Gabriel Borg, gabriel.borg93@gmail.com,
Aida Ibisevic, ibisevicaida@gmail.com

June 4, 2023

Deephit in Torch: Exploring and reimplementing Deephit in pytorch

1 Introduction

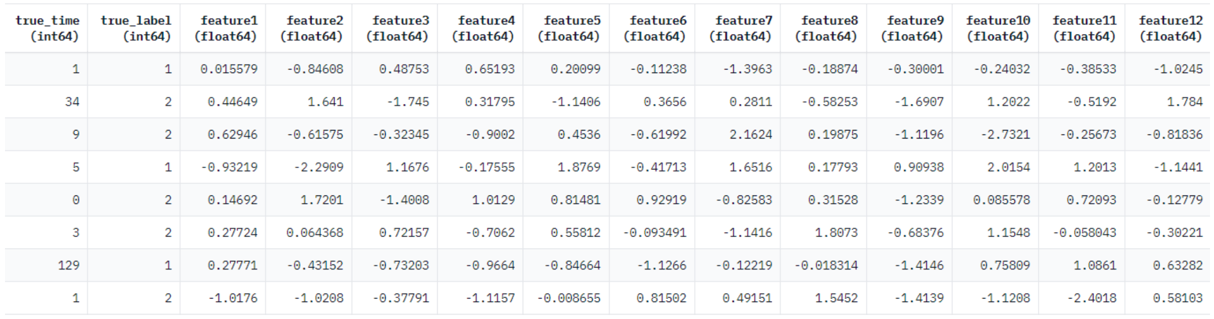
The implementation of the DeepHit model from C. Lee, et al. (2018) in PyTorch for analyzing survival analysis with competing risks is the focus of this project. This section provides an overview of the project scope and introduces the synthetic dataset used for evaluation purposes.

1.1 Project scope

Survival analysis is a statistical method widely used in various domains, particularly in clinical trials within the field of medicine. However, the exploration of multiple time-to-event analysis, specifically competing risks, in conjunction with machine learning, is relatively limited. This project aims to enhance the implementation of the DeepHit model, originally developed in TensorFlow 1, by rewriting it in PyTorch. By leveraging PyTorch's flexibility and simplicity, we can create a more modular and adaptable implementation of the DeepHit model for analyzing competing risks within the context of survival analysis.

1.2 Dataset

The data used to implement DeepHit is a synthetical dataset created by the authors of Deephit in an attempt to model real life data in a controlled environment. It is constructed as a tuple (***x**, s, k*) where **x** are the covariants, s is the final observed time for each patient and *k* is the event at time *s*. The dataset consists of two competing risks and right censored data points. The covariant vector ***x*** for patient *i* consists of three 4-dimensional variables $x = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)})$. See the image below for part of the dataset.



true_time (int64)	true_label (int64)	feature1 (float64)	feature2 (float64)	feature3 (float64)	feature4 (float64)	feature5 (float64)	feature6 (float64)	feature7 (float64)	feature8 (float64)	feature9 (float64)	feature10 (float64)	feature11 (float64)	feature12 (float64)
1	1	0.015579	-0.84608	0.48753	0.65193	0.20099	-0.11238	-1.3963	-0.18874	-0.30001	-0.24032	-0.38533	-1.0245
34	2	0.44649	1.641	-1.745	0.31795	-1.1406	0.3656	0.2811	-0.58253	-1.6907	1.2022	-0.5192	1.784
9	2	0.62946	-0.61575	-0.32345	-0.9002	0.4536	-0.61992	2.1624	0.19875	-1.1196	-2.7321	-0.25673	-0.81836
5	1	-0.93219	-2.2909	1.1676	-0.17555	1.8769	-0.41713	1.6516	0.17793	0.90938	2.0154	1.2013	-1.1441
0	2	0.14692	1.7201	-1.4008	1.0129	0.81481	0.92919	-0.82583	0.31528	-1.2339	0.085578	0.72093	-0.12779
3	2	0.27724	0.064368	0.72157	-0.7062	0.55812	-0.093491	-1.1416	1.8073	-0.68376	1.1548	-0.058043	-0.30221
129	1	0.27771	-0.43152	-0.73203	-0.9664	-0.84664	-1.1266	-0.12219	-0.018314	-1.4146	0.75809	1.0861	0.63282
1	2	-1.0176	-1.0208	-0.37791	-1.1157	-0.008655	0.81502	0.49151	1.5452	-1.4139	-1.1208	-2.4018	0.58103

Figure 1. First rows of the dataset.

The dataset has 30000 data points where each row represents one patient. 15000 of these data-points have been right-censored, 7600 have been subjected to event 1 and the remaining 7400 have been subjected to event 2.

Gabriel Borg, gabriel.borg93@gmail.com,
Aida Ibisevic, ibisevicaida@gmail.com June 4, 2023

2 Method

This section describes the methodology employed to evaluate the performance of the DeepHit model, including its implementation in PyTorch, data collection, preprocessing, loss function calculation, model training, and evaluation.

2.1 Data Collection and Preprocessing

DeepHit requires discrete time steps for model input. To ensure consistency with the original research paper, we followed the same train-test-validation split methodology for our dataset. However, we used 10 time steps in our implementation for simplicity.

We defined two distinct functions for discretizing the target variable. One function utilized quantiles to create equal bin sizes, while the other function employed an equidistant approach to generate equal time step ranges.

During the preprocessing stage, we excluded the "time" and "label" columns from the dataset. The "true time" and "true label" columns were then designated as the target variable, denoted as "y," while the remaining feature columns served as the covariates, denoted as "x."

To efficiently handle the dataset and facilitate batch processing during training, we implemented a dataloader. The dataloader allowed us to load the data in batches, seamlessly integrating with the model training process and optimizing computational resource utilization.

2.2 Implementing loss-functions

In order to implement the loss function, we needed to understand it. We achieved this by thoroughly studying the original implementation. With a clear understanding of the loss calculations, we proceeded to code the DeepHit model in PyTorch, by leveraging vectorization and hardware acceleration with CUDA.

To ensure the correct behavior of the loss function, we test our implementation with different batches of data. Most importantly, we examine its ability to capture time-to-event information, predict survival probabilities, and rank the relative risks of different subjects.

Finally, the total loss is implemented as in the paper including both the ranking loss and the negative log-likelihood loss.

2.2 Model implementation and training

The same architecture as described in the research paper was used. Since the parameters of the model are fairly low we could utilize the dataloader as described in section 2.1 with a batch size of 256 (for both train and validation dataset). This enabled an efficient processing of the data during training. Furthermore, Adam was used as the optimizer with a learning-rate decay implementation and a low “min delta” for early stopping, since the sensitivity to changes in performance from the hyperparameter was rather low.

Empirical hyperparameter tuning was employed due to the time limit. We experimented with different learning rates and batch sizes to optimize the model's performance. Furthermore, the effects of varying the values of sigma and alpha in the loss function was explored. Although the hyperparameter tuning was not extensive, it allowed for iterative adjustment of the parameters based on empirical observations.

2.4 Model evaluation

After completing the model implementation and hyperparameter tuning, the performance of the DeepHit model was evaluated using the Brier score. The Brier score, a widely used metric for assessing the accuracy of probabilistic predictions, particularly in survival analysis, was employed. It measures the mean squared difference between predicted probabilities and observed outcomes. In the end we trained and evaluated 4 different survival models using two different approaches for handling time steps: equidistant and unidistant.

Gabriel Borg, gabriel.borg93@gmail.com,
Aida Ibisevic, ibisevicaida@gmail.com June 4, 2023

3 Results

In this section, we present the findings and outcomes of our study on the implementation of the deephit model, preprocessing and training.

3.1 Time Steps

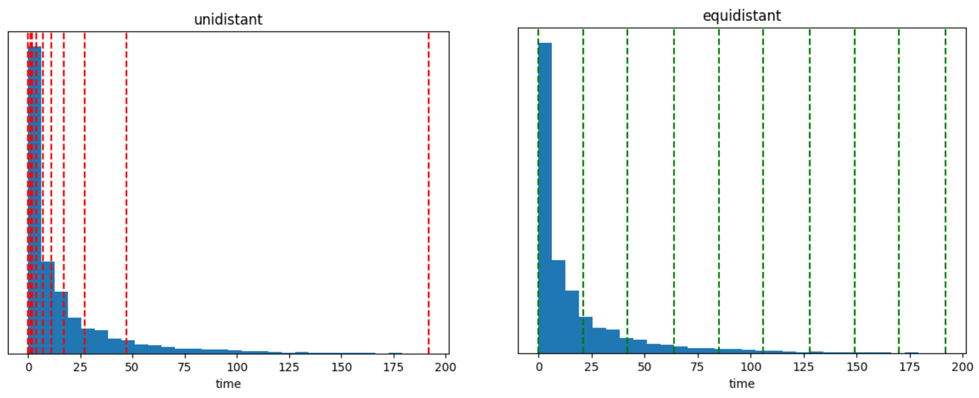


Figure 2. Output from discretizing the time-steps as unidistant and equidistant

3.2 Data distributions

Table 1. Distribution between train, validation and test data

	Train	Validation	Test
Amount of censored data points (%)	9522 (~50%)	2421 (~50%)	3057 (~50%)
Amount of data points subjected to event 1 (%)	4911 (~26%)	1227 (~26%)	1462 (~24%)
Amount of data points subjected to event 2(%)	4767 (~24%)	1152 (~24%)	1481 (~26%)
Total	19200	4800	6000

3.2 Hyperparameters

Network hyperparameters:

```
def fc_net(input_dim, hidden_dim, output_dim, dropout):
    layers = [
        nn.Linear(input_dim, hidden_dim),
        nn.ReLU(),
        nn.BatchNorm1d(hidden_dim),
        nn.Dropout(dropout),
        nn.Linear(hidden_dim, hidden_dim),
        nn.ReLU(),
        nn.BatchNorm1d(hidden_dim),
        nn.Dropout(dropout),
        nn.Linear(hidden_dim, output_dim),
    ]
    net = nn.Sequential(*layers)
    return net

input_dim=12,
input_dim_head=12,
hidden_dim_body=64,
hidden_dim_head=32,
output_dim=10,
discrete_time=10,
nr_event=2,
dropout=0.2,
residual=True
```

Model training hyperparameters:

```
optimizer = Adam(model.parameters(), lr=0.04)
scheduler = lr_scheduler.ExponentialLR(optimizer, gamma=0.98)
epochs = 80
early_stopping_tol = 2
early_stopping_min_delta = 0.01
alpha = 0.2
sigma = 0.1
batch_train_size,   batch_val_size  = 256
```

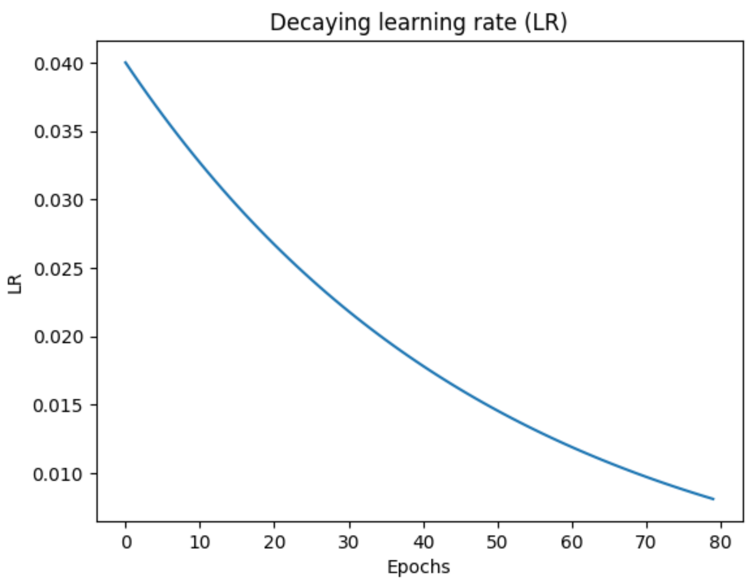


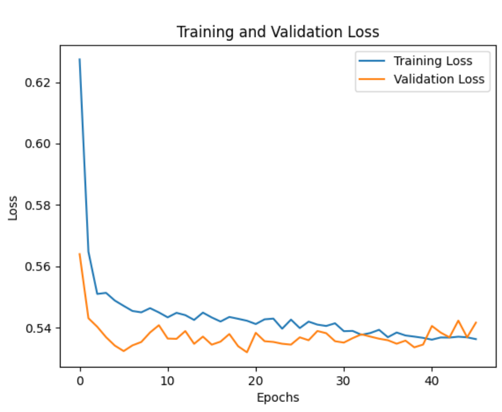
Figure 3. Decay learning rate over 80 epochs using gamma 0.98 and lr = 0.04.

3.3 Training and validation loss

Four different models were trained with the same hyperparameters to calculate the training and validation loss and evaluated on the brier score. These were for each type of time-step implementation one with and without competing risks.

Table 2. Brier score for different set-ups where a) there is only one single event with equidistant time-step, b) we are considering competing risks with an equidistant time step, c) there is only one single event with unidistant time-step and d) we are considering competing risks with a unidistant time step.

Equidistant time-step		**Unidistant time step**	
***a)* Single event**	***b)* Competing risks**	***c)* Single event**	***d)* Competing risks**
0.280	0.279	0.191	0.189



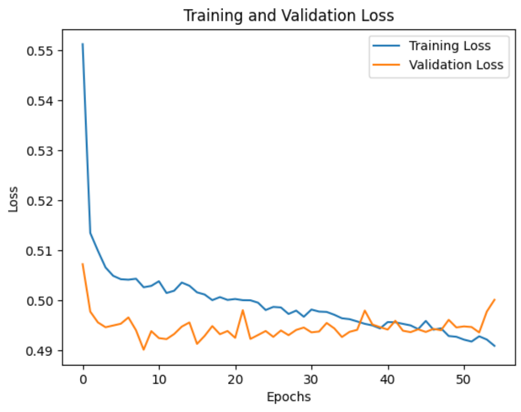
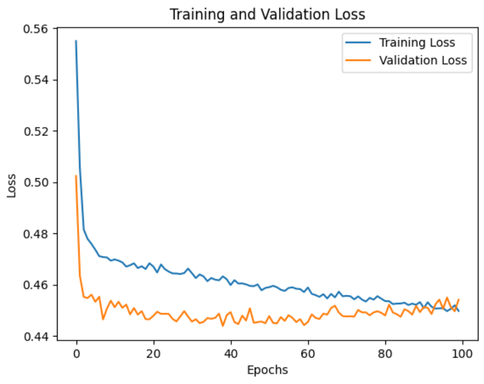


Figure 4. Training and validation loss for single event, column a) in table 2 and Training and validation loss for competing risks, column b) in table 2. Both have time-step equidistant.



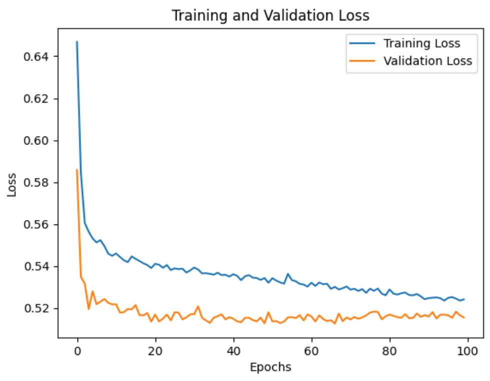


Figure 5. Training and validation loss for single event, column c) in table 2 and Training and validation loss for competing risks, column d) in table 2. Both have time-step unidistant.

Gabriel Borg, gabriel.borg93@gmail.com,
Aida Ibisevic, ibisevicaida@gmail.com

4 Conclusion

In this study, we trained 4 different survival models using two different approaches for handling time steps: equidistant and unidistant. The models were evaluated using the Brier score, a commonly used metric for assessing the accuracy of survival models.

For the models trained with an equidistant time-step, the Brier scores were 0.280 for the single event scenario and 0.279 when considering competing risks. These scores suggest a moderate improvement of the accuracy in predicting the survival outcomes.

In contrast, the models trained with an unidistant time-step showed better results. The Brier scores achieved were 0.191 for the single event scenario and 0.189 when accounting for competing risks. These scores indicate a relatively better performance compared to the equidistant time-step models, suggesting that the unidistant approach captures competing risk more effectively. More importantly, these findings emphasize the importance of carefully considering the time-step strategy when training survival models. The unidistant time-step approach demonstrated relatively improved predictive performance, indicating its potential for enhancing the accuracy of survival predictions. Further research and investigation into the underlying reasons for this improvement would be valuable for refining future survival models.

Overall, our study highlights the significance of selecting an appropriate time-step strategy to optimize the accuracy and reliability of survival models in various applications.

Here is the source code for the project:
https://github.com/Borg93/deephit_pytorch/blob/main/src/preprocess.py

5 Future work

For future work, several key areas can be explored to enhance the model's performance and applicability:

- Evaluation: Implement the CI-index to gain deeper insights into the model's performance on ranking.
- Hyperparameter Tuning: Utilize Bayesian search to conduct a more rigorous optimization of both training and network hyperparameters.
- Early Stopping: Improve the early stopping mechanism to consider the validation loss trend over multiple epochs, stopping training if it consistently increases.
- Time-Step Splitting: Evaluate and investigate further strategies for splitting the time-step into smaller intervals, potentially improving model accuracy and capturing finer details.
- Explainable AI (XAI): Implement XAI techniques to enhance the model's interpretability and make it more applicable in real-life scenarios.

These advancements in future work will contribute to a more robust and effective model.

References

C. Lee, et al. (2018). DeepHit: A Deep Learning Approach to Survival Analysis with Competing Risks. AAAI Conference on Artificial Intelligence (AAAI),