{1-D_i} = \prod_i h(T_i \,|\, \mathbf{x}_i)^{D_i} \exp[-H(T_i \,|\, \mathbf{x}_i)]. \tag{2}$$

We will later refer to this as the *full likelihood.*

3.1. Cox Regression

The Cox proportional hazards model (Cox, 1972) is one of the most used models in survival analysis. It provides a semi-parametric specification of the hazard rate

$$h(t \,|\, \mathbf{x}) = h_0(t) \exp[g(\mathbf{x})], \qquad g(\mathbf{x}) = \boldsymbol{\beta}^T \mathbf{x}, \tag{3}$$

where $h_0(t)$ is a non-parametric *baseline hazard*, and $\exp[g(\mathbf{x})]$ is the *relative risk function.* Here, $\mathbf{x}$ is a covariate vector and $\boldsymbol{\beta}$ is a parameter vector. Note that the linear predictor $g(\mathbf{x}) = \boldsymbol{\beta}^T \mathbf{x}$ does not contain an intercept term (bias weight). This is because the intercept would simply scale the baseline hazard, and therefore not contribute to the relative risk function,

$$h_0(t) \exp[g(\mathbf{x}) + b] = h_0(t) \exp[b] \exp[g(\mathbf{x})] = \tilde{h}_0(t) \exp[g(\mathbf{x})].$$

The Cox model in (3) is fitted in two steps. First, the parametric part is fitted by maximizing the *Cox partial likelihood*, which does not depend on the baseline hazard, then the non-parametric baseline hazard is estimated based on the parametric results. For individual i, let T_i denote the possibly censored event time and $\mathcal{R}_i$ denote the set of all individuals at risk at time T_i (not censored and have not experienced the event before time T_i). Note that $\mathcal{R}_i$ includes individuals with event times at T_i, so i is part of $\mathcal{R}_i$. The Cox partial likelihood, with Breslow's method for handling tied event times, is given by

$$L_{\text{cox}} = \prod_i \left(\frac{\exp[g(\mathbf{x}_i)]}{\sum_{j \in \mathcal{R}_i} \exp[g(\mathbf{x}_j)]} \right)^{D_i}, \tag{4}$$

and the negative partial log-likelihood can then be used as a loss function

$$\text{loss} = \sum_i D_i \log \left(\sum_{j \in \mathcal{R}_i} \exp[g(\mathbf{x}_j) - g(\mathbf{x}_i)] \right). \tag{5}$$

Let $\hat{\boldsymbol{\beta}}$ be the value of $\boldsymbol{\beta}$ that maximizes (4), or equivalently, minimizes (5). Then the cumulative baseline hazard function can be estimated by the *Breslow estimator*

$$\hat{H}_0(t) = \sum_{T_i \leq t} \Delta \hat{H}_0(T_i) \tag{6}$$

$$\Delta \hat{H}_0(T_i) = \frac{D_i}{\sum_{j \in \mathcal{R}_i} \exp[\hat{g}(\mathbf{x}_j)]},$$

where $\hat{g}(\mathbf{x}) = \hat{\boldsymbol{\beta}}^T \mathbf{x}$. If desired, the baseline hazard $h_0(t)$ can be estimated by smoothing the increments, $\Delta \hat{H}_0(T_i)$, of the Breslow estimate, but the cumulative baseline hazard typically provides the information we are interested in.

3.2. Cox with SGD

The Cox partial likelihood is usually minimized using Newton-Raphson's method. In our work, we instead want to fit the Cox model with mini-batch stochastic gradient descent

(SGD), to better scale to large data sets. As the loss in (5) sums over risk sets $\mathcal{R}_i$, which can be as large as the full data set, it cannot be computed in batches. Nevertheless, it is possible to do batched iterations by subsampling the data set (to a batch) and restrict the set $\mathcal{R}_i$ to only contain individuals in the current batch. This scales well for proportional methods such as DeepSurv (Katzman et al., 2018), but would be very computationally expensive for our non-proportional extension presented in Section 3.4. Hence, we propose an approximation of the loss that is easily batched.

Intuitively, we can approximate the risk set $\mathcal{R}_i$ with a sufficiently large subset $\tilde{\mathcal{R}}_i$, and weight the likelihood accordingly with weights w_i,

$$L = \prod_i \left(\frac{\exp[g(\mathbf{x}_i)]}{w_i \sum_{j \in \tilde{\mathcal{R}}_i} \exp[g(\mathbf{x}_j)]} \right)^{D_i}. \tag{7}$$

The weights should ensure that the weighted sum over the subset $\tilde{\mathcal{R}}_i$ in (7) is a reasonable approximation of the full sum over $\mathcal{R}_i$ in (4). By choosing a fixed sample size of the sampled risk sets $\tilde{\mathcal{R}}_i$, we can now optimize the objective by batched gradient descent. The individual i is always included in the sampled risk set $\tilde{\mathcal{R}}_i$ to ensure that each of the products in (7) is bounded above by 1. As the weights w_i do not contribute to the gradients of the logarithm of (7) (as can be seen by differentiating with respect to the model parameters), we can simply drop them from the loss function. Also, in practice we do not compute the loss for $D_i = 0$ as these entries do not contribute to (7). Finally, if we average the loss to make it independent of the data set size, we obtain

$$\text{loss} = \frac{1}{n} \sum_{i: D_i = 1} \log \left(\sum_{j \in \tilde{\mathcal{R}}_i} \exp[g(\mathbf{x}_j) - g(\mathbf{x}_i)] \right), \tag{8}$$

where n denotes the number of events in the data set. In our experiments in Sections 5 and 6, we find that it is often sufficient to sample only *one* individual j from the risk set, which gives us the loss

$$\text{loss} = \frac{1}{n} \sum_{i: D_i = 1} \log \left(1 + \exp[g(\mathbf{x}_j) - g(\mathbf{x}_i)] \right), \quad j \in \mathcal{R}_i \backslash \{i\}. \tag{9}$$

One benefit of (8) is that it is, in a sense, more interpretable than the negative partial log-likelihood in (5). Due to the sample dependence in the mean partial log-likelihood (MPLL), i.e., the expression in (5) divided by n, the magnitude of the MPLL is dependent on the size of the risk sets. Hence, for a change of batch size, the mean partial log-likelihood changes. This prohibits a comparison of losses across different batch sizes. Comparably, the loss in (8) is not affected by the choice of batch size, as the size of $\tilde{\mathcal{R}}_i$ is fixed. As a result, we can derive the range of values we expect the loss to be in. Using (9) as an example, we know that it is typically in the range $(0, 0.693]$, as a trivial $g(\mathbf{x}) = \text{const}$, gives a loss $= \log(2) \approx 0.693$, and the minimum is obtained by letting $g(\mathbf{x}_i) \to \infty$, $g(\mathbf{x}_j) \to -\infty$, which results in a loss that tends towards 0.

Sampling of the risk sets in Cox's partial likelihood is commonly done in epidemiology and formalized through the nested case-control design, originally suggested by Thomas

(1977). In (8), *case* refers to the i's, while the *controls* are the j's sampled from $\mathcal{R}_i \backslash \{i\}$. Goldstein and Langholz (1992) show that for the Cox partial likelihood, the sampled risk sets produce consistent parameter estimators. While their results do not extend to non-linear models, it is still an indication that the loss function in (8) is reasonable.

Our sampling strategy deviates from that of the nested case-control literature in two ways. Firstly, we sample a new set of controls for every iteration, instead of keeping control samples fixed. Secondly, we sample controls with replacement, as this requires less computation than sampling without replacement. Note, however, that we typically sample a single control, in which case it does not matter if we sample with or without replacement.

3.3. Non-Linear Cox

Having established the simple loss function in (8), which can be computed with SGD, the generalization of the relative risk function $\exp[g(\mathbf{x})]$ is rather straightforward. In this paper, we replace the linear predictor $g(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x}$ by a $g(\mathbf{x})$ parameterized by a neural network. While our proposed loss function is not a requirement for the adaptation of a neural network (see, e.g., DeepSurv by Katzman et al., 2018), it really helps for the further extensions in Section 3.4. Also, it has been found that batched iterations can improve predictive performance (Keskar et al., 2016; Hoffer et al., 2017).

Our generalization of $g(\mathbf{x})$ leaves the presented theory in Sections 3.1 and 3.2 essentially unchanged, so we do not repeat the likelihoods and loss functions for this model.

We will later refer to the Cox proportional hazards model parameterized with a neural network as Cox-MLP. To differentiate between minimizing the negative partial log-likelihood in (5), as done by DeepSurv, and our case-control approximation in (8), we will denote the corresponding methods by Cox-MLP (DeepSurv) and Cox-MLP (CC), respectively.

For the non-linear Cox models, the loss does not necessarily have a unique minimizer for $g(\mathbf{x})$. Therefore, we add a penalty to the loss function to encourage $g(\mathbf{x})$ to not deviate too far from zero

$$\text{penalty} = \lambda \sum_{i:D_i=1} \sum_{j \in \tilde{\mathcal{R}}_i} |g(\mathbf{x}_j)|. \tag{10}$$

Here λ is a tuning parameter, and note that i is included in $\tilde{\mathcal{R}}_i$.

3.4. Non-Proportional Cox-Time

The proportionality assumption of the Cox model can be rather restrictive, and parameterizing the relative risk function with a neural net does not affect this constraint. Approaches for circumventing this restriction are typically based on grouping the data based on a categorical covariate and applying a stratified version of the Cox model (Klein and Moeschberger, 2003, chap. 9). We propose a parametric approach that does not require stratification. Continuing with the semi-parametric form of the Cox model, we now let the relative risk function depend on time,

$$h(t \,|\, \mathbf{x}) = h_0(t) \exp[g(t, \mathbf{x})]. \tag{11}$$

In practice, we let $g(t, \mathbf{x})$ handle the time as a regular covariate, which enables $g(t, \mathbf{x})$ to model interactions between time and the other covariates. This is similar to the approach taken in classical survival analysis, where the non-proportional effect of a covariate x may be modeled by including time-dependent covariates like $x \cdot t$ and $x \cdot \log t$.

The model (11) is no longer a proportional hazards model. However, it is still a relative risk model with the same partial likelihood as previously, only now with an additional covariate. Following the approach from Section 3.2, we have the loss function

$$\text{loss} = \frac{1}{n} \sum_{i:D_i=1} \log \left(\sum_{j \in \tilde{\mathcal{R}}_i} \exp[g(T_i, \mathbf{x}_j) - g(T_i, \mathbf{x}_i)] \right), \tag{12}$$

and we include the penalty in (10), with $g(T_i, \mathbf{x}_j)$ replacing $g(\mathbf{x}_j)$. We will later refer to models fitted by (12) as *Cox-Time*.

Note that the loss has the same T_i for both $\mathbf{x}_i$ and the $\mathbf{x}_j$'s. Consequently, if we had used the full risk set $\mathcal{R}_i$ instead of the subset $\tilde{\mathcal{R}}_i$, as is the case for the loss in (5), the loss would become very computationally expensive. In fact, for the full risk set, the time complexity of the loss would be $O(n \cdot |\mathcal{R}_i|) = O(n^2)$, where $|\mathcal{R}_i|$ denotes the size of the risk set. But for (12) we get $O(n \cdot |\tilde{\mathcal{R}}_i|) = O(n)$, as $|\tilde{\mathcal{R}}_i|$ is fixed and small. In the proportional case, to compute the loss in (5) one only needs to compute $g(\mathbf{x}_j)$ once (per iteration) for each j, and reuse that value in all other risk sets. This ensures the linear time complexity for the classical Cox models.

We can find the Breslow estimate for the cumulative baseline hazard $H_0(t)$ using (6) with $\hat{g}(\mathbf{x}_j)$ replaced by $\hat{g}(T_i, \mathbf{x}_j)$. Note that we still need the non-parametric baseline, as $g(t, \mathbf{x})$ is restricted to model interactions between the time and the covariates. To see this, consider $g(t, \mathbf{x}) = a(t, \mathbf{x}) + b(t)$, and observe that $b(t)$ cancels out in the loss.

3.5. Prediction

We can obtain predictions from the relative risk models by estimating the survival function in (1), $\hat{S}(t \,|\, \mathbf{x}) = \exp[-\hat{H}(t \,|\, \mathbf{x})]$. For the proportional hazards models, the relative risk function does not depend on time, enabling us to integrate only over the baseline hazard and compute the relative risk separately,

$$H(t \,|\, \mathbf{x}) = \int_0^t h_0(s) \exp[g(\mathbf{x})] \, ds = H_0(t) \exp[g(\mathbf{x})].$$

By first estimating $H_0(t)$ on the training data with (6), we only need to compute $\exp[g(\mathbf{x})]$ to obtain predictions. Computation of the estimate $\hat{H}_0(t)$ requires a single pass over the whole training set, in addition to sorting the training set by time.

In the case of models with non-proportional hazards, such as models fitted by Cox-Time in Section 3.4, predictions are much more computationally expensive. As the relative risk is time-dependent, we now need to integrate over both the baseline hazard and $g(t, \mathbf{x})$,

$$H(t \,|\, \mathbf{x}) = \int_0^t h_0(s) \exp[g(s, \mathbf{x})] \, ds.$$

In practice, we estimate the cumulative hazards by

$$\hat{H}(t \,|\, \mathbf{x}) = \sum_{T_i \le t} \Delta \hat{H}_0(T_i) \exp[\hat{g}(T_i, \mathbf{x})],$$

where $\Delta \hat{H}_0(T_i)$ is an increment of the Breslow estimate and $\hat{g}(T_i, \mathbf{x})$ is the estimate of $g(T_i, \mathbf{x})$ obtained from the neural network. This is clearly rather computationally expensive as we need to compute $\hat{g}(T_i, \mathbf{x})$ for all distinct event times $T_i \le t$. Furthermore, for continuous-time data, computation of the cumulative baseline hazard through the Breslow estimate,

$$\Delta \hat{H}_0(T_i) = \frac{D_i}{\sum_{j \in \mathcal{R}_i} \exp[\hat{g}(T_i, \mathbf{x}_j)]}, \tag{13}$$

scales quadratically.

To alleviate the computational cost, one can compute the cumulative hazards over a reduced number of distinct time points. Hence, Cox-Time is trained on continuous-time data but produces discrete-time predictions, with the benefit of the discretization happening after the network is fitted. In practice, we perform this discretization by computing the baseline on a random subset of the training data and subsequently control the resolution of the time grid through the sample size.

4. Evaluation Criteria

Metrics for evaluating the performance of methods for time-to-event prediction should account for the censored individuals. In the following, we describe the metrics used in the experimental sections of this paper.

4.1. Concordance Index

In survival analysis, the concordance index, or C-index (Harrell Jr et al., 1982), is arguably one of the most commonly applied discriminative evaluation metrics. This is likely a result of its interpretability, as it has a close relationship to classification accuracy (Ishwaran et al., 2008) and ROC AUC (Heagerty and Zheng, 2005). In short, the C-index estimates the probability that, for a random pair of individuals, the predicted survival times of the two individuals have the same ordering as their true survival times. See Ishwaran et al. (2008) for a detailed description.

As the C-index only depends on the ordering of the predictions, it is very useful for evaluating proportional hazards models. This is because the ordering of proportional hazards models does not change over time, which enables us to use the relative risk function instead of a metric for predicted survival time. It is, however, not obvious how the C-index should be applied for non-proportional hazards models (Gerds et al., 2012; Ishwaran et al., 2008). We will use a metric based on the time-dependent C-index by Antolini et al. (2005), which estimates the probability that observations i and j are concordant given that they are comparable,

$$C^{\text{td}} = \text{P}\{\hat{S}(T_i \,|\, \mathbf{x}_i) < \hat{S}(T_i \,|\, \mathbf{x}_j) \mid T_i < T_j,\, D_i = 1\}. \tag{14}$$

However, to account for tied event times and survival estimates, we make the modifications listed by Ishwaran et al. (2008, Section 5.1, step 3). This is to ensure that predictions independent of $\mathbf{x}$, $\hat{S}(t \,|\, \mathbf{x}) = \hat{S}(t)$, yields $C^{\text{td}} = 0.5$ for unique event times. Note that for proportional hazards models, our metric is equivalent to the regular C-index.

4.2. Brier Score

The Brier score (BS) for binary classification is a metric of both discrimination and calibration of a model's estimates. In short, for N binary labels $y_i \in \{0, 1\}$ with probabilities p_i of $y_i = 1$, the BS is the mean squared error of the probability estimates $\hat{p}_i$, i.e., $\text{BS} = \frac{1}{N} \sum_i (y_i - \hat{p}_i)^2$. To get binary outcomes from time-to-event data, we choose a fixed time t and label data according to whether or not an individual's event time is shorter or longer than t. Graf et al. (1999) generalize the Brier score to account for censoring by weighting the scores by the inverse censoring distribution,

$$\text{BS}(t) = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{\hat{S}(t \,|\, \mathbf{x}_i)^2 \, \mathbb{1}\{T_i \leq t, D_i = 1\}}{\hat{G}(T_i)} + \frac{(1 - \hat{S}(t \,|\, \mathbf{x}_i))^2 \, \mathbb{1}\{T_i > t\}}{\hat{G}(t)} \right]. \tag{15}$$

Here N is the number of observations, $\hat{G}(t)$ is the Kaplan-Meier estimate of the censoring survival function, $\text{P}(C^* > t)$, and it is assumed that the censoring times and survival times are independent.

The BS can be extended from a single duration t to an interval by computing the integrated Brier score

$$\text{IBS} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \text{BS}(s) \, ds. \tag{16}$$

In practice, we approximate this integral by numerical integration, and we let the time span be the duration of the test set. In our experiments we found that using 100 grid points were sufficient to obtain stable scores.

4.3. Binomial Log-Likelihood

The mean binomial log-likelihood is a commonly used metric in binary classification that measures both discrimination and calibration of the estimates of a method. Using the same inverse censoring weighting as for the Brier score, we can apply this metric to censored duration time data,

$$\text{BLL}(t) = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{\log[1 - \hat{S}(t \,|\, \mathbf{x}_i)] \, \mathbb{1}\{T_i \leq t, D_i = 1\}}{\hat{G}(T_i)} + \frac{\log[\hat{S}(t \,|\, \mathbf{x}_i)] \, \mathbb{1}\{T_i > t\}}{\hat{G}(t)} \right]. \tag{17}$$

The binomial log-likelihood can also be integrated in the same manner as (16)

$$\text{IBLL} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \text{BLL}(s) \, ds.$$

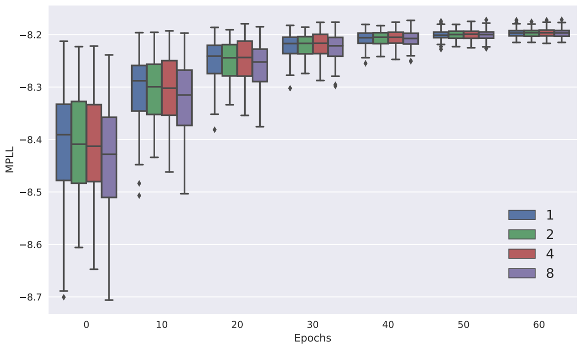

Figure 1: Box plots giving the mean partial log-likelihood (MPLL) of the test sets for different training epochs. The colors show how many controls were sampled during training (in addition to the case).

5. Simulations

To empirically investigate our proposed methodology, we conducted a series of simulations. These experiments are by no means exhaustive but are rather meant to verify that the methods behave as expected. In the following, we let *classical Cox* refer to a Cox regression with $g(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x}$ obtained with the Lifelines python package (Davidson-Pilon et al., 2018). For experimental details exempt from the main article, we refer the reader to Appendix C.3.

We first investigate the behavior of our proposed loss (8). In particular, we want to examine the impact the number of sampled controls has on the fitted models, in addition to how well the results from using our loss agree with those from using the Cox partial log-likelihood, i.e., the loss (5). To this end, we simulated survival times from a proportional hazards model

$$\begin{aligned} h(t \mid \mathbf{x}) &= h_0(t)\exp[g(\mathbf{x})], \\ g(\mathbf{x}) &= \boldsymbol{\beta}^T\mathbf{x}, \end{aligned} \tag{18}$$

with constant baseline hazard $h_0(t) = 0.1$, and $\boldsymbol{\beta}^T = [0.44, 0.66, 0.88]$. The covariates were sampled uniformly from $[-1,\, 1]$. We drew censoring times independent of the covariates with constant hazard $c(t) = \frac{1}{30}$, and, in addition, we censored all individuals that were still under observation at time 30. This resulted in approximately 30 % censored individuals.

We sampled 10,000 individuals for training and 10,000 for testing, and fitted our Cox model by SGD as described in Section 3.2. This method will be referred to as *Cox-SGD*. Four models were fitted by sampling 1, 2, 4, and 8 controls (in addition to the case). The whole experiment was repeated 100 times, and the mean partial log-likelihood (MPLL) of the test sets are visualized in Figure 1. The figure indicates that the number of sampled controls does not affect the rate of convergence, but we note that the computational complexity increases with the number of sampled controls.

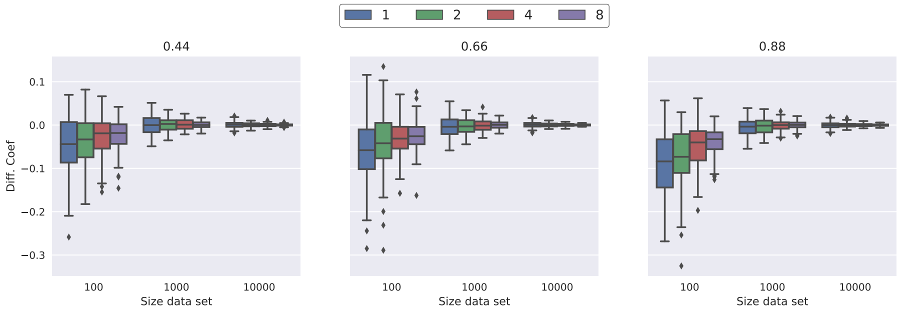

Figure 2: Differences between Cox-SGD and Classical Cox parameter estimates for different data set sizes. The numbers above the plots give the value of the true coefficient. Each box plot is based on 100 observations. The legend above the plots states the number of sampled controls for Cox-SGD.

The experiment was repeated with training sets of 1,000 individuals to verify that the results were not simply due to the size of the training data. The same patterns were found in this setting, so the figure is exempted from the paper.

Next, we compare the parameter estimates obtained from our proposed loss (Cox-SGD) with the estimates obtained with classical Cox regression. For data sets of sizes 100, 1,000, and 10,000, we fitted models to 100 sampled data sets. The differences between the Cox-SGD parameter estimates and the classical Cox estimates are displayed in Figure 2, where the legend above the plots gives the number of controls sampled for the Cox-SGD method. For the data sets of size 100, we observe that the Cox-SGD estimates seem to be slightly smaller than the Cox estimates, and this difference is larger for fewer sampled controls. However, as the data sets increase in size, the estimates for the two methods agree well.

Finally, we want to compare the likelihoods obtained by the two methods. As the mean partial log-likelihood depends on data set size, it is really not comparable across data sets. We will therefore instead use the full likelihood (2), which also depends on the baseline hazard, to compare the methods. Note that the partial likelihood may be interpreted as a profile likelihood, so the full likelihood and the partial likelihood are closely related (see, e.g., Klein and Moeschberger, 2003, p. 258). We obtain cumulative baseline hazard estimates by (6), and baseline hazard estimates by numerical differentiation of the cumulative baseline hazard estimates. We report the mean log-likelihood (MLL) of (2) to compare results for different data set sizes. In Figure 3, we show the difference in the MLL between the Cox-SGD method and the classical method for Cox regression (for each sampled data set). We used the training MLL as we are here only interested in the losses' abilities to optimize the objective, and not the generalization to a test set. From the figure, we observe that, for smaller data sets, more sampled controls in Cox-SGD seems to give likelihood estimates closer to those of a classical Cox regression. As the data sets increase in size, the MLL of the Cox-SGD seems to converge to that of the classical Cox, regardless of control sample

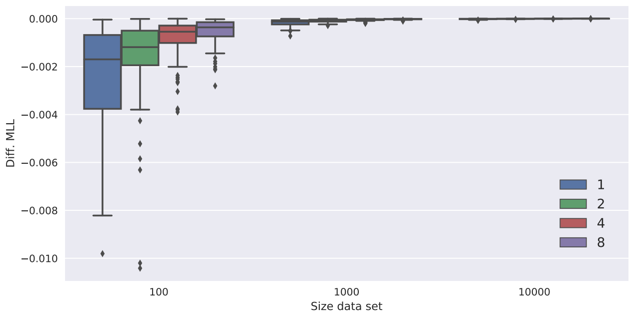

Figure 3: Differences in mean log-likelihood between Cox-SGD and classical Cox using (2). The figure gives results for different data set sizes, and the legend gives the number of sampled controls. Each box plot is based on 100 observations.

size. The MLL's of the classical Cox regression is approximately -2.2, meaning that even for the smallest data sets the differences in MLL for 1 sampled control is around 0.1 %. Hence, it seems that our loss (8) approximates the negative partial log-likelihood rather well. Furthermore, while a higher number of sampled controls can give lower training error, the effect of the number of sampled controls decreases with the size of the data sets.

In the further experiments, we will for simplicity only use one sampled control, as this was found sufficient. Furthermore, in Appendix C we have included simulations for the methods using neural networks for non-linear and non-proportional models, to verify that our proposed methods behave as expected. In summary, these simulations verify that for observations drawn from a proportional hazards model with a non-linear relative risk function, Cox models parameterized by neural networks provide better estimates of the partial log-likelihood than classical Cox models. Further, by drawing observations from a non-proportional relative risk model, the simulations verify that our Cox-Time method (Section 3.4) is able to obtain better estimates of the survival functions than methods which assume proportional hazards.

6. Experiments

In the following, we will evaluate our proposed methods on real data sets and compare their performance to existing methods from the literature. In total, we use five data sets. One large data set for a more in-depth analysis (see Section 6.2), and four smaller data sets commonly used in the survival analysis literature.

6.1. Four Common Survival Data Sets

We base this experimental section on the data sets provided by Katzman et al. (2018), as they are made available through the DeepSurv python package, and need no further prepro-

Data set	Size	Covariates	Unique Durations	Prop. Censored
SUPPORT	8,873	14	1,714	0.32
METABRIC	1,904	9	1,686	0.42
Rot. & GBSG	2,232	7	1,230	0.43
FLCHAIN	6,524	8	2,715	0.70

Table 1: Summary of the four data sets used in the experiments in Section 6.1.

cessing. The data sets include the Study to Understand Prognoses Preferences Outcomes and Risks of Treatment (SUPPORT), the Molecular Taxonomy of Breast Cancer International Consortium (METABRIC), and the Rotterdam tumor bank and German Breast Cancer Study Group (Rot. & GBSG). Katzman et al. (2018) also provide the Worcester Heart Attack Study (WHAS) data set. However, their version of the data set is actually a case-control data set, meaning it contains multiple replications of some individuals, something the authors seem to have overlooked. We replace it with the Assay Of Serum Free Light Chain (FLCHAIN) made available in the survival packages of R (Therneau, 2015). For FLCHAIN, we remove individuals with missing values. Further, we remove the "chapter" covariate, which gives the cause of death. Table 1 provides a summary of the data sets. For a more detailed description, we refer to the original sources (Therneau, 2015; Katzman et al., 2018).

6.1.1. Methods and Hyperparameter Tuning

In Sections 3.3 and 3.4 we presented two new survival methods based on case-control sampling and neural networks: a proportional Cox method and a non-proportional Cox method, which we will refer to as *Cox-MLP (CC)* and *Cox-Time* respectively. We will compare our methods to a classical linear Cox regression referred to as *Classical Cox (Linear)*, DeepHit (Lee et al., 2018), and Random Survival Forests (RSF) (Ishwaran et al., 2008). We will also compare to a proportional Cox method similar to DeepSurv (Katzman et al., 2018), but our version performs batched SGD by computing the negative partial log-likelihood in (5) on a subset of the data set. Furthermore, we choose not to restrict the network structure and optimization scheme to that of Katzman et al. (2018). Hence, this method is identical to our proportional Cox method in Section 3.3, except that it computes the negative partial log-likelihood of a batch, while we use case-control sampling in the loss function. We will refer to these two methods as Cox-MLP (DeepSurv) and Cox-MLP (CC) respectively. We do not compare with Luck et al. (2017) as their method is another proportional hazard method and is therefore restricted in all the same ways as our other proportional methods. As we will show in Section 6.1.2, the proportional hazards assumption is very restrictive, and methods based on this assumption are therefore not able to compete with other methods such as DeepHit, RSF, and Cox-Time.

For the neural networks, we standardize the numerical covariates and encode the categorical covariates by entity embeddings (Guo and Berkhahn, 2016) half the size of the number of categories. For the Classical Cox (Linear) regression, we one-hot encoded the categorical covariates (dummy variables), and in RSF we simply passed the covariates without any transformations. The networks are standard multi-layer perceptrons with the same

Method	SUPPORT	METABRIC	Rot. & GBSG	FLCHAIN
Classical Cox (Linear)	0.598	0.628	0.666	0.790
Cox-MLP (DeepSurv)	0.611	0.636	0.674	0.790
Cox-MLP (CC)	0.613	0.643	0.669	**0.793**
Cox-Time	0.629	0.662	**0.677**	0.790
DeepHit	**0.642**	**0.675**	0.670	0.792
RSF (Mortality)	0.628	0.649	0.667	0.784
RSF (C^{td})	0.634	0.652	0.669	0.786

Table 2: Concordance (C^{td}) for the experiments in Section 6.1.

number of nodes in every layer, ReLU activations, and batch normalization between layers. We used dropout, normalized decoupled weight decay (Loshchilov and Hutter, 2019), and early stopping for regularization. SGD was performed by AdamWR (Loshchilov and Hutter, 2019) with an initial cycle length of one epoch, and we double the cycle length after each cycle. Learning rates were found using the methods proposed by Smith (2017).

As the four data sets are somewhat small, we scored our fitted models using 5-fold cross-validation, where the hyperparameter search was performed individually for each fold. For all the neural networks, we performed a random hyperparameter search over 300 parameter configurations and chose the model with the best score on a held-out validation set. We scored the proportional Cox methods by the partial log-likelihood, and for the Cox-Time method, we used the loss (12). Model hyperparameter tuning for DeepHit and RSF was performed with the time-dependent concordance index (Antolini et al., 2005). However, we also include an RSF tuned in the manner proposed by the authors, i.e., by computing the concordance of the mortality (Ishwaran et al., 2008, Section 4.1). The two versions of RSF are in the following denoted RSF (C^{td}) and RSF (Mortality), respectively. A list of the hyperparameter search spaces can be found in Appendix A.1.

6.1.2. Results

We compare the methods using the metrics presented in Section 4, i.e., the time-dependent concordance, the integrated Brier score, and the integrated binomial log-likelihood. While the concordance solely evaluates a method's discriminative performance, the Brier score and binomial log-likelihood also evaluate the calibration of the survival estimates.

Table 2 shows the time-dependent concordance, or C^{td}, averaged over the five cross-validation folds. As expected, the methods that assume proportional hazards, in general, perform worse than the less restrictive methods, and Cox-MLP (DeepSurv) and Cox-MLP (CC) are very close to each other. We see that RSF (C^{td}) has better concordance than RSF (Mortality), which is expected as RSF (C^{td}) uses the same metric for hyperparameter tuning as for evaluation. Cox-Time seems to do slightly better than the RSF methods, which is impressive as we have not used concordance for hyperparameter tuning. DeepHit seems to have the best discriminative performance overall, but, as we will see next, this comes at the cost of poorly calibrated survival estimates.

Method	SUPPORT	METABRIC	Rot. & GBSG	FLCHAIN
Classical Cox (Linear)	0.217	0.183	0.180	0.096
Cox-MLP (DeepSurv)	0.214	0.176	0.170	**0.092**
Cox-MLP (CC)	0.213	0.174	0.171	0.093
Cox-Time	**0.212**	**0.172**	**0.169**	0.102
DeepHit	0.223	0.184	0.178	0.120
RSF (Mortality)	0.215	0.175	0.171	0.093
RSF (C^{td})	**0.212**	0.176	0.171	0.093

Table 3: Integrated Brier score weighted by estimates of the censoring distribution for the experiments in Section 6.1.

Method	SUPPORT	METABRIC	Rot. & GBSG	FLCHAIN
Classical Cox (Linear)	-0.623	-0.538	-0.529	-0.322
Cox-MLP (DeepSurv)	-0.619	-0.532	-0.514	**-0.309**
Cox-MLP (CC)	-0.615	-0.515	-0.509	-0.314
Cox-Time	-0.613	**-0.511**	**-0.502**	-0.432
DeepHit	-0.637	-0.539	-0.524	-0.487
RSF (Mortality)	-0.619	-0.515	-0.507	-0.311
RSF (C^{td})	**-0.610**	-0.517	-0.507	-0.311

Table 4: Integrated binomial log-likelihood weighted by estimates of the censoring distribution for the experiments in Section 6.1.

Tables 3 and 4 show the integrated Brier score and integrated binomial log-likelihood, both weighted with Kaplan-Meier estimates of the censoring distribution. Here, for both metrics, closer to zero is better. We find that both metrics yield very similar results, as the orderings of the methods are almost identical. First, we see that Cox-Time seems to generally perform the best, but it struggles with the FLCHAIN data set. However, we note that, for FLCHAIN, Cox-MLP (DeepSurv) has the best integrated Brier score and binomial log-likelihood while Cox-MLP (CC) has the best concordance. This indicates that the proportionality assumption is quite reasonable for this data set.

Again, we find that there is very little difference between Cox-MLP (DeepSurv) and Cox-MLP (CC), which is as expected. The RSF methods generally perform well. Note that the two RSF methods perform equally well here, even though RSF (C^{td}) had the best concordance. Classical Cox (Linear) does rather poorly, which was expected as it has very restrictive model assumptions.

Even though DeepHit, in general, had the best concordance, it has the worst integrated Brier score and binomial log-likelihood in three out of the four data sets. The loss function in DeepHit, given by formula (D.2) in Appendix D, is a convex combination of the negative log-likelihood and a ranking loss, determined by a parameter α. For $\alpha = 1$ we obtain only the negative log-likelihood and for $\alpha = 0$ we obtain only the ranking loss. As we

Data set	Size	Churned	Censored	Prop. Censor	Unique users
Train	1,786,333	1,279,358	506,975	0.28	1,582,202
Test	661,748	473489	188,259	0.28	586,001
Validation	198,665	142,104	56,561	0.28	175,801

Table 5: Summary of the KKBox churn data set.

do hyperparameter tuning based on the concordance, we see that α tends towards smaller values, which results in excellent discriminative performance at the cost of poorly calibrated survival estimates.

6.2. KKBox Churn Case Study

Thus far we have compared competing survival methodologies on fairly small data sets. We now perform a case study on a much larger data set, as this is more interesting in the context of neural networks.

The WSDM KKBox's churn prediction challenge was proposed in preparation for the 11th ACM International Conference on Web Search and Data Mining.[2] The competition was hosted by Kaggle in 2017, with the goal of predicting customer churn on a data set donated by KKBox, the leading music streaming service in Asia. The available data provide us the opportunity to create a survival data set with event times and censoring indicators. We stick with the churn definition given by KKBox, were a customer is considered churned if he or she fails to resubscribe within 30 days after the previous subscription expired. Note, however, that our use of the data is not comparable to the Kaggle competition, as we work with survival times from the start of a subscription period, while they consider durations from a fixed calendar date.

KKBox provides multiple data sources, but as we are primarily interested in evaluating our methods, we spend less time on feature engineering and only use a subset of covariates with general customer information (e.g., city, age, price of subscription). Furthermore, a customer that has previously churned and later resubscribed, is treated as a new customer with some covariate information describing the previous subscription history. This gives us a total of 15 covariates. We split the data into a training, a testing, and a validation set, and some information about these subsets are listed in Table 5. A more in-depth description of the KKBox data set can be found in Appendix B.1.

6.2.1. Methods and Hyperparameter Tuning

We use the same methods as in Section 6.1. However, as this data set is very large, we replace the classical Cox regression with our Cox-SGD (Linear) method from Section 3.2.

We standardize and encode the covariates in the same manner as for the smaller data sets. The networks and training are also the same as earlier, but we multiply the learning rate by 0.8 at the start of every cycle, as we found this to give more stable training. The KKBox data set is quite large, so we are not able to explore as large a hyperparameter space as in the previous experiments. Hence, we do not include weight decay, and perform

2. `https://www.kaggle.com/c/kkbox-churn-prediction-challenge`

Method	Layers	Nodes	Dropout	α	σ
Cox-MLP (DeepSurv)	6	256	0.1	-	-
Cox-MLP (CC)	6	128	0	-	-
Cox-Time	8	256	0	-	-
DeepHit	6	512	0.1	0.001	0.5

Table 6: KKBox model architectures. α and σ only applies the DeepHit (see Appendix D).

Method	C^{td}	IBS	IBLL
Cox-SGD (Linear)	0.816	0.127	-0.406
Cox-MLP (DeepSurv)	0.841	0.111	-0.349
Cox-MLP (CC)	0.844	0.119	-0.379
Cox-Time	0.861	**0.107**	**-0.334**
DeepHit	**0.888**	0.147	-0.489
RSF (Mortality)	0.855	0.112	-0.352
RSF (C^{td})	0.870	0.111	-0.352

Table 7: Evaluation metrics for the KKBox data.

a grid search over a small number of suitable parameters. The hyperparameter search is described in detail in Appendix A.2.

The best configurations are given in Table 6. For RSF, the hyperparameter search based on C^{td} yielded 8 covariates sampled for each split and a minimum of 50 observations in each leaf node. With concordance of mortality as the validation criterion, the best fitted model used 2 covariates for splitting, and a minimum leaf node size of 10. Furthermore, we found that 500 trees were sufficient, as there was little improvement compared to 250 trees.

6.2.2. Results

For evaluation, we fitted each of the methods five times and computed the time-dependent concordance index (C^{td}), the integrated Brier score (IBS), and the integrated binomial log-likelihood (IBLL) of the fitted models. The median scores are presented in Table 7. We use the median because the two proportional Cox-MLP methods yielded rather unstable results, where some of the fitted models performed very badly.

From the table, we see that DeepHit continues to outperform the other methods in terms of concordance while having the worst performance in terms of IBS and IBLL. Furthermore, Cox-Time has the best IBS and IBLL, while still providing a decent concordance. RSF continues to do well across all metrics, while again, the tuning based on C^{td} seems to yield better results than tuning base on the concordance of the mortality. Cox-SGD (Linear) does rather poorly, as it is very restricted, and serves more as a baseline in this context. The Cox-MLP methods seem to again perform reasonably close to each other, at least when taking into account that we found both methods to yield rather unstable results. We are

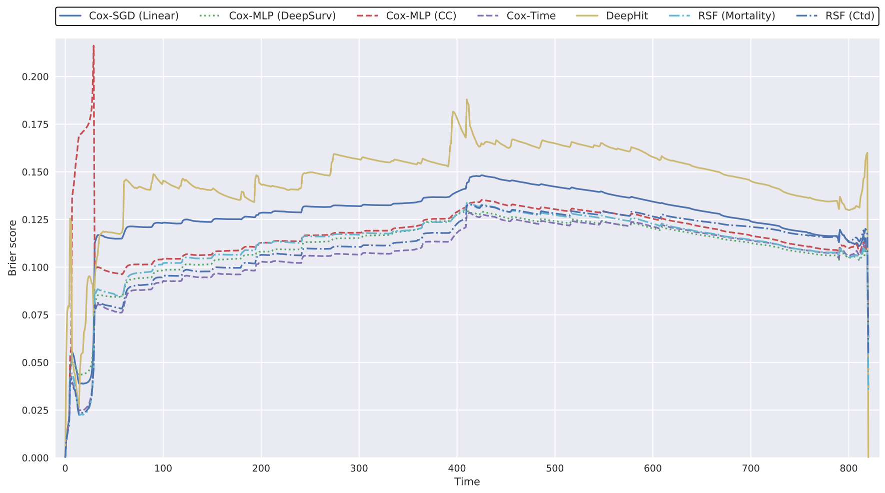

Figure 4: Brier score on KKBox data set. The methods are the same as in Table 7.

not sure why this was the case, but note that the combination of the flexible neural net and the proportionality constraint might be problematic for large data sets.

In Figure 4, we display the Brier scores used to obtain the IBS. Again, we see that DeepHit does poorly for all durations. The instability of the Cox-MLP (CC) is also very apparent for shorter durations. Cox-Time is clearly doing very well for all durations, but interestingly, we observe the Cox-MLP (DeepSurv) provides the best fitted model for larger durations. We could make a corresponding figure for the binomial log-likelihood, but as it is very similar to the Brier score plot, and provide us with no new insights, we have not included it.

6.2.3. Survival Curves

One of the benefits survival models have over binary classifiers is their ability to produce survival curves. In Figure B.1 in Appendix B, we show nine examples of estimated survival curves from Cox-Time on the test data. The curves nicely illustrate the extent of detail the method has learned.

To obtain a more general view of the predictions, we cluster the estimated survival curves of the test set. For an equidistant grid $0 = \tau_0 < \tau_1 < \cdots < \tau_m$, the survival curve of individual i is represented by a vector $[\hat{S}(\tau_0 \,|\, \mathbf{x}_i), \hat{S}(\tau_1 \,|\, \mathbf{x}_i), \ldots, \hat{S}(\tau_m \,|\, \mathbf{x}_i)]$, and by considering these as feature vectors, we apply K-means clustering to the test set with 10 clusters. In Figure 5, we display the cluster centers and the proportions of the test set assigned to each of the clusters. This is a reasonable approach for segmenting customers based on their churning behavior. A natural next step would be to further investigate the clusters, but as we consider this somewhat outside the scope of this paper, we only make

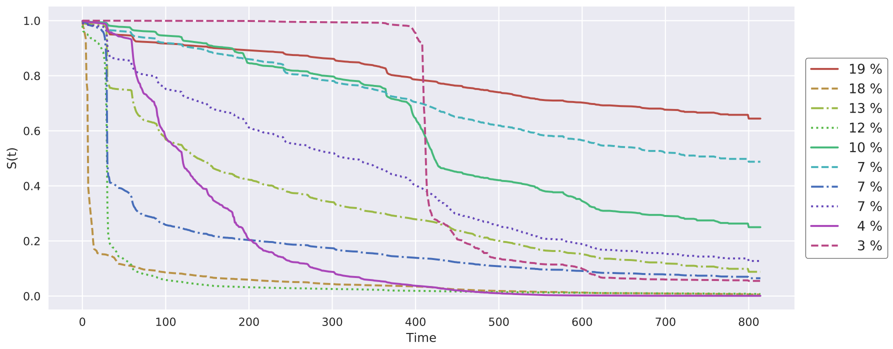

Figure 5: Cluster centers of survival curves from KKBox test data. The centers were generated by K-means clustering on survival curves from Cox-Time. The legend gives the proportion of test data assigned to each cluster (rounded to nearest integer).

a few observations. First, we see that 19 % of the customers are assigned to a cluster that does not provide much detailed information about their behavior, but instead provides a survival curve with a rather constant slope. In sharp contrast, the second largest group (18 %) is at high risk of churning immediately. Furthermore, we observe that many of the curves seem to have higher hazards at the end of each month (drops in the survival curves around 30, 60, 90 days), and we hypothesize that this is a result of customers paying for a full month at a time.

The smallest cluster, constituting only 3 % of the test data, has a sharp drop around day 400. Investigating the covariates of the assigned customers reveals that most of them had prepaid for a 411 days long subscription. However, the large drop after 400 days indicates that only around 25 % of them were interested in continuing their subscription.

Our choice of 10 clusters is mainly motivated by how many curves that can be visualized in a single plot, without being too crowded. Further increasing the number of clusters would likely reveal more detailed behavior.

7. Discussion

In this paper, we propose extensions of the Cox proportional hazards model. By parameterizing the relative risk function of a Cox model with neural networks, we can model rich relationships between the covariates and event times. Furthermore, we allow the networks to model interactions between the covariates and time, resulting in models that are no longer constrained by the proportionality assumption of the Cox model. Building on methods for nested case-control studies, we propose a loss function that can be computed in batches, enabling the models to scale to large data sets. We verify through simulation studies that fitting a Cox model using our proposed loss function gives results close to those obtained using the full Cox partial likelihood.

We compare our suggested methodology with classical Cox regression, random survival forests (RSF), DeepHit, and DeepSurv on 5 real-world data sets, and find that our proposed Cox-Time method performs very well, and has the best overall performance in terms of integrated Brier score (IBS) and integrated binomial log-likelihood (IBLL). DeepHit has, in general, the best discriminative performance, but this comes at the cost of poorly calibrated survival estimates.

Finally, we show how estimated survival curves (event probabilities as functions of time), can be used as a descriptive tool to better understand event-time data sets. This is illustrated by an example where we cluster the survival estimates of a customer churn data set, and show that this customer segmentation provides a useful view of the churning process.

Interesting expansions of our methodology include the extension to handle multiple competing events, time-dependent covariates, dynamic predictions, and recurrent events. Furthermore, it would be of interest to explore other data sources that require more advanced network structures such as convolutions and recurrent neural networks. Finally, less computationally expensive alternatives for creating survival estimates for the Cox-Time method should be explored.

Acknowledgments

This work was supported by The Norwegian Research Council 237718 through the Big Insight Center for research-driven innovation.

Appendix A. Details on Hyperparameter Tuning

In the following, we provide further details about the experiments conducted in Section 6. Here we list hyperparameter configurations and details about the model fitting procedures.

A.1. Four Common Survival Data Sets Tuning

Table A.1 gives the hyperparameter search space used for Rot. & GBSG, SUPPORT, METABRIC, and FLCHAIN. The square brackets describe continuous variables. In the experiments in Section 6.1, we sample 300 random parameter configurations for each method, for each fold of each data set. In the table, "α" and "σ" are given in (D.2) in Appendix D, "Num. durations" are the number of discrete durations (equidistant) used in DeepHit, "λ" is the penalty in (10), "Log duration" refers to a log-transform of the durations passed to Cox-Time, "Ridge" is a ridge penalty used in classical Cox regression, "Split covariates" and "Size leaf" are the number of covariates used for splitting, and the minimum node size of RSF.

A.2. KKBox Tuning

Hyperparameters in the KKBox study were found by a grid search over the relevant parameters in Table A.2. The table consists of three sections, where the top represents the networks, the bottom represents RSF, and the middle contains network parameters that were found on a smaller network with two layers and 128 nodes. "α" and "σ" controls the

Hyperparameter	Values
Layers	{1, 2, 4}
Nodes per layer	{64, 128, 256, 512}
Dropout	[0, 0.7]
Weigh decay	{0.4, 0.2, 0.1, 0.05, 0.02, 0.01, 0}
Batch size	{64, 128, 256, 512, 1024}
α (DeepHit)	[0, 1]
σ (DeepHit)	{0.1, 0.25, 0.5, 1, 2.5, 5, 10, 100}
Num. durations (DeepHit)	{50, 100, 200, 400}
λ (Cox-Time and Cox-MLP (CC))	{0.1, 0.01, 0.001, 0}
Log durations (Cox-Time)	{True, False}
Ridge (Cox (Linear))	$\{1000, 100, 10, 1, 0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}\}$
Split covariates (RSF)	{2, 4, 6, 8}
Size leaf (RSF)	{2, 8, 32, 64, 128}

Table A.1: Hyperparameter search space for experiments on Rot. & GBSG, SUPPORT, METABRIC, and FLCHAIN.

Hyperparameter	Values
Layers	{4, 6, 8}
Nodes per layer	{128, 256, 512}
Dropout	{0, 0.1, 0.5}
α(*)	{0, 0.001, 0.1, 0.2, 0.5, 0.8, 0.9, 0.99, 0.999, 1}
σ(*)	{0.01, 0.1, 0.25, 0.5, 1, 10, 100}
Log durations(*)	{True, False}
Split covariates	{2, 4, 6, 8}
Size leaf	{8, 10, 20, 50}

Table A.2: KKBox hyperparameter configurations. (*) denotes parameters found with a two layer network with 128 nodes.

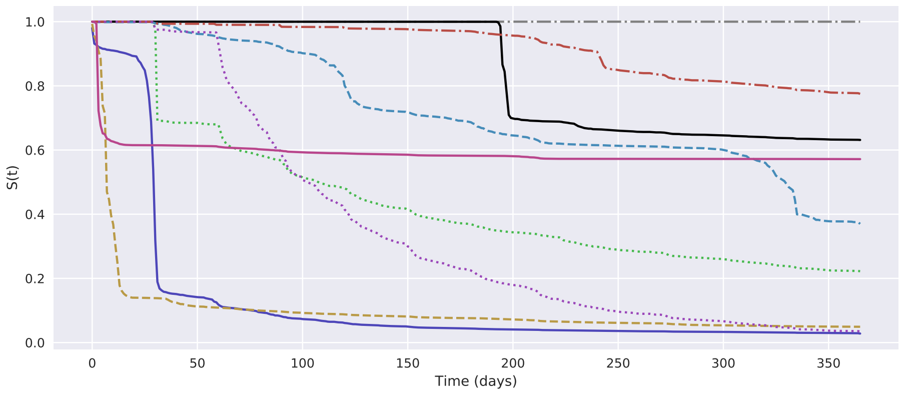

Figure B.1: Survival curves from the case study in Section 6.2, which models the times at which customers of a streaming service churn. Each curve gives the estimated probabilities for a customer not having churned (probabilities of still being a customer at any given time). The time axis shows the number of days since first subscription. The curves are generated by the Cox-Time method from Section 3.4.

loss function of DeepHit, and we assumed it should generalize well across network structures. The same goes for whether or not we should log-transform the durations passed to Cox-Time. Hence, to reduce the hyperparameter search, we found suitable values with a smaller network.

For the proposed Cox-MLP (CC) and Cox-Time, we used a fixed penalty $\lambda = 0.001$ in (10). All networks were trained with batch size of 1028, and the best performing architectures can be found in Table 6.

Appendix B. Details from KKBox Churn Case Study

In the following, we provide some details of the KKBox case study that were exempt from the main article.

In Figure B.1, we show an example of nine survival curves estimated by Cox-Time on the KKBox data set. Each line represents an individual from the test set. It is clear that the Cox-Time method has learned to represent a variety of survival curves.

B.1. KKBox Data Set

KKBox provides data consisting of general customer information (city, age, gender, initial registration), transaction logs listing how customers manage their subscription plans, and user logs containing some aggregate information of the customers' usage of the streaming service. KKBox defines a customer as churned if he or she has failed to resubscribe to their service more than 30 days after their last subscription expired.

Through the transaction information, we can create a data set with survival times and censoring indicators. We keep KKBox's original churn definition of 30 days absence and calculate the survival time from the date of the first subscription or the earliest record of the customer in the transaction logs. Customers that have previously churned but resubscribed, are treated as new customers with some covariate information describing their previous subscription history.

All covariates are calculated at the time of subscription of each customer, and they do not change with time. This is a simplification, as the covariates of a customer are typically not stationary. However, as our objective is to evaluate the proposed methodology, we refrain from doing extensive feature engineering. In this regard, we further disregard the user logs, as we get a reasonable set of covariates from the customer and transaction information.

The data sets are summarized in Table 5. As some of the customers have churned multiple times, and are therefore included multiple times, the table also includes the number of unique customers in the respective data sets.

We have a total of 15 covariates, where 7 are numeric and 8 are categorical. The numerical covariates give the time between subscriptions (for customers that have previously churned), number of previous churns, time since first registration, number of days until the current subscription expires, listed price of the current subscription, the amount paid for the subscription, and age. All numerical covariates, except the number of previous churns, were log-transformed. The categorical covariates are gender (3 categories), city (22 categories), how the customer registered (6 categories), and 5 indicator variables specifying if an individual has previously churned, if the subscription is canceled, if the age is unrealistic, if the subscription is automatically renewed, and if we do not know when the customer first registered.

Appendix C. Additional Simulations

In the following, we provide some additional simulations used to further investigate the proposed methods, and in Appendix C.3 we explain how the simulated data were generated. We again stress that the aim of the simulations is only to verify the expected behavior of our methods, and should not be interpreted as a general evaluation of them.

C.1. Non-Linear Hazards

Continuing from the simulations in Section 5, we do a simple study of the increased flexibility provided by replacing the linear predictor in a Cox regression by a neural network. To evaluate this, we extend the simulations in Section 5 by replacing $g(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x}$ with the non-linear function

$$g(\mathbf{x}) = \boldsymbol{\beta}^T\mathbf{x} + \frac{2}{3}(x_1^2 + x_3^2 + x_1x_2 + x_1x_3 + x_2x_3), \tag{C.1}$$

where x_i denotes element i of $\mathbf{x}$. The simulations are otherwise unchanged from the linear case in Section 5. We sample 10,000 training samples, 10,000 test samples, and 1,000 samples used for validation (validation data is used for early stopping), and fit the Cox-SGD and

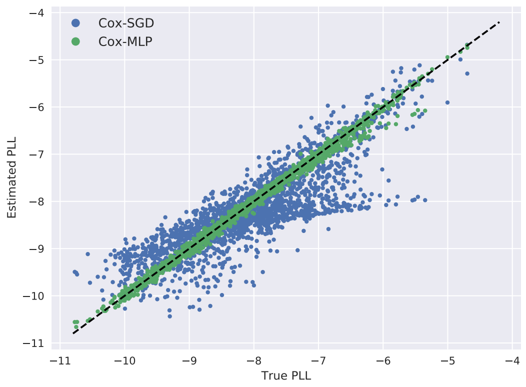

Figure C.1: Partial log-likelihood estimates of Cox-SGD and Cox-MLP plotted against the true partial log-likelihoods, for 2,000 samples of the test set.

classical Cox regression from Section 5. Additionally, we fit the Cox model in Section 3.3, where $g(\mathbf{x})$ is parameterized by a one-hidden layer MLP (multilayer perceptron) with 64 hidden nodes and ReLU activations. This model will be referred to as *Cox-MLP* (we drop (CC) as we do not consider DeepSurv here).

In Figure C.1 we have, for 2,000 individuals of the test set, plotted the individual partial log-likelihood estimates of Cox-SGD and Cox-MLP against the true individual partial log-likelihoods. That is, for each i with $D_i = 1$, we plot

$$\hat{\ell}_i = -\log\left(\sum_{j \in \mathcal{R}_i} \exp[\hat{g}(\mathbf{x}_j) - \hat{g}(\mathbf{x}_i)]\right)$$

against the true ℓ_i. The closer a method estimates $g(\mathbf{x})$ to the true predictor in (C.1), the closer the scatter plot should be to the identity function (straight line with slope 1 through the origin). As expected, we see that Cox-MLP produces likelihood estimates very close to the true likelihoods, while Cox-SGD struggles to represent this non-linear function.

C.2. Non-Proportional Hazards

In our final simulations, we investigate the effect of removing the proportionality constraint in the Cox models. Building on the previous simulations, we add a time dependent term to the risk function. The hazard function is now given by $h(t \,|\, \mathbf{x}) = h_0 \exp[g(t, \mathbf{x})]$, with $h_0 = 0.02$ and

$$\begin{aligned} g(t, \mathbf{x}) &= a(\mathbf{x}) + b(\mathbf{x})\, t, \\ a(\mathbf{x}) &= g_{ph}(\mathbf{x}) + \text{sign}(x_3), \\ b(\mathbf{x}) &= |0.2\,(x_0 + x_1) + 0.5\, x_0 x_1|, \end{aligned}$$

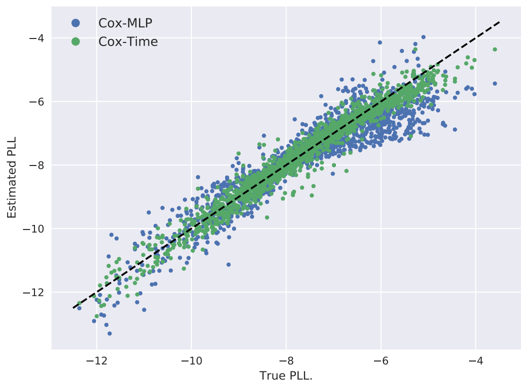

Figure C.2: Partial log-likelihood estimates of Cox-MLP and Cox-Time plotted against the true partial log-likelihoods, for 2,000 samples of the test set.

were $g_{ph}(\mathbf{x})$ is the function in (C.1). The simulations are otherwise unchanged from the previous experiments. We require the term $b(\mathbf{x})$ to be non-negative to ensure that the hazards increase with time. Furthermore, we have added $\text{sign}(x_3)$ to $a(x)$ as this is essentially equivalent to having two different baselines, $\tilde{h}_0 = h_0 \exp[\text{sign}(x_3)] \in \{0.0074, 0.054\}$. Both of these choices are motivated by our attempt to produce reasonable looking survival curves.

We sample 10,000 training samples, 10,000 test samples, and 2,000 samples for validation, and fit a Cox-MLP model and a Cox-Time model. Both Cox-MLP and Cox-Time parameterize g with a 4 hidden layer MLP with ReLU activations, 128 nodes in each layer, and dropout between layers with rate 0.1.

In Figure C.2, we have plotted the estimated partial log-likelihoods of Cox-MLP and Cox-Time against the true partial log-likelihoods for 2,000 samples of the test set. Though the difference between the methods is not very large, Cox-Time appears to capture the true values better than Cox-MLP.

To further illustrate some of the differences between Cox-MLP and Cox-Time, we have in Figure C.3 plotted nine survival curves $\hat{S}(t \,|\, \mathbf{x})$ from the test set. The figure shows both the true curves and the curves estimated by the two methods. It is seen that Cox-Time is able to estimate the true survival curves better than those produced by Cox-MLP.

C.3. Simulation Details

In the following, we explain in detail how we generated our simulated data sets in Section 5 and Appendices C.1 and C.2. We want to generate survival times T^* from relative risk models of the form

$$h(t \,|\, \mathbf{x}) = h_0(t) \exp[g(t, \mathbf{x})]. \tag{C.2}$$

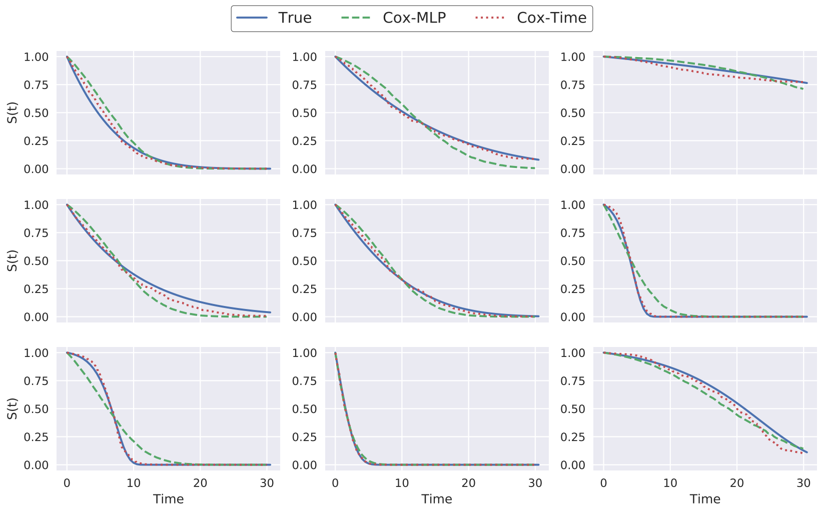

Figure C.3: Examples of survival curves $\hat{S}(t \,|\, \mathbf{x})$ from the test set in the non-proportional simulation study.

If the function g does not depend on t, we have a proportional hazards model, which is just a special case of the relative risk models. Let $H(t \,|\, \mathbf{x})$ denote the continuous (increasing) cumulative hazard $H(t \,|\, \mathbf{x}) = \int_0^t h(s \,|\, \mathbf{x}) \, ds$, and let V be exponentially distributed with parameter 1. Then we can obtain survival times through the inverse cumulative hazard $T^* = H^{-1}(V \,|\, \mathbf{x})$. This can be shown to hold through the hazard's relationship to the survival function in (1),

$$S(t \,|\, \mathbf{x}) = \mathrm{P}(T^* > t \,|\, \mathbf{x}) = \mathrm{P}(H^{-1}(V \,|\, \mathbf{x}) > t) = \mathrm{P}(V > H(t \,|\, \mathbf{x})) = \exp[-H(t \,|\, \mathbf{x})],$$

as V is exponentially distributed with $\mathrm{P}(V > v) = \exp(-v)$. Hence, we can obtain survival times by transforming generated samples from an exponential distribution.

To obtain an analytical expression for the inverse cumulative hazard $H^{-1}(v \,|\, \mathbf{x})$, we restrict the models in (C.2) to have constant baseline h_0, in addition to a simple time dependence in $g(t, \mathbf{x})$. For the linear predictor in Section 5 and the non-linear predictor in Appendix C.1 we simulate with proportional hazards, meaning $g(t, \mathbf{x}) = g(\mathbf{x})$. Hence, we get cumulative hazards and inverse cumulative hazards of the form

$$\begin{aligned} H(t \,|\, \mathbf{x}) &= t \, h_0 \exp[g(\mathbf{x})], \\ H^{-1}(v \,|\, \mathbf{x}) &= \frac{v}{h_0 \exp[g(\mathbf{x})]}. \end{aligned}$$

In the non-proportional simulations in Appendix C.2, we add a time dependent term $g(t, \mathbf{x}) = a(\mathbf{x}) + b(\mathbf{x})\, t$, which gives us

$$
\begin{aligned}
H(t \,|\, \mathbf{x}) &= \frac{h_0 \exp[a(\mathbf{x})] \left(\exp[b(\mathbf{x})\, t] - 1\right)}{b(\mathbf{x})}, \\
H^{-1}(v \,|\, \mathbf{x}) &= \frac{1}{b(\mathbf{x})} \log\left(1 + \frac{v\, b(\mathbf{x})}{h_0\, \exp[a(\mathbf{x})]}\right).
\end{aligned}
$$

Appendix D. DeepHit

DeepHit by Lee et al. (2018) is a discrete survival model for competing risks. However, as we only consider one type of event, we will express the method in terms of a single cause. DeepHit considers time to be discrete, so to fit it to the continuous-time data sets in Section 6, we discretize the event times with an equidistant grid between the smallest and largest duration in the training set. The number of discrete time-points is considered a hyperparameter, given by "Num. durations" in Table A.1.

Now, assume time is discrete with $0 = \tau_0 < \tau_1 < \tau_2 < \cdots < \tau_m$. Let $\mathbf{y}(\mathbf{x}) = [y_0(\mathbf{x}), y_1(\mathbf{x}), \ldots, y_m(\mathbf{x})]^T$ denote the output of a neural net with covariates $\mathbf{x}$ and a softmax output activation. Then, $\mathbf{y}(\mathbf{x})$ can be interpreted as the estimated probability mass function of the duration times $y_j(\mathbf{x}_i) = \hat{\mathrm{P}}(T_i^* = \tau_j \,|\, \mathbf{x}_i)$. The estimated survival function is then given by

$$
\hat{S}(\tau_j \,|\, \mathbf{x}) = 1 - \sum_{k=1}^{j} y_k(\mathbf{x}),
$$

and the discrete negative log-likelihood corresponding to the continuous version in (2), is

$$
\mathrm{loss}_L = - \sum_{i=1}^{N} \left[D_i \log(y_{e_i}(\mathbf{x}_i)) + (1 - D_i) \log(\hat{S}[T_i \,|\, x_i]) \right].
$$

Here, e_i denotes the index of the event time for observation i, i.e., $T_i = \tau_{e_i}$. Furthermore, DeepHit adds a second loss that attempts to improve its ranking capabilities,

$$
\mathrm{loss}_{\mathrm{rank}} = \sum_{i,j} D_i\, \mathbb{1}\{T_i < T_j\} \exp\left(\frac{\hat{S}(T_i \,|\, \mathbf{x}_i) - \hat{S}(T_i \,|\, \mathbf{x}_j)}{\sigma} \right). \tag{D.1}
$$

The loss of DeepHit is a combination of these two losses, where the ranking loss is scaled by a constant. We deviate slightly from the original implementation here and instead use a convex combination of the two,

$$
\mathrm{loss} = \alpha\, \mathrm{loss}_L + (1 - \alpha)\, \mathrm{loss}_{\mathrm{rank}}, \tag{D.2}
$$

where α and σ from (D.1) are considered hyperparameters.

References

Laura Antolini, Patrizia Boracchi, and Elia Biganzoli. A time-dependent discrimination index for survival data. *Statistics in Medicine*, 24(24):3927–3944, 2005.

David R. Cox. Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2):187–220, 1972.

Cameron Davidson-Pilon, Jonas Kalderstam, Ben Kuhn, Andrew Fiore-Gartland, Luis Moneda, Paul Zivich, Alex Parij, Kyle Stark, Steven Anton, Lilian Besson, et al. Camdavidsonpilon/lifelines: v0.14.1, 2018.

Lore Dirick, Gerda Claeskens, and Bart Baesens. Time to default in credit scoring using survival analysis: a benchmark study. *Journal of the Operational Research Society*, 68 (6):652–665, 2017.

David Faraggi and Richard Simon. A neural network model for survival data. *Statistics in Medicine*, 14(1):73–82, 1995.

Stephane Fotso. Deep neural networks for survival analysis based on a multi-task framework. *arXiv preprints arXiv:1801.05512*, 2018.

Thomas A. Gerds, Michael W. Kattan, Martin Schumacher, and Changhong Yu. Estimating a time-dependent concordance index for survival prediction models with covariate dependent censoring. *Statistics in Medicine*, 32(13):2173–2184, 2012.

Larry Goldstein and Bryan Langholz. Asymptotic theory for nested case-control sampling in the Cox regression model. *Annals of Statistics*, 20(4):1903–1928, 1992.

Erika Graf, Claudia Schmoor, Willi Sauerbrei, and Martin Schumacher. Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17-18):2529–2545, 1999.

Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*, 2016.

Frank E. Harrell Jr, Robert M. Califf, David B. Pryor, Kerry L. Lee, and Robert A. Rosati. Evaluating the yield of medical tests. *Journal of the American Medical Association*, 247 (18):2543–2546, 1982.

Patrick J. Heagerty and Yingye Zheng. Survival model predictive accuracy and ROC curves. *Biometrics*, 61(1):92–105, 2005.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *arXiv preprints arXiv:1609.04836*, 2017.

Hemant Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone, and Michael S. Lauer. Random survival forests. *Annals of Applied Statistics*, 2(3):841–860, 2008.

Jared L. Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. DeepSurv: personalized treatment recommender system using a Cox proportional hazards deep neural network. *BMC Medical Research Methodology*, 18(1), 2018.

Nitish S. Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

John P. Klein and Melvin L. Moeschberger. *Survival Analysis: Techniques for Censored and Truncated Data*. Springer, New York, 2. edition, 2003.

Bryan Langholz and Larry Goldstein. Risk set sampling in epidemiologic cohort studies. *Statistical Science*, 11(1):35–53, 1996.

Changhee Lee, William R. Zame, Jinsung Yoon, and Mihaela van der Schaar. Deephit: A deep learning approach to survival analysis with competing risks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Margaux Luck, Tristan Sylvain, Héloïse Cardinal, Andrea Lodi, and Yoshua Bengio. Deep learning for patient-specific kidney graft survival analysis. *arXiv preprint arXiv:1705.10245*, 2017.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

Daniel J. Sargent. Comparison of artificial neural networks with other statistical approaches. *Cancer*, 91(8):1636–1642, 2001.

L. N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

Gian A. Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820, 2015.

Terry M. Therneau. A package for survival analysis in S. Version 2.38, 2015.

Duncan C. Thomas. Addendum to: Methods of cohort analysis: appraisal by application to asbestos mining, by F. D. K. Liddell, J, C. McDonald and D. C. Thomas. *Journal of the Royal Statistical Society: Series A (General)*, 140(4):469–491, 1977.

Dirk Van den Poel and Bart Lariviere. Customer attrition analysis for financial services using proportional hazard models. *European Journal of Operational Research*, 157(1): 196–217, 2004.

Antonio Vigan, Marlene Dorgan, Jeanette Buckingham, Eduardo Bruera, and Mari E. Suarez-Almazor. Survival prediction in terminal cancer patients: a systematic review of the medical literature. *Palliative Medicine*, 14(5):363–374, 2000.

Anny Xiang, Pablo Lapuerta, Alex Ryutov, Jonathan Buckley, and Stanley Azen. Comparison of the performance of neural network methods and Cox regression for censored survival data. *Computational Statistics & Data Analysis*, 34:243–257, 2000.

Safoora Yousefi, Fatemeh Amrollahi, Mohamed Amgad, Chengliang Dong, Joshua E. Lewis, Congzheng Song, David A. Gutman, Sameer H. Halani, Jose E. V. Vega, Daniel J. Brat, et al. Predicting clinical outcomes from large scale cancer genomic profiles with deep survival models. *Scientific Reports*, 7(1):11707, 2017.

Xinliang Zhu, Jiawen Yao, and Junzhou Huang. Deep convolutional neural network for survival analysis with pathological images. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 544–547, 2016.

Xinliang Zhu, Jiawen Yao, Feiyun Zhu, and Junzhou Huang. WSISA: Making survival prediction from whole slide histopathological images. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6855–6863, July 2017.

III