**#Bayesian Learning Artificial Neural Networks for Modeling Survival Data**

**# Motivation**

Accurate predictions of prognostic outcomes are of substantial and pivotal significance in the context of quality care delivery. However, the application of deep learning models to enhance caregiving in healthcare has been limited by concerns related to the reliability of such methods. In this way, models that are robust and which can result in a throughput prediction of such clinical outcomes as survival while at the same time exhibiting high reliability and potential to be generalized to larger populations remain in high demand. As a result, there has been an emerging persistent interest in modeling survival data to leverage the promise deep learning models offer in this regard. This is not surprising given the significance of the healthcare sector, where we are often interested in understanding, for instance, the role that a specific differentially expressed gene plays concerning prognosis or, more generally, understanding how a given treatment regimen is likely to impact patient outcomes and in turn make decisions accordingly to perhaps improve patient outcomes related to care.

**[Margin Figure of some health outcomes]**

Analyzing time-to-event data involves is an inimitable problem given that the outcome of interest might comprise whether or not an event has occurred **(binary outcome)** but also the time when this event occurs (**continuous outcome**) [@feng2021bdnnsurv]. The problem is further complicated by missing data on the survival outcome of interest—**censored data.^[Censoring refers to a concept in survival analysis where the actual time to event is unknown due to such reasons as the loss to follow up, withdrawals, or an exact unknown time of event. In right censoring, the event of interest occurs after the end of the experiment or study, whereas in left censoring, the event occurs before the onset of the study. Interval censoring is when the actual survival time is bounded between some interval]**. The very nature of (censored) survival data makes it impossible to apply classical analysis methods such as **logistic regression**.

Additionally, models based on the Weibull model have restrictive assumptions, including a **parametric form** of the distribution of the time to event. Similarly, the semi-parametric Cox proportional hazards (PH) model [@burden2008bayesian] also has assumptions, a major one being the **proportional hazards assumption:** "the effect of a unit increase in a covariate is multiplicative with respect to the hazard rate." Despite the outcome of interest not always being a hazard rate, it can be a probability; for instance, the PH assumption does not make much sense, especially when we have a substantial number of covariates (we would need each of these covariates to satisfy this assumption). The performance of these methods has also been shown to be poor, especially when the underlying model is incorrectly specified [@feng2021bdnnsurv].

But how can we tackle the problem of modeling survival data amicably? This post reviews an extension of artificial neural networks (ANN) implemented based on the Cox PH model and trained to model survival data using Bayesian learning. In particular, we use a 2-layer feed-forward artificial neural network (ANN) trained using Bayesian inference to model survival outcomes and compare this model to the more traditional Cox proportional hazards model. Compared to previously studied models, we expect the ANN trained using Bayesian inference to perform better following its incorporation of Bayesian inference and neural networks.

First, we introduce \*neural networks in a more general context\*, then discuss \*the neural networks approach to modeling survival data\* and how Bayesian inference has been introduced into these models to enhance their predictive capacity. Next, we introduce an application of the Bayesian learning artificial neural network (BLNN) using an R package of a similar name applied in modeling the effect of identified differentially expressed genes on the survival of patients with primary bladder cancer. Lastly, we compare this model to the more traditional Cox PH model for illustrative purposes and provide an extension code in Python.

**# But what are neural networks?**

With all the hype linked to this deep learning method in the recent past [@**hastie2009elements**], we provide a simplistic idea of what this method is. Defined: Neural networks are:

*> "Computer systems with interconnected nodes designed like neurons to mimic the human brain in terms of intelligence. These networks use algorithms to discover hidden data structures and patterns, correlations, clusters, and classify them, learn and improve over time."*

The idea is to take in simple functions as inputs and then allow these functions to build upon each other. The models are flexible enough to learn non-linear relationships rather than prescribing them as is in kernels or transformations. A neural network takes in an input vector of p features $X=(X\_1, X\_2, \cdots , X\_p)$ and then creates a non-linear function to forecast an outcome variable, $Y$. While varied statistical models such as Bayesian additive regression trees (BART) and random forests exist, neural networks have a structure that contrasts them from these other methods. **Figure 1 shows a feed-forward neural network with an input layer consisting of 4 input features, $X=(X\_1, \cdots, X\_4)$, a single hidden layer with 5 nodes $A\_1, \cdots, A\_5$, a non-linear activation function, $f(X)$ (output layer), and the desired outcome, $Y.$**

**[Figure 10.1 goes here]**

The arrows show that the input layer is feeding into each of the nodes in the hidden layer which in turn feed into our activation function all the way to the outcome in a forward manner hence the name—"feed forward”. A general neural network model has the form:

\begin{align}

f(X) & = \beta\_0 + \sum\_{k=1}^{K} \beta\_k h\_k (X)\\

& = \beta\_0 + \sum\_{k=1}^{K} \beta\_k g(w\_{k0} + \sum\_{j=1}^{p}w\_{kj}X\_j)

\end{align}

In the first modeling step, the $K$ activations in the hidden layer are computed as functions of the features in the input layer, that is:

\begin{align}

A\_k = h\_k(X) &= g(w\_{k0} + \sum\_{j=1}^{p}w\_{kj}X\_j)\\

& =~~\textrm {where} ~g(z) = \textrm{non-linear activation function which has to be specified.}

\end{align}

The $K$ activation functions from the hidden layer then feed their outputs into the output layer so that we have:

\begin{align}

f(X) & = \beta\_0 + \sum\_{k=1}^{K} \beta\_k A\_k

\end{align}

Where $K$ in Figure 1 is 5. Parameters $\beta\_0, \cdots, \beta\_K$, as well as, $w\_{10}, \cdots, w\_{Kp}$ are estimated from the data. Quite a number of options exist for the activation function, $g(z).$ **^[Common activation functions include the sigmoid activation function favoured in classification problems, the rectified linear unit (ReLU) favoured in linear regression problems, tanh, and leaky ReLU.]** The non-linearity of the activation function $g(z)$ allows the model to capture complex non-linear structures as well as interaction effects.

###Rendering math as images

<img src=”https://render.githubusercontent.com/render/math?math=e^{i \pi}=-1”.>

**# The Artificial Neural Network Approach to modeling survival data**

The BLNN implementation of Bayesian inference in artificial neural networks is based on the Cox PH-based neural model described by @sharaf2015two. In particular, the idea is to build a predictive model for survival using a neural network with $K$ outputs. $K$ here defines the number of periods. Using this neural network architecture, Mani et al. estimated a hazard function where for each individual, we have a training vector a $1 \times K $ of hazard probabilities $(h\_{ik})$ defined as:

\[

h\_{ik}=

\begin{cases}

0 & ~\textrm{if} ~ 1\leq k \leq K \\

1 &~\textrm{if} ~ t \leqk \leq K ~ \textrm{and event = 1} \\

\frac{r\_k}{n\_k}~ \textrm{if}~ t \leqk \leq K ~ \textrm{and event = 0}

\end{cases}

\]

where $h\_{ik}=0$ if the event of interest did not occur (patient survived), $h\_{ik} =1$ if event occurred at some time, $t$ and $h\_{ik}=\frac{r\_k}{n\_k}$ if the subject is censored/ lost to follow -up during the course of the study, $t<K$. $h\_{ik}=\frac{r\_k}{n\_k}$ is the Kaplan-Meier (KM) hazard estimate for time interval $k$ and $r\_k$ and $n\_k$ denote the number of events due to the risk factor of interest in time period $k$ and the number at risk in time interval $k.$ The neural network uses the logistic sigmoid activation function defined as:

\[

\Phi (x) = \frac{1}{1+e^{-x}}

\]

The weights for this network are obtained through a minimization of the cross-entropy loss function^[ **formula defined here**]. **Figure 2** shows the architecture of a feed-forward neural network based on the Cox PH model with an input layer consisting of $p$ covariates and a bias term, a single hidden layer with $H$ nodes, and a single bias term. Lastly, we have an output layer with $K$ units, which learn to estimate the hazard probabilities associated with each individual at each time interval. The network's input layer feeds the hidden layer, which feeds the output layer. The "feed-forward" naming convention is derived from this aspect of the architecture. The hazard estimates based on this neural network model are then converted to estimates of survival based on the survival function:

\[

S(t\_k)=\prod\_{l=1}^k (1-h(t\_l))

\]

where $k$ denotes the disjoint intervals and $l$ the number of time periods in which the event occurred.

**Bayesian approach to inference using ANN**

This post focuses on inference using a two-layer feed-forward artificial neural network. Specifically, we describe the Bayesian learning neural networks implemented by @sharaf2020blnn on a neural network-based implementation of the Cox proportional hazard model described above. In training neural networks using conventional methodologies, the aim is to find a local minimum of the error function, an ideology that makes model selection rather difficult. Additionally, as described elsewhere by [**@** **hastie2009elements],** the training of neural networks presents such an issue as overfitting, a situation where, even though the model performs extremely well on the training data, it fails to generalize well on resampling or when applied on unobserved data. Overfitting has been linked to these models having too many weights such that they overfit at the global minimum of $R$ [@lawrence1997lessons; @ hastie2009elements]. According to @ burden2008bayesian :

>" Bayesian regularized artificial neural networks (BRANNs) are more robust than standard  
backpropagation nets and can reduce or eliminate the need for lengthy cross-validation."

In the Bayesian context, the idea is to use prior information about the distribution of the parameter of interest, update this information using the sample data and obtain a posterior distribution for the parameter, $\theta$. BLNN tries to present Hamiltonian energy, $H(w, p)= U(w)+K(p)$ as a joint probability distribution of the neural network's weights, $w$ and momentum, $\textbf{p}$. Given independence between $w$ and $\textbf{p}$, this joint probability is defined as :

\[

P(w, p) = (\frac{1}{z} exp^{-U(w)/z})(\frac{1}{T}exp^{-K(p)/T})

\]

Where:

<center>

$U(w) =$ the negative log-likelihood of the posterior distribution defined as $U(w)=-log[p(w)L(w|D)]$ <br>

$L(w|D) =$ the likelihood function given the data <br>

$K(p) = \sum\_{i=1}^{d}(P\_i^2)/(2m\_i) $ is the kinetic energy corresponding to the negative log-likelihood of the normal distribution with mean, $\mu$ and variance-covariance matrix with diagonal elements, $M=(m\_1, \cdots, m\_d)$<br>

$Z ~\textrm{and}~ T$ are the normalizing constants.

<center/>

The algorithm is summarized as below:

**[Image of the algorithm]**

**Source: Sharaf et. al (2020)**

Details about the implementation of this method can be found [here]( https://rdrr.io/github/BLNNdevs/BLNN/#vignettes).

@sharaf2020blnn utilize a no-U-turn sampler (NUTS), an extension of Hamiltonian Monte-Carlo (HMC) that seeks to reduce the dependence on the number of step parameters used in HMC while retaining the efficiency in generating independent samples. The ANN is trained using both HMC and NUTS with dual averaging. The negative log-likelihood is replaced by network errors, and backpropagation is used to compute the gradients. Network errors and weights are assumed to be normally distributed with mean, $\mu$ but with a non-constant variance, $\sigma^2$. The variance of the prior is known by the precision parameter, $\tau = \frac{1}{\sigma^2}$ aka the hyperparameters which are either assigned to fixed, fine-tuned values or re-estimated based on historical data. The list of hyperparameters allowed in the BLNN implementation is discussed elsewhere [@sharaf2020blnn]. The following section provides a sample application of BLNNs applied to real-world data.

# Bayesian-based neural networks for modeling survival using micro-array data

## Introduction and data description

## Exploratory data analysis and DEG identification

## Sample results

# Discussion

# Python extension