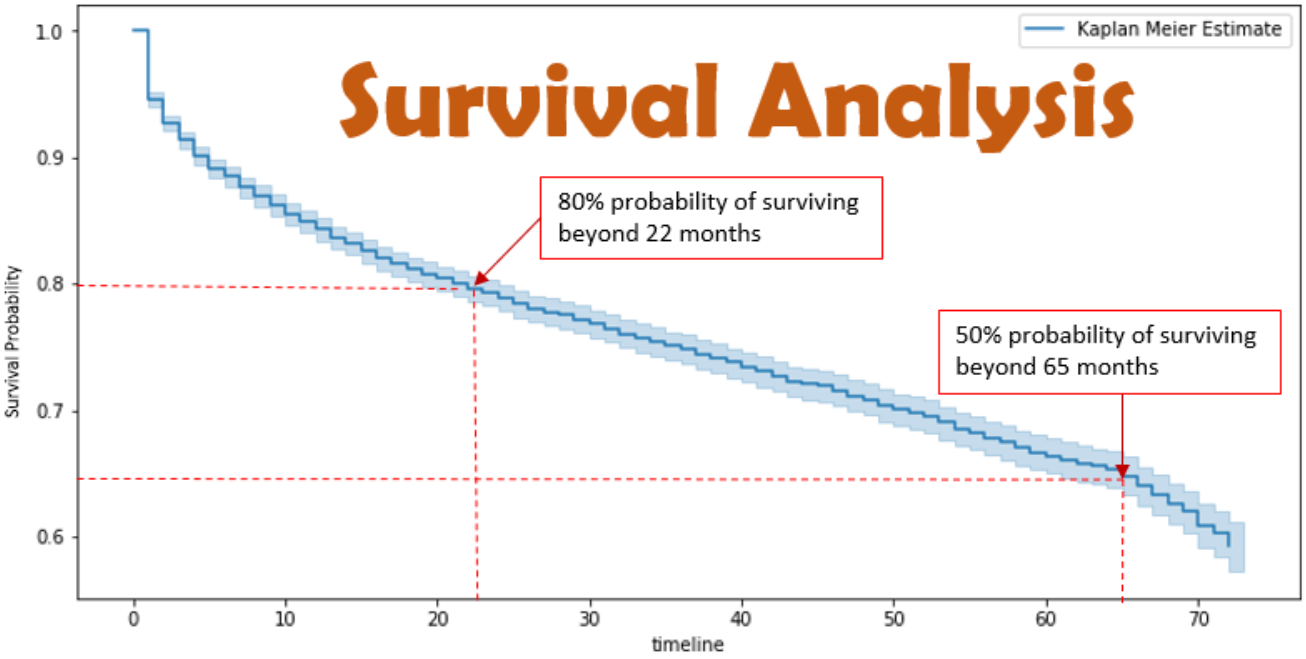
MENAKA SREEMANANANTH



Survival Analysis for Deep Learning - MNIST Dataset

May 8, 2020

1. PROJECT DESCRIPTION

In this project we are going to implement Survival Analysis learning task on MNIST Dataset using a convolution neural network, a Deep Learning technique. Survival Analysis have been implement before using various other algorithms like Support vector Machine, Random Forest or Boosting, where are here we are using CNN.

The Convolution Neural Network is trained to predict survival function on the test data, using a specific loss function for Survival Analysis.They are implemented and learned in as below,

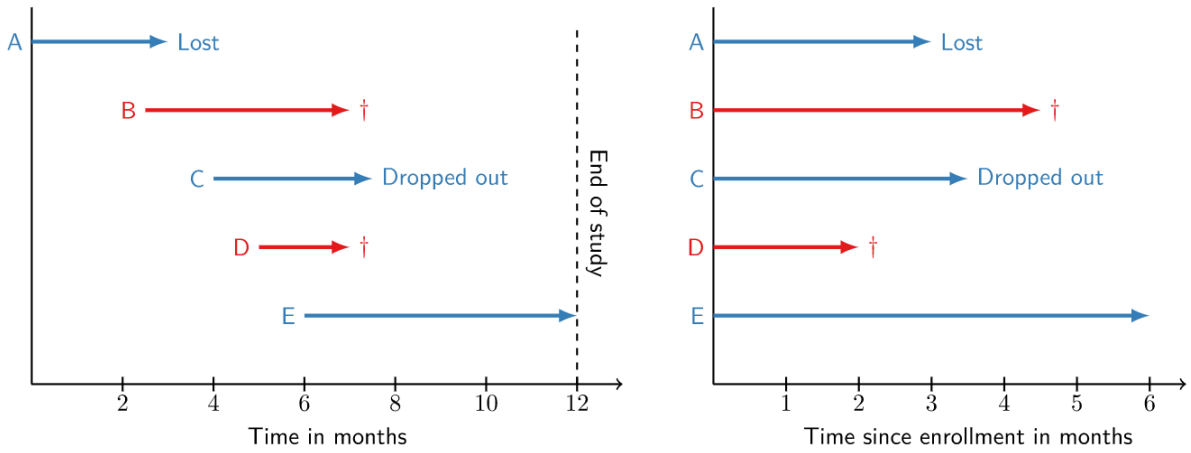
1. We learn and understand about basics and terms of survival analysis (**A. and B.**) .
2. We generate synthetic survival data on MNIST Dataset and visualize them [**2.Data Description**]
3. We implement most popular survival model, and learn how to use them as the loss function
4. Finally we bring them all together and implement a survival analysis model using CNN on MNIST and predict survival analysis on the test data.

A. Survival Analysis -An Overview:

Survival analysis is a field in statistics that is used to **predict when an event of interest** will happen.The field emerged from medical research as a way to model a patient's survival(predicting the time of death) — hence the term "survival analysis". This is different from other machine learning algorithms designed to address classification/prediction problems.

Survival Analysis is designed to estimate the time until an event of interest will occur. It establishes a connection between covariates and the time of an event. The response variable, also called the survival time, is recorded from an initial timestamp until the occurrence of the event of interest (uncensored) or until the subject exits the analysis without experiencing the event; this phenomenon is called censoring. Lets take a an example, a clinical study that has

been carried out over a year (as in the below figure) and from the diagram we can see Patient A did not follow up after three months and patient C Dropped out and Patient E didn't get any



Event (like death) after the one year period. So all of these patients data fall under Censoring. And we have only patient B and D who experienced the event through out the course time.Their data fall under uncensored category.

All the patient records contain time of event t>0 in case of uncensored data. And c>0 in case of censored data. As encoring and experiencing an event are mutually exclusive, we have an event indicator $\delta \in \{0;1\}$ and the observe the survival time 'y>0'.

$$y = \min(t, c) = \begin{cases} t & \text{if } \delta = 1, \\ c & \text{if } \delta = 0. \end{cases}$$

B. Basic Quantities:

Time, as in the survival time **T** is a modeled as a continuous random variable, therefore it can take any real value. **T is non-negative**, therefore it can only take positive real values (0 included). The two most time-to-event analysis can be derived is based on **T.** They are the ***survival function*** and the ***hazard function***.

- The **survival function** that calculates the probability that the event of interest has not occurred by some time t, which is the probability of survival beyond time and is defined as S(t) = P(T>t) . It is non-increasing with $S(0) = 1$, and $S(\infty) = 0$. (absence of an event)
- The **hazard function** $h(t)$ denotes an approximate probability (it is not bounded from above) that an event occurs in the small time interval $[t; t + \Delta]$, the rate at which event is taking place, out of the surviving population at any given time t, under the condition that an individual would remain event-free up to time : (occurrence of an event)

$$h(t) = \lim_{\Delta t \to 0} \frac{P(t \leq T < t + \Delta t \mid T \geq t)}{\Delta t} \geq 0$$

- The hazard function is also known as **conditional failure rate**, **conditional mortality rate,** or **instantaneous failure rate**. In contrast to the survival function, which describes the absence of an event, the hazard function provides information about the occurrence of an event.

2. DATA DESCRIPTION

The MNIST (Mixed National Institute of Standards and Technology) database was introduced by LeCun et al. in 1998. Since then, this dataset has been widely used as a test bed for different machine learning and pattern recognition methods.

The MNIST database contains a total of 70,000 instances. The database comprises two different sources: NIST's Special Database 1 (collected among high-school students) and NIST's Special Database 3 (retrieved from Census Bureau employees).

Each digit is represented as a 28 x 28 grayscale image (examples from the MNIST dataset can be seen in the figure below). These grayscale pixel intensities are unsigned integers, with the values of the pixels falling in the range [0, 255]. All digits are placed on a black background with a light foreground (i.e., the digit itself) being white and various shades of gray.

Fig 1: 100 Samples of MNIST training set.

3. RELATED WORK

Here we are implementing survival Analysis using Deep Neural Network -Convolutional Neural Network. There are lot of other implementation using traditional supervised learnings like RANDOM SURVIVAL FORESTS [5]. Support Vector Machine [6] are carried out and experimented.

4. METHODS

In this section, let's discuss about two most popular methods to determine the S(t) or h(t). They are Kaplan-Meier Model and Cox's Proportional Hazards Model.

I. KAPLAN-MEIER MODEL

A. Generating Synthetic Survival Data from MNIST

Synthetic data is "any production data applicable to a given situation that are not obtained by direct measurement" . Synthetic data are generated to meet specific needs or certain conditions that may not be found in the original (in MNIST) , real data. This can be useful when designing any type of system because the synthetic data are used as a simulation. The most popular non-parametric approach used to estimate the survival function is Kaplan–Meier estimator. The biggest limitation is that it **cannot estimate survival considering covariates**.

Here we are synthetically simulating survival Data for the MNIST Dataset. Its done by following the below,

- The MNIST Dataset is downloaded from TensorFlow directly and normalized (images), then the y_train and y_test are casted into int32 and the files are concatenated together.
- We assign each class label to one of four overall risk groups randomly, meaning some digits will fall to better survival and others to worse survival
- Next, we generate risk scores that indicate how big the risk of experiencing an event is ,relative to each other.

Below is output of the synthetic survival data generated for the MNIST Dataset , we can find the generated risk_score,risk_group of each of the class label[here handwritten digits].

class_label	risk_score	risk_group
0	3.071	3
1	2.555	2
2	0.058	0
3	1.790	1
4	2.515	2
5	3.031	3
6	1.750	1
7	2.475	2
8	0.018	0
9	2.435	2

Fig 2: Synthetic Survival Data for MNIST Dataset.

B. Generate Survival Times from risk scores

For generating the survival times from risk score we are following the "Bender et Al" (Generating Survival Times to Simulate Cox Proportional Hazards Models by exponential distribution)

Exponential Distribution: They are used to design the model whose behavior of units have a constant failure rate (like calculating when something will fail) . The main character of exponential distribution is that they are memoryless indicating that the remaining life of the component independent of its current age. Its probability density function is $f(t \mid \lambda) = \lambda exp(-\lambda t)$, where $\lambda > 0$, λ is a scale parameter that is the inverse of the expectation: $E(T) = \frac{1}{\lambda}$.

The result is a simple time-to event model with no memory, as the hazards rate is constant, $h(t) = \lambda$.

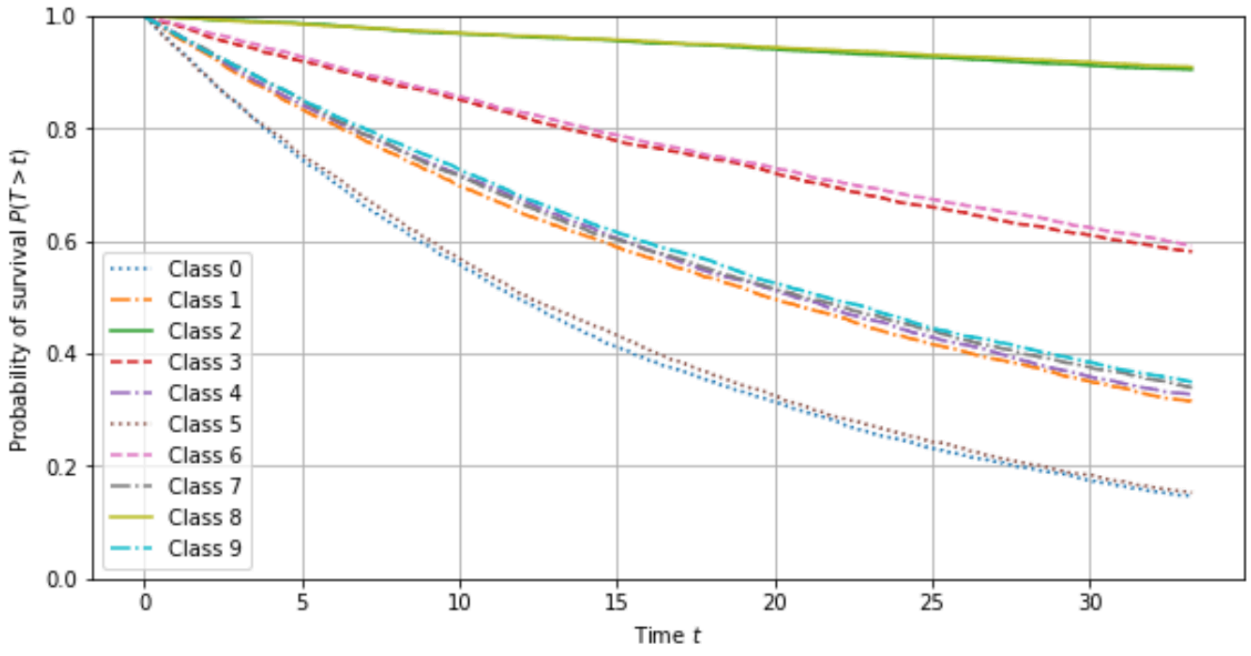
Here the λ is chosen such that the mean survival time is 365 days. Then we randomly censor survival times drawing times of censoring from a uniform distribution such that we approximately obtain the desired amount of 45% censoring.

The generated survival data has an observed time and a boolean event indicator for each MNIST image. We use the generated censored data and estimate the survival function S(t) to see what the risk score actually means in terms of survival.

Kaplan-Meier Estimator is also know as the product of limit estimators used to estimate the survival function from the lifetime data. It is used to measure the fraction of patients living for a certain amount of time after the treatment. The estimator of survival function $S(t)$ is give as,

$$\widehat{S}(t) = \prod_{i:\ t_i \leq t} \left(1 - \frac{d_i}{n_i}\right),$$

By Kaplan-Meier Estimator(using scikit-survival) we estimate the survival function and then we plot a curve,



From the figure ,we see when the survival risk is high in case of digit '0' it falls quickly, where as when the survival rate is low in case of digit '2' and '8' the curve takes a while to reduce.

C. Evaluating Predictions:

An important aspect of survival analysis is that both the training data and test data are censored, meaning we are not able to note the exact time of an event no matter how we split the data. Hence we have to take censoring into account when we measure our performance. A widely used performance measure is 'Harrell's concordance index'.

Harrell's concordance index, given a set of (predicted) risk scores and observed time, checks if the ordering by risk scores is concordant with the ordering by actual survival time. It is also flawed when it comes to highly censored data.

We can take the risk score from which we generated survival times to check how good a model would perform if we knew the actual risk score.

```
: cindex = concordance_index_censored(event_test, time_test, risk_scores[y_train.shape[0]:])

print(f"Concordance index on test data with actual risk scores: {cindex[0]:.3f}")
```

```
Concordance index on test data with actual risk scores: 0.705
```

The perfect result would be an actual risk score of 1.0, but we got an actual risk score to be 0.705. This is because the generated survival survival times are randomly generated and not a deterministic function of the risk score. So any model we train on this data will not exceed this performance value.

II. COX PROPORTIONAL HAZARD MODEL

The Most popular and widely used method to learn from censored data is Cox Proportional Hazard Model. This model is used to predict the survival function of specific unit [2]. This model focus on modeling the hazard function by assuming that its time component

and feature component are proportional [2]. This model **overcomes the limitation** of previous model as **it considers the features or covariates**. It models the hazard function $h(t_i)$ as,

$$h(t|x_{i1},\ldots,x_{ip}) = h_0(t)\exp\left(\sum_{j=1}^{p} x_{ij}\beta_j\right) \Leftrightarrow \log\frac{h(t|\mathbf{x}_i)}{h_0(t)} = \mathbf{x}_i^\top\beta,$$

where $\beta \in \mathbb{R}^p$ are the coefficients associated with each of the features, and no intercept term is included in the model.

Hazard Function is of two parts:

1. baseline hazard function h_0 that only depends on time t
2. The exponential is independent of time and only depends on covariates x_i

Cox's proportional hazards model is fitted by maximizing the partial likelihood function, which is based on the probability that the i-th individual experiences an event at time t_i, given that there is one event at time point t_i.By specifying the above hazard function, the baseline function h_0 is not needed and can be eliminated for finding the coefficients β, we get the below mathematical expression,

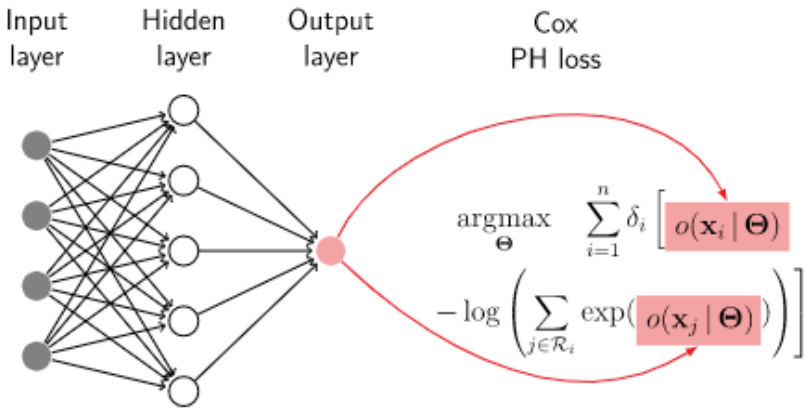
$$\begin{aligned}
&P(\text{subject experiences event at } y_i \mid \text{one event at } y_i)\\
=&\frac{P(\text{subject experiences event at } y_i \mid \text{event-free up to } y_i)}{P(\text{one event at } y_i \mid \text{event-free up to } y_i)}\\
=&\frac{h(y_i|\mathbf{x}_i)}{\sum_{j=1}^{n} I(y_j \geq y_i)h(y_j|\mathbf{x}_j)}\\
=&\frac{h_0(y_i)\exp(\mathbf{x}_i^\top\beta)}{\sum_{j=1}^{n} I(y_j \geq y_i)h_0(y_j)\exp(\mathbf{x}_j^\top\beta)}\\
=&\frac{\exp(\mathbf{x}_i^\top\beta)}{\sum_{j\in\mathcal{R}_i}\exp(\mathbf{x}_j^\top\beta)}.
\end{aligned}$$

Where β is

$$\hat{\beta} = \arg\max_{\beta} \log PL(\beta) = \sum_{i=1}^{n} \delta_i \left[\mathbf{x}_i^\top \beta - \log \left(\sum_{j \in \mathcal{R}_i} \exp(\mathbf{x}_j^\top \beta) \right) \right].$$

A. Non-linear Survival Analysis with Neural Networks

From the above derivation we come to a conclusion that Cox's proportional model is a linear model. i.e., the predicted risk score is a linear combination of features. This model can be easily converted to a non-linearity case by just replacing the linear predictor with the output of a neural network with parameters $\boldsymbol{\Theta}$. The below diagram helps us understand a little better,



This model was proposed by Fraggi and Simon [3] in 1995. The explored a Multilayer Perceptron Model. This same loss can be used for advanced architectures like CNN or Recurrent Neural Networks. Hence we are using the same loss function but its also a little tricky for establishing both training and for evaluation.

B. Computing Loss Function

The most problematic part when trying to implement the COX PH loss function is the inner sum over the risk set: $\sum_{j \in \mathcal{R}_i} \exp(\mathbf{x}_j^\top \beta)$. Risk Set is is defined as $\mathcal{R}_i = \{j \mid y_j \geq y_i\}$.

It sort the data once in descending order by survival time and then incrementally update the inner sum, which leads to a linear complexity to compute the loss (ignoring the time for sorting).

Another issue is with the subjects with the smallest uncensored survival time. It's impractical to keep the whole data set in the GPU memory. If we use mini batch, we encounter

1. We won't be able to compute exact loss, as we may not have access to all the risk sets.
2. We need to sort each mini Bach by observed time, instead of the whole set.

But for practical reasons we go with mini batch, as long as the batch contains several uncensored samples, otherwise the outer sum in the partial likelihood function would be over an empty set.

Here we have implemented the sum over the risk set by multiplying the exponential of the predictions (as a row vector) by a squared boolean matrix that contains each sample's risk set as it's rows. The sum over the risk set for each sample is then equivalent to a row-wise summation. [Code in the .ipynb file , under Computing the Loss function section]

C. Creating a Convolutional Neural Network for Survival Analysis on MNIST

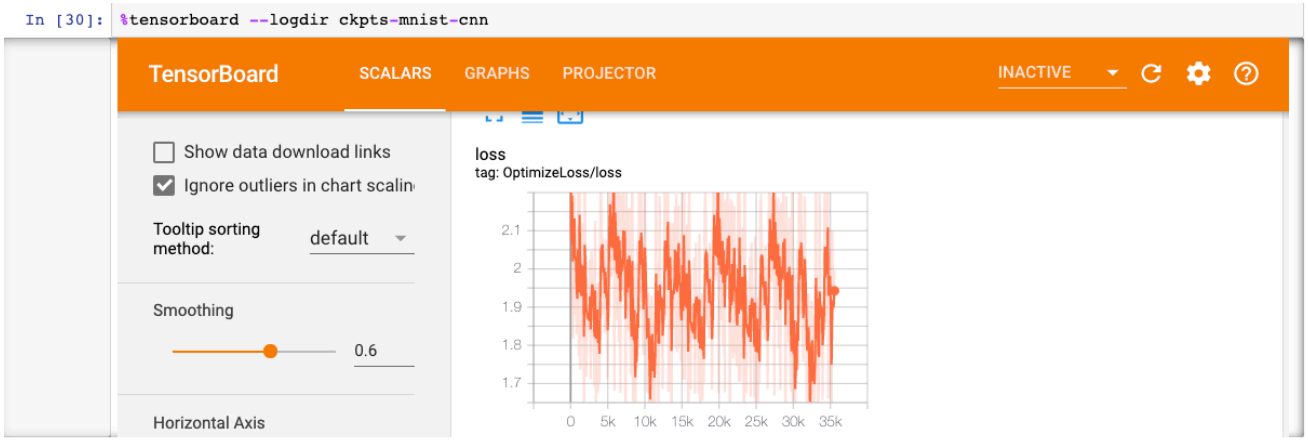
We have created a convolutional neural network (CNN) to learn a high-level representation from MNIST digits.Thereby we have estimated each image's survival function. The CNN follows the LeNet architecture [4] where the last linear has one output unit that corresponds to the predicted risk score. The predicted risk score, together with the binary event indicator and risk set, are the input to the Cox PH loss.

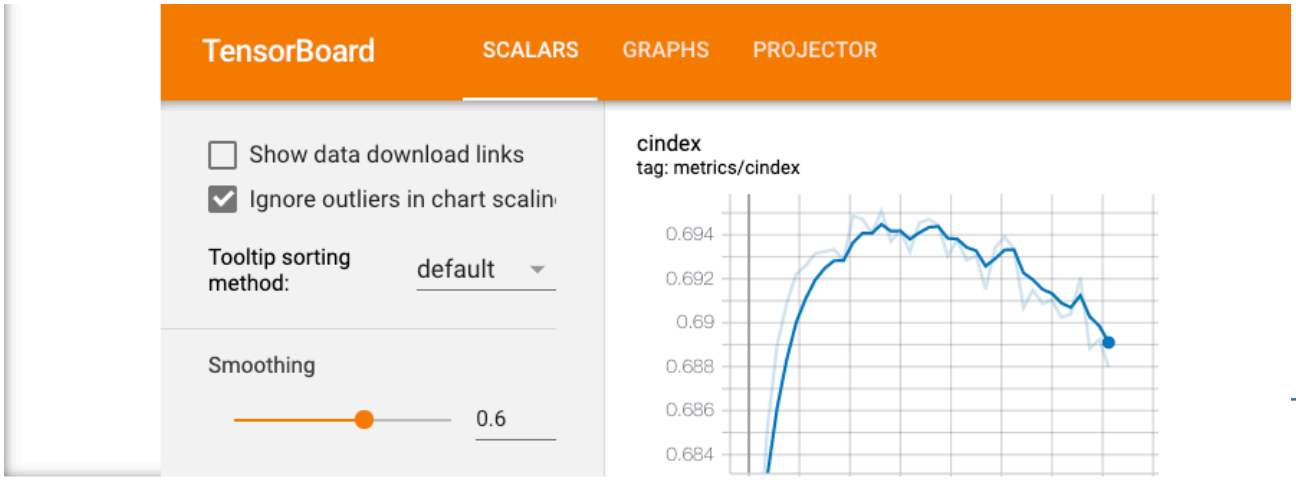
```
risk_score = model(features, training=is_training)
```

D. Monitoring Training Process in TensorFlow

To monitor the training process, we computed the concordance index with respect to a separate validation set. Similar to the Cox PH loss, the concordance index needs access to predicted risk scores and ground truth of *all* samples in the validation data. While we had to opt for computing the Cox PH loss over a mini-batch, this is not recommended for the validation data.

For small batch sizes and/or high amount of censoring, the estimated concordance index would be quite volatile, which makes it very hard to interpret. In addition, the validation data is usually considerably smaller than the training data, therefore we can collect predictions for the whole validation data and compute the concordance index accurately.



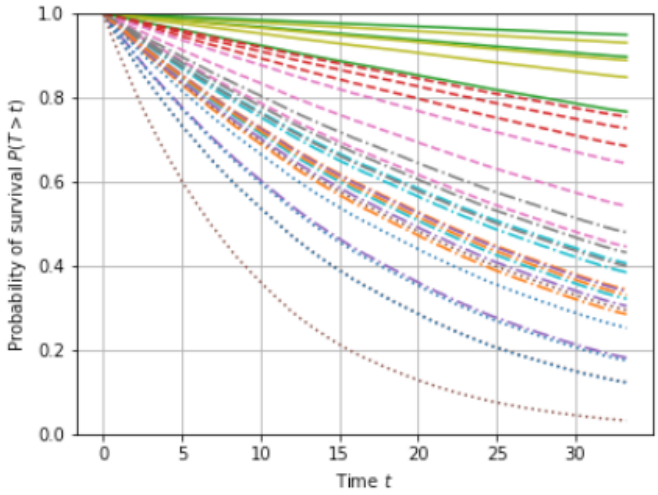


We can make a couple of observations:

1. The final concordance index on the validation data is close to the optimal value we computed above using the actual underlying risk scores.
2. The loss during training is quite volatile, which stems from the small batch size (64) and the varying number of uncensored samples that contribute to the loss in each batch. Increasing the batch size should yield smoother loss curves.

E. Predicting Survival Functions

Using Breslow's estimator we are predicting the hazard function for each individual survival function, done by analogous to linear Cox PH model.



Solid lines correspond to images that belong to risk group 0 (with lowest risk), which the model was able to learn. Samples from the group with the highest risk are shown as dotted lines. Their predicted survival functions have the steepest descent, confirming that the model correctly identified different risk groups from images.

3. DISCUSSION

In this project we have analyzed, what survival analysis is and what are the main models used its importance how to implement them using a synthetic data generated for the MNIST image Dataset. This is a very vast field on interest but to be able to get started and take a deep dive into it will be very helpful in future.

4. FUTURE WORK

As discussed by the Author, I am planning on implementing the Mobadersany et al. dataset and understand how they work on a real Medical Dataset.

5. CONCLUSION

In this project, we used a Convolution neural network for survival analysis on Synthetic Data extracted from MNIST Dataset. First we Looked into the dataset and applied a simple Kaplan-Meier Model and the addressed Cox Proportional Hazard Model using CNN. Obviously MNIST is not a medical data. But the knowledge and inference can be transferred when we have to perform a survival analysis on a medical data.

6. REFERENCES

1. https://k-d-w.org/blog/2019/07/survival-analysis-for-deep-learning/
2. Stephane Fotso. " Deep Neural Networks for Survival Analysis Based on a Multi-Task Framework".
3. DAVID FARAGGI AND RICHARD SIMON. "A NEURAL NETWORK MODEL FOR SURVIVAL DATA"
4. Yann LeCun Leon Bottou Yoshua Bengio and Patrick Haner."GradientBased Learning Applied to Document Recognition" 1998.
5. Hemant Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone and Michael S. Lauer. "RANDOM SURVIVAL FORESTS"
6. Sebastian Pölsterl1, Nassir Navab1,2, and Amin Katouzian. "Fast Training of Support Vector Machines for Survival Analysis"