Modeling Progression Free Survival in Breast Cancer with Tensorized Recurrent Neural Networks and Accelerated Failure Time Models

Yinchong Yang YINCHONG.YANG@SIEMENS.COM
Ludwig Maximilians University Munich
Siemens AG, Corporate Technology, Munich

Peter A. Fasching PETER.FASCHING@UK-ERLANGEN.DE
Department of Gynecology and Obstetrics,
University Hospital Erlangen, Erlangen

Volker Tresp VOLKER.TRESP@SIEMENS.COM
Ludwig Maximilians University Munich
Siemens AG, Corporate Technology, Munich

Abstract

In this work we attempt to predict the progression-free survival time of metastatic breast cancer patients by combining state-of-the-art deep learning approaches with traditional survival analysis models. In order to tackle the challenge of sequential clinical records being both high-dimensional and sparse, we propose to apply a tensorized recurrent neural network architecture to extract a latent representation from the entire patient history. We use this as the input to an Accelerated Failure Time model that predicts the survival time. Our experiments, conducted on a large real-world clinical dataset, demonstrate that the tensorized recurrent neural network largely reduces the number of weight parameters and the training time. It also achieves modest improvements in prediction, in comparison with state-of-the-art recurrent neural network models enhanced with event embeddings.

1. Introduction

With the increasing availability of Electronic Health Records (EHR) data in clinics, there is a growing interest in predicting treatment prescription and individual patient outcome by extracting information from these data using advanced analytics approaches. Especially the latest success of deep learning in image and natural language processing has encouraged the application of these state-of-the-art techniques in modeling clinical data as well. Convolutional Neural Networks (CNNs), particularly the deeper architectures with multiple layers, turn out to be capable of not only handling natural images (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; Szegedy et al., 2015), but also medical imaging data for, e.g., segmentation and captioning (Kayalibay et al., 2017; Shin et al., 2016; Kisilev et al., 2011). On the other hand, due to the fact that natural language and medical records share the same sequential nature, Recurrent Neural Networks (RNNs), which have proven to be powerful in language modeling (Kim et al., 2015; Mikolov, 2012) and machine translation (Sutskever et al., 2014; Cho et al., 2014), are more frequently applied to medical event data to predict, e.g., medication prescription and patient endpoints, with promising per-

formances (Choi et al., 2016, 2015). The major advantage of RNNs in handling medical records lies in their ability to handle sequential inputs of variable lengths in a more generic way than sliding window approaches (Bengio et al., 2003; Esteban et al., 2015).

In language modeling, the input to the RNN model is typically a word (or even character) embedding in form of a dense and real vector that represents the word in a latent space. The embedding idea has inspired its application on medical event data such as Choi et al. (2015); Esteban et al. (2016); Choi et al. (2016); the reason is that the sequential medical records are often of categorical type and, after binary-coding, the derived feature vector can become very sparse in a high dimensional space. Choi et al. (2015) terms such type of input features as a multi-hot vector and show that they are suboptimal to serve as direct inputs to plain RNN models. The embedding layer, though reducing the size of the input-to-hidden weight matrix in an RNN model, still requires parameters determined by the dimension of the raw multi-hot input and the embedding size.

To handle the high dimensional multi-hot sequential input, we propose in this work a simpler, more efficient and direct approach by factorizing the input-to-hidden weight matrix in an RNN model based on tensor factorization. Novikov et al. (2015) first proposed to perform Tensor-Train Factorization (Oseledets, 2011) on weight matrices in Neural Networks to tackle the challenge of input redundancy, such as in fully-connected layers following a convolutional one, or simple feed-forward layers that consume raw pixel data. This tensorization is proven to be highly efficient and can significantly reduce the over-parameterization in the weights without sacrificing much of the model expressiveness. Following their work, we argue that the same challenge of redundant information in high dimensional and sequential multi-hot vectors can be addressed in a similar way by integrating the Tensor-Train factorization into RNNs, with the nice property that the factorization is learned efficiently, together with the rest of the RNN, in an end-to-end fashion.

We conduct experiments on a large real-world dataset consisting of thousands of metastatic breast cancer patients from Germany. We attempt to predict for each patient her/his Progression-Free Survival time based on her/his medical history of variable length and background information, using an Accelerated Failure Time (AFT) model in combination with the RNN models. The former (Wei, 1992) has always been a promising tool for survival analysis, especially if one wants to directly model the individual survival time instead of the hazard. Gore et al. (1984) and Bradburn et al. (2003), for instance, have demonstrated the application of AFT in modeling progression-free survival in case of breast and lung cancer, respectively.

The rest of the paper is organized as follows: in Sec. 2 we give an overview of related works that have inspired our own; in Sec. 3 we provide an introduction to our cohort and data situation; in Sec. 4 we present the novel techniques of our model in detail and provide experimental results in Sec. 5. Finally, Sec. 6 provides a conclusion and an outlook on future works.

2. Related Work

Handling sequential EHR data. Due to the sequential nature of EHR data, there have been recently multiple promising works studying clinical events as sequential data. Many of them were inspired by works in natural language modeling, since sentences can

be easily modeled as sequences of signals. Esteban et al. (2015) adjusted a language model based on the sliding window technique in Bengio et al. (2003), taking into account a fixed number of events in the past. This model was extended in Esteban et al. (2016) by replacing the sliding window with RNNs, which improved the predictions for prescriptions decision and endpoints. Lipton et al. (2015) applied LSTM to perform diagnosis prediction based on sequential input. A related approach with RNNs can also be found in Choi et al. (2015) to predict diagnosis and medication prescriptions. This RNN implementation was further augmented with neural attention mechanism in Choi et al. (2016), which did not only show promising performances but also improved the interpretability of the model.

RNN for sequence classification and regression. The RNN models in these works were implemented in a many-to-many fashion. That is to say, at each time step the RNN generates a prediction as output, since the target in these works is provided at every time step. In our work, on the other hand, cancer progression is not expected to be observed regularly. Consequently, we implement many-to-one RNN models that consume a sequence of input vectors, and generate only one output vector. This setting can be found in a variety of sequence classification/regression tasks. Koutnik et al. (2014) used such RNN architectures to classify spoken words and handwriting as sequences. RNNs have also been applied to classify the sentiment of sentences such as in the IMDB reviews dataset (Maas et al., 2011). The application of RNNs in the many-to-one fashion can also be seen as to encode a sequence of variable length into one fixed-size representation (Sutskever et al., 2014) and then to perform prediction as decoding based on this representation.

Tensor-Train Factorization and Tensor-Train Layer. The *Tensor-Train Factorization* (TTF) was first introduced by Oseledets (2011) as a tensor factorization model with the advantage of being capable of scaling the factorization to an arbitrary number of dimensions. Novikov et al. (2015) showed that one could reshape a fully connected layer into a high dimensional tensor, which is to be factorized using TTF, and referred to it as a *Tensor-Train Layer* (TTL). This idea was applied to compress very large weight matrices in deep Neural Networks where the entire model was trained end-to-end. In these experiments they compressed fully connected layers on top of convolutional layers, and also proved that a Tensor-Train layer can directly consume pixels of image data such as CIFAR-10, achieving the best result among all known non-convolutional layers. Then in Garipov et al. (2016) it was shown that even the convolutional layers themselves can be compressed with TTLs. Yang et al. (2017) first applied TTF to the input-to-hidden layer in RNN variants. They showed that such modification not only reduced the number of weight parameters in orders of magnitude, but also could boost prediction accuracy in classification of real-world video clips, which typically involve extremely large input dimensions.

3. Cohort

3.1 Data Extraction

In Germany, approximately 70,000 women suffer from breast cancer and the mortality is approximately 33% every year (Kaatsch et al., 2013; Rauh and Matthias, 2008). In many of these cases, it is the progression, especially the metastasis of the cancer cells to vital organs, that actually causes the patient's death. Our dataset, provided by the PRAEGNANT study network (Fasching et al., 2015), was collected on patients suffering from metastatic

breast cancer and warehoused in the secuTrial®, which is a relational database system. After querying and pre-processing, we could retrieve information on 4,357 valid *patient cases*, which we define as a patient-time pair, i.e., at that specific time a metastasis and/or recurrence is observed on that patient. The first patient was recruited in 2014 and the currently last patient in 2016, but their earliest medical records date back to 1961.

3.2 Feature Processing

There are two classes of patient information that are potentially relevant for modeling the PFS time:

First, the *static* information includes 1) basic patient information, 2) information on the primary tumor and 3) history of metastasis before entering the study. In total we observe 26 features of binary, categorical or real types. We perform binary-coding on the former both cases and could extract for each patient case i a static feature vector denoted with $\boldsymbol{m}_i \in \mathbb{R}^{118}$.

The *sequential* information includes 4) local recurrences, 5) metastasis 6) clinical visits 7) radiotherapies, 8) systemic therapies and 9) surgeries. These are time-stamped clinical events observed on each patient throughout time. In total we have 26 sequential features of binary or categorical nature. Binary-coded, they yield for a patient case i at time step t a feature vector $\boldsymbol{x}_i^{[t]} \in \{0,1\}^{196}$. We denote the whole sequence of events for this patient case i up to her/his last observation at T_i before progression (in form of either local recurrences or metastasis) using a set of $\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}$. The length of the sequences vary from 1 to 35 and is on average 7.54.

Due to the binary-coding, each $\boldsymbol{m}_i$ and $\boldsymbol{x}_i^{[t]}$ yield on average sparsities of 0.88 and 0.97, respectively, for each i and t.

3.3 Distribution of the Target Variable

We denote the number of days till the next recorded progression using z_i, and assume it to be a realization of a random variable Z_i to serve as the target variable of our model.

In total, we attempt to model the distribution of PFS time as Generalized Regression Model (GRM):

$$Z_i \overset{i.i.d.}{\sim} \mathcal{F}(\Theta_i) \quad \text{where} \quad \Theta_i = g^{-1}(\boldsymbol{\eta}_i^{[T_i]}). \tag{1}$$

where Z_i is a variable independently conditioned on some features that represent the patient case, where g^{-1} is analogous to the inverse link function in GRMs and maps the time dependent input $\boldsymbol{\eta}_i^{[T_i]}$ to the set of distribution parameters Θ_i.

We would assume Z_i to be Log-Normal distributed and specify Eq. (1) to be:

$$Z_i \overset{i.i.d.}{\sim} \log\mathcal{N}(\mu_i, \sigma_i) \text{ with } \mu_i = g^{-1}(\boldsymbol{\eta}_i^{[T_i]}) = g^{-1}\left(f\left(\boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}\right)\right). \tag{2}$$

In the section that follows we elaborate in detail the construction of the function f, which maps the static and sequential features of a patient case into a latent representation $\boldsymbol{\eta}_i^{[T_i]}$ to serve as the input to the AFT model, a special case of GRM.

4. Methods

In this section, we first give an introduction to the Tensor-Train Layer; we then use this to replace the weight matrix mapping from the input vector to hidden state in RNN models. Thereafter, we briefly review the Accelerated Failure Time model that consumes the latent patient representation produced by the Tensor-Train RNN.

4.1 Tensor-Train Recurrent Neural Networks

Tensor-Train Factorization of a Feed-Forward Layer Novikov et al. (2015) showed that the weight matrix $\boldsymbol{W}$ of a fully-connected feed-forward layer $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ can be factorized using Tensor-Train (Oseledets, 2011), where the factorization and the weights are learned simultaneously. This modification is especially effective, if the layer is exposed to high-dimensional input with redundant information, such as pixel input and feature maps produced by convolutional layers. In this work, we show that the challenge of multi-hot representation of sequential EHR data can also be handled using TTL by integrating it into RNNs.

A fully-connected feed-forward layer, in form of:

$$\hat{\boldsymbol{y}}(j) = \sum_{i=1}^{M} \boldsymbol{W}(i,j) \cdot \boldsymbol{x}(i) + \boldsymbol{b}(j), \ \forall j \in [1, N] \text{ with } \ \boldsymbol{x} \in \mathbb{R}^M, \ \boldsymbol{y} \in \mathbb{R}^N, \tag{3}$$

can be equivalently rewritten as

$$\begin{aligned} \widehat{\boldsymbol{\mathcal{Y}}}(j_1, j_2, ..., j_d) = \sum_{i_1=1}^{m_1} \sum_{i_2=1}^{m_2} ... \sum_{i_d=1}^{m_d} \boldsymbol{\mathcal{W}}((i_1, j_1), (i_2, j_2), ..., (i_d, j_d)) \cdot \boldsymbol{\mathcal{X}}(i_1, i_2, ..., i_d) \\ + \boldsymbol{\mathcal{B}}(j_1, j_2, ..., j_d), \end{aligned} \tag{4}$$

so long as the input and output dimensions can be factorized as $M = \prod_{k=1}^{d} m_k, \quad N = \prod_{k=1}^{d} n_k$. Therefore, the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are reshaped into two tensors with the same number of dimensions: $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m_1 \times m_2 \times ... \times m_d}, \boldsymbol{\mathcal{Y}} \in \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$, respectively, and the mapping function becomes $\mathbb{R}^{m_1 \times m_2 \times ... \times m_d} \rightarrow \mathbb{R}^{n_1 \times n_2 \times ... \times n_d}$. Though mathematically equivalent, Eq. 4 represents a more general description of Eq. 3 and, more importantly, provides a high dimensional weight *tensor* of $\boldsymbol{\mathcal{W}}$ that can be factorized using TTF:

$$\widehat{\boldsymbol{\mathcal{W}}}((i_1, j_1), (i_2, j_2), ..., (i_d, j_d)) \overset{TTF}{=} \boldsymbol{\mathcal{G}}_1^*(i_1, j_1, :, :) \ \boldsymbol{\mathcal{G}}_2^*(i_2, j_2, :, :) \ ... \ \boldsymbol{\mathcal{G}}_d^*(i_d, j_d, :, :), \tag{5}$$

where a $\boldsymbol{\mathcal{G}}_k^* \in \mathbb{R}^{m_k \times n_k \times r_{k-1} \times r_k}$ is termed as a core tensor, specified by ranks r_k for $k \in [0, d]$. Now instead of explicitly storing the full tensor $\boldsymbol{\mathcal{W}}$ of size $\prod_{k=1}^{d} m_k \cdot n_k = M \cdot N$, we only store its TT-format, i.e., the set of low-rank core tensors $\{\boldsymbol{\mathcal{G}}_k^*\}_{k=1}^{d}$ which can approximately reconstruct $\boldsymbol{\mathcal{W}}$.

For the rest of the paper, we denote a fully-connected layer of $\hat{\boldsymbol{y}} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, whose weight matrix $\boldsymbol{W}$ is factorized with TTF as $\hat{\boldsymbol{y}} = TTL(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x})$, or $TTL(\boldsymbol{W}, \boldsymbol{x})$, if no bias is required.

Tensor-Train RNN In this work we investigate the challenge of modeling high dimensional sequential data with RNNs. For this reason, we factorize the matrix mapping from

the input to the hidden state with a TTL as in Yang et al. (2017). More specifically, in case of LSTM, a particular form of RNN by Hochreiter and Schmidhuber (1997); Gers et al. (2000) and GRU, another variant by Chung et al. (2014), we TT-factorize the matrices that map from the input vector to the gating units as in Eq. (6):

$$
\begin{aligned}
&\text{TT-GRU:} && \text{TT-LSTM:} \\
&\boldsymbol{r}^{[t]} = \text{sig}(TTL(\boldsymbol{W}^r, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^r \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^r) && \boldsymbol{k}^{[t]} = \text{sig}(TTL(\boldsymbol{W}^k, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^k \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^k) \\
&\boldsymbol{z}^{[t]} = \text{sig}(TTL(\boldsymbol{W}^z, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^z \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^z) && \boldsymbol{f}^{[t]} = \text{sig}(TTL(\boldsymbol{W}^f, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^f \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^f) \\
&\boldsymbol{d}^{[t]} = \tanh(TTL(\boldsymbol{W}^d, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^d (\boldsymbol{r}^{[t]} \circ \boldsymbol{h}^{[t-1]})) && \boldsymbol{o}^{[t]} = \text{sig}(TTL(\boldsymbol{W}^o, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^o \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^o) \\
&\boldsymbol{h}^{[t]} = (1 - \boldsymbol{z}^{[t]}) \circ \boldsymbol{h}^{[t-1]} + \boldsymbol{z}^{[t]} \circ \boldsymbol{d}^{[t]}, && \boldsymbol{g}^{[t]} = \tanh(TTL(\boldsymbol{W}^g, \boldsymbol{x}^{[t]}) + \boldsymbol{U}^g \boldsymbol{h}^{[t-1]} + \boldsymbol{b}^g) \\
& && \boldsymbol{c}^{[t]} = \boldsymbol{f}^{[t]} \circ \boldsymbol{c}^{[t-1]} + \boldsymbol{k}^{[t]} \circ \boldsymbol{g}^{[t]} \\
& && \boldsymbol{h}^{[t]} = \boldsymbol{o}^{[t]} \circ \tanh(\boldsymbol{c}^{[t]}).
\end{aligned} \tag{6}
$$

For the sake of simplicity, we denote a many-to-one RNN (either with or without Tensor-Train, either GRU or LSTM) using a function ω: $\boldsymbol{h}_i^{[T_i]} = \omega(\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i})$, where $\boldsymbol{h}_i^{[T_i]}$ is the last hidden state as in Eq. (6).

In order to also take into account the static features (Esteban et al., 2016) such as patient background and primary tumor, we concatenate the last hidden state of the RNN with the latent representation of the static features as $\boldsymbol{\eta}_i^{[T_i]} = [\boldsymbol{h}_i^{[T_i]}, \boldsymbol{q}_i]$ with $\boldsymbol{q}_i = \psi(\boldsymbol{V}\boldsymbol{m}_i)$, where $\boldsymbol{V}$ is a standard trainable weight matrix and ψ denotes a non-linear activation function. Finally, we have specified the function f with respect to Eq. (2) as:

$$
g(\mu_i) = \boldsymbol{\eta}_i^{[T_i]} = f\left(\boldsymbol{m}_i, \{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i}\right) = \left[\psi(\boldsymbol{V}\boldsymbol{m}_i),\ \omega(\{\boldsymbol{x}_i^{[t]}\}_{t=1}^{T_i})\right]. \tag{7}
$$

The vector $\boldsymbol{\eta}_i^{[T_i]}$ represents the static patient information as well as the medical history of patient i up to time step T_i. In the context of GRM-like models, $\boldsymbol{\eta}_i^{[T_i]}$ would be the raw *covariates*, while in our case, it is a more abstract *latent representation* generated from a variety of raw and potentially less structured features. This point of view provides us with an interface between representation learning and the GRM models.

The vector $\boldsymbol{\eta}_i^{[T_i]}$ also functions as an abstract patient profile that represents all relevant clinical information in a latent vector space, where patients with similar background information and medical history would be placed in a specific neighborhood. This very characteristics of the latent vector space is key to the latest success of deep or representation learning, because it facilitates the classification and regression models built on top of it, which is, in our case, an AFT model that is presented as follows.

4.2 Accelerated Failure Time Model

The AFT model is a GRM-like parametric regression that attempts to capture the influence that the features in $\boldsymbol{\eta}_i$ have on a variable Z_i, which describes the survival time till an event is observed:

$$
Y_i = \boldsymbol{\eta}_i^T \boldsymbol{\beta} + R_i, \text{ with } Y_i = \log(Z_i), \tag{8}
$$

where $\boldsymbol{\beta}$ is the weight vector and $R_i \overset{i.i.d.}{\sim} \mathcal{D}_R$ is the residual whose distribution can be specified by $\mathcal{D}_R$. Common choices for $\mathcal{D}_R$ are Normal, Extreme Value and Logistic distributions. The variable Z would correspondingly be Log-Normal, Weibull/Exponential and

Log-Logistic distributed, respectively. As Eq. (8) suggests, an AFT model assumes that the covariates have a multiplicative effect on the survival time, 'accelerating' —either positively or negatively— the baseline time till which the event of interest occurs. To see that one only has to rewrite Eq. (8) as $Z_i = Z_0 \exp\left(\boldsymbol{\eta}_i^T \boldsymbol{\beta}\right)$ with $Z_0 = \exp\left(R_i\right)$, so that the baseline survival time Z_0 is accelerated to a factor of $\exp\{\boldsymbol{\eta}_i^T \boldsymbol{\beta}\}$. In other words, if one covariate j increases by a factor of δ, the failure time is to be accelerated by a factor of $\exp\{\delta\beta_j\}$, so long as all other covariates remain the same. The Proportional Hazard Cox Regression, on the other hand, assumes such a multiplicative effect over the hazard.

We are specifically assuming that the target variable Z_i follows a Log-Normal distribution and that, implicitly, the residuals are normal distributed, i.e., $R_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma_i)$. One could therefore plug Eq. (8) into the normal distribution assumption and have $\frac{\log(Z_i) - \boldsymbol{\eta}_i^T \boldsymbol{\beta}}{\sigma_i} \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$, which allows us to perform training with the Mean Squared Logarithmic Error.

5. Experiments

5.1 Experimental Details and Evaluation Approach

We conduct 5-fold cross-validations by splitting the dataset into proportions of 0.8 / 0.2 disjoint subsets for training / test tasks. We train our models with 0.25 dropout (Srivastava et al., 2014) rate for the weights in (TT-)RNNs, and 0.025 ridge-penalization in feed-forward layers, with the Adam (Kingma and Ba, 2014) step rule for 200 epochs. Since the dimension of 196 can be represented with prime factors $2^2 \times 7^2$, we experiment two different settings of $[7, 28]$ and $[4, 7, 7]$ to factorized the input dimension. The corresponding factorizations of the RNN output's dimension are $[8, 8]$ and $[4, 4, 4]$, respectively. In the first case, the Tensor-Train-Factorization becomes equivalent to a two mode PARAFAC/CP (Kolda and Bader, 2009) model, and we therefore denote GRU and LSTM with this setting as CP-GRU and CP-LSTM, respectively, for the sake of simplicity. We also experiment two different TT-ranks of 4 and 6. All models are trained with objective function of Mean Squared Logarithmic Error (Chollet, 2015). We set the size of the non-linear mapping of the static feature $\boldsymbol{q}_i$ to be 128.

Since the target Z follows a Log-Normal instead of Normal distribution, it is not appropriate to measure the results in term of Mean Squared Error (MSE) as is with usual regression models. We therefore report the more robust metric of Median Absolute Error (MAE) defined as $MAE = median_i(|z_i - \hat{z}_i|)$. Even on the logarithmic scale of Y, MSE is not a reliable metric in this case either, since the Squared Error of a $y_i = \log z_i$ would increase in Z exponentially as y_i increases. In other words, two similar Squared Errors on the logarithmic scale Y might imply totally different errors in Z. We therefore report the coefficient of determination as $R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$ on the logarithmic scale of Y.

All our models are implemented in Theano (Bastien et al., 2012) and deployed in Keras (Chollet, 2015). The experiments were conducted on a NVIDIA Tesla K40c Processor.

5.2 Prediction of Progression-Free Survival

As weak baselines we first report the performance of standard Cox and AFT Regression using the R package `survival`(Therneau, 2015; Terry M. Therneau and Patricia M. Gramb-

Model	TT-Rank	MAE	R^2	Time(sec.)	#Parameters
GRU	-	156.7 ± 6.6	0.295 ± 0.018	3,598	74,880
LSTM	-	159.5 ± 13.7	0.274 ± 0.014	5,956	99,840
Emb.+GRU	-	136.2 ± 14.8	0.635 ± 0.009	4,957	24,832
Emb.+LSTM	-	136.8 ± 11.9	0.633 ± 0.014	6,170	28,928
CP-GRU	4	135.6 ± 10.0	0.630 ± 0.018	**1,689**	1,568
	6	136.8 ± 9.3	0.623 ± 0.015	1,773	2,352
TT-GRU	4	136.5 ± 9.4	0.632 ± 0.020	1,940	**752**
	6	136.2 ± 11.4	0.634 ± 0.019	2,178	1,464
CP-LSTM	4	135.2 ± 8.4	0.637 ± 0.018	3,390	1,792
	6	133.7 ± 10.7	0.625 ± 0.023	3,530	2,688
TT-LSTM	4	140.0 ± 10.4	**0.645**± 0.025	3,729	816
	6	**132.9** ± 9.9	0.630 ± 0.017	4,050	1,560

Table 1: Experimental results: average MAE, R^2, average training time, the number of all parameters responsible for mapping the raw input to the hidden state in RNNs.

sch, 2000). The Cox Regression (in term of median survival estimate in the package) yields an average MAE score over the cross-validation of 214.5 and the AFT 208.7. The input to both models are the raw features aggregated w.r.t time axis. Such aggregation is also applied in Esteban et al. (2015) and is proven to be a reasonable alternative solution to handle time stamped features, since each entry represents the number of feature values observed up to a specific time step, though ignoring the order in which the events were observed.

We apply two further classes of baseline models: First, we expose GRU and LSTM directly to the raw sequential features and then, secondly, we add a *tanh* activated embedding layer of size 64, between the raw input feature and the RNNs, following the state-of-the-art of Choi et al. (2015). The corresponding results are presented in the first four rows in Tab. 1. Compared to aggregated sequential features, RNNs are indeed able to generate representations that facilitate the AFT model on top of them, reducing the MAE from over 200 to ca. 150. The embedding layers as input to RNNs turn out to yield even better results of 136, with less parameters but longer training time. This confirms the point made in Choi et al. (2015) that such input features of both high dimensionality and sparsity are suboptimal input to RNNs.

In further experiments we test our TT-GRU, TT-LSTM, CP-GRU and CP-LSTM implementations with different TT-ranks. Though exposed to the raw features, these tensorized RNNs yield prediction quality comparable with –and sometimes even better than– the state-of-the-art embedding technique and require on average ca. 40% of the training time and $2\% - 10\%$ of parameters. Compared with plain RNNs, they merely require 1% to 3% of the parameters.

In contrast to Novikov et al. (2015) where tensorization slightly decreased the prediction quality, we actually observe on average a modest improvement with TT-RNNs. For instance, the MAE is improved from 136.2 to 135.6 in case of GRU and from 136.8 to 132.9 in case of LSTM. Such MAE measures around 135 implies that these model can provide a prediction of PFS time with an accuracy of plus-minus four months time for the non-extreme patient cases.

$\mathcal{F}_0$	Weibull	Exponential	Log-Logistic	Log-Normal
p-value	3.7e-4	$\leq$2.2e-16	$\leq$2.2e-16	0.064

Table 2: Two-sided Kolmogorov-Smirnov Tests of the distribution of the residual under the H_0 of variable distributions $\mathcal{F}_0$.

Please note that in an RNN model, the input-to-hidden weight matrix becomes over-parameterized if the input feature turns out to be highly sparse and/or to consist of high proportion of redundant information. In earlier works, this challenge is tackled using an explicit embedding layer that transforms the raw feature into a more compact input vector to the RNNs. The TT-RNNs, on the other hand, provide an alternative solution that directly tackles the over-parameterization in the weight matrix, in that the full-sized weight matrix is constructed using a much smaller number of *'meta'* weights, i.e., the core tensors in the Tensor-Train model.

Secondly, comparing different TT-settings, it is trivial that a higher TT-rank requires more parameters and longer training time. Compared with the CP-factorization of $196 = 7 \times 28$, the real TT-factorization of $196 = 4 \times 7 \times 7$ of the weight matrix leads to a smaller number of parameters, but the difference in training time is less extreme. This can be explained with Eq. (5), which demonstrates that the number of core tensors also influences the computation complexity. More specifically, the multiplication among core tensors is strictly successive in $k = 1, 2, ..., d$ and cannot be parallelized in CPUs or GPUs. In other words, a chain of multiplication of small core tensors might, in extreme cases, take even longer to compute than the multiplication of two large matrices.

In order to verify the Log-Normal distribution assumption, we conduct Kolmogorov-Smirnov-Tests as in R Core Team (2016) on the modeling residuals. We report in Tab. 2 the p-values corresponding to alternative distribution assumptions. The hypothesis of Weibull-, Exponential and Log-Logistic-distribution can be therefore rejected with rather high significance. Since we cannot reject the Log-Normal distribution assumption even with the largest common significant level of 5%, our assumption of Z_i to be Log-Normal distributed in Eq. (1) can be verified.

5.3 Estimation of Individual Survival and Hazard Functions

Beside the PFS time, the AFT model also allows us to calculate individual survival and hazard functions for each patient case. The individual survival and hazard function can be derived from Eq. (8) and takes the form of:

$$S(Z_i = z|\boldsymbol{\eta}_i) = 1 - \Phi\left(\frac{\log(z) - \boldsymbol{\eta}_i^T\boldsymbol{\beta}}{\sigma_i}\right), \tag{9}$$

$$\lambda(Z_i = z|\boldsymbol{\eta}_i) = -\frac{\partial}{\partial z}S_i(z|\boldsymbol{\eta}_i) = \frac{\phi\left(\frac{\log(z)-\boldsymbol{\eta}_i^T\boldsymbol{\beta}}{\sigma_i}\right)}{\left(1 - \Phi\left(\frac{\log(z)-\boldsymbol{\eta}_i^T\boldsymbol{\beta}}{\sigma_i}\right)\right) z\sigma_i}, \tag{10}$$

where Φ and ϕ are the cumulative distribution function and probability density function of a standard normal distribution, respectively. Here σ_i can be estimated using

$$\sigma_i \approx \hat{\sigma}_i = \mathbb{V}_{i^* \sim data_{train}}\left(\log(Z_{i^*}) - \boldsymbol{\eta}_{i^*}^T\boldsymbol{\beta}\right)^{\frac{1}{2}}, \tag{11}$$

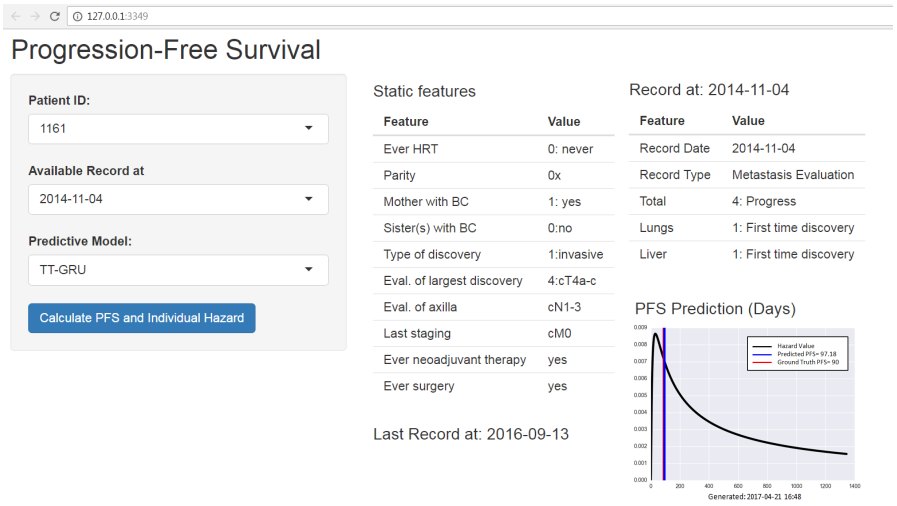

Figure 1: A prototype App for patient data querying and PFS prediction, implemented with R Shiny Chang et al. (2015), illustrated with information of a patient from our test set.

under the conditional i.i.d. assumption in Eq. (1). In a realistic application scenario in clinics, providing the individual hazard function is as important as providing physicians with a prediction of the PFS time. For the clinic we have developed a prototype interface (Fig. 1) to predict the PFS time and individual hazard function as well as to query patient information. For illustrative purposes, we calculate hazard function for a patient from test set so that, for comparison, we also mark the ground truth PFS.

6. Conclusion and Future Works

We have first applied tensorized RNNs to handle high dimensional sequential inputs in form of medical history data; Second, we showed that one can join deep/Representation Learning with GRM-like models in a Encoder-Decoder fashion (Sutskever et al., 2014), where the RNN encoder produces better representative input for the GRM-like decoder. Our empirical results demonstrate that the tensorized RNNs greatly reduce the number of parameters and the training time of the model, while retaining –if not improving– the prediction quality compared to the state-of-the-art embedding technique. We also showed that when applying an AFT model on top of RNNs, one could calculate the PFS time as well as provide physicians with individual survival and hazard functions, improving the usability of our model in realistic scenarios. In the future we would like to integrate attention mechanisms as Choi et al. (2016) into the TT-RNNs, in order to improve the model's interpretability. This would enables the physician to trace back to event(s) that have contributed most to the prediction. In this work we implicitly reshape our multi-hot input solely for computational convenience. We find it therefore necessary to explore possibilities to perform the reshaping in accordance with the actual input structure. Furthermore it seems also appealing to include other distribution assumptions for the AFT model.

References

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *journal of machine learning research*, 3(Feb):1137–1155, 2003.

Mike J Bradburn, Taane G Clark, SB Love, and DG Altman. Survival analysis part ii: multivariate data analysis-an introduction to concepts and methods. *The British Journal of Cancer*, 89(3):431, 2003.

Winston Chang, Joe Cheng, J Allaire, Yihui Xie, and Jonathan McPherson. Shiny: web application framework for r. *R package version 0.11*, 1, 2015.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Edward Choi, Mohammad Taha Bahadori, and Jimeng Sun. Doctor ai: Predicting clinical events via recurrent neural networks. *arXiv preprint arXiv:1511.05942*, 2015.

Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *Advances in Neural Information Processing Systems*, pages 3504–3512, 2016.

François Chollet. Keras: Deep learning library for theano and tensorflow. `https://github.com/fchollet/keras`, 2015.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Cristóbal Esteban, Danilo Schmidt, Denis Krompaß, and Volker Tresp. Predicting sequences of clinical events by using a personalized temporal latent embedding model. In *Healthcare Informatics (ICHI), 2015 International Conference on*, pages 130–139. IEEE, 2015.

Cristóbal Esteban, Oliver Staeck, Yinchong Yang, and Volker Tresp. Predicting clinical events by combining static and dynamic information using recurrent neural networks. *arXiv preprint arXiv:1602.02685*, 2016.

P.A. Fasching, S.Y. Brucker, T.N. Fehm, F. Overkamp, W. Janni, M. Wallwiener, P. Hadji, E. Belleville, L. Häberle, F.A. Taran, D. Luftner, M.P. Lux, J. Ettl, V. Muller, H. Tesch, D. Wallwiener, and A. Schneeweiss. Biomarkers in patients with metastatic breast cancer and the praegnant study network. *Geburtshilfe Frauenheilkunde*, 75(01):41–50, 2015.

Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Sheila M Gore, Stuart J Pocock, and Gillian R Kerr. Regression models and non-proportional hazards in the analysis of breast cancer survival. *Applied Statistics*, pages 176–195, 1984.

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

Peter Kaatsch, Claudia Spix, Stefan Hentschel, Alexander Katalinic, Sabine Luttmann, Christa Stegmaier, Sandra Caspritz, Josef Cernaj, Anke Ernst, Juliane Folkerts, et al. Krebs in deutschland 2009/2010. 2013.

Baris Kayalibay, Grady Jensen, and Patrick van der Smagt. Cnn-based segmentation of medical imaging data. *arXiv preprint arXiv:1701.03056*, 2017.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*, 2015.

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

Pavel Kisilev, Eli Sason, Ella Barkan, and Sharbell Hashoul. Medical image captioning: learning to describe medical image findings using multi-task-loss cnn. 2011.

Tamara G. Kolda and Brett W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 2009.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

Tomas Mikolov. Statistical language models based on neural networks. 2012.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33 (5):2295–2317, 2011.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. URL `https://www.R-project.org/`.

Claudia Rauh and W Matthias. Interdisziplinäre s3-leitlinie für die diagnostik, therapie und nachsorge des mammakarzinoms. 2008.

Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, New York, 2000. ISBN 0-387-98784-3.

Terry M Therneau. *A Package for Survival Analysis in S*, 2015. URL `https://CRAN.R-project.org/package=survival`. version 2.38.

Lee-Jen Wei. The accelerated failure time model: a useful alternative to the cox regression model in survival analysis. *Statistics in medicine*, 11(14-15):1871–1879, 1992.

Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *Proc. of the International Conference on Machine Learning (ICML)*, 2017.