A Recurrent Neural Network Survival Model: Predicting Web User Return Time

Georg L. Grob[1], Ângelo Cardoso[2⋆], C. H. Bryan Liu[2], Duncan A. Little[2], and Benjamin Paul Chamberlain[1,2]

[1] Imperial College London, London, UK grobgl@gmail.com
[2] ASOS.com, London, UK bryan.liu@asos.com

Abstract. The size of a website's active user base directly affects its value. Thus, it is important to monitor and influence a user's likelihood to return to a site. Essential to this is predicting *when* a user will return. Current state of the art approaches to solve this problem come in two flavors: (1) Recurrent Neural Network (RNN) based solutions and (2) survival analysis methods. We observe that both techniques are severely limited when applied to this problem. Survival models can only incorporate aggregate representations of users instead of automatically learning a representation directly from a raw time series of user actions. RNNs can automatically learn features, but can not be directly trained with examples of non-returning users who have no target value for their return time. We develop a novel RNN survival model that removes the limitations of the state of the art methods. We demonstrate that this model can successfully be applied to return time prediction on a large e-commerce dataset with a superior ability to discriminate between returning and non-returning users than either method applied in isolation.

Keywords: User return time · web browse sessions · Recurrent neural network · Marked temporal point process · Survival analysis.

1 Introduction

Successful websites must understand the needs, preferences and characteristics of their users. A key characteristic is *if* and *when* a user will return. Predicting user return time allows a business to put in place measures to minimize absences and maximize per user return probabilities. Techniques to do this include timely incentives, personalized experiences [16] and rapid identification that a user is losing interest in a service [2]. A related problem is user churn prediction for non-contractual services. In this case a return time threshold can be set, over which a user is deemed to have churned. Similar prevention measures are often put in place for users with high churn risks [5].

This paper focuses on predicting user return time for a website based on a time series of user sessions. The sessions have additional features and so form

⋆ Now with Vodafone Research and ISR, IST, Universidade de Lisboa

a *marked temporal point processes* for each user. As some users do not return within the measurement period, their return times are regarded as *censored*. The presence of missing labels makes the application of standard supervised machine learning techniques difficult. However, in the field of survival analysis, an elegant solution to the missing label problem exists which has been transferred to a variety of settings [24] [31].

Recurrent Neural Networks (RNNs) have achieved significant advances in sequence modelling, achieving state-of-art performance in a number of tasks [33, 34, 19]. Much of the power of RNNs lie in their ability to automatically extract high order temporal features from sequences of information. Many complex temporal patterns of web user behaviour can exist. This can include noisy oscillations of activity with periods of weeks, months, years, pay days, festivals and many more beside. Exhaustively handcrafting exotic temporal features is very challenging and so it is highly desirable to employ a method that can automatically learn temporal features.

In this paper, we predict user return time by constructing a recurrent neural network-based survival model. This model combines useful aspects of both RNNs and survival analysis models. RNNs automatically learn high order temporal features from user 'sessions' and their associated features. They can not however, be trained with examples of users who do not return or the time since a user's latest session. Survival models can include information on users who do not return (right-censored users) and the time since their last session, but cannot learn from a sequence of events.

Our main contribution is to develop a RNN-based survival model which incorporates the advantages of using a RNN and of using survival analysis. The model is trained on sequences of sessions and can also be trained with examples of non-returning users. We show that this combined model outperforms both RNNs and survival analysis employed in isolation. We also provide the code implementation for use by the wider research community.[3]

2 Background

We are interested in predicting the return times of users to a website. We select a period of time during which users must be observed visiting a site and call this the *Activity window*. We declare a separate disjoint period of time called the *Prediction window* from which we generate return time labels and make predictions and both windows are illustrated in Figure 1. There are necessarily two types of user: returning and non-returning. We consider a user as non-returning if they do not have any sessions within the Prediction window, and a returning user as those who do. As suggested by Wangperawong et al. [35] we record data for some time preceding the Activity window (called the *Observation window*) to avoid producing a model that would predominantly predict non-returning users. This set up allows us to define the set of users active in the

[3] `https://github.com/grobgl/rnnsm`

activity window; $\mathcal{C}$, the set of returning users, $\mathcal{C}_{\text{ret}}$ and the set of non-returning users $\mathcal{C}_{\text{non-ret}}$.

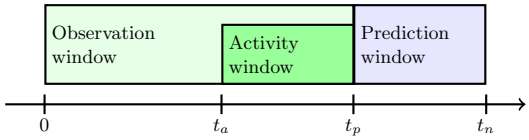


Fig. 1: Illustration of the observation window, the activity window, and the prediction window. The x-axis denotes time. We use observations occurring within the observation window $[0, t_p]$ on users active in the activity window $[t_a, t_p]$. We then predict their return times with respect to the prediction window $(t_p, t_n]$.

We follow the definition of return time suggested by Kapoor et al. [26], that is, $d_{j+1}^{(i)} = t_{j+1}^{(i)} - t_j^{(i)}$, denotes the period between the end of the ith user's jth session and the beginning of the succeeding session. A session occurs when a user browses a website or mobile app. The ith user's jth session, $s_j^{(i)} = \left(t_j^{(i)}, \boldsymbol{y}_j^{(i)}\right)$, has an associated start time $t_j^{(i)} \in [0, \infty)$ and a vector of n features $\boldsymbol{y}_j^{(i)}$. A user's browsing history can therefore be represented as a sequence of sessions, $\mathcal{S}^{(i)}$, where $\mathcal{S}^{(i)} = \left(s_1^{(i)}, s_2^{(i)}, \ldots\right)$.

2.1 Survival analysis

Survival analysis models the time until an event of interest occurs [27]. In the context of users' return time prediction, return time is equivalent to survival time.

Harzard and survival functions Here we clarify the notation and standard results used throughout the paper, as defined in [32]. T is a random variable denoting the lifetime of an individual. The probability density function for T, corresponding to the probability of the event of interest occurring at time t is written as:

$$f(t) = \lim_{\delta t \to 0} \frac{P(t < T \leq t + \delta t)}{\delta t}, \tag{1}$$

with $F(t)$ as the corresponding cumulative density function. The survival function $S(t)$, denoting the probability of the event of interest not having occurred by time t, is defined as:

$$S(t) = P(T \geq t) = 1 - F(t) = \int_t^{\infty} f(z)\, \mathrm{d}z. \tag{2}$$

The hazard function, which models the instantaneous rate of occurrence given that the event of interest did not occur until time t, is defined as:

$$\lambda(t) = \lim_{\delta t \to 0} \frac{P(t \leq T < t + \delta t | T \geq t)}{\delta t} = \frac{f(t)}{1 - F(t)} = \frac{f(t)}{S(t)}. \tag{3}$$

The hazard function is related to the survival function by

$$-\frac{d \log S(t)}{dt} = \lambda(t). \tag{4}$$

Censoring Censoring occurs when labels are partially observed. Klein and Moeschberger [27] provide two definitions of censoring (1) an *uncensored observation* when the label value is observed and (2) *right-censored observation* when the label value is only known to be above an observed value.

In the context of return time prediction, some users do not return to the website during the observation period. We label these users as non-returning, but some will return to the website after the observation period. Figure 2 shows a selection of returning and non-returning users. To estimate the average return time, it is not sufficient to only include returning users as this underestimates the value for all users. Including non-returning users' time since their latest session still underestimates the true average return time.

To address this problem, we must incorporate censoring into our survival model. This is achieved using a likelihood function that has separate terms to account for censoring:

$$L(\boldsymbol{\theta}) = \prod_{i \in \text{unc.}} P(T = T_i|\boldsymbol{\theta}) \prod_{j \in \text{r.c.}} P(T > T_j|\boldsymbol{\theta}) = \prod_{i \in \text{unc.}} f(T_i|\boldsymbol{\theta}) \prod_{j \in \text{r.c.}} S(T_j|\boldsymbol{\theta}), \tag{5}$$

where $\boldsymbol{\theta}$ is a vector of model parameters. *unc.* and *r.c.* denote the uncensored and right-censored observations respectively. T_i and T_j denote the exact value of the uncensored observation and the minimum value of the right-censored observation respectively. For simplicity we assume there are no observations which are subject to other types of censoring.

2.2 Cox Proportional Hazards Model

The Cox proportional hazards model [11] is a popular survival regression model. It is applied by Kapoor et al. [26] to predict the return time of users.

The model assumes that one or more given covariates have (different) multiplicative effects on a base hazard. The model can be defined in terms of its hazard function:

$$\lambda(t|\boldsymbol{x}) = \lambda_0(t) \exp\left(\boldsymbol{x}^T \boldsymbol{\beta}\right), \tag{6}$$

where $\lambda_0(t)$ is the baseline hazard, $\boldsymbol{x}$ a vector of covariates, and β_i the multiplier for covariate x_i. The model implicitly assumes the parameters for each covariate can be estimated without considering the baseline hazard function.

Various methods to estimate the multipliers $\boldsymbol{\beta}$ exist [12, 25, 3], and we use that featured in the *lifelines* library [14], which maximises the Efron's partial likelihood [17] for $\boldsymbol{\beta}$ to obtain the estimate $\hat{\boldsymbol{\beta}}$.

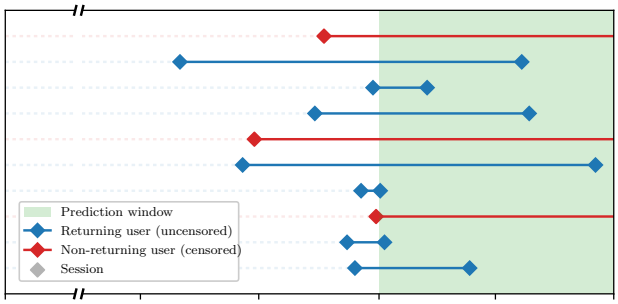


Fig. 2: Visualisation of censored return times. The horizontal axis represents time and the vertical axis different users. The shaded area represents the prediction window. The solid lines represent the return time after users' last sessions in the observation time frame – the value we are aiming to predict. The return times of users that do not return in the prediction time frame are censored. We do not know their actual return time. However, we do know that their return time spans at least across the entire prediction window.

The estimated baseline hazard is then computed as described by Cox and Oakes [13]:

$$\hat{\lambda}_0\left(t_{(i)}\right) = \frac{d_{(i)}}{\sum_{j \in \mathcal{R}\left(t_{(i)}\right)} \exp\left(\hat{\boldsymbol{\beta}}^T \boldsymbol{x}_j\right)}, \tag{7}$$

where $t_{(i)}$ denotes the i unique ordered time of an event of interest, $d_{(i)}$ is the number of events of interest occurring at time $t_{(i)}$, and $\mathcal{R}\left(t_{(i)}\right)$ is the set of individuals for whom the event of interest has not occurred by time $t_{(i)}$. The users' return times can then be estimated by calculating their expected survival time.

The Cox proportional hazards model is particularly suitable for this problem as it allow us to include right-censored observations (users who appeared not returning) as training examples. Their value is only known to be above a certain value, which corresponds to the time between the last session in the observation time frame and the end of the prediction time frame.

2.3 Recurrent Neural Networks, LSTM, and Embeddings

A recurrent neural network (RNN) is a feedforward neural network where the output of a hidden unit at the current timestep is fed back into the hidden unit so that it forms part of the input for the preceding timesteps. This allows RNNs to learn from sequences of events ordered chronologically or otherwise. The power of an RNN lies in its ability to learn from the current state of the sequence within the context of what has gone before. This context is stored as an internal memory within the hidden units of the RNN. For modelling time series data, sequences of events are discretised in time [20, 4, 6].

Long Short-Term Memory (LSTM) units [23] and Gated Recurrent Units (GRUs) [8] were developed to overcome the problems associated with learning long-term dependencies in traditional RNNs [1]. LSTMs and GRUs solve this issue by learning what information they should keep from the previous time step and what information they should forget.

It is also common to add embedding layer(s) to neural networks to transform large and sparse information info dense representations before the actual training of the network [10, 18]. The embedding layer will automatically learn features (in the form of the dense representation's individual components) for the neural network's consumption.

First popularised by Mikolov et al. [30] to encode words in documents, embedding layers have been shown to encode various categorical features with a large number of possible values well [28, 7]. In this paper we use the embedding layer implementation in Keras [9] with a TensorFlow backend.

2.4 Recurrent Temporal Point Processes

Temporal point processes model times of reoccurring events, which may have markers (features) associated with them, such as click rates and duration for web sessions. Manzoor et al. [29] modeled both the timing and the category of a user's next purchase given a history of such events using a Hawkes process, which assumes the occurrence of past events increases the likelihood of future occurrences [22].

Du et al. [15] propose the recurrent marked temporal point process (RMTPP) to predict both timings and markers (non-aggregated features) of future events given a history of such events. They assume events have exactly one discrete marker and employ RNNs to find a representation for the event history, which then serves as input to the hazard function. The paper demonstrates that such process can be applied in a wide variety of settings, including return times of users of a music website.

The RMTPP model is formulated as follows. Let $\mathcal{H}_t$ be the history of events up to time t, containing event pairs $(t_j, y_j)_{j \in \mathbb{Z}^+}$ denoting the event timing and marker respectively. The conditional density function corresponds to the likelihood of an event of type y happening at time t: $f^*(t, y) = f(t, y|\mathcal{H}_t)$.[4]

A compact representation $\boldsymbol{h}_j$ of the history up to the jth event is found through processing a sequence of events $\mathcal{S} = (t_j, y_j)_{j=1}^n$ with an RNN. This allows the representation of the conditional density of the next event time as:

$$f^*(t_{j+1}) = f(t_{j+1}|\boldsymbol{h}_j). \tag{8}$$

Given $\boldsymbol{h}_j$, the hazard function of the RMTPP is defined as follow:

$$\lambda^*(t) = \exp\Big(\underbrace{\boldsymbol{v}^{(t)\top}\boldsymbol{h}_j}_{\substack{\text{past}\\ \text{influence}}} + \underbrace{w(t - t_j)}_{\substack{\text{current}\\ \text{influence}}} + \underbrace{b^{(t)}}_{\substack{\text{base}\\ \text{intensity}}}\Big), \tag{9}$$

[4] The $*$-notation is used to denote the conditioning on the history.

where $\boldsymbol{v}^{(t)}$ is the hidden representation in the recurrent layer in the RNN (which takes only $\boldsymbol{h}_j$, the representation of past history, into account), $t - t_j$ is the absence time at the time of prediction (the current information), w is a specified weight balancing the influence from the past history from that of the current information,[5] and $b^{(t)}$ is the base intensity (or bias) term of the recurrent layer.

The conditional density is given by swapping the terms in Equation (3) and integrating Equation (4):

$$\begin{aligned} f^*(t) &= \lambda^*(t) \exp\left(-\int_{t_j}^{t} \lambda^*(\tau)\, d\tau\right) \\ &= \exp\left(\boldsymbol{v}^{(t)\top}\boldsymbol{h}_j + w(t-t_j) + b^{(t)} + \frac{1}{w}\exp\left(\boldsymbol{v}^{(t)\top}\boldsymbol{h}_j + b^{(t)}\right)\right. \\ &\qquad \left. - \frac{1}{w}\exp\left(\boldsymbol{v}^{(t)\top}\boldsymbol{h}_j + w(t-t_j) + b^{(t)}\right)\right). \end{aligned} \tag{10}$$

The timings of the next event can then be estimated by taking the expectation of the conditional density function:

$$\hat{t}_{j+1} = \int_{t_j}^{\infty} t f^*(t)\, \mathrm{d}t. \tag{11}$$

The architecture of the RNN is illustrated in Figure 3. The event markers are embedded into latent space. The embedded event vector and the event timings are then fed into a recurrent layer. The recurrent layer maintains a hidden state $\boldsymbol{h}_j$ which summarises the event history. The recurrent layer uses a rectifier as activation function and is implemented using LSTM or GRUs.

The parameters of the recurrent layer ($\boldsymbol{v}^{(t)}$ and $b^{(t)}$) are learned through training the RNN using a fully connected output layer with a single neuron and linear activation, with the negative log-likelihood[6] of observing a collection of example sequences $\mathcal{C} = \left\{\left(t_j^{(i)}, y_j^{(i)}\right)_{j=1}^{n^{(i)}}\right\}_{i\in\mathbb{Z}^+}$ defined as:

$$-\ell(\mathcal{C}) = -\sum_i \sum_j \log f\left(t_{j+1}^{(i)} \middle| \boldsymbol{h}_j\right). \tag{12}$$

3 Method

Survival models can only accept a vector of features aggregated for each user as input. By using aggregated features, we discard a significant proportion of information contained in the time series of events. Unlike survival models, RNNs

[5] Du et al. [15] specified a different fixed value for w in their models fitted under different datasets. We use Bayesian Optimisation to find the best w in our experiments.

[6] The original log-likelihood function in [15] took into account both the likelihood of the timing of the next event and the likelihood of the marker taking certain value. The later is omitted for simplicity as we do not deal with marker prediction here.

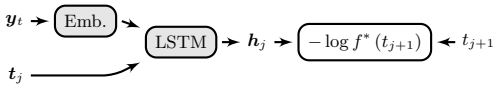
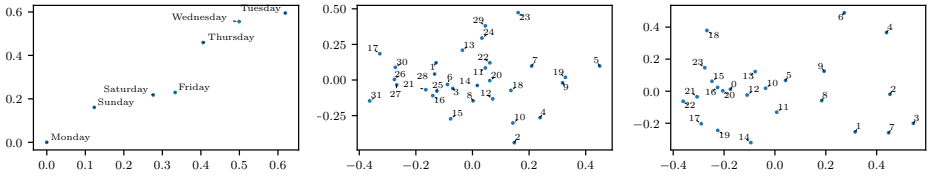


Fig. 3: Architecture of the recurrent neural network used in the RMTPP model to learn a representation $\boldsymbol{h}_j$ of an event history consisting of pairs of timings and markers (t_i, y_i). $\boldsymbol{t}_j$ and $\boldsymbol{y}_j$ represent the timings and events of the history up to the j^{th} event. $\boldsymbol{h}_j$ is learned through minimising the negative log-likelihood of the $j+1^{\text{th}}$ event occurring at time t_{j+1}. The hidden representation $\boldsymbol{h}_j$ is then used as parameter to a point process.

are capable of utilising the raw user history and automatically learning features. However, censored data can not be included. Omitting censored users causes predictions to be heavily biased towards low return times. We remove the limitations of RNNs and Cox proportional hazard models by developing a novel model that can incorporate censored data, use multiple heterogeneous markers and automatically learn features from raw time series data.



(a) Day of week embeddings (two-dimensional).

(b) Day of month embeddings (five-dimensional).

(c) Hour of day embeddings (four-dimensional).

Fig. 4: Embeddings for discrete features used in the simple recurrent neural network model. The embeddings translate discrete inputs into vectors of specified dimension. We can observe a clusters of late night and early morning hours in (c), and a separation of weekend days from weekdays in (a), suggesting that viable representations are found. The embeddings are found by training a model with embeddings layers as input (alongside the remaining inputs) and then evaluating the embeddings for each possible input. Dimensionality reduction through PCA is used to produce the visualisations in case the embedding vector has more than two dimensions.

3.1 Heterogeneous Markers

In many practical settings, multiple heterogeneous markers are available describing the nature of events. Markers can be be both discrete and continuous. To encode discrete markers we use an embedding layer. We also embed cyclic features, an example being the hour of an event. Instead of encoding the hour as $\{0, 1..23\}$ we learn an embedding that is able to capture the similarity between e.g. the hours 23:00 and 0:00 (see Figure 4 for a visualisation of the embeddings on some of the features used). Embedding layers solve this problem through mapping discrete values into a continuous vector space where similar categories

are located nearby to each other. We train the network to find mappings that represent the meaning of the discrete features with respect to the task, i.e. to predict the return time. We apply a unit norm constraint to each embedding layer, enforcing the values to be of a similar magnitude to the non-categorical variables, which are also normalized.

To avoid an expensive search for a suitable number of embedding dimensions during training, we perform a preliminary simulation. We train a model with a high number of dimensions per feature and use Principal Component Analysis (PCA) to reduce the dimensionality to the minimum number required to account for more than 90% of the initial variance.

The embeddings and the non-categorical features are fed into a single dense layer, which produces the input to the LSTM. Figure 5 shows how the model processes heterogeneous input data.

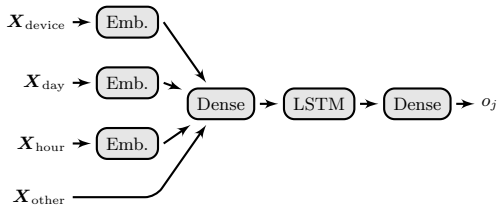


Fig. 5: Adapted recurrent marked temporal point process architecture. Embedding layers are used for each discrete feature. The embeddings and the remaining continuous features are fed into a dense layer and then the LSTM layer. The LSTM layer finds a hidden representation of the user's session history up to the j^{th} session. A value o_j is obtained from the hidden representation through a single-neuron output layer with linear activation. The negative log-likelihood of the next session occurring at time t_{j+1} is used to train the model.

3.2 Recurrent Neural Network Survival Model (RNNSM)

We combine ideas from RNNs and survival analysis to produce a model that is able to incorporate non-returning users and automatically learn features in a survival regression setting. To achieve this we use the survival function as a factor for right-censored (non-returned user) observations. We start with the log likelihood function defined in Equation (5)

$$\ell\left(\mathcal{C}\right) = \sum_{m} \sum_{i \in \text{unc.}} \log f^{*}\left(t_{i+1}^{(m)}\right) + \sum_{n} \sum_{j \in \text{r.c.}} \log S^{*}\left(t_{j+1}^{(n)}\right), \tag{13}$$

which is a sum over all session intervals for all users.

The first term is thus the log-likelihood for a single returning user at time t_{j+1} given an embedded representation of the user's session history up to their latest session at time t_j as $\boldsymbol{h}_j$. This log-likelihood is defined as

$$\begin{aligned} &\ell_{\text{ret}}\left(t_{j+1}\right) = \log f^{*}\left(t_{j+1}\right) \\ &= o_j + w\left(t_{j+1} - t_j\right) + \frac{1}{w}\exp\left(o_j\right) - \frac{1}{w}\exp\left(o_j + w\left(t_{j+1} - t_j\right)\right), \end{aligned} \tag{14}$$

where o_j represents the output of the fully-connected layer after jth step of the input sequence: $o_j = \boldsymbol{v}^T \boldsymbol{h}_j + b$. For the expression in the second term we substitute the survival function given by

$$S^*(t) = \exp\left(-\int_{t_j}^{t} \lambda^*(\tau)\, d\tau\right) = \exp\left(\frac{1}{w}\exp(o_j) - \frac{1}{w}\exp(o_j + w(t - t_j))\right) \tag{15}$$

from Equation (10) to get the log-likelihood term for a single censored user:

$$\ell_{\text{non-ret}}(t_{j+1}) = \log S^*(t_{j+1}) = \frac{1}{w}\exp(o_j) - \frac{1}{w}\exp(o_j + w(t - t_j)), \tag{16}$$

where t_{j+1} refers to the time between the jth user's last session in the observation window and the end of the prediction window.

We can now express a loss function that incorporates examples of non-returned users. Note that in a sequence of active days of a non-returning user, only the last return time is censored. The loss for all users is given by

$$-\ell(\mathcal{C}) = -\sum_i \sum_j \ell\left(t_{j+1}^{(i)}\right), \tag{17}$$

where

$$\ell\left(t_j^{(i)}\right) = \begin{cases} \ell_{\text{non-ret}}\left(t_j^{(i)}\right), & \text{if} \quad i \in \mathcal{C}_{\text{non-ret}} \text{ and } j = n^{(i)} + 1 \\ \ell_{\text{ret}}\left(t_j^{(i)}\right), & \text{otherwise} \end{cases} \tag{18}$$

and $\mathcal{C} = \mathcal{C}_{\text{ret}} \cup \mathcal{C}_{\text{non-ret}}$ denotes the collections of all users' session histories, consisting of the histories of returning and non-returning users.

3.3 Return Time Predictions

We predict the return time, which is the time between two sessions: $d_{j+1} = t_{j+1} - t_j$ using the expectation of the return time given the session history:

$$\hat{d}_{j+1} = \mathbb{E}[T|\mathcal{H}_j] = \int_{t_j}^{\infty} S^*(t)\, dt. \tag{19}$$

This integral does not in general have a closed form solution, but can easily be evaluated using numerical integration.

However, this expression allows the model to predict users to return before the start of the prediction window (see Figure 2). Therefore we need to censor the predictions by finding $\mathbb{E}[T|T > t_p]$ where t_p is the start of the prediction window. We show that the conditional expected return time can be derived from the expected return time through applying the definition of the survival function.

$$\begin{aligned} & \mathbb{E}[T] = P(T > t_p)\,\mathbb{E}[T|T > t_p] + P\left(\overline{T > t_p}\right)\mathbb{E}\left[T\middle|\overline{T > t_p}\right] \\ \Leftrightarrow \quad & \mathbb{E}[T|T > t_p] = \frac{\mathbb{E}[T] - P(T \le t_p)\,\mathbb{E}[T|T \le t_p]}{P(T > t_p)}. \end{aligned} \tag{20}$$

Using Equation (2), we obtain:

$$\begin{aligned}\mathbb{E}\left[T|T>t_s\right] &= \frac{\int_0^\infty S(z)\,dz - \left(1-S\left(t_s\right)\right)\int_0^{t_s} S\left(z\right)\,dz}{S\left(t_s\right)} \\ &= \frac{\int_{t_s}^\infty S(z)\,dz}{S\left(t_s\right)} + \int_0^{t_s} S\left(z\right)\,dz.\end{aligned} \tag{21}$$

4 Experiments

Here we compare several methods for predicting user return time, discussing the advantages, assumptions and limitations of each and providing empirical results on a real-world dataset. We experiment with six distinct models: (1) a baseline model, using the time between a user's last session in the observation time frame and the beginning of the prediction time frame ("Baseline"); (2) a simple RNN architecture ("RNN");[7] (3) a Cox proportional hazard model ("CPH") (4) a Cox proportional hazard model conditioned on absence time ("CPHA" — see Section 3.3); (5) a RNN survival model ("RNNSM"); (6) a RNN survival model conditioned on absence time ("RNNSMA" — see Section 3.3).

The dataset is a sample of user sessions from ASOS.com's website and mobile applications covering a period of one and a half years. Each session is associated with a user, temporal markers such as the time and duration, and behavioural data such as the number of images and videos viewed during a session. The dataset is split into training and test sets using split that is stratified to contain equal ratios of censored users. In total, there are 38,716 users in the training set and 9,680 users in the test set. 63.6% of users in both sets return in the prediction window. In the test set, the targeted return time of returning users is 58.04 days on average with a standard deviation of 50.3 days.

Evaluating the models based solely on the RMSE of the return time predictions is problematic because churned users can not be included. We therefore use multiple measures to compare the performance of return time models. These are the root mean squared error [26, 15],[8] concordance index [21],[9] non-returning AUC, and non-returning recall.[10]

4.1 Result on Performance Metrics

We report the performance metrics on the test set after training the models in Table 1. As the test dataset only contains users that were active in the activity window, the baseline model predicts that all users will return.

[7] It consists of a single LSTM layer followed by a fully connected output layer with a single neuron. The loss function is the mean squared error using only returning users as there is no target value for non-returning users.

[8] Unlike in the cited publications, our dataset contains a large proportion of non-returning users. The score thus only reflects the performance on returning users.

[9] This incorporates the knowledge of the censored return time of non-returning users.

[10] For AUC and recall, we treat the users who did not return within the prediction window as the positive class in a binary-classification framework.

	Baseline	CPH	CPHA	RNN	RNNSM	RNNSMA
RMSE (days)	43.25	49.99	59.81	**28.69**	59.99	63.76
Concordance	0.500	0.816	**0.817**	0.706	0.739	0.740
Non-returning AUC	0.743	0.793	0.788	0.763	**0.796**	0.794
Non-returning recall	0.000	0.246	0.461	0.000	0.538	**0.605**

Table 1: Comparison of performance of return time prediction models. The RMSE is just for returning users. Best values for each performance metric are highlighted in bold.

The CPH model uses an aggregated representation of each user's session history and additional metrics such as the absence time. The CPH model outperforms the baseline in each performance metric with the exception of the RMSE. This suggests that the improved non-returning recall rate can therefore be partially attributed to the model learning a positive bias for return times. This effect is even more pronounced for the CPHA model. However, the improvement in the concordance score demonstrates that, beyond a positive bias, a better relative ordering of predictions is achieved. Both CPH models perform particularly well in terms of the concordance score, suggesting that their predictions best reflect the relative ordering.

The RNN model cannot recall any non-returning users as its training examples only included returning users. However, the RMSE score demonstrates that the RNN model is superior in terms of predicting the return time of returning users and thus that sequential information is predictive of return time.

Finally, the recurrent neural network-based survival model (RNNSM) further improves the recall of non-returning users over the CPHA model without notable changes in the RMSE. More importantly it obtains the best performance for non-returning AUC, meaning it is the best model to discriminate between returning and non-returning users in the prediction window. Applying the absence-conditioned expectation to obtain predictions from the RNNSM further improves the model's performance on non-returning recall. However, the concordance scores of both RNNSM models suggest that the relative ordering is not reflected as well as by the CPH model.

4.2 Prediction error in relation to true return time

To evaluate the performance of each model in more detail we group users by their true return time, rounded down to a week. We then evaluate each model's performance based on a number of error metrics. This is to assess the usefulness of each model; for example, a model which performs well on short return times but poorly on long return times would be less useful in practice than one that performs equally well on a wide distribution of return times.

Root mean squared error The RMSE in relation to the return time in weeks is shown for each return time prediction model in Figure 6. The majority of users in the dataset return within ten weeks; we see that for the baseline model and

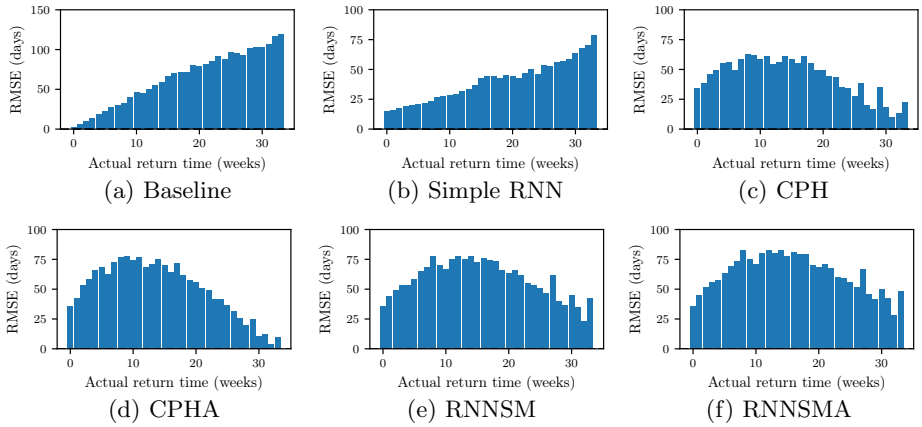


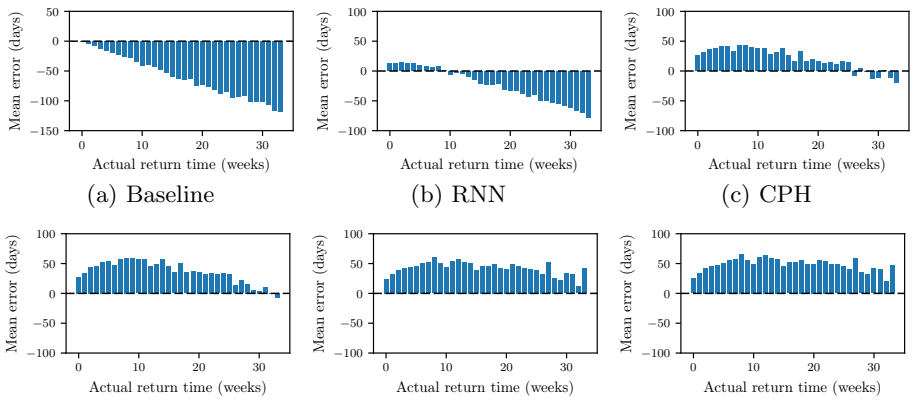
Fig. 6: Root mean squared error (RMSE) in relation to true return time compared between return time prediction models. In order to get a more detailed impression of each model's performance in relation to the true return time, we group users by their true return time rounded down to weeks. For each group, we then find the RMSE. Note the adjusted scale for the baseline model (a).

the RNN model the RMSE for these users is relatively low, this gives a low overall RMSE. However, for users who have longer return times both of those models perform increasingly poorly for increasing true return times.

For the models that incorporate training examples of both non-returning and returning users we see a different pattern. The performance for users with longer return times is generally better than those returning earlier. This demonstrates that these models are able to use censored observations to improve predictions for returning users. While the overall RMSE is lower for the CPH model compared the RNNSM (see Table 1), the distribution of errors is skewed, with a higher RMSE for earlier returning users.

We also see the effect of the absence-conditioned return time expectation. For the CPH model there is an increase in performance for users with very long return times, however there is a significant negative impact on users with shorter return times. This results suggest that the absence-conditioned expectation is more suitable for the RNNSM as it seems to have little effect on the RMSE distribution whilst improving the non-returning recall as can be seen in Table 1.

Mean error Figure 7 shows the mean error for each group and each model. The baseline model will always underestimate the true return time as by definition it can only predict a value equal to or lower than the true return time. The RNN model's performance is worse for users returning later, this is due to the restriction of predicting all users to return within a certain window leading to a negative bias for users returning later. The CPH model and the RNNSMs both overestimate the return times of the majority of users. It is possible to subtract the mean prediction error on the training set from these predictions in order to reduce the error, however this would lead to a reduction in non-returning AUC as overall return times would be reduced.



(a) Baseline (b) RNN (c) CPH

(d) CPHA (e) RNNSM (f) RNNSMA

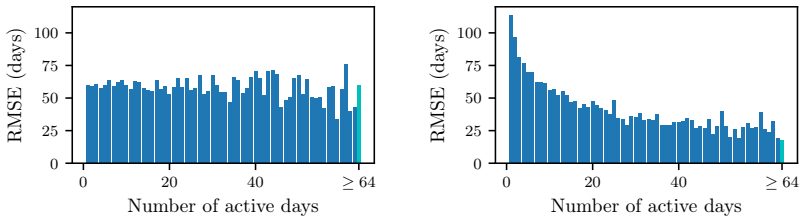
Fig. 7: Mean error per in relation to true return time comparison for all return time prediction models. Users are grouped by their true return time to the nearest week. This allows us to determine how the prediction bias is related to the true return time. We see that models which include non-returning users in training have a positive bias on the prediction for returning users, those that don't have a negative bias. Note the adjusted range for the baseline model (a).

4.3 Error in relation to number of active days

In this section we group users by their number of active days, an active day is a day on which a user had a session. We plot the RMSE in days for the CPHA model and RNNSMA model in Figure 8. These are the two best performing models which include returning users in terms of non-returning AUC and recall. We use up to 64 active days per user – we therefore group all users with 64 or more active days. We can immediately see that the RNNSM is able to make better predictions for users with a a higher number of active days. This is not the case for the CPHA model. This demonstrates that for users with more active days (longer RNN input sequences) the RNNSM model improves greatly. This again indicates that the sequence information captured by the RNNSM is predictive of user return and is preferable for users with a larger number of sessions.

5 Discussion

We have developed the RNNSM, a novel model that overcomes the weaknesses of survival models and RNNs for user return time prediction. We have highlighted the importance of including right-censored observations in return time prediction models and extended the Cox proportional hazard model to include users' absence time. We found that for modelling recurring events, a limitation of existing survival regression models is that they only operate on aggregate representations instead of raw time-series data. We addressed this problem by using a RNN point process model, which combines the advantages of survival



(a) CPHA (b) RNNSMA

Fig. 8: The number of days a user is active for does not affect the prediction quality of the CPHA model, while a greater number of active days improves the performance of the RNNSMA. The last bar in both charts represents all users with 64 or more active days.

regression and overcomes the limitation of RNNs by including censored observations. We extended the RMTPP model to include any number of session markers and developed a method of training the model using censored observations. We further demonstrated how to include users' absence times. The RNNSM successfully learns from sequences of sessions and outperforms all other models in predicting which users are not going to return (non-returning AUC).

References

1. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks **5**(2), 157–166 (1994)
2. Benson, A.R., Kumar, R., Tomkins, A.: Modeling user consumption sequences. In: WWW'16. pp. 519–529 (2016)
3. Breslow, N.: Covariance analysis of censored survival data. Biometrics **30**(1), 89–99 (1974)
4. Cai, X., Zhang, N., Venayagamoorthy, G.K., Wunsch, D.C.: Time series prediction with recurrent neural networks trained by a hybrid PSO–EA algorithm. Neurocomputing **70**(13), 2342–2353 (2007)
5. Chamberlain, B.P., Cardoso, A., Liu, C.H.B., Pagliari, R., Deisenroth, M.P.: Customer lifetime value prediction using embeddings. In: KDD'17. pp. 1753–1762. ACM (2017)
6. Chandra, R., Zhang, M.: Cooperative coevolution of elman recurrent neural networks for chaotic time series prediction. Neurocomputing **86**, 116–123 (2012)
7. Cheng, H.T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., Shah, H.: Wide & deep learning for recommender systems. In: DLRS 2016 (RecSys'16). pp. 7–10. ACM (2016)
8. Cho, K., van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259 (2014)
9. Chollet, F., et al.: Keras. https://github.com/fchollet/keras (2015)
10. Covington, P., Adams, J., Sargin, E.: Deep neural networks for youtube recommendations. In: RecSys'16. pp. 191–198. ACM (2016)
11. Cox, D.R.: Regression Models and Life-Tables. Journal of the Royal Statistical Society. Series B (Methodological) **34**(2), 187–220 (1972)

12. Cox, D.R.: Partial likelihood. Biometrika **62**(2), 269–276 (1975)
13. Cox, D.R., Oakes, D.: Analysis of Survival Data, vol. 21. CRC Press (1984)
14. Davidson-Pilon, C.: Lifelines (2016), https://github.com/camdavidsonpilon/lifelines
15. Du, N., Dai, H., Trivedi, R., Upadhyay, U., Gomez-Rodriguez, M., Song, L.: Recurrent marked temporal point processes: Embedding event history to vector. In: KDD'16. pp. 1555–1564. ACM (2016)
16. Du, N., Wang, Y., He, N., Song, L.: Time-sensitive recommendation from recurrent user activities. In: NIPS'15. pp. 3492–3500. MIT Press (2015)
17. Efron, B.: The efficiency of cox's likelihood function for censored data. Journal of the American Statistical Association **72**(359), 557–565 (1977)
18. Flunkert, V., Salinas, D., Gasthaus, J.: Deepar: Probabilistic forecasting with autoregressive recurrent networks. arXiv preprint arXiv:1704.04110 (2017)
19. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence **31**(5), 855–868 (2009)
20. Han, M., Xi, J., Xu, S., Yin, F.L.: Prediction of chaotic time series based on the recurrent predictor neural network. IEEE Transactions on Signal Processing **52**(12), 3409–3416 (2004)
21. Harrell, F.E., Lee, K.L., Mark, D.B.: Multivariable prognostic models: Issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. Statistics in Medicine **15**, 361–387 (1996)
22. Hawkes, A.G.: Spectra of Some Self-Exciting and Mutually Exciting Point Processes. Biometrika **58**(1), 83–90 (1971)
23. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
24. Ishwaran, H., Kogalur, U.B., Blackstone, E.H., Lauer, M.S.: Random survival forests. Ann. Appl. Stat. **2**(3), 841–860 (2008)
25. Kalbfleisch, J.D., Prentice, R.L.: Marginal likelihoods based on cox's regression and life model. Biometrika **60**(2), 267–278 (1973)
26. Kapoor, K., Sun, M., Srivastava, J., Ye, T.: A hazard based approach to user return time prediction. In: KDD'14. pp. 1719–1728. ACM (2014)
27. Klein, J.P., Moeschberger, M.L.: Survival analysis: techniques for censored and truncated data. Springer Science & Business Media (2005)
28. Li, L., Jing, H., Tong, H., Yang, J., He, Q., Chen, B.C.: NEMO: Next career move prediction with contextual embedding. In: WWW '17 Companion. pp. 505–513 (2017)
29. Manzoor, E., Akoglu, L.: Rush!: Targeted time-limited coupons via purchase forecasts. In: KDD'17. pp. 1923–1931. ACM (2017)
30. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
31. Rajesh, R., Perotte, A., Elhadad, N., Blei, D.: Deep Survival Analysis. In: Proceedings of the 1st Machine Learning for Healthcare Conference. pp. 101–114 (2016)
32. Rodríguez, G.: Survival Models. In: Course Notes for Generalized Linear Statistical Models (2010), http://data.princeton.edu/wws509/notes/c7.pdf
33. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: NIPS'14. pp. 3104–3112 (2014)
34. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: A neural image caption generator. In: CVPR'15. pp. 3156–3164 (2015)
35. Wangperawong, A., Brun, C., Laudy, O., Pavasuthipaisit, R.: Churn analysis using deep convolutional neural networks and autoencoders. arXiv preprint arXiv:1604.05377 (2016)