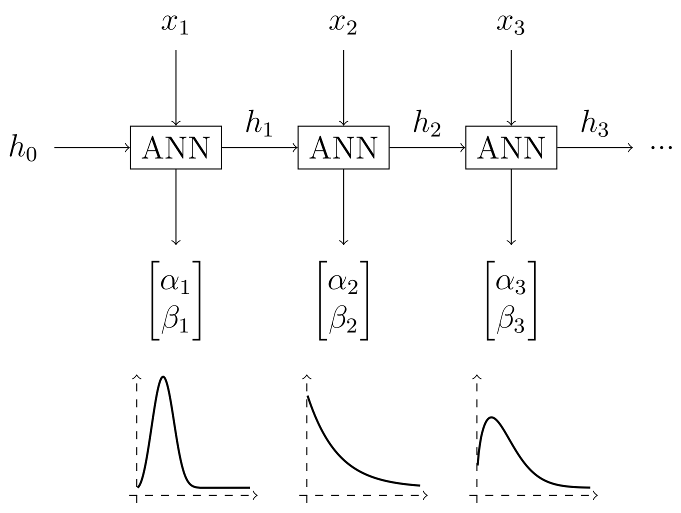

x_1
x_2
x_3
h_0
ANN
h_1
ANN
h_2
ANN
h_3
$\cdots$
$\begin{bmatrix}\alpha_1\\\beta_1\end{bmatrix}$
$\begin{bmatrix}\alpha_2\\\beta_2\end{bmatrix}$
$\begin{bmatrix}\alpha_3\\\beta_3\end{bmatrix}$

WTTE-RNN : Weibull Time To Event Recurrent Neural Network

A model for sequential prediction of time-to-event in the case of discrete or continuous censored data, recurrent events or time-varying covariates

Master's thesis in Engineering Mathematics & Computational Science

EGIL MARTINSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

WTTE-RNN : Weibull Time To Event Recurrent Neural Network

A model for sequential prediction of time-to-event in the case of discrete or continuous censored data, recurrent events or time-varying covariates

EGIL MARTINSSON

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

WTTE-RNN : Weibull Time To Event Recurrent Neural Network
A model for sequential prediction of time-to-event in the case of discrete or continuous censored data, recurrent events or time-varying covariates.
EGIL MARTINSSON

Supervisor: Fredrik D. Johansson, Dep. of Computer Science and Engineering
Examiner: Peter Damaschke, Dep. of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Simplest RNN architecture for the WTTE-RNN. In each timestep we use features x_t to update hidden state h_t and predict a distribution over time to the next event.

Typeset in LaTeX
Latex Template adapted from Frisk (2015)
Gothenburg, Sweden 2017

WTTE-RNN : Weibull Time To Event Recurrent Neural Network
A model for Sequential prediction of time-to-event in the case of discrete or continuous censored data, recurrent events or time-varying covariates
EGIL MARTINSSON
egil.martinsson[at]gmail.com
Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg

Abstract

In this thesis we propose a new model for predicting time to events: the Weibull Time To Event RNN. This is a simple framework for time-series prediction of the time to the next event applicable when we have any or all of the problems of continuous or discrete time, right censoring, recurrent events, temporal patterns, time varying covariates or time series of varying lengths. All these problems are frequently encountered in customer churn, remaining useful life, failure, spike-train and event prediction.

The proposed model estimates the distribution of time to the next event as having a discrete or continuous Weibull distribution with parameters being the output of a recurrent neural network. The model is trained using a special objective function (log-likelihood-loss for censored data) commonly used in survival analysis. The Weibull distribution is simple enough to avoid sparsity and can easily be regularized to avoid overfitting but is still expressive enough to encode concepts like increasing, stationary or decreasing risk and can converge to a point-estimate if allowed. The predicted Weibull-parameters can be used to predict expected value and quantiles of the time to the next event. It also leads to a natural 2d-embedding of future risk which can be used for monitoring and exploratory analysis.

We describe the WTTE-RNN using a general framework for censored data which can easily be extended with other distributions and adapted for multivariate prediction. We show that the common Proportional Hazards model and the Weibull Accelerated Failure time model are special cases of the WTTE-RNN.

The proposed model is evaluated on simulated data with varying degrees of censoring and temporal resolution. We compared it to binary fixed window forecast models and naive ways of handling censored data. The model outperforms naive methods and is found to have many advantages and comparable performance to binary fixed-window RNNs without the need to specify window size and the ability to train on more data. Application to the CMAPSS-dataset for PHM-run-to-failure of simulated Jet-Engines gives promising results.

Keywords: Censored, censoring, Weibull, Waiting Time, Recurrent Neural Networks, churn, time varying covariates, failure, Machine Learning, Deep Learning.

Acknowledgements

This thesis was in part written during an internship focusing on the problem at hand. I'd like to thank all the smart people at Tictail, especially Natalie, Jens, and Siavash for their help and inspiration. You gave me invaluable tutoring which helped me in my professional development and in understanding the problem which this thesis is aimed at solving. Working with this problem has been so fun that it was hard to stop. I would therefore like to thank my grandmother Mette Åberg for pushing me to finish up and hand in this thesis.

Egil Martinsson, Gothenburg, 2016

Contents

List of Definitions

List of Figures

1 Introduction

Times between events of interest are usually called event, survival, life, failure, hitting, stopping, or waiting- times. Modeling and predicting them is one of the core problems in classical statistics [1]. Examples are predicting the time between earthquakes or recessions, predicting the time a patient has left to live given treatment history, predicting the remaining useful life of a machine, setting the premium for a life insurance or estimating the timeframe in which a customer is expected to make a repeat purchase if ever.

The usual perspective on these types of problems is to predict the time *between* events. In this thesis we take the position that we want to make predictions about the time to the *next* event from where we are right now. This means that in each day, hour or second we want to use currently available data to update our prediction about how close we are to the next event.

We assume that we have recorded events with timestamps. These are placed on a timeline stretching from when we started recording until the last recording. In the discrete case we get a sequence $0, 0, 1, 0, 0 \ldots$ interpreted as occurence or non-occurence of events in the subintervals. This is visualized in figure 1.1.

As we're sequentially moving forward through time (x-axis) the current waiting time (y-axis) drops to 0 when we reach the next event creating a reversed sawtooth-wave. A coordinate (t, y_t) on this wave thus represents the time t and how far y_t it is to the next event from t. This thesis is solely focused on trying to use information available up to step t to approximate y_t when we have any or all of the problems of censoring, recurrent events, temporal patterns, time varying covariates or time series of varying lengths. We will briefly motivate each of them below.

Between the last recorded event and the end of the observation window we only know the lack of events. In that period we conclude that there was *at least* a certain amount of time until the next unseen event. This partial data-problem is called **censoring** [2]. See section 2.3 for details. Examples are when our dataset contains live patients, online web-servers, dormant volcanoes or customers. The current time to the next unseen event - be it death, downtime, eruptions or repeated purchases - is always censored. Censored data contains valuable information so we don't want to throw it away. In order to do so we need to apply special techniques. In some cases (i.e. death) an event nullifies the possibility of another. When this is not the case we are dealing with **recurrent events**.

Often we have more data than the event-logs themselves. We may have both data which is constant over time (static features) and data that changes over time (usually called **time-varying covariates** [1, 3]). If we take the example of e-commerce customers we typically have static data such as region and time of registration and

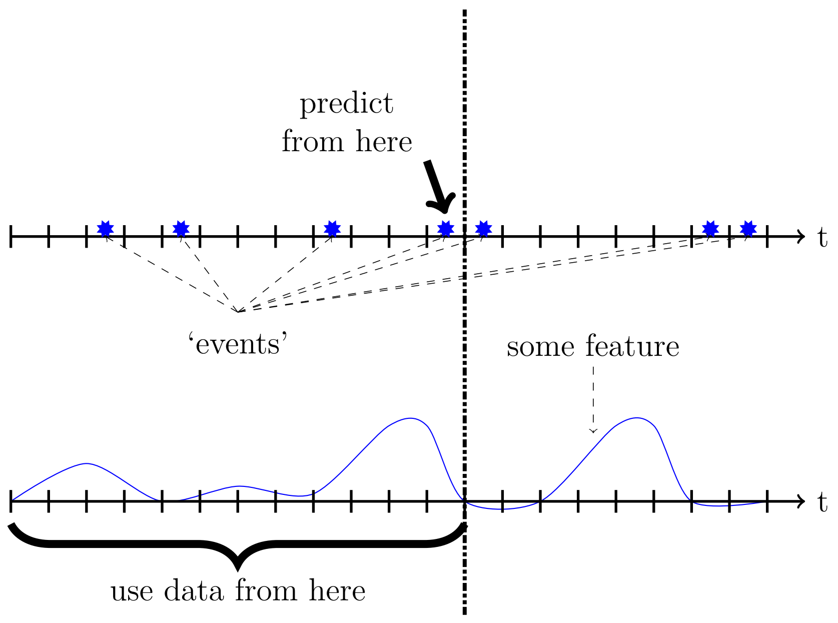


Figure 1.1: A timeline from the perspective of sequential prediction. In each step we want to use historic event- and feature- data to make predictions about time to future event.

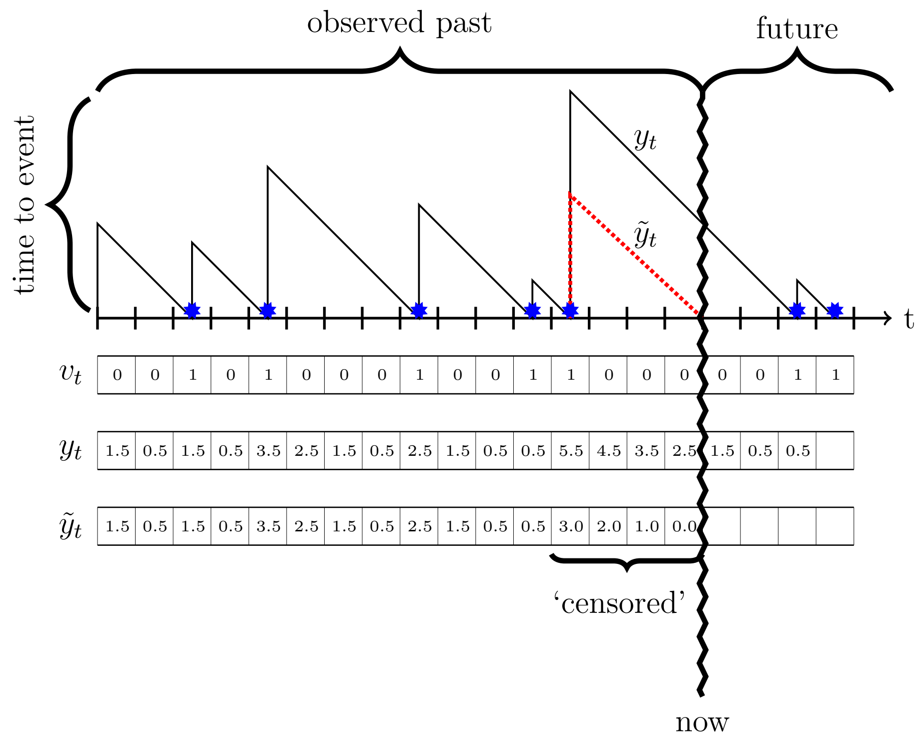

observed past
future
time to event
y_t
$\tilde{y}_t$
t
v_t 0 0 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 1 1
y_t 1.5 0.5 1.5 0.5 3.5 2.5 1.5 0.5 2.5 1.5 0.5 0.5 5.5 4.5 3.5 2.5 1.5 0.5 0.5
$\tilde{y}_t$ 1.5 0.5 1.5 0.5 3.5 2.5 1.5 0.5 2.5 1.5 0.5 0.5 3.0 2.0 1.0 0.0
'censored'
now

Figure 1.2: From the perspective of training: We have recorded events until now, written as indicators v_t. Prior to the last recorded event we know the true time-to-event so $\tilde{y}_t = y_t$. After we only have the censored time to event $\tilde{y}_t \leq y_t$ to train on.

time varying data like number of opened newsletters, clicked ads, lost carts and time since last purchase. If we want to build a model with static features we can simply treat each interval between events as the outcome of an experiment with parameters as functions of the static features. If we want to use time varying covariates to continuously update predictions we need other methods.

It's easy to see that there may be **temporal relationships** in the data that may have effect on the outcome. Unless we want to transform our variables to explicitly take these into account we need some way of learning the temporal patterns themselves. Examples could be to learn drug-interaction effects on patient life.

A further complication is that the length of each sequence may differ. If we don't have fixed-length sequences - which by the very nature of the problem we rarely have - we need some flexible method of dealing with it.

We may summarize the problem as follows:

Given that we've observed sequences of features and events where each sequence may be of varying length and time-to-event may be censored; find a way to to use feature history at each step to predict the time to the next event.

A very simple example that sums up the whole problem can be seen in figure 1.2. Assume that we've taken feature measurements (ex. readings from a barometer) and recorded some events, say lightning strikes. We want a model able to make real time predictions. Here by predicting say time to the next lightning using historic data of lightning strikes and barometric pressure.

This seems like a very generic forecasting problem. As such we'd expect there to be an abundance of models. Rather surprisingly there's no standard sequential model for censored data. Instead, as we'll show in section 2.4, the usual method of tackling the issue of censoring, time-varying covariates and recurrent events in waiting-time prediction seems to be by re-framing or avoiding it completely.

The goal with this thesis is to describe a model that seems like an intuitive, empirically sound and computationally feasible way of learning long term dependencies in sequences of varying length in order to continuously predict and make inference about waiting-times.

The proposed solution is very simple and quite obvious. We discretize the timeline into steps and assume a parametric distribution for the time to the next event. We then let the parameters of this distribution be functions of data. This way we may compare density at points (uncensored data) and probability over regions (censored data) by utilizing a special log-likelihood for the parametric model as loss-function. In particular, by letting the parameters in each step be outputs of a recurrent neural network we may learn temporal patterns in sequences of varying lengths. We focus on the Weibull model as the parametric model but this can easily be switched out in the framework we will present.

1.1 Method

We have used mathematical methods to theoretically justify the model and experiments to verify its performance. The experiments were designed to test performance and exemplify its use. In the first experiment (section 4.1) we explore the dynamics of simple gradient based estimation using the model. This can be explicitly shown since we controlled the parameters generating the data. In section 4.2 we tested the model on a well known dataset of uncensored failure times and high dimensional feature data. This shows how the model can be used in practice.

In the remaining experiments we compare the models with some relevant baselines on generated data. This also served the purpose of exploring which negative and biasing effect training with censored data had on predictions. In section 4.3 we compared performance with some naive but realistic baselines. This was evaluated using generated (non-random) ground truth data. In section 4.4 we compare our model to a standard model type for censored data. Note that this approach (which we call the *sliding box model* tackles the problem in a completely different way, but as its output is a special case of the WTTE-RNN we can compare them.

1.2 Scope

In this thesis we touch on many subjects that come together to form the model. We have tried to cover all of them but we have had to limit the depth of the analysis to that which is relevant in answering the basic questions.

In the theoretical sections we cover the basics necessary to understand the quantities that we are trying to predict and to give some probabilistic justification of our model. The basic tools stems from the theory of stochastic processes and survival analysis which is a vast research-area meaning that we can only scratch the surface. We limited ourselves to give the most basic tools and to craft a notational machinery so that we can talk about waiting times in an efficient manner. This was done by condensing some known theory and low-hanging mathematical fruit and present them in the context of machine learning and optimization. For an in-depth discussion on survival analysis we refer to [4].

Apart from the probabilistic background we focused on answering the questions of whether there's unique solutions, if we can guarantee convergence and which numerical caveats that might be hidden when calculating gradients. We hope that this this is useful for someone that wishes to understand, use and in particular extend the model. We tried answering when the loss function leads to a probabilistic model that makes sense. In the case of recurrent events we didn't fully manage to construct a complete generative scenario leading to our log-likelihood loss function. We did on the other hand show through experiments that it works well in practice for a range of problems.

We will briefly discuss Recurrent Neural Networks but for a more thorough introduction we refer to [5].

1.3 Thesis outline

The findings can be divided into a theoretical background describing probabilistic aspects of models for times to events, a theoretical description of how we assemble these pieces into the proposed model and experiments where we test it.

In preliminaries (chapter 2) we develop a mathematical framework to talk about waiting times. This is used to show how data can be generated (section 2.2). We give an introduction to the Weibull Distribution (section 2.5) to motivate its use and a very short introduction to Recurrent Neural Networks (section 2.6). Previous work and similar models is then described (section 2.4) using the machinery presented earlier.

In chapter 3 we give a thorough description of the proposed model. The learning-problem is defined in terms of an optimization problem which is given both theoretical and practical motivations. We analyze numerical caveats and possible tweaks. A proposed workflow for using it is given and some proposed applications, with particular focus on churn-prediction.

In experiments (chapter 4) we show some results and performance with real and generated data.

Discussion (chapter 5) summarizes the findings and discuss model strengths, weaknesses and suggests further work.

2
Preliminaries

In order to understand the WTTE-RNN we will give a description of its pieces. We will start of with some definitions to lay the groundwork for a general discussion about censoring and modeling of waiting times and censored data. We then use this background to present some related models (section 2.4), which the reader can skip to immediately if so inclined. Survival analysis is quite a mature field but we have not found any single source covering it in a comprehensive mathematical and minimal manner relevant to our problem. This is the motivation for this chapter. Almost everything presented here are standard statistical concepts or conclusions relying only on basic manipulations. It can be found in one form or another in standard textbooks such as [4]. We have tried to condense the core results using the most common notation and terminology when possible.

Definition 2.1 (Waiting times)**.** We refer to data or random variables that are modeling time between-, since- or to events as *waiting times.* We assume that waiting times T are always positive and unbounded, $T \in [0, \infty)$. Waiting times may either be continuous or discrete.

Some properties that makes this kind of data interesting is:

- Abundance of data. Whenever we have events with timestamps we have timings and times between such events.
- The experiment itself interferes with our ability to observe it. We can't observe it until it's over and the time it takes to conduct an experiment is the experiment itself.
- Positive and unbounded. $T \in [0, \infty)$. This is an assumption that might not always make sense.

In some contexts an event occurring nullifies the probability of another occurring. This is the case when modeling survival e.g. lifelengths of patients. This can be seen as a special case of the focus of this thesis, namely events happening on a timeline in succession, *recurrent events.*

Definition 2.2 (Recurrent events)**.** Given a timeline $[0, \infty)$ and marked points that may be more than one, we call these points *recurrent events.* The time to the next event from some position on this timeline is called *time-to-event* or less specific *waiting time.* Time between such events are typically called *interarrival times* [4].

Time to events are easily visualized as a kind of reversed sawtooth-wave as shown in figure 1.2.

2.1 Probability theory for waiting times

This section gives the basic building blocks and terminology of survival-analysis in the context of temporal problems. Another way to see it is that this section defines the quantities that the WTTE-RNN is trying to predict.

We will start off with the case of the time to one event at a time and then adapt it to the case when we have recurring events. Without loss of generality we initially focus on continuous distributions and later show how to transform them to discrete distribution. These can in turn be used to generate data.

All our theory stems from a useful representation of a positive probability distribution, the simple concept of *hazard* and *cumulative hazard*-functions.

Definition 2.3 (Hazard Function $\lambda(t)$ (HF)). A hazard function λ is a function with the properties:

1. $0 \leq \lambda(t)\ \forall t \geq 0$
2. $\int_0^\infty \lambda(w)dw = \infty$

Definition 2.4 (Cumulative Hazard Function (CHF)). We denote the integrated Hazard Function with Λ (pronounced 'Lambda') and call it a *Cumulative Hazard Function*:

$$\Lambda(t) = \int_0^t \lambda(w)dw$$

Proposition 2.5. *$\Lambda(t)$ induces a CDF for a positive random variable T with CDF:*

$$\Pr(T \leq t) = F_T(t) = 1 - e^{-\int_0^t \lambda(w)dw} = 1 - e^{-\Lambda(t)}$$

and PDF:

$$f_T(t) = F_T'(t) = \frac{\partial}{\partial t}(1 - e^{-\Lambda(t)}) = e^{-\Lambda(t)}\lambda(t)$$

Proof. Observing that $0 \leq \lambda(t)$ so $\Lambda(t)$ is an increasing function and $f_T(t) \geq 0$. Furthermore with $e^{-\Lambda(0)} = 1 \to F_T(0) = 0$ and $e^{-\Lambda(\infty)} = 0 \to F(\infty) = 1$. We see that the statement holds. □

Conversely, for every distribution we may find Λ and λ as defined above:

Proposition 2.6. *For each CDF F_T there exists Λ such that $\Lambda = -\log(1 - F_T)$*

Proof is omitted since it's self evident. Note that since $\Lambda(t)$ is strictly increasing it has an inverse and so the quantile function $F_T^{-1}(p) = \Lambda^{-1}(-log(1-p))$.

In terms of survival, higher values of the hazard function indicates higher mortality given survival up to that point. This can also be seen from the identity

$$\lambda(t) = \lim_{\epsilon \to 0} \frac{Pr(t < T \leq t + \epsilon | T > t)}{\epsilon}$$

This motivates why λ is also called the *instantaneous risk* or the *failure rate* [2].

In survival analysis we're mainly concerned with the time *remaining* to an event so we spend more time talking about the right tail of distributions. As a convenience we introduce notation for this:

Definition 2.7 (Survival function)**.** The function for the probability of an event occurring after time t (survival up to time t) is called the *survival function* $S(t)$:

$$S(t) = \Pr(t < T) = 1 - F(t) = e^{-\Lambda(t)}$$

In figure 2.1 we show an example of how the quantities so far are related.

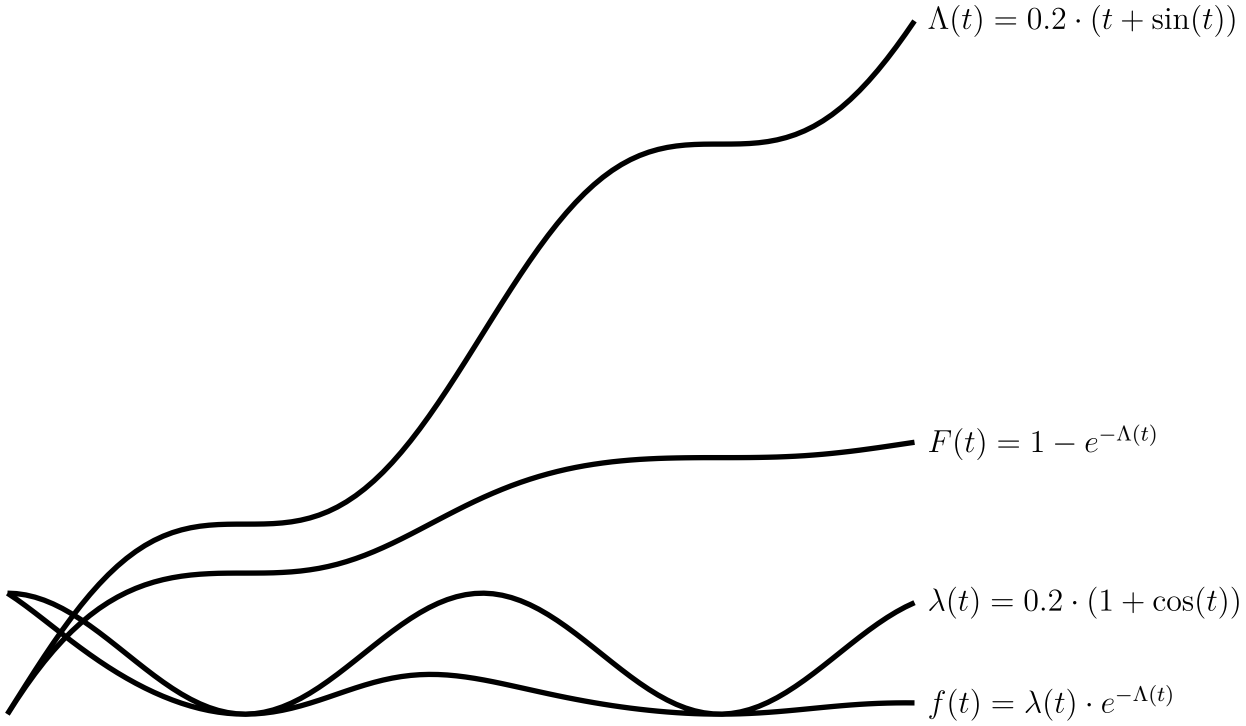


Figure 2.1: Relationship between Cumulative Hazard Λ, Hazard $\lambda = \Lambda'$, Cumulative Distribution Function F and Probability Density Function $f = F'$.

In our context it's also useful to define a shorthand for the distribution of T given that an event have yet to occurred at time t:

Definition 2.8 (Conditional Excess Distribution)**.** If $Y_t = \{T - t\} | \{T > t\}$ we call Y_t the *conditional excess distribution* of T at t. If T has CDF F_T we denote the CDF of Y_t with $F(t, s)$:

$$\begin{aligned} F(t,s) = && Pr(Y_t \leq s) = && Pr(T \leq t + s | T > t) \\ = && \frac{Pr(T \leq t + s \cap T > t)}{Pr(T > t)} = && \frac{F_T(t+s) - F_T(t)}{1 - F_T(t)} \\ = && \frac{S_T(t) - S_T(t+s)}{S_T(t)} = && \frac{e^{-\Lambda(t+s)} - e^{-\Lambda(t)}}{e^{-\Lambda(t)}} \\ = && 1 - e^{-(\Lambda(t+s) - \Lambda(t))} && \end{aligned}$$

We interpret this as 'at time t the probability of an event within s time is $F(t, s)$'

Its PDF:

$$f(t,s) = \frac{\partial}{\partial s} \frac{F_T(t+s) - F_T(t)}{1 - F_T(t)} = \frac{f_T(t+s)}{1 - F_T(t)} = \frac{f_T(t+s)}{S_T(t)} = e^{-(\Lambda(t+s) - \Lambda(t))} \lambda(t+s)$$

For notational convenience we will also use the survival-function in this context and let $1 - F(t, s) = S(t, s)$.

As will be shown, conditional excess distributions are very useful for modeling recurrent events.

2.1.1 Waiting times for recurrent events

We are interested in recurring events over a timeline. Note how simple the conditional excess distribution got when we formulated it using the general notion of HFs and CHFs instead of regular CDF/PDF-notation. For recurrent events we've constructed a more general concept which makes it easier to talk about waiting time. We will construct the quantities discussed above using what we introduce as a generalized Cumulative Hazard Function:

Definition 2.9 (Recurrent Cumulative Hazard Function (RCHF))**.** We call a function $R : \mathbb{R}^m \times \mathbb{R}_+ \to \mathbb{R}_+$ such that $\forall X \in \mathbb{R}^m$:

1. $\frac{\partial R}{\partial s}(x, s) \geq 0 \, \forall s \geq 0$
2. $R(x, s) \to \infty$ as $s \to \infty$
3. $R(x, s) = 0 \forall s \leq 0$

a *Recurrent Cumulative Hazard Function.*

From the definition of a hazard function, $\frac{\partial R}{\partial s}(x, s) = R_s(x, s)$ with x fixed is a hazard for events over the coming time $s \in [0, \infty)$. By proposition 2.5 we know that we may use this to create a probability distribution $1 - e^{-R(x,s)}$. We can see it as if $\Pr(S \leq s|X = x) = 1 - e^{-R(x,s)}$. The proposed model we will let x be the input data.

For the generative framework we will be concerned with the special case when $x = t$ represents the current position on a timeline from where we want to make predictions about the future, represented by s. In this framework the coordinate (t, s) can be interpreted as *looking s steps into the future from time t.*

When R_s has an additional property that we will call *symmetry* we can seamlessly connect it to conditional excess distributions.

Definition 2.10 (Symmetric RCHF)**.** When the RCHF has the additional property $R(t, s) = R(0, t + s) - R(0, t)$ we call it *symmetric.*

Proposition 2.11. *For every symmetric RCHF there is some CDF $F_T(t)$ such that its conditional excess CDF can be written $F(t, s) = 1 - e^{-R(t,s)}$*

Proof. In order to show this we need to pick some R and create a valid CDF F_T and then show that this is identical to the definition of a conditional excess distribution as in definition 2.8.

Clearly $R(t, s) \geq 0$ and $R(t, s) \to \infty \; \forall t$ as $s \to \infty$ giving $R(t, s)$ the desired properties to define PDFs and CDFS in s for each fixed t like in proposition 2.5. Abusing notation we define F and f as:

$$
\begin{aligned}
F(t,s) &= 1 - e^{-R(t,s)} \\
f(t,s) &= \frac{\partial}{\partial s} F(t,s) = e^{-R(t,s)} R_s(t,s)
\end{aligned}
$$

Further set $F(0,t) = 1 - e^{-R(0,t)} = F_T(t)$. Now set the LHS as the definition of a conditional excess CDF:

$$
\begin{aligned}
\frac{F_T(t+s) - F_T(t)}{S_T(t)} &= \frac{(1 - e^{-R(0,t+s)}) - (1 - e^{-R(0,t)})}{e^{-R(0,t)}} \\
&= 1 - e^{-(R(0,t+s) - R(0,t))} = 1 - e^{-R(t,s)} = 1 - F(t,s)
\end{aligned}
$$

Where we used that $R(t,s) = R(0,t+s) - R(0,t)$.

□

This is really a reiteration of prop. 2.5 using the new notation. The concept of symmetry is interesting as it tells us how the RCHF must look like to induce a conditional excess distribution over the same time line and thus gives us a hint about what to think about when analyzing random hazard-processes. It might be apparent but its still worth mentioning, if R is symmetric there is a unique hazard function integrated over $[t, t+s]$ which defines it:

Proposition 2.12. $R(t,s) = R(0,t+s) - R(0,t) \iff R(t,s) = \Lambda(t+s) - \Lambda(t)$

Proof. $\Longrightarrow$: If R is as above then $R_s(t,s) = R_s(0,t+s)$. As R_s has the properties of a hazard function use the notation $R_s(0,t+s) := \lambda(t+s)$. We thus have $R(t,s) = \int_0^s R_s(t,x)dx = \int_0^s \lambda(t+x)dx = R(t,s) - R(t,0)$ since $R(t,0) = 0$. By the definition of a cumulative hazard function we have $R(t,s) = \Lambda(t+s) - \Lambda(t)$.

$\Longleftarrow$: Conversely, with $R(t,s) = \Lambda(t+s) - \Lambda(t)$ then it's easily verified that R has the properties of a recurrent cumulative hazard function. Furthermore $R(0,x) = \Lambda(x)$ and so $\Lambda(t+s) - \Lambda(t) = R(0,t+s) - R(0,t) = R(t,s)$ □

It might be helpful to consider $(x,y,z) = (t,s,R_s(t,s))$ as coordinates to a surface in 3d for $t,s \geq 0$. When R is symmetric the surface will be symmetric around $t = s$ as each absolute (wall clock) point in time has the same hazard value $R_s(t,s)$. See for example figure 4.13.

If R isn't symmetric we may assign different different hazard values $R_s(t,s)$ to the same absolute time $t+s$ depending on from which t we are making the prediction. One interpretation of $R(t,s)$ is therefore that it gives the *belief* at t about the distribution over future events at time $t+s$. See for example the induced distribution of figure 4.5.

2.1.2 Discretized case

It's common to deal with discrete time (seconds, days, months) so it's useful to use the same framework for discrete- as for continuous problems. We could deal with discrete time by dividing the timeline using some arbitrary steplength and index the resulting intervals as done by [3]. To simplify calculations we will instead assume unit steplength.

Here let Λ be some cumulative hazard function as above. Use the notation $\Lambda(t+s) - \Lambda(t) = R(t,s)$ with R a symmetric RCHF

Definition 2.13 (Step Cumulative Hazard Function $d(t)$ (SCHF))**.** We define the SCHF $d(t)$ to be the hazard function integrated over integer intervals $[t, t+1)$ for $t = 0, \ldots$

$$d(t) = R(t,1) = \Lambda(t+1) - \Lambda(t) = \int_t^{t+1} \lambda(w)dw$$

We can extend our definition of conditional excess distributions to encompass discretized random variables.

Definition 2.14 (Conditional Excess Cumulative Mass Function $P(t,s)$ (CECMF))**.** Let T_d be the discretization of the random variable T with CHF Λ s.t $T_d = t \iff T \in [t, t+1)$. We define the probability that there will be *at least* 1 event *within* s timesteps from t as

$$\begin{aligned}
P(t,s) &= \Pr(T_d \leq t+s | T_d \geq t) \\
&= \Pr(T \leq t+s+1 | T \geq t) \\
&= 1 - \exp\left(-R(t,s+1)\right) \\
&= 1 - \exp\left(-\sum_{k=t}^{t+s} d(k)\right)
\end{aligned}$$

Definition 2.15 (Conditional Excess Probability Mass Function $p(t,s)$ (CEPMF))**.** The probability that there will be *an* event *in* s steps from t:

$$\begin{aligned}
p(t,s) &= \Pr(T_d = t+s | T_d \geq t) \\
&= \Pr(T \in [t+s, t+s+1) | T \geq t) \\
&= S(t,s) - S(t,s+1) \\
&= e^{-R(t,s)} - e^{-R(t,s+1)} \\
&= e^{-R(t,s)}(1 - e^{-d(t+s)}) \\
&= e^{-R(t,s+1)}(e^{d(t+s)} - 1) \\
&= e^{-\sum_{k=t}^{t+s} d(t)}(e^{d(t+s)} - 1)
\end{aligned}$$

Definition 2.16 (Discrete survival function $S_d(t)$)**.** The discrete survival function is the probability of survival of t steps or more i.e. that the event happens at step t or later.

$$S_d(t) = Pr(T_d \geq t) = e^{-\Lambda(t)} \tag{2.1}$$

For a discretized variable $S_d(t) = S(t) = Pr(T \geq t)$. Note that even though the SCHF have very similar properties to the continuous hazard function, we refrained from using the name 'discrete hazard'. The reason being that we reserve the word 'hazard' for its common usage, namely when it's the quota between the probability of event and the probability of no event so far, i.e. the PMF and the survival function.

Definition 2.17 (Discrete Hazard Function λ_d)**.** Let T be a positive random variable. For the discrete random variable T_d induced from it, we define its *discrete hazard* as [3]:

$$\begin{aligned}\lambda_d(t) = \quad & \frac{\Pr(T_d = t)}{S_d(t)} = \quad & Pr(T_d = t|T_d \geq t) = \quad & Pr(T \in [t, t+1)|T \geq t) \\ = \quad & F(t,1) = \quad & 1 - e^{-(R(t,1))} = \quad & 1 - e^{-d(t)}\end{aligned}$$

An interesting aspect of the discrete hazard function is that we can factor the survival function as $S_d(t,s) = (1-\lambda_d(t)) \cdot \ldots \cdot (1-\lambda_d(t+s)) = e^{-d(t)-\ldots-d(t+s)}$ [3]. A probabilistic interpretation is that for survival of s steps we need s independent non-events to happen. This interpretation will prove useful when generating data.

Before moving on we will state a short proposition about the expected values of a unit discretized distribution T_d of T:

Proposition 2.18. $E[T] \leq E[T_d] + 1$

Proof. We can rewrite the expectations as:

$$E[T] = \sum_{t=0}^{\infty} \int_t^{t+1} x f(x) dx$$

$$E[T_d] = \sum_{t=0}^{\infty} t \int_t^{t+1} f(x) dx$$

And since $xf(x) \leq (t+1)f(x)$ for $x \in [t, t+1]$ the same holds for the sum over each $t = 0, 1, \ldots$ so:

$$E[T] \leq \sum_{t=0}^{\infty} (t+1) \int_t^{t+1} f(x) dx = E[T_d] + 1$$

□

So the expected value of a unit discretized random value is at most one step above the expected value of the continuous distribution. This is a useful estimate when we can calculate the continuous expected value but not the discretized or vice versa.

2.2 Generating recurrent events using hazards

In this section we will show how continuous hazard functions can be discretized and used to generate data.

We used this method to generate data for the experiments of chapter 4, but discussing them here serves a more important purpose. In the real world we have no idea what kind of processes that generated the data i.e where the data *came from* - that's why we are using machine learning in the first place. This model makes use of probability theory and assumptions about the distribution over the time to the next event for learning. The easiest way to understand these assumptions is to dwell in an idealized version of the real data-generating process (a generative model). This makes it easier to recognize whether these assumptions holds in practice.

To generate event data define the random variable V_t as an event occurring or not in intervals indexed $t = 0, 1, 2, \ldots$:

$$V_t = \begin{cases} 1 & \text{if } U_t \leq 1 - e^{-d(t)} \\ 0 & \text{else} \end{cases}$$

where U_t is a Uniform random variable. Inversely we get:

$$\Pr(V_t = 1) = \Pr\left(\text{event in interval } [t, t+1)\right) = 1 - e^{-d(t)}$$

So V_t is a Bernoulli-trial with parameter $\theta_t = 1 - e^{-d(t)}$. We will show that at each step, the time to the next event, i.e. the number of steps s to the next bit which is turned on ($V_{t+s} = 1$) has the distribution described by the CECMF:

Proposition 2.19. *By generating recurrent events as above, the distribution of the time to the next event is given by the CEPMF*

Proof. We may make the above statement a little more clear. Assume that we're standing at step t. Define V_k as above with $k = t, \ldots$. If T_t is the discrete random variable generated at step t indicating time to the next event with distribution induced by Λ, then we claim that:

$$\begin{gathered} Pr(\text{At time } t \text{ there's } s \text{ steps to the next event}) = \\ Pr(V_t = 0, V_{t+1} = 0, \ldots, V_{t+s} = 1) = Pr(T_t = s) = p(t, s) \end{gathered}$$

We can get some intuition about this by thinking about how a geometrically distributed random variable X with PMF $\Pr(X = s) = (1 - \theta)^{s-1}\theta$ is constructed. Here we need a sequence of $s - 1$ failed independent Bernoulli-trials followed by one success, where each trial has *identical* success probability θ. Let $s > 0$ and imagine a geometric experiment with *different* probabilities of success, namely $\theta_t = 1 - e^{-d(t)}$:

$$\begin{aligned} & Pr(V_t = 0, V_{t+1} = 0, \ldots, V_{t+s} = 1) \\ &= Pr(V_t = 0) \cdot \Pr(V_{t+1} = 0) \cdot \ldots \cdot \Pr(V_{t+s} = 1) \\ &= Pr(V_{t+s} = 1) \prod_{k=t}^{t+s-1} Pr(V_k = 0) \\ &= \theta_{t+s} \prod_{k=t}^{t+s-1} (1 - \theta_k) \\ &= (1 - e^{-d(t+s)}) \prod_{k=t}^{t+s-1} e^{-d(k)} \\ &= (1 - e^{-d(t+s)}) e^{-\sum_{k=0}^{s-1} d(t+k)} \\ &= p(t, s) \end{aligned}$$

And since $Pr(V_t = 1) = 1 - e^{-d(t)} = p(t, 0)$ we are done. ☐

To convert the event indicators to discrete time-to-event data we simply count the steps to the next event:

$$\begin{gathered} y_t = \arg\min_k v_{t+k} \\ \text{s.t } v_{t+k} = 1 \text{ OR } k = n \end{gathered}$$

Note that $y_t = 0 \iff v_t = 0$. This should be interpreted as if, at time t, there will be an event in the coming time-interval $[t, t+1)$ which is not known at time t.

The case when there is no event until the end of time, $k = n$ means we have censored data so we set $u_t = 0$.

To generate non-censored data we generate events over a longer time-horizon, say $t = 0, \dots, n, \dots, N$ choosing $N >> n$ such that the probability of no events for $t = n, \dots, N$ is negligible $\Pr(V_n = V_{n+1} \dots = V_N = 0) \approx 0$. Calculate the waiting times as

$$\begin{gathered} y_t = \arg\min_k v_{t+k} \\ \text{s.t } v_{t+k} = 1 \text{ OR } k = N \end{gathered}$$

and keep y^j for $t = 0, \dots, n$.

With this in mind the we may get some more intuition around hazard functions. When we discretize the timeline $p(t, 0) = \lambda_d(t)$ gives us the probability of an event in each of them indexed $t = 0, \dots$. By shrinking the width ϵ of this interval and generating events, eventually $\lambda_d \to 0$ as $\epsilon \to 0$. Recall the identity $\lambda(t) = \lim_{\epsilon \to 0} \frac{Pr(t<T \leq t+\epsilon | T>t)}{\epsilon}$. This motivates why $\lambda(t)$ indicates how high concentration-, or the *rate* of events we are expected to encounter around t.

Note that we do not need an explicit form of $\lambda(t)$ to generate events, only some positive values $d(t)$. This fact will be used in the following example to show how events can be generated.

2.2.1 Example: Generating random events from a random process

In this example we will show how events can be generated using a RCHF. This is then used to illustrate the difficulties in predicting it.

Assume that we have an event-density i.e hazard determined by the process given by the solution to some Stochastic Differential Equation (SDE), the Ito-integral:

$$X(t) = X(0) + \int_0^t \sigma(s) dB(s) + \int_0^t \mu(s) ds$$

where we informally write its *stochastic differential* :

$$dX(t) = \sigma(t) dB(t) + \mu(t) dt$$

with $\mu(s)$ and $\sigma(s)$ deterministic functions and $B(t)$ brownian motion so that $dB(t) = B(t + dt) - B(t)$ [6]. Now set $\lambda(t) = |X(t)|$ and $\Lambda(t) = \int_0^t |X(t)| dt$ s.t

$$Pr(\text{event within } s \text{ time at time t}) = 1 - \exp(-\int_0^s |X(t+\tau)| \, d\tau)$$

So the hazard function is a Markov random process.

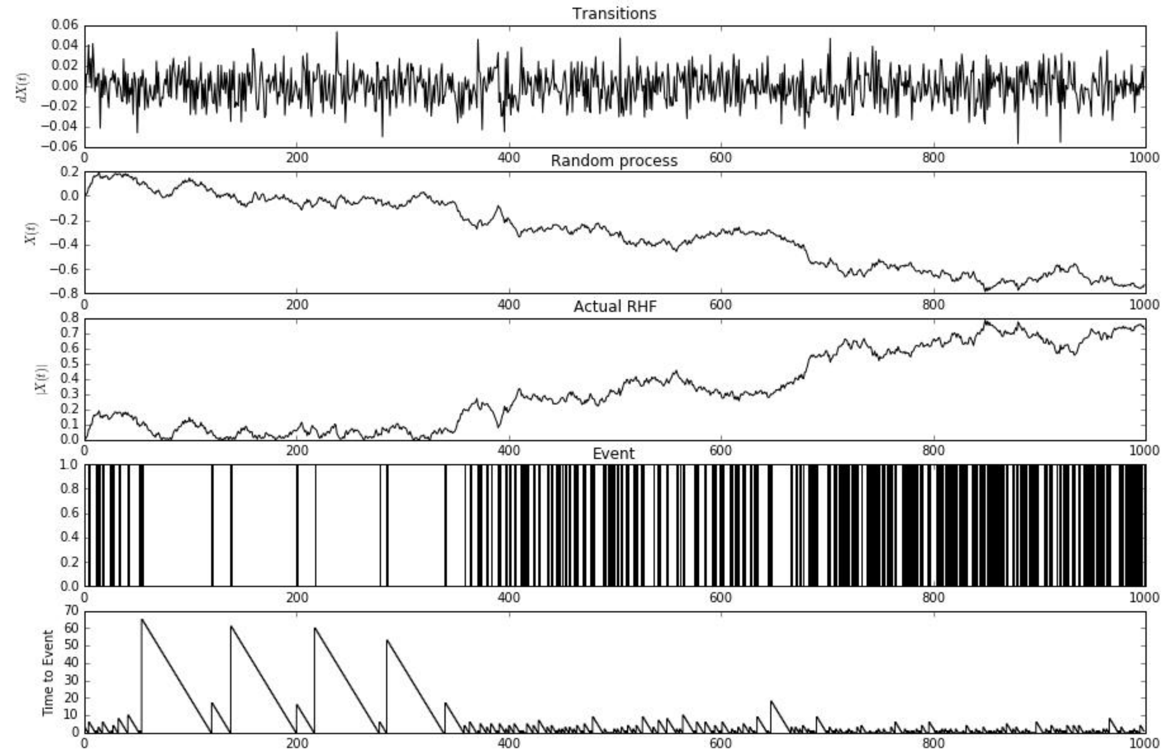


Figure 2.2: Example of Gaussian noise $dX(t)$, its Brownian motion $X(t)$, some random hazard function $|X(t)| = \lambda(t)$ and events generated from it using $d(t) = \Lambda(t+1) - \Lambda(t) \approx \lambda(t)$

At prediction-time we only - at best - have information about the process $X([0,t])$ and event-times leading up to present time t. We may not have information about the future hazard as it hasn't yet been realized. Note that the expected trajectory of the hazard will change depending on its current state or rather what we know about its current state. The goal is therefore to estimate $E\left[\Lambda(t+s) - \Lambda(t) | \text{Info at time } t\right] := R(t,s)$.

The *info at time t* are the features. Realistically these could be a sequence of snapshots of the process $\{X(t_i)\}_{i=1}^n$. More realistically still, we've only observed transitions in a noise-variable, say the transitions $\{dB(t_i) = B(t_i) - B(t_{i-1})\}_{i=1}^n$.

At training-time, if we've observed the hazard-function we want to find a way to predict its trajectory. Since this is an unrealistic assumption, we really only have feature-history and the events themselves rather than their event-density to infer the process.

2.3 Censoring

In this section we will define the problem of censoring and present the common loss function (log-likelihood) used when we have this type of data.

Censoring is one of the special problems with waiting times. As the time it takes to conduct a waiting-time experiment is the experiment itself we may not always be able to observe it from start to end. This phenomena is called censoring:

Definition 2.20 (Censored data)**.** Let T be some positive random variable indicating time-to-event. We say that a datapoint/observation of T is *censored* whenever we haven't observed it pointwise. It may also be useful to talk about it as the *actual* and *observed* time of the experiment. The actual waiting time is always contained in the observed. When the actual is equal to the observed we have an uncensored observation and otherwise a censored observation. In particular, we call it:

Uncensored: When T is known to have been t: $T = t$. Event occurred exactly at time t.

Left censored: When T is known to have been *below* a treshold t but unknown by how much: $T \in (0, t)$. After time t we know that the event *has* occurred but not exactly when.

Right censored: When T is known to have been *above* a treshold t but unknown by how much: $T \in (t, \infty)$. After time t we know that the event *hasn't* occurred but we know it will eventually.

Interval censored: T is known to have been *in* an interval, $T \in [a, b]$.

Even though all three types of censoring are frequently encountered in the wild, this thesis will solely focus on right-censoring. This follows from that if we have an uninterrupted observation window and looking forward, the *time to event* can only be right censored.

The random process leading up to right censoring is usually described as if we've observed the random variable pair (X, Δ) to equal (x, u) where $X = \min(T, C)$ with T the waiting time, C the random censoring variable and $\Delta = I(T \leq C)$ the failure indicator s.t:

$C < T$	$\Delta = 0$	Observation censored
$C \geq T$	$\Delta = 1$	Observation uncensored

A high-level distinction between the processes causing censoring is if the censoring is *informative* or *non-informative*. For more detailed descriptions we recommend [4].

Definition 2.21 (Informative censoring)**.** Let $X = \min(T, C)$ with T parameterized by θ. If

1. $C \perp T|\theta$

2. $C \perp \theta$

We say that we have *non-informative censoring*, otherwise we call the censoring process *informative*. In the latter case, observing C gives us information about the distribution of T or its parameters.

Even if both conditions needs to hold for non-informative censoring it's useful to separate them. The first implies that $\Pr(C,T|\theta) = \Pr(C|\theta)\Pr(T|\theta)$ while the second furthermore implies $\Pr(C,T|\theta) \propto \Pr(T|\theta)$. If for example T and C are IID random variables with common parameter θ we have 1) but not 2).

The assumption of non-informative censoring will be central motivating the use of observations (x_i, δ_i) to estimate the distribution of T.

2.3.1 Log-Likelihood for right censored observations

The idea for using log-likelihood for censored data is that we assume the censoring process to be independent from the parameter we are optimizing for meaning that we can factor it out from the likelihood function. In this section we will prove that under the assumption of non-informative censoring we get a very nice log-likelihood formulation. Before going into details we present the basic definition of Log Likelihood using the notation of Casella [7].

Definition 2.22 (Likelihood function)**.** Let $f_{\mathbf{X}|\theta}(\mathbf{x})$ denote the joint pdf or pmf of the sample $\mathbf{X} = \{X_1, \dots, X_n\}$ where f has parameter θ. Then given that $\mathbf{X} = \mathbf{x} = \{x_1, \dots, x_n\}$ is observed, the function

$$\mathfrak{L}(\mathbf{x}, \theta) = f_{\mathbf{X}|\theta}(\mathbf{x}) \tag{2.2}$$

Is called the *likelihood function*, or *likelihood* for short [7].

We are interested in the likelihood as a loss function for θ. By maximizing the likelihood function we maximize the probability (or 'likelihood' when we have a continuous distribution) of our observation $\mathbf{X} = \mathbf{x}$ given its parameter θ. Note that if $\mathbf{X}$ consists of independent identically distributed random variables observed as $X_k = x_k$ then their joint distribution factor and $\mathfrak{L}(\mathbf{x}, \theta) = \prod_{k=1}^{n} f_{X|\theta}(x_k)$ and so the log of the likelihood will be a sum of some function $\phi(x_k, \theta)$ involving θ and those not $g(x_k)$, meaning that the $g(x_k)$'s have no influence on the maximization-procedure why we disregard them and say that the likelihood is proportional ($\propto$) the right hand side.

$$\log(\mathfrak{L}(\mathbf{x}, \theta)) = \sum_{k=1}^{n} \log(f_{X|\theta}(x_k)) = \sum_{k}^{n} \phi(x_k, \theta) + g(x_k) \propto \sum_{k}^{n} \phi(x_k, \theta) \tag{2.3}$$

We are now ready to discuss the loss-function that we are interested in.

Theorem 2.23 (Likelihood for right-censored data)**.** *Let T be a random waiting-time parametrized by θ. Under the assumption of Non-informative right censoring we can write the likelihood as: [8]:*

$$\mathfrak{L}(t, \theta) \propto \begin{cases} Pr(T = t|\theta) & \textit{if uncensored} \\ Pr(T > t|\theta) & \textit{if right censored} \end{cases}$$

Proof. Let $X = \min(T, C)$ with T the waiting time, C the random censoring variable and $\Delta = I(T \leq U)$ the failure indicator. The joint PDF for (X, Δ) can be written as:

$$f_{X,\Delta}(x,u) = \frac{d}{dx}F_{X,\Delta}(x,u) = \lim_{h\to 0}\frac{1}{h}\Pr(\{x \leq X \leq x+h\} \cap \{\Delta = u\})$$

Where we skip writing out that the pdfs contain θ to simplify notation. Since we have non-informative censoring the joint PDF of (T, C) is:

$$T \perp C \implies f_{T,C}(t,c) = f_T(t)f_C(c) \tag{2.4}$$

Consider the case when we have censored data so that $u = 0$:

$$\begin{aligned}
\frac{1}{h}\Pr(\{x \leq X \leq x+h\} \cap \{\Delta = 0\}) &= \\
\frac{1}{h}\Pr(\{x \leq \min(T,C) \leq x+h\} \cap \{T < C\}) &= \\
\frac{1}{h}\Pr(\{x \leq T \leq x+h\} \cap \{T < C\}) &= \\
\frac{1}{h}\int_x^{x+h}\int_t^\infty f_T(t)f_C(c)dcdt &= \\
\frac{1}{h}\int_x^{x+h} f_T(t)\int_t^\infty f_C(c)dcdt &= \\
\frac{1}{h}\int_x^{x+h} f_T(t)S_C(t)dt &\to f_T(x)S_C(x)
\end{aligned}$$

So $f_{X,\Delta}(x,0) = f_T(t)S_C(t)$.

Now consider the case when we have uncensored data so that $u = 1$:

$$\begin{aligned}
\frac{1}{h}\Pr(\{x \leq X \leq x+h\} \cap \{\Delta = 1\}) &= \\
\frac{1}{h}\Pr(\{x \leq \min(T,C) \leq x+h\} \cap \{C \leq T\}) &= \\
\frac{1}{h}\Pr(\{x \leq C \leq x+h\} \cap \{C \leq T\}) &= \\
\frac{1}{h}\int_x^{x+h}\int_c^\infty f_C(c)f_T(t)dtdc &= \\
\frac{1}{h}\int_x^{x+h} f_C(c)S_T(c)dc &\to f_C(x)S_T(x)
\end{aligned}$$

So $f_{X,\Delta}(x,1) = f_C(x)S_T(x)$.

This means that given $C \perp T|\theta$ we can factor the likelihood as:

$$f_{X,\Delta}(x,u) = (f_T(x)S_C(x))^u \cdot (f_C(x)S_T(x))^{1-u} = \left[f_T(x)^u S_T(x)^{1-u}\right] \cdot \left[f_C(x)^{1-u}S_C(x)^u\right] \tag{2.5}$$

And if furthermore $C \perp \theta$ we know that f_C and S_C are not functions of θ so the likelihood will have the form:

$$\mathfrak{L}(t, \theta) = f_{X,\Delta}(x, u) \propto f_T(x)^{1-u} S_T(x)^u \tag{2.6}$$

□

Corollary 2.24 (Likelihood for right-censored data, discrete case)**.** *Let T, C, X, Δ be as above but now T and C are discrete with support $\{0, 1, \ldots\}$. Then*

$$\mathfrak{L}(t, \theta) \propto \begin{cases} Pr(T = t|\theta) & \textit{if uncensored} \\ Pr(T > t|\theta) & \textit{if right censored} \end{cases}$$

Proof. We will show as above that the joint distribution of (X, Δ) neatly factors.

Censored case:

$$\begin{aligned} \Pr(\{\min(T, C) = x\} \cap \{C < T\}) = \\ \Pr(\{C = x\} \cap \{T \in \{x + 1, x + 2, \ldots\}\}) = \\ \Pr(C = x) \cdot \Pr(T > x) \end{aligned}$$

Uncensored case:

$$\begin{aligned} \Pr(\{\min(T, C) = x\} \cap \{T \leq C\}) = \\ \Pr(\{T = x\} \cap \{C \in \{x, x + 1, \ldots\}\}) = \\ \Pr(T = x) \cdot \Pr(C \geq x) \end{aligned}$$

Using the above we see that the likelihood factors as:

$$\Pr(X = x \cap \Delta = u) = \Big[\Pr(T = x)^u \cdot \Pr(T > x)^{1-u}\Big] \cdot \Big[\Pr(C = x)^{1-u} \cdot \Pr(C > x)^u\Big]$$

□

Note the risk of doing an off-by-one error. Even though it's similar to the continuous likelihood we have $Pr(T_d > t|\theta) = S_d(t + 1)$.

As a final remark, there's one censoring process which is particularly relevant for the problem we have. If the censoring time C is fixed in advance we will show that we only need the first condition for non-informative censoring to get the sought likelihood function. One can argue that if we have recorded events for a week and want to train our model today then from a week ago the censoring time would be $C = 7$ and so $X = \min(T, 7)$.

Corollary 2.25 (Likelihood when censoring time C is fixed and $T \perp C|\theta$)**.** *Let $X = min(T, C)$, T a waiting time, C a censoring variable and $\Delta = I(T \leq C)$ a failure indicator as in theorem 2.23 but assume that $C = c$ is known. Assume further that we only have $T \perp C|\theta$, not necessarily $C \perp \theta$ Then the joint distribution of $(X, \Delta)|(C, \theta)$ is*

$$\Pr(\{X \leq t\} \cap \{\Delta = u\} | \{C = c\} \cap \{\theta\}) = \begin{cases} \Pr(T \leq min(t,c)|\theta) & u = 1 \\ \Pr(T > c|\theta) & u = 0,\, t \leq c \\ 0 & u = 0,\, t > c \end{cases}$$

Meaning that the log-likelihood will factor as in theorem 2.23.

Proof. We show this by expanding the events explicitly and use basic identities for conditional probability to show that the proposed right hand side follows.

In the first case, for uncensored data when $\Delta = u = 1$:

$$\begin{aligned} \Pr(\{X \leq t\} \cap \{\Delta = 1\} | \{C = c\} \cap \theta) = \\ \Pr(\{min(T,C) \leq t\} \cap \{T \leq C\} | \{C = c\} \cap \theta) = \\ \Pr(\{T \leq t\} \cap \{T \leq c\} | \{C = c\} \cap \theta) = \\ \Pr(T \leq \min(t,c) | \{C = c\} \cap \theta) \overset{*}{=} \\ \Pr(T \leq \min(t,c)|\theta) \end{aligned}$$

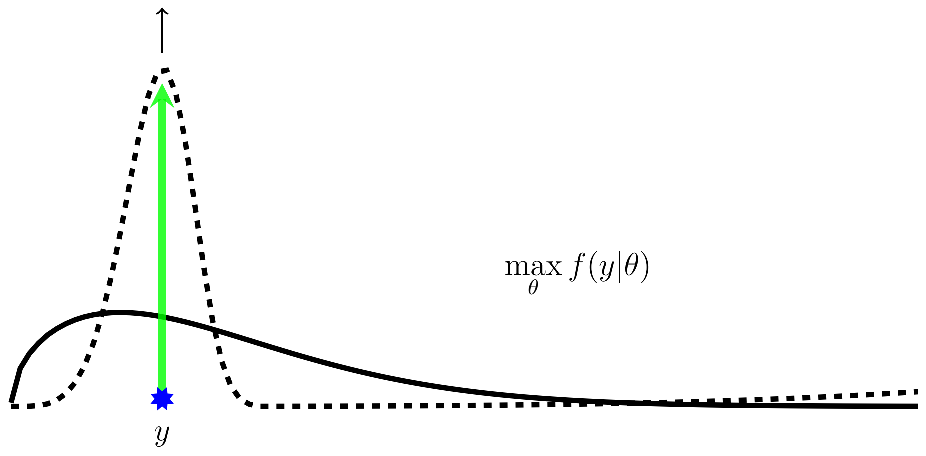
And for uncensored data when $\Delta = u = 0$:

$$\begin{aligned} \Pr(\{X \leq t\} \cap \{\Delta = 0\} | \{C = c\} \cap \theta) = \\ \Pr(\{min(T,C) \leq t\} \cap \{T > C\} | \{C = c\} \cap \theta) = \\ \Pr(\{C \leq t\} \cap \{T > c\} | \{C = c\} \cap \theta) \overset{*}{=} \\ \begin{cases} \Pr(T > c|\theta) & u = 0,\, t \leq c \\ 0 & u = 0,\, t > c \end{cases} \end{aligned}$$

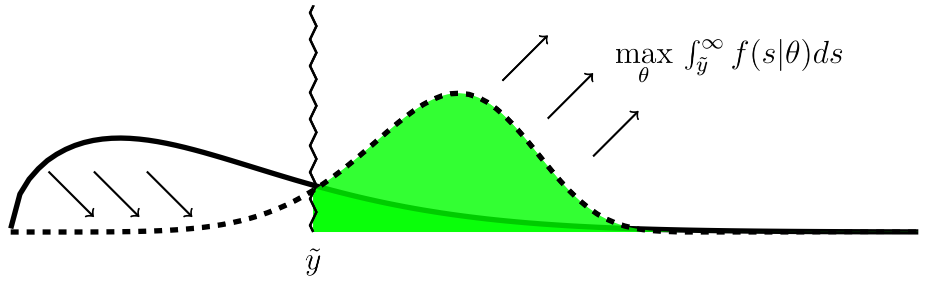
Where '$\overset{*}{=}$' follows from $\{T \perp C\}|\theta \iff \Pr(T|C \cap \theta) = \Pr(T|\theta)$. □

We could have simply stated thm. 2.23, cor. 2.24 and 2.25 without proof as the results are well known. As we discovered, this fact seems to have led to some kind of tragedy of the commons leading most treatments of the subject to simply disregard to state the detailed assumptions and proofs. We need to understand these details to be able to safely make assumptions and understand the biases that may be introduced in the model if they are violated. Since we couldn't find a succinct treatment we wrote it down here.

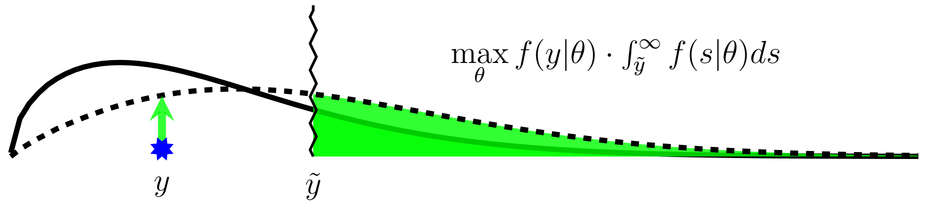
Even though the probabilistic justification of the likelihood is quite abstract, we could have constructed it using intuition alone by designing some loss-function that rewards correct point estimates (uncensored points) and penalizes incorrect region estimates (censored points i.e pointing out unvisited regions). We illustrate this with figure 2.3:



(a) Uncensored observation: Push the pdf up at event



(b) Censored observation: Push mass over the point of censoring



$\max_\theta f(y|\theta) \cdot \int_{\tilde{y}}^{\infty} f(s|\theta)ds$
y
$\tilde{y}$

(c) Uncensored and censored observation: Compromise

Figure 2.3: How to reach optimal fit with data that is uncensored (a), censored (b) or both (c). Result shown here are actual fits with f the Weibull pdf.

So to maximize terms from censored observations we push the density function over to the right of the point of censoring. To maximize terms from uncensored observations we push the density function upwards around those points. As pdfs/cmfs always integrates to 1 these are competing interests for the likelihood function and so the optimal θ (if it exists) will be a compromise.

The form of the likelihood presented above might be easier for interpretation but there's other formulations that's much better for calculation.

Proposition 2.26 (Likelihood for right-censored data, alternative form)**.** *Let* (t, u) *be an observation with* u *the failure indicator s.t* $u = 1$ *means that we have an uncensored observation and* $u = 0$ *a right censored observation. Under the assumption of non-informative censoring we may write:*

$$\log(\mathfrak{L}) = u \cdot \log(\lambda(t)) - \Lambda(t)$$

in the continuous case and

$$\begin{aligned}\log\left(\mathfrak{L}_d\right) =& u\cdot\log(p(t)) - (1-u)\Lambda(t+1)\\ =& u\cdot\log\left(e^{d(t)}-1\right) - \Lambda(t+1)\end{aligned}$$

in the discretized case.

Proof. **Continuous case:**

$$\begin{aligned}\mathfrak{L} =& f(t)^u\cdot\Pr\left(T>t\right)^{1-u}\\ =& f(t)^u\cdot S(t)^{1-u}\\ =& e^{-u\cdot\Lambda(t)}\cdot\lambda(t)^u\cdot e^{-(1-u)\cdot\Lambda(t)}\\ =& \lambda(t)^u\cdot e^{-(u+1-u)\cdot\Lambda(t)}\\ =& \lambda(t)^u\cdot S(t)\\ \iff&\\ \log\left(\mathfrak{L}\right) =& u\cdot\log\left(\lambda(t)\right) - \Lambda(t)\end{aligned}$$

Discretized case:

Let T_d be the discretization of random variable T s.t $T_d = t \iff T\in[t,t+1)$ with t integer:

$$\mathfrak{L}_d = \Pr(T_d=t)^u\cdot\Pr\left(T_d>t\right)^{1-u} \tag{2.7}$$
$$= \Pr(T_d=t)^u\cdot S_d(t+1)^{1-u} \tag{2.8}$$
$$= \Pr\left(T\in[t,t+1)\right)^u\cdot\Pr\left(T\geq t+1\right)^{1-u} \tag{2.9}$$
$$= \left(S(t)-S(t+1)\right)^u\cdot S(t+1)^{1-u} \tag{2.10}$$
$$= \left(e^{-\Lambda(t)}-e^{-\Lambda(t+1)}\right)^u\cdot e^{-(1-u)\cdot\Lambda(t+1)} \tag{2.11}$$
$$= e^{-u\cdot\Lambda(t+1)}\cdot\left(e^{(\Lambda(t+1)-\Lambda(t))}-1\right)^u\cdot e^{(u-1)\cdot\Lambda(t+1)} \tag{2.12}$$
$$= \left(e^{d(t)}-1\right)^u\cdot e^{-\Lambda(t+1)} \tag{2.13}$$
$$\iff \tag{2.14}$$
$$\log\left(\mathfrak{L}_d\right) = u\cdot\log\left(e^{d(t)}-1\right) - \Lambda(t+1) \tag{2.15}$$

Where we used the definition $d(t) = \Lambda(t+1) - \Lambda(t)$. □

Some of the intuition about optimization with censored data is more explicitly shown in these equation. We see that for discrete and continuous, censored or not, the log-likelihood will always have a negative term $-\Lambda$ strictly decreasing with t that punishes us whenever $t\nearrow$.

2.4 Previous work: Models for censored data

This section surveys the most notable models for censored data. We have identified a few central approaches to tackling the problem. These involve workarounds like reformulating it as a binary (or categorical) prediction problem, seeing it as a 'learning to rank'-problem and survival approaches. We will start with the cumbersome - but very popular - workaround [9–14].

2.4.1 Sliding box model : Binary prediction workaround

The idea of this model is to hard code the inference questions into binary states for each timestep. We do this by predicting whether an event will happen within a preset timeframe. This means that any classification algorithm can be used at the expense of having to create (arguably mind-bending) predefined definitions of the target-values.

Formally we describe the setup for this problem as trying to estimate the parameter θ_t in a bernoulli-distribution. We will call this model the 'Sliding Box Model':

Definition 2.27 (Sliding Box Model)**.**

$$\begin{aligned} B_t &\sim \text{Bernoulli}(\theta_t) \\ b_t &= \begin{cases} 1 & \text{If event happened in } [t, t+\tau) \\ 0 & \text{Else} \end{cases} \\ \Pr(B_t = b_t) &= \theta_t^{b_t} \cdot (1-\theta_t)^{1-b_t} \end{aligned}$$

Here τ is some preset treshold parameter and θ_t is the probability of event within τ time from timestep t. We set θ_t as a function of data available at that time $\theta_t = g(x_{0:t})$, i.e. estimate the probability of an event happening in our sliding box of width τ.

$$\underset{\theta_t}{\text{maximize}}\, \mathfrak{L}(\theta_t) := \theta_t^{b_t} \cdot (1-\theta_t)^{1-b_t}$$

An example is to predict each day whether an event happens in the coming month ($\tau = 30$).

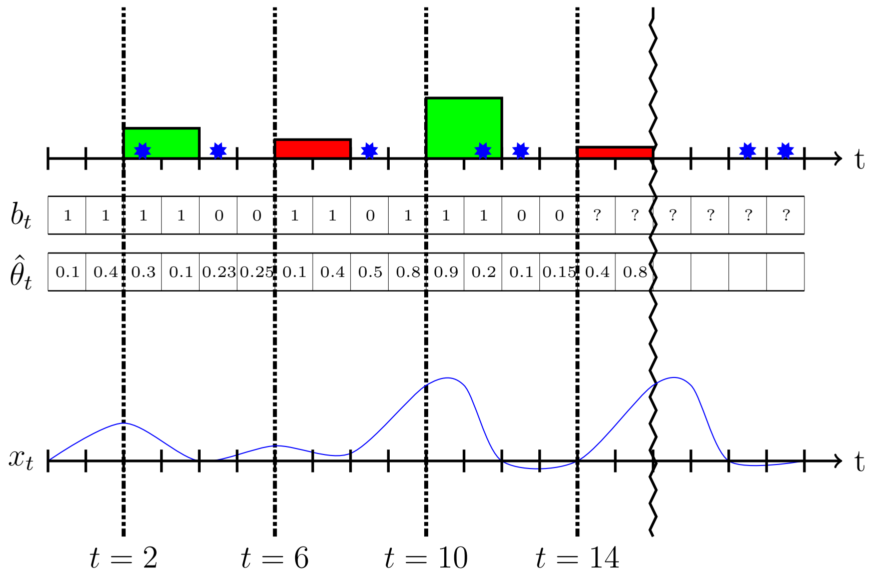


Figure 2.4: Sliding Box model. Estimate the probability θ_t using features x_t that there'll be an event (blue stars) within τ time from current position (vertical lines). Here we illustrate how target values b_t are calculated by sliding the box of width $\tau = 2$ over the timeline and setting $b_t = 1$ if any future event is covered by the box. For prediction we estimate $\hat{\Pr}(B_t = 1) = \hat{\theta}_t = g(x_t)$, here shown as box-height, as a function g from features x_t. The last τ steps of recorded data can be predicted but can't be trained on as we can't conclude target values. See also figure 1.11.23.1

The most obvious strength with the sliding-box model is its simplicity. This simplicity might hide very tough modeling decisions such as choosing a τ to make sense. By tempering with τ we change both the meaning, usage and training of the model which can be a slow, iterative and biasing process in a modeling situation.

Another limitation is loss of τ timesteps of training data at the end of the sequence. As illustrated in figure 2.4 we can't use the last steps to learn the distribution $\Pr(V_t|\theta_t)$ as we can't exclude that an event *won't* happen just beyond the boundary. To predict further ahead in time (raising τ) we get less data to train on which is worrisome as we expect that we need *more* training data to predict further into the future.

The binary states can be defined in obvious (*dead/alive in next timestep at t*), or arbitrary ways (*Customer will make a purchase within τ days at timestep t*). Here inference is limited to predicting one timestep ahead. Forward-looking formulations (*Customer will stay with company*) [9] are particularily pervasive in churn-prediction (see section 3.3). It's not rare that there's an underlying arbitrary definition as the one mentioned lurking behind such formulations.

Models for more fine-grained prediction have been built using the same framework but turning it into a multi-category classification problem. This can be done by letting target vector $\vec{v}$ represent J fixed intervals controlled by $\tau_0 = 0 < \tau_1 < \tau_2 \ldots, \tau_J$ i.e $v_j = 1 \iff T \in [\tau_j, \tau_{j+1})$, see [15, 16].

In churn-modeling there's been recent interest in using RNNs together with the sliding box, see the popular blog post by [17] and the paper by [18]. In section 4.4 we compare this to the proposed model.

2.4.2 Making it a 'learning to rank'-problem

Often we only need to be able to order the predicted waiting times among subjects. Examples are ordering patients according to their predicted time left to live, predict in which order machines are expected to fail or to rank customers according to when they are expected to return. In statistics this type of data is called data measured on an ordinal scale.

Consider ordering a set of observations t_1, t_2, and t_3. Note that the the order of a sequence can be reduced to a set of pairwise comparisons:

$$D_{ij} = I(t_i \leq t_j)$$

Resulting in an upper triangular matrix if the t_i's are indexed according to their ordering:

$$\begin{matrix} t_1 = 1.2 \\ t_2 = 1.9 \\ t_3 = 5.4 \end{matrix} \implies D = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

We can call this matrix D the *actual ordering*.

The machine-learning task - *learning to rank* - is thus to chose a function (model) f that combines features to predict/estimate the true ordering, $\hat{D}_{ij} = f(x_i, x_j)$.

It's easy to show that the sum $\sum_{i \neq j} |D_{ij} - \hat{D}_{ij}|$ is proportional to the Wilcoxon- or Mann-Whitney-U statistic and after normalization equivalent to the Area Under the Curve or Gini-index for pairwise comparison of scores and true class [19]. This problem can be relaxed with the appropriate model and loss function to transform it into a differentiable binary classification problem. Maximizing the sum is then optimization for the Area Under the Curve.

Imagine now that instead of observing t_2 we only observed its censoring time $x_2 = 1.5 = \min(t_2, c_2)$, so we know that $1.5 \leq t_2$. This means that we only have *partially ordered* observations. The *observed ordering* is therefore:

$$\begin{matrix} t_1 = 1.2 \\ t_2 \leq 1.5 \\ t_3 = 5.4 \end{matrix} \implies \tilde{D} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & ? \\ 0 & ? & 1 \end{pmatrix}$$

Where we know that $\tilde{D}_{2,1} = 1$ and $\tilde{D}_{1,2} = 0$ since $t_1 \leq 1.5 \leq t_2$ but we cannot infer the ordering of t_2 and t_3 by only knowing the censoring time for t_2. The machine-learning task is then the same as above but during training we disregard- or center the '?'-elements in the loss-function.

When we have partially ordered sets, the mean correctness of prediction of the known elements of $\tilde{D}$ is called the *Concordance Index* or *Area Under The Survival Curve*. Building models optimizing for this and variants of it is a new and busy field. There's notable ongoing research on using neural networks [20] and gradient boosting [21, 22] for applications to waiting-time prediction.

The fact that we are explicitly predicting an ordering give us some robustness but the computational cost is high as the number of pairs grows quadratically with the number of observations. If we have many sequences each consisting of many timesteps this quickly gets out of hand.

2.4.3 Survival-approaches

Survival Analysis can be seen as the statistical field dealing with random waiting times and censored observations. As the name reveals, the main focus is typically *analysis* so predictive black-box-models has long been standing down against classical frequentist lightweight approaches. The overwhelmingly most popular models for prediction- and analysis of covariate effect on outcome are the Accelerated Failure Time-model (AFT) and the Proportional Hazards- model (PH) [3, 4]. In both cases we fix some baseline hazard-function $\lambda_0(t)$ and some function of feature data $g(x)$.

Definition 2.28 (Cox Proportional Hazards model)**.**

$$\begin{aligned}\lambda(t,x) &= e^{g(x)} \cdot \lambda_0\left(t\right)\\ \log(\mathfrak{L}) &= u \cdot [g(x) + \log(\lambda_0(t))] - e^{g(x)} \cdot \Lambda_0\left(t\right)\\ &\underset{g}{\propto} u \cdot g(x) - e^{g(x)} \cdot \Lambda_0\left(t\right)\end{aligned}$$

So the PH-model uses features to vertically stretch the hazard function.

Definition 2.29 (Accelerated Failure Time model)**.**

$$\begin{aligned}\lambda(t,x) &= e^{g(x)} \cdot \lambda_0\left(t \cdot e^{g(x)}\right)\\ \log(\mathfrak{L}) &= u \cdot \left[g(x) + \log(\lambda_0(t \cdot e^{g(x)}))\right] - \Lambda_0\left(t \cdot e^{g(x)}\right)\end{aligned}$$

And the AFT uses features to stretch the hazard function vertically and horizontally.

Typically λ_0 is the exponential hazard function i.e a constant and the link function $g(x) = w^T x$ where the latter makes it a linear-regression problem. One can show that the only λ_0 making the AFT and PH coincide are the Weibull baseline hazard with the exponential as a special case [4].

It's arguably even more popular to discretize time so the problem becomes estimating the probability of event in each such interval. The loss in the PH-case is then called *partial likelihood.* For details regarding this formulation see [4] but we'll give a a brief description. If y_i is the waiting time for observation i and u_i indicates if this is uncensored, the loss for the i'th observation becomes:

$$\log(\mathfrak{L}_i) = (1-u) \cdot \log\left[\frac{e^{g(x_i)}}{\sum_{j:y_i \leq y_j} e^{g(x_j)}}\right]$$

Where the baseline probability of event in the interval $[\lfloor y_i \rfloor, \lfloor y_i \rfloor + 1)$ is canceled out because it appears in both the numerator and the denominator. This is the models main feature making it ideal for comparing risks among subjects.

So far so good but both these models aren't very expressive, i.e in statistical jargon they rely on strict assumptions on the data and can't capture many nonlinear-relationships or interaction effects. This is particularly true when using linear link-function g (regression) which is overwhelmingly the most popular approach [4, 23]. But there are examples of using other g's.

PH-models have been extended with ANNs [23–25] and SVM's [26]. AFTs have been ensembled through gradient boosting [27]. For these models they assumed random censoring and optimized the partial likelihood. Other related approaches that's been proposed been to minimizing log-likelihood weighted by estimations of the inverse censoring-probability, most famously implemented for *survival forests* [8]. See [28] for a discussion about inverse censoring-probability weighting (IPCW).

To use time-varying covariates or recurrent events one typically uses the same (non-temporal) models with modified input data. For each subject, each observation of the time-to-events y_t, time-varying features x_t and censoring-indicators u_t at time t are treated as a single observation pair (y_t, x_t, u_t) [4].

2.4.3.1 Threshold regression

First Hitting Time Model is a collection of methods for time-to-event and survival modeling. Here events are seen as the result of some stochastic process $\{X(t)\}$. The time to those events from a previous point in time are referred to as a *first hitting time* (FHT) from the idea of an event as occurring when an observable- or non-observable (latent) stochastic process reaches a threshold/boundary. This random process could for example be the gasoline-level in a car resulting in a stop when it reaches zero. This perspective is called *first passage time models*. In other cases the hazard function itself is seen as the random process [29] reminiscent of the example in preliminaries, section 2.2.1.

The process determining events are called the *parent process*. Typically this process is assumed to be equipped with Markov-properties [1, 30–32]. To incorporate time-varying covariates we let the parent process vary with some observable *marker process* $\{Y(t)\}$ that gives us information about the progress of the parent process [30].

In summary this perspective seems most popular in the life-sciences. Research here is often limited by small datasets, lots of censoring and high dimensional feature data warranting careful understanding of the underlying assumed process. We have not found any general framework that can easily be extended with arbitrary distributions for the waiting time.

2.4.3.2 Tobit Kalman Filter

A problem-setting which closely resembles ours can be found in the context of Prognostics & Health Monitoring (PHM). The focus is predicting what they call RUL (Remaining Useful Life) [33] of machines, synonymous to a failure time. Usually this is predicted over a timeline using time series of sensor data i.e time-varying covariates. Data often stems from run-to-failure experiments so censoring is not a central topic of discussion. In section 4.2 we apply the proposed model to a RUL-dataset stemming from a machine learning competition (CMAPSS-dataset [34]). Point-estimating Recurrent Neural Networks with Kalman-filter preprocessing won the original competition, see [34, 35]. As they where point-estimating with uncensored data these RNN-implementations are regular regression-RNNs and hence have little overlap with our method. This is the reason why we chose not to do any deeper comparisons with our model.

As expected in a classical engineering field, Kalman Filters and their variations are the most common approaches. Kalman filters have been extended to accommodate censored data, and research on this topic seems to have come in fashion quite recently [36–40]. These models are known as *Tobit Kalman-filters*.

Tobit-models are part of the standard toolbox of economic research, in particular Tobit-Regression. The model is nothing but a Normal distribution with some added link-function to let the mean-parameter μ be a function of features. The weights are found by optimizing over the censored log-likelihood discussed in section 2.3. *partial regression* [16] is a closely related framework. Tobit Kalman filters are Kalman Filters where the latent variable is sometimes only partially observed (censored) [40].

We did not go deeper into either Tobit-Regression in general or the the Tobit Kalman-filters in particular. First of all this thesis is not about comparing RNNs to Kalman filters which is a wider research question. Second, standard Normal Kalman filters are not appropriate for waiting time data. Normal distribution assumes values on the whole real line while Waiting times are positive so the normal assumption is erroneous. As we will see in chapter 3 the proposed framework can easily be extended to use a Normal distribution making it a sort of Tobit model. To train with discretized or censored data we need to compute its CDF and as the Normal distribution has no closed form CDF it gets computationally cumbersome.

Apart from the Tobit Kalman Filter we have not found any inherently temporal application or adaptation of any of the models discussed. All models using time-varying covariates to predict time-to-event have used fixed-width feature vectors with modified input data. The timeline is split into intervals (such as months, days), features aggreated over them and treated like single observations. Temporal information are explicitly stored (typically from window-functions like cumsum, cummean, cummax, lag, change) which needs to be hand-engineered. The models themselves are memoryless.

2.4.4 Other

There are approaches that are harder to categorize such as modeling the process afterwards. This is relevant in the context of stochastic processes such as change-point detection [41]. Other notable methods is to see it as a density-estimation problem i.e estimation of the hazard function. There are numerous examples of wavelets being used for this purpose such as [42] who posteriori- fitted wavelets to recurrent event-data. We would argue that explicit modeling of a process after its done using observations from its outcome is a very different problem than predicting its future states using feature-data.

2.5 The Weibull distribution

Distributions appropriate for waiting times as we've defined them are positive and has infinite support. This covers a very large class of distributions. The most commonly used distributions in survival-analysis and reliability engineering are the *exponential-*, *Log-logistic-*, *Exponential-logarithmic- gamma-*,*Rayleigh*, *Erlang* and the *Weibull* distribution in the continuous case and the *poisson-*, *geometric* and the *discrete weibull* in the discrete case [4, 43, 44].

In this thesis we will use the Weibull- distribution and its discrete variant. This distribution has some some great properties which motivates the choice:

- Numerically stable closed form CDF and Quantile function
- Easily discretized
- Unimodal but expressive
- Empirically feasible
- Location-Scale transformation
- Regularization mechanisms

Here we will show these properties one by one. Why each of them matters to our model will become clear in the next chapter (3).

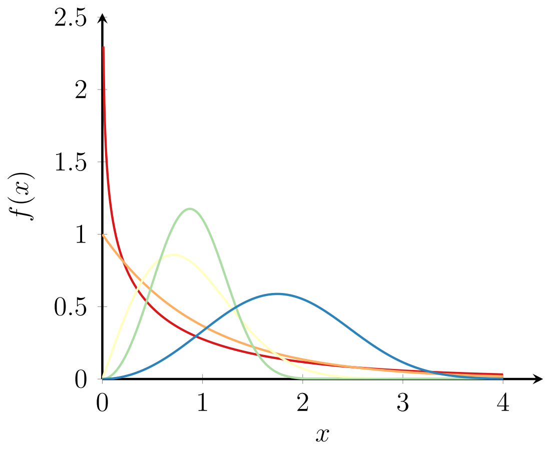
The continuous Weibull distribution is usually parametrized as:

Definition 2.30 (PDF, CDF, CHF and HF of the Weibull distribution)**.**

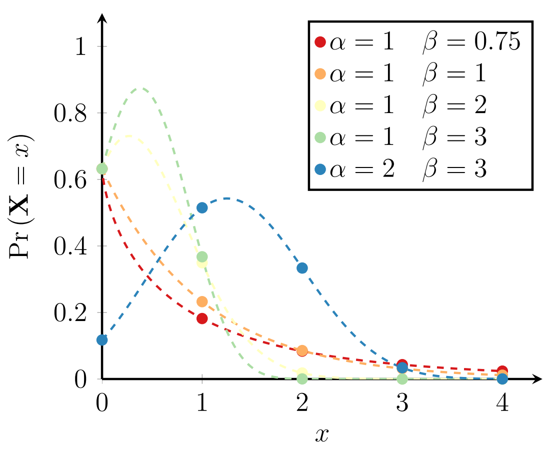
$$\begin{aligned} F(x) &= 1 - \exp\left[-\left(\frac{t}{\alpha}\right)^{\beta}\right] \\ f(x) &= \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1} \exp\left[-\left(\frac{t}{\alpha}\right)^{\beta}\right] \\ \Lambda(x) &= \left(\frac{t}{\alpha}\right)^{\beta} \\ \lambda(x) &= \left(\frac{t}{\alpha}\right)^{\beta-1} \cdot \frac{\beta}{\alpha} \end{aligned}$$

Where $t \in [0, \infty)$, scale paramater $\alpha \in (0, \infty)$, shape parameter $\beta \in (0, \infty)$. We say that a random variable $T \sim \text{Weibull}(\alpha, \beta)$

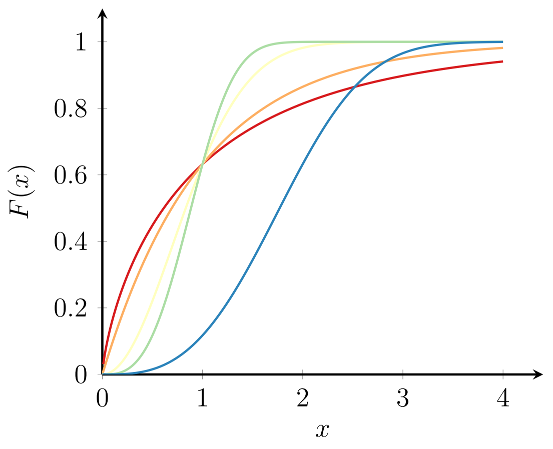
There's many other parametrizations of the Weibull functions. Among notable variations is to use $a = \alpha^{-\beta}$ s.t $\Lambda(t) = at^{\beta}$.



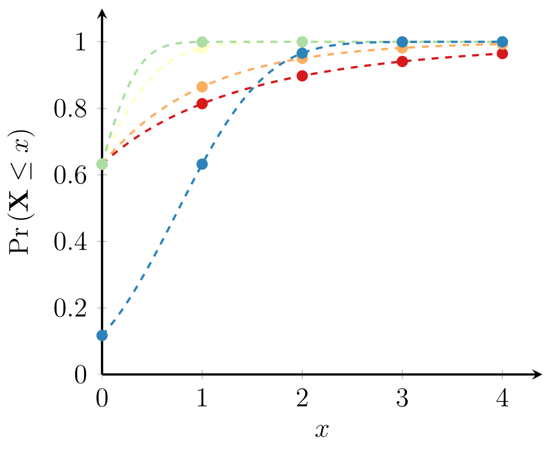
(a) PDF



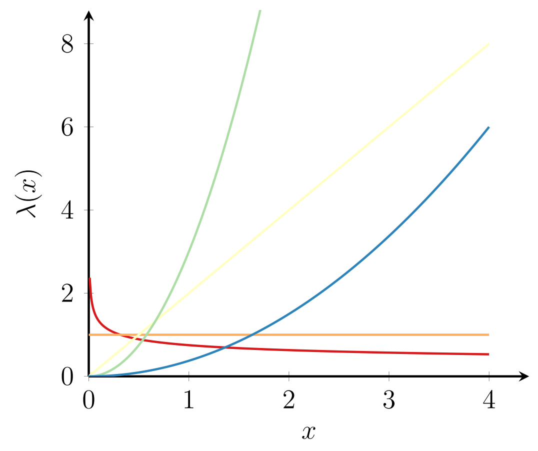
(b) PMF



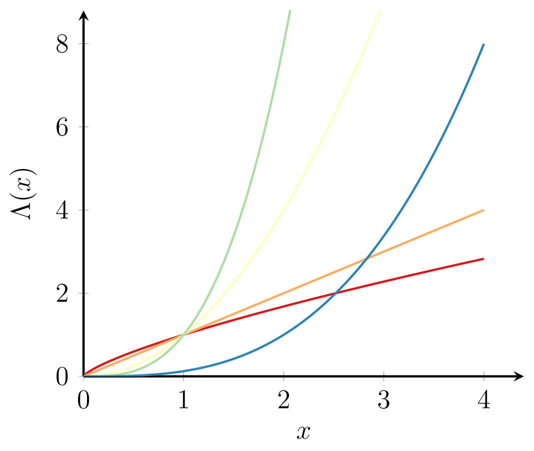
(c) CDF



(d) CMF



(e) Hazard Function



(f) Cumulative Hazard Function

Figure 2.5: Distribution functions for the continuous and discrete Weibull

2.5.1 Properties

The Weibull distribution is *unimodal* meaning it has at most one peak. Apart from that it's extremely expressive. When $\beta \leq 1$ the PDF is strictly decreasing, when $\beta = 1$ the hazard is constant. When $\beta = 1$ we have the exponential in the continuous case and the geometric distribution in the discrete case. If $Z_k \sim N(0, \sigma)$, $k = 1, 2$ one can show that $|Z| \sim \text{Weibull}(\sqrt{2}\sigma, 2)$ also called the Rayleigh-distribution [45]. As β gets larger more and more mass is concentrated as a peak around $t = \alpha$. In the limit $\beta \to \infty$ it converges to a *degenerate distribution*, i.e. a deterministic distribution with pdf being the Dirac delta-function.

mean :	$\alpha\Gamma(1 + 1/\beta)$
median :	$\alpha(\ln(2))^{1/\beta}$
mode :	$\begin{cases} \alpha\left(\frac{\beta-1}{\beta}\right)^{\frac{1}{\beta}} & \beta > 1 \\ 0 & \beta \leq 1 \end{cases}$
quantile function $F^{-1}(p)$:	$\alpha \cdot (-\log(1-p))^{\frac{1}{\beta}}$
variance :	$\alpha^2\left[\Gamma\left(1+\frac{2}{\beta}\right) - \left(\Gamma\left(1+\frac{1}{\beta}\right)\right)^2\right]$
characteristic function :	$\sum_{n=0}^{\infty} \frac{(it)^n \alpha^n}{n!}\Gamma(1 + n/\beta)$

Table 2.1: Basic quantitities of the Weibull-Distribution

To generate Weibull-data, let $U \sim$ Uniform and use the quantile function $F^{-1}(U) = \alpha \cdot (-\log(U))^{\frac{1}{\beta}} \sim$ Weibull. A rule of thumb is that $t = \alpha$ is always approximately the 0.632th quantile, which stems from the fact that $F(\alpha) = 1 - e^{-1} \approx 0.632$ [45].

In order to get an understanding of what the parameters does its useful to consider them after doing a *location-scale*-transformation [45].

Proposition 2.31. *(Location-Scale transformation)*
If $T \sim$ Weibull(α, β) then $\log(T) \sim$ *Gumbel*$(\log(\alpha), \frac{1}{\beta})$

Proof. If $Y \sim \text{Gumbel}(\mu, \sigma)$ it has CDF $1 - \exp\left[-\exp\left[\frac{y-\mu}{\sigma}\right]\right]$ where $\mu \in \mathbb{R}$ and $\sigma > 0$ are location- and scale- parameters respectively.

$$\begin{aligned}
\Pr(\log(X) \leq y) =& \Pr(X \leq e^y) = 1 - \exp\left[-\left(\frac{e^y}{\alpha}\right)^\beta\right] \\
=& 1 - \exp\left[-\exp\left[\beta\left(y - \log\alpha\right)\right]\right] \\
=& 1 - \exp\left[-\exp\left[\frac{y - \log\alpha}{\frac{1}{\beta}}\right]\right] \\
=& 1 - \exp\left[-\exp\left[\frac{y-\mu}{\sigma}\right]\right]
\end{aligned}$$

□

Here the log of the Weibull α is a location parameter and $1/\beta$ a scale-parameter. This type of parametrization should make sense to someone confident with the

Normal distribution. When training Weibull-regression models the goal is usually to estimate $\mu = \log(\alpha)$ as a linear combination of features [45].

Another property is the *Minimum closure property*:

Proposition 2.32. *(Minimum Closure-property)*

If $T_k \sim Weibull(\alpha_k, \beta)$ and $T_k \perp T_j$ then $min(T_1, \ldots, T_n)$ is again Weibull with $\alpha^* = (\sum_{i=1}^n {\alpha_i}^{-\beta})^{-\frac{1}{\beta}}$. [45]

Proof.

$$Pr(min(T_1 \ldots, T_n) > t) = \prod_k Pr(T_k > t) = \exp\left[-t^\beta \sum_k {\alpha_k}^{-\beta}\right] = \exp\left[-\left(\frac{t}{\alpha^*}\right)^\beta\right]$$

□

So the minimum of independent Weibulls with the same β are again Weibull. This is a property which brings the thoughts to the closedness of the Normal distribution under summation [45]. Considering how expressive the Weibull is, this is a pretty powerful property. If we have a system that breaks whenever one of its components breaks, each of them having independent failure times as above, then the failure of the whole system is Weibull. This in itself should be a hint to why the Weibull often fits well. But there's an even stronger theorem called the *Fisher-Tippett-Gnedenko theorem*, or the *extreme value theorem*. The theorem states that the limiting distribution of a sequence that shifts and scales the min or max over an increasing set of IID random variables only has three different possible forms where one is the Weibull-distribution [46].

We have adapted the original proof of by Fisher [46] to the special case of the minimum of positive random variables. We will show that if the distribution of a minimum of n positive IID random variables scaled by a sequence a_n converges to a non-degenerate distribution as $n \to \infty$ then this limit must be the Weibull distribution. The name comes from the implication that if a system breaks with the failure of the weakest of its identical components the failure time is approximately Weibull distributed [45].

Theorem 2.33 (Weakest Link Property)**.** *Let $T_n = \min\{X_1, \ldots, X_n\}$ where X_k are positive IID random variables. If $Pr\left(\frac{T_n}{a_n} \leq t\right) \to Pr\left(T \leq t\right)$ where T is not degenerate $\implies T \sim$ Weibull.*

Proof. First note that if

$$\Pr\left(\frac{T_n}{a_n} \leq t\right) = \Pr\left(X_1 > a_1 \cdot t\right) \cdot \ldots \cdot \Pr\left(X_n > a_n \cdot t\right) = S_X(a_n t)^n \to S(t)$$

Then so is the case for T_{nm} as $n \to \infty$. By then letting $m \to \infty$ we can see it as taking the minimum of the minimums of m sets of n random variables. As the limiting distribution in each subgroup should be the same as that of the limit over those groups we expect the following functional equation of the limiting survival function S to hold:

$$S(a_n t)^n = S(t) \tag{2.16}$$

By evaluating equation 2.16 at $t = \frac{x}{a_n}$ we observe that $S(x)^n = S(\frac{x}{a_n})$ and since a_n is arbitrary so far we may write:

$$S(t)^n = S(a_n t)$$

By evaluating the above at $a_m t$ we get

$$S(a_m t)^n = S(t)^{mn} = S(a_{mn} t) = S(a_n \cdot a_m t)$$

So $a_{mn} = a_n \cdot a_m$. This leads us to a differential equation for a_n:

$$a(xy) = a(x)a(y) \implies \begin{cases} ya'(xy) = a(x)'a(y) \\ xa'(xy) = a(x)a(y)' \end{cases}$$

$$\implies \frac{a'(x)a(y)}{y} = \frac{a(x)a(y)'}{x} \iff x\frac{a'(x)}{a(x)} = y\frac{a(y)'}{a'(y)}$$

$$\implies \frac{a'(x)}{a'(x)} = -\frac{1}{\beta x} \iff \log(a(x)) = -\frac{1}{\beta}\log(x) + d \iff a(x) = d \cdot x^{-\frac{1}{\beta}}$$

Where $S(t)^1 = S(a_1 t) \implies a(1) = d = 1$. So

$$S^n(n^{-\frac{1}{\beta}}t) = S(t) \implies n\Lambda(n^{-\frac{1}{\beta}}t) = \Lambda(t)$$

Now try $\Lambda(t) = t^\beta$:

$$n\Lambda(n^{-\frac{1}{\beta}}t) = n(n^{-\frac{1}{\beta}}t)^\beta = t^\beta = \Lambda(t)$$

□

The weakest link- and minimum closure property gives us a hint that whenever the life of a system is dependent on the life of all its independent components, we expect the Weibull to be a good model for it's failure time. The Weibull has been found useful in modeling structural failures, maximal annual rainfall, earthquake return times [47] strength of textile fibers [46] and more.

As we've shown (2.1.2), we can easily transform a continuous distribution to a discrete one. Let T and T_d be the continuous and discrete Weibull respectively:

Definition 2.34 (CMF, PMF of the Discrete Weibull distribution)**.**

$$Pr(T_d \leq t) = Pr(T \leq t+1) = 1 - \exp\left[-\left(\frac{t+1}{\alpha}\right)^\beta\right]$$

$$Pr(T_d = t) = Pr(t \leq T \leq t+1) = \exp\left[-\left(\frac{t}{\alpha}\right)^\beta\right] - \exp\left[-\left(\frac{t+1}{\alpha}\right)^\beta\right]$$

This parametrization differs from the most popular one. We chose this for computational reasons and to make the switch between discrete- and continuous seamless. The regular parametrization is to set the CMF to $1 - q^{(t+1)^\beta}$ with $q \in (0,1)$ [48].

2.5.2 Log-Likelihood

Before we dig into the likelihood properties it's useful to summarize some of the hazard-related quantities for the Weibull. For the continuous Weibull we have

$$\begin{aligned}\Lambda(t) &= \left(\frac{t}{\alpha}\right)^{\beta} \\ \lambda(t) &= \left(\frac{t}{\alpha}\right)^{\beta-1}\frac{\beta}{\alpha}\end{aligned}$$

To calculate the loss with censored observations we need to evaluate the log of the survival function. In the case of the Weibull distribution this is easily done. Using the alternative form of the likelihood from proposition 2.26 might make us overlook this feature. In the continuous case we write the log-likelihood as:

$$\begin{aligned}\log(\mathfrak{L}) &= \log\left[f(t)^{u}\cdot S(t)^{1-u}\right] \\ &= u\cdot\log[\lambda(t)] - \Lambda(t) \\ &= u\cdot\log\left[t^{\beta-1}\alpha^{-\beta+1}\alpha^{-1}\beta\right] - \left(\frac{t}{\alpha}\right)^{\beta} \\ &= u\cdot\log\left[t^{\beta-1}\alpha^{-\beta}\cdot\beta\right] - \left(\frac{t}{\alpha}\right)^{\beta} \\ &= u\cdot[(\beta-1)\cdot\log(t) - \beta\cdot\log(\alpha) + \log(\beta)] - \left(\frac{t}{\alpha}\right)^{\beta} \\ &\propto u\cdot[\beta\cdot[\log(t)-\log(\alpha)] + \log(\beta)] - \left(\frac{t}{\alpha}\right)^{\beta} \\ &= u\cdot\left[\beta\cdot\log(\frac{t}{\alpha}) + \log(\beta)\right] - \left(\frac{t}{\alpha}\right)^{\beta}\end{aligned}$$

Where in the last step we dropped the $-u\cdot\log(t)$-term since it's not a function of α or β so it doesn't influence the maximization-procedure. Here $\propto$ should be read 'proportional to'.

The discrete log-likelihood is not as nice, but it's closed form which is not the case for many discrete distributions.

$$\begin{aligned}\log(\mathfrak{L}_d) &= u\cdot\log\left(e^{d(t)}-1\right) - \Lambda(t+1) \\ &= u\cdot\log\left[\exp\left(\alpha^{-\beta}\cdot\left((t+1)^{\beta}-t^{\beta}\right)\right)-1\right] - \alpha^{-\beta}(t+1)^{\beta}\end{aligned}$$

Where we used

$$\begin{aligned}d(t) &= \left(\frac{t+1}{\alpha}\right)^{\beta} - \left(\frac{t}{\alpha}\right)^{\beta} \\ &= \alpha^{-\beta}\cdot\left((t+1)^{\beta}-t^{\beta}\right)\end{aligned}$$

Note that $\left(\frac{t}{\alpha}\right)^{\beta} = e^{\beta\cdot\log(\frac{t}{\alpha})}$ where the right hand side gives us a better idea of how the expression is evaluated on a computer. We need to be careful whenever $\alpha\approx 0$ and $t=0$ especially when using auto-differentiation. In implementations where the gradients aren't compiled like Tensorflow we need to write out the optimizations explicitly. One such is that we only need to evaluate $(t+1)^{\beta}$, t^{β} and $\alpha^{-\beta}$ once.

2.6 Recurrent Neural Networks

$$\begin{tikzcd}
& x_1 \arrow[d] & x_2 \arrow[d] & x_3 \arrow[d] & \\
h_0 \arrow[r] & \boxed{\text{ANN}} \arrow[r, "h_1"] \arrow[d] & \boxed{\text{ANN}} \arrow[r, "h_2"] \arrow[d] & \boxed{\text{ANN}} \arrow[r, "h_3"] \arrow[d] & \cdots \\
& y_1 & y_2 & y_3 &
\end{tikzcd}$$

Figure 2.6: RNN : In each step we input feature x_t and hidden state h_{t-1}, forward hidden state h_t and output predicted value y_t

Recurrent Neural Networks (RNNs) is a method of applying Artificial Neural Networks (ANNs) to temporal data such as sequences, time-series, speech or text by feeding neural outputs h_t of the activations of the preceding step back into the network.

Essentially it's a state-space model like Hidden Markov Models, mapping sequence history to hidden space. One can also imagine RNNs as a deep neural network (DNN) that reuses parameters in each layer corresponding to a timestep.

Training of RNNs is just like gradient-based training of a deep neural network. In practice we encounter some technical issues as the number of layers (timesteps) will be large and that each layer has the same parameters. We call this training procedure back-propagation through time (BPT) [5].

RNNs are extremely expressive and flexible. In theory RNNs are turing complete and can thus learn to perform computation and recognize any temporal pattern. The most famous type of RNN-architecture that avoids many of the practical issues of Gradient Descent-based learning with RNNs is the *Long Short Term Memory*-RNN (LSTM) and the simpler Gated Recurrent Unit (GRU). Most notable applications of RNNs have been using these types of architectures [5].

2.6.1 Simple RNN example

Say that the RNN outputs $\hat{y}_t$ in each step and we have square loss:

$$error_t = (y_t - \hat{y}_t)^2 \tag{2.17}$$

In order to learn the weights we need gradients for the function to answer the question 'how much does a change in parameter effect the loss function?$'$ and move the parameters in the direction where the loss gets smaller. The direction is given by the gradient w.r.t the parameters:

$$\nabla error_t = -2(y_t - \hat{y}_t)\nabla\hat{y}_t \tag{2.18}$$

I.e we have a DNN where we get feedback on how good the prediction is at each layer. Since a change in parameter will change every layer in the DNN (timestep) and every layer contributes to the forthcoming outputs this needs to be accounted for.

Take a simple one neuron-one layer network to see this, with x_t some input feature at timestep t and $\hat{y}_t$ the previous prediction:

$$\begin{aligned}
\hat{y}_{t+1} &= f(a + bx_t + c\hat{y}_t) \\
\frac{\partial}{\partial a}\hat{y}_{t+1} &= f'(a + bx_t + c\hat{y}_t) \cdot c \cdot \frac{\partial}{\partial a}\hat{y}_t \\
\frac{\partial}{\partial b}\hat{y}_{t+1} &= f'(a + bx_t + c\hat{y}_t) \cdot (x_t + c \cdot \frac{\partial}{\partial b}\hat{y}_t) \\
\frac{\partial}{\partial c}\hat{y}_{t+1} &= f'(a + bx_t + c\hat{y}_t) \cdot (\hat{y}_t + c \cdot \frac{\partial}{\partial c}\hat{y}_t) \\
&\Longleftrightarrow \\
\nabla\hat{y}_{t+1} &= f'(a + bx_t + c\hat{y}_t) \cdot \left(\begin{bmatrix} 0 \\ x_t \\ \hat{y}_t \end{bmatrix} + c\nabla\hat{y}_t \right)
\end{aligned}$$

We see is that in order to calculate $\nabla\hat{y}_{t+1}$ we calculate i.e roll out $\nabla\hat{y}_t$ which can be done recursively.

With δ the learning rate one training step is then:

$$\begin{bmatrix} \tilde{a} \\ \tilde{b} \\ \tilde{c} \end{bmatrix} \leftarrow \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \delta \cdot (y_t - \hat{y}_t)\nabla\hat{y}_t \tag{2.19}$$

By choosing appropriate loss-functions LSTM's have been used to learn distributions [49]. Most commonly this distribution is a Bernoulli- or a categorical distribution for classification tasks but it can be anything. Let

$$h_t = f(x_t, h_{t-1}, w)$$

Be the update of the hidden state vector where w are the weights for the hidden layer. With

$$\theta_t = g(x_t, h_{t-1}, \tilde{w})$$

as the output layer where theta is parametrizing $p(y|\theta_t)$, p being some assumed distribution for the target value y that we're interested in predicting.

We will not go deeper into the intricacies of using and training RNNs. For a more thorough introduction we recommend [5].

3
Proposed model

In this chapter we formulate the model and discuss it and its usage from a theoretical standpoint. We will start of with an abstract definition of the model (definition 3.1, 3.2). We will then reiterate this using more details (section 3.1).

Definition 3.1 (TTE-RNN)**.** Let

1. y be an observed waiting time,
2. u be an indicator s.t $u = 1$ if y is uncensored $u = 0$ else.
3. $R(\theta, y) = \Lambda(y)$ be some RCHF parametrized by θ
4. θ be the output of a recurrent neural network

If $y \in (0, \infty)$ is continuous and the loss function is

$$error = -\log(f(y)^u S(y)^{1-u}) = -u \cdot \log[\lambda(y)] + \Lambda(y) \tag{3.1}$$

With $\lambda(y) = \Lambda'(y)$ or if $y \in \{0, 1, \ldots\}$ is discrete and the loss function is

$$error = -\log(p(y)^u S(y+1)^{1-u}) = -u \cdot \log\left[e^{d(y)} - 1\right] + \Lambda(y+1) \tag{3.2}$$

With $d(t) = \Lambda(y+1) - \Lambda(y)$ we call the resulting model a TTE-RNN.

Definition 3.2 (WTTE-RNN)**.** The Weibull Time To Event RNN is a special case of definition 3.1. When $\Lambda(y) = \left(\frac{y}{\alpha}\right)^\beta$ we call the model a WTTE-RNN.

In order to understand the model think about what we want to do : In each step we want to make a good guess about the time to the next event using historic information available.

Since we need to be able to compare regions (censored observations) and points (uncensored observations) we need to map observations to some kind of space where this is possible. In preliminaries (chapter 2) we discussed different ways of doing this. We propose to see this as the problem of estimating a probability distribution over the location of the *next* event. In other words, trying to predict the *time* to the next event. With censored data this makes it a survival-problem.

To mathematically motivate the use of the loss function we need at least independence between waiting time and the censoring point (cor. 2.25). This is usually the case if the day that we train the model (censoring point) doesn't have any specific meaning for the outcomes that we're trying to predict.

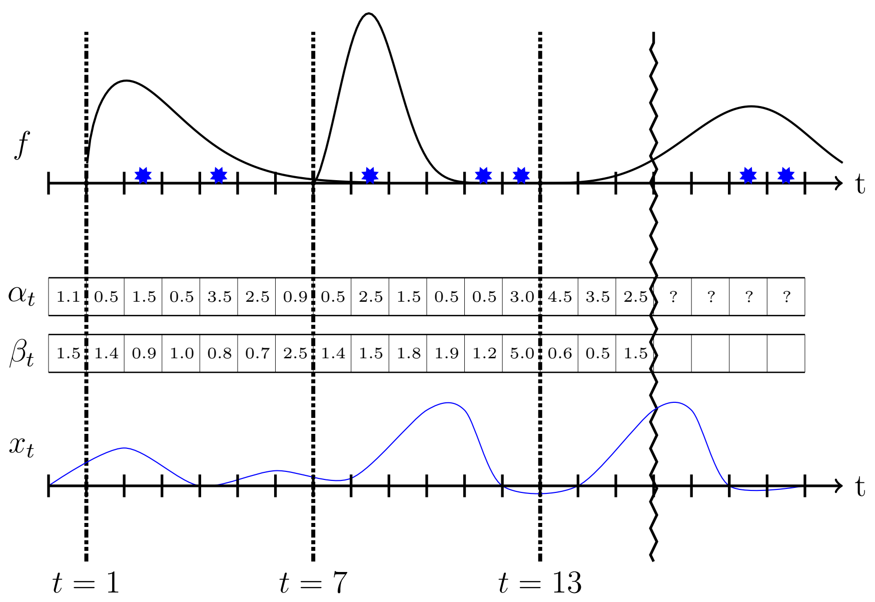
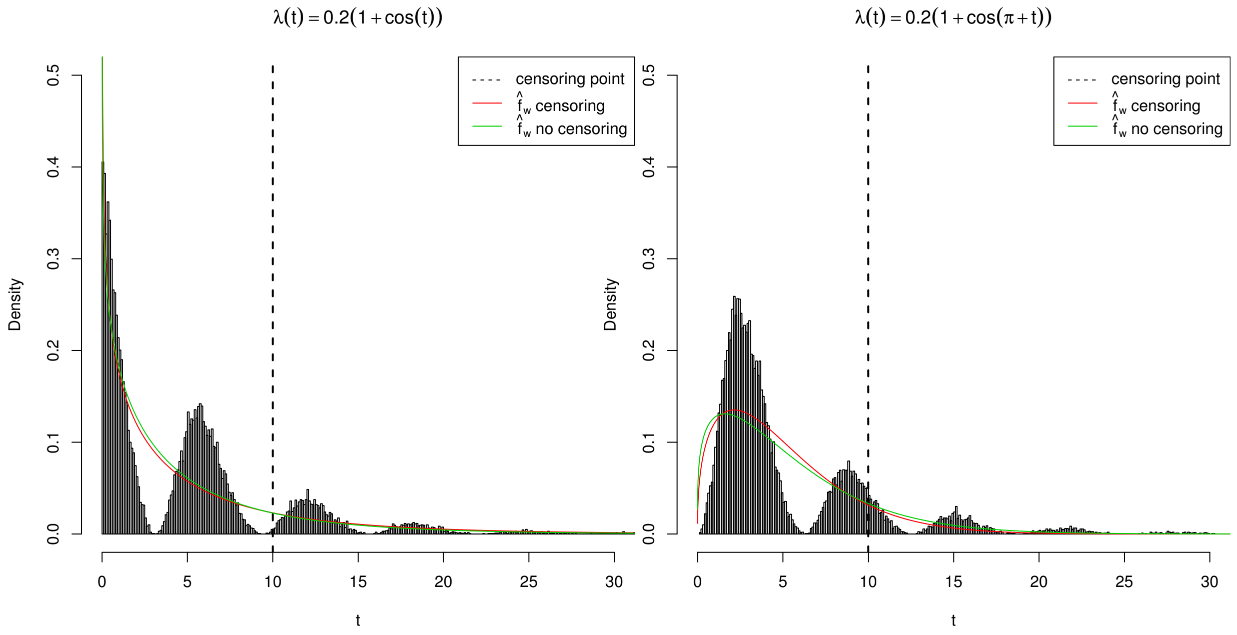


Figure 3.1: Illustrated prediction with the Weibull-model. We have outlined the pdf-prediction $f(s; \alpha_t, \beta_t)$ from steps $t = 1, 7, 13$. Here we imagine that α_t and β_t are functions of features x_t

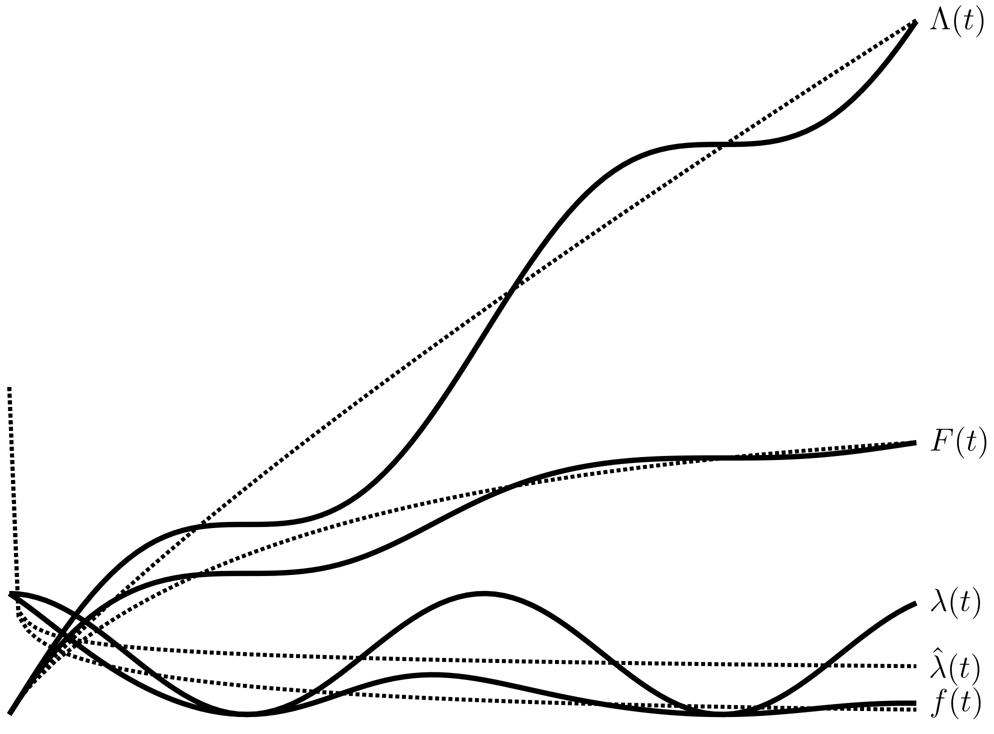
When we have time-to-events over a timeline the target value will behave like a time series of countdowns and jumps (see figure 1.2). To motivate a loss function of being a sum of log-loss terms we also need independence between timesteps. Here we will make the (unproven) Markov assumption of independence given hidden RNN-state. This way we can use the log-likelihood for censored data with good conscience.

We want to stress that the probabilistic arguments above should not worry the reader too much. The optimization problem could be formulated the same way without the probabilistic reasoning: When we have uncensored data we want to concentrate the mass towards an observed point. When we have right censored data we want to push away mass from the region we know *not* to have contained an observation. See figure 2.3. This is exactly what the log-likelihood for censored observations does.

Without specifying a parametric model we can pose this as a functional optimization problem, i.e. trying to find a function minimizing an objective function. Here we want to approximate the true RCHF by finding a $\hat{R}(x, s)$ that maximizes the log-likelihood of the data. By approximating the true $R(x, s)$ by some simple enough parametric model we get a tractable problem. We argue that the Weibull distribution is such a model, see figure 3.2.



(a) **(b)**



(c)

Figure 3.2: Continuous Weibull pdf fitted with and without censoring (occurring at $t = 10$) to data generated from the high-resolution discretized distribution of (a) $\lambda(t) = 0.2(1 + \cos(t))$ and (b) $\lambda(t) = 0.2(1 + \cos(\pi + t))$ using standard algorithms [4]. Figure (c) shows how the uncensored fit from (a). In many applications Weibull is an adequate approximation of very unruly distributions and as we see censoring has little effect on the resulting fit.

3.1 The optimization problem

In this section we will give a minimal mathematical description of the setup of the optimization problem and the chosen simplification.

Variables and parameters

Parameters

Symbol	Range	Description
Indices & sets		
t	$0, 1, 2, \ldots, T$	Timesteps of a sequence
J	$1, 2, \ldots$	# Features
Data		
y_t	$[0, \infty)$	Time to event at timestep t
u_t	$\{0,1\}$	0 if timestep t is right censored. 1 otherwise
x_t	$\mathbb{R}^J$	Feature vector at timestep t
$x_{0:t}$	$\mathbb{R}^{t+1 \times J}$	Seq. of feature vectors until t

Functionals & Variables

Symbol	Range	Description
Functionals		
$\hat{R}(x_{0:t}, s)$	$[0, \infty)$	Estimated RCHF evaluated at s
$\alpha(x_{0:t})$,$\beta(x_{0:t})$	$(0, \infty)$	Positive functions
Variables		
w	$\mathbb{R}^m$	Parameters for RNN

Where we used the notation $\hat{R}(x_{0:t}, s)$ to stress that at time t we estimate a cumulative hazard function using input data $x_{0:t}$ and evaluate it *over the future* $s \in [0, \infty)$. Here s is a variable that we control when using R for prediction while $x_{0:t}$ is the history of features known at t i.e. the input data governing the shape of the function $R(x_{0:t}, s) = g(s)$.

3.1.1 Objective functions

All the objective functions below have the same basic shape, namely as either maximizing the pdf or the survival-function: $\log\left[f(t)^u \cdot S(t)^{1-u}\right]$. Here we just put them on the alternative form described in proposition 2.26.

3.1.1.1 The functional optimization problem

Assume we don't have any distributional assumptions on where the data comes from. We are merely trying to find a functional (function) that fits it.

The main problem is then to find a functional $\hat{R}$ that maximizes the log-likelihood for censored observed waiting times y given feature data $x_{0:t}$ observed until timesteps t for each timestep:

$$\underset{\hat{R}\in\mathbf{S}}{\text{maximize}}\,\mathfrak{L}(\hat{R},y,u,x) := \sum_{t=0}^{T}\left(u_t \cdot \log\left[\hat{R}_s(x_{0:t},y_t)\right] - \hat{R}(x_{0:t},y_t)\right) \tag{3.3}$$

$$\begin{aligned}
&\textit{where } \hat{R}\in\mathbf{S} \;\; \textit{s.t. } \forall x: && && (3.4)\\
&\hat{R}_s(x,s) \geq 0 && \forall\, s \geq 0 && (3.5)\\
&\hat{R}(x,s) \to \infty && \text{as } s \to \infty && (3.6)\\
&\hat{R}(x,0) = 0 && \forall\, s \leq 0 && (3.7)\\
& && && (3.8)
\end{aligned}$$

The optimum is some Recurrent Cumulative Hazard function R which fits the data. When y represents discrete data we use the the discrete log-likelihood from proposition 2.26 with $\hat{d}(x_{0:t},y_t) = \hat{R}(x_{0:t},y_t+1) - \hat{R}(x_{0:t},y_t)$ as:

$$\underset{\hat{R}\in\mathbf{S}}{\text{maximize}}\,\mathfrak{L}_d(\hat{R},y,u,x) := \sum_{t=0}^{T}\left(u_t \cdot \log\left[e^{\hat{d}(x_{0:t},y_t)} - 1\right] - \hat{R}(x_{0:t},y_t+1)\right) \tag{3.9}$$

3.1.1.2 Weibull functional optimization problem

The space $\mathbf{S}$ above in which to search for a solution is vast. Note that $x_{0:t}$ may contain any information available at time t including the position on the timeline t itself and information about previous events.

Proceeding from here we choose to zoom in on a tiny subspace where the cumulative hazard function has the shape of the Weibull distribution.

Let the parameters α and β of the Weibull-distribution be functions of input data $x_{0:t}$ available at timestep t and y_t time to next event at t as above. Using the Weibull Cumulative Hazard:

$$\begin{aligned}
\hat{R}(x_{0:t},y_t) &= \left(\frac{y_t}{\alpha(x_{0:t})}\right)^{\beta(x_{0:t})}\\
\hat{R}_s(x_{0:t},y_t) &= \left(\frac{\beta(x_{0:t})}{\alpha(x_{0:t})}\right)\left(\frac{y_t}{\alpha(x_{0:t})}\right)^{\beta(x_{0:t})-1}
\end{aligned}$$

So from 2.5.2 the new functional optimization problem becomes:

$$\begin{aligned}
\underset{\substack{\alpha>0\\\beta>0}}{\text{maximize}}\,\mathfrak{L}(\alpha,\beta,y,u,x) &:= \sum_{t=0}^{T}\left(u_t\cdot\log\left[\left(\frac{\beta(x_{0:t})}{\alpha(x_{0:t})}\right)\left(\frac{y_t}{\alpha(x_{0:t})}\right)^{\beta(x_{0:t})-1}\right] - \left(\frac{y_t}{\alpha(x_{0:t})}\right)^{\beta(x_{0:t})}\right)\\
&\propto \sum_{t=0}^{T}\left(u_t\cdot\left[\beta(x_{0:t})\cdot\log(\frac{y_t}{\alpha(x_{0:t})}) + \log(\beta(x_{0:t}))\right] - \left(\frac{y_t}{\alpha(x_{0:t})}\right)^{\beta(x_{0:t})}\right)
\end{aligned} \tag{3.10}$$

Where we can drop the constraints on α and β by letting them be some intermediary continuous positive function $g : \mathbb{R} \to \mathbb{R}_+$ such as e^z, $max(\epsilon, z)$, $\log(1 + e^z)$. When we have discrete data we use the discretized form of the log-likelihood i.e the log-likelihood for the discrete Weibull.

From here we may proceed and let z be the outputs of any type of model that can be trained using a differentiable loss-function. This include models trained with Gradient Descent such as artificial neural networks and gradient boosting machines. As our stated goal is to learn temporal patterns and use the whole feature-sequence $x_{0:t}$ we proceed with describing the case when the parameters are outputs of an RNN.

3.1.1.3 Recurrent Neural Network Parametrization

In each step we let α, β be the outputs of a RNN with positive output layer. There's endless possible configurations and architectures for such an RNN. The only limitation we set is that the RNN is outputting a strictly positive and arguably unbounded vector of two components in each step. RNNs with step-to-step (many to many) output is usually (particularly in the Tensorflow-community) referred to as a 'char-level-rnn' [50]. In order to get an understanding for the model we will describe a basic architecture:

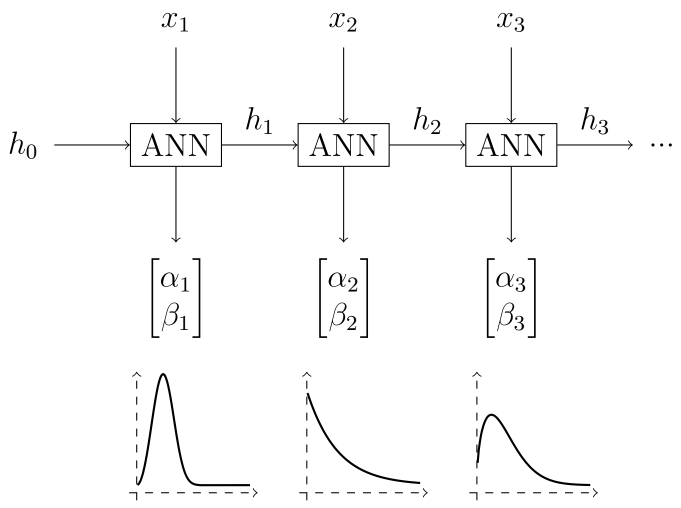

x_1
x_2
x_3
h_0
ANN
h_1
ANN
h_2
ANN
h_3
$\cdots$
$\begin{bmatrix}\alpha_1\\\beta_1\end{bmatrix}$
$\begin{bmatrix}\alpha_2\\\beta_2\end{bmatrix}$
$\begin{bmatrix}\alpha_3\\\beta_3\end{bmatrix}$

Figure 3.3: Basic architecture of the RNN. In each step we output estimated Weibull-parameters giving us a distribution over time to the next event.

Let $h_t = i(x_t, h_{t-1}, w_h)$ be the hidden state vector, a function of previous hidden state h_{t-1}, feature data x_t at timestep t and weights for the hidden layer w_h. Let $\vec{z}_t = o(h_t, w_o) \in \mathbb{R}^2$ be some two-dimensional output from the output-layer and let g be some positive activation function. This can be described as:

$$
\begin{aligned}
h_t &= i(x_t, h_{t-1}, w_h) \\
z_t &= o(h_t, w_o) \in \mathbb{R}^2 \\
\begin{bmatrix}\alpha_t \\ \beta_t\end{bmatrix} &= g(z_t)
\end{aligned}
$$

We can of course simplify this by writing the outputs in each step as:

$$
\begin{bmatrix}\alpha_t \\ \beta_t\end{bmatrix} = f(x_t, h_{t-1}, w)
$$

Where f is some neural network and h_{t-1} the value of its activation-functions at the previous step. The network architecture is showed in 3.3.

The unconstrained optimization problem in the continuous and discrete case respectively is then to find w maximizing the log-loss:

$$
\begin{aligned}
\underset{w}{\text{maximize}}\ \log(\mathfrak{L}(w, y, u, x)) &:= \sum_{t=0}^{T} \left(u_t \cdot \left[\beta_t \cdot \log(\frac{y_t}{\alpha_t}) + \log(\beta_t) \right] - \left(\frac{y_t}{\alpha_t}\right)^{\beta_t} \right) \\
\underset{w}{\text{maximize}}\ \log(\mathfrak{L}_d(w, y, u, x)) &:= \sum_{t=0}^{T} \left(u_t \cdot \left[\exp\left[\left(\frac{y_t+1}{\alpha_t}\right)^{\beta_t} - \left(\frac{y_t}{\alpha_t}\right)^{\beta_t} \right] - 1 \right] - \left(\frac{y_t+1}{\alpha_t}\right)^{\beta_t} \right)
\end{aligned}
$$

See section 2.6 and [5] on how we do gradient based learning for optimal w.

3.2 Properties of the Weibull-loss

3.2.1 Gradients

In this section we will discuss the theoretical properties of the Weibull log-loss. The main result that we will discuss is that the continuous-time Weibull loss has a unique optimum.

The gradients of the continuous Weibull log-likelihood can be written

$$\log(\mathfrak{L}) \propto u \cdot [\beta \cdot [\log(t) - \log(\alpha)] + \log(\beta)] - \left(\frac{t}{\alpha}\right)^{\beta}$$

$$\begin{aligned}
\frac{\partial \log(\mathfrak{L})}{\partial \beta} &= u \cdot \left[\log(\frac{t}{\alpha}) + \beta^{-1}\right] - \log(\frac{t}{\alpha}) \cdot \alpha^{-\beta} t^{\beta} \\
&= u \cdot \beta^{-1} + \log(\frac{t}{\alpha}) \cdot (u - \alpha^{-\beta} t^{\beta}) \\
\frac{\partial \log(\mathfrak{L})}{\partial \beta^2} &= -u \cdot \beta^{-2} - \log(\frac{t}{\alpha})^2 \cdot \alpha^{-\beta} t^{\beta}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \log(\mathfrak{L})}{\partial \alpha} &= -u \cdot \beta \cdot \alpha^{-1} + \beta \cdot t^{\beta} \alpha^{-\beta-1} = \alpha^{-1}\beta \cdot (-u + t^{\beta}\alpha^{-\beta}) \\
\frac{\partial \log(\mathfrak{L})}{\partial \alpha^2} &= u \cdot \beta \cdot \alpha^{-2} - \beta \cdot (\beta+1) \cdot t^{\beta} \alpha^{-\beta-2} \\
&= \alpha^{-2}\beta \cdot (u - (\beta+1) \cdot t^{\beta}\alpha^{-\beta})
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \log(\mathfrak{L})}{\partial \alpha \partial \beta} &= -u \cdot \alpha^{-1} + t^{\beta}\alpha^{-\beta-1} + \beta \cdot \alpha^{-1} \frac{\partial}{\partial \beta} e^{\beta \cdot \log(\frac{t}{\alpha})} \\
&= -u \cdot \alpha^{-1} + t^{\beta}\alpha^{-\beta-1} + \beta \cdot t^{\beta} \cdot \alpha^{-\beta-1} \cdot \log(\frac{t}{\alpha}) \\
&= \alpha^{-1}\left[-u + t^{\beta}\alpha^{-\beta}\left[1 + \beta \cdot \log(\frac{t}{\alpha})\right]\right]
\end{aligned}$$

The regular technique for regression and estimation with the Weibull is to focus on the loss after using the location-scale-transformation (proposition 2.31) and work on the Gumbel-random variable $Y = \log(T)$ directly. In doing so, Scholz (1996) showed how the MLE is unique for multiply censored data and can be found using standard algorithms as there's no other local maximas [51]. We will not reiterate his proof since it's quite long, instead we make a short appeal to the intuition why this is the case. The Gumbel CHF,Hazard and logloss are:

$$\Lambda_G(y) = e^{\frac{y-\mu}{\sigma}} \tag{3.11}$$

$$\lambda_G(y) = \frac{1}{\sigma} e^{\frac{y-\mu}{\sigma}} \tag{3.12}$$

$$\implies \tag{3.13}$$

$$\log(\mathfrak{L}_G) = u \cdot \left[\frac{y-\mu}{\sigma} - \log(\sigma)\right] - e^{\frac{y-\mu}{\sigma}} \tag{3.14}$$

Think about having a sum of one uncensored observation and many censored observations. The log-likelihood (equation 3.14) for all observations will have the the (rightmost) term $-\Lambda_G(t) = -e^{\frac{y-\mu}{\sigma}}$ which is *always* maximized when $\mu \nearrow \infty$. Censored observations only consists of this term since $u = 0$. Uncensored observations ($u = 1$) will have the term $\frac{y-\mu}{\sigma}$ which is *always* maximized when $\mu \searrow -\infty$. When we maximize the whole log-likelihood we thus get a competition between pushing μ up contributing exponentially decreasing returns for all observations and linear costs for the uncensored observations and pushing μ down contributing exponentially increasing costs and linear returns. Somewhere in the middle we find the optimum. A similar argument can be made about σ [45].

Uniqueness is of course not true if we only have censored observations. Intuitively we know that a censored observation $T > t$ motivates us to push the mass over to the other side of t. More mass comes over as $\alpha \to \infty$ with diminishing returns for the loss function so the gradient will tend to 0 but never disappear. Consider when $u = 0$ so that $\frac{\partial \log(\mathfrak{L})}{\partial \alpha} = \beta t^{\beta} \alpha^{-\beta-1} := 0$. This is only achieved when $\alpha \to \infty$ or $\beta \to 0$. To reach $\beta = 0$ via gradient descent we would need $\frac{\partial \log(\mathfrak{L})}{\partial \beta} < 0$ which is never the case when $t < \alpha$ so for highly censored data β is expected to remain stable while α will explode but settle at a large value. If there's *only* censored observations $\alpha \to \infty$. Numerical instability and exploding gradients with highly censored data is a well known problem. This happens when we are far away from the optimum and the log-likelihood is $-\infty$. Extra care is therefore recommended to be taken during initial learning and if we experience overflow we can reinitialize [51]. When we are using stochastic gradient descent there is a (slim) probability that we will get a sequence of gradient-steps with batches only containing censored observations. The probability of this happening is proportional to the share of censored observations.

3.2.1.1 Discrete case

The gradient of the discrete Weibull is too complex to get any useful analysis from writing it out explicitly. Note instead that in general for discretized distributions we have:

$$\begin{aligned} \log(\mathfrak{L}_d) \propto u \cdot \log\left(e^{d(t)} - 1\right) - \Lambda(t+1) \implies \\ \frac{\partial \log(\mathfrak{L}_d)}{\partial \theta} &= u \cdot \frac{e^{d(t)}}{e^{d(t)} - 1} \cdot \frac{\partial d(t)}{\partial \theta} - \frac{\partial \Lambda(t+1)}{\partial \theta} \\ &= u \cdot \frac{\Lambda_\theta(t+1) - \Lambda_\theta(t)}{1 - e^{-d(t)}} - \Lambda_\theta(t+1) \end{aligned}$$

We need to think about all the numerical caveats of the continuous distribution discussed above but here we see that we also need to be particularly worried about when $1 - e^{-d(t)} \approx 0$ and $e^{d(t)} - 1 \approx 0$. We have not found any convincing proofs that the discretized loss has the uniqueness-properties of the continuous distribution. We have on the other hand had very good empirical results when using it with proper initialization. See the experiments in section 4.1 for intuition about how it works.

3.2.2 Extensions

3.2.2.1 Multivariate

It's easy to imagine situations where we want to predict the time to multiple types of events using the same feature-data, say a vector of random waiting times $\vec{\mathbf{T}}_t$.

If we make the naive assumption that the components are independent given their parameters, $T^k \perp T^l | \theta$, with θ the output of the same neural network, the appropriate loss function is simply the sum of the log-likelihood for each component. This would be akin to the Naive Bayes Classifier. To make it a truly multivariate problem we need to think up some appropriately expressive covariance structure but easy enough to have a closed form calculation of its integral.

One application of a multivariate model could be to predict 'when and how much' meaning that one of the components wouldn't even be derived from events.

In other circumstances we might think its multivariate problem but it really isn't. If we're really interested in the time to the first of these events we can use the regular 1d Weibull. This follows from the minimum closure- and Weakest Link-property (prop 2.32 and thm 2.33).

3.2.2.2 Other distributions

We have intentionally formulated us s.t it should be easy to switch out the Weibull to something else. Note that we only need a valid hazard function to specify a model. This could also be learned directly. Note that hazard functions (and CHFs) are closed under positive linear combinations. This means that we can create new multimodal distributions by combining simple basis CHFs $\Lambda(t) = \sum_k \Lambda_k(t)$. These could for example be Weibull CHFs.

3.2.2.3 Opportunities to regularize

As we have shown, when $\beta \nearrow \infty$ all mass of the distribution is centered around $t = \alpha$. Remember the location-scale transformation of section 2.5. Here we showed that $\sigma = 1/\beta$ is a scale parameter for the log of the Weibull. Just as we can control the peakedness of a normal-distribution when using a Gaussian Kernel by not allowing $\sigma \searrow 0$ we could avoid overfitting data by penalizing large values of β.

To regularize we simply add some growing function g of β_t, say $g(x) = e^{\delta x - l}$, to the original loss-function:

$$\tilde{\mathfrak{L}}(w, y_{0:t}, u_{0:t}, x_{0:t}) = \mathfrak{L}(w, y_{0:t}, u_{0:t}, x_{0:t}) - g(\beta_t) \tag{3.15}$$

In the experiments we also found that for numerical stability it was useful to initialize and keep β close to 1 to let α settle in as β would go to 0 if α was initialized too small. Properly initializing α was the simpler alternative which we recommend over this technique.

3.3 Usage & Applications

In order to implement this model we only need a step-to-step implementation of some *RNN* where we switch out the loss-function to the Weibull-loss function. That is, assuming that we have defined the sequences of features, time-to-events and censoring-indicators. In this section we discuss some of the intricacies of doing this discussed in an applied context *churn prediction*.

3.3.1 Example : A model for churn-prediction

Predicting customer churn is the problem of trying to predict whether a customer will stay or not. Creating a good definition of this business-critical concept is known to be a hard modeling problem. The typical way of modeling this is to frame it as a binary prediction problem using the sliding-box model (section 2.4.1) [14].

Usually we have recurrent events (like repeat purchases, payment of subscription) and actions (contacted support, opened newsletter, browsed page, lost cart etc) and static features (sex, country, currency etc). Even though the concept of a customer not returning is easy to grasp, defining what should constitute churn is hard. One may find some baseline ('did buy alot, now buys a little') or some line in sand ('hasn't bought for 100 days'). When using the sliding box model we need to hard-code this into the target-value by setting some threshold. When someone goes above this treshold they are considered churned. With our approach we only need to define what the qualifying event should be and predict the time to it, pushing the tresholding problem to after the model is trained. Here *churn* would be the predicted *lack of events*. When customers have a rise in their predicted time of returning we can quickly act, even using different thresholds for different groups or base it upon the change in predicted value.

At prediction time we only need to update the hidden state with the daily feature state. From the predicted α and β assigned to each customer and day we get an embedding that can be used to compare customers and predict the churn rate for our whole customer stock. By adding the individual densities we can easily visualize how many of the current customers are expected to have made a repeat purchase in the upcoming month or so. This might also be useful in estimating the expected lifetime value of the whole customer stock.

From the estimated parameters we can directly read out metrics such as expected mean, median and characteristics of their event rate (hazard function). By analyzing the predicted hazard we can operationalize useful concepts like whether customers are predicted to have a decreasing ($\beta < 1$), constant ($\beta = 1$), or increasing ($\beta > 1$) event rate and how much this rate will be (α).

Fixing α one can interpret this as:

- $\beta < 1$: Rate of event expected to be decreasing (They return now or never).
- $\beta = 1$: Rate of event expected to be constant (We don't know).
- $\beta > 1$: Rate will increase (and the probability will peak around a specific time ex. Christmas).

Plotting customers onto a $\alpha \times \beta$ grid should be very helpful in understanding how the current customer base looks like. Weibull-plots are well known concepts in reliability engineering and can therefore be interpreted by many engineers. The path that a customer has taken on this plane may tell a story about where in the customer life-cycle they are and have been before. See figure 4.8 to see an example. It may also be useful to use the hidden state of the RNN for a wider embedding based on recency- and frequency of customers. This could be used for exploratory analysis to find clusters among customers or to train other models.

3.3.2 A note on using sequences of varying length

With sequences of varying length we need to interpret the probabilistic aspects of the model accordingly or weight the model during training to get the sought interpretation. We will use the case of churn-prediction to illustrate this.

Imagine that we have recorded data for users since the first signup and we'd had exponential growth since. By stacking these timelines from when they started we get something like figure 3.4.

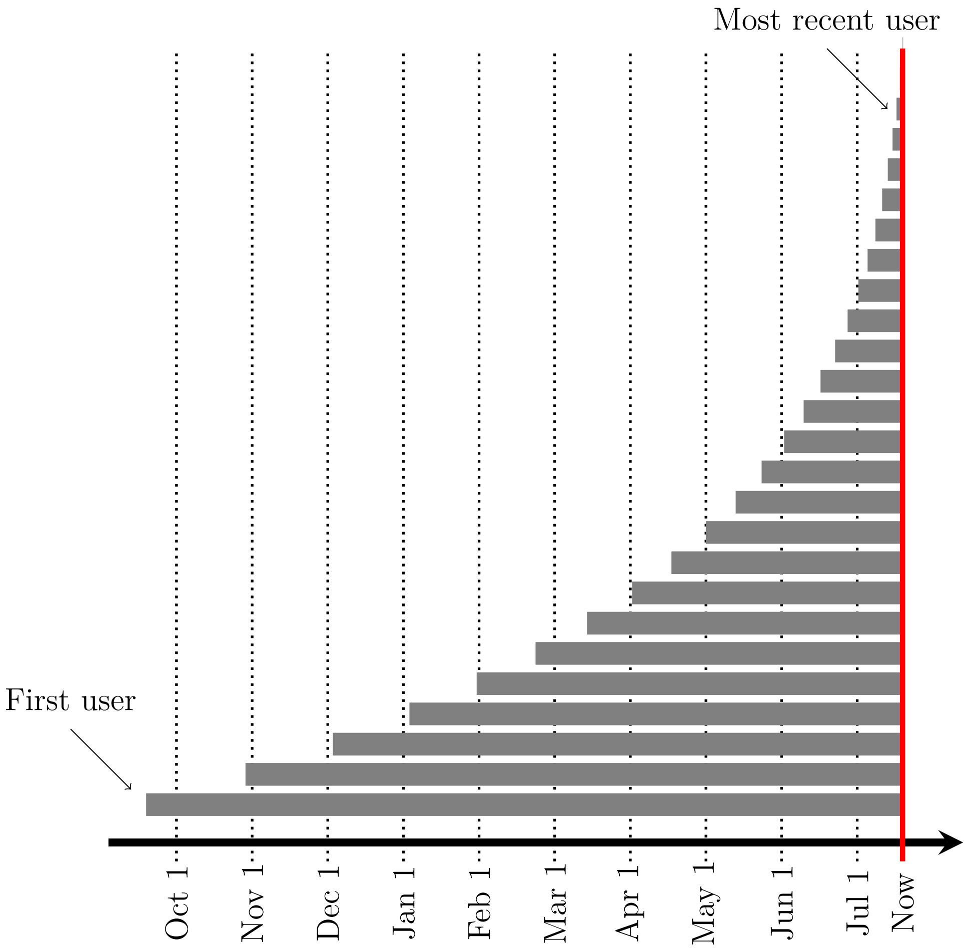


Figure 3.4: Stacked timelines of available data per user

This type of data-collecting process is typically called *staggered entry*. The continuous- or high-resolution data will be aggregated at fixed times and indexed per user n and timestep $t = 0, \ldots, T_n$. These intervals can be cut out from every second to every 30 days in user- or in wall-clock- time. To avoid confusion, note that our indexing is always per user. We will briefly discuss different schemes of weighting the loss function and its implications.

Let $\ell_t^n = \log(\mathfrak{L}_t^n)$ be the log-likelihood for timestep t sequence n and $\omega(t, T)$ be a weighting-function. Consider the goal of maximizing the loss function, the weighted sum $\tilde{\ell} = \sum_n^N \sum_t^{T_n} g(t, T_n)\ell_t^n$. In practice we will use stochastic gradient descent and sample terms in this sum randomly but this makes no difference for the discussion. A non-exhaustive list of possibilities is then:

1)	$g(t, T_n) = I(\tau_1 < t + (T_0 - T_n))I(\tau_2 < T_n)$	Use timesteps in some rectangle*
2)	$g(t, T_n) = 1$	Timesteps has same influence
3)	$g(t, T_n) = \frac{1}{T_n}$	Users has same influence
4)	$g(t, T_n) = \tilde{g}(\frac{t}{T_n})$	Weight by recency, user time
5)	$g(t, T_n) = \tilde{g}(t, T_n, T_0)$	Weight by recency, wall-clock time

Now consider how we can interpret the prediction at time t. Assume that we have feature data for new users $x_{0:T_n}^n$ that we've used to estimate the parameter θ_t^n. Fix ρ and set y^n s.t $\rho := \hat{\Pr}(W \leq y^n | \theta_t)$. As an example, if $\rho = 0.5$ then y_n is the predicted median.

1) Use timesteps in some rectangle*: Here we choose only the last τ timesteps from sequences that we've observed for at least τ timesteps. This can be seen as selecting/inscribing a fitted rectangle among the stacked timelines in figure 3.4. As each user contributes the same to the loss, a (hypothesized) interpretation of ρ is:

- *If we repeat the experiment* $(\rho \cdot 100)\%$ *of the predicted users will have an event within* τ *days*

Which seems close to what we would hope to get out from such a model.

Note that when τ is smaller we get more users represented in the dataset (rectangle gets higher) but observe them for shorter times and with more censoring as it will only contain the most recent timesteps.

2) Timesteps has same influence This is what happens if we don't apply weighting. An interpretation of ρ is:

- *In* $(\rho \cdot 100)\%$ *of recorded timesteps an event will occur within* y *days*

So this is a retrospective probability of what has occurred in the dataset of given shape. Note that long-time users will have a big influence on the loss.

If we disregard probabilistic aspects and use the model to rank users according to their risk-set, this could make sense as we know more about these users and if the individual patterns are apparent one can speculate that it wouldn't skew the RNN-weights too bad. One such pattern could be 'no activity' which might be easily learned.

3) Users has same influence This is what happens when we take mean-error per sequence. Every user will then have the same influence on the loss regardless of how long they've been observed. In all the experiments we trained using this formulation.

In experiment of section 4.2 we evaluated calibration using 2), i.e over all the timesteps and the calibration was found to be decent. In experiment of section 4.4 we evaluated using 1) and had similar results. Apart from that we have not dug deeper into how weighting affects the prediction as we've found very little written about it. We assume that the underlying data distribution has big effect on the outcome. One thing to keep an eye on is that we assume waiting times to be in range $(0, \infty)$. Big values of waiting times may not even be possible to observe if they occur at all. This will likely lead to some consistent bias. Ways to tackle this is proposed as future work.

4

Experiments

In this chapter we will show the model in action. We implemented the loss function in Tensorflow which supports automatic differentiation.

```
def weibull_logLik_continuous(a_, b_, y_, u_,name=None):
    ya = tf.div(y_+1e-35,a_)
    return(
        tf.mul(u_,
               tf.log(b_)+tf.mul(b_,tf.log(ya))
              )-
        tf.pow(ya,b_)
    )

def weibull_logLik_discrete(a_, b_, y_, u_, name=None):
    with tf.name_scope(name):
        hazard0 = tf.pow(tf.div(y_+1e-35,a_),b_)
        hazard1 = tf.pow(tf.div(y_+1,a_),b_)
    return(tf.mul(u_,tf.log(tf.exp(hazard1-hazard0)-1.0))-hazard1)
```

We added some small numbers $1e-35$ to avoid numerical instability.

4.1 Basic gradient-based estimation with the Weibull distribution

In this experiment we performed sanity checks of our implementation and showed how gradient descent-based estimation works when using the censored likelihood-loss.

4.1.1 Setup : Generate Weibull data

The goal was to see whether we can recover the correct parameters when from Weibull data with different levels of censoring. This was shown for continuous and discrete generated data with medium ($\alpha = 20$) and low ($\alpha = 2$) resolution and different initializations. β was kept at 2 during all experiments.

Data was generated using the censoring- and discretization- mechanism described in section 2.3:

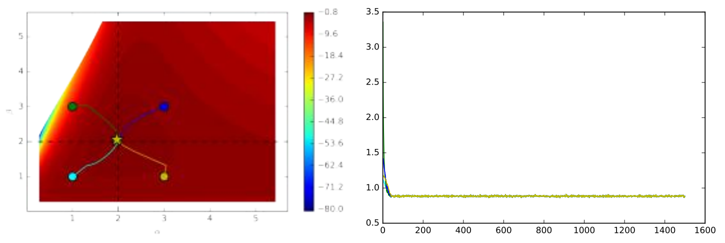
$$\begin{aligned}
U &\sim \text{Uniform}([0,1]) \\
W &= \alpha \cdot (-\log(U))^{\frac{1}{\beta}} \\
W_d &= \text{floor}(W) \\
X &= \min(W, c) \\
X_d &= \min(W_d, c) \\
\Delta &= W \leq c
\end{aligned}$$

With $X = x$ and the failure indicators $\Delta = u$ being the variable we can observe. Here we fix c s.t a predetermined % of the observations were censored during the experiments. This censoring boundary is shown as a vertical line in the histogram of figure 4.1 and 4.2. In the real world this type of censoring would occur when we want to estimate parameters after observing an experiment for c days.

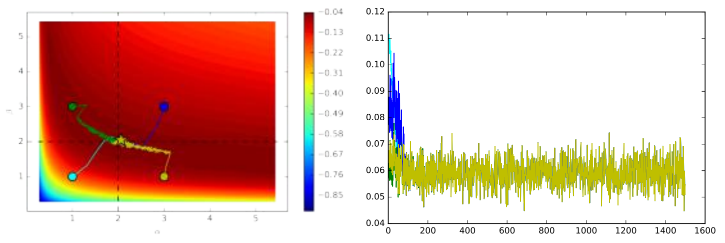
In all of the experiments we let parameters be a softplus-function of an unbounded bias-weight which we optimize for. Batch size was fixed to 10000 datapoints generated for each of the 1000 training-steps. In the plots we show trajectories from when we used the RMSprop-optimizer but using Adam-optimizer gave similar results.

4.1.2 Results : Smooth learning with proper initialization

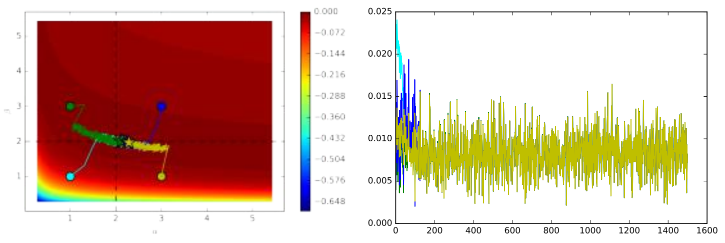
Initialization was found to be very important. If initialized too far from the true value the gradients has a high probability of exploding resulting in parameters going NaN. This was particularly apparent for the discrete Weibull and becomes an increasing problem with more and more censoring. After this conclusion $\hat{\alpha}$ and $\hat{\beta}$ was initialized close (factor of 0.5 and 1.5) to their actual values. This was less of a problem when using optimizers without momentum such as RMSprop. When initialized adequately close we had no exploding gradients regardless of choice of optimizers or learning rate. We chose to display the results with RMSprop here as the path is more steadfast while Adam tended to jump to the correct point in the first couple of iterations.



(a) No censoring



(b) $> 99\%$ censoring



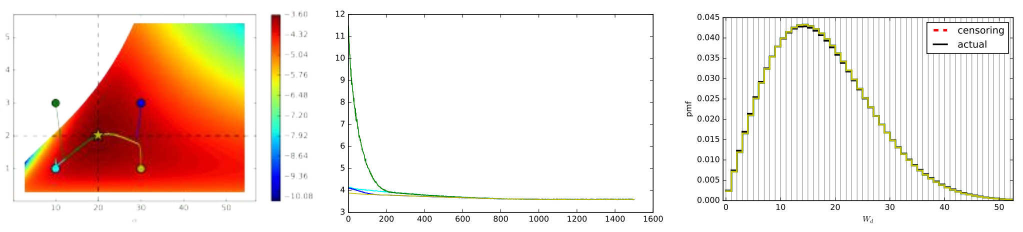
(c) $> 99.9\%$ censoring

Figure 4.1: Continuous Weibull $\alpha = 2$, $\beta = 2$. Four different trajectories from different initializations (circles). Contourplot shows log-likelihood for a batch of 10000 datapoints. Histogram are overlaid with respective final step (star in contourplot).

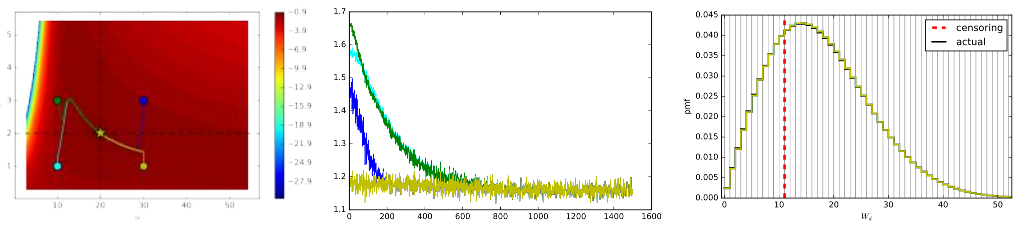
In the experiment with continuous Weibull (see figure 4.1) we could set almost any level of censoring and would still converge to the correct parameters. This is expected as it's shown that the likelihood function for censored data has a unique optimum that we're expected to reach with gradient descent (see section 3.2.1).

With high censoring the convergence was markedly slower and more noisy. This is probably due to the fact that with more censoring the probability of seeing an uncensored value gets smaller so there's less to learn shape, from explaining the speed. The log likelihood will also have a larger magnitude for censored observations which explains the noise. Note that in the real world we have finite data so if the censoring probability is high the few observations below the censoring-threshold will have high influence on determining the shape of the distribution.

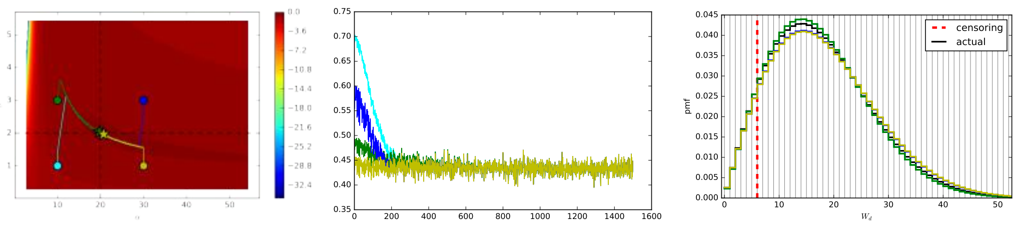
In the contour-plots of figure 4.1 we see how - regardless of censoring, the optimizer will find its way down into the valley and step towards the optimum. In our plots there's some apparent differences between initializing above or below the generating alpha. When plotting the scale parameter α on a logarithmic the rate of convergence looks almost identical.



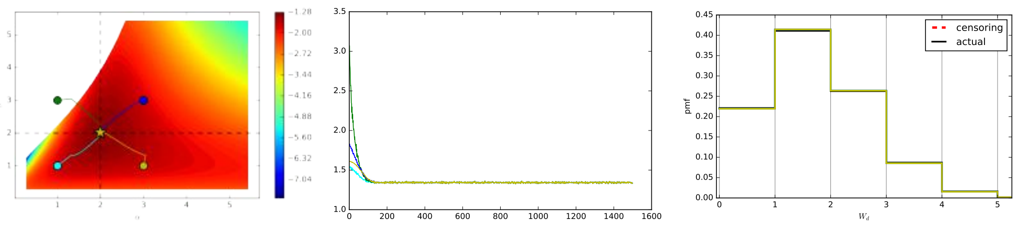
(a) ($\alpha = 20$) No censoring



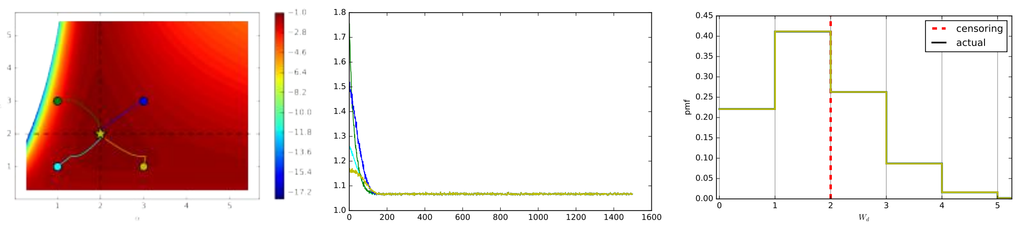
(b) ($\alpha = 20$) $> 73.9\%$ censoring



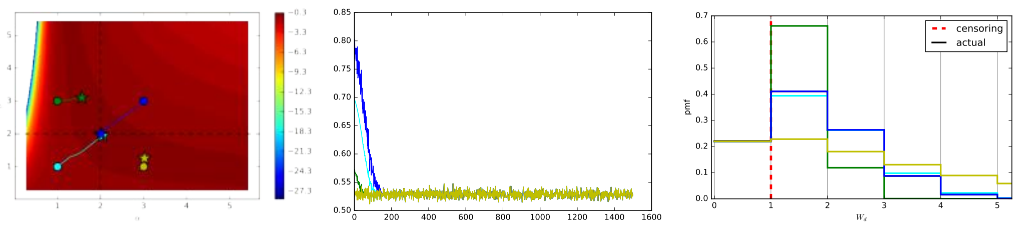
(c) ($\alpha = 20$) $> 91.4\%$ censoring



(d) ($\alpha = 2$) No censoring



(e) ($\alpha = 2$) $> 36.8\%$ censoring



(f) ($\alpha = 2$) $> 77.9\%$ censoring

Figure 4.2: Discrete Weibull with different generating α. Trajectories from four different initializations (circles). Contourplot shows log-likelihood for a batch of 10000 datapoints. Histogram are overlaid with respective final step (star in contourplot).

In the discrete experiments we found - as expected - that higher resolution, i.e higher scale-parameter α, will lead to a more robust fit (figure 4.2) as the the problem will look more and more like a continuous one. For discrete Weibull we've found no mathematical proof that the likelihood has a unique optimum for all levels of censoring. In practice we found that with no censoring the correct parameters were recovered. With higher censoring it seems, judging by the contour plots, that the optimum is shifting away from the generating parameters.

In the discrete case the behavior during convergence has alot to do with the number of bins that the discretization induces. When fixing $C = c$ (seen as the vertical red dotted line in the histograms) s.t only one bin is shown green and yellow got stuck in a local minimum. 4.2f.

With one bin available too us the only information given is the probability of $W_d = 0$. The resulting log-likelihood will be identical to a Bernoulli-log-likelihood with a numerically unstable representation $\theta = \Pr(W_d = 0) = 1 - e^{-\alpha^{-\beta}}$. Any point $(\hat{\alpha}, \hat{\beta})$ on this line is as good of a guess. This means that it's only luck determining if the guess is right or not.

With more bins, i.e $W_d \in \{0, 1, \ldots C\}$ we can see it as if it gets easier to fit a line to match the observed bins. As the censored observations will push mass to the right the error typically shows up in the form of a too fat right tail but initialization clearly plays an important role here. This is most apparent in the contour-plots in figure 4.2b and 4.2c.

4.1.3 Notes on initialization

Initializing α below the true value seems to slowly push α and β down until the gradient is vanishing leading to convergence. By the logarithmic nature of α it takes longer to go one unit step from above to down to the correct value than from below up to the correct value. For other values of the actual β the loss surface ravine gets narrower but less steep in the α-direction, making convergence of α even harder.

One idea for initializing α is to take the $1 - e^{-1} \approx 0.63$'th quantile in the continuous case as its an estimator for α. This does not work in the discrete case. For simplicity we found that initializing the model as an exponential- or geometric distribution ($\beta = 1$) with a parameter given by this distributions Maximum Likelihood for censored data worked well for a range of values of α and β. The MLEs are defined in the following.

Definition 4.1 (Exponential MLE/initialization).

$$\begin{aligned} Y \sim & exp(\alpha) \equiv Weibull(\alpha, 1) \\ \Lambda(y) =& \frac{y}{\alpha} \\ \log(\mathfrak{L}(\alpha)) =& \sum_i u_i \cdot \log(\frac{1}{\alpha}) - \frac{y_i}{\alpha} \\ =& - n_u \cdot \log(\alpha) - \frac{\sum_i y_i}{\alpha} \\ \iff & \\ \alpha_{MLE} =& \frac{y_i}{n_u} \end{aligned}$$

With a continuous WTTE-RNN, If we initialize the biases using this MLE we call it *exponential initialization*

Definition 4.2 (Geometric MLE/initialization).

$$\begin{aligned} Y \sim & Geometric(\alpha) \equiv Weibull_d(\alpha, 1) \\ d(y) =& \Lambda(y+1) - \Lambda(y) = \frac{1}{\alpha} \\ \log(\mathfrak{L}(\alpha)) =& \sum_i u_i \cdot \log(e^{\frac{1}{\alpha}} - 1) - \frac{y_i + 1}{\alpha} \\ =& n_u \cdot \log(e^{\frac{1}{\alpha}} - 1) - \frac{n + \sum_i y_i}{\alpha} \\ \iff & \\ \alpha_{MLE} =& - \frac{1}{\log(1 - \frac{n_u}{n + \sum_i y_i})} \end{aligned}$$

With a discrete WTTE-RNN, If we initialize the biases using this MLE we call it *geometric initialization*

Initializing using this assumption is the same as initializing the model in the continuous case as a Proportional Hazards-model or equivalently an exponential Accelerated Failure Time model. This means we make no assumption on whether the hazard is increasing- or decreasing. A problem with this strategy is that in most real world situations we have thinner tails than the exponential and geometric distribution assumes since the true data distribution does not contain infinity. If the real distribution has $\beta << 1$, $\beta >> 1$ or small α we get thin tails. Using the geometric/exponential assumption ($\beta = 1$) would in that case lead to a gross overestimation of the true α. As seen from these experiments initializing above the actual values is worse than below. One middle way is therefore to use domain knowledge to guess a reasonable truncated expected value and then set β to 1 or naively setting $n_u = n$ in the MLE and treat all observations as uncensored, leading to a smaller α. More research on this would be worthwhile.

As initialization of parameters has alot of influence on stability and convergence, having variational nodes [49] (nodes which are perturbed with some noise during forward propagation) at the output layer could be a remedy if one gets stuck. We did not test this but propose it as future work.

4.2 C-MAPSS : high-dimensional uncensored data

In this section we will illustrate an application of the model to dataset without censoring. The *CMAPSS dataset* [33] is well known in the Prognostics and Health Management (PHM) community as a reference for evaluating predictive performance. The data comes from hundreds of simulated run-to-failure experiments of aircraft engines. Each experiment is a sequence of length proportional to the survival length of the machine. The experiments are all initialized using different initial values and operational settings and during simulation mixed with random effects. This results in sequences of 26 noisy sensor measurements. The goal is to use the features for stepwise prediction of the remaining useful life (RUL) or failure time of the aircraft engines.

The dataset was first used in the *PHM08 Prognostics Data Challenge* [34] where a training set was made available and a test- and validation set was kept by the organizers. The test- and validation- set came from a shifted and different distribution of data and had added outliers in it. Researchers were allowed to evaluate their predictions on the test data multiple times while the final score was evaluated on a final third validation set.

4.2.1 Setup : Predict failure time of aircraft engines

We did not follow the PHM08-challenge way of treating data as their test/train-data split wasn't done using regular cross-validation techniques. Instead we created our own random fold by choosing a subset of the CMAPSS data (*trainFD002* and *trainFD004*) resulting in a dataset of 418 sequences. This was then randomly split to 169 validation- and 249 training-sequences. None of the engines brake down before 128 steps. To limit training time this uneventful period was removed and the longest sequence was then truncated to 254 steps. Figure 4.4 gives a good indication of the distribution of sequence lengths.

The architecture of the LSTM was $26 \times 100 \times 10 \times 2$ with the 100-node layer being the only recurrent i.e the hidden state size was 100. All hidden layers used tanh-activation except the softPlus output-layer. Feature data was normalized using training-set weights. Even though the waiting times spanned a large range (0-254) the existence of 0-time observations and the inherently discrete nature of the data led us to use the discrete Weibull model.

The biases of the output layer was initialized so that we get a baseline geometric model i.e $\hat{\beta} = 1$ and $\hat{\alpha} = 71.04$ with $\hat{\alpha}$ the MLE for the geometric distribution with $\hat{p} = 1 - e^{-\frac{1}{\alpha}}$ calculated using the mean tte $\bar{y}$ from training data i.e $\hat{\alpha} = -\frac{1}{\log(1-\frac{1}{\bar{y}+1})} \approx \bar{y}$. Loss was calculated as mean over timesteps. With batch-size 1 this means that each individual training sequence was given equal weight regardless of length. The model was trained for 60k iterations as shown in figure 4.3.

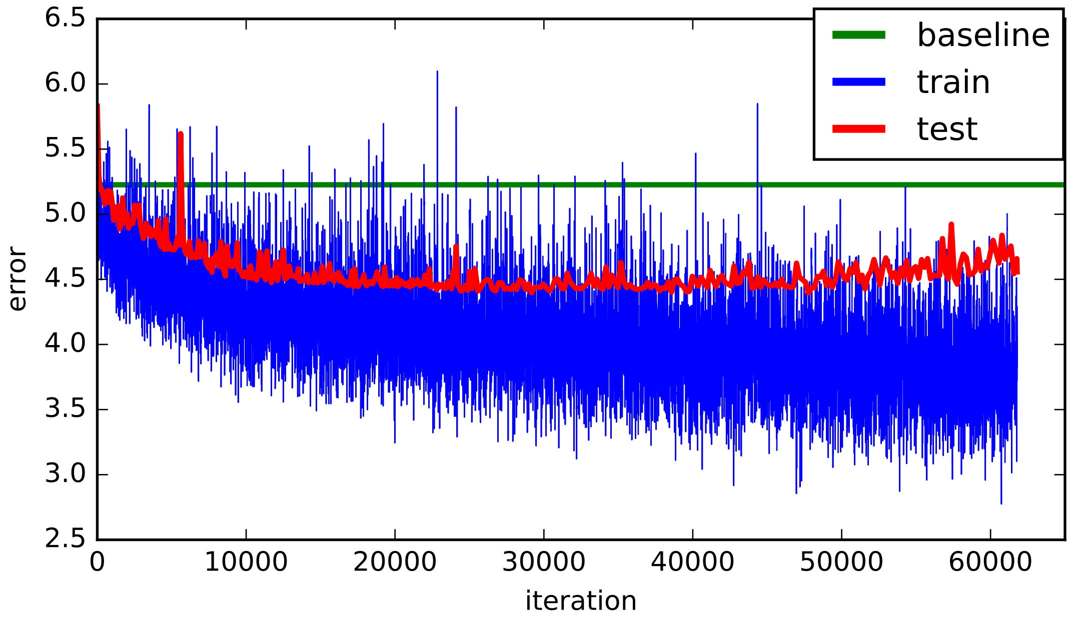
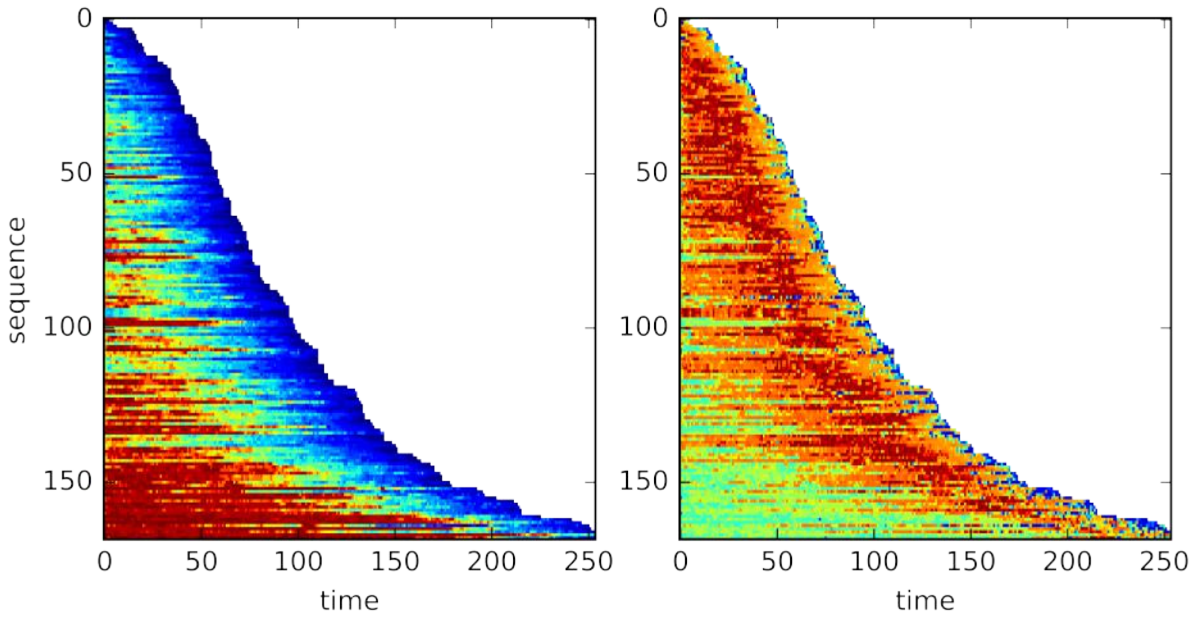


Figure 4.3: Training-error in blue (batch-size 1) smoothed with moving average (window size 10). Test set-error (red) evaluated every 100th step until 25k iterations and every 200th step until the 61786th iteration. The baseline is the simple geometric model evaluated on the test-set.



(a) Predicted α **(b)** Predicted β

Figure 4.4: Stacked timelines of predicted α and β for the test set. Sequences ordered by sequence length. As α is the log-location it can be interpreted as a point estimate while β is roughly the inverse log-scale so higher β is an indicator of confidence. We see that as we get closer to failure the predicted failure time drops and the confidence gets higher.

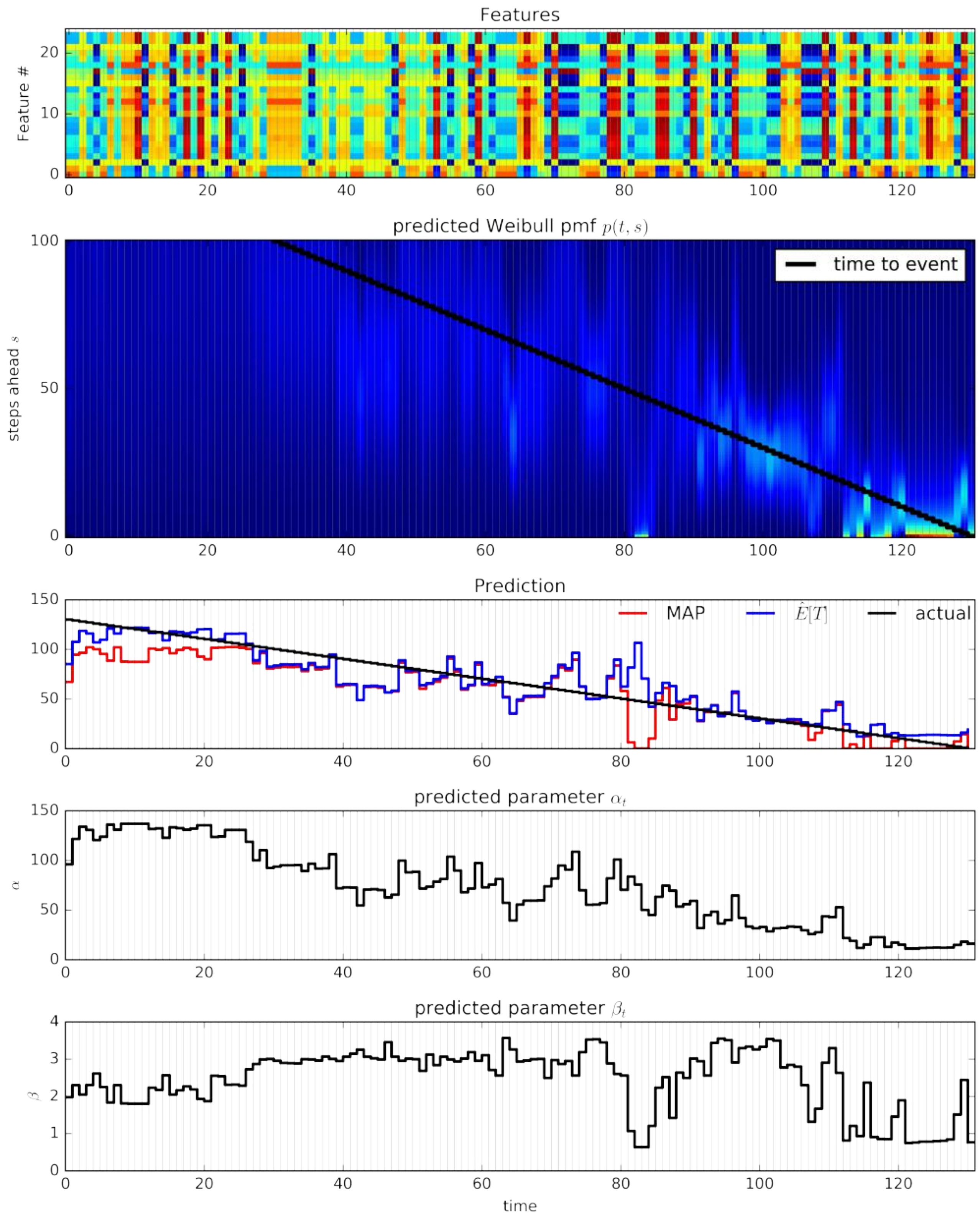
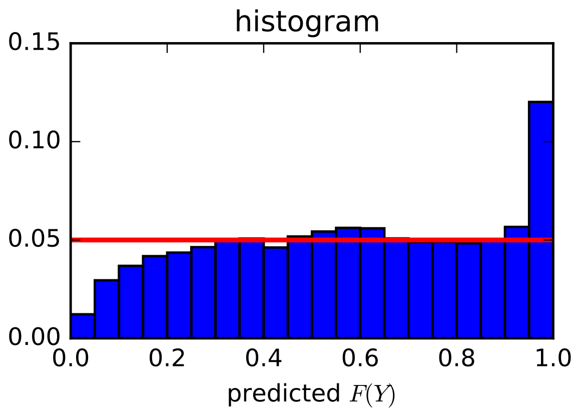


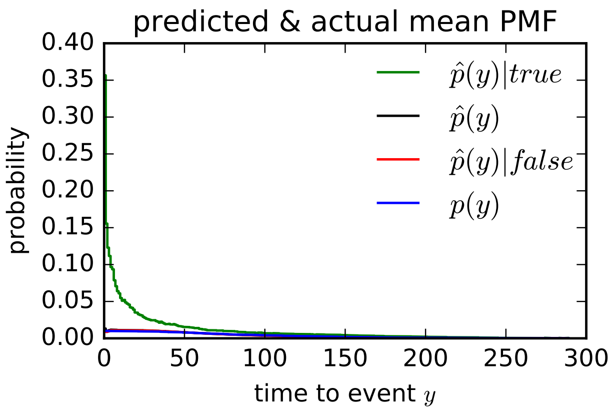
Figure 4.5: Example of a predicted sequence in the test set. Features are the 26 sensor measurements in each timestep. The pmf shows the predicted probability of the location of the time to event as a heatmap. Predicted MAP and expected value vs the actual time to event is highlighted below.

4.2.2 Result : Promising performance

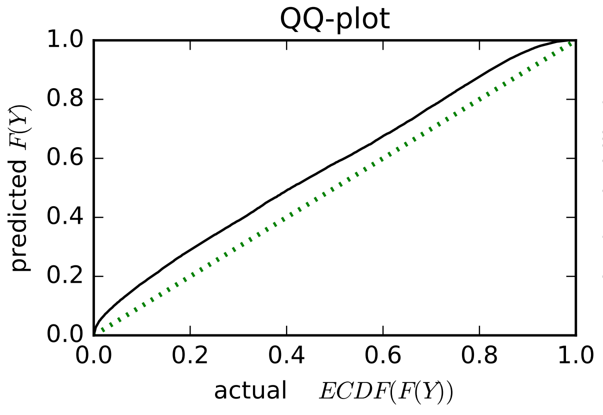
A big strength with the Weibull-model over models doing naked-point estimates is its probabilistic interpretation. To safely use these probabilities one needs good calibration i.e that the predicted number of events should happen within the predicted time. This seems roughly to be the case looking at figure 4.6.



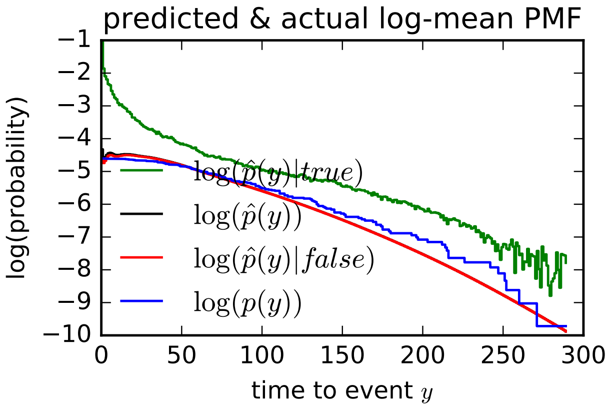
(a)



(b)



(c)



(d)

Figure 4.6: Calibration of predicted probabilities in the test set. For a continuous random variable Y we expect $F(Y) \sim Uniform$ so the histogram (a) of the predicted CMF F evaluated at observed time to event $F(Y)$ should be approximately flat. The model is skinny tailed also implied by QQ-plot (b). By calculating the mean of the predicted pmfs $\hat{p}(y)$ and varying y we can compare the observed (blue) number of observed time to events at $Y = y$ to the predicted overall (black) and among the observations at $Y = y$ (green) and those not (red).

Here we updated the model using mean loss per sequence meaning that each sequence have the same influence on the loss function regardless of its length as discussed in section 3.3.2. In figure 4.6 we show performance in terms of unweighted prediction, so each sequence step contributes the same meaning that longer sequences have more influence. As the models aren't trained on this metric we expected much poorer performance than what we found so this is a good result.

There are of course some problems and we should therefore to be careful when interpreting the probabilities. In figure 4.6a it's apparent that the model is skinny tailed (left skewed) i.e putting to low probability on high times to event. We can see this highlighted in figure 4.6b and 4.6d. The mean predicted PMF (black line covered by the red line) is below the observed number of time to events (blue) as it gets higher. The fact that the probability for observations observed at y (green line) is far above those not (red line) is an indicator of both good calibration and performance.

In the original PHM08-conference challenge a predefined score-function was used to compare algorithm performance which penalized late predictions with the argument that an unexpected breakdown is worse than unnecessary maintenance [33]. This was calculated by taking the mean of

$$s = \begin{cases} e^{\frac{y-\hat{y}}{13}} - 1 & \hat{y} < y \\ e^{\frac{\hat{y}-y}{10}} - 1 & \hat{y} > y \end{cases} \tag{4.1}$$

Since we split the dataset in a different way than in the original PHM08 challenge the scores are not directly comparable but the top 20 researchers on the leader board had scores ranging from 436 to 2400. The two top-scoring contestants used intricate proprietary algorithms that trained Recurrent Neural Networks after preprocessing data with Kalman-Filters [35]. With our approach the only preprocessing done was training-set normalization. When using α (roughly the 63d predicted percentile) as a point-predictor (shown in figure 4.7) we got a mean test-set score of 275.3. Using the 67th predicted percentile lead to the lowest predicted score (244.5).

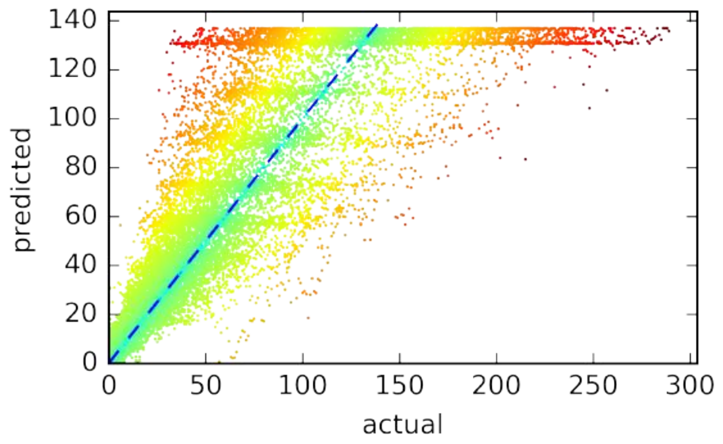
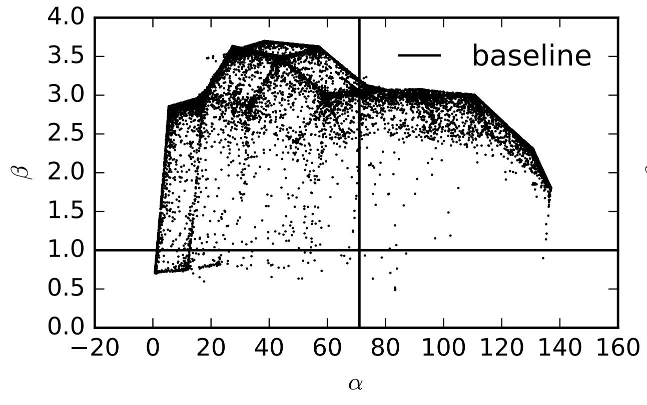
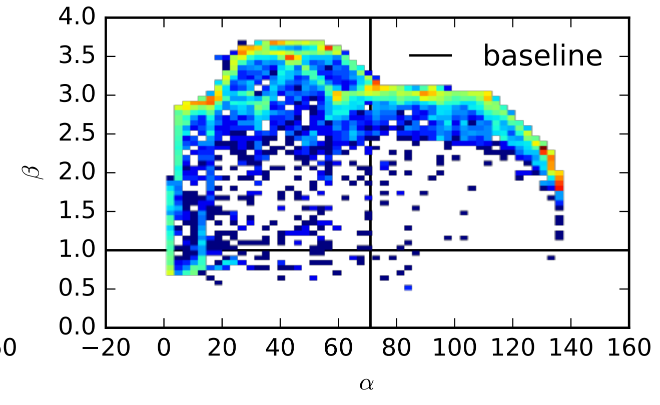


Figure 4.7: Predicted vs actual time to event in the test set colored by the PHM-challenge score function. Blue line marks the sought region predicted=actual. Here we use α as the point estimator reaching a mean score of 275.3.

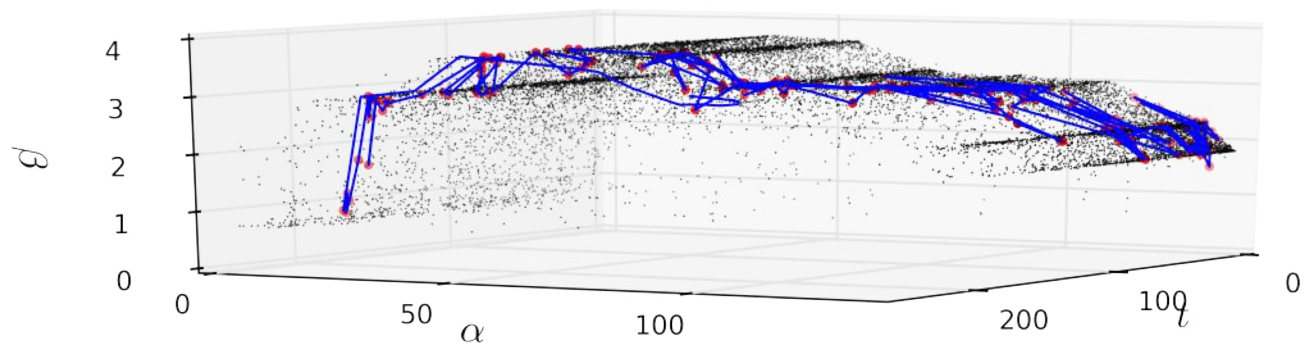
Besides using the parameters for point estimation, analyzing them could possibly be a useful way of mapping sequences to a two-dimensional plane as suggested in section 3.3.1. When plotting the predicted α and β as shown in figure 4.8 we uncover some kind of hidden graph-like pattern. When plotting one sequence at a time it's apparent that each sequence traverses this graph on its way to failure.



(a) scatterplot of β vs α



(b) 2d histogram showing density



(c) 3d-scatter with highlighted trajectory (right to left) of the sequence shown in figure 4.5.

Figure 4.8: Showing the hidden graph-like structure in the distribution of predicted alpha and beta. In (a) and (b) we highlight the baseline parameters for reference.

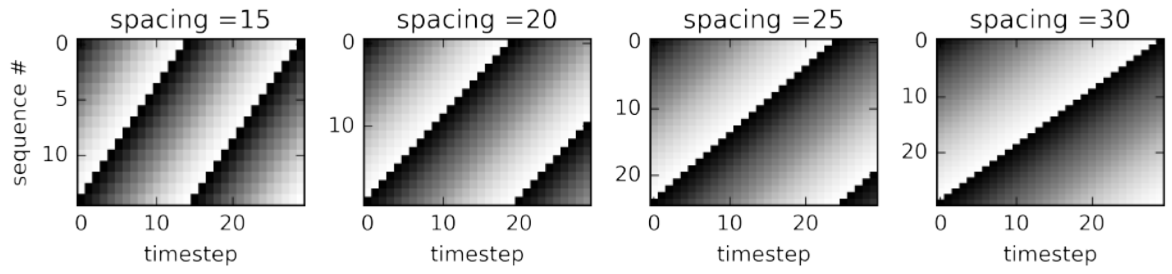
In an applied setting analyzing and clustering sequences based on which region they are in the $\alpha - \beta$-grid could be useful way of getting actionable rules for when to do maintenance.

4.3 Naive strategies with censored data

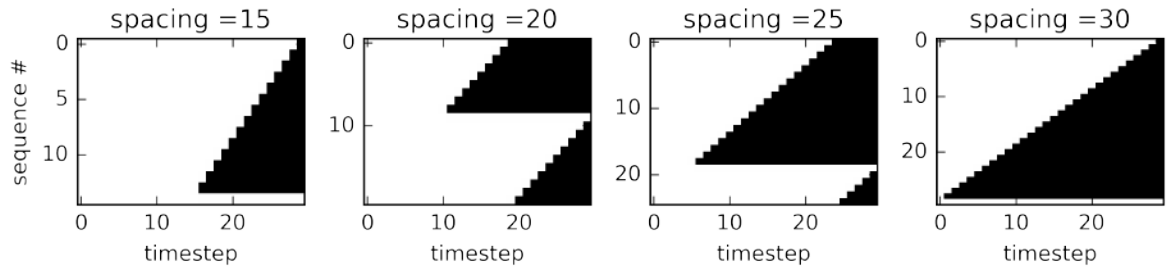
The purpose of this experiment was to compare performance of the WTTE-RNN with different strategies of handling censored data. We call these baselines *naive* because they are the most obvious and off-the-shelf way of trying to avoid making assumptions about the censoring process and thus use the censored log-likelihood loss functions. As we will show, all these methods lead to severe bias and deteriorating performance on test set data compared the proposed method.

4.3.1 Setup : Predict evenly spaced events

Imagine that we have an infinite sequences of evenly spaced events, say with spacing d. If we place an observation window of width $n = 30$ over this sequence we can observe a total of d different sequences. The goal is to predict the number of steps to the next event without knowing which of these sequences we're shown i.e were in the the sequence we start out. The only feedback is whether an event occured in the previous step. Sequences that does not have an event at the last step will get censored observations from the last observed event until the end of the sequence. We used 4 different datasets created using spacing $d = 15, 20, 25, 30$ leading to less and less recurrent events and more and more censoring. When $d = 30$ only 1/30 sequences are uncensored at the last timestep.



(a) (Uncensored) Time to event



(b) Censoring indicator

Figure 4.9: Stacked timelines showing all sequences in the four datasets.

Since we know the ground truth in this data we can compare performance of different modeling strategies, all having identical architecture but with different loss functions:

Without censored data : Train on the ground truth target value. No points are censored

Use censored points : Train using log-likelihood for censored observations. We also tried this with penalized β for numerical stability.

Pretend: Train the model with the censored waiting times but with regular log-likelihood i.e as above but flag all waiting times as uncensored.

Discard: Simply remove the timesteps that are censored from the objective function.

We used LSTMs with 1 feature input, 4 hidden state nodes and 2 output layer nodes. The architectures was thus $1 \times 4 \times 2$. The output-layer biases were initialized s.t $\beta = 1$ and for α we used the mean of the ground truth waiting times to estimate the scale parameter of the assumed geometric distribution (geometric initialization using ground truth data, see definition 4.2). The reason for using ground truth data was that the alternative - inferring this parameter with censored waiting timess - would overestimate the biases more than we already would. This in turn leads to numerical hardships and differences between the strategies that was not the focus of this experiment. In a real world application the location of the mean of the waiting times can most likely be corrected using domain knowledge.

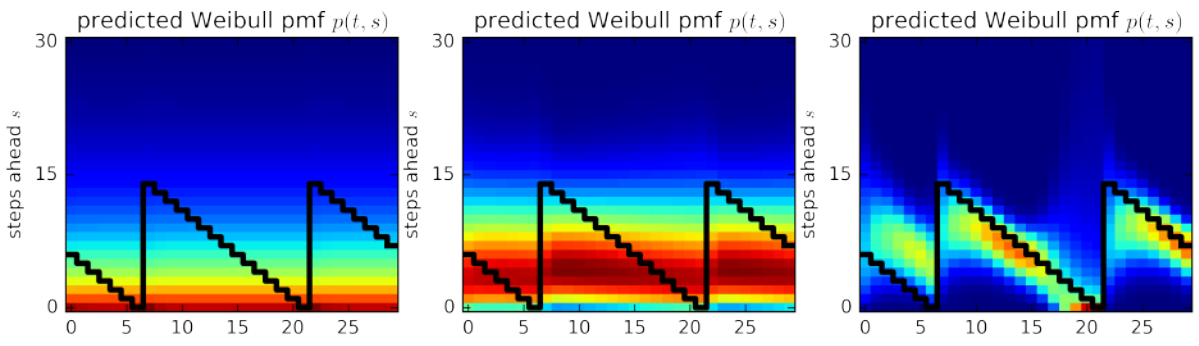
In each training step all d possible sequences was passed. The only random elements in training was the initialization which was controlled through iterating over the same 10 different seeds for each model. As feature data we fed a single binary variable *was_event* being 1 whenever the last step was an event else 0 and 0.5 in the first sequence timestep. Every 50th trainingstep we calculate the error using ground truth targets and save the mean of the predicted α's and β's.

We trained using RMSPropOptimizer with epsilon set to $1e{-}5$. Using optimizers with momentum such as Adamoptimizer led to more numerical instability. When getting close to (near attainable) perfect fit momentum-optimizers would start rushing, causing exploding gradients. In more random circumstances we found this to be a useful behavior but here we were more interested in the general trend over the 1000 iterations. Learning rate was set to $\frac{1}{100+0.4i}$ so it decreased from 0.01 to 0.002.

For the censored log-likelihood-model we tried both with and without regularization. In the regularized version we added a regularization term as discussed in section 3.2.2.3

$$penalty = -e^{\delta(\beta - \bar{\beta})} = -e^{1.25(\beta - 8)}$$

creating an artificial boundary after $\beta = 8$. This prevented β from becoming too high thus hindering overconfident predictions. This in turn led to numerical stability.



(a) iteration 1 (b) iteration 100 (c) iteration 200

Figure 4.10: Example prediction for the uncensored model during training.

4.3.2 Result : Censored & regularized strategy wins

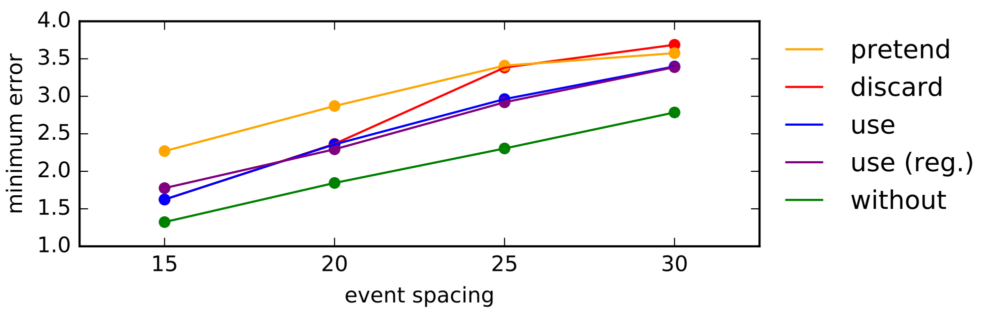
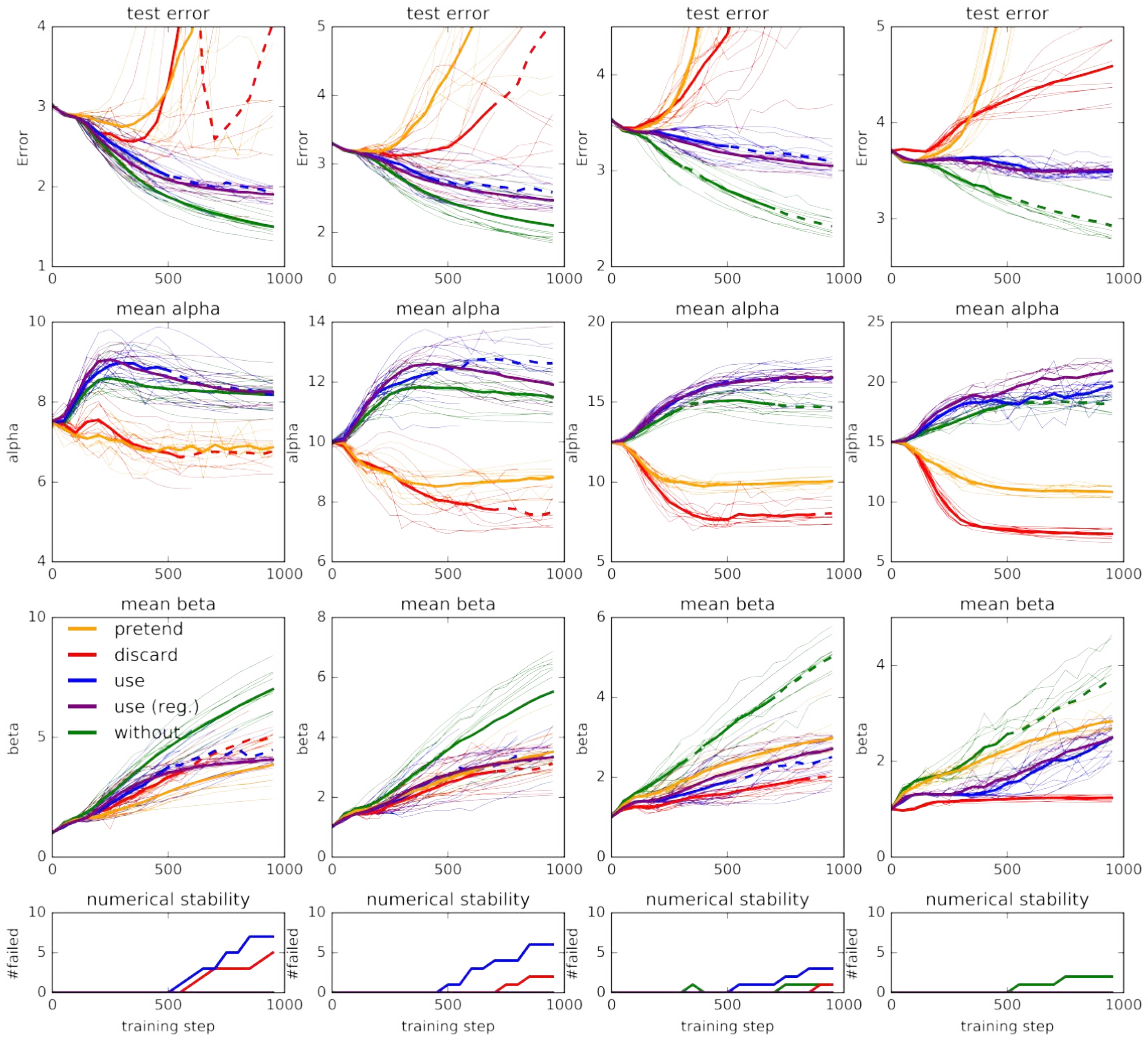


Figure 4.11: Minimum error per model for different datasets (event spacings)

With increased spacing there's more and more steps in the beginning where the model can't know the true value. Here it will learn to update its prediction as it gets more information in form of non-events. This means that larger event-spacing means more noise and larger total error. This also leads to more censoring.

In all the experiments *pretend* and *discard* had deteriorating performance on test data. As expected, the predicted α's drops too low for these models as they mistakenly underestimate by learning to predict the censored times instead of the time to events.

In figure 4.12 we see that some models tended to fail during training due to numerical overflow. This typically happens when β grows too big too fast. We stopped training whenever any of the predicted parameters would be NaN. This was particularly common for the censored model without regularization. The regularized version had a 100% survival rate throughout training. Even though we didn't train until convergence here we found no significant loss in performance.



(a) $d = 15$ **(b)** $d = 20$ **(c)** $d = 25$ **(d)** $d = 30$

Figure 4.12: Test error and mean predicted α and β for the 10 runs of every model and different spacing d. Thin line marks the real trajectory and thick lines the mean over models. When any model experienced numerical failure the mean was taken over the 'survivors', marked with dashed lines. The most viable strategy seemed to be using regularized censored log-likelihood (purple) when ground-truth data is unavailable (green).

4.4 Comparison to the sliding-box-model

In this experiment we compared the Weibull model to the the standard modeling approach, what we call the *sliding box model.* The purpose was to compare performance in a realistic use case. See section 2.4.1 for a description of this model.

The relevance of this baseline solely depends on its wide use. Comparison is of course problematic as the models predicts different things and optimized for different tasks. The biggest theoretical strength of our model is its promise of giving a sense of magnitude of the time to the next event. Note that if predicted box probability θ_t from the sliding box model is small it only tells us that the probability of an event within τ days is small. Interpreting this as if the probability of an event in $\tau + 1$ is small too is an extrapolation that we do not explicitly optimize for in the sliding-box model. Making actionable decisions on this model could therefore be problematic and the inference questions need to be explicitly formulated before training.

But since we *can* compare the models we will. We will compare the models on how much mass the WTTE-RNN assigns before τ vs θ_t. Formally, let $F(t)$ be the predicted Weibull CDF, then the probability of at least one event inside the box is

$$\hat{\theta} = F(\tau)$$

Using this estimation we have

$$\begin{aligned}
\hat{\mathfrak{L}} &= \hat{\theta}^v \cdot (1 - \hat{\theta})^{1-v} \\
&= F(\tau)^v \cdot S(\tau)^{1-v} \\
&= (1 - e^{-\Lambda(\tau)})^v \cdot e^{-(1-v)\cdot\Lambda(\tau)} \\
&= (e^{\Lambda(\tau)} - 1)^v \cdot e^{-\Lambda(\tau)}
\end{aligned}$$

Note how this is identical to the discrete likelihood in equation 2.14 where we've discretized a continuous variable using step-length τ and observe it only when $W_d = 0$ and know that it's censored otherwise. By the scale invariance- property of the Weibull we may also see it as simply a Weibull with $\alpha \propto \tau$. I.e. the sliding-box model is a censored discrete Weibull with poor time-resolution w.r.t the fixed censoring variable τ. The theoretical conclusion should be that the box-model is expected to be as good as this special case of the Weibull model. To have an identical comparison we could use this fact and corrupt the input data for the Weibull model leading to this special case. This wouldn't teach us much as they'd be almost identical apart from the output layer. Instead we trained the Weibull model just like we would do normally, teaching it to learn the distribution over all possible values $0, 1, \ldots$ of the target value and then compare predictions.

4.4.1 Setup : Generate random sequences of random length

We're trying to simulate a situation were we have recorded event-sequences of varying lengths, for example when we have customers that entered our dataset at different times. Here we let sequences vary randomly between 10 and 100 steps. We can imagine this as if a company started 100 days (or months, years) ago and that we only consider customers that been around for more than 10 days. We simulated data by using a randomly generated hazard function:

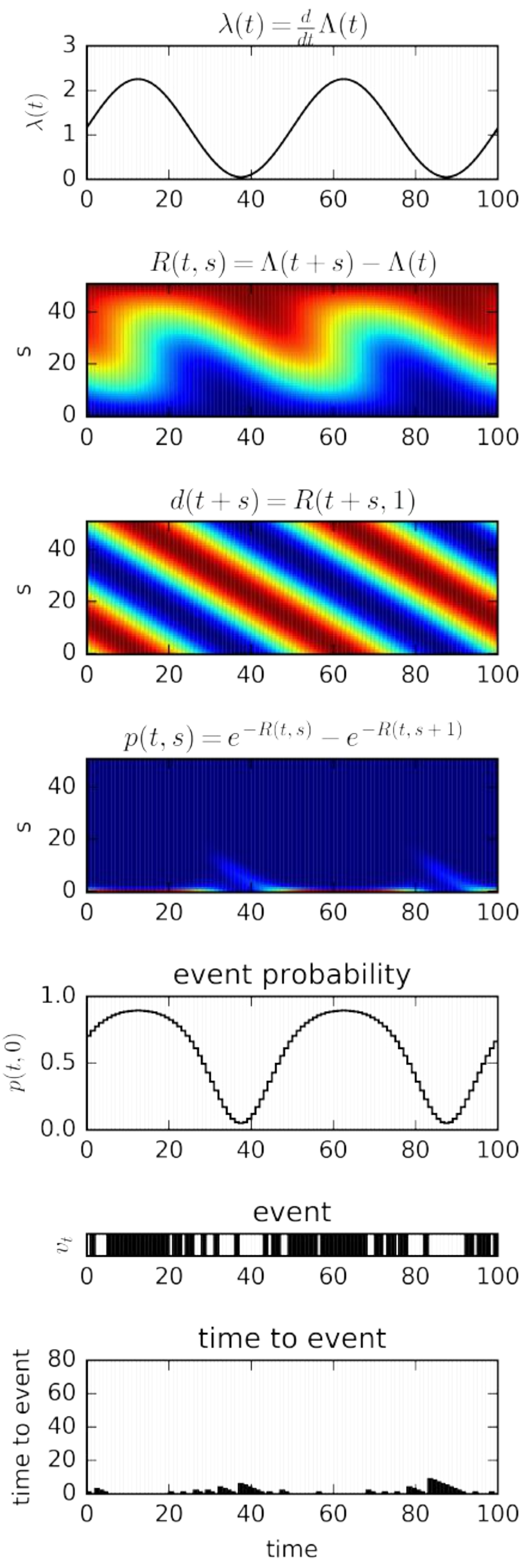
$$\begin{aligned}
\lambda_j(t) &= c_0 + c_1(1+\cos(2\pi U_j + c_2 t)) \\
\Lambda_j(t+s) - \Lambda_j(t) &= (c_0+c_1)\cdot s + \frac{c_1}{c_2}\left(\sin(2\pi U_j + c_2(t+s)) - \sin(2\pi U_j + c_2 t)\right) \\
U_j &\sim \text{Uniform}[0,1]
\end{aligned}$$

With two different settings for the constants: *high* and *low* probability of event.

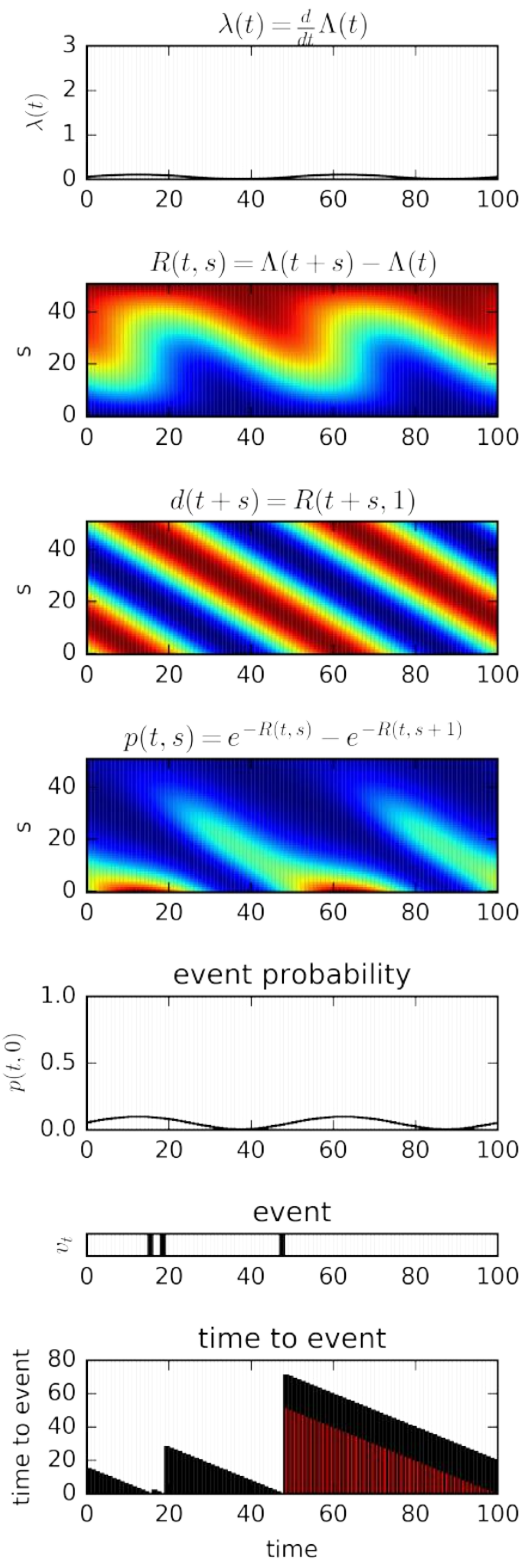
To generate the data we used the technique of section 2.2 which can be summarized as discretely generating events for each unit timestep $t = 1, \dots, n$ in each sequence j with probability $1 - e^{-d_j(t)}$. The transformations involved is shown in figure 4.13.

The settings was chosen to simulate two types of signals. One is the case when the signal from the underlying process is strong and the event probability is high. We call this the *high*-setting. In the other we have more noise and lower event probability. This is called the *low*-setting. The high-setting leads to a low resolution discrete distribution, i.e small predicted α for the Weibull model and low levels of censoring. The low-setting leads to noisy long-term dependencies and lots of censoring (see (red) censored time to event of example sequence in figure 4.14b).

The sliding-box model described in section 2.4.1 transforms the problem into a binary prediction problem by discretely predicting whether an event will take place within a preset τ steps. We refer to this happening as an *event in box*. For the *high* and *low* setting we chose values of τ so that this probability was low, medium and high. Vertical lines in the histograms of figure 4.14 illustrates where the box-widths τ lies in the respective raw time-to-event distributions.

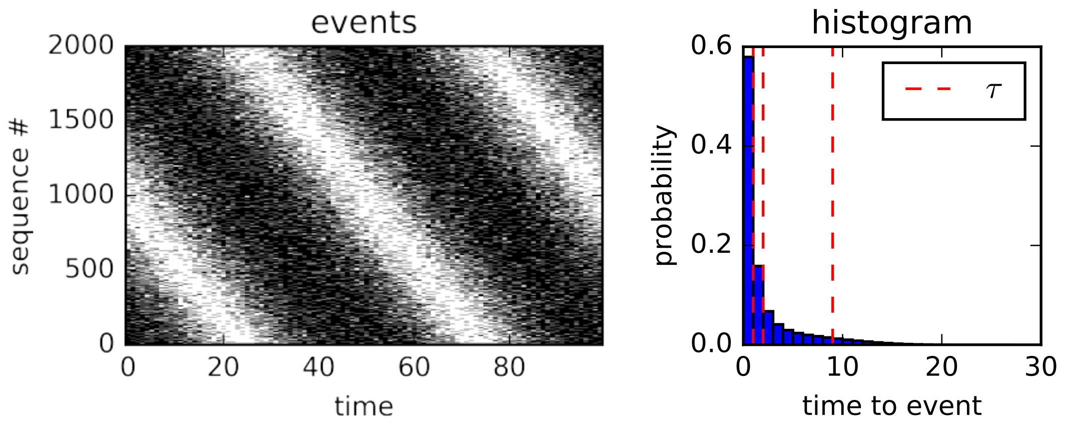


(a) *'high' : high signal/low censoring*

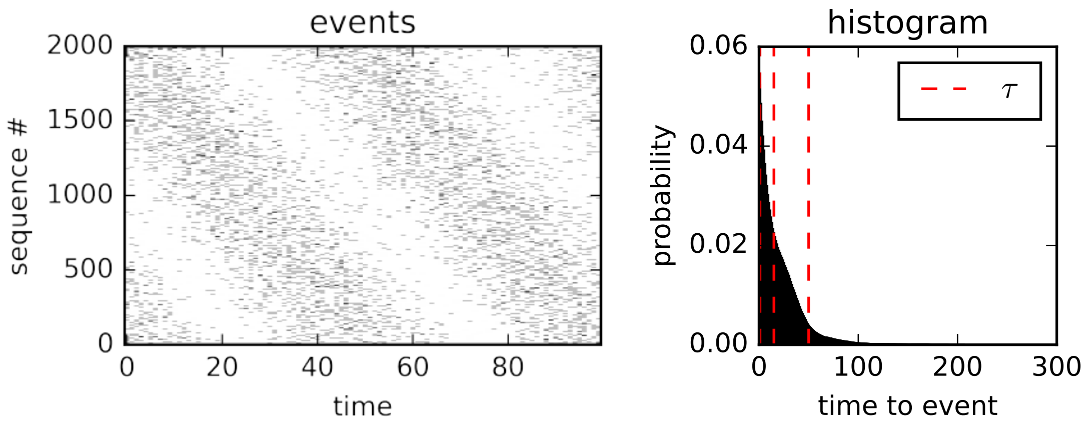


(b) *'low' : low signal/high censoring*

Figure 4.13: Realization of data generating process : Hazard function λ is a randomly shifted sinusoid. Integrated over unit intervals it forms the CEPMF $p(t,s)$. Events generated as $V_t = (U_t \leq p(t,0))$ with $U \sim$ Uniform. Censored time to events in red.



(a) Setting : *high*



(b) Setting : *low*

Figure 4.14: Example of batch of 2000 generated sequences of length 100 for the high- and low- setting vertically sorted by their randomly shifted period. Histogram shows the distribution of real time to events calculated from each of the 500×100 timesteps with the box-width τ superimposed. During training we randomly chose the sequence lengths $n \in [10, 100]$ which can be seen as masking all but the first n timesteps and calculating the censored time to event with step n being the horizon. Models were evaluated on the full 100 step sequences after calculating the true time to event.

4.4.1.1 Training

To get some reference how well the WTTE-RNN worked we also trained a Weibull baseline model (constant model), a Weibull-ANN, a Weibull LSTM and a sliding-box LSTM. The latter is the WTTE-RNN and the focus of the study. Feature input was a shifted event-indicator *was_event*, i.e $x_{t+1} = 1$ if an event happened in the previous step i.e $v_t = 1$.

The architecture of the models is summarized below.

Weibull Baseline	(W-bl)	0×2	softplus
Weibull ANN	(W-ann)	1×2	softplus
Weibul LSTM	(W-lstm)	$1 \times 30 \times 2$	tanh $\times$ softplus
Sliding Box LSTM	(box-lstm)	$1 \times 30 \times 2 \times 1$	tanh $\times$ tanh $\times$ sigmoid

Note that the box-lstm had 1 extra layer compared to the W-lstm.

The models were trained for 1500 training steps were all models were fed the same data, randomly generated in each step. This was repeated 5 times for all combinations of settings and box-width i.e $5 \cdot 2 \cdot 3$ times. Weibull models were not in any way affected by the box-width τ apart from fixing the random seeds so that the Sliding Box- and Weibull models that were compared was also trained on the same data.

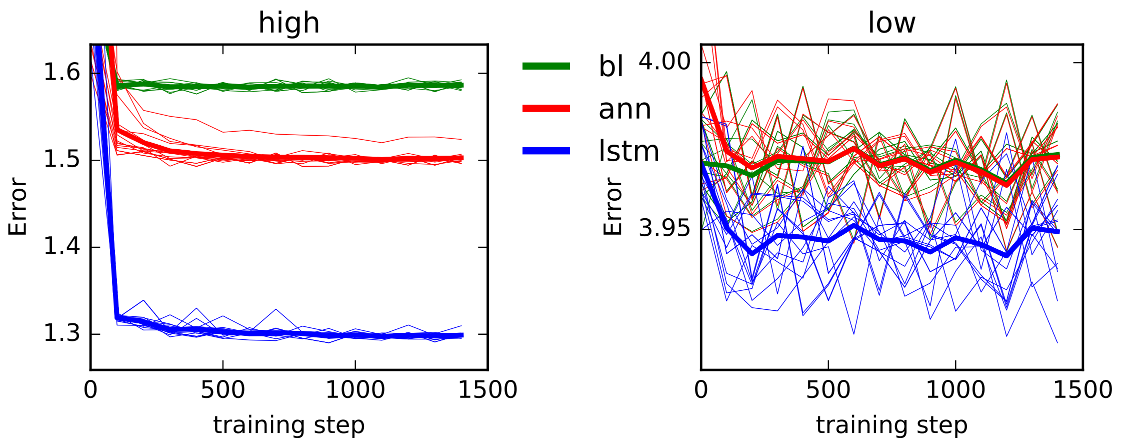
With a sliding box model of width τ, given sequence length n we would in a real situation only be able to train the model on the first $n - \tau$ steps of the recorded sequences since we wouldn't know how to define target-value for the τ last steps. With the Weibull model we can train on all n steps but censored datapoints will be frequent towards the rightmost boundary. Weibull models were trained using discrete Weibull log-likelihood for censored data. Sliding box-model was trained using cross-entropy loss (Bernoulli-log likelihood) optimized for sigmoid output layer. Each training step can be summarized as:

1. Randomly pick a sequence length $n \in [10, 100]$
2. Generate *batch_size* (censored) sequences of length n.
    - Update box model using the $n - \tau$ first generated sequence steps
    - Update Weibull using all sequence steps.

batch_size was set to 200 for the high- and 500 for the low- setting. We initialized the output-layer biases for each model and seed using data from a first training-batch. We used geometric-distribution MLE for censored data to initialize the biases of the Weibull model. The sliding-box model bias was initialized s.t given centered input it would output the baseline probability of event in box.

4.4.1.2 Evaluation

Every 100th training step 2000 random sequences (and uncensored waiting time data) of length 100 was generated. Weibull models were evaluated on uncensored data to see training progress as seen in figure 4.15. Sliding box and and Weibull-models prediction of probability in box was evaluated similarly. The sliding box prediction in each sequencestep is compared with the Weibull-models (trained and evaluated using the same seed) predicted in-box-probability $\theta_t^{\mathrm{W}} = F(\tau)$.



(a) Setting : *high* **(b)** Setting : *low*

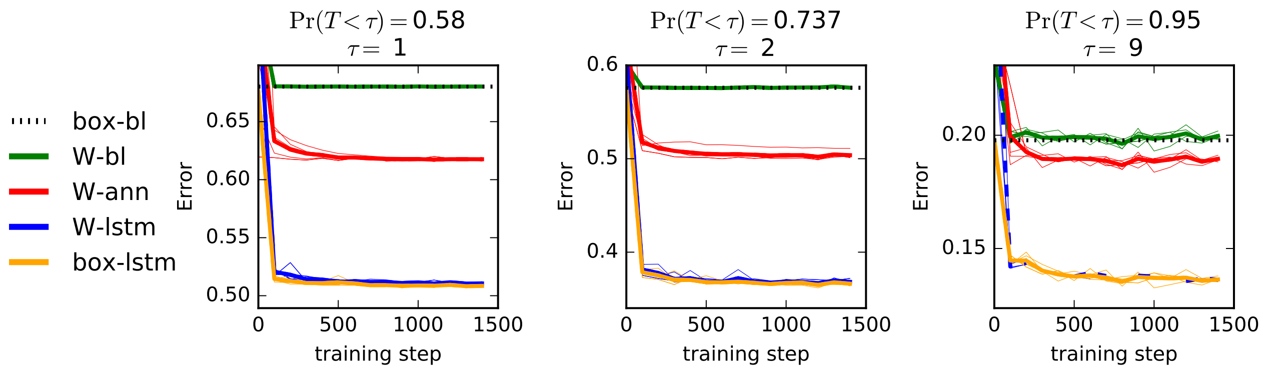
Figure 4.15: Weibull evaluation error for the models. Thin lines marks each seed while thick lines marks mean over seeds. The more noisy 'low'-setting lead to more noisy training even though learning rate was low and diminishing and batch size was high.

4.4.2 Results : Sliding box wins a rigged game

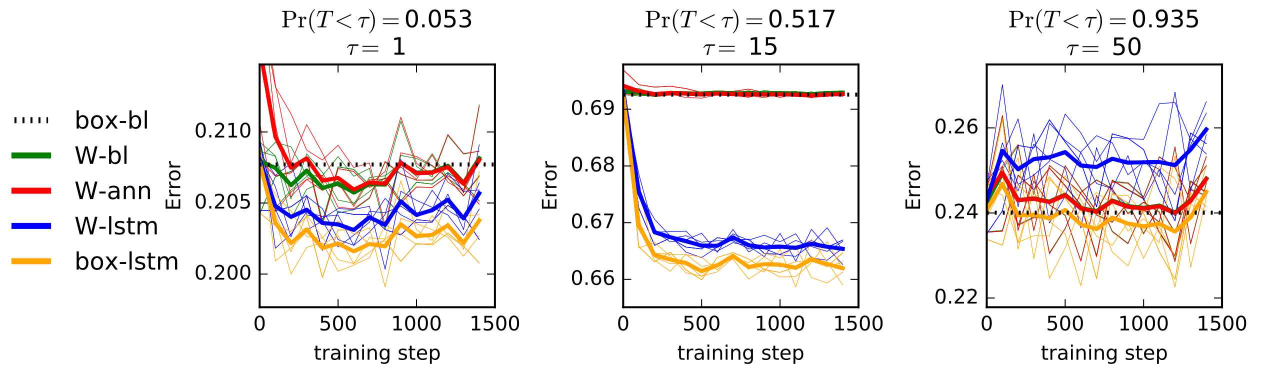
In figure 4.16a and 4.16b we show the comparison in performance with the box-model. As expected, the sliding box model outperforms the Weibull model on nearly all the tasks. In nearly all the tasks the W-lstm beats the baseline and the ANN. Evaluated using the Weibull-log likelihood with uncensored data the Weibull LSTM beat both the baseline and the ANN both in terms of mean evaluation error during training (highlighted in figure 4.15) and in minimum error during training.

High-setting With the *high*-setting, were we had very little noise and a clear signal to pick up, the race was close (figure 4.16a). When $\tau = 1$ it's the same as binary predicting event or not in next step. Baseline probability of this was high already (0.582) so this is a comparison roughly to the area of the predicted Weibull pdf before the median. When raising τ W-lstm must have been very confident as we get some numerical errors (see $\tau = 9$ of figure 4.16a) which implies that the predicted probabilities approached 0 or 1. The few datpoints we see shows a training-trajectory which is on level with the box-model.

Low-setting With the *low* setting the box-model had a mean-error obviously below the Weibull models for all settings of τ. For the extreme cases of $\tau = 50$, $\Pr(T \leq \tau) = 0.925$ the Weibull baseline- and ANN even outperformed the W-lstm (see figure 4.16a- 4.16b). How did this happen?



(a) Setting : *high*



(b) Setting : *low*

Figure 4.16: Box evaluation error during training. Thin lines marks independent runs, thick lines marks mean over the 5 runs. Dashed lines means that some of the output contained numerical failures for some run typically by giving predictions to close to 0 or 1. By $\Pr(T \leq \tau)$ we mean the baseline probability of event in box.

A tempting explanation would be that it's due to the Weibull model being poorly specified in the right tail, but as we had no such problems for the *high* setting at $\tau = 9$ it doesn't cut it. As the non-temporal ANN and baseline didn't have problems for the low-setting it's a hint that the LSTM learned to recognize that it approaches a censoring point. This behavior would be very problematic. We saw this behavior when lowering the signal even further and letting the model train for 100k iterations. This would lead the model to 'learn artifacts' by recognizing the censoring point. The effect is that α gets predicted higher and higher for the later part of the sequence eventually causing test error to deteriorate (overfitting).

There's some problem with this explanation on this data using $\tau = 50$. When only looking at the box error in the first part of the sequence (first τ steps) we found that the W-LSTM box-error was still higher than the other models (but not as much).

The best explanation is therefore that the heavy censoring shifted the distribution to the right which had a big impact on the right tail probabilities causing poor calibration for this task. Note that we did not have particularly poor calibration for other values of τ at the high setting as the W-lstm performed relatively well there. Taking into account that the W-lstm performed better than the ANN and baseline evaluated using Weibull-loss and uncensored data (4.15b) we conclude that during optimization the gain being good in the left tail must have outweighed the cost of error in the right tail.

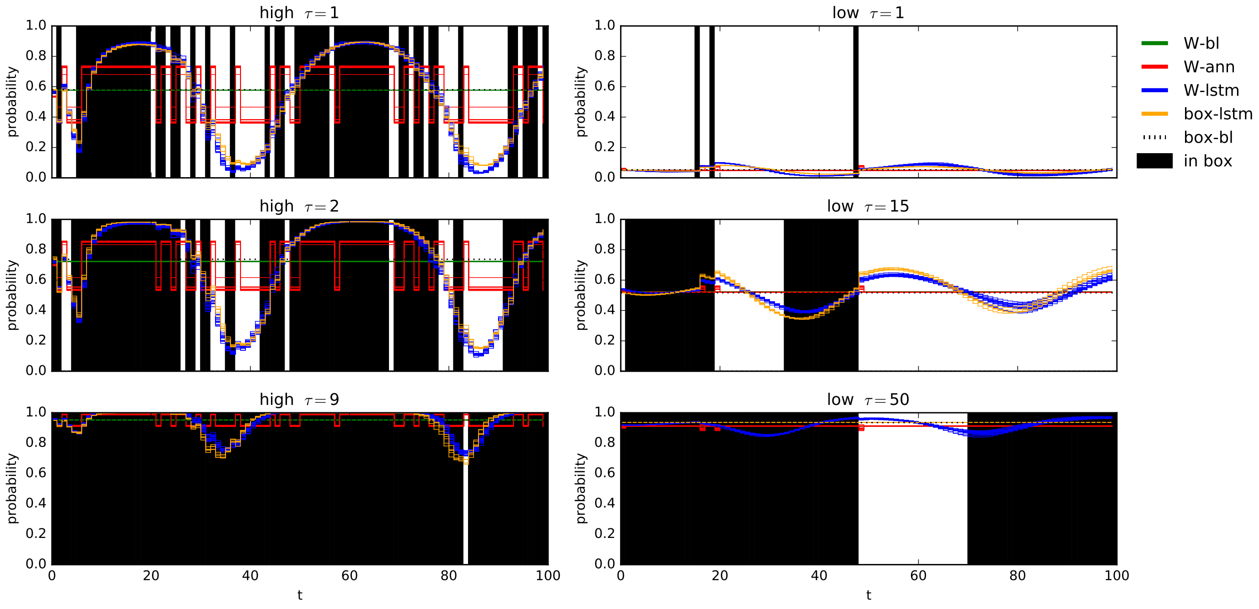


Figure 4.17: The target indicator overlaid with the predicted probability for each of the settings. Note that when $\tau = 1$ it coincides with the event-indicator. It's easy to see that feature input for the models were lagged event-indicators.

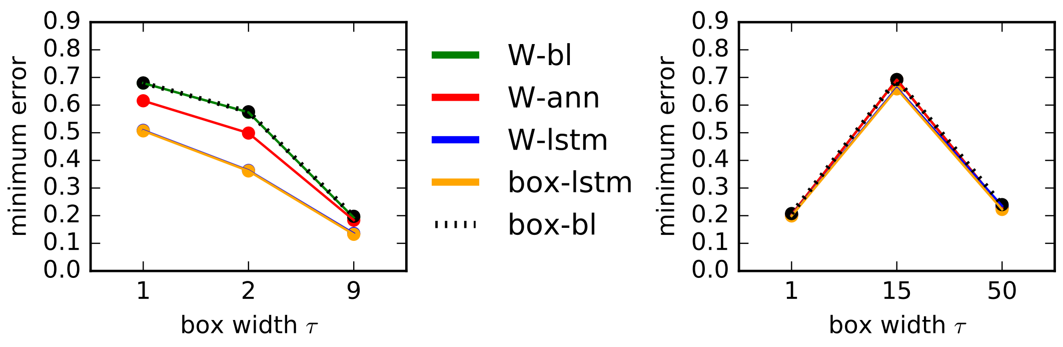
Analysis of predictions The box-errors (Bernoulli-log-likelihood) are mainly reliant on the calibration of the predicted probabilities. This explains some of the box-lstms lead. If we look at the actual predictions we see another image appear. We generated a sequence of events for both settings and made predictions for all the models at the end of training. The result is shown in figure 4.17.

Looking at the predicted baselines (green and dotted) we see a perfect overlap for all values of τ. This essentially says that the predicted number of events within τ steps using the Weibull-models roughly corresponds to the observed. In other words Weibull models (in general) are fairly calibrated and the censoring doesn't seem to have shifted the distribution in any notable way.

For the most problematic task, the low-setting with $\tau = 50$ (bottom right) the box-lstm doesn't seem to have learned anything. The W-lstm on the other hand clearly reacts to input data and temporal patterns in what looks like a reasonable way. What seems to be wrong is the location (i.e calibration) of the predicted probabilities not the relative size of them.

This gives some basis to believe the Weibull model could perform better than the Box-model after proper calibration when we have very rare events. The first reason is that the Weibull model gets to see longer sequences as it can use all training data. The second is that that the Weibull model will get continuous feedback in each step unlike the box-model which only gets binary feedback. We did not evaluate the models performance on scale-invariant measures such as AUC. A hypothesis is that the Weibull models would outperform the Box-model but verifying this is proposed as future work.

The remaining question is why the Weibull model gets a higher error than the box-model in all tasks, as seen in figure 4.16. The first explanation is that the Weibull model is not trained specifically for any of the tasks, it's trained for all of them. The box-model is only trained for one specific τ at a time so theoretically this would always win, assuming that it could find a minimum. So as stated before it's a rigged competition favoring the box-model. The second answer is that the difference was remarkably small. When taking the minimum achieved score during training (figure 4.18) we can barely tell the difference (but it's there). See figure 4.18.



(a) Setting : *high* **(b)** Setting : *low*

	High			Low		
τ	1	2	9	1	15	50
W-bl	0.68	0.574	0.193	0.203	0.692	0.229
W-ann	0.616	0.499	0.184	0.203	0.692	0.229
W-lstm	0.509	0.364	0.136	0.2	0.663	0.235
box-lstm	0.507	0.362	0.133	0.199	0.659	0.223

Figure 4.18: Minimum box-error achieved per model, setting and τ. The box-lstm is right below the weibull lstm.

To summarize this experiment we conclude that the sliding box will probably outperform the W-lstm on predefined tasks but after proper recalibration it's possible that the W-lstm could be better than the sliding box when we have long temporal patterns and rare events. In all circumstances we conclude that if we do want to build a sliding box model, it's reasonable to first train a Weibull-model and play around with it to figure out a reasonable hyperparameter τ.

5 Discussion

We framed the problem as if we from a given point in time want to predict where the *next* event might be, or equivalently the value of the current *time to event*.

Instead of making a point-estimate of where the current time to event is, we use a probability distribution (think about *kernels*) that predicts the *region* where the time of this event is likely to be. One may think of it as a heatmap were we use a function to assign more heat to where we believe the next value is. By choosing a basic shape of how this region may look, we can extrapolate to regions which we don't have any data for i.e very large waiting times. This means that we can choose an ordering over where we think it's more likely to see an event and rank predictions for new observations. By choosing this heat function to be positive and integrate to 1 we get a sense of magnitude. Instead of just being able to order the observations, the heat-level or density gets a fixed probabilistic interpretation. By the fact that a distribution sums to 1, a non-observation, i.e knowing that an observation was *not* in a region is information about where it must have been. This makes our model a type of latent variable model where we're trying to predict the distribution generating waiting times. By choosing a known probability distribution the quantities and computations associated with it are easily interpreted and communicated. We chose the discrete and continuous Weibull distribution as a model for this region.

We've shown that the Weibull-distribution can take many shapes by adjusting its two parameters. This makes it expressive enough for many applications. We have shown that the shape of the distribution can be controlled to avoid over-fitting by penalizing to large β's. In experiments this also proved to lead to more numerical stability. As a model for inference we've given theoretical and empirical justifications through the Weakest-link property (Fisher-Tippett-Gnedenko theorem 2.33) why the time to the next event tends to have a distribution similar to the Weibull. We may liken this to the fact that the shape of the Gaussian distribution makes it expressive enough to be useful as a kernel with the added benefit that this shape frequently occurs in nature (explained by the central limit theorem).

To estimate parameters we described the log-likelihood loss function used in survival analysis. We discussed necessary assumptions to use it. We argue that the likelihood for censored data should be used with less anxiety because in experiments we've seen that it works well in practice even when all assumptions are not met. This can be likened to the fact that in many machine-learning applications we use for example Gaussian kernels and optimize different likelihood functions without worrying too much about probabilistic assumptions. The most convincing argument so far is the fact that it works in practice.

By discretizing the continuous distribution we get a discrete time model with the same expressiveness and shape, and we may extend much of what we know about the continuous model to the discrete case. One can argue that if the process generating events is truly continuous then the Weakest Link Property is an argument to assume an underlying continuous Weibull distribution for the time to the next event. If the continuous random variable is Weibull it follows that the discretized random variable is a discrete Weibull.

From experiments we've seen that the resolution of the waiting time has a large effect on how robust the model is to levels of censoring. If the distribution is truly continuous we would recover the correct parameters for almost any level of censoring. In practice we will almost always have some type of discreteness in the data. In these scenarios using the discrete likelihood is more justified.

We've also given a general framework for going from continuous to discrete models and back again by using continuous distributions to generate discrete events and by using smaller and smaller intervals we again get back to the continuous distribution. The seamless switching between discrete- and continuous can be a very helpful feature to speed up the initial process of testing out a model when using different event- and feature-definitions and time-resolutions. The model can seamlessly be extended to using other types of distributions by switching out the hazard-functions. It already has some known distributions built in as special cases. By fixing $\beta = 1$ we have a regular exponential RNN in the continuous case and a geometric RNN in the discrete case. In the continuous case this would be called a Proportional Hazards RNN. By fixing β to any scalar value we have an Accelerated Failure Time RNN. By fixing $\beta = 2$ we could also call it a continuous- or discrete Rayleigh-RNN or Rayleigh AFT-RNN.

In the results we saw that the model is - surprisingly - very close in performance to a sliding box model, defined on a metric that is not optimized for. It's clear that the Weibull model answers more questions and is much more actionable than the sliding-box model as we don't need to specify a threshold beforehand. If this is not an issue we found that the Sliding box is the better alternative as it achieved lower errors on test data. We did find some small empirical evidence that the Weibull model might be better if the box-width τ is very large even though the predictions had some problems with calibration. This is motivated by the fact that the Weibull model can be trained on more data and that it gets continuous feedback during training on how far off it is in the prediction. The sliding box will by its very nature only get binary feedback.

The point with this thesis is to describe a model that's useful for applications. We've identified and described the problem of churn-prediction and failure time prediction as natural usecases. We've discussed how one can use the β's and α's directly to spot different tendencies, like high immediate risk, peaks of activity and more. When using this at a large scale - with many sequences like customer or health data - we could have a powerful tool to effectively monitor and identify changing behavior indicating a lost interest or declining health.

To implement this model one needs first to decide how to aggregate features, what constitutes an event, which time resolution to chose for prediction and event-aggregation and whether to use discrete or continuous time. The answers to these questions are domain-specific with no definite answers. Once we have sequences of features $x^n_{0:t}$, censoring indicators $u^n_{0:t}$ and time-to-events y^n_t for each sequence n the practicalities moves to the domain of choosing a good RNN-architecture. The only constraints on the RNN we need to know is that we need a step-to-step architecture with an output-vector of two components that are strictly positive and unbounded. We may then train the network just like any RNN but with the Weibull-log likelihood loss-function. The details of training RNNs was deemed outside the scope of the thesis but as we propose modifying the output layer and the loss function, we will summarize some of the main points.

5.1 Lessons from training and implementation

To learn with the continuous- and discrete Weibull loss we need to be vary of initialization of output layer bias. This is no different from initialization of other types of neural networks but as the Weibull loss involves exponentials and logarithms we may avoid numerical difficulties with a few simple steps. One technique could be to normalize the magnitude of the target value y_t, leading to low magnitude bias and weights for the output layer for scale parameter α. The reason for not adopting this this approach in the experiments was that it added another step in order to do inference. Instead we used unnormalized waiting times for all the experiments which in retrospect led to more hazzle than proper normalization might have done. Here we found that initializing the bias of the output layer for the respective parameters s.t they initially operate below their predicted mean values was safer than initializing above. As a rule of thumb, setting $\beta \approx 1$ and α to the mean of the censored waiting times worked well for high resolution time to event. With low resolution discrete waiting times the geometric MLE is more reasonable. Using the MLE for censored waiting times typically leads to overestimation of the α-bias. In both censored and uncensored discrete- and continuous case the MLE is a function of the mean waiting time. We therefore recommend using domain knowledge to set this as a reasonable initial value.

We didn't test the effect of using the continuous log loss with discrete values but one may assume that it can lead to numerical difficulties and have biasing effect if we really have discrete values. We've shown that the discrete Weibull model works well with high resolution problems. Whenever $\alpha >> 200$ the chance of numerical failures would grow but it seemed like the predicted β was the main culprit. Forcing $\beta < 10$ through regularization was an efficient remedy.

5.2 Future work

Finding a TTE-process motivating the log-loss One of the hardest problems that we've faced was the problem of trying to define a generative framework that holds a Markov property, justifying the use of a log-loss assuming independence between timesteps. The time to the next event - if it's predictable now - is necessarily dependent on the future. Despite this it might be possible to formulate a random process for the hazard function s.t the MLE would coincide with the censored log-loss. One approach that we tried (and failed) was to try to find a positive random process Λ having the property that its expected value at a future step given current feature state is an RCHF of predetermined form, i.e $E[\Lambda(t+s)|X(t)] = R(X(t), s)$ with R being the Weibull RCHF as a function of the feature state vector $X(t)$. It would be interesting to see if this is possible.

Effect of varying length and weighting We only briefly discussed the ramifications of different weighting schemes on the loss function. The calibration in all experiments were reasonable even when trained with a different weighting-scheme than how it was evaluated. There was some bias here and if we're not careful its easy to introduce systematic bias in the probabilistic interpretation. This could have big and dangerous consequences in certain applications if the probabilities are not taken with a grain or salt or its interpretation adapted to the situation.

In the experiments we also found that after long training the RNNs would occasionally 'learn' to identify censoring, i.e overestimate the waiting time as it gets closer to the censoring point. This behavior - a kind of overfitting - is unwanted. Using Inverse Censoring Probability Weighting (see section 2.4) seems like a promising method of steering away the model from learning artifacts of censoring but we found no clear way on how to implement this for our problem.

Extensions We did not extend the model with multivariate target or other parametric models. This should be fairly easy to implement and may be useful in applications. Of particular interest would be to see computationally feasible multivariate covariance structures. There might even be reasons to use multivariate models to speed up training of univariate models. Consider the case when we're predicting an event defined by the realization of a set of events i.e a compound event. This could for example be a user having to perform a sequence of steps in order to be considered signed up or churned. Another example, for earthquake prediction we might be able to predict big quakes by using a distribution over time to quakes of different Richter scales and gradually weighting up the importance of the big quake. This could have some hand-holding effect on learning.

We tried to keep the discussion on a general level when possible so as to show the simplicity of switching out the Weibull to other distributions. The framework discussed in this thesis can hopefully be applied using other types of CHFs and gradient based learning frameworks other than ANNs and RNNs.

In order to keep it simple and explicit we used very naive implementations of the loss function. It would be natural to use the general framework for location-scale distributions used by most packages implementing Weibull-regression such as GAMLSS [52]. This consists of working on the log of the waiting times i.e working with the Gumbel distribution rather than the Weibull. As discussed previously there's both numerical- and computational reasons for why this makes more sense. A first move towards this goal could be to switch out the output activation function for α from softPlus to the exponential.

The RNN-architectures used were single layered vanilla LSTMs without dropout. This was by design - looking for a specific RNN architecture was outside the scope of this thesis, and with more complexity it would make it harder to generalize. Surely the Weibull distribution will impose specific dynamics on the network. Its possible that we could speed up learning by designing the output layers and hidden state updates to account for concepts like the minimum closure property. It's also reasonable to believe that the location- and scale- outputs should be given separate output layers so as to not correlate them too much.

In many applications the Weibull distribution might be better specified for values in the left tail than the right tail. In all experiments we saw that the error mostly showed up in the right tail. This is due to the fact that infinite waiting times is not really something we get to see very often in practice. There might be ways to truncate the right tail to get better predictions.

Comparison to other models At the time of publishing this thesis there's lots of things happening in this field. Both w.r.t developments for recurrent neural networks but particularly in the field of RUL-prediction.

We compared this framework to naive strategies and the log loss of the sliding box. This could be done in a more systematic way. The comparison to the sliding box was heavily dependent on the calibration of the models rather than the models ability to rank. A similar experiment where AUC for both models is compared would be interesting. In this context it would also be called for to compare the model with *learn-to-rank* models trained by optimizing similar metrics. It might also be interesting to see comparison to the various Kalman-Filter- and Tobit-regression based frameworks that have recently been published.

More standard datasets One issue that we found was the lack of datasets to compare performance on for these kinds of tasks. The CMAPSS is a step in the right direction. Because of the problematic test-train split of the original challenge dataset, without overfitting its impossible to make fair and competitive submissions. This thesis have largely been occupied with defining a metric to optimize over. When models are trained for different metrics its natural that the resulting models are hard or even wrong to compare. We would be very happy to find a set of datasets that consisted of sequences of events and features. The data could come from a range of fields such as PHM/RUL/Failure-prediction, health monitoring, Neural spike train data, Web event prediction, customer churn etc.

5.3 Conclusion

The purpose of this thesis was to describe and test a framework aimed at solving many of the problems associated with predicting waiting times: censoring, discrete or continuous data, time-varying covariates and sequences of varying length. We conclude that the proposed model (the WTTE-RNN) seems to be applicable and work well under all these circumstances.

The main contribution of this thesis has been to assemble some standard theoretical pieces to build something new. With respect to theory we've condensed the theoretical underpinnings of the classical field of Survival analysis - in particular using censored log-likelihood loss - and applied it in a new setting. In essence this thesis can be seen as adding and documenting an objective function that we think should be part of the general deep-learning toolbox.

The proposed model seems like the obvious solution for a wide range of problems. With this in mind we have been surprised and worried about finding so little similar work. As far as we know, this is the first documented discussion about censored log-likelihood loss in the context of RNNs in general and Weibull ANNs/RNNs in particular. Academias apparent lack of interest for this simple solution has led us to tread very cautiously and remain suspicious about the performance of the model. Throughout the work on this thesis we expected to find the approach to be inherently faulty. In the end we found no evidence that (given the assumption of uninformative censoring) the model is flawed or leads to biased predictions. The experiments showed that it worked well, in particular compared to the baselines which we compared it to.

As a side-effect we also hope that our description of the Sliding Box model to be useful. During the course of writing this thesis we've found many examples of the sliding box-model being applied in industry and a few papers referencing it but little discussion about the details of implementing it or its pros and cons. In particular we've found no academic papers detailing or comparing its use in the context of RNNs.

In general we found that all the research areas dealing with censored data have problems with transparency and would benefit from being less domain specific. It's hard to find a serious discussion about censoring that does not contain undecipherable medical jargon. We think that everybody would benefit if censored data and time-to-event prediction was treated in a more general mathematical machine learning framework. This would simplify collaboration and lead to new perspectives. Many research areas have benefited from taking this path, consider speech recognition, NLP and image processing.

As a concluding remark we are hopeful and curious about seeing more machine learning models that can be trained with censored data. We hope that this thesis might have made some small contribution in making standard concepts in survival analysis available to the wider machine learning community.

Bibliography

[1] Mei-Ling Ting Lee, GA Whitmore, and Bernard A Rosner. "Threshold regression for survival data with time-varying covariates". In: *Statistics in medicine* 29.7-8 (2010), pp. 896–905.

[2] Rupert G Miller Jr. *Survival analysis.* Vol. 66. John Wiley & Sons, 2011.

[3] Germán Rodríguez. "Lecture Notes on Generalized Linear Models." PhD thesis. 2010. URL: `http://data.princeton.edu/wws509/notes/`.

[4] John D. Kalbfleisch Ross L. Prentice. *The Statistical Analysis of Failure Time Data (Wiley Series in Probability and Statistics).* 2nd ed. 2002. ISBN: 047136357X,978047136357

[5] Klaus Greff et al. "LSTM: A search space odyssey". In: *arXiv preprint arXiv:1503.04069* (2015).

[6] Fima C Klebaner et al. *Introduction to stochastic calculus with applications.* Vol. 57. World Scientific, 2005.

[7] George Casella and Roger L Berger. *Statistical inference.* Vol. 2. Duxbury Pacific Grove, CA, 2002.

[8] Torsten Hothorn et al. "Survival ensembles". In: *Biostatistics* 7.3 (2006), pp. 355–373.

[9] Jonathan Burez and Dirk Van den Poel. "Handling class imbalance in customer churn prediction". In: *Expert Systems with Applications* 36.3 (2009), pp. 4626–4636.

[10] Shin-Yuan Hung, David C Yen, and Hsiu-Yu Wang. "Applying data mining to telecom churn management". In: *Expert Systems with Applications* 31.3 (2006), pp. 515–524.

[11] Guo-en Xia and Wei-dong Jin. "Model of customer churn prediction on support vector machine". In: *Systems Engineering-Theory & Practice* 28.1 (2008), pp. 71–77.

[12] Scott A Neslin et al. "Defection detection: Measuring and understanding the predictive accuracy of customer churn models". In: *Journal of marketing research* 43.2 (2006), pp. 204–211.

[13] Yu Zhao et al. “Customer Churn Prediction Using Improved One-Class Support Vector Machine”. In: *Advanced Data Mining and Applications: First International Conference, ADMA 2005, Wuhan, China, July 22-24, 2005. Proceedings*. Ed. by Xue Li, Shuliang Wang, and Zhao Yang Dong. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 300–306. ISBN: 978-3-540-31877-4. DOI: `10.1007/11527503_36`. URL: `http://dx.doi.org/10.1007/11527503_36`.

[14] Bart Larivière and Dirk Van den Poel. “Investigating the role of product features in preventing customer churn, by using survival analysis and choice modeling: The case of financial services”. In: *Expert Systems with Applications* 27.2 (2004), pp. 277–285.

[15] CL Chi, WN Street, and WH Wolberg. “Application of artificial neural network-based survival analysis on two breast cancer datasets.” In: *AMIA... Annual Symposium proceedings/AMIA Symposium. AMIA Symposium.* 2006, pp. 130–134.

[16] Elia Biganzoli et al. “Feed forward neural networks for the analysis of censored survival data: a partial logistic regression approach”. In: *Statistics in medicine* 17.10 (1998), pp. 1169–1186.

[17] Matt Peters. *Deep Learning for Customer Churn Prediction.* `https://moz.com/devblog/deep-learning-for-customer-churn-prediction/`. Blog. 2015.

[18] Zolidah Kasiran et al. “Customer Churn Prediction using Recurrent Neural Network with Reinforcement Learning Algorithm in Mobile Phone Users”. In: *International Journal of Intelligent Information Processing* 5.1 (2014), p. 1.

[19] Harald Steck et al. “On ranking in survival analysis: Bounds on the concordance index”. In: *Advances in neural information processing systems.* 2008, pp. 1209–1216.

[20] Jonas Kalderstam. “Neural Network Approaches To Survival Analysis”. PhD thesis. Lund University, 2015.

[21] Yifei Chen et al. “A gradient boosting algorithm for survival analysis via direct optimization of concordance index”. In: *Computational and mathematical methods in medicine* 2013 (2013).

[22] Andreas Mayr, Benjamin Hofner, and Matthias Schmid. “Boosting the discriminatory power of sparse survival models via optimization of the concordance index and stability selection”. In: *BMC bioinformatics* 17.1 (2016), p. 288.

[23] Rashmi Joshi and Colin Reeves. “Beyond the Cox model: artificial neural networks for survival analysis part II”. In: *Proceedings of the eighteenth international conference on systems engineering.* 2006, pp. 179–184.

[24] David Faraggi and Richard Simon. “A neural network model for survival data”. In: *Statistics in medicine* 14.1 (1995), pp. 73–82.

[25] David Faraggi, Michael LeBlanc, and John Crowley. “Understanding neural networks using regression trees: an application to multiple myeloma survival data”. In: *Statistics in medicine* 20.19 (2001), pp. 2965–2976.

[26] Vanya Van Belle et al. “Improved performance on high-dimensional survival data by application of Survival-SVM”. In: *Bioinformatics* 27.1 (2011), pp. 87–94.

[27] Matthias Schmid and Torsten Hothorn. “Flexible boosting of accelerated failure time models”. In: *BMC bioinformatics* 9.1 (2008), p. 269.

[28] Sanne JW Willems and M Fiocco. “Inverse Probability Censoring Weights for Routine Outcome Monitoring Data”. In: (2014).

[29] Odd O Aalen, Håkon K Gjessing, et al. “Understanding the shape of the hazard rate: A process point of view (with comments and a rejoinder by the authors)”. In: *Statistical Science* 16.1 (2001), pp. 1–22.

[30] Mei-Ling Ting Lee and GA Whitmore. “Threshold regression for survival analysis: modeling event times by a stochastic process reaching a boundary”. In: *Statistical Science* (2006), pp. 501–513.

[31] Xin He et al. “A model for time to fracture with a shock stream superimposed on progressive degradation: the Study of Osteoporotic Fractures”. In: *Statistics in medicine* 34.4 (2015), pp. 652–663.

[32] Mei-Ling Ting Lee and GA Whitmore. “Proportional hazards and threshold regression: their theoretical and practical connections”. In: *Lifetime data analysis* 16.2 (2010), pp. 196–214.

[33] Abhinav Saxena et al. “Damage propagation modeling for aircraft engine run-to-failure simulation”. In: *Prognostics and Health Management, 2008. PHM 2008. International Conference on.* IEEE. 2008, pp. 1–9.

[34] Emmanuel Ramasso and Abhinav Saxena. “Performance Benchmarking and Analysis of Prognostic Methods for CMAPSS Datasets.” In: *International Journal of Prognostics and Health Management* 5.2 (2014), pp. 1–15.

[35] Felix O Heimes. “Recurrent neural networks for remaining useful life estimation”. In: *Prognostics and Health Management, 2008. PHM 2008. International Conference on.* IEEE. 2008, pp. 1–6.

[36] Borja Ibarz-Gabardos and Pedro J Zufiria. “A Kalman filter with censored data”. In: *Intelligent Signal Processing, 2005 IEEE International Workshop on.* 2005, pp. 74–79.

[37] Piero Baraldi, Francesca Mangili, and Enrico Zio. “A kalman filter-based ensemble approach with application to turbine creep prognostics”. In: *IEEE Transactions on Reliability* 61.4 (2012), pp. 966–977.

[38] Cory Miller et al. “Estimation of mobile vehicle range & position using the tobit Kalman filter”. In: *53rd IEEE Conference on Decision and Control.* IEEE. 2014, pp. 5001–5007.

[39] Bethany Allik, Michael J Piovoso, and Ryan Zurakowski. “Recursive estimation with quantized and censored measurements”. In: *American Control Conference (ACC), 2016*. American Automatic Control Council (AACC). 2016, pp. 5130–5135.

[40] Bethany Allik et al. “The Tobit Kalman Filter: An Estimator for Censored Measurements”. In: *IEEE Transactions on Control Systems Technology* 24.1 (2016), pp. 365–371.

[41] Robert C Wilson, Matthew R Nassar, and Joshua I Gold. “Bayesian online learning of the hazard rate in change-point problems”. In: *Neural computation* 22.9 (2010), pp. 2452–2476.

[42] Anestis Antoniadis, Gérard Grégoire, and Guy Nason. “Density and hazard rate estimation for right-censored data by using wavelet methods”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 61.1 (1999), pp. 63–84.

[43] Mei-Cheng Wang, Jing Qin, and Chin-Tsang Chiang. “Analyzing recurrent event data with informative censoring”. In: *Journal of the American Statistical Association* 96.455 (2001), pp. 1057–1065.

[44] Ratan Dasgupta. “Characterization Theorems for Weibull Distribution with Applications”. In: *Journal of Environmentl Statistics* 6.4 (Sept. 23, 2014), pp. 1–25. ISSN: 1945-1296. URL: `http://jes.stat.ucla.edu/v06/i04`.

[45] Fritz Scholz. “Inference for the Weibull distribution”. In: (2008).

[46] Ronald Aylmer Fisher and Leonard Henry Caleb Tippett. “Limiting forms of the frequency distribution of the largest or smallest member of a sample”. In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 24. 02. Cambridge Univ Press. 1928, pp. 180–190.

[47] Dionissios T Hristopulos, Manolis P Petrakis, and Giorgio Kaniadakis. “Weakest-Link Scaling and Extreme Events in Finite-Sized Systems”. In: *Entropy* 17.3 (2015), pp. 1103–1122.

[48] Toshio Nakagawa and Shunji Osaki. “The discrete Weibull distribution”. In: *IEEE Transactions on Reliability* 24.5 (1975), pp. 300–301.

[49] Junyoung Chung et al. “A recurrent latent variable model for sequential data”. In: *Advances in neural information processing systems*. 2015, pp. 2962–2970.

[50] Piotr Bojanowski, Armand Joulin, and Tomas Mikolov. “Alternative structures for character-level RNNs”. In: *arXiv preprint arXiv:1511.06303* (2015).

[51] FW Scholz and Boeing Phantom Works. “Maximum likelihood estimation for type I censored Weibull data including covariates”. In: 1996.

[52] D Mikis Stasinopoulos, Robert A Rigby, et al. “Generalized additive models for location scale and shape (GAMLSS) in R”. In: *Journal of Statistical Software* 23.7 (2007), pp. 1–46.