Improving Neural Networks for Predicting Sepsis from Imbalanced, Multivariate Time Series Data with High Missing Rates

- mi107f21-

Project Report

Simon Dam Nielsen
Martin Simonsen
Mathias Højer Svendsen

Aalborg University
Department of Computer Science

Department of Computer Science
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Improving Neural Networks for Predicting Sepsis from Imbalanced, Multivariate Time Series Data with High Missing Rates

Project Period:
Spring 2021

Project Group:
mi107f21

Participants:
Simon Dam Nielsen
Martin Simonsen
Mathias Højer Svendsen

Supervisor:
Thomas Dyhre Nielsen

Page Numbers: 89 (163 with appendix)

Date of Completion: June 17, 2021

Abstract:

In this project, we attempt to create a neural network model, which outperforms an XGBoost model for predicting sepsis. The data, we have available is highly imbalanced, multivariate time series data with high missing rates. In order to mitigate the problems that arise from this type of data, we experiment with modifications of the following time series models: LSTM, TCN, BRITS, and GRU-D. We propose a neural network architecture, which utilizes extracted features from the data while incorporating the time series models. We experiment with the following: class weighted loss function, demographics extracted features, missingness representations, observation rates extracted features, and delta representation extracted features. Through our experiments, we observe that the neural network models benefit from the observation rate, and BRITS and GRU-D show the best results on separate datasets. We also observe that the missingness representations are beneficial as inputs to the models. Finally, we conclude that for one dataset, our model is preferable in a clinical setting compared to XGBoost, due to its calibration, despite XGBoost's superior performance metrics.

Preface

This report is the specialization report done by three 4th semester masters students of Software Civil Engineering from Aalborg University. We investigate how to improve neural network models' performance for sepsis predictions while also considering model calibration. We evaluate the models on two datasets created from electronic health records: PhysioNet Computing in Cardiology Challenge 2019 dataset , which is publicly available [1], and Processed CROSS-TRACKS dataset, which we have access to through our employment and tasks at Enversion A/S. As part of our terms of employment, we have signed a non-disclosure agreement as well as received the appropriate data protection training. All data presented in this report is anonymous. We want to thank our supervisor Thomas Dyhre Nielsen for his feedback and guidance throughout this project. We also want to thank Enversion A/S for giving us access to Processed CROSS-TRACKS dataset, especially Bo Thiesson and Mads Kristensen for helping us get started on their systems, and Simon Lauritsen for preparing the data for us.

Aalborg University, June 17, 2021

Readers Guide

The target group for this report is Software master students with specialization within machine learning. We use the IEEE citation style, where a citation is a number within square brackets, where the number refers to the entry in the bibliography.

We refer to chapters as "Chapter #", where # is the chapter number, and to sections as "Section #.#", where the first # is the chapter and the second # is the section number in the chapter. Figures, tables, equations are referred to as "Figure #.#", "Table #.#", and "Equation #.#" respectively, where "#.#" follow the same convention as for sections, where the first # is the chapter and the second is the figure/table/equation number in the chapter. If a letter is used as # in a reference it is a reference to the appendix.

Since this is based on our previous semester, some section in this report is a copy of a section from our previous semester report with some minor adjustments. The sections that are copied from our previous semester report are marked with a number on the heading that corresponds to a footnote, which says "This section is a slightly modified version of # from our previous semester report [2].", where # is the section it is copied from. If a section is marked with such a footnote, the section and all of its subsections are copied from our previous semester report.

Summary

In recent years the amount of data from electronic health records has increased dramatically. This has led to a large amount of research applying deep learning methods to this data, in order to solve real-world problems. One of these problems is the prediction of sepsis in patients admitted to a hospital.

This report is written in collaboration with Enversion A/S, a company that develops deep learning solutions to assist medical professionals. One area they develop solutions for is predicting sepsis in patients. Their current best performing model for this task is a gradient boosted decision tree model called XGBoost. However, they prefer a neural network based model. In this project, we attempt to create a neural network based model, that is better than XGBoost at predicting sepsis.

We start by analyzing the data we have available, a publicly available dataset, which we refer to as PhysioNet Computing in Cardiology Challenge 2019 dataset , and a dataset not publicly available from the CROSS-TRACKS cohort, which we refer to as Processed CROSS-TRACKS dataset. We find that both of these datasets are highly class imbalanced and have high missing rates. We also find that framing the data properly to fit the real-world usage is important.

We then form a problem statement, which defines the purpose of this project: to find a well-performing, well-calibrated neural network model for predicting sepsis.

We then begin designing the model, where we start by listing the most important characteristics of our model. Here we choose that our model must be able to take advantage of the temporal aspect of the time series aspect. The model must also handle the missing values in the dataset.

From this, we find two models, GRU-D and BRITS, and also two models from our previous semester report, LSTM and TCN, which we build on top of in our experiments. Additionally, we consider a model architecture, which can utilize some extracted features from the data and use that data as an additional input to the model.

We then perform a set of experiments on these models and determine how different approaches affected the models. From these experiments, we found that using an observation rate as an extracted feature helped the models better predict sepsis. We also found that for LSTM, adding the same missingness representations as those used by GRU-D and BRITS benefitted it.

Since we saw a great performance increase by adding the observation rate as an extracted feature, we experiment with adding the data representation XGBoost uses as another extracted feature to the neural network models. This did not improve performance, but it shows some potential.

We conclude that using the observation rates as an extracted feature improved performance. Additionally, the missingness representations used by GRU-D and BRITS are useful inputs to other neural networks as well.

We conclude that BRITS is the best performing neural network model on PhysioNet Computing in Cardiology Challenge 2019 dataset , however it was still inferior to XGBoost. We also conclude that GRU-D is the best performing neural network model on Processed CROSS-TRACKS dataset, and on this dataset, it is preferable over XGBoost due to its better calibration, despite its lower performance.

Finally, we conclude that all of the models we consider in this project need to be researched further before they can be used in a real-world clinical setting.

Contents

Part I

Problem Analysis

Chapter 1

Introduction

In the past decade, the amount of data from Electronic Health Records (EHR) has increased dramatically. This increase has resulted in a large amount of research attempting to apply deep learning methods to EHR data to assist in a clinical setting [3]. The prediction of sepsis (described in Appendix B) is one area many researchers attempts solve by using deep learning [4][5]. The fatality rate of sepsis is high and around 31.5 million cases of sepsis are found each year worldwide [4]. Sepsis is hard to diagnose and starting treatment early significantly reduces the mortality rate [4].

In our previous project [2], we investigated how the calibration and performance of state of the art models for predicting sepsis were affected by various factors. One thing we found was, that predicting whether a patient develops sepsis within the next 24 hours is a difficult task for the deep learning approaches we tested.

One reason for this is that creating a well-calibrated model is difficult. Calibration (described in Section 8.2) is important in areas such as healthcare, where a wrong decision can lead to serious injury or even death. Many modern neural networks are poorly calibrated even though they achieve good performance [6]. Many factors can affect calibration such as the size of the model, whether batch normalization is used, and more [2][6], and we need to consider these aspects when we design our model. Another aspect we need to consider is the relationship between calibration and performance. Both our previous report and [6] find that in general, when the model performs better, the calibration gets worse. This hints towards a trade-off between performance and calibration.

One of the challenges we had with creating a well-performing model was the class imbalance between the sepsis-positive and -negative patients. We found that changing the ratio of the classes by undersampling the negative cases in the training set negatively affected both performance and calibration, while oversampling had no significant effect. Therefore, we have to try other approaches that could make the performance of the models better when using a class imbalanced dataset.

Another challenge in EHR data is that the measurements of e.g. heart rate or SaO2 content in the blood are taken irregularly, which means that a lot of the time we do not have data for these measurements and is therefore missing. In our previous project, we briefly tested simple imputation methods to handle the missing values, but further design considerations are required to get as much information out of the patterns in the missing data.

This project is written in collaboration with Enversion A/S (Enversion). Enversion researches how to apply machine learning methods to EHR data, and develops products to assist medical professionals. One area Enversion researches is deep learning models for early

detection of sepsis in patients. Their current best performing model is based on the Python library "XGBoost", which we refer to as XGBoost. XGBoost is a gradient boosted decision trees model, which is currently Enversion's best performing model, but it suffers from poor calibration. Enversion is interested in a well-calibrated deep neural network model with equal or better performance than their XGBoost model. Neural networks are desirable for Enversion as they are well-researched, and a lot of tools and methods for analyzing them exist. One of these tools is used for explaining which features in the data that the model received contributed the most to the model's prediction. This is useful information for the medical staff as it tells them which measurements the prediction is based on and does therefore make them more likely to trust the model's prediction.

In order to determine the best approach to develop a model for predicting sepsis, we start by examining the data that is available for sepsis prediction in the following chapter.

Chapter 2

Description of Data

In this chapter, we describe and analyze the datasets we use in this project. We start by providing a general overview of the datasets, before individually describing them in more detail. We then describe the data representation Enversion uses for their XGBoost model. After this, we describe and analyze different framings for representing the data and choose one that reflects the real-world usage the most. Finally, we analyze the missing values and the ratio of sepsis-positive to sepsis-negative admissions.

2.1 Overview

In this section, we provide a brief overview of the two datasets. The first dataset is from "*Early Prediction of Sepsis from Clinical Data – the PhysioNet Computing in Cardiology Challenge 2019*" [7], which consists of EHR data, where the goal is to predict sepsis. We refer to this dataset as PhysioNet Computing in Cardiology Challenge 2019 dataset or CinC2019 for short.

The second dataset consists of EHR data from the CROSS-TRACKS cohort, which has been preprocessed by Enversion [8]. We refer to this as Processed CROSS-TRACKS dataset or PCT for short. PCT has been accessible to us through our employment and tasks at Enversion A/S, through a virtual environment on their servers. As part of our terms of employment, we have signed a non-disclosure agreement as well as received the appropriate data protection training. All data presented in the report is anonymous and the dataset is not publicly available.
By using two datasets, we can:

- Evaluate how well our models generalize due to the different characteristics of the datasets.
- Allow others to reproduce the results of this report for CinC2019.
- Run experiments on CinC2019 locally, which is often quicker and easier, allowing for more exploration and experimentation in this project.

Both datasets consist of clinically relevant measurements collected over time for patients admitted to a hospital. An event is created every time a new measurement is recorded. These events are processed such that each clinically relevant measurements have one value for each hour. If no measurements occurred with the hour, the value is missing and stored as NaN. The data for a patient's admission is represented as one-hour timesteps, where each timestep contains all the clinically relevant measurements. We refer to these clinically relevant measurements as features.

Both datasets have vital sign features and laboratory value features. The vital sign features consist of values for monitoring the health of patients, e.g. heart rate, temperature, and blood pressure. The laboratory value features consist of values from laboratory experiments, e.g. the content of a certain substance in the blood. See Appendix A for a complete description of the vital signs and laboratory values in CinC2019 and PCT. In the following two sections, we describe CinC2019 and PCT in more detail.

2.2 PhysioNet Computing in Cardiology Challenge 2019 dataset

CinC2019 consists of two datasets with around $20,000$ EHRs each. We refer to these two datasets as dataset CinC2019A and CinC2019B, where the data in CinC2019A comes from Beth Israel Deaconess Medical Center and the data in CinC2019B comes from Emory University Hospital [1]. Each EHR contains data for a patient admitted to the intensive care unit (ICU) at that hospital. The EHR contains a total of 40 features and a sepsis label at each timestep, and are sorted according to the following categories:

- $1-8$: Vital Signs.
- $9-34$: Laboratory Values.
- $35-40$: Demographics - General information about the patient, such as age and gender.
- 41: Sepsis Label - Boolean label which indicates whether the patient is diagnosed with sepsis. The label changes from 0 to 1, six hours before the patient develops sepsis.

Figure 2.1 shows an illustration of the data for a patient's admission.

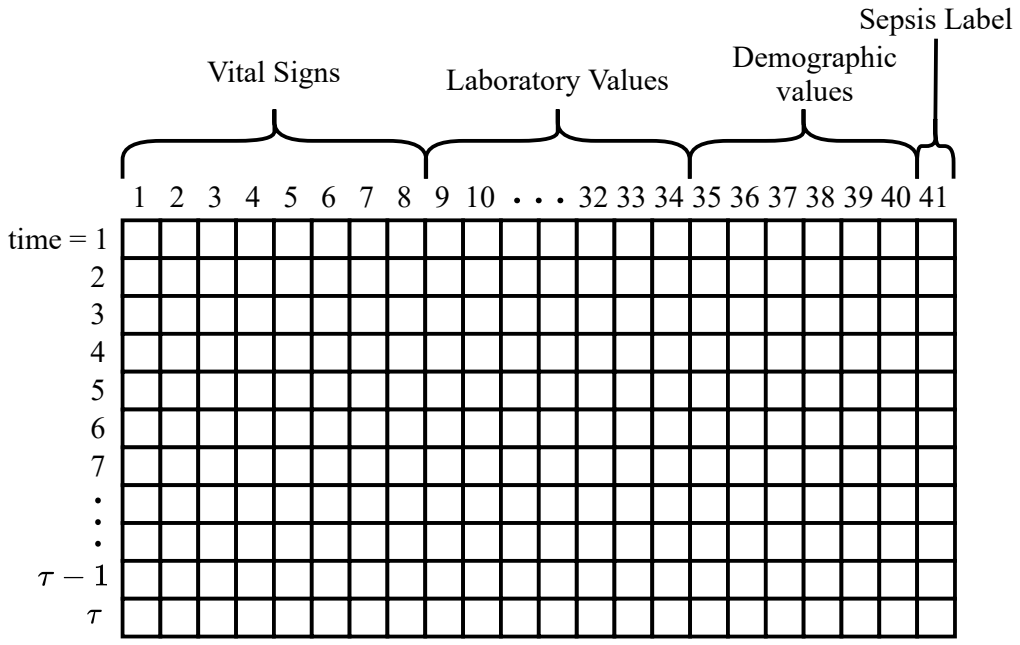


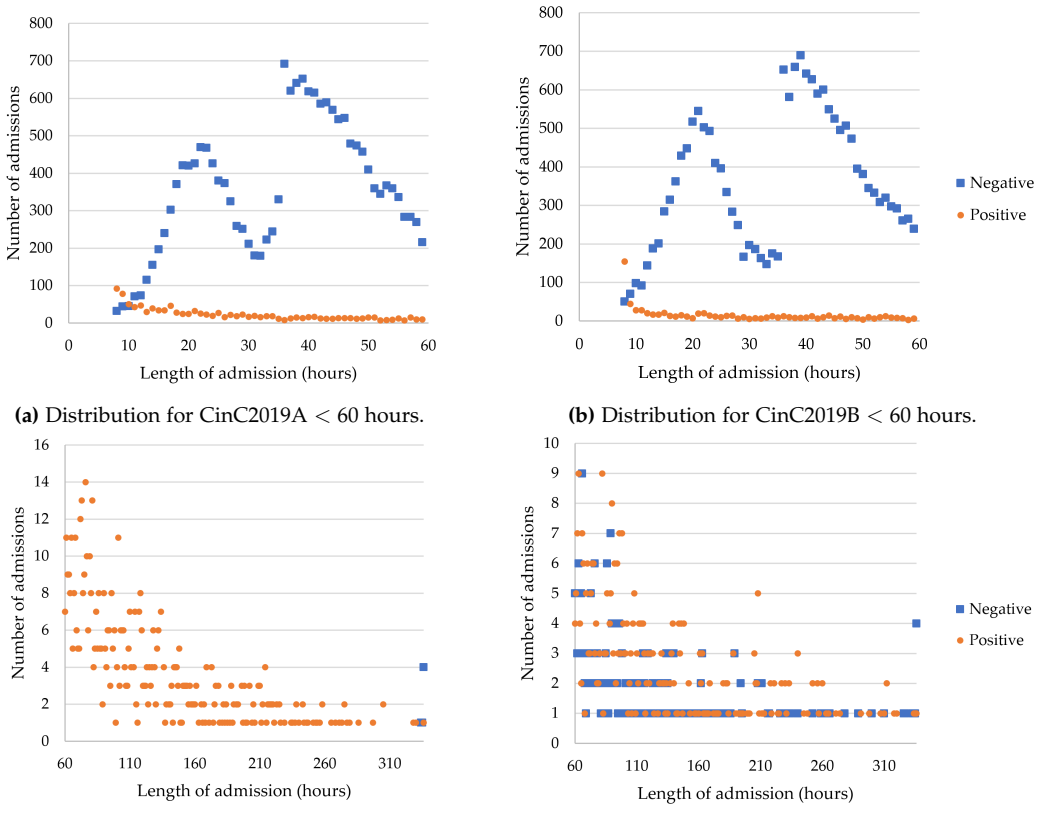
Figure 2.1: Data structure for the data of a patient's admission for a patient in CinC2019.

Additionally, admissions of less than eight hours are not included in the dataset, and admissions, where the patient develops sepsis before the fourth hour, are also not included. Admissions longer than two weeks are truncated to two weeks.

2.2.1 Distribution of Samples

Since CinC2019 admissions can have arbitrary length, we analyze the distribution of positive and negative samples given their admission time.

Figure 2.2 shows this distribution by plotting the number of sepsis-positive and -negative patients with respect to their admission length. Figure 2.2a and Figure 2.2b show the distribution of positive and negative samples with an admission length < 60 hours, for CinC2019A and CinC2019B respectively. Figure 2.2c and Figure 2.2d show the same, but with an admission length ≥ 60 hours. They are split like this, due to the significant drop in negative cases with admission lengths ≥ 60 hours. There is no indication in [1] why this occurs.



(a) Distribution for CinC2019A < 60 hours.

(b) Distribution for CinC2019B < 60 hours.

(c) Distribution for CinC2019A ≥ 60 hours.

(d) Distribution for CinC2019B ≥ 60 hours.

Figure 2.2: Distribution of positive and negative admissions given the length of admission for CinC2019A and CinC2019B.

The percentile of patients with an admission < 60 is shown in Table 2.1. This shows that the positive samples are more evenly spread out across the above and below the 60 hour mark, whereas the admission lengths for the negative samples are primarily < 60 hours. We also see that the positive to negative ratio changes for admission lengths ≥ 60 hours.

	CinC2019A	CinC2019B
% of all samples < 60	96.86%	96.95%
% of negative samples < 60	99.97%	98.82%
% of positive samples < 60	64.69%	66.11%
positive to negative ratio < 60	1 : 16	1 : 25
positive to negative ratio ≥ 60	105 : 1	1.7 : 1

Table 2.1: Percentile of samples < 60 hours admission length, and positive to negative ratio for admission lengths < 60 or ≥ 60.

Another interesting observation is how the positive samples are distributed. From Figure 2.3, we see the positive samples are more likely to have shorter admission lengths. In Figure 2.2a and Figure 2.2b, we even see that positive samples outnumber the negative samples for the first few admission lengths.

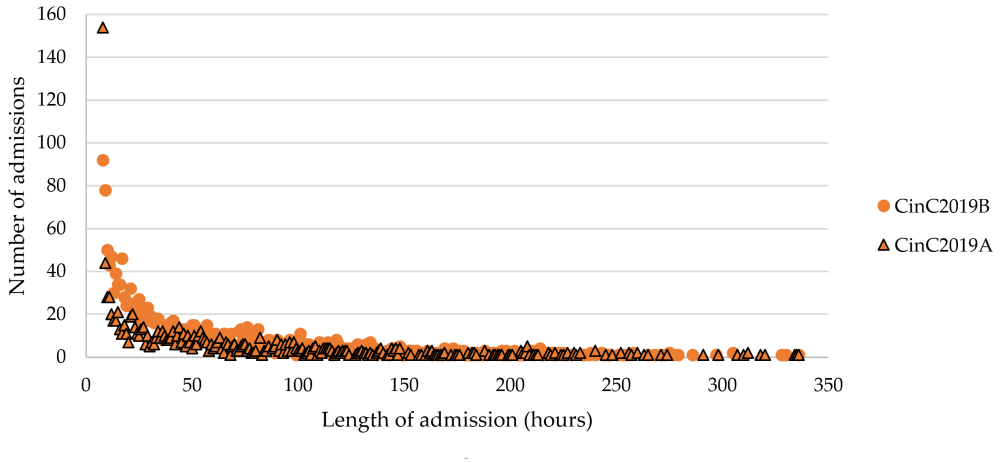


Figure 2.3: Distribution of positive samples from CinC2019A and CinC2019B given admission length.

The shortest admission length in CinC2019 is eight hours. When we write "a% and b%" in this paragraph, we refer to CinC2019A and CinC2019B respectively. Looking at the positive samples, we see that 5.1% and 13.5% of these samples have this exact admission length. This is in contrast to the negative samples, where it is only 0.2% and 0.3%. Considering admission lengths $\leq$ 16 hours we see that 25% and 29.9% of positive samples lies within this range, however, only 5.2% and 7.6% of negative samples lies within this range.

In summary, the negative samples vastly outnumber the positive samples. However, for admission lengths ≥ 60 hours, positive samples outnumber the negative samples. Additionally, there are far fewer negative samples for admission lengths ≥ 60 hours than < 60 hours. We also see that the shortest admission length is eight hours, and it is the most common admission length for the positive samples. Finally, we see that positive samples tend to have short admission lengths.

2.2.2 Short Admissions

During our analysis of the distribution of samples, we find that CinC2019 has some short admissions, where all the sepsis labels are 1. Therefore, we do not exactly know, when these patients developed sepsis. This is because the sepsis labels are set to 1 six hours before sepsis onset, and when all sepsis labels are 1, sepsis onset could be at any of the first six timesteps. However, patients, where sepsis onset occurs before the fourth hour, are not included in CinC2019 [1]. Therefore, we know sepsis onset is 4-6 hours after admission start in these cases. To avoid using data after sepsis onset, we only include the first four hours of the data for the short admissions where all the sepsis labels are 1.

2.3 Processed CROSS-TRACKS dataset

The population of patients in PCT is described as follows [9]:

- Hospitalized in a general ward at Horsens Regional Hospital from September 1, 2012 to December 12, 2018.
- 18 years old or older.
- Admission length is between 24 hours and 50 days.
- 19,976 admissions at the hospital.
- 6.25% sepsis prevalence of admissions.
- 13,134 unique residents.

PCT consists of data Enversion has processed from this population. The data is processed such that, a patient is evaluated one or more times during the admission. A sample is created each time the patient is evaluated for sepsis development. From the time of this evaluation, the sample contains the previous 24 hours up to this point. This data is used to predict whether the patient develops sepsis within some time frame, called the prediction window. The prediction window is 24 hours, meaning that if the patient develops sepsis within 24 hours, this sample is positive. This is described in more detail in Section 2.4 and Section 2.5.

PCT consists of 25 features, which are split into the following categories:

- $1-6$: Vital Signs.
- $7-25$: Laboratory Values.

An illustration of a sample from PCT can be seen in Figure 2.4.

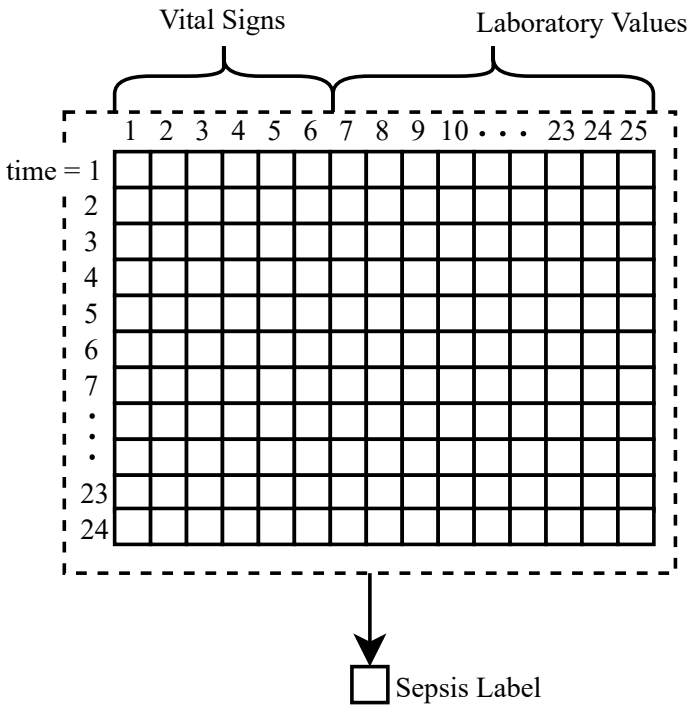


Figure 2.4: Data structure of a sample in PCT.

2.4 Delta Representation

The datasets we describe in Section 2.2 and Section 2.3 are time series, as they are made of measurements for each hour. Since XGBoost does not directly embed the temporal aspect of the time series data, Enversion has created a data representation, where the temporal aspect is encoded into the data, referred to as the delta representation.

If we wish to predict, at time t, whether a patient develops sepsis within some time span in the future, then we need to define how much data before t we consider, called the observation window, and how large this future time span is, called the prediction window. An example of an observation and prediction window can be seen in Figure 2.5.

The delta representation can be made by transforming the time series data we have from CinC2019 and PCT. This is done by splitting the observation window into two sections of the same length and aggregating them into two timesteps, as shown in Figure 2.5.

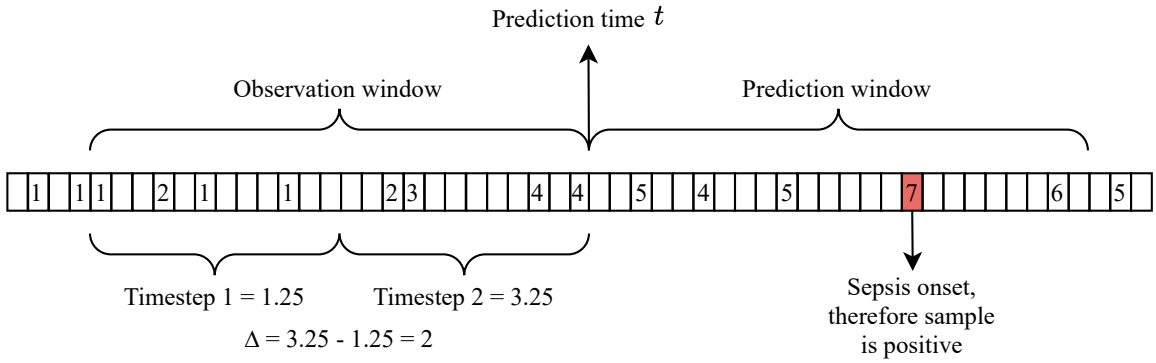


Figure 2.5: Timesteps used for the delta representation for one feature, where the patient develops sepsis within the prediction window (indicated by the red square). Each square represents an hour and if they contain a number there is a measurement for the given feature.

All the measured values within the timesteps are aggregated using the mean value, but other aggregation methods, such as min and max, can also be used. If only one of the two timesteps for a given feature has a value, the aggregated value for that timestep is assigned to both timesteps. For each feature in the time series representation, we make two features for the delta representation, one with the value of the last timestep and one with the difference between the two timesteps as shown in Figure 2.6. If we consider the example in Figure 2.5 as data for feature 1, then the value of F_1 and ΔF_1 in Figure 2.6 will be 3.25 and 2 respectively.

F_1	F_2	F_3	...	F_n	ΔF_1	ΔF_2	ΔF_3	...	ΔF_n

Figure 2.6: Feature vector for the delta representation.

2.5 Framing the Data to Fit the Problem

In this section, we describe different ways to frame the problem of sepsis detection, based on [9]. We define framing as how the model will be used in the real world, which includes aspects such as at which times in the admission the model makes an evaluation, and how the prediction of the model is supposed to be interpreted.

Choosing the right framing is important such that it reflects the real-world usage of the model. If the framing significantly differs from the real-world use case, then the model might not be applicable in the real world. Additionally, when comparing models, it is important to compare them using the same framing. If we do not use the same framing, metrics can become incomparable due to various factors, e.g. the difference in positive to negative class ratio.

The framing of choice can impact the model in various ways. Lauritsen et al. [9] show that a model can interpret a feature differently depending on the framing. For one framing,

a higher value of the feature correlates with a higher chance of sepsis. For another framing, this relationship is inverted, where a higher value correlates to a lower chance of sepsis. This is problematic as a feature should hold the same meaning across different framings.

Another problem is that different framings almost always have different positive to negative ratios. In our previous project [2], we saw that changing the positive to negative ratio of the training data affected the calibration of models when evaluated on data with the original positive to negative ratio. It is important that a model is well-calibrated when used in safety-critical settings, like in a hospital [6]. We describe calibration of models in more detail in Section 8.2.
Figure 2.7 illustrates the four framings used in [9], which we describe in the following sections.

2.5.1 Fixed Time to Onset

For the fixed time to onset framing, we have one sample per admission. If the patient develops sepsis and we use a prediction window of 12 hours, the sample is taken 12 hours before the time of sepsis onset. If the patient does not develop sepsis, the sample is taken at a random time during the admission.

2.5.2 Sliding Window

In the sliding window framing, we have multiple samples per patient, each starting with a fixed interval between them (e.g. one hour). If the patient develops sepsis within the prediction window, it is a positive sample and negative otherwise.

2.5.3 Sliding window with dynamic inclusion

The sliding window with dynamic inclusion is similar to the sliding window framing, but samples only start getting collected when some criteria are met. In Figure 2.7, we see the sliding window first starts when the SOFA score is above 0, which is a score used when diagnosing patients with sepsis. For more information about the SOFA score, see Appendix B.

2.5.4 On clinical demand

For the on clinical demand framing, we also have multiple samples per patient. Here, each sample is taken when the clinical staff performs an early warning score (EWS) assessment. The sample is positive if the patient develops sepsis within the prediction window and negative otherwise.

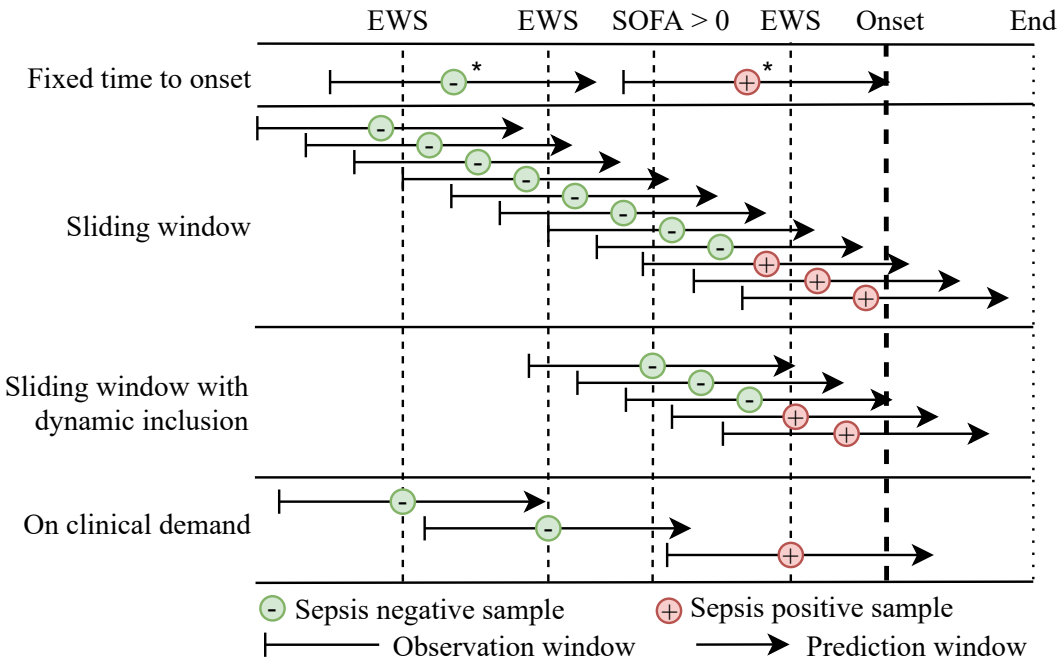


Figure 2.7: Visualization of the four framings. The * on fixed time to onset indicates that each sample is from different admissions. This figure is based on Figure 2 from [9]

.

2.5.5 Choice of Framings

In this section, we discuss the four framings, and how they reflect the real-world usage for CinC2019 and PCT. From this discussion, we choose one framing for CinC2019 and one framing for PCT. We start by discussing framings for CinC2019.

PhysioNet Computing in Cardiology Challenge 2019 dataset

As CinC2019 does not have data for EWS assessments, we discard on clinical demand. We also discard sliding window with dynamic inclusion as we consider it important to evaluate sepsis over the entire admission as patients admitted to an ICU are in a critical condition. As mentioned in Section 2.2, CinC2019 consists of data from ICUs. In ICUs, data is collected more frequently by the clinical staff than in the general wards [9].

Fixed time to onset does not capture this aspect very well, as we only have a single sample per admission. Essentially, the clinical staff is expected to randomly evaluate the patient once during the admission and then never again, regardless of how the patient's condition evolves during the admission.

Sliding window captures this aspect well by providing a sample regularly (e.g. every hour) from the admission. In this framing, the patient is continuously evaluated at regular intervals. We argue that regularly sampling data during an admission matches well with continuously evaluating a patient's condition, and therefore choose the sliding window framing for CinC2019. As the data is aggregated over one hour, one hour is the shortest interval we

can use. Since the patients are at an ICU, using longer intervals than one hour might not be frequent enough. Therefore, we use one-hour intervals for the sliding windows framing.

Processed CROSS-TRACKS dataset
As mentioned in Section 2.3, the data from PCT is collected from general wards. Due to this, the data is collected less frequently than at ICUs and with more random intervals [9]. Therefore, we need to consider a framing that fits well with these circumstances.

We choose to discard fixed time to onset due to the same points as mentioned for CinC2019. Especially due to the restriction of only evaluating a patient once during the admission.

Sliding window and sliding window with dynamic inclusion are decent candidates. Since patients are not evaluated as frequently as in CinC2019, we can simply increase the interval between samples, which simulates this. However, we have data for when EWS assessments were performed for PCT. The time these assessments were performed directly correlates to when the clinical staff considered it important to evaluate the status of the patient. The sliding window framings only provide multiple samples at arbitrary points. Since we know when the EWS assessments are performed, and we want to limit the number of framings we have to consider, we argue that on clinical demand most closely reflects the real-world usage. Due to this, we choose on clinical demand as the framing for PCT.

2.6 Prediction Window Details

We have now presented a general description of CinC2019 and PCT and chosen the framings. We want to analyze the characteristics of the datasets with the applied framings. However, there are still some details regarding the prediction window which can affect these analyses. Due to this, we start by choosing the size of the prediction windows.

Processed CROSS-TRACKS dataset
Enversion has prepared PCT with a 24-hour prediction window. Because changing the prediction window size for PCT is not easily done, and we do not have the domain knowledge to suggest a better prediction window size for a real-world use-case, we use a 24-hour prediction window for PCT.

PhysioNet Computing in Cardiology Challenge 2019 dataset
As we have access to the raw CinC2019 time series data, we need to make more considerations of how to handle the data. In [1], they describe a utility function for sepsis-positive patients for CinC2019, which describes the utility of predicting true positives and false negatives at time t in relation to sepsis onset. Here they define that predicting a true positive more than 12 hours before sepsis onset has negative utility. 12 to 6 hours before onset has a positive increasing utility, which peaks at 6 hours. The utility then decreases after 6 hours before sepsis onset and is negative again at 3 hours after sepsis onset. Since we use the sliding window framing for CinC2019, we evaluate the patient at every timestep. Based on this, we decide to use a

12-hour prediction window as there is some utility for predicting sepsis correctly at every timestep from 12 hours before sepsis onset.

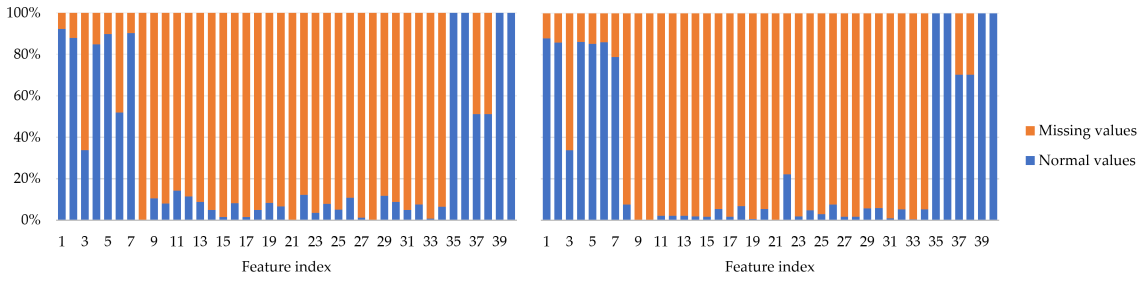
2.7 Missing Values Analysis

In this section, we analyze the missing values in CinC2019 and PCT both as time series and after converting that data to the delta representation.

2.7.1 Time Series [1]

When analyzing CinC2019, we use the raw time series data, before the sliding window framing is applied. As we do not have the raw data available for time series PCT, we analyze it with the on clinical demand framing.

PhysioNet Computing in Cardiology Challenge 2019 dataset
We find the missing rate for each feature, by counting the number of missing values and actual values for each feature in the timesteps, for each dataset, which is displayed in Figure 2.8. Both datasets in Figure 2.8 show that some features are missing values more frequently than others. The observation rate is often related to its category, described in Section 2.2.



(a) Missing values in CinC2019A.

(b) Missing values in CinC2019B.

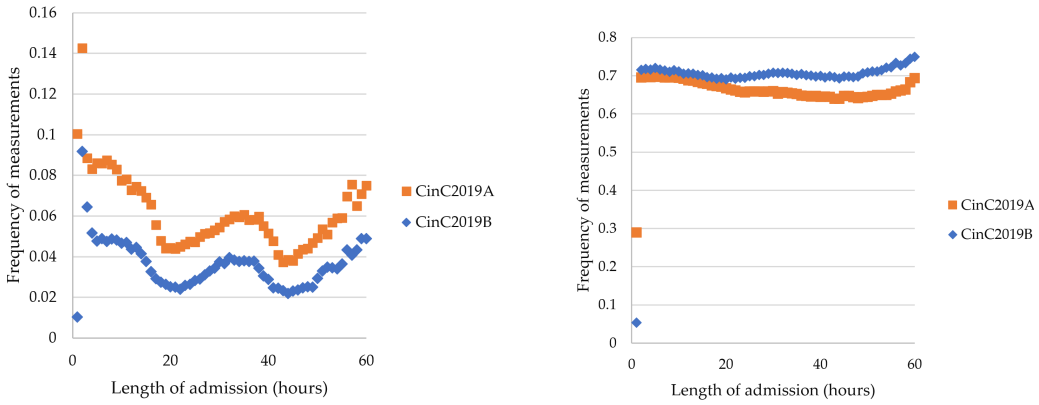
Figure 2.8: Illustration of the observation rate in blue and missing rate in orange for CinC2019.

6 out of 8 features in the Vital Signs category include measurements for over 80% of the data points. The third feature, which is temperature, is missing over 60% of the values in both CinC2019A and CinC2019B, despite being a vital sign. The eighth feature, which is end-tidal carbon dioxide, has 100% missing values in CinC2019A and around 95% missing values in CinC2019B. Features within the laboratory values category are very sparse and close to all features are missing 90% of the values. Missing values in the demographic features are only present in the features indicating which ICU the patient is located at. The only demographic feature that changes value over time is the length of admission feature (called ICULOS), which is increased by one at every hour. The other demographic features: age, gender, ICU, and admission time remain the same.

[1] This section is a slightly modified version of section 2.2 and 2.3 from our previous semester report [2].

To find out how the frequency of measurements changes over time, we find the percentage of patients with measurements for each lab value at every timestep. In Figure 2.9a, we plot the average percentage over all the lab values to get one point for each timestep. Figure 2.9a shows that CinC2019 has a few more measurements in the beginning, and seems to peak every 24 hours.

Figure 2.9b shows the frequency of vital sign measurements. The frequency is high and stays consistent across the admission.



(a) Lab values. **(b)** Vital signs.

Figure 2.9: Frequency of measurements over time in CinC2019.

Concerning the sparsity of the data, we examine how frequently each of the measurements occur by taking the average time between the measurements of each patient's features, and plot the average value for each feature, as seen in Figure 2.10. Here we see that most vital signs are measured hourly, while lab values are measured between every 3 hours to once a day.

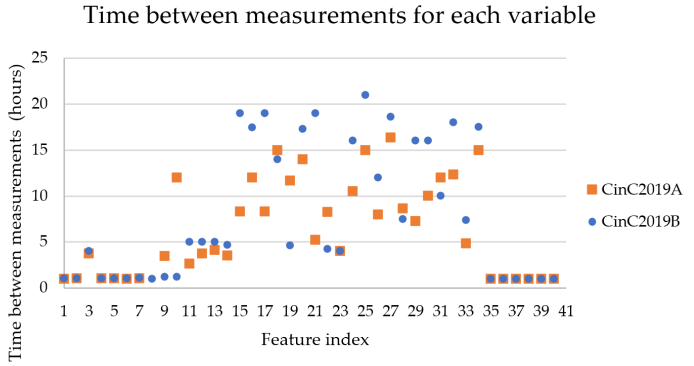


Figure 2.10: Average time between measurements for each feature.

Processed CROSS-TRACKS dataset

The missing rate for PCT is around 90% for the vital signs and more than 97% for laboratory values, which is significantly higher than for CinC2019.

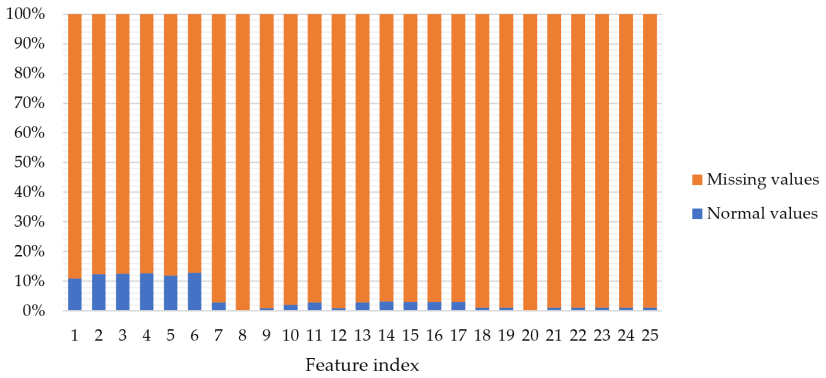
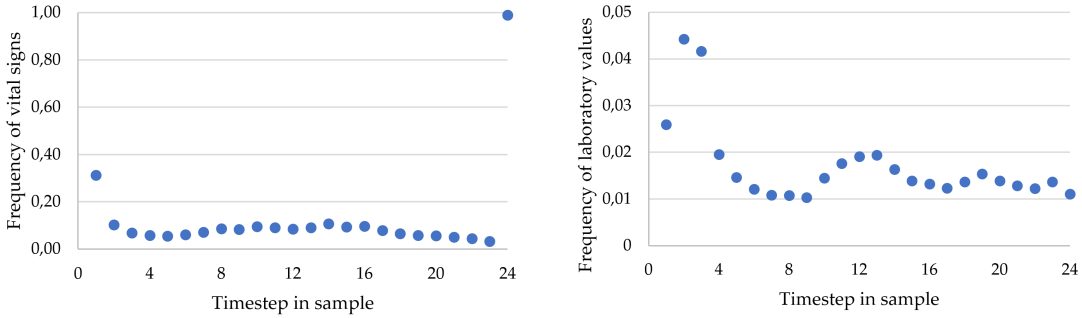


Figure 2.11: Illustration of the observation rate in blue and missing rate in orange with orange in PCT.

If we look at how the frequency of measurements change over time, as shown in Figure 2.12a and Figure 2.12b, we see that there are more measurements at the beginning of the sample, and for vital signs, there is a spike in the last timestep. We suspect that there are more vital sign measurements in the last hour because the clinical staff is preparing to perform the EWS assessment. We see that the frequency of measurements are much lower compared to CinC2019, which supports our assumption that data from general wards is more sparse than data from ICUs.



(a) Vital signs.

(b) laboratory values.

Figure 2.12: Frequency of vital sign- and laboratory value measurements over time in PCT.

For CinC2019, we did an "average time between measurements"-analysis. However, we do not do this for PCT, as we do not have data for the entire admission. This makes the time between measurements uncertain for samples with no or one value for a feature.

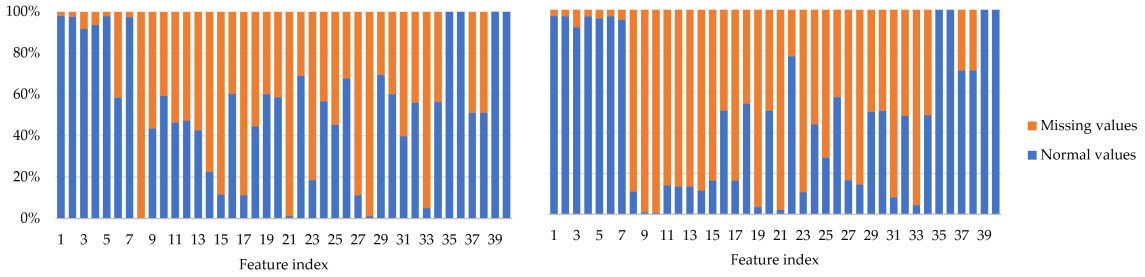
2.7.2 Delta Representation

As the values for samples in the delta representation are aggregated, we analyze the missing values for the delta representation. While we have a timestep feature and a delta feature in the delta representation for each feature in the time series representation, we consider the missing rate for the timestep feature and delta feature as one in this section. This is because the delta feature is only missing if the timestep feature is missing and vice versa. We generally expect to see lower missing rates for the delta representation compared to the time series representation,

due to the aggregation step when creating the delta representation. The analysis for CinC2019 is done with the sliding window framing and an observation window of 12 hours. For PCT, we use the on clinical demand framing with an observation window of 24 hours. We describe the reason for these observation window sizes in Chapter 9.

PhysioNet Computing in Cardiology Challenge 2019 dataset

Figure 2.13 shows the missing rates for each feature in the CinC2019 delta representation. Compared to the missing rates shown in Figure 2.8, the missing rates for the delta representation are generally lower.



(a) Missing values in CinC2019A.

(b) Missing values in CinC2019B.

Figure 2.13: Illustration of the frequency of missing values in CinC2019 with the sliding window framing and delta representation.

We see the most significant change in missing rates for the laboratory values. On the time series representation, the laboratory values' missing rates are around 90%. However, Figure 2.13 shows multiple laboratory values have missing rates as low as 40%. In Figure 2.13b, we even see feature 22 with a missing rate slightly above 20%. Overall, we generally see lower missing rates for vital signs and laboratory values compared to the time series representation

Processed CROSS-TRACKS dataset

The missing rates of each feature of PCT's delta representation are shown in Figure 2.14. Compared to the missing rates of the time series representation (Figure 2.11), we see significantly lower missing rates for vital signs. In the time series representation, the vital signs consist of around 90% missing values, whereas they are close to 0% missing values for the delta representation, with the exception of feature 1. In the delta representation, the laboratory values have between 90% to 75% missing values, as opposed to the time series representation with 97% missing values.

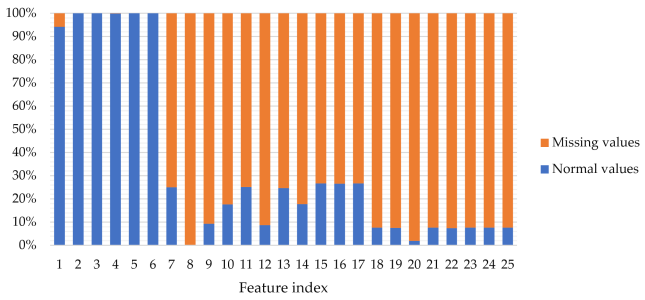


Figure 2.14: Illustration of the frequency of missing values in PCT with the delta representation.

Similarly to CinC2019, we see that the missing rates are significantly lower for the delta representation compared to the time series representation.

2.8 Patient Count & Positive to Negative Ratio

In this section, we describe the ratio between positive and negative samples in CinC2019 and PCT. Table 2.2 shows the number of positive and negative samples, and the ratio between them, for the raw data and the selected framings chosen in Section 2.5.5.

Dataset	Framing	Positive	Negative	Total	Positive to negative ratio
CinC2019A	None	1,790	18,546	20,336	1:11
CinC2019B	None	1,142	18,858	20,000	1:18
CinC2019	None	2,932	37,404	40,336	1:13
CinC2019A	Sliding window	19,228	764,185	783,413	1:41
CinC2019B	Sliding window	11,591	746,030	757,621	1:65
CinC2019	Sliding window	30,819	1,510,215	1,541,034	1:49
PCT	On clinical demand	2,636	337,814	340,450	1:128

Table 2.2: Counts and ratios of positive and negative samples of the datasets given their framings.

For CinC2019, we can see that applying the sliding window framing to raw data makes the imbalance of the positive and negative class larger. However, PCT is still more imbalanced than CinC2019 with the sliding window framing.

2.9 Summary

In this section, we summarize our choices and analyses from this chapter. We present our main findings in Table 2.3.

A major characteristic of CinC2019 and PCT is a large number of missing values. For the time series representation, we tend to see higher missing rates compared to the delta representation. PCT tends to have higher missing rates than CinC2019. Vital signs in both datasets tend to have lower missing rates compared to the laboratory values. Additionally, we

have to take the positive to negative ratio into account. Both datasets are imbalanced, with a majority of samples being negative. This provides a challenge since it can be difficult to train a model on imbalanced data [10]. However, attempting to balance this can affect the calibration of a model [2]. We need to consider these aspects when we create our model.

Attribute	CinC2019A	CinC2019B	PCT
Selected framing	Sliding window		On clinical demand
Vital signs	8		6
Laboratory-categories (e.g. Blood tests)	25		19
Average time between vital signs (hours)	1.4	1.6	N/A
Average time between laboratory measurements (hours)	11	13	N/A
Missing rate vital (Time series)	34%	32%	90%
Missing rate lab (Time series)	93%	96%	97%
Missing rate vital (Delta)	21%	15%	1%
Missing rate lab (Delta)	59%	73%	82%
Demographic features	Yes		No
Aggregation time	One hour		
Number of samples	783,413	757,621	340,450
Positive to negative ratio	1:41	1:65	1:128
Country	USA		Denmark
Sepsis definition	Sepsis-3 definition		

Table 2.3: Comparison between the datasets from PhysioNet and Enversion.[2]

[2]This table is a slightly modified version of Table 2.3 from our previous semester report [2].

Chapter 3

Problem Statement

In this chapter, we summarize our findings from our introduction (Chapter 1) and data description (Chapter 2) and from that propose a problem statement. From Chapter 2, we found that CinC2019 and PCT have relevant characteristics, which are worth considering when addressing the problem of sepsis prediction.

In Section 2.5, we found that framing the data correctly to fit the problem is an important aspect, and therefore chose separate framings for CinC2019 and PCT, which we considered best reflects the real-world usage. It is important to use the correct framing, as it affects how the model understands the data and the calibration of the model. We saw in Table 2.2, that applying the chosen framings on the two datasets results in more class imbalance. From our previous semester project, we found that it can be difficult to train a model when the datasets are very imbalanced [2]. Therefore it is something, that we have to consider during our model development.

We considered how the missing values are represented in CinC2019 and PCT in Section 2.7. Here we saw that PCT has significantly higher missing rates for the time series representation than CinC2019. We also compared the features and saw that there is a significant difference between the missing rates of the vital signs and the laboratory values for both CinC2019 and PCT. Because the missing rate is a significant aspect of CinC2019 and PCT, we have to address it when creating a model for sepsis prediction.

From our previous semester project, we also found the importance of calibration when creating models used in healthcare [2]. Both the performance and calibration of the model are important, and we note that better performance often results in worse calibration. Therefore, we consider this trade-off between performance and calibration.
From this, we construct the following problem statement:

> **How can we create a well-performing, well-calibrated neural network model for predicting sepsis from high missing rate EHR data?**

- How should the missing values be handled?
- Which evaluation criteria should be used to evaluate the models' performance and calibration?
- How do we balance the trade-off between performance and calibration?
- How does our model compare to Enversion's existing model (XGBoost)?

Part II

Model Design

Chapter 4

Characteristics

In this chapter, we describe the relevant characteristic we consider when designing models for CinC2019 and PCT. These characteristics are based on our problem statement and findings in our data analysis (Chapter 2). In the following sections, we describe the characteristics, which we have separated into two groups of importance: required and preferred.

4.1 Required

Model type
As mentioned in our problem statement, we only consider neural networks, due to the many tools and methods for analyzing them.

Time Series
The models must also be able to operate on CinC2019 and PCT, which are time series datasets. In Section 2.4, we describe that models that are not designed for time series data can use the delta representation. However, information is lost when creating the delta representation from the time series representation. Due to this, and to limit the scope of this project, we only focus on creating models that is designed to capture the temporal aspect of the data.

Missing rate
In Section 2.7, we see that the missing rate in CinC2019 and PCT typically is above 80%, except for vital signs in CinC2019, where it is between 10% to 60%. We addressed this problem in our problem statement, and therefore require the missing values to be handled. We see two ways to handle them: 1) Replacing the missing values with an estimate before giving the model the data (imputation). 2) Making a model that can handle data with the missing values. We used simple imputation methods in our previous semester project [2] to handle the missing values. Therefore, in this semester, we attempt to handle the missing values as an integrated part of the model development.

4.2 Preferred

Simplicity
It is easier to explain a simpler model compared to a complex one, where "explain" both refer to explaining how the model works to the medical staff and model explainability, which is how

much each input feature contributed to the model's prediction. While simplicity is a preferable characteristic it is not strictly necessary, as we are more interested in a well-performing and well-calibrated model.

Not Recurrent

Enversion uses tools for explaining the impact that each feature has on the prediction of the neural network. These tools do not work well with RNNs, which makes RNNs less desirable to use. Despite this, RNNs are one of the most common types of neural networks for time series data [2], and discarding them limits the types of models we can create, and also how well they can perform. For these reasons, it is preferable to create a non-RNN model, but this is not required.

Handle Class Imbalance

The imbalance of positive and negative class ratio in data can hinder a model from learning, and thus reduce its performance [11]. This problem is known as the class imbalance problem (CIP). Due to the imbalanced class ratio in PCT and CinC2019, it is preferred that the model addresses the CIP.

Chapter 5

State of the Art

In this chapter, we present models and ideas which can help create a model that has the described characteristics from Chapter 4. We base this chapter on papers that describe state of the art methods tested on similar datasets to CinC2019 and PCT. We separate this into two parts, one that focuses on models and one that focuses on ideas that can be added to an existing model.

5.1 Models

To get an overview of state of the art models for datasets similar to CinC2019 and PCT, we read the papers about the models described in the review "*A Review of Deep Learning Methods for Irregularly Sampled Medical Time Series Data*" [12], which compares state of the art methods for time series classification on multiple datasets, including PhysioNet's Challenge 2012 dataset (CinC2012) and CinC2019. CinC2012 is used for mortality prediction, where CinC2019 is used for predicting sepsis, but they are otherwise similar, as both are time series data and share many of the same features [13]. To not only base our research on one source, we also include other papers we found during our research. From this research, we found that most of the models, used on the datasets similar to CinC2019 and PCT, were recurrent neural networks (RNN). As we do not have time to experiment with the aspects of all of the models, we select the two models that we deem most suitable to build on top of to achieve most of the desired characteristics from Chapter 4. The models we choose are GRU-D and BRITS. As we do not consider the other models further, we give an overview of them and a short description of why we did not choose them in Appendix D.

GRU-D and BRITS use two representations to exploit the missing patterns in multivariate time series data. The first representation indicates when a measurement is observed, called mask, and the second representation indicates when a measurement was last observed, called time interval. These representations are collectively referred to as the missingness representations. We argue GRU-D and BRITS are the best candidate models, as they use the missingness representations to handle the missing values, and achieve better performance. While GRU-D and BRITS both use the missingness representations, they attempt to solve the problem of missing values in two different ways. GRU-D focuses on weighing the importance of the last observed measurement based on the missingness representation and the mean value for the given feature. BRITS focuses on handling the missing values with imputation, based on the observed measurements and the missingness representations. BRITS attempt to achieve

a more accurate imputation by using a bidirectional approach, which means that it considers the time series data going forward and backward in time. Another reason to consider GRU-D is because it is often used as a baseline in comparisons with other models and often performs well in these comparisons [12][14][15][16]. We describe the missingness representations and the models in further detail in Section 6.4 and Section 6.5.

5.2 Ideas

Besides finding relevant models, we also describe ideas that might be a useful addition to the models we design. From our research on how to handle the characteristics of CinC2019 and PCT (e.g. imbalanced classes, high missing rate, and multivariate time series) we find the following:

LSTM-CNN

As Long short-term memory (LSTM) networks and convolutional neural networks (CNN) find different characteristics in the data, Li et al. [17] suggest that combining them may give better results than choosing one of them. Li et al. [17] combine the two types of architectures by giving them the same input and concatenates their output as shown in Figure 5.1.

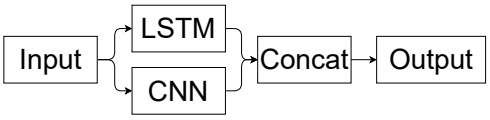


Figure 5.1: LSTM-CNN.

This is a simple idea that is easy to implement, but it does not address the challenges for our data (CIP and high missing rate). Therefore, we choose not to experiment with this idea.

Weighting Positive and Negative samples

Both CinC2019 and PCT have imbalanced classes, which is problematic due to frequency bias [10]. In our previous semester, we tried to solve the CIP by sampling the data, but found that it worsens the calibration and performance of the models [2]. Geng et al. [18] suggest using a loss function that weighs the classes differently, to mitigate the CIP. They found that a weighted loss function improved the performance of most of their models, but they did not address how it affected the models' calibration. Fernando et al. [10] also used a weighted loss function, which improved the performance and calibration of their models.

As the classes in the datasets are imbalanced, the CIP is something we have to consider. Therefore, we experiment with weighting the classes in the loss function. We describe how we are going to weigh the classes in the loss function in Section 11.1.

Extracting Additional Features

Singh et al. [19] found that the observation rate of different measurements is higher for

sepsis-positive patients compared to sepsis-negative patients. FiO2, EtCO2, and Lactate are examples of features, that on average have an observation rate 2-4 times higher for sepsis-positive patients than sepsis-negative patients for CinC2019 [19]. The observation rate of the features is something that we can extract from CinC2019 and PCT.

Extracting additional features from the data is something that we want to experiment with, as we have seen multiple papers describing additional data representations that could improve a model's performance [20][19][21]. Besides the observation rate feature, we might also consider extracting other features from CinC2019 and PCT. However, as models for time series data do not have a direct way of using tabular features (features with a single value, that do not change over time), we discuss possible solutions to this problem in Section 7.6.

Chapter 6

Neural Network Theory

In this chapter, we establish the theory underlying the state of art architectures we chose to build on top on in Section 5.1. We do not describe the fundamental neural network theory here, but we have included the fundamental theory description from our previous semester project [2] in Appendix C.1.

As the state of the art methods we have chosen are recurrent, we start by given an introduction to RNNs. We then describe the theory of LSTMs, as we use it as the recurrent network in BRITS. Before describing the theory of GRU-D, we establish the concepts of GRU, which is a specialization of RNN, from which we extend the GRU-D concepts. Lastly, we describe the theory of BRITS.
Before describing the underlying theory for these models, we introduce the syntax that we use in this chapter:

- Vectors are lowercase boldface like $\boldsymbol{x}$.
- Matrices are uppercase boldface like $\boldsymbol{X}$.
- For matrix multiplication and addition we use $\cdot$ and $+$ respectively.
- $\odot$ refers to element-wise multiplication.
- $\circ$ refers to concatenation.

The syntax for multivariate time series data with D features and τ timesteps is denoted as $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(\tau)}) \in \mathbb{R}^{\tau \times D}$, where $x_d^{(t)}$ is the value for feature d at timestep t.

6.1 Recurrent Neural Networks [1]

This section is based on information from [22]. As neural networks are fixed in the number of inputs, they are not able to handle the processing of sequential data of variable length, for example in the form of EHRs. Sequential data is data where one value comes before another. For our case, it relates to time, as we have time series data. Conceptually, an RNN is a cell composed of a number of units/hidden states; one state for each timestep in the input sequence. Each state propagates information to the state of the following timestep, which is where the recurrence occurs in an RNN. The states of the RNN shares a set of parameters, meaning that an RNN cell has a fixed number of parameters independent of the length of the input sequence. This independence allows an RNN to take input sequences of variable length. A general representation of an RNN cell can be seen in Figure 6.1, where:

[1] This section is a slightly modified version of Section 4.2 from our previous semester report [2].

- $\boldsymbol{x}$ is an input vector containing sequential data.
- $\boldsymbol{o}$ is an output vector containing outputs for each hidden state. However, this can also be modeled as a single output from the last hidden state.
- $\boldsymbol{h}$ is a hidden state that is updated through each recurrence step. The cyclic arrow on $\boldsymbol{h}$ represents the recurrence, and that information from one hidden state can be propagated to the following hidden state.

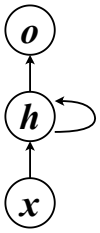


Figure 6.1: Cyclic representation of RNN cell.

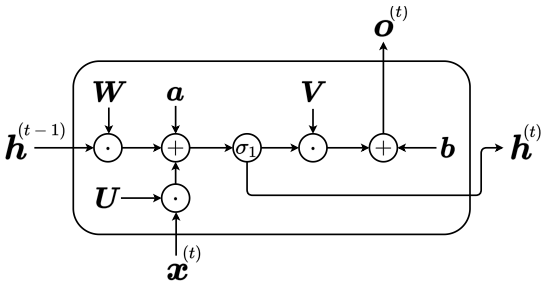


Figure 6.2: Representation of a single hidden state in an RNN cell.

Figure 6.1 shows the general structure of an RNN cell, however, it is not visible which parameters are shared between the hidden states (across the network). Therefore, Figure 6.2 shows a more detailed representation of a single recurrent step of an RNN cell.

- t is the current timestep, where $t \in [1..\tau]$ and τ is the number of timesteps.
- $\boldsymbol{h}^{(t-1)}$ is the previous hidden state, from which information is propagated.
- $\boldsymbol{h}^{(t)}$ is the current hidden state, from which information is propagated to the subsequent hidden state $\boldsymbol{h}^{(t+1)}$.
- $\boldsymbol{x}^{(t)}$ is the input at timestep t from the input sequence $\boldsymbol{x}$.
- $\boldsymbol{o}^{(t)}$ is the output at timestep t. Here, we have the two possibilities for outputs, either the RNN cell outputs at each timestep, as shown in Figure 6.2, or it omits all outputs besides the one at timestep τ.
- $\boldsymbol{W}$, $\boldsymbol{V}$, and $\boldsymbol{U}$ are weight matrices, which are used to parameterize inputs and outputs of the RNN cell. The dimensionality of the matrices are: $\boldsymbol{W} \in \mathbb{R}^{m \times m}$, $\boldsymbol{V} \in \mathbb{R}^{|\boldsymbol{x}| \times m}$, and $\boldsymbol{U} \in \mathbb{R}^{m \times |\boldsymbol{x}|}$, where m is the number of features in the hidden state.
- $\boldsymbol{a}$ and $\boldsymbol{b}$ are biases. These biases are vectors of length m.
- σ_1 is an activation function.

The weight matrices and biases are shared between hidden states. This parameter sharing means that however many recurrent steps are performed, the number of network parameters stays the same.

Forward propagation in an RNN is done through Equation 6.1a, and Equation 6.1b, which is directly related to the representation in Figure 6.2. Equation 6.1c is used to compute predictions, where a second activation function (σ_2) is used.

$$\boldsymbol{h}^{(t)} = \sigma_1\Big(\boldsymbol{W} \cdot \boldsymbol{h}^{(t-1)} + \boldsymbol{a} + \boldsymbol{U} \cdot \boldsymbol{x}^{(t)}\Big) \tag{6.1a}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{V} \cdot \boldsymbol{h}^{(t)} + \boldsymbol{b} \tag{6.1b}$$

$$\hat{\boldsymbol{y}}^{(t)} = \sigma_2\Big(\boldsymbol{o}^{(t)}\Big) \tag{6.1c}$$

Equation 6.1 can be adjusted to only output once, if only $\boldsymbol{o}^{(\tau)}$ is calculated, omitting all other timesteps.

In Figure 6.2 the information propagated to the following hidden state is the output of activation function σ_1. However, it is also possible to model the information propagated as the output $\boldsymbol{o}^t$. In that case, Equation 6.2 is used.

$$\boldsymbol{o}^{(t)} = \sigma_1\Big(\boldsymbol{W} \cdot \boldsymbol{o}^{(t-1)} + \boldsymbol{a} + \boldsymbol{U} \cdot \boldsymbol{x}^{(t)}\Big)\boldsymbol{V} + \boldsymbol{b} \tag{6.2}$$

6.1.1 Backpropagation Through Time

The gradients for optimizing an RNN are calculated following the same procedure as for finding gradients using backpropagation (see Section C.1). This means that we want to find the partial derivatives of the loss function $\boldsymbol{C}$ with respect to each of the parameters, $\boldsymbol{W}$, $\boldsymbol{V}$, $\boldsymbol{U}$, $\boldsymbol{a}$, and $\boldsymbol{b}$. When looking at the recurrent representation at Figure 6.1, it can be problematic to see, how gradients are calculated. If the RNN is instead unfolded as shown in Figure 6.3, it resembles a neural network, which makes it more clear, how the gradients are calculated. As the gradients are calculated in relation to timesteps, this form of backpropagation is called backpropagation through time (BPTT).

In Figure 6.3, and in the following description of BPTT, we consider an RNN with a single output at $\boldsymbol{h}^{(\tau)}$.

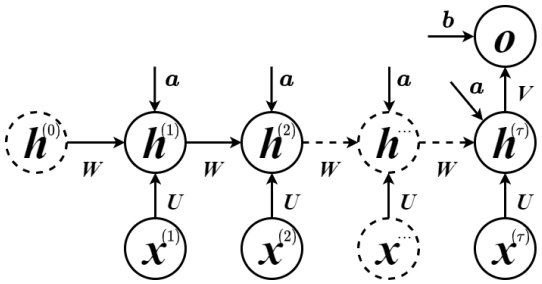


Figure 6.3: Unfolded RNN cell where information is propagated from the hidden state directly, and where the cell has a single output.

When considering $\boldsymbol{V}$ or $\boldsymbol{b}$, the gradient is calculated as:

$$\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{V}} = \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{o}} \cdot \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{V}} \tag{6.3a}$$

$$\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{b}} = \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{o}} \cdot \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{b}} \tag{6.3b}$$

As $\boldsymbol{o}$ is directly dependent on $\boldsymbol{V}$ and $\boldsymbol{b}$, no hidden state needs to be considered when calculating the gradients. This makes the gradients more trivial to calculate, compared to the gradients related to $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{a}$.

As $\boldsymbol{C}$ depends on $\boldsymbol{W}$ multiple times in a forward propagation of an RNN, the partial derivative of $\boldsymbol{C}$ with respect to $\boldsymbol{W}$, shown in Equation 6.4a, is more complex. This is also the case for the gradients with respect to $\boldsymbol{U}$ and $\boldsymbol{a}$, which are calculated with Equation 6.4b and Equation 6.4c respectively. To distinguish between gradients related to $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{a}$ at different timesteps, we subscript these with t of the hidden state. This means that if this notation was introduced in Equation 6.1a, $\boldsymbol{W}$, $\boldsymbol{U}$, and $\boldsymbol{a}$ would be subscripted with t. It is important to note that this is only a notation and does not mean that parameters are not shared.

$$\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{W}} = \sum_{t=1}^{\tau} \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{o}} \cdot \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^{(t)}} \cdot \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{W}_{(t)}} \tag{6.4a}$$

$$\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{U}} = \sum_{t=1}^{\tau} \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{o}} \cdot \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^{(t)}} \cdot \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{U}_{(t)}} \tag{6.4b}$$

$$\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{a}} = \sum_{t=1}^{\tau} \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{o}} \cdot \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^{(t)}} \cdot \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{a}_{(t)}} \tag{6.4c}$$

When training a neural network, the gradients related to the parameters are used in the optimization step, to try to reach a local optimum, which is also the case for RNNs. This means that the gradients calculated using Equation 6.3 and Equation 6.4, can be used to optimize the RNN.

6.1.2 Vanishing/Exploding Gradient Problem

In deep neural networks it is possible that the problem of vanishing/exploding gradients occurs [23], which is when the gradient becomes extremely small or extremely big. This problem can occur when the output of a neural network is based on a parameter that is shared through numerous layers.

It is more clear what the problem is if Equation 6.4a is expanded according to the chain rule. If $\tau = 3$, the expansion can be seen in Equation 6.5.

$$\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{W}} = \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{o}} \cdot \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^{(3)}} \cdot \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \cdot \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \cdot \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{W}_{(1)}} + \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{o}} \cdot \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^{(3)}} \cdot \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \cdot \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{W}_{(2)}} + \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{o}} \cdot \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{h}^{(3)}} \cdot \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{W}_{(3)}} \tag{6.5}$$

From Equation 6.5 it can be seen that $\frac{\partial \boldsymbol{C}}{\boldsymbol{W}}$ is a product of many gradients, and if τ increases, the number of gradients also increases. If the gradients are below 1, the product diminishes

towards 0, which can result in a vanishing gradient. Contrary to this, if the gradients are above 1 it can result in an exploding gradient. Vanishing gradient is the most common case.

If the vanishing gradient occurs, early hidden states in an RNN end up having a little (near to zero) effect on how the weights are updated using BPTT. Therefore, the RNN is less likely to learn dependencies on information early in the input sequence.

6.2 Long Short-Term Memory [2]

This section is based on information from [22] and [24]. Long short-term memory (LSTM) is a variation of RNN. An LSTM additionally makes use of a previous cell state, forget-, input- and output-gates encapsulated in a cell structure, seen in Figure 6.4. The gates are used to restrict certain information from being propagated to the following states while letting other information through.

In Figure 6.4 the forget-, input-, and output gates are denoted as σ_f, σ_i, and σ_o respectively. The gates are used to proportionate how much information is propagated from the previous cell state, the input, and the previous hidden state, to the following hidden state and output. The σ_t activation functions are usually *tanh* functions, but can be interchanged with other activation functions [25]. Some parameters seen in Section 6.1 are also applied inside the gates of an LSTM, namely, the weight matrices $\boldsymbol{U}$ and $\boldsymbol{W}$, and the bias vector $\boldsymbol{a}$. These weights and biases are not shown in Figure 6.4, but the application of these can be seen in Equation 6.6 - 6.8. These parameters exist for each of the gates. For example, $\boldsymbol{W}^{(f)}$, $\boldsymbol{U}^{(f)}$ and $\boldsymbol{a}^{(f)}$ denote the weight matrices, and the bias used by the forget gate. Aside from these, there is an additional set of parameters $\boldsymbol{U}$, $\boldsymbol{W}$, and $\boldsymbol{a}$ for the cell.

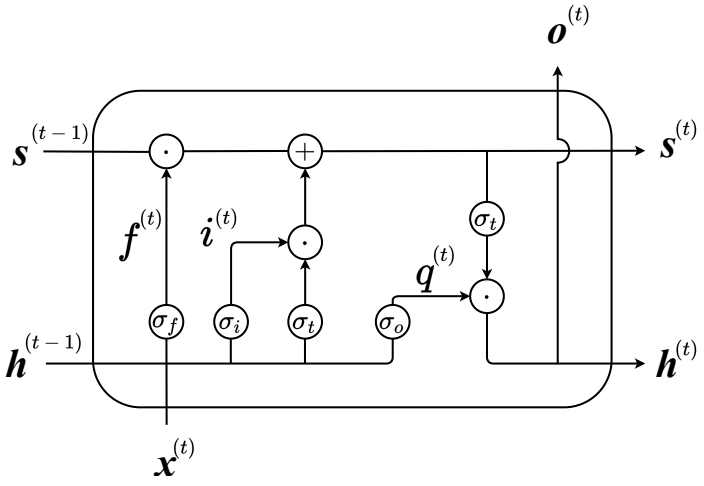


Figure 6.4: An LSTM cell with forget, input and output gates denoted as σ_f, σ_i, and σ_o respectively.

[2]This section is a slightly modified version of section 4.2.1 from our previous semester report [2].

Forward propagation in an LSTM is computed using Equation 6.6 - 6.8 which are related to Figure 6.4.

$$f^{(t)} = \sigma_f\Big(\boldsymbol{a}^{(f)} + \boldsymbol{U}^{(f)} \cdot \boldsymbol{x}^{(t)} + \boldsymbol{W}^{(f)} \cdot \boldsymbol{h}^{(t-1)}\Big) \tag{6.6a}$$

$$i^{(t)} = \sigma_i\Big(\boldsymbol{a}^{(i)} + \boldsymbol{U}^{(i)} \cdot \boldsymbol{x}^{(t)} + \boldsymbol{W}^{(i)} \cdot \boldsymbol{h}^{(t-1)}\Big) \tag{6.6b}$$

Equation 6.7a shows how the σ_f and σ_i gates together with the previous cell state $s^{(t-1)}$ are used to compute the new cell state $s^{(t)}$. Equation 6.7b shows how the output of σ_o is calculated, denoted $q^{(t)}$.

$$s^{(t)} = f^{(t)}s^{(t-1)} + i^{(t)}\sigma_t\Big(\boldsymbol{a} + \boldsymbol{U} \cdot \boldsymbol{x}^{(t)} + \boldsymbol{W} \cdot \boldsymbol{h}^{(t-1)}\Big) \tag{6.7a}$$

$$q^{(t)} = \sigma_o\Big(\boldsymbol{a}^{(o)} + \boldsymbol{U}^{(o)} \cdot \boldsymbol{x}^{(t)} + \boldsymbol{W}^{(o)} \cdot \boldsymbol{h}^{(t-1)}\Big) \tag{6.7b}$$

The output of the cell and the information propagated to the following state are calculated using Equation 6.8.

$$\boldsymbol{o}^{(t)} = \boldsymbol{h}^{(t)} = \sigma_t\Big(s^{(t)}\Big)q^{(t)} \tag{6.8}$$

As for finding gradients in RNNs, gradients for LSTMs are also calculated using BPTT. So, if an LSTM is unfolded as with an RNN, the gradients are also calculated according to back-propagation, as we describe in Appendix C.1. In LSTMs, the cell state propagates two outputs to the following cell state, one calculated using Equation 6.7a, denoted $\boldsymbol{s}^{(t)}$, and the other calculated using Equation 6.8, denoted $\boldsymbol{h}^{(t)}$. From the equations and from Figure 6.4 we see that information is propagated to $s^{(t)}$ from $s^{(t-1)}$ both directly, through the forget gate, and as a product of the input gate and the output of activation function, σ_t. This means that when backpropagating gradients from $s^{(t)}$ to $s^{(t-1)}$, the gradient is calculated as the sum of four partial derivatives. If these gradients sum to around 1, the gradient $\frac{\partial s^{(t)}}{\partial s^{(t-1)}}$ does not necessarily have to vanish towards 0. As the gradient does not necessarily diminish towards 0, it is possible that an LSTM can learn to use information from earlier timesteps, contrary to an RNN. This is also evident from the case where the forget gate outputs 1, as information, thereby, is propagated unrestricted from $s^{(t-1)}$ to $s^{(t)}$.

6.3 Gated Recurrent Unit

This section is based on information from [22] and [26]. Gated recurrent units (GRU) is a variation of RNN. The motivation of creating GRU was to reduce the number of parameters compared to LSTM without compromising performance. As described in Section 6.2, the LSTM has three gates, forget-, input- and output-gates, and each of these has a unique weight matrix and bias vector. GRU reduces the number of gates to two gates, the reset gate and the update gate. The cell structure of a GRU cell is shown in Figure 6.5.

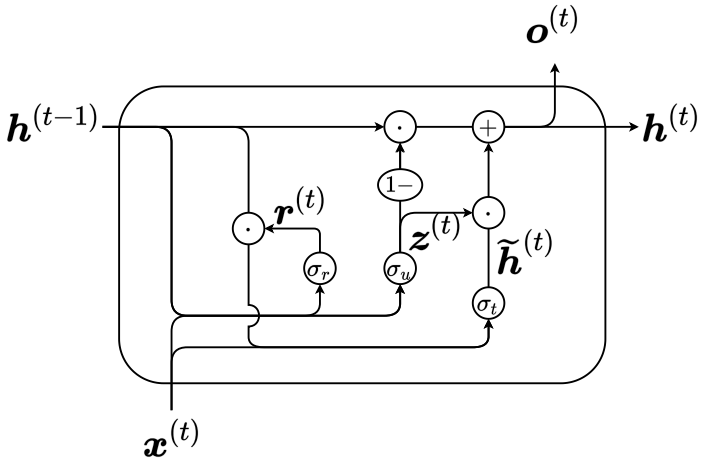


Figure 6.5: Cell of a standard GRU.

While the reset gate is similar to the forget gate in an LSTM, the update gate functions as the input- and output gate of the LSTM simultaneously, which reduces the number of parameters. Both gates (σ_r and σ_u) use the sigmoid activation function. The output of the reset gate is calculated with:

$$\boldsymbol{r}^{(t)} = \sigma_r \left(\boldsymbol{W}_r \cdot \boldsymbol{x}^{(t)} + \boldsymbol{U}_r \cdot \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_r \right) \tag{6.9}$$

which uses weight matrices for the input and the hidden state ($\boldsymbol{W}_r$ and $\boldsymbol{U}_r$) and a bias vector ($\boldsymbol{b}_r$), to determine the amount of information from the input and the hidden state, that is propagated. The reset gate uses the input and previous hidden state and restricts the information that is propagated to the subsequent hidden states, i.e. how much of the past information to forget.

$\boldsymbol{r}^{(t)}$ is used together with the input and previous hidden state, to calculate a candidate activation, which represents a candidate hidden state, based on the current input and the previous hidden state with some of its information forgotten, in:

$$\widetilde{\boldsymbol{h}}^{(t)} = \sigma_t \left(\boldsymbol{W} \cdot \boldsymbol{x}^{(t)} + \boldsymbol{U} \left(\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)} \right) + \boldsymbol{b} \right) \tag{6.10}$$

where σ_t uses the *tanh* activation function. As each value in $\boldsymbol{r}^{(t)}$ gets closer to 0, the more information in $\boldsymbol{h}^{(t-1)}$ is forgotten.

The update gate is likewise used to proportionate how much of the previous information is propagated forward, and is computed similarly to the reset gate:

$$\boldsymbol{z}^{(t)} = \sigma_u \left(\boldsymbol{W}_z \cdot \boldsymbol{x}^{(t)} + \boldsymbol{U}_z \cdot \boldsymbol{h}^{(t-1)} + \boldsymbol{b}_z \right) \tag{6.11}$$

The output of the update gate is applied to both the candidate activation $\widetilde{\boldsymbol{h}}^{(t)}$, and previous hidden state $\boldsymbol{h}^{(t-1)}$, to calculate the current hidden state in:

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{z}^{(t)}) \odot \boldsymbol{h}^{(t-1)} + \boldsymbol{z}^{(t)} \odot \widetilde{\boldsymbol{h}}^{(t)} \tag{6.12}$$

This way, $\boldsymbol{z}^{(t)}$ is used to control how much of the previous state $\boldsymbol{h}^{(t-1)}$ and the candidate state $\widetilde{\boldsymbol{h}}^{(t)}$ is propagated forward. By adding the results together, the new hidden state $\boldsymbol{h}^{(t)}$

has forgotten information about the values determined by the reset gate and highlighted the values determined by the update gate.

Finding the gradients for a GRU is calculated using BPTT like for an RNN, which we have described in Section 6.1.1. From Equation 6.9-6.12 and Figure 6.5, we see that information is backpropagated directly from $\boldsymbol{h}^{(t)}$ to $\boldsymbol{h}^{(t-1)}$, through the reset gate, and through the update gate. As there are three paths from $\boldsymbol{h}^{(t)}$ to $\boldsymbol{h}^{(t-1)}$ when backpropagating the gradients, the gradients are calculated as the sum of the partial derivatives for each path. Similar to the LSTM, GRU also mitigates the vanishing gradient problem with its gates. The case where the update gate σ_u outputs 0 exemplifies this well, as all information is propagated from $\boldsymbol{h}^{(t-1)}$ to $\boldsymbol{h}^{(t)}$.

6.4 GRU-D

In this section, we describe the deep learning model, GRU-D. The model is proposed by Che et al. [20] and is based on GRU. GRU-D uses two data representations to capture missingness patterns, called masking and time interval. The model architecture uses these representations in order to utilize the missingness patterns to achieve more accurate predictions.
Before describing the GRU-D model, we describe the missingness representations.

6.4.1 Missingness Representations

There are two missingness representations, the masking representation, and time interval representation.

The masking representation $\boldsymbol{M}$ for $\boldsymbol{X}$ is boolean and represents whether the value is missing or not. The value for the mask $\boldsymbol{M}$ for feature d at timestep t is given by:

$$m_d^{(t)} = \begin{cases} 1 & if\ x_d^{(t)}\ has\ a\ value \\ 0 & otherwise \end{cases} \tag{6.13}$$

The time interval representation $\boldsymbol{\Delta}$ for $\boldsymbol{X}$ represents the number of hours since a value was last observed for each feature. As the measurements in CinC2019 and PCT are aggregated over an hour, the value for the time interval $\boldsymbol{\Delta}$ for feature d at timestep t is given by:

$$\delta_d^{(t)} = \begin{cases} 1 + \delta_d^{(t-1)} & t > 1, m_d^{(t-1)} = 0 \\ 1 & t > 1, m_d^{(t-1)} = 1 \\ 0 & t = 1 \end{cases} \tag{6.14}$$

An example of the missingness representations is shown in Figure 6.6.

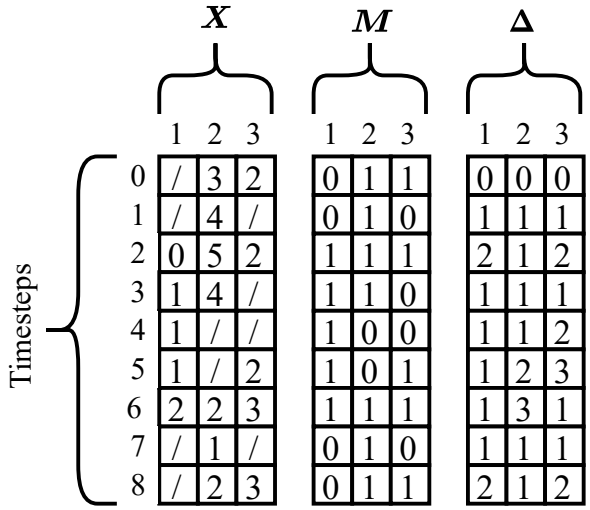


Figure 6.6: Example of representations of missing data.

Che et al. [20] propose concatenating these missingness representations with the time series data as:

$$\boldsymbol{x}^{(t)} \leftarrow \left[\boldsymbol{x}^{(t)} \circ \boldsymbol{m}^{(t)} \circ \boldsymbol{\delta}^{(t)}\right], \tag{6.15}$$

and giving it to a standard GRU, where $\boldsymbol{x}^{(t)}$ is imputed with zeros. They show this gives better results when used as input to a standard GRU compared to only giving the time series data. They also suggest $\boldsymbol{x}^{(t)}$ can be imputed with other methods, such as mean or LOCF imputation. However, the GRU-D model they propose utilizes these representations in the model to get even better performance.

6.4.2 The GRU-D model

GRU-Decay (GRU-D) is a modification of a standard GRU, which uses the missingness representations in a decay term for the GRU network's hidden states and inputs. Besides the addition of the decay, GRU-D is a standard GRU, which we described in Section 6.3.

Figure 6.7 shows a GRU-D cell, where the additions to the standard GRU cell are encapsulated in the green and blue sections, which represent the input decay and hidden state decay respectively.

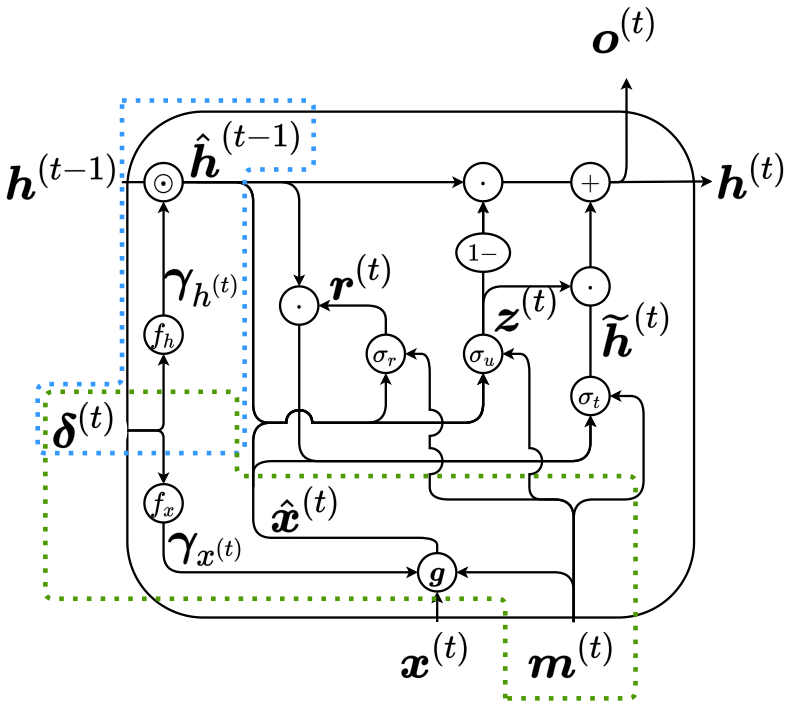


Figure 6.7: Cell of the GRU-D model. The modifications from a standard GRU are encapsulated in the green and blue sections. To make the figure simpler, g represents Equation 6.18, which is used to calculate $\hat{x}^{(t)}$. f_h and f_x refers to Equation 6.17

GRU-D uses the decay term to reduce the impact of older information, due to the assumption that recent observations are more relevant than earlier observations. The decay term gradually reduces the impact of older observations, by using the missingness representations, where the longer time since a value has been observed, the less impact the previous information has. The decay term $\gamma^{(t)}$ is calculated based on $\delta^{(t)}$ with:

$$\gamma^{(t)} = f(W_\gamma \delta^{(t)} + b_\gamma) \tag{6.16}$$

where f is a monotonically decreasing function bounded between 0 and 1 and $W\gamma$ and $b_\gamma^{(t)}$ are the weights and biases for $\gamma^{(t)}$. $\gamma^{(t)}$ uses different weights and biases for the input and hidden state decay, which is subscripted with x and h respectively. As with all the other parameters of GRU-D, the weights and biases for the decay term are learned during training of the model. GRU-D used the exponentiated negative rectifier for f:

$$f(\cdot) = exp\{-max(0, \cdot)\} \tag{6.17}$$

The decay is applied to the input, shown in the green section of Figure 6.7, with:

$$\hat{x}_d^{(t)} = m_d^{(t)} \cdot x_d^{(t)} + \left(1 - m_d^{(t)}\right) \cdot \left(\gamma_{x_d^{(t)}} \cdot x_d^{(t')} + (1 - \gamma_{x_d^{(t)}}) \cdot \widetilde{x}_d\right) \tag{6.18}$$

where $x_d^{(t')}$ is the last observed value for feature d before the current timestep and $\widetilde{x}_d$ is the mean of the feature d. The masking representation is used to indicate whether a value for

feature d is observed at timestep t. If a value is observed, then the current value is used as input. If a value is not observed, the decay term is used to calculate a weighted average between the last observed value $x_d^{(t')}$ and the mean of the feature d $\widetilde{x}_d$, which is used as the input.

As the extracted features in the previous hidden state have information from earlier, GRU-D also uses decay on the previous hidden state, shown in the green section of Figure 6.7, with:

$$\hat{\boldsymbol{h}}^{(t-1)} = \gamma_{h^{(t)}} \odot \boldsymbol{h}^{(t-1)} \tag{6.19}$$

The formulas for a standard GRU (Equation 6.9 to 6.12) is updated with the decay, which gives the formulas of GRU-D (Equation 6.20 to 6.23):

$$\boldsymbol{r}^{(t)} = \sigma_r \left(\boldsymbol{W}_r \cdot \hat{\boldsymbol{x}}^{(t)} + \boldsymbol{U}_r \cdot \hat{\boldsymbol{h}}^{(t-1)} + \boldsymbol{V}_r \cdot \boldsymbol{m}^{(t)} + \boldsymbol{b}_r \right) \tag{6.20}$$

$$\boldsymbol{z}^{(t)} = \sigma_u \left(\boldsymbol{W}_z \cdot \hat{\boldsymbol{x}}^{(t)} + \boldsymbol{U}_z \cdot \hat{\boldsymbol{h}}^{(t-1)} + \boldsymbol{V}_z \cdot \boldsymbol{m}^{(t)} + \boldsymbol{b}_z \right) \tag{6.21}$$

$$\widetilde{\boldsymbol{h}}^{(t)} = \sigma_t \left(\boldsymbol{W} \cdot \hat{\boldsymbol{x}}^{(t)} + \boldsymbol{U} \left(\boldsymbol{r}^{(t)} \odot \hat{\boldsymbol{h}}^{(t-1)} \right) + \boldsymbol{V} \cdot \boldsymbol{m}^{(t)} + \boldsymbol{b} \right) \tag{6.22}$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{z}^{(t)}) \odot \hat{\boldsymbol{h}}^{(t-1)} + \boldsymbol{z}^{(t)} \odot \widetilde{\boldsymbol{h}}^{(t)} \tag{6.23}$$

The differences are that $\boldsymbol{x}^{(t)}$ have been replaced with $\hat{\boldsymbol{x}}^{(t)}$ from Equation 6.18, $\boldsymbol{h}^{(t-1)}$ have been replaced with $\hat{\boldsymbol{h}}^{(t-1)}$ from Equation 6.19, and the masking vector $\boldsymbol{m}^{(t)}$ and its weights ($\boldsymbol{V}_z$, $\boldsymbol{V}_r$, $\boldsymbol{V}$) have been added.

6.5 Bidirectional Recurrent Imputation for Time Series

In this section, we describe the theory of the imputation and classification method, BRITS [21]. BRITS is proposed as an effective multi-task learning algorithm for imputation of missing time series data, and classification. To achieve this, BRITS uses several concepts which we cover in this section.

6.5.1 General Architecture

BRITS adapts RNNs as its internal architecture and therefore extends some of the RNN concepts we described in Section 6.1. Any recurrent model can be used in the internal architecture. However, the description of BRITS is based on a regular RNN, as it is the simplest RNN, which makes it easier to explain the intuition of BRITS. BRITS uses the recurrent architecture to learn how to impute the missing values in the time series data, by estimating subsequent input values based on previous observations and data representations indicating missing values and their patterns.

We start by providing a simpler overview of BRITS based on Figure 6.8, which we extended from the RNN cell from Figure 6.2. Note that this is a simplified representation of a BRITS cell, and we describe the complete representation in Section 6.5.3. Besides the RNN components,

the BRITS cell is split into three parts, visualized by the colored dotted sections. The green section estimates the values of the subsequent input. The red section handles the missing values in the input and creates an alternative representation. The blue section handles the decay of previous states, based on how many timesteps it has been since a value has been observed.

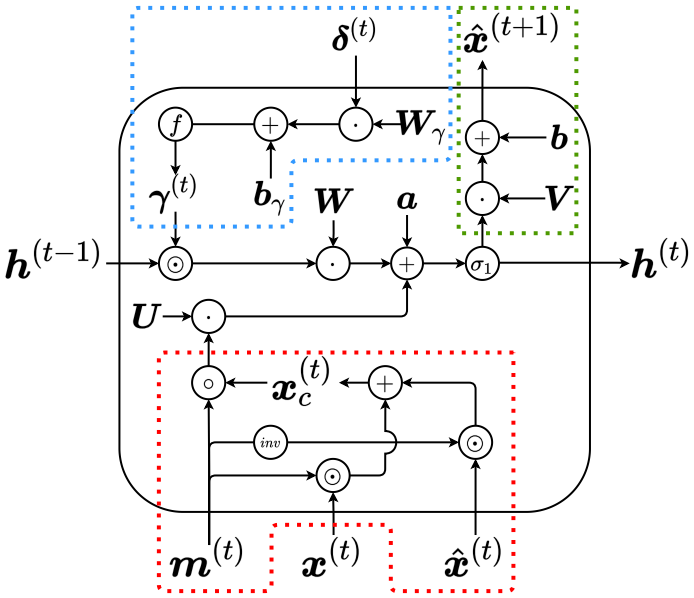


Figure 6.8: Representation of a simplified state in a BRITS cell. The additions are encapsulated in the color coded sections. f refers to Equation 6.17.

We now provide a more in-depth explanation of how exactly these three components work.

Estimating input

In order to impute the missing values for the subsequent timestep $t+1$, BRITS attempts to estimate $\boldsymbol{x}^{(t+1)}$. This estimation ($\hat{\boldsymbol{x}}^{(t+1)}$) is calculated in the green section in Figure 6.8 from the hidden state with:

$$\hat{\boldsymbol{x}}^{(t+1)} = \boldsymbol{V} \cdot \boldsymbol{h}^{(t)} + \boldsymbol{b} \tag{6.24}$$

Imputing Missing Input

$\hat{\boldsymbol{x}}^{(t)}$ is used to impute the input with missing values. This is done by using the masking representation, described in Section 6.4, to indicate which of the input values are missing. BRITS imputes the input by replacing the missing values in the input $\boldsymbol{x}^{(t)}$ with the corresponding estimated values in $\hat{\boldsymbol{x}}^{(t)}$, which we refer to as a complement variable $\boldsymbol{x}_c^{(t)}$.

$\boldsymbol{x}_c^{(t)}$ consists of values from either the estimated value ($\hat{\boldsymbol{x}}^{(t)}$) or the input value ($\boldsymbol{x}^{(t)}$). $\boldsymbol{x}_c^{(t)}$ is calculated as:

$$\boldsymbol{x}_c^{(t)} = \boldsymbol{m}^{(t)} \odot \boldsymbol{x}^{(t)} + (1 - \boldsymbol{m}^{(t)}) \odot \hat{\boldsymbol{x}}^{(t)} \tag{6.25}$$

which uses the masking representation, $\boldsymbol{m}^{(t)}$, to specify whether a value is missing for timestep t. This is done with elementwise multiplication, as $\boldsymbol{m}_d^{(t)}$ is 0, when the value is missing and 1 when it is not. Additionally, the complement variable is concatenated with the masking vector, which is then used as input in the RNN cell. All this is done in the red section of Figure 6.8. An example of imputing the missing values with Equation 6.25 can be seen in Figure 6.9.

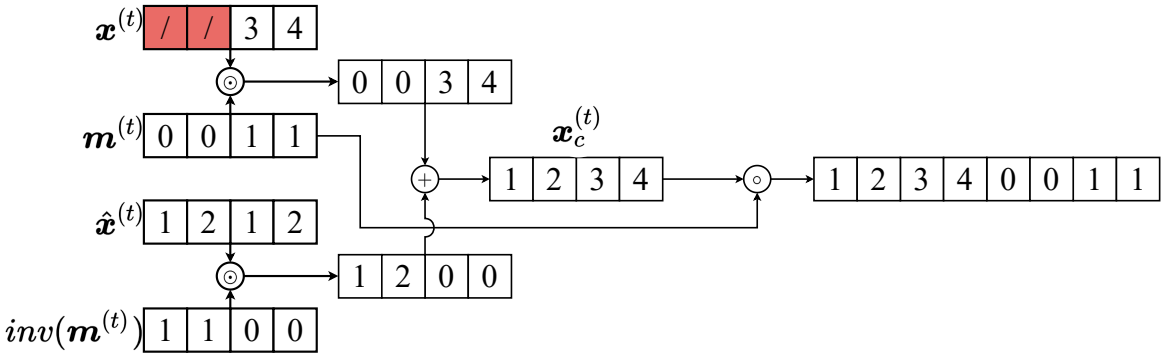


Figure 6.9: Example of the imputation section done in BRITS.

Temporal Decay

The purpose of the temporal decay term is to take the time since an input value has been observed into account, and use this to reduce the magnitude of the values in $\boldsymbol{h}^{(t-1)}$. The longer time since a value has been observed, the more $\boldsymbol{h}^{(t-1)}$ is decayed. The temporal decay term is calculated the same way as the decay term used in GRU-D with Equation 6.16 and Equation 6.17. The temporal decay term ($\gamma^{(t)}$) is determined within the blue dotted marking in Figure 6.8.

Hidden state

To incorporate the estimated input and the temporal decay in the calculation of $h^{(t)}$, the equation for a regular RNN hidden state (Equation 6.1a) is extended to:

$$\boldsymbol{h}^{(t)} = \sigma_1(\boldsymbol{W}[h^{(t-1)} \odot \gamma^{(t)}] + \boldsymbol{U}[x_c^{(t)} \circ m^{(t)}] + a) \tag{6.26}$$

First, it extends the calculation by decaying the previous hidden state $\boldsymbol{h}^{(t-1)}$ with $\gamma^{(t)}$. Instead of using $\boldsymbol{x}^{(t)}$ as input (as in Equation 6.1a), Equation 6.26 concatenates the complement variable $\boldsymbol{x}_c^{(t)}$ with the masking representation $\boldsymbol{m}^{(t)}$ and uses that as input.

6.5.2 Calculating loss

In this section, we identify how the losses in BRITS are calculated. BRITS uses three types of losses. The first loss is the error of how well $\hat{\boldsymbol{x}}^{(t)}$ estimated the missing values. The second loss is the error of the predicted class label compared to the true class label. The third loss is

the error of the consistency in the bidirectional estimation of input values. In the following sections, we describe these losses in more detail.

Estimation loss

BRITS learns to better estimate the input values, based on the estimation loss:

$$\ell^{(t)} = \langle m^{(t)}, \mathcal{L}_e(x^{(t)}, \hat{x}^{(t)}) \rangle \tag{6.27}$$

which uses an estimation loss function $\mathcal{L}_e$ between the estimated value and the input value, for each time step, where $m^{(t)}$ is used to indicate whether a value is missing, and if so, the estimated loss for that value does not contribute to the total loss. When a value in $\boldsymbol{x}^{(t)}$ is missing, $\hat{\boldsymbol{x}}^{(t)}$ is still used to estimate $\hat{\boldsymbol{x}}^{(t+1)}$. This is a problem, because the loss for $\hat{\boldsymbol{x}}^{(t)}$ cannot be computed, it is unknown how well of an estimation it is. To better comprehend this problem, we show an example in Figure 6.10 that traverses the recurrent states of BRITS, with some missing values.

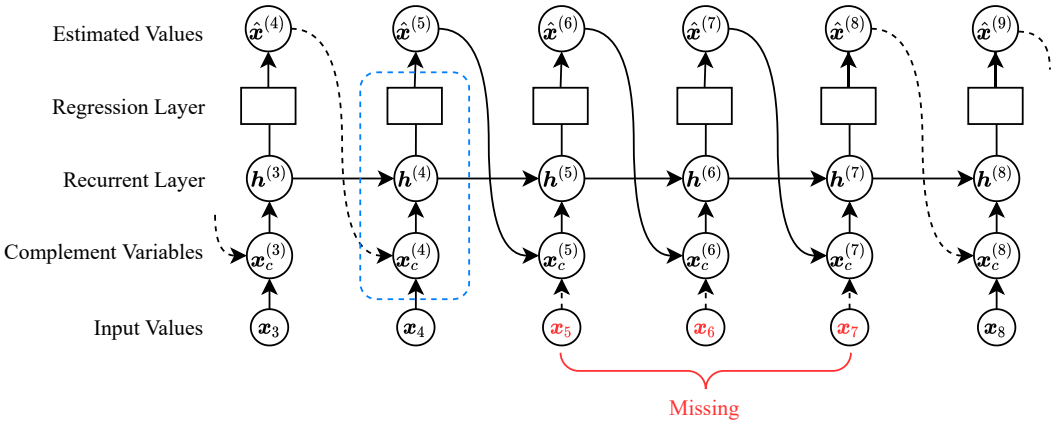


Figure 6.10: An example of the information flow in the BRITS architecture, with three missing values input. The content in the blue dotted square is an example of a simplified illustration of a single BRITS cell from Figure 6.8. Therefore, all computations in the BRITS cell where timestep $t = 4$ are done within the blue dotted square. The regression layer refers to Equation 6.24.

In this example, input values $\boldsymbol{x}^{(5)}, \boldsymbol{x}^{(6)}, \boldsymbol{x}^{(7)}$ are missing. To simplify this example, we assume that all values in these vectors are missing. This means that we cannot calculate the estimation loss $\ell^{(5)}$ immediately, and have to delay the calculation until the next input value is observed, which is $\boldsymbol{x}^{(8)}$. Therefore, the estimation loss is delayed for $t = 5, 6, 7$, which makes the estimated value of $\hat{\boldsymbol{x}}^{(8)}$ depend on the estimated values of $\hat{\boldsymbol{x}}^{(5)}$ to $\hat{\boldsymbol{x}}^{(7)}$. This results in a delayed loss, which first is accounted for in $\ell^{(8)}$.

Classification loss

If the output from each timestep is used for the prediction, then the prediction label $\hat{y}$ is calculated with Equation 6.28,

$$\hat{y} = f_{out}(\sum_{t=1}^{\tau} \alpha^{(t)} \boldsymbol{h}^{(t)}) \tag{6.28}$$

where f_{out} is a fully connected layer, and $\alpha^{(t)}$ is a weight, which for example can be $1/\tau$ for all timesteps, which results in the mean. BRITS is then optimized by minimizing the accumulated loss:

$$\frac{1}{\tau} \sum_{t=1}^{\tau} \ell^{(t)} + \mathcal{L}_{out}(y, \hat{y}) \tag{6.29}$$

where $\mathcal{L}_{out}$ is the output loss between the predicted class label $\hat{y}$ and the ground truth class label y. As exemplified with Figure 6.10, BRITS backpropagates the gradients in the opposite direction of the solid lines. Because $\boldsymbol{x}^{(5)}$ to $\boldsymbol{x}^{(7)}$ were missing, backpropagation through their estimated values is done instead of to their corresponding inputs. This means that the delayed, possibly accumulated, loss $\ell^{(8)}$ is backpropagated through several timesteps. The loss may therefore be less meaningful for timesteps farther from the deriving timestep.

Consistency loss

BRITS attempts to reduce the problem caused by the delayed loss, by considering the time series data with a bidirectional recurrent approach. This means that the same approach is taken but in the reverse direction of the time series data. This results in the forward estimation- and loss sequence ($\{\hat{\boldsymbol{x}}^{(1)}, \ldots, \hat{\boldsymbol{x}}^{(\tau)}\}$ and $\{\ell^{(1)}, \ldots, \ell^{(\tau)}\}$), and backward estimation and loss sequence ($\{\hat{\boldsymbol{x}}^{(1)\prime}, \ldots, \hat{\boldsymbol{x}}^{(\tau)\prime}\}$ and $\{\ell^{(1)\prime}, \ldots, \ell^{(\tau)\prime}\}$).

To enforce consistency between the forward- and backward estimations, a consistency loss is used:

$$\ell_{cons}^{(t)} = Discrepancy(\hat{\boldsymbol{x}}^{(t)}, \hat{\boldsymbol{x}}^{(t)\prime}) \tag{6.30}$$

where a discrepancy function, such as mean squared error, is used to determine the estimation similarity. As Equation 6.29 is the loss function used to optimize BRITS, it is extended to account for the backward loss and the consistency loss:

$$\frac{1}{\tau} \sum_{t=1}^{\tau} (\ell^{(t)} + \ell^{(t)\prime} + \ell_{cons}^{(t)}) + \mathcal{L}_{out}(y, \hat{y}) \tag{6.31}$$

where $\hat{y}$ becomes the mean of the predicted class label in the forward and backward direction. The final estimation that is used as the imputed value for each timestep t, is the mean of $\hat{\boldsymbol{x}}^{(t)}$ and $\hat{\boldsymbol{x}}^{(t)\prime}$.

6.5.3 Correlated features

So far we have described BRITS with the assumption that the estimation of a specific value $\hat{\boldsymbol{x}}_d^{(t)}$ is not influenced by any other features in the same timestep. A visualization of this can be seen in Figure 6.11. Here, the estimation in the blue square is calculated based on previous values indicated by green squares. The red squares indicate values of other features for the

same timestep, which are not used in the estimation of the blue value, as they are assumed to be uncorrelated.

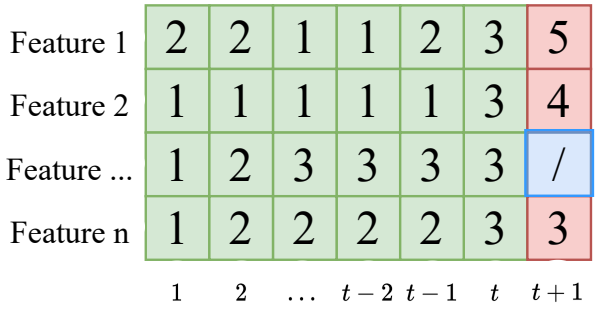


Figure 6.11: Example of how features so far have been estimated.

The reason for this assumption was to simplify the concepts of BRITS, by introducing the fundamental concepts, which is then extended to capture the entirety of BRITS. In this section, we relax this assumption and describe how it extends the concepts. To account for the correlation between features in the same timestep, a new estimation vector is used:

$$\hat{\boldsymbol{z}}^{(t)} = \boldsymbol{W}_z \cdot \boldsymbol{x}_c^{(t)} + \boldsymbol{b}_z \tag{6.32}$$

where $\boldsymbol{x}_c^{(t)}$ is the complement variable, from Equation 6.25, and $\boldsymbol{W}_z$, $\boldsymbol{b}_z$ are feature weight and bias parameters. Equation 6.33b is used to create a combined estimate between this new estimate $\hat{\boldsymbol{z}}^{(t)}$ and the previous estimate $\hat{\boldsymbol{x}}^{(t)}$, where the weight $\beta^{(t)}$ from Equation 6.33a is used to weigh each of the estimates.

$$\boldsymbol{\beta}^{(t)} = \sigma(\boldsymbol{W}_\beta[\gamma^{(t)} \circ \boldsymbol{m}^{(t)}] + \boldsymbol{b}_\beta) \tag{6.33a}$$

$$\hat{\boldsymbol{c}}^{(t)} = \boldsymbol{\beta}^{(t)} \odot \hat{\boldsymbol{z}}^{(t)} + (1 - \boldsymbol{\beta}^{(t)}) \odot \hat{\boldsymbol{x}}^{(t)} \tag{6.33b}$$

The weight vector $\boldsymbol{\beta}^{(t)} \in [0,1]^D$ is learned by using the masking vector $\boldsymbol{m}^{(t)}$, and the temporal decay term $\gamma^{(t)}$. This way, it learns how to weigh the estimations based on previous values ($\hat{\boldsymbol{x}}^{(t)}$), and estimations based on features from the same timestep ($\hat{\boldsymbol{z}}^{(t)}$), by considering whether the values are missing, and the time since previous observation.

As the estimated value $\hat{\boldsymbol{x}}^{(t)}$ were used in Equation 6.25 to calculate the complement variable, the combined estimate $\hat{\boldsymbol{c}}^{(t)}$ is used to calculate a combined complement variable $\boldsymbol{c}_c^{(t)}$ with the same approach, in Equation 6.34:

$$\boldsymbol{c}_c^{(t)} = \boldsymbol{m}^t \odot \boldsymbol{x}^{(t)} + (1 - \boldsymbol{m}^{(t)}) \odot \hat{\boldsymbol{c}}^{(t)} \tag{6.34}$$

This new variable functions as a replacement of $\hat{\boldsymbol{x}}^{(t)}$, as it is already derived by it. Notice therefore that Equation 6.34 is simply an extension of Equation 6.25, which has replaced $\hat{\boldsymbol{x}}^{(t)}$ with $\hat{\boldsymbol{c}}^{(t)}$. This also applies to the hidden state $h^{(t)}$ in Equation 6.26, such that the combined complement variable $\boldsymbol{c}_c^{(t)}$ replaces $\boldsymbol{x}_c^{(t)}$ in Equation 6.35:

$$\boldsymbol{h}^{(t)} = \sigma(\boldsymbol{W}_h[\boldsymbol{h}^{(t-1)} \odot \gamma^{(t)}] + \boldsymbol{U}_h[\boldsymbol{c}_c^{(t)} \circ \boldsymbol{m}^{(t)}] + \boldsymbol{a}) \tag{6.35}$$

These changes leads to the following extension from the loss function $\ell^{(t)}$ in Equation 6.27, such that it also considers the estimation that were based on other features, and the combined estimation:

$$\ell^{(t)} = \mathcal{L}_e(\boldsymbol{x}^{(t)}, \hat{\boldsymbol{x}}^{(t)}) + \mathcal{L}_e(\boldsymbol{x}^{(t)}, \hat{\boldsymbol{z}}^{(t)}) + \mathcal{L}_e(\boldsymbol{x}^{(t)}, \hat{\boldsymbol{c}}^{(t)}) \tag{6.36}$$

where BRITS uses mean squared error as the loss function for $\mathcal{L}_e$ [21]. This loss is calculated for all timesteps, both in the forward and backward direction of the time series data.

Chapter 7

Model Designs

In this chapter, we describe the general architecture of the models we want to build on top of in our experiments. In Chapter 5 we chose to build on top of GRU-D and BRITS. To see how the explicit handling of missing values in these models compares to a model that gets imputed data, we compare them to the two best performing models from our last project, LSTM and TCN [2] and use these as baselines. As models can not handle the missing values and therefore need imputed data, we start by describing the imputation method we use for these baselines.

The figures in this chapter are based on an observation window of 24 hours and the hyperparameters we find in our preliminary hyperparameter tuning (described in Section E.5).

7.1 Imputation for Baselines

One of the imputation methods we used previous semester was mean/mode imputation, which is a commonly used baseline [27]. Mean/mode imputation is an imputation method, where the values are imputed with either the mean or mode (most common) value of all observed values for that feature. The mean value is used for numeric features, e.g. heart rate, and mode is used for class features, e.g. gender. As this imputation method introduces a bias [28], we do not consider it further.

Instead, we consider imputation methods with relevant characteristics for CinC2019 and PCT, i.e. they operate on time series data with a high missing rate. Forward Imputation, also known as Last Observation Carried Forward (LOCF) imputation, is an imputation method used on time series data [29], which sometimes is used as a baseline [20]. LOCF imputes a missing value with the last observed value, hence its name, as previously observed values are carried forward in the time series. Due to its simplicity and that it is made for time series data, we use LOCF as a baseline method for handling missing values.

7.2 TCN

The TCN is one of the models that we chose to use as a baseline and it uses the same architecture as the TCN from our previous semester [2]. One of the reasons why we want to experiment with TCN, is that it is not recurrent. This is preferable for Enversion, as they use a tool for explaining which features contributed most to the prediction, which does not work with RNNs, but it does with TCNs.

The architecture of TCN is shown in Figure 7.1.

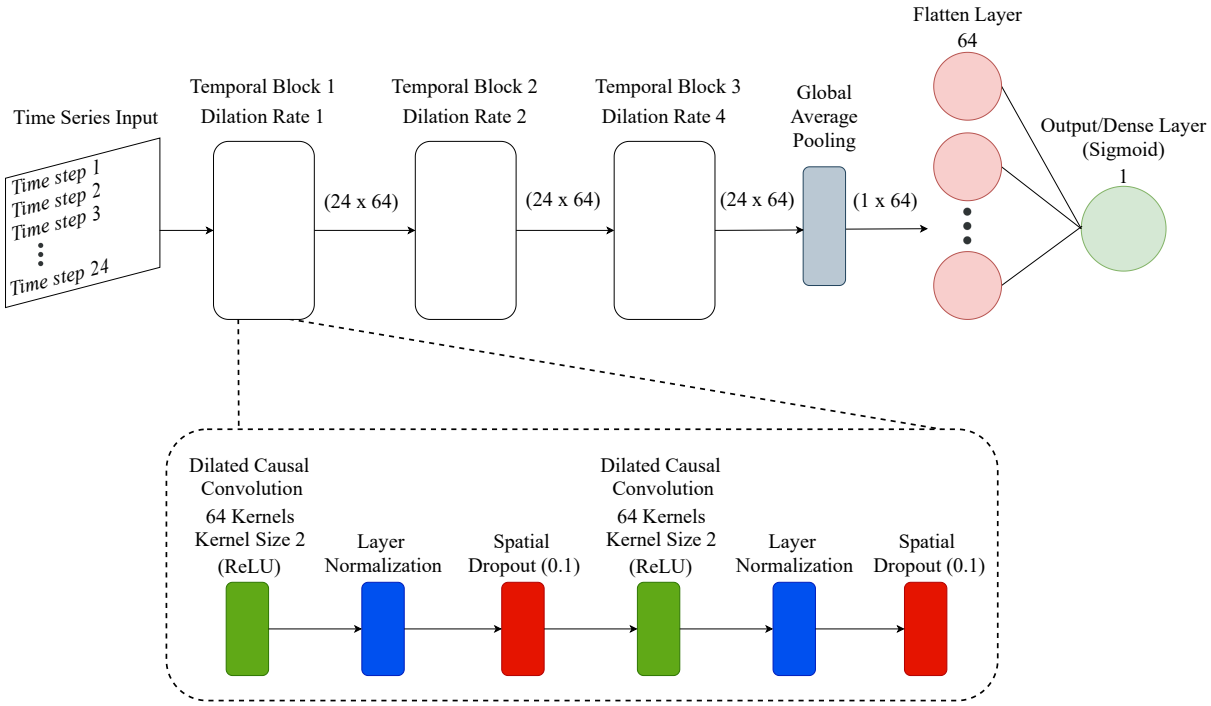


Figure 7.1: General architecture of TCN.[1]

This description of the TCN architecture uses theoretical terms that are described in Appendix C.2. The core part of the TCN is its temporal blocks, which contains two sequences of a dilated casual convolutional layer, followed by layer normalization and spatial dropout. The number of temporal blocks in a TCN can vary, with double the dilation rate for each consecutive temporal block. The temporal blocks are followed by a global average pooling, a flatten layer, and an output layer with a sigmoid function, resulting in the model's probabilistic prediction for whether the patient develops sepsis.

7.3 LSTM

LSTM is the second model that we chose to use as a baseline, which follows the same architecture as in our previous semester [2], shown in Figure 7.2.

[1]The TCN architecture is a slightly modified version of Figure 7.3 from our previous semester report [2].

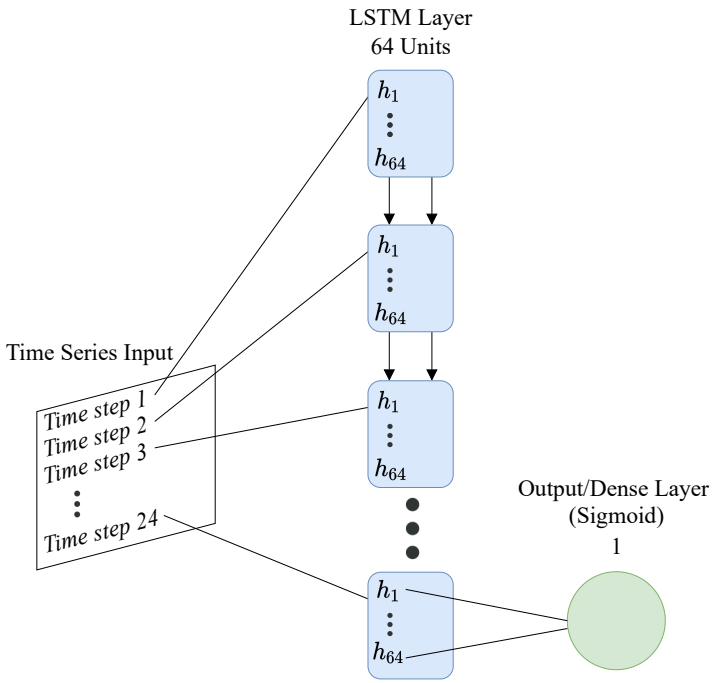


Figure 7.2: General architecture of LSTM.[2]

The LSTM model consists of an LSTM layer, with one cell state per timestep in the time series input, and n units in each cell. Following the LSTM layer is a dense layer with a sigmoid activation function. This gives a probabilistic output, which is the model's prediction for whether the patient develops sepsis.

7.4 GRU-D

GRU-D is another recurrent model, which is similar to the LSTM in design. The major difference between the LSTM and GRU-D in the general architecture is the input given to the model. Alongside the time series data, GRU-D is given the missingness representations as input. The architecture of GRU-D is shown in Figure 7.3.

[2]The LSTM architecture is a slightly modified version of Figure 7.2 from our previous semester report [2].

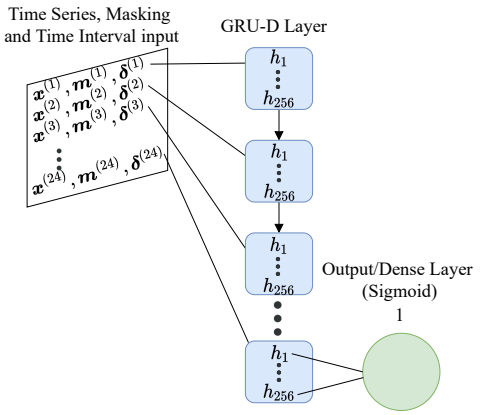


Figure 7.3: General architecture of GRU-D.

As with the LSTM, GRU-D has a cell state for each of the time steps. Each cell is also given the masking- and time interval representation, for the corresponding timestep. Following the GRU-D is a dense layer with a sigmoid function, giving the model's probabilistic prediction.

7.5 BRITS

BRITS is a model that uses a recurrent layer in its internal structure. In the official paper [21], they state that they use an LSTM as the recurrent layer, but any recurrent layer can be used. Therefore, we use LSTMs as the recurrent layer as well. The general architecture of BRITS is seen in Figure 7.4.

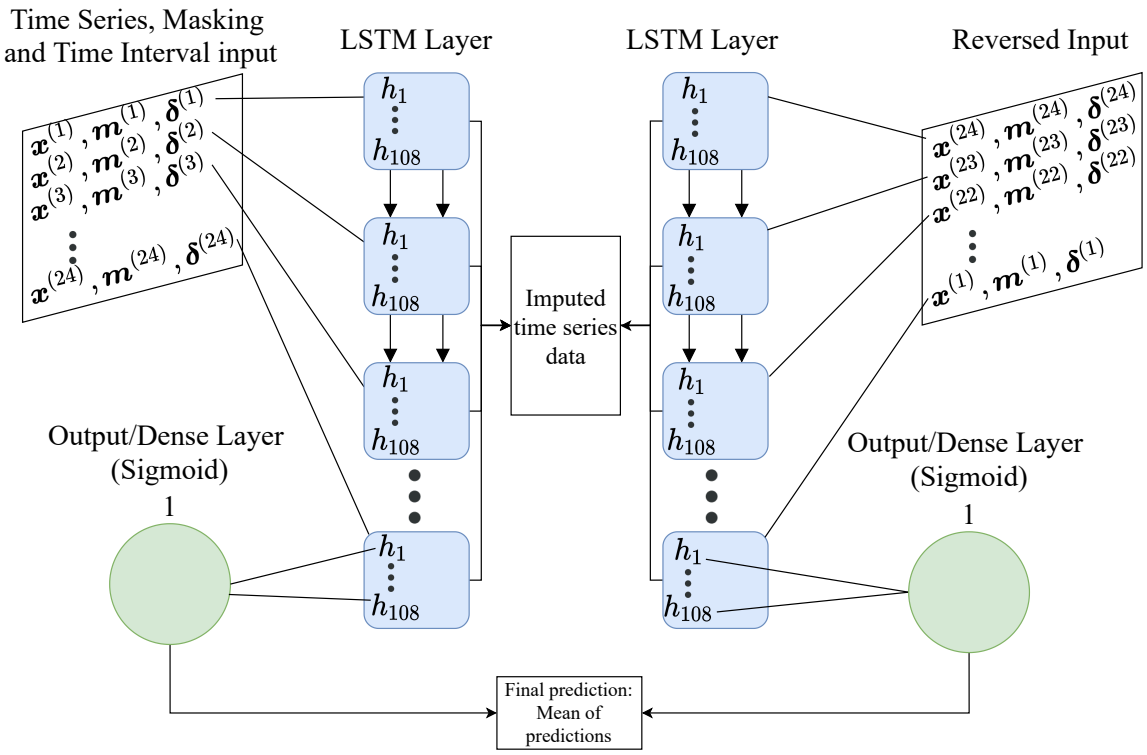


Figure 7.4: General architecture of BRITS.

BRITS uses the same input as GRU-D, which is the time series data, masking- and time interval representations. BRITS is a bidirectional recurrent approach, and therefore uses a second LSTM, which operates on the time series data in reverse.

Each cell in the LSTM outputs the imputed data for the corresponding timestep. As the cells in both recurrent layers output imputed data, the final imputation is the mean of each output. All the imputed time steps are combined to form the time series with imputed values.

Following both LSTMs, a dense layer with a sigmoid function is used. This results in two probabilistic predictions for whether the patient develops sepsis, one which is based on normal time series, and one based on the reversed time series. The mean of each prediction is used as BRITS' final probabilistic prediction.

7.6 Extracted Features

One of the ideas that we described in Chapter 5, is to extract additional features from the time series data and input this to the models. The features we extract are tabular, which means that each feature has one value that do not change over time. However, as our models only take time series data as input, the data or the model have to be modified to make the data and model compatible. In this section, we describe multiple ways of doing this and then select the approach we determine has the best potential. From our research, we found the following four ways to do it:

Inserting the extracted feature at every timestep similarly to how the demographic features are inserted in CinC2019. For example, if we extract the observation rate for a feature, then that rate would simply be inserted as an additional constant feature for every timestep.

Using the extracted features to calculate the initial hidden state of an RNN as Horn et al. did for the RNNs they compared their model to in [15]. They gave the demographics features to a neural network with one hidden layer with the same number of hidden units as the RNN and used it to calculate the initial hidden state of the RNN.

Use one model for predicting on time series data and another model to predict on the extracted features and use both predictions to determine the final prediction. This approach is called ensemble learning [30]. There exist many different ways to combine the predictions where one of the simplest is to take the average prediction of the models.

Create a neural network with two sub-networks with separate inputs, and combine their outputs before predicting which allows us to create a prediction based on both the time series information and the extracted features. An illustration of the architecture of this approach is shown in Figure 7.5.

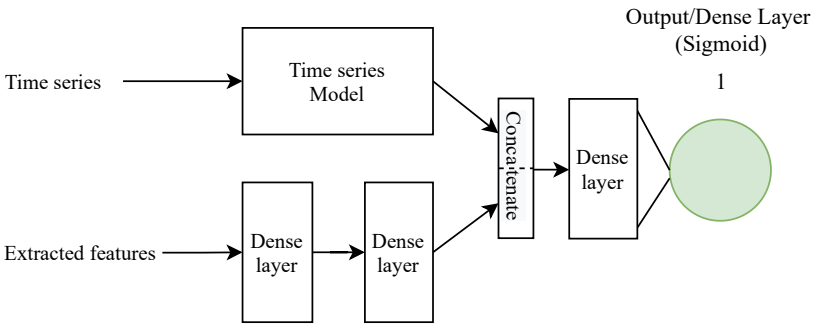


Figure 7.5: Using extracted features as additional input to the network.

where the time series data is given to the time series model as usual, and the extracted features are given to the first dense layer. The output of the second dense layer and the output of the time series model are concatenated, and given to a dense layer, followed by another dense layer with a sigmoid activation function, which gives the final probabilistic prediction.

Having neural networks with multiple inputs is not new and combining the outputs of two networks have also been done as we exemplified with the LSTM-CNN in Section 5.2. However combining neural networks in this way does not appear to be widely researched in academia, as we only were able to find one paper that uses this concept, which only has two citations [31].

7.6.1 Choice of Method

As extracting additional features is not the only thing we experiment with, we only select one of the methods to use in our experiment. We discard the option of using the same value at every timestep as we see it as a way to force the data to fit the model rather than making a model that fits the data. Using the extracted features to calculate the initial hidden state of the RNN cannot be used with the TCN. As all the hidden states in an RNN are based on the previous hidden state, the last hidden state should have information from the extracted features. However, at every timestep, some information from the previous hidden state is forgotten, which may reduce the amount of information from the extracted features in the initial hidden state. We, therefore, suspect that the amount of information from the extracted features is diminished when the hidden state for the final timestep is calculated. Therefore, we choose not to do it with the initial hidden state. The last two options, using an ensemble and using a model with mixed inputs are a bit similar as they both combine the output of two sub-models. The mixed input approach allows the model to consider the analysis from the time series together with the additional features, whereas the ensemble approach analyzes them separately. The ensemble approach does therefore not learn to consider the time series data and extracted features in the same context. For this reason, we choose to use the mixed input approach.

Part III

Experiments

Chapter 8

Evaluating Models

In this chapter, we describe how we evaluate our models. We state in our problem statement that we want to create a well-performing and well-calibrated model. These concepts can be measured using many different metrics, and therefore, we need to define how we measure them. We start by defining the metrics we use to evaluate a model's performance and calibration. Then we make some considerations we need to take into account when we compare the results of different models.

8.1 Performance

We plan to use Area Under Receiver Operating Characteristic (AUROC), Area Under Precision Recall Curve (AURPC), and Decision Curves (DC) for evaluating the performance of the models.

8.1.1 AUROC [1]

When evaluating the performance of the models we use AUROC, which is a recognized metric for evaluating medical diagnostic systems [32].

$$TPR = \frac{TP}{TP + FN} \tag{8.1a}$$

$$FPR = \frac{FP}{FP + TN} \tag{8.1b}$$

By evaluating a model by plotting true positive rate (TPR, Equation 8.1a) over the false positive rate (FPR, Equation 8.1b) at different probability thresholds, we can generate a curve that shows the relationship between TPR and FPR, also named ROC curve. The TPR is given by the true positives (TP) and the false negatives (FN). The FPR is given by the false positives (FP) and the true negatives (TN). An example of a ROC curve can be seen in Figure 8.1, where the area under the ROC curve is the AUROC of a given model. A perfect model has a ROC curve represented by the green line in Figure 8.1, and a model that makes random guesses is represented by the red line in Figure 8.1.

A more likely outcome is the blue line, which represents a model that is somewhere in between random and perfect.

[1] This section is a slightly modified version of Section 8.1.1 from our previous semester report [2].

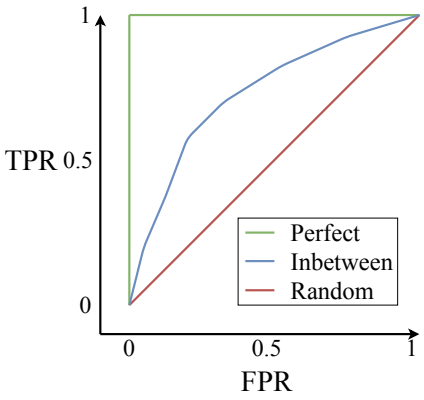


Figure 8.1: An ROC curve for a perfect model (green), random model (red), and a model somewhere in between (blue).

8.1.2 AUPRC [2]

Area under precision recall curve (AUPRC) is another metric for evaluating the performance of the model. The metric works by plotting the precision over the recall of a model, with different probability thresholds [33]. According to [34], if the data is imbalanced, AUPRC should also be considered. If a dataset contains a high ratio of negative samples compared to positive samples, a change in the number of false positives in Equation 8.1b will have a small impact on the FPR. On the other hand, if the change in false positives is large, this will directly affect the precision (Equation 8.2a), as false negatives are omitted. As the datasets we use in the experiments are imbalanced, we choose to also use the AUPRC metric for evaluating the performance of the models.

$$Precision = \frac{TP}{TP + FP} \tag{8.2a}$$

$$Recall = \frac{TP}{TP + FN} \tag{8.2b}$$

Figure 8.2 shows a perfect, random, and inbetween curve, in green, red, and blue respectively, where the random or baseline model is determined by the class ratio in the dataset [35].

$$Random = \frac{P}{P + N} \tag{8.3}$$

The AUPRC of a random model is computed as seen in Equation 8.3, where P and N is the number of positive and negative data samples respectively. The random model shown in Figure 8.2 is based on a dataset with a $1 : 9$ ratio of positive to negative samples, resulting in a baseline of 0.1.

[2]This section is a slightly modified version of Section 8.1.2 from our previous semester report [2].

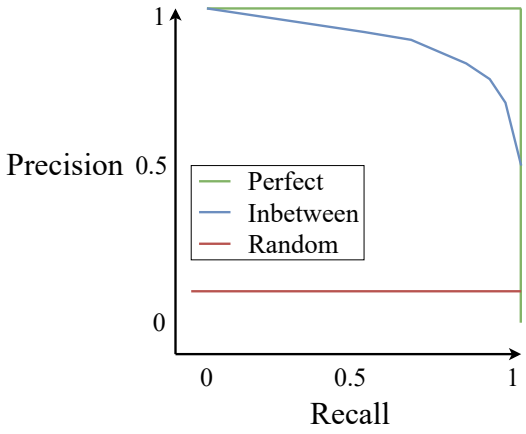


Figure 8.2: A PR curve for a perfect model (green), random model (red), and a model somewhere in between (blue).

The AUPRC for a random model used on the dataset we use in this project is shown in Table 8.1.

Dataset	Positive	Negative	AUPRC for a random model (baseline)
CinC2019A	19,228	764,185	0.025
CinC2019B	11,591	746,030	0.015
CinC2019	30,819	1,510,215	0.020
PCT	2,636	337,814	0.008

Table 8.1: AUPRC for a random model for each dataset.

The more an AUPRC increases from the baseline value, the better, meaning that a dataset with a low baseline has a lower threshold for what is considered a good AUPRC.

8.1.3 Net Benefit and Decision Curves

The consequence of misclassifying a sepsis-positive patient may be more severe compared to misclassifying a sepsis-negative patient. One metric that takes this into account is net benefit (NB), which can weigh true positives higher than false positives. Net benefit describes the benefit of treating the sepsis-positive patients minus the harm of starting treatment on sepsis-negative patients. NB is calculated as:

$$NB = \frac{true\text{-}positive\ count}{n} - \frac{false\text{-}positive\ count}{n} \cdot \left(\frac{p_t}{1-p_t}\right) \tag{8.4}$$

where n is the total number of samples, and p_t is the threshold probability for when the model's output should be interpreted as sepsis-positive [36]. As this threshold increases, models generally make fewer positive predictions. Increasing the threshold also makes the models weigh false positive predictions higher.

As we do not have the clinical expertise to know the optimal value for p_t in a real-world scenario, we make a decision curve (DC), which is the NB as a function of p_t. The clinical staff

can base p_t on how many sepsis-negative patients they allow to treat compared to how many sepsis-positive they treat. If they allow one false positive per true positive, then the threshold is 0.5 and if they allow nine false positives per true negative, then the threshold is 0.1. If the net benefit is positive for a model at the chosen threshold, then it is more beneficial for the medical staff to treat patients based on the model's predictions compared to not treating any patients. Figure 8.3 is an illustration of the DC for the following models:

- **A perfect model** will always have the highest possible NB regardless of the threshold.
- **Treat none** will always have zero benefit as no samples are classified as positive regardless of the threshold.
- **Treat all** will have a high NB when the threshold is low. When the threshold is increased, so is the weight of the false positive, which results in a drop in NB.
- **The example model** illustrates a more realistic DC for a real-world model.

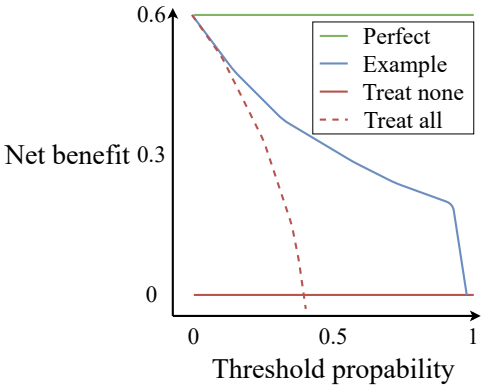


Figure 8.3: A DC for a perfect model (green), treat none (red), treat all (dotted red), and an example model (blue).

As it can be hard to tell from the graph if the model makes too few true positive predictions or too many false positive predictions, we also plot the true positive rate (TPR) and true negative rate (TNR) over the threshold in a separate graph.

8.2 Calibration [3]

The calibration of a model indicates how well the confidence output of a model fits with the actual probability of an event. The confidence output corresponds to the model's predicted probability of the event, as a value between 0 and 1. If the model is well-calibrated, the higher the confidence is, the more likely the event is, and vise versa.

If a neural network outputs ten predictions for the next ten days, where each prediction has a confidence of 0.3 for rain, we expect it to rain on three out of ten days. This means that the confidence output should be equal to the actual probability. However, this is not always the case for neural networks, as proposed by the paper by Guo et al. [6], where they discover that modern neural networks are often miscalibrated. This is problematic if such a network

[3] This section is a slightly modified version of Section 5.2 from our previous semester report [2].

is used for high risk decision making, such as whether to brake in self driving cars, or as a second opinion system for clinical use. For example, if a network has a low confidence of 0.1 that a patient gets sepsis, and $\frac{9}{10}$ patients with similar symptoms get sepsis, we have a bad calibration that might lead to a missed diagnosis. Perfect calibration can be formally defined as [6]:

$$\boldsymbol{P}(\hat{Y} = Y|\hat{P} = p) = p, \forall p \in [0, 1] \tag{8.5}$$

Given $\hat{P}$, the predicted confidence, we assess the probability of the predicted class label $\hat{Y}$ being equal to the actual ground truth class label Y. For a perfectly calibrated model, this expression should be equal to the confidence for a specific prediction p. The left hand side of Equation 8.5 corresponds to the accuracy of a model, and the right hand side corresponds to the confidence of a model. Since perfect calibration cannot be measured with a finite dataset, the accuracy and confidence are approximated by splitting the predictions into M bins, B_m, and then calculated for each bin using Equation 8.6 and Equation 8.7.

$$acc(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbf{1}(\hat{y}_i = y_i) \tag{8.6}$$

Equation 8.6 shows how to compute accuracy for a binary classifier. Here, $\hat{y}_i$ is the predicted class label for sample i, y_i is the ground truth class label for sample i, and $\mathbf{1}$ is an indicator function. The indicator function is used to determine whether the predicted label is equal to the ground truth label, where 1 is returned if they are equal, and 0 is returned otherwise. Note that each prediction is considered positive, such that x predictions with a confidence of 0.2 should achieve an accuracy of 0.2 to be perfectly calibrated.

$$conf(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i \tag{8.7}$$

Equation 8.7 shows how to compute the confidence for a binary classifier. Here, $\hat{p}$ is the predicted probability that sample i is positive, such that the confidence of a bin equals the average predicted probability for that bin. By partitioning the predictions into bins, we can compute the difference between accuracy and confidence for each bin as a measure for calibration error.

In the following sections, we describe three calibration metrics as well as the reasons for using them when evaluating our models.

8.2.1 Expected Calibration Error

The calibration measure presented by Guo et al. [6] is called Expected Calibration Error (ECE), and is defined as:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)| \tag{8.8}$$

Here, the predictions are partitioned into M bins, for a total number of n data samples across all bins. Each bin includes the predictions from a specific prediction confidence interval, e.g.

all predictions with confidence between 0 and 0.2. A perfectly calibrated model has an ECE value of zero, and the higher the value the worse calibrated the model is.

The comparison between confidence and accuracy can be visualized in a reliability diagram, as seen in Figure 8.4, with confidence on the x-axis and accuracy on the y-axis. As mentioned earlier, perfect calibration is when the confidence is equal to the accuracy, which can be seen as the diagonal line. The blue bars represent the average accuracy for a given bin and the transparent red bars represent the difference between the calibration of the model and perfect calibration. When the bar is above the diagonal, as with the sixth bin, it indicates that the model is underconfident for that bin, since the accuracy is higher than the confidence. Likewise, if the bar is below the diagonal, as with the last bin, it indicates that the model is overconfident, since the confidence is higher than the accuracy.

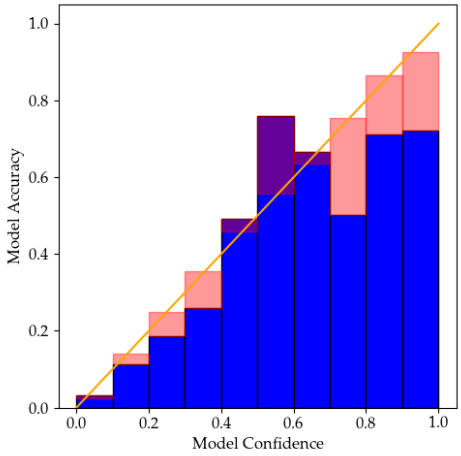


Figure 8.4: Example reliability diagram for the ECE measure.

While ECE is widely used, Nixon et al. [37] explores some issues with it. We do not go into great detail about them here, but only provide a short description:

- Generally, only a few bins contribute to the calibration error.
- There is a bias-variance trade-off relating to the number of bins. By increasing the number of bins, the bias decreases due to the finer granularity, however, the bins will have fewer samples, leading to higher variance in the bins.
- Overconfident and underconfident predictions in the same bin can cancel each other out, resulting in a smaller error. While it (most likely) will not push a very uncalibrated model to low calibration error, it can make it difficult to compare two models, and determine whether an improvement was due to the better calibration or a higher cancellation effect.

Despite these issues, we still consider ECE a useful metric. Each bin can show where the model might be poorly calibrated by creating a reliability diagram. However, in this case it is important to consider the number of samples making up a bin. If a bin has high error, it might simply be due to chance, if only a few samples makes up that bin.

8.2.2 Adaptive Calibration Error

Nixon et al. [37] presents another calibration measure, called Adaptive Calibration Error (ACE). ACE is adapted to consider all predictions in a multi-class classification setting, and attempts to tackle the bias-variance trade-off, which it does by using a different binning scheme. The equation for ACE is defined as:

$$ACE = \frac{1}{KR} \sum_{k=1}^{K} \sum_{r=1}^{R} |acc(r,k) - conf(r,k)| \tag{8.9}$$

Here, K is the number of classes, R is the number of ranges, and $acc(r,k)$ is the accuracy for class k in range r. The ranges work similarly to bins, but rather than including an interval for confidence, each range includes a specific number of the total predictions, sorted by confidence value, such that all predictions are spread evenly across the R ranges. This prevents the cases where the bins are imbalanced, meaning that some bins include the majority of predictions, and other bins include only a small number of predictions.

Nixon et al. [37] motivates the use of adaptive calibration ranges by arguing that in order to best estimate the calibration error, the metric should focus on the regions where predictions are made. For these reasons, we consider ACE a valuable second calibration metric.

8.2.3 Maximum Calibration Error

Naeini et al. [38] presents the calibration measure Maximum Calibration Error (MCE). MCE calculates the worst-case deviation between confidence and accuracy, which may be desirable to minimize in high-risk applications, where reliable confidence is absolutely necessary [6]. MCE is given by Equation 8.10.

$$MCE = \max_{m \in 1,\ldots,M} |acc(B_m) - conf(B_m)| \tag{8.10}$$

MCE is calculated similarly to ECE, but instead of calculating the weighted average of all the bins, the error is simply the error of the bin with the highest error. Considering the reliability diagram, this error corresponds to the bin that deviates the most from the diagonal. This is useful when considering imbalanced datasets [39], which is the case for our datasets, as discussed in Section 2.8. Since the majority of our data samples are negative, the majority of prediction confidences may be in the lower interval bins, which means that higher interval bins may be badly calibrated. However, since ECE uses a weighted average for each bin, a bin with a very small number of predictions does not have a big impact on the overall ECE value. In these cases, MCE is able to find badly calibrated bins, since it takes the bin with the highest error regardless of the number of samples in the bin.

However, it is important to note that MCE is likely to be high for bins with very few samples, due to the variance of these bins. It is therefore important to consider the number of samples, which makes up that bin, when considering MCE.

8.3 Comparing Models

We state in our problem statement (Chapter 3), that we want to compare our model to the XGBoost model. As we suspect that one model does not dominate all the other models, we describe important aspects on how to interpret the metrics (described in Section 8.1 and Section 8.2), when we evaluate and compare models.

Regarding the performance metrics, we mention in Section 8.1 that AUROC is likely to be high, due to a large number of negative samples in CinC2019 and PCT. This is because FPR will be small if the number of TN is high. This problem is not present for AUPRC, since TN is not included in the equation. AUPRC only considers TP, FP, and FN, which is useful when we have a large number of negative cases. Therefore, the AUPRC considers how well a model can predict the positive cases, which the clinical staff are likely to be more interested in, while still considering when it is unable to predict the negative cases. AUROC tells us more about how well a model predicts any of the classes (both positive and negative). For these reasons, an increase in AUPRC is preferable even if it results in a small decrease in AUROC. However, it is important to note that it is not preferable to get a small increase in AUPRC for a large decrease in AUROC.

While AUPRC considers how well a model predicts positive samples, it does not consider that the consequence of misclassifying a sepsis-positive patient may be more severe compared to misclassifying a sepsis-negative patient. In this case, we consider NB and DC to evaluate the model's performance. Since we do not have the expertise to know what the correct threshold value for NB is, our analysis focuses more on the DC, and we compare the performance of models at different thresholds.

An important aspect of NB is that it depends on the model's calibration, where miscalibration always results in worse NB [40]. As mentioned in Section 8.2, calibration is an important metric for models in clinical settings, but if we want to interpret the NB and DC, we need to have a well-calibrated model. To get a better understanding of the DC, we also consider the TPR and TNR in relation to the DC.

ECE gives useful information about the calibration of the model by providing an overall metric of how well-calibrated all the predictions are. However, one issue is that most of the predictions tend to be in one or a few bins [37]. Therefore, when we evaluate a model on ECE, we need to consider the distribution of the predictions making up the bins. For example, a well-calibrated bin with 90% of the samples might result in a low ECE, regardless of the remaining bins. A bin with few samples is likely to have high variance and the error for that bin might not be representative of the actual error of the confidence range, as the calibration error can not accurately be estimated from few samples. This also affects MCE, since MCE is based on the bin with the highest error, which is likely to have a low number of samples. This problem is not present for ACE, as each range consists of the same number of samples. When estimating the calibration, it is important to consider all three selected metrics, as each of them provides useful information.

Chapter 9

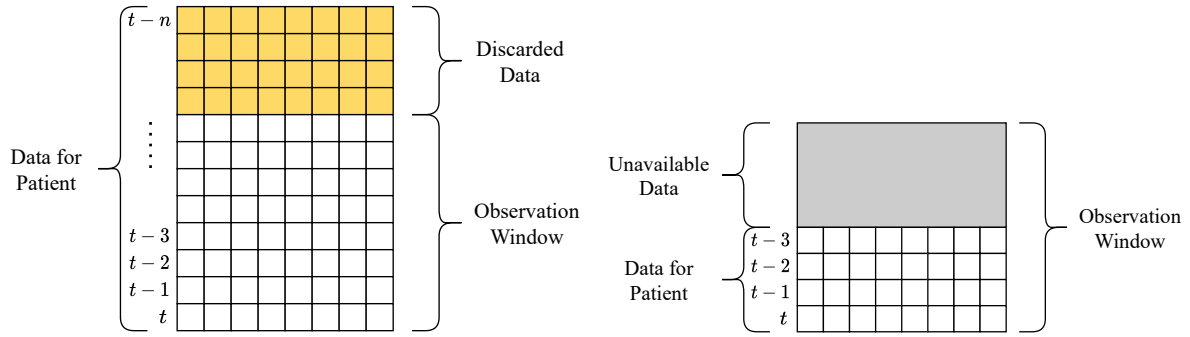
Observation Window Details

As a final consideration before we begin describing our experiments, we need to define the observation window for the framings we chose for CinC2019 (sliding window) and PCT (on clinical demand) in Section 2.5.

9.1 Observation Window Types

In this section, we describe different observation window types. As we describe in Section 2.4, if we attempt to predict at time t, whether a patient develops sepsis, the number of hours of previously observed data we use to make that prediction is called the observation window.

The observation window size can have two types: fixed size and variable size. The size of the fixed size observation window is the same for all samples in the dataset, while the size of the variable size observation window varies across samples. As each sample with a fixed size observation window uses the same number of timesteps, it allows for more types of neural networks to be used [22]. This has the effect of discarding data not in the observation window, for samples with more timesteps than the length of the observation window, as seen in Figure 9.1a. If the sample has fewer timesteps than the size of the observation window, as in Figure 9.1b, it is necessary to handle this, such that every sample in the dataset has the same number of timesteps.



(a) Sample with excessive timesteps.

(b) Sample with insufficient timesteps.

Figure 9.1: Two data samples with different number of timesteps, and how fixed size observation window affects them.

For the variable size observation window, all the timesteps can be used. Thus, it is unnecessary to handle the case where there are too few timesteps for the observation window. However,

using a variable size observation window limits the type of neural networks which can be used [22].

We also introduce a variant to variable size observation window: max size observation windows. The max size observation window is the same as variable size observation, but with a limit to how big the observation window can be. This allows the window to be any size up to and including the maximum window size. By doing this, we can limit the amount of data we consider when making a prediction, which can be useful if we assume the most recent data is most relevant.

9.2 Processed CROSS-TRACKS dataset

PCT is created by Enversion with a 24-hour prediction window and a fixed size observation window, where all patients with admissions shorter than 24 hours have been discarded. The maximum size of the observation window is therefore 24 hours. In our previous project [2], we observed that models trained on PCT had slightly better calibration and performance metrics with an observation window size of 24 compared to 12. Due to this, we choose a 24-hour observation window for PCT.

9.3 PhysioNet Computing in Cardiology Challenge 2019 dataset

In order to choose the type of observation window for CinC2019, it is important to consider how it affects the real-world use-case. For the fixed size observation window, we need to handle the case where the size of the window is larger than the available data. One solution is to make the observation window the size of the smallest input. We discard this option, as the smallest input is four timesteps (see Section 2.2.2), which we consider insufficient data for meaningful predictions, especially when we have many samples with more data.

Another solution is to simply discard the samples with insufficient data. There are two problems with this approach. The first problem is that our positive samples are more represented at smaller admission lengths than negative samples are. Discarding these would result in a different positive to negative ratio than for the real-world usage, which can lead to poor calibration [2]. The second problem is that it would imply that the medical staff uses the model only after a certain admission time. E.g. if the size of the observation window is 12 hours, then the model would only be useful after the patient has been admitted for 12 hours or more. However, this does not correspond well to the real-world use-case, as the majority of patients develop sepsis early in their admission (see Figure 2.3). Due to these problems, we choose not to discard samples based on insufficient data.

As shown in Figure 9.1b, when we have insufficient data, this option requires some assumptions on how to create the unavailable data. These assumptions introduce more com-

plexity in our experiments. Therefore, in order to limit the scope of this project, we do not choose this option.

Variable and max size observation window does not have these problems, as we can use short samples without having to consider insufficient data. Additionally, all the models we experiment with supports variable observation window size. In our previous project [2], we observed that there was no considerable change in using 12, 24, 36, and 60 hours for the observation window size. Therefore, in order to limit the computation time of our models, we choose a max size observation window. We also note that 97.5% of the admissions have at least 12 hours of data, where only 76% have at least 24 hours. For these reasons, we choose a 12-hour observation window for CinC2019.

Chapter 10

Experiment Setup

In this section, we describe the experiment setup we use when we perform our experiments. By using a common setup across all experiments, we can evaluate how well one change affected the model compared to another, which is information we can use for further experimentation.

10.1 Implementation Environment

We start by defining the environment we train and evaluate our models in. We use TensorFlow 2.4.1 as our machine learning framework [41], as this is the most recent version of TensorFlow at the writing of this project. For the XGBoost model, we use the XGBoost Python library version 1.3.3 [42]. We also use Python 3.7.x.

10.2 Data

CinC2019 uses the sliding window framing and PCT uses the on clinical demand framing. As we describe in Section 2.6, we use a 12 hour prediction window for CinC2019 and 24 hour for PCT. We use a 12-hour max size observation window for CinC2019 and a 24-hour fixed size observation window for PCT, as we describe in Chapter 9. We also split the data into a train, validation, and test set with 70%, 15%, and 15% of the data respectively. The data is split such that data from one admission is only present in one of the sets. The training set is used for training the model, the validation is used to determine whether the models overfit to the training data and to evaluate the models between experiments, and the test set is used for the final validation to ensure that we have not overfitted the models to the validation set.

The metrics we report for our experiments are from the validation set, and we use the test set when we do the final evaluation of our models after the experiments. The data given to the TCN and LSTM is imputed with LOCF imputation. For XGBoost we use the delta representation, as we describe in Section 2.4.

As we describe in Section 2.2, CinC2019 consists of two datasets referred to as CinC2019A and CinC2019B. The datasets has data from 1,790 and 1,142 positive admissions respectively, which might be too few for the model to learn from [22]. Therefore, we conduct a preliminary experiment where we evaluate how using the datasets separately affects the model performance, which is described in further detail in Appendix E.1. From the result of this experiment, we choose to use the data from the two hospitals combined as one dataset (CinC2019).

10.3 Training the Model

In each experiment, we make one change for each model, using predefined values, e.g. the number of layers in a model. For each of these changes, we train the model five times. As the neural networks are initialized with random weight, we train each model five times to determine if the change in performance is due to the changed factor in the experiment or the random initialization of the weights.

After training the model, we calculate the AUROC and AUPRC and plot the DC and graphs for TPR and TNR curves on the validation data. To consider the calibration of the model, we calculate the value of the ECE, ACE, and MCE metrics on the validation data. For ECE, we also show a reliability diagram and a histogram of the distribution of the confidences on the validation data.

When training, one important aspect to consider is overfitting. Overfitting occurs when a model makes predictions based on patterns that appear in the training data, but not in the testing data. This can be observed when the value of a metric does not follow the same trend for the training data compared to the testing data [43]. In order to evaluate whether the model is overfitted to the training data, we plot the loss over epochs for both the training data and the validation data. In order to avoid overfitting, we train the models with early stopping. Early stopping is a technique, where we monitor the value of a metric, and if that metric gets worse, we stop training [44]. The metric we monitor is the AUPRC of the validation set at the end of each epoch. However, it is not guaranteed that the AUPRC will consistently increase over each epoch. It can decrease for one or more epochs, even though it will increase in the following epochs. In order to mitigate this problem, we use "patience" for early stopping. Patience is a feature available in TensorFlow, which allows us to wait a specified number of epochs before we stop training, even though the AUPRC gets worse [45]. We use a patience of five epochs, which means that we continue training five epochs after our best observed value. Additionally, the final model we save is the model with the best validation AUPRC.

Finally, the models we compare XGBoost to are based on TCN, LSTM, BRITS, GRU-D. We use the Adam optimizer [46] for optimizing the NN models. We choose this as it worked well in our previous project [2]. We use the parameters recommended by Kingma and Ba [46], which is: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. For the loss function, we use binary cross entropy, as defined by Equation C.3.

For XGBoost, Enversion has already chosen the following hyperparameters for their use-case, which we also use:

- Max depth: 7
- Learning rate: 0.15
- Number of rounds for boosting: 75
- Lambda: 8
- Minimum child weight: 7

We treat XGBoost as a black box, as the purpose of this project is to create a better neural network model compared to Enversion's XGBoost model. For more information on the hyperparameters, see the XGBoost documentation [47].

As neural networks perform differently depending on their hyperparameters, we conduct a preliminary experiment where we tune the models' hyperparameters. For TCN, we tune the type of activation function, number of temporal blocks, and number of filters. For LSTM, BRITS and GRU-D, we tune the number of units. We describe why we decide to tune these hyperparameters and how we choose their values in Appendix E.5. The hyperparameters we choose to use in our experiments are:

- TCN: Relu Activation function, 2 temporal blocks, and 64 filters.
- LSTM: 64 Units.
- BRITS: 108 Units.
- GRU-D: 128 Units.

10.4 Summary

In summary, we use the following setup for our experiments:

- Python 3.7.x with TensorFlow 2.4.1 and XGBoost 1.3.3.
- Train each model five times.
- Early stopping with a patience of five epochs and save the best performing model.
- Loss over epochs for training and validation data.
- AUROC, AUPRC, ECE, ACE, and MCE values and DC plot from validation data after training the model.
- On clinical demand for PCT, with fixed size observation window of 24 and prediction window of 24 hours.
- Sliding window for CinC2019, with max size observation window of 12 hours and prediction window of 12 hours.
- CinC2019 is used as the combined dataset of CinC2019A and CinC2019B.
- Train/validation/test set is 70%/15%/15% of the data respectively.
- We test on TCN, LSTM, BRITS, XGBoost, and GRU-D models.
- XGBoost uses the delta representation.
- XGBoost uses the same hyperparameters as those provided by Enversion.
- TCN, LSTM, BRITS and GRU-D uses the hyperparameters from our hyperparameter tuning.
- LOCF imputation for TCN and LSTM.
- Adam as an optimizer (with parameters as recommended by Kingma and Ba [46]).
- Loss function: Binary Cross Entropy.

Chapter 11

Description of Experiments

In this chapter, we describe the experiments we perform in Chapter 12. The experiments are derived from the ideas we present in Chapter 5.

11.1 Class Weighted Loss Function

As CinC2019 and PCT are very imbalanced, we want to try to alleviate the problems that result from the CIP. TensorFlow provides a simple approach to this problem, where the loss is calculated differently depending on the class [48]. We use TensorFlow's approach to assign a higher weight to the minority class. We note that this is not the same approach used by Geng and Luo [18], or Fernando et al. [10], which were the papers that initially inspired the idea of weighing the classes differently. Simply multiplying the loss for the minority class with a weighting factor would increase the magnitude of the total loss, which would cause larger changes in the network weights than without it. However, changing the magnitude of the total loss is avoided by calculating the weights for each class as:

$$w_n = \frac{t}{n + p \cdot r} \tag{11.1a}$$

$$w_p = r \cdot w_n \tag{11.1b}$$

where t is the total number of samples, p and n are the number of positive and negative samples respectively, and r is the ratio of how much higher the loss should be for the samples from the positive class compared to the negative class.

The purpose of this experiment is to test different values for r. As we show in Table 2.2, the ratio of positive to negative samples for PCT is $1 : 128$. Since we have combined CinC2019A and CinC2019B, the ratio is $1 : 49$ for CinC2019. We therefore experiment with $r = 49$ for CinC2019 and $r = 128$ for PCT. Additionally, we also experiment with $r = 4$ and $r = 16$ to see the results of placing the weighting in the middle.

11.2 Extracting Additional Features

In this section, we explore ideas about extracting different features from the data, as we mention Chapter 5. We propose a model architecture in Section 7.6, that incorporates the idea of including extracted features in the model, where the extracted features are represented as a

vector to a separate network, which is then joined with the time series model. The first feature we consider to extract is the demographics in CinC2019, as this easily can be represented as a vector. Since all the demographics, except ICULOS, do not change over time steps, simply extracting them as a separate input, might be beneficial for the model. ICULOS describes how long the patient has been admitted to the ICU. To represent this as a single value, we use the value at the first timestep in the observation window. Due to this, we choose to experiment with demographics as a separate vector. We note that we do not perform this experiment on PCT, as PCT contains no demographic features.

The second feature we wish to extract is the observation rate, as Singh et al. [19] reports that sepsis-positive patients tend to have more measurements for certain features. We therefore also want to experiment with the observation rate as a separate vector input to the model, with a value of the observation rate for each feature. We calculate the observation rate for a feature f as $\frac{n_f}{\tau}$, where n_f is the number of observed measurements for feature f and τ is the number of timesteps in the observation window. For example, two measurements for a 12-hour observation window would result in an observation rate of $\frac{2}{12}$.

11.3 Missingness Representations

Che et al. [20] reports that using the missingness representations (described in Section 6.4) on a regular GRU improves its performance. Due to this, we perform an experiment where we add the missingness representations to the input of the LSTM and TCN. As neither the TCN nor the LSTM has any specific way of handling this data, we concatenate the time series data with the missingness representations along the feature axis using Equation 6.15 and gives this as input to the models.

Chapter 12

Experiments

In this chapter, we present the results from the experiments described in Chapter 11. To see how the changes we make in the experiments affect the models' performance and calibration, we first evaluate the models without any changes. We use this as a reference point, and refer to these models as baselines when we compare the experiment results to them.

The performance for CinC2019 and PCT is shown in Table 12.1. While the AUPRC of the models is low, this is expected. As described in Section 8.1.2, the baseline AUPRC is determined as the number of positive samples over the total number of samples, and because CinC2019 and PCT have many negative samples compared to positive samples, the baseline AUPRC is small (0.020 and 0.008 respectively). Lauritsen et al. [9] show the AUPRC of XGBoost with different data framings, where XGBoost had an AUPRC of 0.014 using the clinical demand framing (with a $1:244$ sepsis-positive to -negative ratio, and 0.004 baseline AUPRC). The only difference between the data Lauritsen et al. used and PCT is the observation and prediction window size, which is 12 hours in [9] and 24 hours in PCT. As the AUPRC baseline for the data used in [9] is half of the AUPRC baseline for PCT, we expect the AUPRC of XGBoost to be higher than 0.014 in our experiments. Our XGBoost baseline show an AUPRC of 0.140, which is higher than we expected. However, it seems reasonable, as Lauritsen et al. get an AUPRC of 0.385 when using the fixed time to onset framing with a $1:15$ sepsis-positive to -negative ratio.

Baseline	CinC2019		PCT	
Model	AUROC	AUPRC	AUROC	AUPRC
TCN	0.753 ± 0.011	0.083 ± 0.006	0.750 ± 0.007	0.032 ± 0.006
LSTM	0.764 ± 0.007	0.076 ± 0.003	0.746 ± 0.011	0.029 ± 0.004
BRITS	0.769 ± 0.012	0.089 ± 0.007	0.687 ± 0.052	0.020 ± 0.002
GRUD	0.731 ± 0.009	0.066 ± 0.001	0.741 ± 0.011	0.025 ± 0.002
XGBoost	0.816 ± 0.000	0.114 ± 0.000	0.831 ± 0.000	0.140 ± 0.000

Table 12.1: AUROC and AUPRC for all models on CinC2019 and PCT.

The ECE and ACE for all models on CinC2019 are around 0.007 except for TCN, which is around 0.01. For PCT they are between 0.001 and 0.002. However, by looking at the reliability diagrams and sample confidence distributions in Figure F.1 and Figure F.2 for CinC2019 and Figure G.1 and Figure G.2 for PCT, we see almost all predictions are in the first bin and only the first few bins are well calibrated. This shows that the models are badly calibrated, but ECE is low, as it weighs the calibration error for each bin, based on the number of samples in each

bin. Looking at the remaining nine bins, we see that the models either do not predict with high confidence or are poorly calibrated in these bins. Therefore, we focus more on the diagrams than on the calibration metrics when we compare how well the models are calibrated.

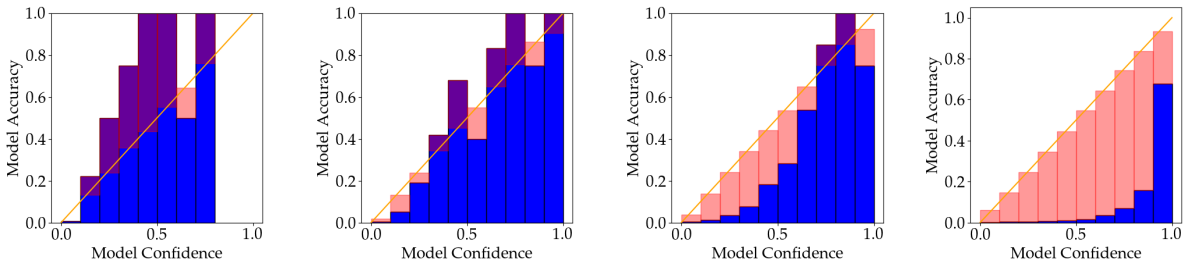
12.1 Class Weight Experiment Results

In this section, we describe the experiment of weighing the loss differently depending on the class label. As described in Section 11.1, we experiment with different values of the ratio r in Equation 11.1. The values that we use depend on the dataset, and are: 4, 16 and 49 for CinC2019, and 4, 16 and 128 for PCT.

The most interesting points are as follows:

- For CinC2019, generally all models get worse calibration and performance, as the ratio increases.
- The same applies for PCT, except for BRITS and GRU-D, which seem to receive slightly better performance.
- Increasing the ratio results in more even distributions of predictions, but in turn, makes the models more overconfident.

For CinC2019, the calibration and performance tend to get worse, the larger ratio we use. We see the same trend for PCT, with some variation on GRU-D, BRITS, and XGBoost. GRU-D has a slight increase in AUROC, but the best ratio is inconclusive, as they show results within the error margins. BRITS shows an improvement in both AUROC and AUPRC. XGBoost appears to become better calibrated with a ratio of 4 when looking at the reliability diagrams in Figure 12.1.

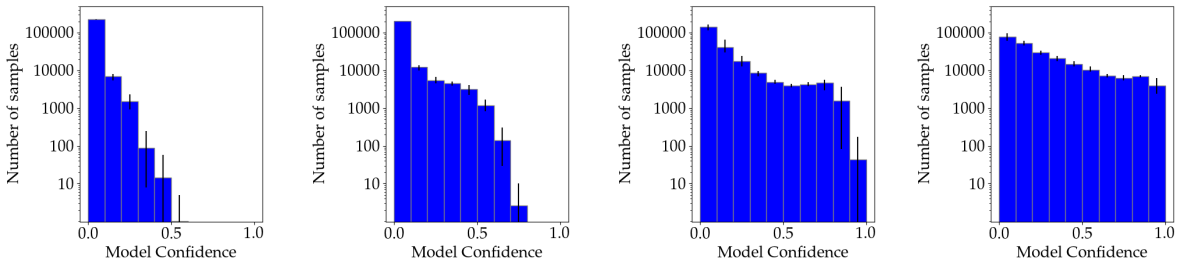


(a) XGBoost baseline **(b)** Ratio: 4 **(c)** Ratio: 16 **(d)** Ratio: 128

Figure 12.1: Reliability diagrams for XGBoost for the class weight experiment with PCT.

However, the high confidence bins for XGBoost contain less than 10 predictions each, which is too few predictions to determine whether these bins are well-calibrated. As the calibration metrics of XGBoost are worse as a result of worse calibration in the first two bins, it is debatable if using a ratio of 4 improved the calibration. With class ratios higher than 4, XGBoost becomes more overconfident.

We observe that increasing the ratio results in higher confidence predictions, which are more evenly distributed. Figure 12.2 shows an example of this, where we see the confidence distribution changes as the ratio increases.

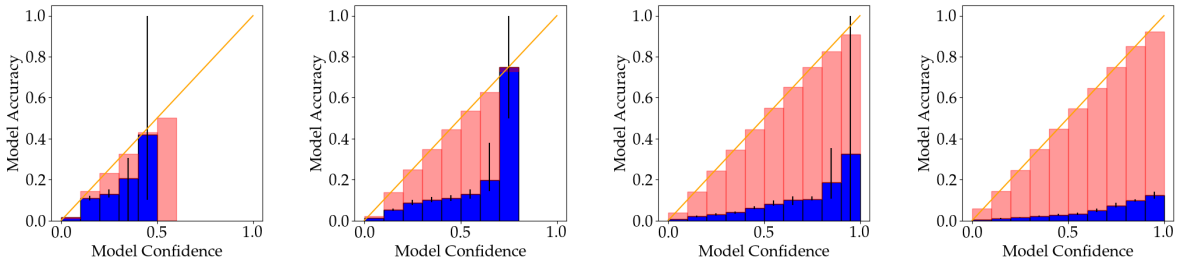


(a) LSTM baseline **(b)** Ratio: 4 **(c)** Ratio: 16 **(d)** Ratio: 49

Figure 12.2: Sample confidence distribution graph of LSTM for the class weight experiment with CinC2019.

Here we see that the LSTM makes more predictions with high confidence as the ratio increases.

While the models more often make high confidence predictions, we observe that the models become overconfident, as the ratio increases. We show an example of this with the LSTM in Figure 12.3.



(a) LSTM baseline **(b)** Ratio: 4 **(c)** Ratio: 16 **(d)** Ratio: 49

Figure 12.3: Reliability diagrams of LSTM for the class weight experiment with CinC2019.

Here, we see a consistent pattern, where the model becomes more overconfident, as the ratio increases.

12.2 Demographic Experiment Results

In this section, we describe how using the demographic features as a separate input to the model, affects the calibration and performance. As described in Section 11.2, this experiment is only performed on CinC2019, as PCT does not have demographic features.
The most interesting points are:

- Worse performance.
- The models predict with lower confidences.
- The models become more overconfident.

We see that using the demographic features as a separate input to the model does not yield any beneficial results. The calibration metrics seem unchanged to the baselines, but the performance metrics are worse. We note that BRITS appear to have a significantly low MCE (Table F.5), but this is due to the model only having 3 bins with low confidence predictions (Figure F.12c). Compared to the baselines, all models predict with lower confidences, and are more overconfident.

12.3 Missingness Representation Experiment Results

In this section, we describe how adding the missingness representations to the input of the TCN and LSTM, affects the calibration and performance.
The most interesting points are:

- For CinC2019, the models have better performance.
- For PCT, similar results to the baselines, with the exception of larger error margins in TCN's performance.

The biggest difference in this experiment is in the performance of the models. The performance metrics for the models and baseline models, for both CinC2019 and PCT, are shown in Table 12.2.

		Baseline		Missingness Representation	
Dataset	Model	AUROC	AUPRC	AUROC	AUPRC
CinC2019	TCN	0.753 ± 0.011	0.083 ± 0.006	0.772 ± 0.016	0.087 ± 0.005
	LSTM	0.764 ± 0.007	0.076 ± 0.003	0.779 ± 0.007	0.091 ± 0.003
PCT	TCN	0.750 ± 0.007	0.032 ± 0.006	0.747 ± 0.014	0.040 ± 0.020
	LSTM	0.746 ± 0.011	0.029 ± 0.004	0.758 ± 0.006	0.031 ± 0.003

Table 12.2: AUROC and AUPRC for the TCN and LSTM for CinC2019 and PCT.

We observe that the models with CinC2019 generally improve by using the missingness representations. The TCN seems to improve in both AUROC and AUPRC, but can still be interpreted to be within the error margins of the baseline. It is more noticeable that the LSTM benefits from the missingness representations, especially in AUPRC.

For PCT, we observe that the TCN results are inconclusive, as they seem to achieve similar results, but have larger error margins compared to the other models and the baselines. The LSTM with PCT however, seems to achieve slightly better performance, but nothing out of the error margins. It is therefore difficult to determine whether the missingness representations benefit the LSTM with PCT as well.

12.4 Observation Rate Experiment Results

In this section, we describe how extracting the observation rate features from the time series data, and giving it as a separate input to the models, affects their calibration and performance. The most interesting points are:

- XGBoost, LSTM, and TCN either stay unchanged in their performance and calibration or are slightly better.
- BRITS' AUPRC improve.
- GRU-D on PCT has significantly better AUPRC and calibration, while the improvement on CinC2019 is not as significant.
- The models seem to benefit more from the observation rate on PCT compared to CinC2019.
- On PCT: TCN, BRITS, and GRU-D make more high confidence predictions.

LSTM and XGBoost on both CinC2019 and PCT show no benefit on performance or calibration TCN on CinC2019 also shows no benefit. The TCN on PCT appears to have an increase in AUPRC, however, whether this is the case is difficult to determine due to the large error margins. On both CinC2019 and PCT, BRITS' AUPRC increases, while the AUROC remains relatively unchanged.

We observe that the observation rate greatly benefits the GRU-D model, compared to the other models. One benefit is in the performance metrics of the GRU-D models, which we show in Table 12.3.

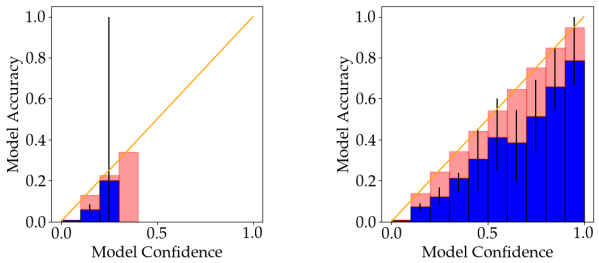
Dataset	Model	AUROC	AUPRC
CinC2019	Baseline	0.731 ± 0.009	0.066 ± 0.001
	GRU-D	0.767 ± 0.007	0.084 ± 0.003
PCT	Baseline	0.741 ± 0.011	0.025 ± 0.002
	GRU-D	0.735 ± 0.016	0.126 ± 0.012

Table 12.3: Performance metrics of the baseline GRU-Ds and the GRU-Ds in this experiment

We observe here that for CinC2019, the GRU-D models show a consistent improvement in both AUROC and AUPRC. For PCT, the AUROC remains relatively unchanged, while we see a significant change in AUPRC.

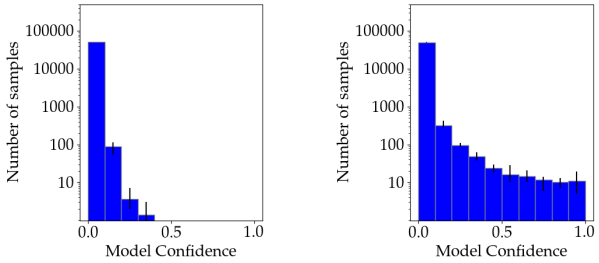
Based on the calibration metrics for the models with PCT, it seems that the calibration of the models is unaffected by the observation rate. However, by looking at the reliability diagrams and prediction distributions, we observe that TCN, BRITS, and GRU-D make more predictions with higher confidence.

To show this, we give an example in Figure 12.4, where we show the reliability diagrams and prediction distributions for the GRU-D model.



(a) Baseline Reliability diagram.

(b) Observation rate reliability diagram.



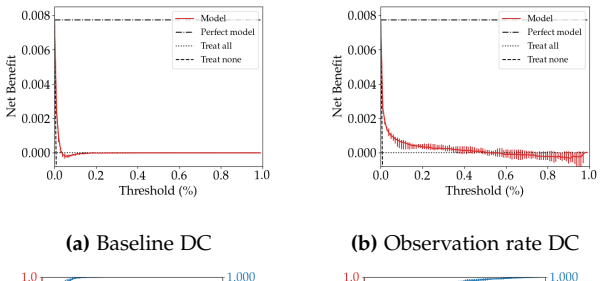
(c) Baseline prediction distributions.

(d) Observation rate prediction distributions.

Figure 12.4: Reliability diagrams and prediction distributions of GRU-D with PCT.

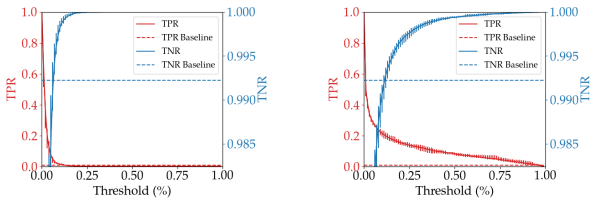
The models for CinC2019 do not make higher confidence predictions, but they already made higher confidence predictions in the baseline compared to PCT, and while the models for PCT benefit from the observation rate, we see that the GRU-D shows a significant benefit.

The significant benefit in GRU-D is also reflected in its DC and TPR-TNR graph, which we show in Figure 12.5.



(a) Baseline DC

(b) Observation rate DC



(c) Baseline TPR-TNR

(d) Observation rate TPR-TNR

Figure 12.5: NB and TPR-TNR of GRU-D with PCT.

Here, we see that the GRU-D has positive NB until a threshold of 0.4, which is an observation that we have not made for any model in any of the other experiments excluding XGBoost, which likewise achieves similar NB as the GRU-D in this experiment.

In the TPR-TNR graph, we observe that the values for the TPR curve are significantly higher at all thresholds compared to the baseline experiment. If we look at the threshold where the TNR curve crosses the TNR baseline for both experiments, then we see that the TPR is significantly higher compared to the baseline. This indicates that the observation rate helps the model find more positive samples. However, we still see a slightly worse TNR curve, although it should be noted that the TNR curve is only shown between 0.985 to 1.0, which can make the effect appear more exaggerated.

Chapter 13

Follow-up Experiment

By analyzing our experiment results, we see that the experiment that resulted in the biggest improvement was the observation rate experiment. In that experiment, the observation rate of the features was extracted from the time series data, and given as separate input to the models. GRU-D was the model that presented the biggest change in performance, with a significant increase in AUPRC. It was also well-calibrated in comparison to the other models in our experiments. However, it did not surpass the AUROC and AUPRC of XGBoost.

One of the differences between our models and XGBoost is that XGBoost uses the delta representation, as we described in Section 2.4. The delta representation can be represented as a vector, and therefore be fed to the network in a similar way as with the previously extracted features. As XGBoost shows decent results with the delta representation, we want to experiment how using the delta representation as an extracted feature, affects the neural networks.

We base this new experiment on our previous observations and therefore use an architecture that incorporates the parts of the experiments that we found beneficial. In the following experiment, the TCN, LSTM, BRITS, and GRU-D are given the observation rate as part of its standard architecture, as it either improved or did not change the results (Section 12.4). This results in the architecture shown in Figure 13.1.

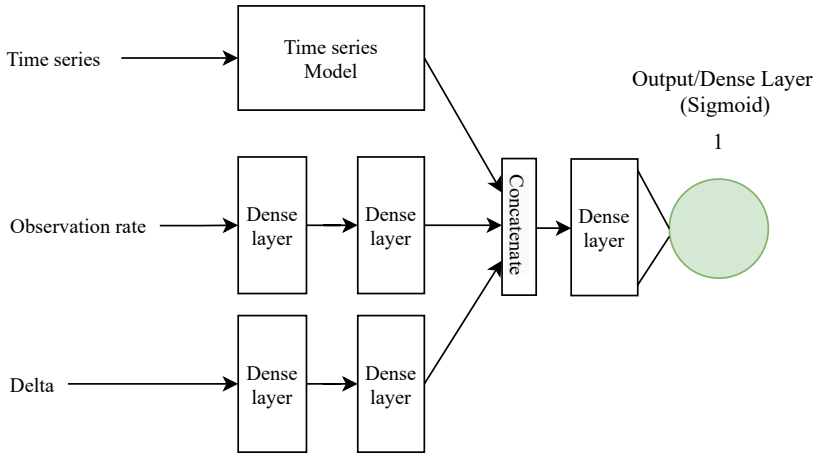


Figure 13.1: Using observation rate and delta as additional input to the network.

Additionally, as the LSTM showed a consistent improvement by using the missingness representation in Section 12.3, the missingness representations are given as input to the LSTM as well.

Due to these additional inputs, we compare the results of the delta representation experiment to the results of the observation rate experiment and the missingness representation for LSTM.

13.1 Delta Representation

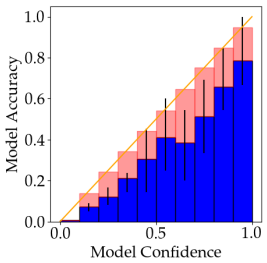
In this section, we describe the delta experiment, where we use the delta representation as an extracted feature.
The most interesting points are:

- For CinC2019, the TCN, LSTM, and GRU-D do not benefit from the delta representation.
- It is inconclusive whether BRITS trained on CinC2019 benefits from the delta representation.
- TCN, LSTM, and BRITS trained on PCT does not benefit from the delta representation.
- The GRU-D trained on PCT is worse at distributing its predictions, with an outlier in the highest confidence prediction, where it has substantially more predictions.

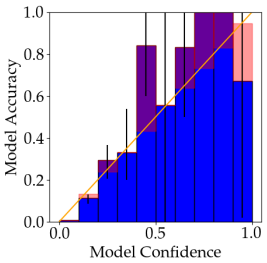
In this experiment, we observe that on CinC2019 the TCN and LSTM do not benefit from the delta representation. The LSTM has worse performance metrics and is less confident, while TCN and GRU-D seem unchanged. For BRITS, we see more high confidence predictions. However, at closer inspection, this only happen in one of the five runs, while the remaining four did not have any high confidence predictions. The high confidence predictions are also overconfident.

For PCT, we observe that the TCN, LSTM, and BRITS do not benefit from the delta representation. These three models have very few predictions with high confidence, and it is, therefore, difficult to interpret their calibration in the reliability diagrams.

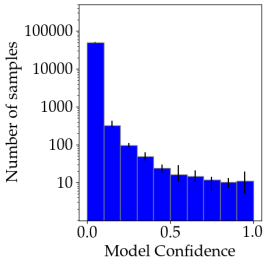
On PCT, the GRU-D model changes significantly in its prediction. In the observation rate experiment, the number of predictions in each bin consistently decreased according to the confidence, as seen in Figure 13.2.



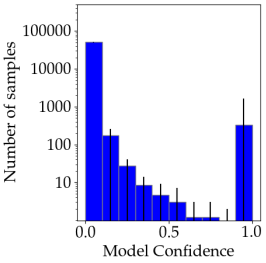
(a) Observation rate reliability diagram.



(b) Delta feature reliability diagram



(c) Observation rate prediction distributions.



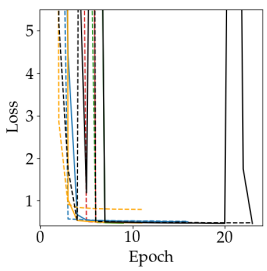
(d) Delta feature prediction distributions.

Figure 13.2: Reliability diagrams and prediction distributions of GRU-D with PCT.

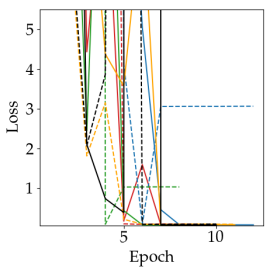
When the GRU-D is given the delta representation as an additional input, the number of predictions in the bins ranging from 0.3 to 0.9 significantly decreases, as seen in Figure 13.2d. However, it also significantly increases the number of predictions in the 0.9 to 1.0 bin. Determining the calibration of the GRU-D model with the delta representation from the reliability diagram (Figure 13.2b) is difficult, as most bins contain few predictions. In addition to this, most bins have large error margins, which show large variation between runs.

AUROC and AUPRC of GRU-D are on average lower with the delta representation. However, both AUROC and AUPRC also have large error margins compared to the observation rate experiment.

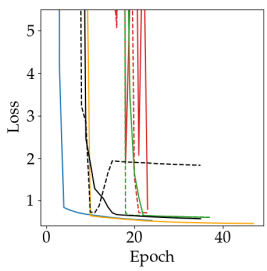
We observe that the delta representation affects training and validation loss of BRITS and GRU-D in an incomprehensible way, for both CinC2019 and PCT (Figure 13.3).



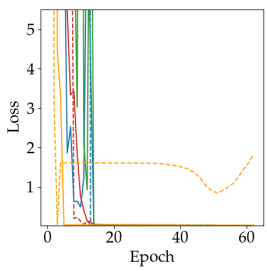
(a) CinC2019 BRITS



(b) CinC2019 GRU-D



(c) PCT BRITS



(d) PCT GRU-D

Figure 13.3: Training and validation loss for BRITS and GRU-D.

Therefore, while there are problems with the delta representation, the large number of samples in the 0.9-1.0 confidence range for GRU-D is an interesting observation. For these reasons, we cannot conclude whether the delta representation helped GRU-D (or the other models), as there could be potential for improvements with further research.

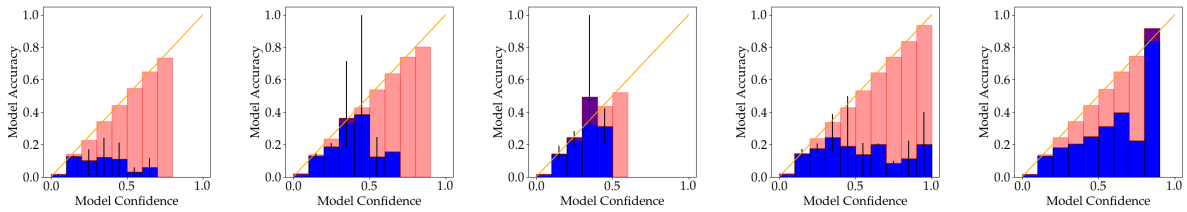
Chapter 14

Final Experiment Results

Our previous experiments have focused on how we could improve upon the baseline experiment on the validation set. In this experiment, we use our findings from the previous experiments to create a set of the best performing and best calibrated models. We then evaluate the performance and calibration of these models on the test set. We found that using the observation rate features gave the best result for the neural networks, and for LSTM we also use the missingness representations. As we did not observe a clear improvement for XGBoost over its baseline in any of the experiments, we decide to use the baseline XGBoost configuration.

To verify that we did not overfit our models to the validation set, we compare the result from this experiment with the results from the observation rate experiment, to see if we see similar results. We see that all models achieve similar results except for a small change in AUPRC, where it has dropped around 0.01 for most models on PCT and increased around 0.01 for most models on CinC2019. Since we see similar performance, we conclude that we have not overfitted our models to the validation set.

On CinC2019 all of the models rarely make predictions with a confidence over 0.9 and are in general overconfident as seen in Figure 14.1.



(a) Reliability diagram for TCN. **(b)** Reliability diagram for LSTM. **(c)** Reliability diagram for BRITS. **(d)** Reliability diagram for GRU-D. **(e)** Reliability diagram for XGBoost.

Figure 14.1: Reliability diagrams for the final experiment.

When looking at the performance metrics in Table 14.1, we see that XGBoost is the best performing model, but the BRITS model is relatively close to its performance on AUPRC.

Final	AUROC	AUPRC
TCN	0.754 ± 0.024	0.081 ± 0.009
LSTM	0.774 ± 0.010	0.098 ± 0.003
BRITS	0.777 ± 0.007	0.113 ± 0.007
GRUD	0.761 ± 0.011	0.093 ± 0.006
XGBoost	0.818 ± 0.000	0.122 ± 0.000

Table 14.1: AUROC and AUPRC for CinC2019.

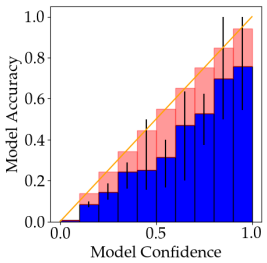
The reliability diagrams for PCT in Figure 14.2a show that GRU-D is close to well-calibrated on all bins, but is a little overconfident. The other models primarily make predictions with less than 0.2 confidence.

Looking at the AUPRC in Table 14.2, we see that it is a lot higher for GRU-D and XGBoost, which is also reflected in the DC graphs in Figure G.29.

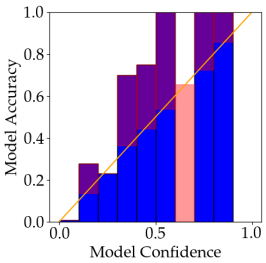
Final	AUROC	AUPRC
TCN	0.751 ± 0.014	0.040 ± 0.014
LSTM	0.743 ± 0.010	0.031 ± 0.006
BRITS	0.748 ± 0.009	0.028 ± 0.001
GRUD	0.730 ± 0.010	0.102 ± 0.008
XGBoost	0.820 ± 0.000	0.131 ± 0.000

Table 14.2: AUROC and AUPRC for PCT.

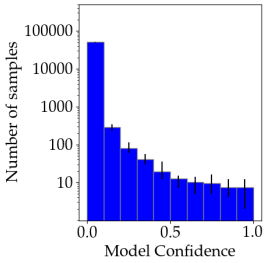
XGBoost has a higher AUPRC than GRU-D on PCT, but XGBoost appears to be worse calibrated. From the reliability diagrams and sample confidence distribution graphs in Figure 14.2, we see that GRU-D makes more high confidence predictions than XGBoost, which is good as long as the confidence of the predictions matches the models' accuracy. We also see that XGBoost is underconfident, but due to the few samples in the bins with higher confidence than 0.4, it is hard to conclude if these bins are well-calibrated.



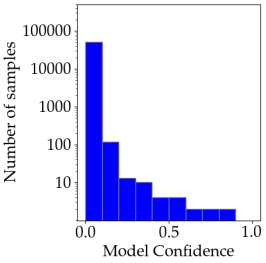
(a) Reliability diagram for GRU-D



(b) Reliability diagram for XGBOOST



(c) Sample confidence distribution graph for GRU-D



(d) Sample confidence distribution graph for XGBOOST

Figure 14.2: Reliability diagrams and sample confidence distribution graphs for GRU-D and XGBoost.

Part IV

Discussion

Chapter 15

Discussion

In this chapter, we discuss the choices we made during this project in the context of the results from our experiments. As we described in Chapter 2, CinC2019 and PCT are both imbalanced datasets with high missing rates. In the following sections, we discuss the CIP and high missing rates in the context of the experiment results.

15.1 The Class Imbalance Problem

From our experiments, we see that the imbalanced data tend to make it very difficult for the models to make high confidence predictions. The class weight experiment we described in Section 12.1 was the primary experiment for attempting to solve the CIP. While we saw that the models made higher confidence predictions, they also became very overconfident, for higher ratios. As we did not see any conclusive performance gains or losses on the neural network models, we suspect that the confidences of the predictions are simply spread more evenly across the confidence ranges in this experiment, and not actually better at predicting the samples. Due to this, we do not consider weighing the classes differently with the method we described in Section 11.1 as a useful solution to the CIP. Despite this, there are still other approaches to using weighted loss functions as those presented by Geng and Luo [18] and Fernando et al. [10], which could be interesting to consider if we had more time.

However, we did see some improvement regarding the CIP with the observation rate experiment we described in Section 12.4. The baselines for PCT generally predict with very low confidences. However, when we introduced the observation rate, TCN, BRITS, and GRU-D predicted a significantly larger number of predictions with higher confidences. This could indicate that the neural network models cannot properly distinguish between the samples, just by using the data in PCT. One reason for this could be that certain information is not available or is too difficult to infer from the data. If we look at the delta experiment we described in Section 13.1, we noted that GRU-D on PCT made the largest number of predictions between 0.9-1.0 that we observed of any models (see Figure G.22d). While adding the delta representation to the input of the GRU-D did not result in a better model, this could support this notion that the models can benefit from additional information.

Due to this, we suspect that providing more useful information to the models can help them overcome the CIP, as they could easier distinguish the positive and negative samples. While our approach of weighing the loss function did not solve the CIP, there are still other

weighted loss function approaches that attempt to solve this problem, such as those presented by Geng and Luo [18] and Fernando et al. [10].

15.2 High Missing Rates

High missing rates were a prominent characteristic for both CinC2019 and PCT as we described and analyzed in Section 2.7. BRITS and GRU-D both attempted to handle the missing values of the data, but with two different approaches. BRITS attempted to impute the data, and then use that data to predict, while GRU-D considered the last observed value along with some decay towards the mean.

If we consider the final experiment we described in Chapter 14, we saw that BRITS had higher AUPRC compared to the other neural network models on CinC2019, while GRU-D had higher AUPRC on PCT. This was also a trend we saw in the observation rate (Section 12.4) and delta representation (Section 13.1) experiments. Considering the missing rates we described in Section 2.7, then the approach of BRITS might be better suited for CinC2019, while the approach of GRU-D might be more suited to PCT. Our reasoning for this is that the higher missing rates of PCT make the data more difficult to impute, as BRITS has less information to base its imputation on. Since the missing rates for CinC2019 are lower, BRITS might be able to more easily create reasonable imputation estimates on this dataset. For GRU-D, since it tries to reduce the information that is carried forward based on the time since the last observation, this might be a better approach for datasets with higher missing rates, like PCT.

If these assumptions are true, it might be beneficial to handle features with different missing rates differently in the models. For example, in CinC2019 and PCT, the laboratory values had higher missing rates than the vital signs. If we apply these assumptions, it could be beneficial to handle the missing values of the vital signs with a more imputation oriented approach, and for the laboratory, values handle them with a "time since last observed"-approach.

As we saw in the missingness representations results we described in Section 12.3, the LSTM appeared to benefit from these representations. While it was inconclusive for the TCN, we argue that this representation is useful for all models working with this type of data. The reason for this is that the representations introduce a way to explicitly represent missing values in the data.

It can be difficult to determine why the TCN did not benefit from the missingness representations the same way the LSTM did. One reason might be that the TCN's first convolution layer is likely to merge measurements with the missingness representation. Instead of merging all the information together at first, the TCN might benefit from analyzing the missingness representations individually with more than one convolution layer and then later combine this information with the time series. This was different from the LSTM, where the missingness representations were available at every timestep, together with the analysis the model has done so far (represented as the hidden state). We argue changing the TCN to address this issue could make it benefit from the missingness representations. While the LSTM and TCN

do not have the functionality to directly handle this information, we still consider it useful information for the models to have. Additionally, more considerations of how to handle these representations in the TCN and LSTM could also be beneficial.

As BRITS and GRU-D both tended to perform better than LSTM and TCN, we consider it important that the models need to incorporate the features of the data into their design. The design choices made for BRITS and GRU-D make sense for handling missing data well, and we argue that this is one of the reasons why they tended to perform better than LSTM and TCN. If we were to experiment further, we would consider model designs that better utilizes extracted information to improve the model's prediction.

15.3 Overfitting

We often observed that the neural network models tended to overfit to the training data. It is difficult to determine why overfitting often occurred, but one reason might be that many of the samples could contain a lot of the same data. While we can not know this for the samples in PCT, it is true for CinC2019 due to the sliding window framing, where only the first and last timestep is different when the window is moved an hour. For CinC2019, the models are trained on many samples that share a lot of the same data, which we suspect could lead to quicker convergence.

Overfitting is a difficult problem to solve that needs to be addressed with further experimentation. Some methods to address this problem already exists, such as dropout [49], weight decay [50], changing the number of neurons in each layer or the number of layers [51].

15.4 Comparison with XGBoost

In this section, we describe how the TCN, LSTM, BRITS, and GRU-D compare to Enversion's XGBoost model. We start by evaluating the models' performance on CinC2019. From our final experiment (Chapter 14), we saw that none of the models outperformed XGBoost in AUROC or AUPRC. BRITS was the model that showed the best performance, shown in Table 14.1.

While we saw low ECE and ACE, the reliability diagrams show that most of the models are either overconfident or do not make many predictions with high confidences. As Figure 14.1c shows, BRITS did not make any predictions above 0.6. While XGBoost appears to be the best calibrated model, all the models are overconfident. We, therefore, argue that XGBoost is superior to the models we have presented on CinC2019. However, XGBoost still has major problems, such as calibration and distribution of confidences.

For PCT, GRU-D was the best performing neural network model as we showed in Table 14.2, but it did still not surpass XGBoost in AUROC or AUPRC. However, here we can see an interesting observation in the calibration and distribution of confidences of predictions on Figure 14.2. GRU-D makes more higher confident predictions than XGBoost, and also appears to be better calibrated. While GRU-D is still overconfident, we still consider the calibration

more desirable than XGBoost, which does not have many predictions in the higher bins. Despite the worse performance metrics for GRU-D compared to XGBoost for PCT, we argue that the better calibration makes GRU-D more desirable in a clinical setting. However, GRU-D still needs further research before we consider it useful for real-world applications.

We see potential in the models we have proposed here, especially BRITS and GRU-D with observation rates. We do not consider any of the models we investigated here ready for use in a real hospital, due to the problem we have described in this chapter. However, we argue that we have found some ideas that benefit the neural network models we tested that attempt to predict sepsis from imbalanced data with high missing rates. In particular, we argue that the observation rates and missingness representations are useful information as input to the model. Therefore, we argue that designing models to handle these ideas as a part of their design is a useful approach. However, further research on how to design models, that can handle this information better, is needed.

Chapter 16

Conclusion

In Chapter 1, we mentioned that there is an increasing amount of research concerning the application of deep learning methods on EHR data, to assist in clinical settings. However, many modern neural networks are not well-calibrated, which is necessary for real-world usage in clinical settings. From Chapter 2, we presented two datasets: PhysioNet Computing in Cardiology Challenge 2019 dataset (CinC2019) and Processed CROSS-TRACKS dataset (PCT), both of which are highly imbalanced, multivariate time series data with high missing rates. We also found that it was important to frame the data to reflect the real-world use case.
Based on this, we created the following problem statement in Chapter 3:

> **How can we create a well-performing, well-calibrated neural network model for predicting sepsis from high missing rate EHR data?**

To answer the problem statement, we searched for inspiration from state of the art model architectures and ideas, particularly ones that could handle the high missing rates of CinC2019 and PCT. From this, we constructed our architectures that incorporated neural networks with inputs of various types. We set up a series of experiments that tested these architectures on two sepsis EHR datasets. In these experiments, we measured the performance and calibration of our models with the AUROC, AUPRC, ECE, ACE, and MCE metrics. In addition to these metrics, we used reliability diagrams, confidence distributions, and decision curves, to better evaluate the performance and calibration of the models. These graphs were useful as they allowed us to identify multiple cases where the models were not well-calibrated, despite their low calibration metrics, as well as helping us determine how well the models were performing.

Based on our results, we conclude that the best neural network models were BRITS and GRU-D with the observation rate as an extracted feature. We come to this conclusion as BRITS performed the best on CinC2019, while GRU-D performed the best on PCT. The TCN and LSTM were not the best performing neural network models on either dataset. However, the LSTM showed that the missingness representations can be beneficial for models, that do not have special design decisions to exploit these representations. Additionally, we conclude that the observation rate as an extracted feature, was useful for our models as it provided information, which was not available otherwise.

Despite BRITS being the best neural network model on CinC2019, we conclude that it was still inferior to Enversion's XGBoost. While XGBoost also suffered from poor calibration on CinC2019, BRITS was also poorly calibrated and had overall worse performance and calibration than XGBoost. In the comparison of GRU-D and Enversion's XGBoost on PCT, we

saw that XGBoost was superior in performance, and GRU-D was superior in calibration. We conclude that GRU-D is more desirable in a clinical setting compared to XGBoost, despite its lower performance.

Finally, we conclude that all the models need to be researched further to be ready for use in a real-world clinical setting.

Chapter 17

Future Works

In this chapter, we discuss ideas on how to continue the work presented in this report.

As we saw the observation rate was a positive addition to our models, we argue that further research on how best to incorporate this in the models is needed. We already explored some ideas in Section 7.6, which could be useful. On top of how to handle the observation rate, it could also be useful to consider more extracted features. As we saw in the delta experiment (Section 13.1), the GRU-D model on PCT had a significant number of predictions with a confidence between 0.9 and 1.0, despite performing poorly. This could mean that the delta representation as an extracted feature could be beneficial to the neural network models and needs further research to figure out the best way to handle the delta values in the models.

As we discussed in Chapter 15, the TCN did not benefit from the missingness representations. We suspect the reasoning for this is that only the first convolution layer has access to the missingness data before it is merged with the time series data. One solution to this could be to have three TCNs, two for each of the missingness representations and one for the time series data, and then join their output when making a prediction. This way the model could analyze the missingness representations without merging them with the time series data, yet still utilize the missingness representations in the context of the time series data. BRITS and GRU-D are the only two approaches we considered that were designed to handle the missingness representations. Therefore, it could be worth researching other model designs, that handle these missingness representations differently.

The CIP is something that also needs to be addressed with datasets like CinC2019 and PCT. We already proposed using other weighted loss functions in Section 15.1 to mitigate the CIP, but different approaches could also be considered. As we discussed in Section 15.1, the problem might be that the samples are too similar and it is too difficult for the models to distinguish them. Therefore, extracting other useful features to distinguish the samples could also be a promising approach to solve the CIP.

Bibliography

[1] M. A. Reyna, C. S. Josef, R. Jeter, S. P. Shashikumar, M. B. Westover, S. Nemati, G. D. Clifford, and A. Sharma, "Early prediction of sepsis from clinical data: The physionet/computing in cardiology challenge," *Critical Care Medicine*, vol. 48, no. 2, p. 210–217, February 2020.

[2] M. H. Svendsen, M. Simonsen, S. D. Nielsen, M. N. Stenkær, L. Østergaard, and C. K. Frydkjær, "Analyzing calibration of state of the art deep learning architectures for electronic health records," January 2020.

[3] B. Shickel, P. J. Tighe, A. Bihorac, and P. Rashidi, "Deep ehr: A survey of recent advances in deep learning techniques for electronic health record (ehr) analysis," *IEEE Journal of Biomedical and Health Informatics*, vol. 22, no. 5, p. 1589–1604, Sep 2018. [Online]. Available: http://dx.doi.org/10.1109/JBHI.2017.2767063

[4] J. Fagerström, M. Bång, D. Wilhelms, and M. S. Chew, "Lisep lstm: A machine learning algorithm for early detection of septic shock," *Scientific Reports*, vol. 9, no. 1, 2019.

[5] S. Lauritsen, M. Kalør, E. Kongsgaard, K. Lauritsen, M. Jørgensen, J. Lange, and B. Thiesson, "Early detection of sepsis utilizing deep learning on electronic health record event sequences," *Artificial Intelligence in Medicine*, vol. 104, April 2020.

[6] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *CoRR*, vol. abs/1706.04599, December 2017. [Online]. Available: http://arxiv.org/abs/1706.04599

[7] M. Reyna, C. Josef, R. Jeter, S. Shashikumar, B. Moody, M. B. Westover, A. Sharma, S. Nemati, and G. Clifford, "Early prediction of sepsis from clinical data – the physionet computing in cardiology challenge 2019," August 2019. [Online]. Available: https://physionet.org/content/challenge-2019/1.0.0/

[8] September 2020. [Online]. Available: https://www.tvaerspor.dk/

[9] S. M. Lauritsen, B. Thiesson, M. J. Jørgensen, A. H. Riis, U. S. Espelund, J. B. Weile, and J. Lange, "The consequences of the framing of machine learning risk prediction models: Evaluation of sepsis in general wards," January 2021.

[10] K. R. M. Fernando and C. P. Tsokos, "Dynamically weighted balanced loss: Class imbalanced learning and confidence calibration of deep neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, January 2021.

[11] H. He and E. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, p. 1263–1284, September 2009.

[12] C. Sun, S. Hong, M. Song, and H. Li, "A review of deep learning methods for irregularly sampled medical time series data," October 2020.

[13] A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P. Ivanov, R. Mark, J. Mietus, G. Moody, C. Peng, and H. Stanley, "Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012," *Artificial Intelligence Review*, pp. 215–220, 2012.

[14] S. N. Shukla and B. M. Marlin, "Interpolation-prediction networks for irregularly sampled time series," *arXiv preprint arXiv:1909.07782*, September 2019.

[15] M. Horn, M. Moor, C. Bock, B. Rieck, and K. Borgwardt, "Set functions for time series," in *International Conference on Machine Learning*. PMLR, September 2020, pp. 4353–4363.

[16] Q. Tan, M. Ye, B. Yang, S. Liu, A. J. Ma, T. C.-F. Yip, G. L.-H. Wong, and P. Yuen, "Data-gru: Dual-attention time-aware gated recurrent unit for irregular multivariate time series," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, p. 930–937, April 2020.

[17] P. Li, M. Abdel-Aty, and J. Yuan, "Real-time crash risk prediction on arterials based on lstm-cnn," *Accident Analysis & Prevention*, vol. 135, p. 105371, Febuary 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0001457519311108

[18] Y. Geng and X. Luo, "Cost-sensitive convolutional neural networks for imbalanced time series classification," *Intelligent Data Analysis*, vol. 23, no. 2, p. 357–370, January 2019.

[19] J. Singh, K. Oshiro, R. Krishnan, M. Sato, T. Ohkuma, and N. Kato, "Utilizing informative missingness for early prediction of sepsis," *2019 Computing in Cardiology Conference (CinC)*, September 2019.

[20] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific Reports*, vol. 8, no. 1, April 2018.

[21] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, and Y. Li, "Brits: Bidirectional recurrent imputation for time series," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/734e6bfcd358e25ac1db0a4241b95651-Paper.pdf

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[23] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, November 2012. [Online]. Available: http://arxiv.org/abs/1211.5063

[24] C. Olah, "Understanding lstm networks," 2015, [Accessed 03-12-2020]. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[25] 2020, [Accessed 13-11-2020]. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

[26] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[27] W.-C. Lin and C.-F. Tsai, "Missing value imputation: a review and analysis of the literature (2006–2017)," *Artificial Intelligence Review*, vol. 53, no. 2, p. 1487–1509, 2019.

[28] A. R. T. Donders, G. J. van der Heijden, T. Stijnen, and K. G. Moons, "Review: A gentle introduction to imputation of missing values," *Journal of Clinical Epidemiology*, vol. 59, no. 10, pp. 1087–1091, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0895435606001971

[29] J. M. Lachin, "Fallacies of last observation carried forward analyses," *Clinical Trials*, vol. 13, no. 2, p. 161–168, 2015.

[30] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.

[31] Z. Yuan, Y. Jiang, J. Li, and H. Huang, "Hybrid-dnns: Hybrid deep neural networks for mixed inputs," May 2020.

[32] K. Hajian-Tilaki, "Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation," *Caspian journal of internal medicine*, vol. 4, pp. 627–635, 09 2013.

[33] B. Ozenne, F. Subtil, and D. Maucort-Boulch, "The precision–recall curve overcame the optimism of the receiver operating characteristic curve in rare diseases," *Journal of Clinical Epidemiology*, vol. 68, no. 8, pp. 855 – 859, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0895435615001067

[34] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[35] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PloS one*, vol. 10, p. e0118432, 03 2015.

[36] A. J. Vickers and E. B. Elkin, "Decision curve analysis: A novel method for evaluating prediction models," *Medical Decision Making*, vol. 26, no. 6, p. 565–574, December 2006.

[37] J. Nixon, M. Dusenberry, L. Zhang, G. Jerfel, and D. Tran, "Measuring calibration in deep learning," *CoRR*, vol. abs/1904.01685, April 2019. [Online]. Available: http://arxiv.org/abs/1904.01685

[38] M. Pakdaman Naeini, G. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using bayesian binning," *Proceedings of the ... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, vol. 2015, pp. 2901–2907, April 2015.

[39] L. Huang, J. Zhao, B. Zhu, H. Chen, and S. V. Broucke, "An experimental investigation of calibration techniques for imbalanced data," *IEEE Access*, vol. 8, pp. 127 343–127 352, July 2020.

[40] B. Van Calster and A. J. Vickers, "Calibration of risk prediction models: impact on decision-analytic performance," *Medical decision making : an international journal of the Society for Medical Decision Making*, vol. 35, no. 2, p. 162—169, February 2015. [Online]. Available: https://doi.org/10.1177/0272989X14547233

[41] "Tensorflow," [Accessed 17-5-2021]. [Online]. Available: https://www.tensorflow.org/versions/r2.4/api_docs/python/tf

[42] "Xgboost documentation," [Accessed 18-5-2021]. [Online]. Available: https://xgboost.readthedocs.io/en/latest/

[43] D. Poole and A. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*, 2nd ed. Cambridge, UK: Cambridge University Press, 2017. [Online]. Available: http://artint.info/2e/html/ArtInt2e.html

[44] L. Prechelt, *Early Stopping - But When?* Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 55–69. [Online]. Available: https://doi.org/10.1007/3-540-49430-8_3

[45] TensorFlow, "tf.keras.callbacks.earlystopping," [Accessed 03-06-2021]. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

[46] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, December 2014.

[47] XGBoost, "Xgboost parameters," [Accessed 03-06-2021]. [Online]. Available: https://xgboost.readthedocs.io/en/latest/parameter.html#parameters-for-tree-booster

[48] TensorFlow, "Classification on imbalanced data," [Accessed 26-5-2021]. [Online]. Available: https://www.tensorflow.org/tutorials/structured_data/imbalanced_data#class_weights

[49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[50] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[51] H. Hippert, D. Bunn, and R. Souza, "Large neural networks for electricity load forecasting: Are they overfitted?" *International Journal of forecasting*, vol. 21, no. 3, pp. 425–434, 2005.

[52] F. Gul, M. K. Arslantas, I. Cinel, and A. Kumar, "Changing definitions of sepsis," June 2017, [Accessed 20-10-2020]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5512390/

[53] C. Nickson, "Pao2/fio2 ratio (p/f ratio)," April 2020, [Accessed 20-10-2020]. [Online]. Available: https://litfl.com/pao2-fio2-ratio/

[54] U. I. Services, "Basic laboratory tests complete blood counts (cbc)," 2015, [Accessed 20-10-2020]. [Online]. Available: https://www.uiservices.com/wp-content/uploads/2015/01/CompleteBloodCounts.pdf

[55] WebMD, "Understanding low blood pressure – the basics," 2019, [Accessed 20-10-2020]. [Online]. Available: https://www.webmd.com/heart/understanding-low-blood-pressure-basics#1

[56] W. M. Reference, "What is a bilirubin test?" 2019, [Accessed 20-10-2020]. [Online]. Available: https://www.webmd.com/a-to-z-guides/bilirubin-test#1

[57] N. K. Foundation, "Creatinine: What is it?" 2017, [Accessed 20-10-2020]. [Online]. Available: https://www.kidney.org/atoz/content/what-creatinine

[58] BrainLine, "What is the glasgow coma scale?" 2018, [Accessed 20-10-2020]. [Online]. Available: https://www.brainline.org/article/what-glasgow-coma-scale

[59] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[60] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," *CoRR*, vol. abs/1611.05267, November 2016. [Online]. Available: http://arxiv.org/abs/1611.05267

[61] Z. Zhang, "Derivation of backpropagation in convolutional neural network (cnn)," *University of Tennessee, Knoxville, TN*, October 2016.

[62] J. Bouvrie, "Notes on convolutional neural networks," November 2006.

[63] L. Boué, "Deep learning for pedestrians: backpropagation in cnns," *arXiv preprint arXiv:1811.11987*, November 2018.

[64] S. N. Shukla and B. M. Marlin, "Interpolation-prediction networks for irregularly sampled time series," 2019.

[65] "Papers with code - physionet challenge 2012 benchmark (time series classification)," [Accessed 20-04-2021]. [Online]. Available: https://paperswithcode.com/sota/time-series-classification-on-physionet

[66] X. Tang, H. Yao, Y. Sun, C. Aggarwal, P. Mitra, and S. Wang, "Joint modeling of local and global temporal dynamics for multivariate time series forecasting with missing values," November 2019.

[67] J. Yoon, W. R. Zame, and M. van der Schaar, "Estimating missing data in temporal data streams using multi-directional recurrent neural networks," November 2017.

[68] S. Yang, M. Dong, Y. Wang, and C. Xu, "Adversarial recurrent time series imputation," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, August 2020.

[69] Q. Suo, W. Zhong, G. Xun, J. Sun, C. Chen, and A. Zhang, "Glima: Global and local time series imputation with multi-directional attention learning," in *2020 IEEE International Conference on Big Data (Big Data)*, December 2020, pp. 798–807.

[70] Y. Lee, E. Jun, and H.-I. Suk, "Multi-view integration learning for irregularly-sampled clinical time series," January 2021.

[71] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, and Y. Li, "Code of nips18 paper: Brits: Bidirectional recurrent imputation for time series," [Accessed 03-06-2020]. [Online]. Available: https://github.com/caow13/BRITS

Part V

Appendix

Appendix A

Feature Description of Datasets [1]

Feature number	Measurement
1	Temperature
2	SpO2
3	Heart rate
4	Diastolic BP
5	Respiratory Frequency
6	Systolic BP
7	B-Leukocytes
8	B-Neutrophils
9	B-Platelets
10	eGFR
11	P-Albumin
12	P-Bilirubine
13	P-C-reactive protein
14	P-Glucose
15	P-Potassium
16	P-Creatinine
17	P-Sodium
18	P(aB)-Hydrogen carbonate
19	P(aB)-Potassium
20	P(aB)-Chloride
21	P(aB)-Lactate
22	P(aB)-Sodium
23	P(aB)-pCO2
24	P(aB)-pH
25	P(aB)-pO2

Table A.1: Features of dataset prepared by Enversion from Cross-Track's dataset [8].

[1] This appendix is a slightly modified version of Appendix C from our previous semester report [2].

Feature number	Measurement	Description
Vital signs (Feature 1-8)		
1	HR	Heart rate (beats per minute)
2	O2Sat	Pulse oximetry (%)
3	Temp	Temperature (Deg C)
4	SBP	Systolic BP (mm Hg)
5	MAP	Mean arterial pressure (mm Hg)
6	DBP	Diastolic BP (mm Hg)
7	Resp	Respiration rate (breaths per minute)
8	EtCO2	End-tidal carbon dioxide (mm Hg)
Laboratory values (Feature 9-34)		
9	BaseExcess	Measure of excess bicarbonate (mmol/L)
10	HCO3	Bicarbonate (mmol/L)
11	FiO2	Fraction of inspired oxygen (%)
12	pH	N/A
13	PaCO2	Partial pressure of carbon dioxide from arterial blood (mm Hg)
14	SaO2	Oxygen saturation from arterial blood (%)
15	AST	Aspartate transaminase (IU/L)
16	BUN	Blood urea nitrogen (mg/dL)
17	Alkalinephos	Alkaline phosphatase (IU/L)
18	Calcium	(mg/dL)
19	Chloride	(mmol/L)
20	Creatinine	(mg/dL)
21	Bilirubin_direct	Bilirubin direct (mg/dL)
22	Glucose	Serum glucose (mg/dL)
23	Lactate	Lactic acid (mg/dL)
24	Magnesium	(mmol/dL)
25	Phosphate	(mg/dL)
26	Potassium	(mmol/L)
27	Bilirubin_total	Total bilirubin (mg/dL)
28	TroponinI	Troponin I (ng/mL)
29	Hct	Hematocrit (%)
30	Hgb	Hemoglobin (g/dL)
31	PTT	partial thromboplastin time (seconds)
32	WBC	Leukocyte count ($count \cdot 10^3/\mu L$)
33	Fibrinogen	(mg/dL)
34	Platelets	($count \cdot 10^3/\mu L$)
Demographics (Feature 35-40)		

35	Age	Years (100 for patients 90 or above)
36	Gender	Female (0) or Male (1)
37	Unit1	Administrative identifier for ICU unit (MICU)
38	Unit2	Administrative identifier for ICU unit (SICU)
39	HospAdmTime	Hours between hospital admit and ICU admit
40	ICULOS	ICU length-of-admission (hours since ICU admit)

Table A.2: Features of dataset from PhysioNet [7].

Appendix B

Sepsis [1]

In this section, we describe the disease sepsis, and how it is diagnosed, based on the article [52]. Sepsis is a severe medical condition, where damage to tissue and organs is caused by the immune system's response to an infection. If bacteria from an infection spreads to the bloodstreams, it might spread to other organs, which can be fatal. The immune system reacts to the infection by lowering the blood pressure and thereby slowing down the blood flow, making it harder for the bacteria to reach the organs. However, this reduces the oxygen flow to the organs, which can cause them to fail, resulting in organ and tissue damage.

Sepsis is described as a life-threatening disease, as the mortality rate ranges between $\sim$ 20% to $\sim$ 40%, and is one of the most common causes of death in intensive care units (ICUs). The number of sepsis cases has doubled over the last 10 years, however, the overall mortality rate has decreased due to advancements within health care. Early detection of sepsis can be difficult, as the signs of the disease can be divergent.

As of the time of writing, there have been a total of three sepsis definitions. With the sepsis-3 definition, the old criteria were replaced with a new system, Sequential Organ Failure Assessment (SOFA). Now, sepsis is diagnosed with the SOFA scoring system, which evaluates the condition of six organ systems. This score can be used to determine whether any of the organ systems have reduced functionality, which may indicate that the patient has sepsis. The six organ systems that the SOFA scoring system evaluate are:

- **Respiratory System**: By measuring the partial pressure of oxygen in the arterial blood [53].
- **Coagulation**: By measuring the number of platelets in the blood. A higher number of platelets results in a higher chance of blood clotting [54].
- **Cardiovascular**: By measuring hypotension, which is a low systolic blood pressure [55].
- **Liver**: By measuring the amount of bilirubin present in the liver. A high amount of bilirubin is a sign of diseases [56].
- **Renal**: By measuring the amount of creatinine in the renal or urine output of the patient. Creatinine is a waste product in the blood, and is the result of muscle attrition [57].
- **Central Nervous System**: By measuring disruptions in brain function with the Glasgow Coma Scale (GCS). GCS analyses the mental status of the patient through a set of criteria and assign points to the patient according to their brain functions. A lower GCS score signifies less consciousness in the patient [58].

[1] This appendix is a slightly modified version of Appendix A from our previous semester report [2].

A score of 0 to 4 can be assigned to each of the six evaluated organ systems, meaning that the total score can be between 0 and 24. If a patient experiences symptoms that result in a SOFA score with an increase of two, the patient is diagnosed with sepsis. The number of points can reflect the mortality of the patient, as more points can indicate a more severe case of sepsis. For example, with a score ranging between 0 and 6, the mortality of the patient can be expected to be less than 10%, and with a score above 15, the expectation is 90%.

Appendix C

Theory

C.1 Artificial Neural Networks [1]

In this section, we give an introduction to artificial neural networks, based on information from [22] and [59]. Artificial neural networks are a group of algorithms inspired by the biological neural network, like the human brain. Throughout this report, we refer to artificial neural networks as neural networks.

A neural network consists of an input layer, a number of hidden layers, and an output layer, each containing a number of neurons. Between two consecutive layers, a number of weights connect the neurons in layer $l-1$ to neurons in layer l. The weights are denoted w_{jk}^l, where j is the index of the neuron in layer l and k is the index of the neuron in layer $l-1$. Each neuron contains an internal value, that we denote z_j^l, and an output, denoted a_j^l. a_j^l is the output of an activation function, σ, with input z_j^l, as shown in Equation C.1b. z_j^l is calculated as the sum of products of all weights connected to the neuron and the output of the activation function for the corresponding neuron from the previous layer, a_k^{l-1}, shown in Equation C.1a. An activation function is used to make a neural network approximate nonlinear functions. Without an activation function, a neural network is only able to approximate linear functions, which is undesirable in many cases. An example of a nonlinear activation function is sigmoid $\left(\frac{1}{1+exp(-x)}\right)$, which produces outputs between 0 and 1. Additionally, neural networks use biases, which are used to shift the function of the neural network.

$$z_j^l = \sum_k w_{jk}^l \cdot a_k^{l-1} \tag{C.1a}$$

$$a_j^l = \sigma(z_j^l) \tag{C.1b}$$

In the following description, the bias is augmented as an extra activation from the previous layer, always having a value of 1, which is analogous to having a bias neuron with an output of 1, connected with weights to all neurons in a layer. Additionally, we will only be using σ to symbolize an activation function, although activation functions can be different between layers.

Instead of representing the activations from a layer as individual scalars, the activations can also be represented in the form of a vector, $\boldsymbol{a}^l$, where the elements are each activation from the layer. The weights for a layer l can be represented as a matrix $\boldsymbol{W}^l$, where index $\boldsymbol{W}^l[2,3]$

[1] This section is a slightly modified version of Section 4.1 from our previous semester report [2].

refers to w_{23}^l. With this representation, we can calculate the activations of a layer $\boldsymbol{a}^l$ using matrix multiplication as shown in Equation C.2:

$$\boldsymbol{z}^l = \boldsymbol{W}^l \cdot \boldsymbol{a}^{l-1} \tag{C.2a}$$

$$\boldsymbol{a}^l = \sigma(\boldsymbol{z}^l) \tag{C.2b}$$

These equations encapsulate the forward propagation of the neural network, from input to output.

C.1.1 Training a Neural Network

In this section, we explain how a neural network is trained. The basic principle of training a neural network is to update the parameters of the network based on the gradient of a loss function, C, given a set of input-output pairs. C is a function that measures the correctness of the prediction in relation to the ground truth labels. This measure is also known as the loss or loss. One example of a loss function is cross-entropy, which can be used when doing classification. In our case, we do binary classification when classifying whether a person gets sepsis or not.

$$C = -\frac{1}{|D|} \sum_{(x,y)\in D} (y \cdot \log(\hat{y}(x)) + (1-y) \cdot \log(1-\hat{y}(x))) \tag{C.3}$$

Equation C.3 calculates the loss for a neural network, in relation to a dataset D, where D contains the input-output pairs (x, y). $\hat{y}(x)$ is the output of the activation function of the output layer in the network, a^L.

C.1.1.1 Backpropagation

The purpose of backpropagation is to calculate the gradients of the loss function with respect to the parameters of the neural network.

The gradients of the loss function are calculated as the partial derivatives of C with respect to each weight in $\boldsymbol{W}$, denoted as $\frac{\partial C}{\partial w_{jk}^l}$. Using the chain rule, this can be rewritten as shown in Equation C.4, where the partial derivatives on the right-hand side each represent a backward step in the backpropagation.

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} \tag{C.4}$$

As $\frac{\partial C}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l}$ is used for calculating the gradients with respect to all weights to neuron j in layer l, we introduce δ_j^l as an error term such that $\delta_j^l = \frac{\partial C}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l}$.

If we introduce $\nabla_{\boldsymbol{a}} \boldsymbol{C}$ as a vector of partial derivatives of C with respect to each activation in the output layer, shown in Equation C.5a, then the error term of the output layer can be calculated as shown in Equation C.9a. σ' is the derivative of the activation function with

respect to the inputs to that activation function, z_j^L, such that the output of σ' is the vector shown in Equation C.5b.

$$\boldsymbol{\nabla}_{\boldsymbol{a}}\boldsymbol{C} = \begin{pmatrix} \frac{\partial C}{\partial a_1^L} \\ \vdots \\ \frac{\partial C}{\partial a_{|a^L|}^L} \end{pmatrix} \tag{C.5a}$$

$$\sigma'(\boldsymbol{z}^L) = \begin{pmatrix} \frac{\partial a_1^L}{\partial z_1^L} \\ \vdots \\ \frac{\partial a_{|a^L|}^L}{\partial z_{|a^L|}^L} \end{pmatrix} \tag{C.5b}$$

The error term of the output layer is used to calculate the error term of the preceding layer, using Equation C.9b, which is in turn used to further propagate the error term backward through the network. This means that the error term has to include these extra backward steps, which can be seen in the expansion of the chain rule (Equation C.4), shown in Equation C.6.

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_i \left(\frac{\partial C}{\partial a_i^{l+1}} \cdot \frac{\partial a_i^{l+1}}{\partial z_i^{l+1}} \cdot \frac{\partial z_i^{l+1}}{\partial z_j^l} \right) \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} \tag{C.6}$$

The expansion introduces the term $\frac{\partial z_i^{l+1}}{\partial z_j^l}$ shown in Equation C.7, where the right hand side is derived from Equation C.1.

$$\frac{\partial z_i^{l+1}}{\partial z_j^l} = w_{ij}^{l+1} \cdot \sigma'(z_j^l) \tag{C.7}$$

In Equation C.8, the partial derivatives inside the summation in Equation C.6 is substituted by the error term from the following layer and Equation C.7.

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_i \left(\delta_i^{l+1} \cdot w_{ij}^{l+1} \cdot \sigma'(z_j^l) \right) \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} \tag{C.8}$$

The summation in Equation C.6 can be expressed in the form of matrix-vector multiplication, calculating all error terms for a layer, as shown in Equation C.9b.

$$\boldsymbol{\delta}^L = \boldsymbol{\nabla}_{\boldsymbol{a}}\boldsymbol{C} \odot \sigma'(\boldsymbol{z}^L) = \begin{pmatrix} \frac{\partial C}{\partial z_1^L} \\ \vdots \\ \frac{\partial C}{\partial z_{|a^L|}^L} \end{pmatrix} = \begin{pmatrix} \delta_1^L \\ \vdots \\ \delta_{|a^L|}^L \end{pmatrix} \tag{C.9a}$$

$$\boldsymbol{\delta}^l = ((\boldsymbol{W}^{l+1})^T \cdot \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{z}^l) = \begin{pmatrix} \frac{\partial C}{\partial z_1^l} \\ \vdots \\ \frac{\partial C}{\partial z_{|a^l|}^l} \end{pmatrix} = \begin{pmatrix} \delta_1^l \\ \vdots \\ \delta_{|a^l|}^l \end{pmatrix} \tag{C.9b}$$

Now that the error terms are introduced, $\frac{\partial z_j^l}{\partial w_{jk}^l}$ from Equation C.4 and Equation C.6 is the only additional term that needs to be considered. $\frac{\partial z_j^l}{\partial w_{jk}^l}$ is calculated as the derivative of Equation C.1a with respect to w_{jk}^l, which has the result: a_k^{l-1}. Therefore, the gradients of C with respect to each parameter in the neural network are calculated as shown in Equation C.10.

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l \cdot a_k^{l-1} \tag{C.10}$$

C.1.1.2 Optimization

When the gradients have been computed, they can be used to update the parameters of the neural network. One way of doing this is through gradient descent, which updates the parameters along their gradients towards a local optimum for the loss function. Equation C.11 calculates the delta values for each weight, which is how much the weights are increased or decreased in the optimization step.

$$\Delta w_{jk}^l = -\mu \cdot \frac{1}{m} \cdot \sum_i^m \frac{\partial C_{x_i}}{\partial w_{jk}^l} \tag{C.11}$$

Here, μ is the learning rate, which is a small positive real number that adjusts how much the parameters of the neural network are updated in a single training step. The delta values are calculated as the learning rate multiplied by the average of the gradients for data samples x_i. If we consider *simple gradient descend*, the gradients are calculated for the entire dataset, meaning that m denotes the total number of data samples in Equation C.11. This can be time consuming for large datasets, but can be improved by using *stochastic gradient descend*. In stochastic gradient descend, a randomly selected batch of training samples are considered at a time, and the average gradient for the training samples are calculated. In this case, m in Equation C.11 denotes the number of samples in the batch.

Choosing the value of the learning rate is important for the training of the network. A high learning rate makes larger changes to the parameters and thus converges faster, whereas a low learning rate makes smaller changes and is better at fine-tuning parameters. Therefore, it is advantageous to have a high learning rate early in the training process, to converge faster, and then change to a low learning rate later in the training process, to fine-tune parameters closer to the optimum. *Adam* is a method that uses this concept of adaptive learning rate, which finds individual learning rates for updating different parameters in the network.

C.2 Convolutional Neural Networks [2]

In this section, we describe convolutional neural networks (CNNs) based on information from [59]. The neural networks we describe in Section C.1 have layers, where every neuron in that layer is connected to every neuron in the previous layer. This means that every neuron in layer

[2] This section is a slightly modified version of Section 4.3 from our previous semester report [2].

l, considers every activation from layer $l-1$, which might not be preferable for some types of data. For example, the data for a patient in CinC2019 and PCT is represented as a number of features for a series of timesteps. If we consider a neural network with $features \cdot timesteps$ neurons in its input layer, then the neurons in the hidden layer will consider every feature for every timestep. The neurons ignore the temporal structure of the data, as it treats data from early in the admission on the same basis as data in the final part of admission. The temporal structure then has to be inferred from the data by the network. CNNs provide an architecture, which, among others, tackle this issue, by introducing three ideas: local receptive fields, shared weights, and pooling.

C.2.1 Local Receptive Fields

As opposed to a fully connected neural network, a neuron in a convolutional layer l is only connected to some of the neurons in layer $l-1$. Since we can consider the data we work with to be two-dimensional, one axis for time and one axis for features, we can consider the input as a grid of neurons of size $h \times i$, where h is the number of timesteps and i is the number of features. Considering layer $l-1$ as a grid of neurons is useful for defining *how* the neurons in layer l are connected to neurons in $l-1$. Since we want to encode the temporal structure of our data in the network architecture, each neuron in layer l is connected (by weights) to a region of neurons in layer $l-1$. This region of neurons is called the local receptive field (LRF) for that convolutional neuron.

In our case, we want the LRFs for the neurons in layer l to be the neurons for t timesteps. The LRF for the first neuron in layer l contains the neurons from position $(1,1)$ to (t,i) from layer $l-1$, the second contains the neurons from position $(2,1)$ to $(t+1,i)$, and so forth. More concisely, Equation C.12 defines the LRF for neuron j in layer l (LRF_j^l).

$$LRF_j^l = \{(j+0,1),\ldots,(j+t-1,1),(j+0,2),\ldots,(j+t-1,i)\}, \tag{C.12}$$

The position on the form (x,y) denotes the neuron from layer $l-1$ at position (x,y), when representing layer $l-1$ as a grid.

An alternative perspective is to consider the LRFs for the neurons in layer l to be moved one value at a time along the time axis, which is illustrated in Figure C.1.

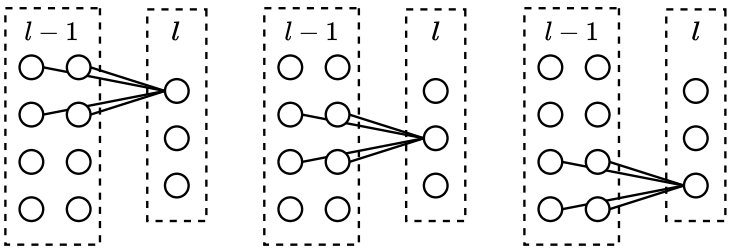


Figure C.1: The LRF for the three neurons in layer l.

In this case, we slide the LRFs by one, but it is also possible to slide them by any value. This value is called the stride. Figure C.2 shows an example where the stride $s = 2$.

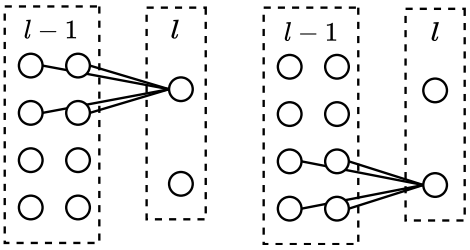


Figure C.2: The LRF for the two neurons in layer l, with a stride $s = 2$.

The LRF definition can be expanded to account for stride as shown in Equation C.13.

$$LRF_j^l = \Big\{\Big((j-1)\cdot s+1,1\Big),\ldots,\Big((j-1)\cdot s+t,1\Big),\Big((j-1)\cdot s+1,2\Big),\ldots,\Big((j-1)\cdot s+t,i\Big)\Big\} \tag{C.13}$$

Consequently, layer l will have $\lceil\frac{h-t+1}{s}\rceil$ neurons, as the stride affects how many neurons are necessary to create the LRFs.

C.2.2 Shared Weights & Biases

CNNs use shared weights and biases, which means that some neurons share the same weights. Equation C.14 shows the output of the jth neuron in a convolutional layer.

$$a_j^l = \sigma(b^l + \sum_{m=1}^{t}\sum_{n=1}^{i} w_{m,n}^l \cdot a_{j+m,n}^{l-1}) \tag{C.14}$$

σ is the activation function, b^l is the shared weight to the bias neuron, w^l is the shared weights arranged in an array of size $t \times i$, and $a_{x,y}^{l-1}$ is the activation from the previous layer at position x, y. Since the weights are shared, we can say one layer finds one "feature" across the entire input. This is useful, as one pattern (for example, an increase in heart rate) can be useful regardless of its position in the sequence. This makes the convolution layer translation invariant, as it does not matter where in the sequence the pattern is found.

The map from the input layer to the hidden layer is often called a feature map. The shared weights and bias are often called the kernel or filter. Using shared weights also reduces the number of weights to the size of the kernel.

So far, we have only described a convolutional layer with one feature map, but a convolutional layer will often have many feature maps. Multiple feature maps can be implemented by adding more neurons to the convolutional layer with the same LRFs, but using other kernels. Therefore, the output can be considered two-dimensional, with the (convoluted) timesteps on one axis and the feature maps for each kernel on the other.

C.2.3 Pooling Layers

Pooling layers are typically placed after a convolution layer and are used to condense the output of the feature maps. A pooling layer consists of pooling units, and, similarly to convolutional layers, each unit has its own LRF. As opposed to convolutional neurons, a pooling

unit's LRF only contains neurons from one feature map from the previous layer. A pooling layer attempts to summarize all the feature maps from the previous layer. While many types of pooling exist, we describe max pooling, as it is commonly used. The activation of a max pooling unit is simply the highest activation from its LRF.

Figure C.3 shows max pooling on an input layer of 4 timesteps with 2 features. The max pooling layer has two max pooling units with a pooling size of 2 and a stride of $s = 2$. We can see the values 3 and 7 being pooled from the first feature map and 4 and 8 for the second feature map.

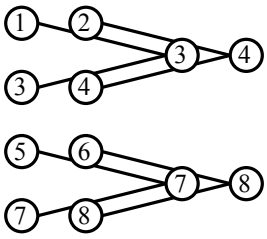


Figure C.3: Max pooling with a pooling size of 2 and stride $s = 2$.

C.2.4 Padding

Padding allows the output of a convolutional or pooling layer to be calculated for positions where the kernel or filter would otherwise include out of bounds values. As previously mentioned, a convolutional layer with one kernel requires $\lceil \frac{h-t+1}{s} \rceil$ neurons, assuming the input layer is $h \times i$. We describe a padding method often referred to as "same" padding, where we add neurons to the previous layer such that our convolutional layer has $\lceil \frac{h}{s} \rceil$ neurons.

Figure C.4 shows an example of padding a layer $l-1$ such that the number of neurons in layer l is 4 instead of 2. We can see that same padding places padding neurons both at the beginning and the end of the feature maps.

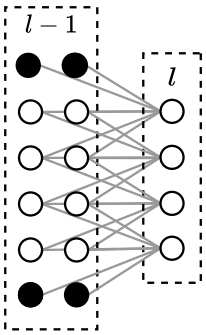


Figure C.4: A layer $l-1$ with 4×2 neurons, with same padding. Layer l has one kernel and $t = 3$.

C.2.4.1 Causal Padding

Another type of padding is causal padding. Causal padding works similarly to same padding, but instead of placing the padding neurons at the edges of the feature maps, it simply places

all of them in the first indexes. This is useful for maintaining the causal ordering of the data. For example, in the same padding example from Figure C.4, the first neuron in l considers the first and second non-padding neuron in $l-1$. Figure C.5 shows causal padding and how the first neuron in l only considers the first non-padding neuron in $l-1$.

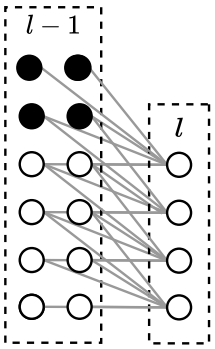


Figure C.5: Example of causal padding of a layer $l-1$ with 4×2 neurons, where $t = 3$ for layer l.

C.2.5 Dilated Convolutions

Dilated convolutions are a type of convolutions, where the neurons in the LRF for a neuron are spread out as opposed to adjacent to each other. This can be defined by the dilation rate d. The LRF for a dilated convolutional neuron can be given by Equation C.15.

$$\begin{aligned} LRF_j^l = \Big\{ &\Big((j-1)\cdot s+1+(0\cdot d),1\Big),\ldots,\Big((j-1)\cdot s+1+\big((t-1)\cdot d\big),1\Big), \\ &\Big((j-1)\cdot s+1+(0\cdot d),2\Big),\ldots,\Big((j-1)\cdot s+1+\big((t-1)\cdot d\big),i\Big)\Big\} \end{aligned} \tag{C.15}$$

Figure C.6 shows how the dilated convolutional layer l is connected to $l-1$ with a stride $s = 1$, dilation rate $d = 2$, and the size of the LRF $t = 2$. We can see how the first neuron in layer l only considers the neurons on the first and third row in layer $l-1$.

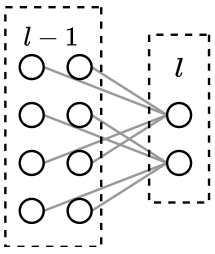


Figure C.6: Layer l, where $t = 2$ with a dilation rate $d = 2$ connected to a layer $l-1$ with 4×2 neurons.

C.2.6 Temporal Convolutional Networks

A Temporal Convolutional Network (TCN) is a network, which attempts to apply CNNs to temporal data by utilizing causal dilated convolutions [60]. This is achieved by adding multiple causal dilated convolutional layers after each other with increasing dilation rate. By increasing the dilation rate of each layer, the values from the input that is needed to calculate

the output increases, without increasing the size of the kernels. Assuming layer 0 is the first dilated convolution layer and layer l is the lth, then layer l will have a dilation rate of $d = t^l$.

Figure C.7 shows three causal dilated convolutional layers each with one kernel where $t = 2$ and an increasing dilation rate. If we consider the LRF of the neuron τ, then its LRF only contains two neurons from the previous layer, which also consider two neurons from the layer before that, and so forth. As we get to the input layer, we can see that τ is reliant on 8 input values. We can double this amount by adding layer $l = 3$, which results in every output considering 16 values from the input.

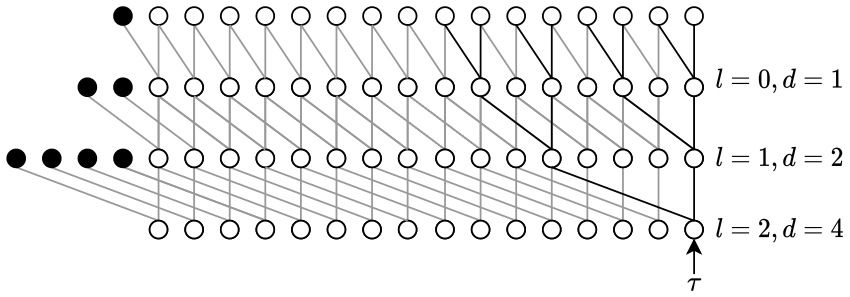


Figure C.7: A TCN with three dilated convolutional layers with an increasing dilation rate.

C.2.7 Training

Training a CNN follows the same procedure as described in Section C.1, but introduces two new types of layers, convolutional layers and pooling layers. Therefore, we describe the training process for these two types of layers. The overall training process is the same, where we find gradients through backpropagation and then update the weights. However, new equations are needed to find the gradients for convolutional and pooling layers.

Convolutional layers are trained by updating the weights in the kernels, with respect to the gradients found using backpropagation. The gradients are found through Equation C.16 [61].

$$\frac{\partial C}{\partial w_{x,y}} = \sum_j \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{x,y}} \tag{C.16}$$

We can derive the gradient $\frac{\partial C}{\partial w_{x,y}}$ in convolutional layer l by summing the gradients for every output in z^l. z_j^l is the input to the activation function, same as in Section C.1. By updating the weights in the kernel, instead of independently updating the weights between each neuron as with a normal neural network, we maintain weight sharing between convolutional neurons.

Pooling layers do not have any parameters. Backpropagation through a pooling layer only involves propagating the gradients. When backpropagating through a pooling layer, the gradient to a pooling unit is upsampled to all the neurons in the pooling unit's LRF [62]. The specific method for upsampling a gradient depends on the type of pooling layer. For max

pooling, the upsampling method propagates the gradient to the neuron with the highest activation in the LRF of each pooling unit, and propagates a gradient of 0 to the remaining neurons [63].

Appendix D

State of the Art Models

D.1 Description of Models

In this section, we give a overview of the models, we have read papers about during research for this project.

SeFT

[15] encodes the multivariate time series data into a set-encoding for their proposed Set Functions for Time Series (SeFT) model architecture. The set-encoding consists of a set of tuples on the form (t, v, f) where v is the value of feature f at timestep t for each sample. They summarize each set with a set function, which is given as input to an attention layer alongside the set and two query vectors. The output of the attention layer is given to an FFNN for classification. On CinC2019 they report a better AUROC and a worse AUPRC on SeFT compared to IP-NETS and GRU-D, which we describe in the following sections.

IP-NETS

IP-NETS is an Interpolation-Prediction network, which consists of two parts, an interpolation network and a classification network [64]. The classification network can be changed to any classification model, but a GRU is used in the paper. The interpolation network interpolates the time series input and gives the interpolated time series data to the classification network. IP-NETS is the second best performing network on Papers with Code on CinC2012 [65] and the second best on CinC2019 in the review [12].

GRU-D

The GRU-D model is a GRU with input decay and hidden state decay, which uses a data representation that indicates missing values in the time series data and a data representation that indicates the time since the last observed value [20]. We refer to these data representations as the missingness representations. They show that the model performs better than a standard GRU with multiple different imputation methods on CinC2012. They also show that appending the missingness representations to the time series input improves the performance of a standard GRU. Additionally, we found GRU-D is a commonly used model for comparison in multiple papers, where it often performs well in comparison to the other models [12][15][14][16].

DATA-GRU

The Dual-Attention Time-Aware Gated Recurrent Unit (DATA-GRU) uses a GRU and two attention mechanisms, an unreliability-aware attention, and a symptom-aware attention [16]. The unreliability-aware attention is based on the fact that imputed values are less reliable than observed values, and the symptom-aware attention is used for learning information about how the EHR data is sampled. DATA-GRU is the best performing model on CinC2019 and second best on CinC2012 [12].

LGnet

LGnet is an LSTM with a memory module for generating global estimates for the missing values [66]. The paper focuses more on forecasting rather than classification, but it is the third best performing model for the classification task for CinC2019 and CinC2012 in [12].

BRITS

BRITS (Bidirectional Recurrent Imputation for Time Series) is a recurrent method, that heavily focuses on imputation, which enhances its classification [21]. BRITS consists of two RNNs, one which handles the standard time series (forwards in time) and one which handles the time series data in reverse (going backward in time). Both RNNs predict the missing values and classify the data. It, among others, uses the difference between the missing value estimations to optimize the imputation for both RNNs. The model's final classification prediction is the average classification prediction from the two RNNs. The error in the classification is used to optimize the classification for both RNNs.

M-RNN

The M-RNN paper [67] expresses a lack of methods that consider imputation across features and time simultaneously. Therefore, they propose M-RNN, an RNN architecture with two sequential blocks. The first block considers the features of the data separately across time, which is given to the second block that considers the data across features. In its original paper, the method is tested in a classification setting, with an RNN using the imputed data [67]. Despite the method showing the best AUROC of the presented methods in the paper, more recent methods, such as BRITS, outperforms M-RNN in both imputation and classification settings [21][68][69].

D.2 Selecting Models

In this section, we discuss which of the models we want to base the models we experiment with on. As we do not have time for testing all the models, we choose the models that match the desired characteristics the best. One of the important characteristics is that it should work on time series data with a high missing rate. All of the models work with the missing rate of CinC2019, but we do not have any information about how well the models will perform with the higher missing rate of PCT. Singh et al. [19] show that there is a connection between

the observation rate and whether the patient develops sepsis. Therefore, we choose to discard the models that only replace the missing values and not actively uses the information in the missingness. These discarded models are IP-NETS and LGnet.

SeFT is interesting because it uses a different data representation and is not recurrent like the remaining models. However, SeFT performs worse than IP-NETS and GRU-D based on the AUPRC in their experiment on CinC2019 [15]. Due to the worse performance, and because SeFT requires a data representation conversion from time series, we limit the work load of the project by narrowing our choices down to models which use time series data.

BRITS and M-RNN both focuses on imputation with RNNs, and show almost equal performance in the review [12]. However, other sources show that BRITS outperforms M-RNN in both imputation, and classification settings [21, 68, 69]. Due to their similar approach, we choose to experiment with BRITS over M-RNN.

DATA-GRU is one of the best performing models according to [12], which makes it a good candidate. However, we could only find one citation besides the review, which uses it for performance comparison. This source shows GRU-D performs better than DATA-GRU for MIMIC-III and CinC2012 [70] Because it is not a well-tested model and GRU-D might perform better, we do not experiment with DATA-GRU.

As GRU-D is often used as a baseline in comparisons with other models and often performs well in the comparisons [12][14][15][16], we also choose to experiment with GRU-D.

In summary, we choose to experiment with GRU-D and BRITS. To have a baseline, we also choose to experiment with the TCN and LSTM model from our last project [2], and use LOCF imputation for handling the missing values for these models.

Appendix E

Preliminary Experiments

E.1 PhysioNet Computing in Cardiology Challenge 2019 dataset Experiment

As we describe in Section 2.2, CinC2019 consists of two datasets referred to as CinC2019A and CinC2019B. This presents us with a choice of whether to combine the two datasets as one, use them separately, or do both. We can also experiment on only one of the datasets, but this would limit our ability to reason about how the model works on multiple hospitals. While this results in less time spent per experiment, we do not consider this a good trade-off and do therefore not consider it further. We start by outlining the pros and cons of each approach in Table E.1.

Approach	Pros	Cons
CinC2019A and CinC2019B as one dataset	• Larger dataset. • More positive samples (2,932). • Better understanding about the models' ability to predict sepsis across multiple hospitals.	• Data from multiple hospitals is an additional variable. • Longer training time per model.
CinC2019A and CinC2019B as separate datasets	• Able to evaluate models on both hospitals separately. • Avoids mixing hospital data.	• Smaller datasets. • Smaller number of positive samples (1,790 and 1,142). • Train a model for each dataset.
CinC2019A and CinC2019B both as one dataset and separate	• All the pros of the other approaches. • More data.	• Longer training time. • More time needed for analysis.

Table E.1: Table of pros and cons for the different approaches of handling CinC2019A and CinC2019B.

One of the major aspects to consider is the number of positive samples available in the dataset(s), and whether or not the models can learn the positive samples. Goodfellow et

al. argue (in 2016) a rough rule of thumb is to have 5,000 samples for each class in order to achieve acceptable performance [22]. While the number of required samples can be debated for different problems, it highlights the problem that too few samples in a class can be problematic. We, therefore, need to consider whether the number of positive samples in our datasets becomes a problem if we choose to consider them separately.

Evaluating our models on a dataset from two hospitals can provide us with insights into how well they perform across hospitals. It can be preferable to remove that variable from the dataset, and simply evaluate them separately. Using CinC2019A and CinC2019B as one dataset and separately provides the most amount of data, but this data might also not benefit us. Additionally, experiments will take longer to perform and analyze, which limits the number of experiments we can do.

This is no easy choice, as all approaches have pros and cons. Due to this, we perform a preliminary experiment, where we evaluate the consequences of these three approaches. We choose to only run this experiment with the TCN and LSTM from our previous semester report [2], as we are not attempting to do an exhaustive analysis. For the TCN, we use two temporal blocks with 64 kernels. For the LSTM, we use 128 units. We use the setup we describe in Chapter 10 on CinC2019A, CinC2019B and CinC2019A + CinC2019B with both the TCN and the LSTM.

A general issue we observe is that the models tend to not make predictions with high confidences. We see that there is a very small number of predictions above 0.3. Even though we also see this when using CinC2019A and CinC2019B as one dataset, this problem may be caused by having too few positive samples in the dataset. However, it is important to note that we did not consider the hyperparameters for these models in great detail. Due to this, better tuned models might alleviate this problem. Still, in order to maximize the number of positive samples, we choose to combine CinC2019A and CinC2019B as one dataset, which we simply refer to as CinC2019 from now on. Performance and calibration metrics and the relevant graphs for this experiment can be seen in the following sections.

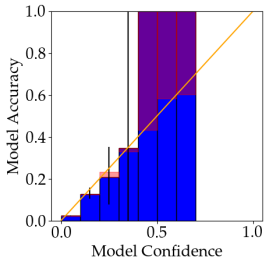
E.2 Preliminary Experiment CinC2019A

preliminary	ECE	ACE	MCE
tcn	$\mathbf{0.003 \pm 0.004}$	$\mathbf{0.004 \pm 0.003}$	$\mathbf{0.406 \pm 0.275}$
lstm	$\mathbf{0.005 \pm 0.003}$	$\mathbf{0.006 \pm 0.002}$	$\mathbf{0.453 \pm 0.279}$

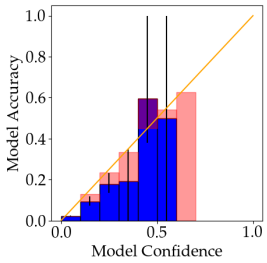
Table E.2: Calibration metrics

preliminary	AUROC	AUPRC
tcn	$\mathbf{0.744 \pm 0.009}$	$\mathbf{0.075 \pm 0.002}$
lstm	$\mathbf{0.747 \pm 0.007}$	$\mathbf{0.073 \pm 0.002}$

Table E.3: Performance metrics

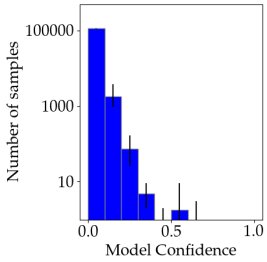


(a) reliability-diagram-dpa-Preliminary-tcn

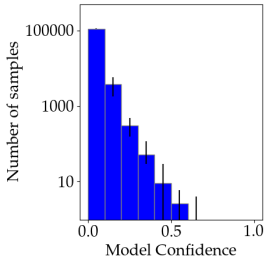


(b) reliability-diagram-dpa-Preliminary-lstm

Figure E.1: Reliability diagrams for the experiment.

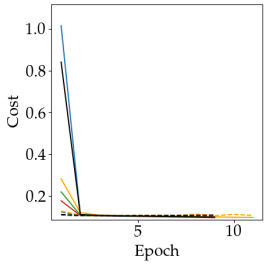


(a) sample-confidence-distribution-graph-dpa-Preliminary-tcn

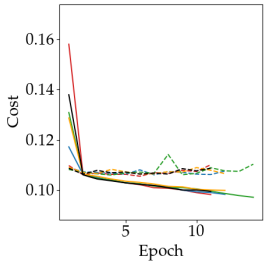


(b) sample-confidence-distribution-graph-dpa-Preliminary-lstm

Figure E.2: Sample confidence distribution graph for the experiment.

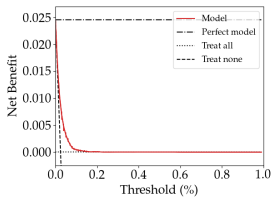


(a) history-graph-dpa-Preliminary-tcn

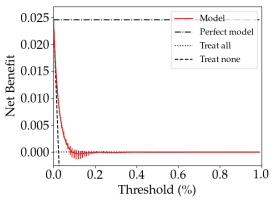


(b) history-graph-dpa-Preliminary-lstm

Figure E.3: Training and validation loss for each epoch for the experiment.

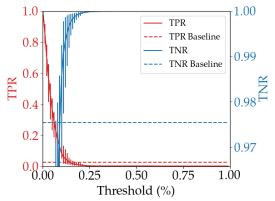


(a) dc-graph-dpa-Preliminary-tcn

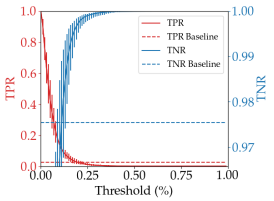


(b) dc-graph-dpa-Preliminary-lstm

Figure E.4: DC.



(a) TPRTNR-graph-dpa-Preliminary-tcn



(b) TPRTNR-graph-dpa-Preliminary-lstm

Figure E.5: TPR and TNR.

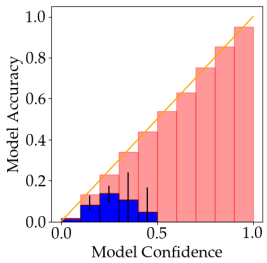
E.3 Preliminary Experiment CinC2019B

preliminary	ECE	ACE	MCE
tcn	$\mathbf{0.005 \pm 0.002}$	$\mathbf{0.007 \pm 0.003}$	$\mathbf{0.768 \pm 0.257}$
lstm	$\mathbf{0.003 \pm 0.001}$	$\mathbf{0.005 \pm 0.002}$	$\mathbf{0.583 \pm 0.143}$

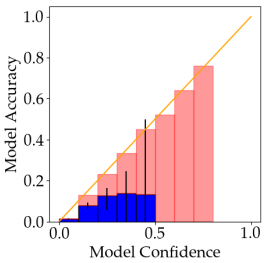
Table E.4: Calibration metrics

preliminary	AUROC	AUPRC
tcn	$\mathbf{0.716 \pm 0.013}$	$\mathbf{0.045 \pm 0.005}$
lstm	$\mathbf{0.737 \pm 0.019}$	$\mathbf{0.048 \pm 0.002}$

Table E.5: Performance metrics

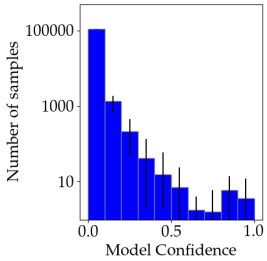


(a) reliability-diagram-dpb-Preliminary-tcn

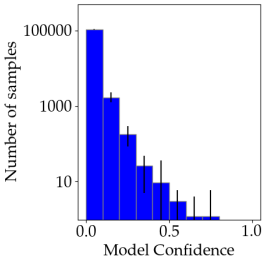


(b) reliability-diagram-dpb-Preliminary-lstm

Figure E.6: Reliability diagrams for the experiment.

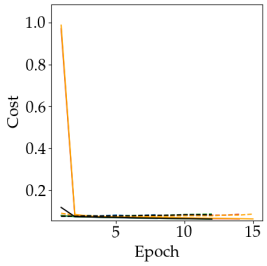


(a) sample-confidence-distribution-graph-dpb-Preliminary-tcn

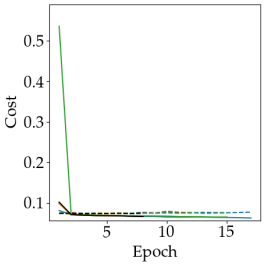


(b) sample-confidence-distribution-graph-dpb-Preliminary-lstm

Figure E.7: Sample confidence distribution graph for the experiment.

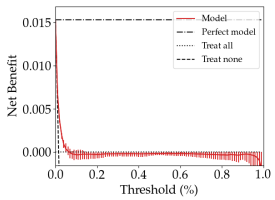


(a) history-graph-dpb-Preliminary-tcn

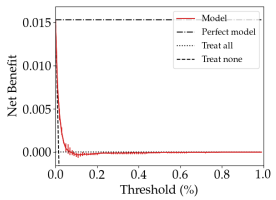


(b) history-graph-dpb-Preliminary-lstm

Figure E.8: Training and validation loss for each epoch for the experiment.

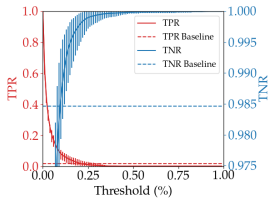


(a) dc-graph-dpb-Preliminary-tcn

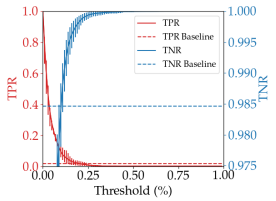


(b) dc-graph-dpb-Preliminary-lstm

Figure E.9: DC.



(a) TPRTNR-graph-dpb-Preliminary-tcn



(b) TPRTNR-graph-dpb-Preliminary-lstm

Figure E.10: TPR and TNR.

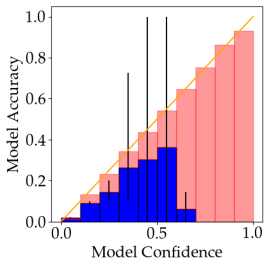
E.4 Preliminary Experiment CinC2019

preliminary	ECE	ACE	MCE
tcn	**0.009 ± 0.002**	**0.008 ± 0.003**	**0.828 ± 0.180**
lstm	**0.008 ± 0.003**	**0.007 ± 0.003**	**0.366 ± 0.440**

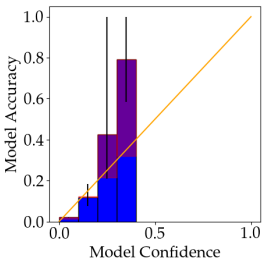
Table E.6: Calibration metrics

preliminary	AUROC	AUPRC
tcn	**0.729 ± 0.011**	**0.062 ± 0.004**
lstm	**0.745 ± 0.009**	**0.065 ± 0.002**

Table E.7: Performance metrics

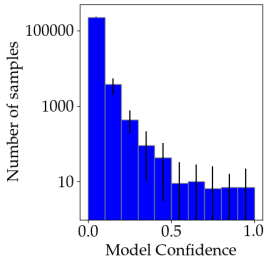


(a) reliability-diagram-dp-Preliminary-tcn

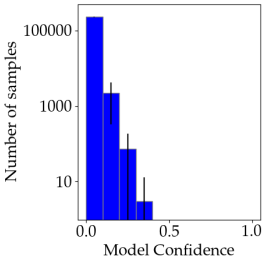


(b) reliability-diagram-dp-Preliminary-lstm

Figure E.11: Reliability diagrams for the experiment.

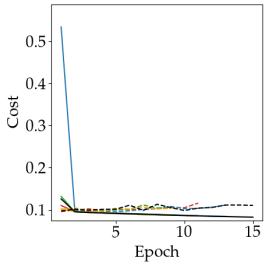


(a) sample-confidence-distribution-graph-dp-Preliminary-tcn

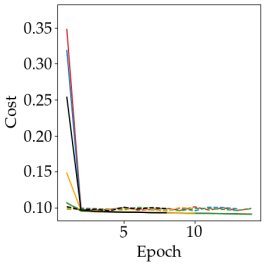


(b) sample-confidence-distribution-graph-dp-Preliminary-lstm

Figure E.12: Sample confidence distribution graph for the experiment.

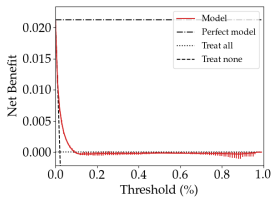


(a) history-graph-dp-Preliminary-tcn

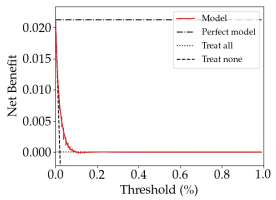


(b) history-graph-dp-Preliminary-lstm

Figure E.13: Training and validation loss for each epoch for the experiment.

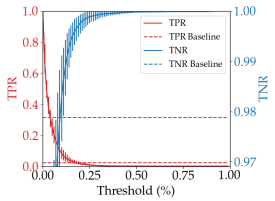


(a) dc-graph-dp-Preliminary-tcn

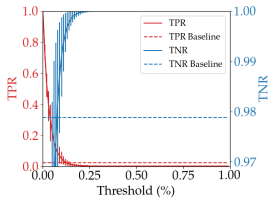


(b) dc-graph-dp-Preliminary-lstm

Figure E.14: DC.



(a) TPRTNR-graph-dp-Preliminary-tcn



(b) TPRTNR-graph-dp-Preliminary-lstm

Figure E.15: TPR and TNR.

E.5 Hyperparameter Tuning

In this section, we find the baseline hyperparameters we use in our experiments. Before we consider how we find the values for these hyperparameters, we need to establish which of the hyperparameters we consider changing. Testing every possible combination can be done by performing an exhaustive search for every combination of every hyperparameter. However, the downside to this is that it is very time consuming. Therefore, we start by selecting the hyperparameters we consider most relevant for each model.

As we describe in Section 7.2, the TCN is made of temporal blocks, consisting of causal, dilated, convolution layers. We choose to tune the number of temporal blocks, the number of filters, and the type of activation function. When we change the filters or the activation function, this change is made to all the convolution layers.

As for LSTM, BRITS, and GRU-D, we choose to tune the number of units.

E.5.1 Tuning

In this section, we describe how we test the hyperparameters and how we evaluate their effects on the models. The goal is not to do a comprehensive analysis of the effects of the hyperparameters, as the optimal values for these hyperparameters might change as we make changes to our models in our experiments. Instead, the goal is to provide a general idea of how the hyperparameters affect our models.

In order to do this, we utilize a random search for the hyperparameters. Random search randomly chooses a unique combination of hyperparameters, trains the model with these

hyperparameters, and then shows the results, and repeats with another unique combination of hyperparameters. To maximize the number of combinations of hyperparameters we test, we only test each configuration once. Also, we only consider the AUPRC, AUROC, and ECE on the validation data.

A minimum and maximum value, and the step size is defined for every numerical hyperparameter. For example, for the number of filters in the TCN, we choose a minimum value of 16 and a maximum value of 128 and a step size of 16. This results in the following possible values for the number of filters: [16, 32, 48, . . . , 112, 128].
The tuning parameters are as follows:

- TCN
  - Filters
    - Min: 16
    - Max: 128
    - Step: 16
  - Temporal Blocks
    - Min: 1
    - Max: 5
    - Step: 1
  - Activation
    - ReLU
    - TanH
- LSTM, BRITS, and GRU-D
  - Units
    - Min: 32
    - Max: 512
    - Step: 32

E.5.2 Results

For TCN, we saw that ReLU generally outperforms TanH as the activation function. Two temporal blocks perform best. Changing the number of filters did not provide any meaningful change. For these reasons, we choose to use ReLU, with two temporal blocks for TCN, and 64 filters.

For LSTM, we observe that 64 units performs the best, and therefore choose 64 units for LSTM.

For BRITS, we see no difference in performance when changing units. Therefore, we use 108 units for BRITS, as this is the number of units Cao et al. [21] uses in their original implementation [71].

For GRU-D, we generally see better performance when increasing the number of units, up to 128 units. Using more than 128 units does not appear to benefit the model. For this reason, we choose to use 128 units for GRU-D.

Appendix F

PhysioNet Computing in Cardiology Challenge 2019 dataset

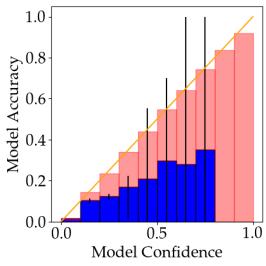
F.1 Baseline

baseline	ECE	ACE	MCE
TCN	0.011 ± 0.002	0.010 ± 0.002	0.688 ± 0.279
LSTM	$\mathbf{0.007 \pm 0.001}$	$\mathbf{0.006 \pm 0.001}$	$\mathbf{0.363 \pm 0.236}$
BRITS	$\mathbf{0.007 \pm 0.002}$	$\mathbf{0.007 \pm 0.001}$	$\mathbf{0.225 \pm 0.182}$
GRUD	$\mathbf{0.007 \pm 0.001}$	$\mathbf{0.006 \pm 0.001}$	0.692 ± 0.174
XGBOOST	0.008 ± 0.000	$\mathbf{0.007 \pm 0.000}$	0.453 ± 0.000

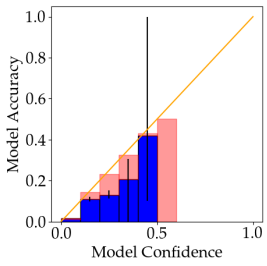
Table F.1: Calibration metrics

baseline	AUROC	AUPRC
TCN	0.753 ± 0.011	0.083 ± 0.006
LSTM	0.764 ± 0.007	0.076 ± 0.003
BRITS	0.769 ± 0.012	0.089 ± 0.007
GRUD	0.731 ± 0.009	0.066 ± 0.001
XGBOOST	$\mathbf{0.816 \pm 0.000}$	$\mathbf{0.114 \pm 0.000}$

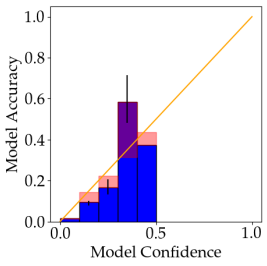
Table F.2: Performance metrics



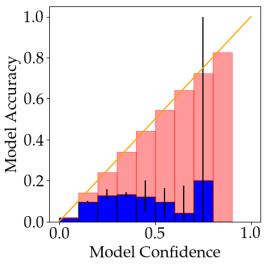
(a) reliability-diagram-dp-baseline-TCN



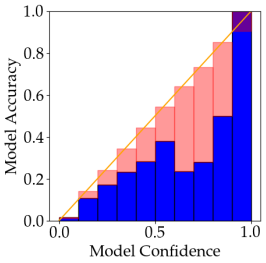
(b) reliability-diagram-dp-baseline-LSTM



(c) reliability-diagram-dp-baseline-BRITS

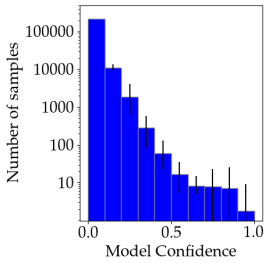


(d) reliability-diagram-dp-baseline-GRUD

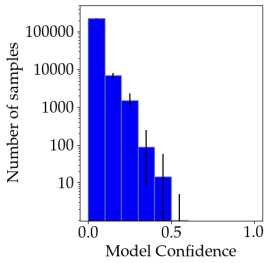


(e) reliability-diagram-dp-baseline-XGBOOST

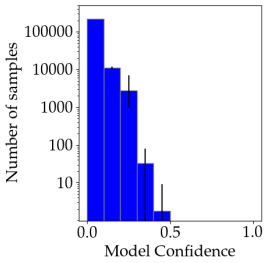
Figure F.1: Reliability diagrams for the experiment.



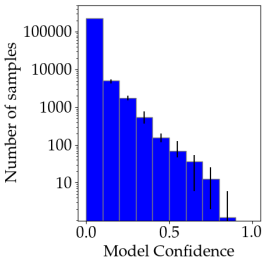
(a) sample-confidence-distribution-graph-dp-baseline-TCN



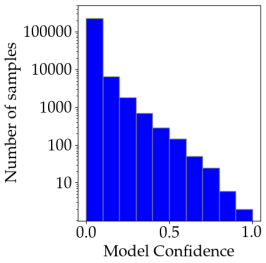
(b) sample-confidence-distribution-graph-dp-baseline-LSTM



(c) sample-confidence-distribution-graph-dp-baseline-BRITS

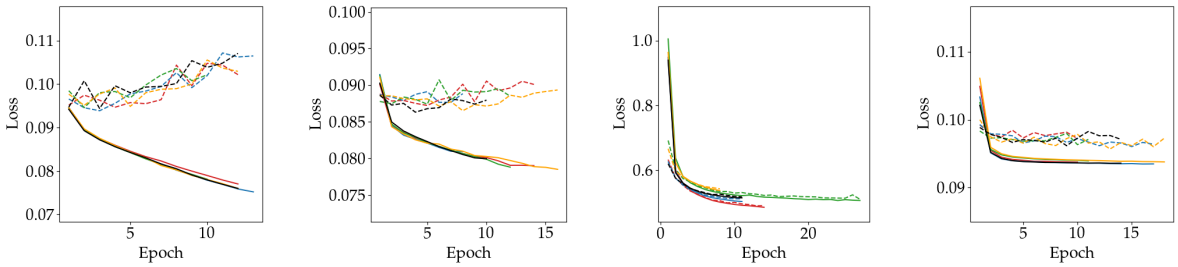


(d) sample-confidence-distribution-graph-dp-baseline-GRUD



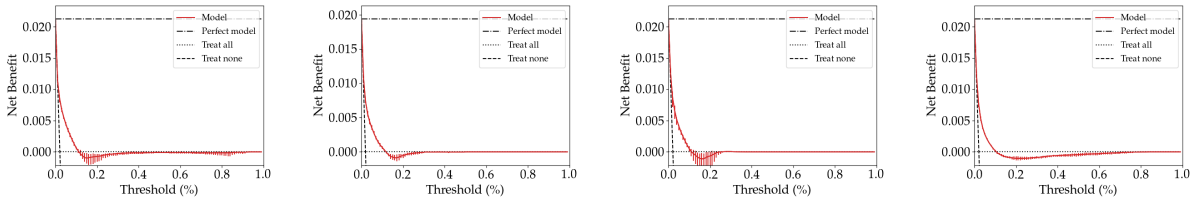
(e) sample-confidence-distribution-graph-dp-baseline-XGBOOST

Figure F.2: Sample confidence distribution graph for the experiment.

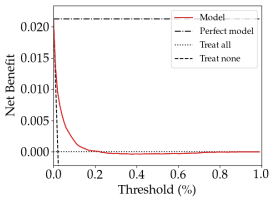


(a) history-graph-dp-baseline-TCN **(b)** history-graph-dp-baseline-LSTM **(c)** history-graph-dp-baseline-BRITS **(d)** history-graph-dp-baseline-GRUD

Figure F.3: Training and validation loss for each epoch for the experiment.

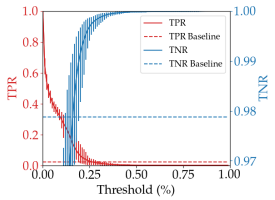


(a) dc-graph-dp-baseline-TCN **(b)** dc-graph-dp-baseline-LSTM **(c)** dc-graph-dp-baseline-BRITS **(d)** dc-graph-dp-baseline-GRUD

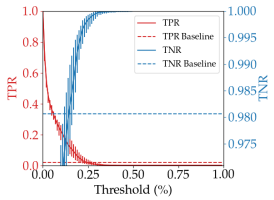


(e) dc-graph-dp-baseline-XGBOOST

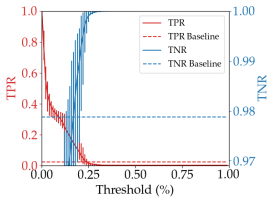
Figure F.4: DC.



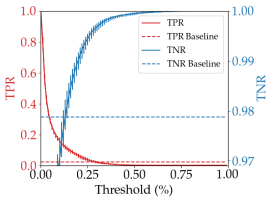
(a) TPRTNR-graph-dp-baseline-TCN



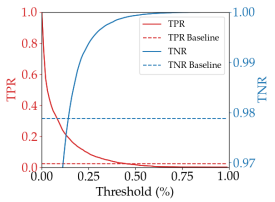
(b) TPRTNR-graph-dp-baseline-LSTM



(c) TPRTNR-graph-dp-baseline-BRITS



(d) TPRTNR-graph-dp-baseline-GRUD



(e) TPRTNR-graph-dp-baseline-XGBOOST

Figure F.5: TPR and TNR.

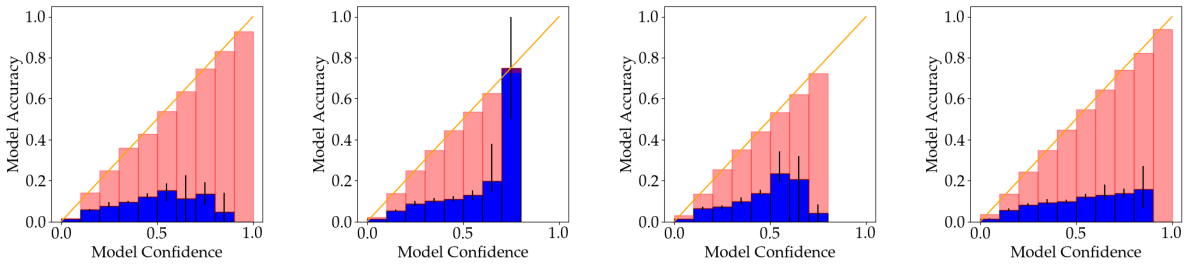
F.2 Class Weight Experiment

class-weight	ECE	ACE	MCE
4-TCN	0.029 ± 0.004	0.031 ± 0.001	0.738 ± 0.165
4-LSTM	0.030 ± 0.003	0.030 ± 0.003	$\mathbf{0.454 \pm 0.057}$
4-BRITS	0.035 ± 0.009	0.035 ± 0.009	$\mathbf{0.536 \pm 0.211}$
4-GRUD	0.041 ± 0.006	0.041 ± 0.006	0.762 ± 0.209
4-XGBOOST	$\mathbf{0.024 \pm 0.000}$	$\mathbf{0.024 \pm 0.000}$	$\mathbf{0.506 \pm 0.000}$
16-TCN	0.100 ± 0.014	0.102 ± 0.013	0.735 ± 0.049
16-LSTM	0.113 ± 0.020	0.113 ± 0.020	0.755 ± 0.120
16-BRITS	0.122 ± 0.021	0.122 ± 0.021	0.682 ± 0.162
16-GRUD	0.152 ± 0.014	0.152 ± 0.014	0.807 ± 0.048
16-XGBOOST	0.086 ± 0.000	0.086 ± 0.000	0.664 ± 0.000
49-TCN	0.166 ± 0.016	0.167 ± 0.015	0.796 ± 0.005
49-LSTM	0.231 ± 0.024	0.231 ± 0.024	0.799 ± 0.019
49-BRITS	0.288 ± 0.034	0.288 ± 0.034	0.778 ± 0.019
49-GRUD	0.338 ± 0.023	0.338 ± 0.023	0.831 ± 0.013
49-XGBOOST	0.172 ± 0.000	0.172 ± 0.000	0.752 ± 0.000

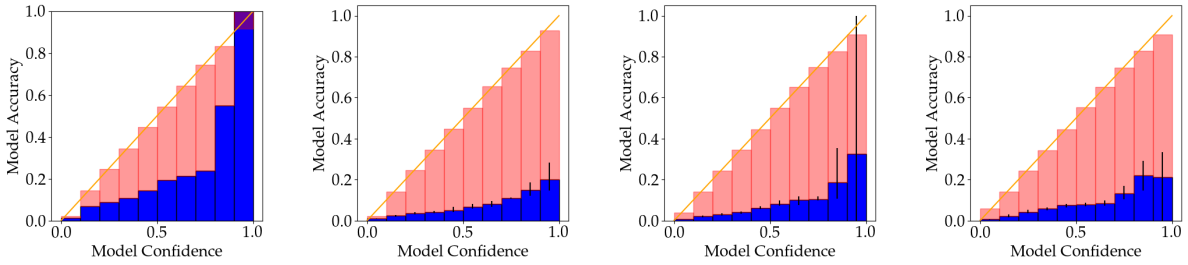
Table F.3: Calibration metrics

class-weight	AUROC	AUPRC
4-TCN	0.778 ± 0.013	0.082 ± 0.004
4-LSTM	0.776 ± 0.007	0.076 ± 0.002
4-BRITS	0.775 ± 0.012	0.091 ± 0.005
4-GRUD	0.741 ± 0.013	0.068 ± 0.002
4-XGBOOST	$\mathbf{0.812 \pm 0.000}$	$\mathbf{0.111 \pm 0.000}$
16-TCN	0.775 ± 0.020	0.085 ± 0.003
16-LSTM	0.775 ± 0.012	0.078 ± 0.004
16-BRITS	0.793 ± 0.004	0.097 ± 0.004
16-GRUD	0.747 ± 0.008	0.068 ± 0.003
16-XGBOOST	0.805 ± 0.000	0.096 ± 0.000
49-TCN	0.776 ± 0.005	0.082 ± 0.003
49-LSTM	0.774 ± 0.004	0.075 ± 0.003
49-BRITS	0.790 ± 0.005	0.094 ± 0.005
49-GRUD	0.739 ± 0.010	0.068 ± 0.003
49-XGBOOST	0.783 ± 0.000	0.084 ± 0.000

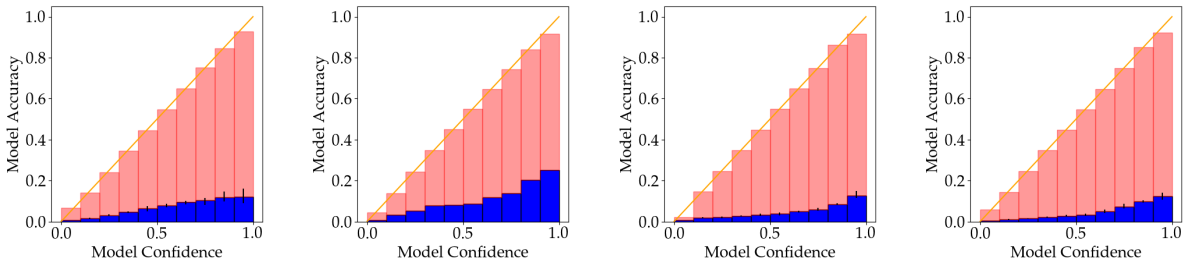
Table F.4: Performance metrics



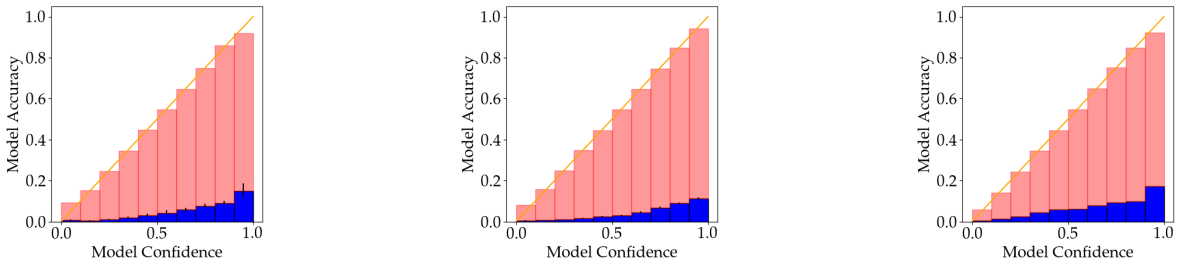
(a) reliability-diagram-dp-class-ratio-4-TCN **(b)** reliability-diagram-dp-class-ratio-4-LSTM **(c)** reliability-diagram-dp-class-ratio-4-BRITS **(d)** reliability-diagram-dp-class-ratio-4-GRUD



(e) reliability-diagram-dp-class-ratio-4-XGBOOST **(f)** reliability-diagram-dp-class-ratio-16-TCN **(g)** reliability-diagram-dp-class-ratio-16-LSTM **(h)** reliability-diagram-dp-class-ratio-16-BRITS

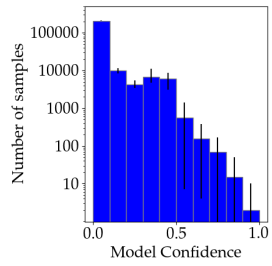


(i) reliability-diagram-dp-class-ratio-16-GRUD **(j)** reliability-diagram-dp-class-ratio-16-XGBOOST **(k)** reliability-diagram-dp-class-ratio-49-TCN **(l)** reliability-diagram-dp-class-ratio-49-LSTM

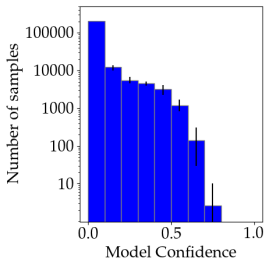


(m) reliability-diagram-dp-class-ratio-49-BRITS **(n)** reliability-diagram-dp-class-ratio-49-GRUD **(o)** reliability-diagram-dp-class-ratio-49-XGBOOST

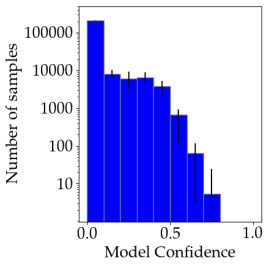
Figure F.6: Reliability diagrams for the experiment.



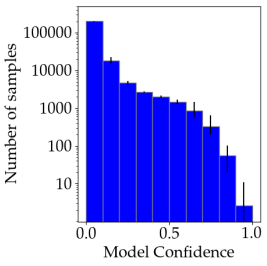
(a) sample-confidence-distribution-graph-dp-class-ratio-4-TCN



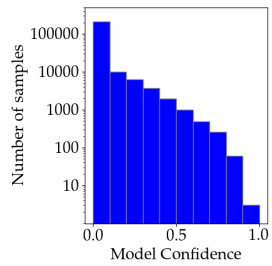
(b) sample-confidence-distribution-graph-dp-class-ratio-4-LSTM



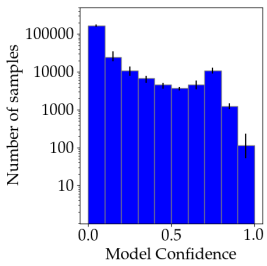
(c) sample-confidence-distribution-graph-dp-class-ratio-4-BRITS



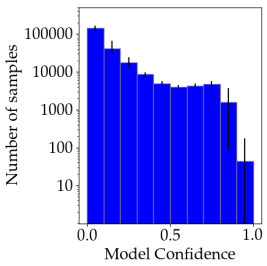
(d) sample-confidence-distribution-graph-dp-class-ratio-4-GRUD



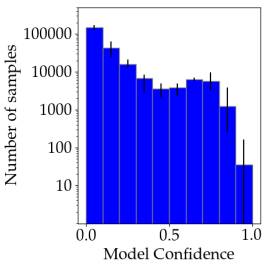
(e) sample-confidence-distribution-graph-dp-class-ratio-4-XGBOOST



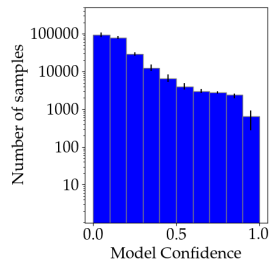
(f) sample-confidence-distribution-graph-dp-class-ratio-16-TCN



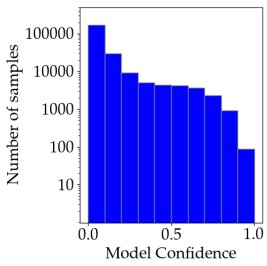
(g) sample-confidence-distribution-graph-dp-class-ratio-16-LSTM



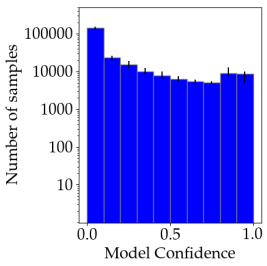
(h) sample-confidence-distribution-graph-dp-class-ratio-16-BRITS



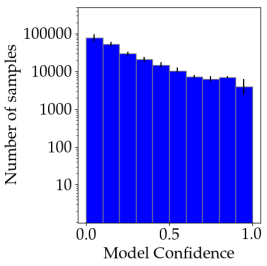
(i) sample-confidence-distribution-graph-dp-class-ratio-16-GRUD



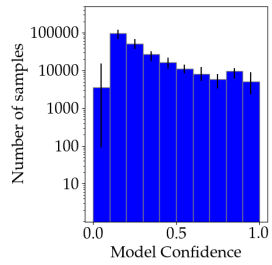
(j) sample-confidence-distribution-graph-dp-class-ratio-16-XGBOOST



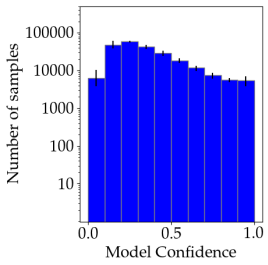
(k) sample-confidence-distribution-graph-dp-class-ratio-49-TCN



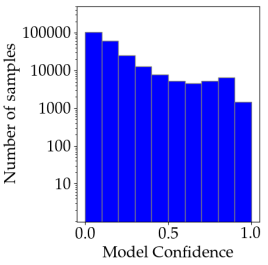
(l) sample-confidence-distribution-graph-dp-class-ratio-49-LSTM



(m) sample-confidence-distribution-graph-dp-class-ratio-49-BRITS

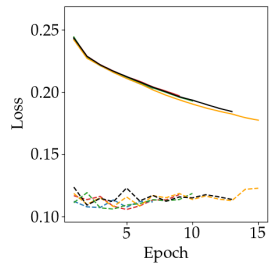


(n) sample-confidence-distribution-graph-dp-class-ratio-49-GRUD

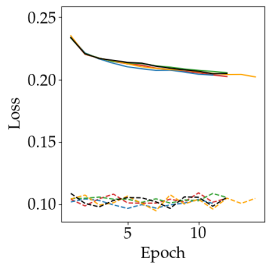


(o) sample-confidence-distribution-graph-dp-class-ratio-49-XGBOOST

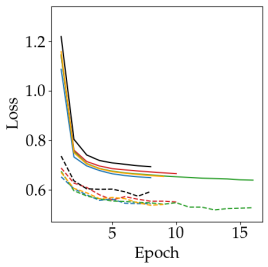
Figure F.7: Sample confidence distribution graph for the experiment.



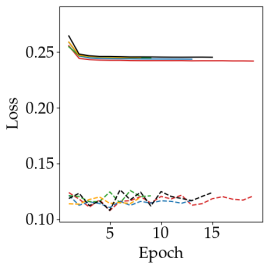
(a) history-graph-dp-class-ratio-4-TCN



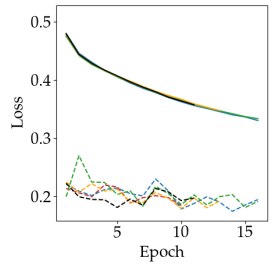
(b) history-graph-dp-class-ratio-4-LSTM



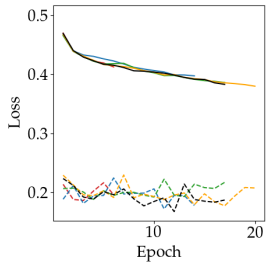
(c) history-graph-dp-class-ratio-4-BRITS



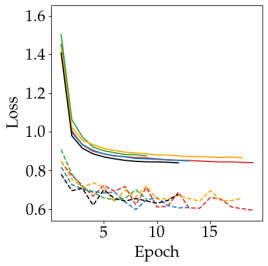
(d) history-graph-dp-class-ratio-4-GRUD



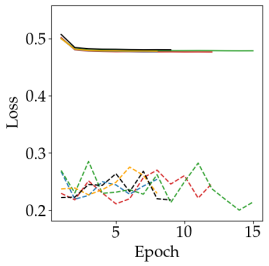
(e) history-graph-dp-class-ratio-16-TCN



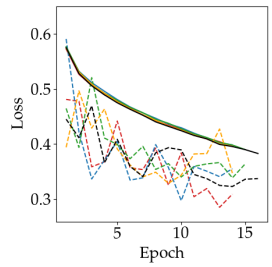
(f) history-graph-dp-class-ratio-16-LSTM



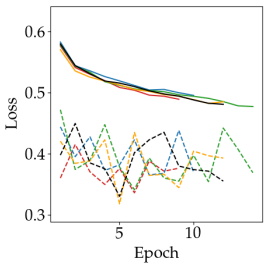
(g) history-graph-dp-class-ratio-16-BRITS



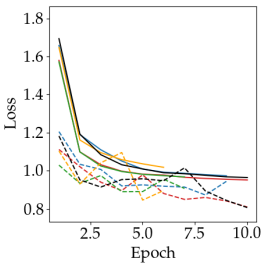
(h) history-graph-dp-class-ratio-16-GRUD



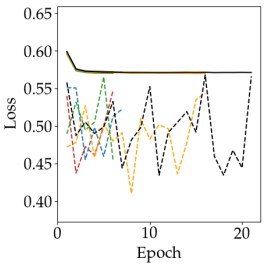
(i) history-graph-dp-class-ratio-49-TCN



(j) history-graph-dp-class-ratio-49-LSTM

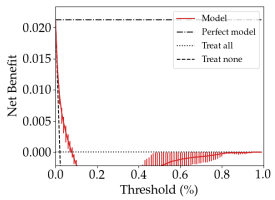


(k) history-graph-dp-class-ratio-49-BRITS

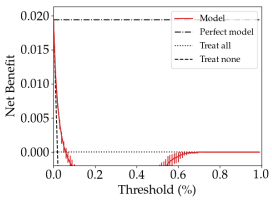


(l) history-graph-dp-class-ratio-49-GRUD

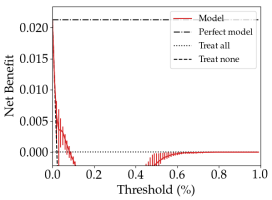
Figure F.8: Training and validation loss for each epoch for the experiment.



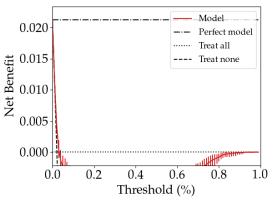
(a) dc-graph-dp-class-ratio-4-TCN



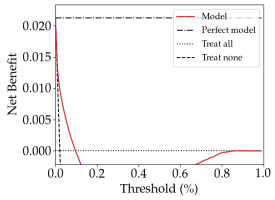
(b) dc-graph-dp-class-ratio-4-LSTM



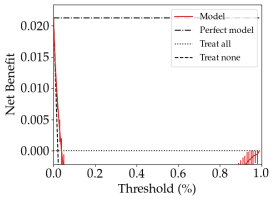
(c) dc-graph-dp-class-ratio-4-BRITS



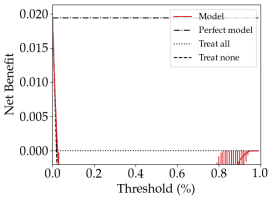
(d) dc-graph-dp-class-ratio-4-GRUD



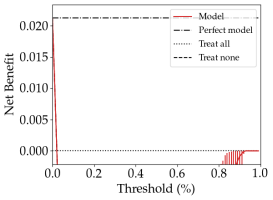
(e) dc-graph-dp-class-ratio-4-XGBOOST



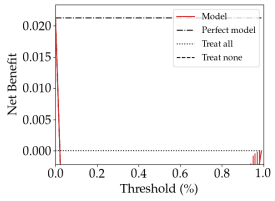
(f) dc-graph-dp-class-ratio-16-TCN



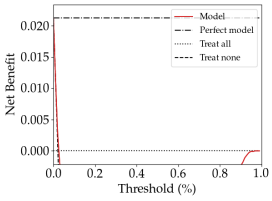
(g) dc-graph-dp-class-ratio-16-LSTM



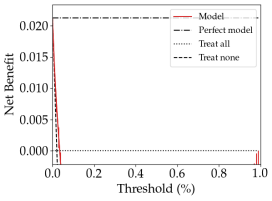
(h) dc-graph-dp-class-ratio-16-BRITS



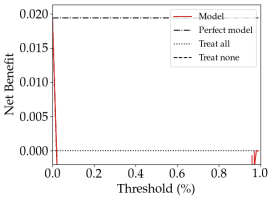
(i) dc-graph-dp-class-ratio-16-GRUD



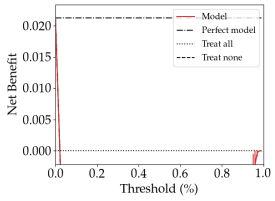
(j) dc-graph-dp-class-ratio-16-XGBOOST



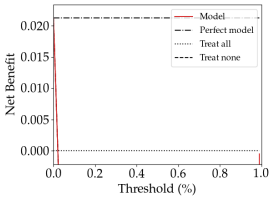
(k) dc-graph-dp-class-ratio-49-TCN



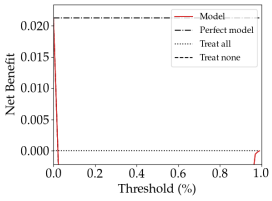
(l) dc-graph-dp-class-ratio-49-LSTM



(m) dc-graph-dp-class-ratio-49-BRITS

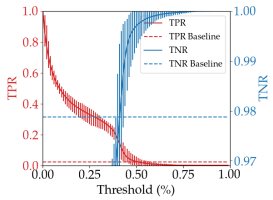


(n) dc-graph-dp-class-ratio-49-GRUD

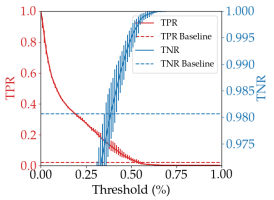


(o) dc-graph-dp-class-ratio-49-XGBOOST

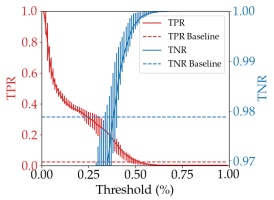
Figure F.9: DC.



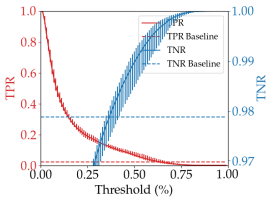
(a) TPRTNR-graph-dp-class-ratio-4-TCN



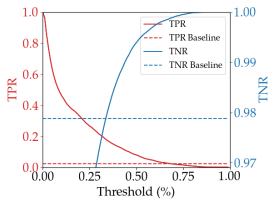
(b) TPRTNR-graph-dp-class-ratio-4-LSTM



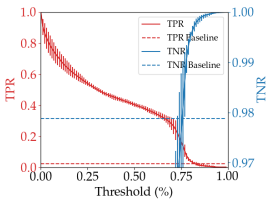
(c) TPRTNR-graph-dp-class-ratio-4-BRITS



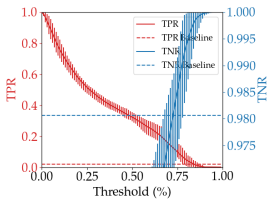
(d) TPRTNR-graph-dp-class-ratio-4-GRUD



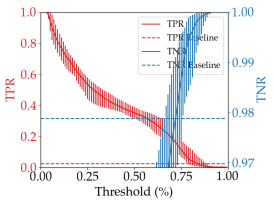
(e) TPRTNR-graph-dp-class-ratio-4-XGBOOST



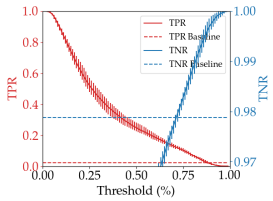
(f) TPRTNR-graph-dp-class-ratio-16-TCN



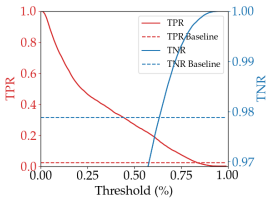
(g) TPRTNR-graph-dp-class-ratio-16-LSTM



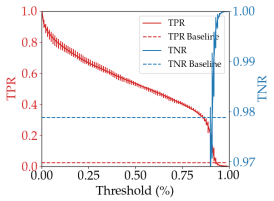
(h) TPRTNR-graph-dp-class-ratio-16-BRITS



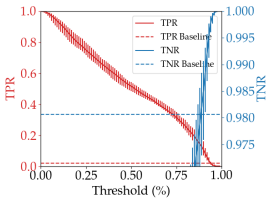
(i) TPRTNR-graph-dp-class-ratio-16-GRUD



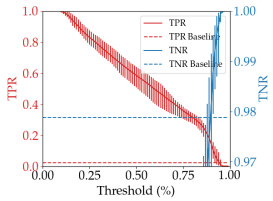
(j) TPRTNR-graph-dp-class-ratio-16-XGBOOST



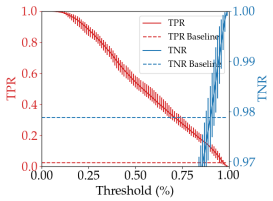
(k) TPRTNR-graph-dp-class-ratio-49-TCN



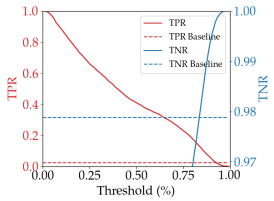
(l) TPRTNR-graph-dp-class-ratio-49-LSTM



(m) TPRTNR-graph-dp-class-ratio-49-BRITS



(n) TPRTNR-graph-dp-class-ratio-49-GRUD



(o) TPRTNR-graph-dp-class-ratio-49-XGBOOST

Figure F.10: TPR and TNR.

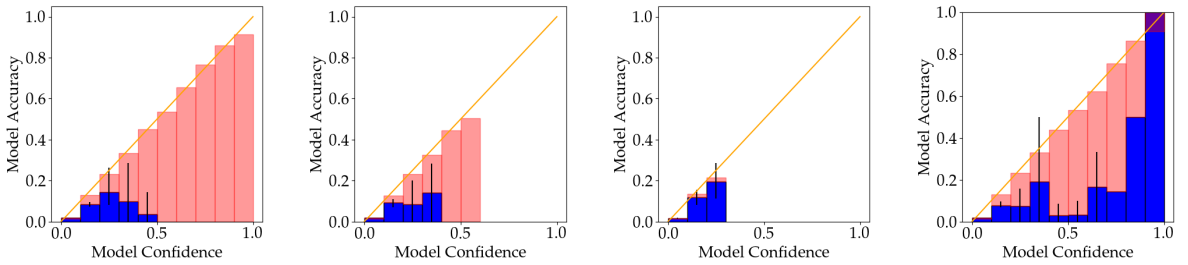
F.3 Demographics Experiment

demographics	ECE	ACE	MCE
TCN	$\mathbf{0.007 \pm 0.004}$	$\mathbf{0.007 \pm 0.002}$	0.654 ± 0.327
LSTM	$\mathbf{0.007 \pm 0.002}$	$\mathbf{0.007 \pm 0.002}$	$\mathbf{0.278 \pm 0.184}$
BRITS	$\mathbf{0.005 \pm 0.002}$	$\mathbf{0.005 \pm 0.002}$	$\mathbf{0.066 \pm 0.040}$
GRUD	$\mathbf{0.008 \pm 0.002}$	$\mathbf{0.008 \pm 0.002}$	0.425 ± 0.256

Table F.5: Calibration metrics

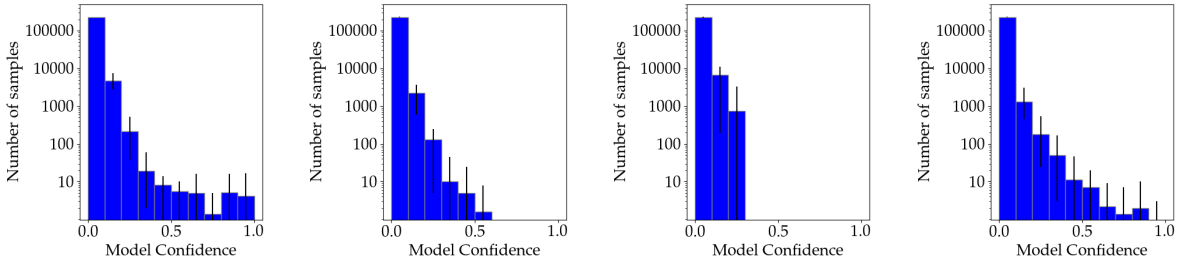
demographics	AUROC	AUPRC
TCN	$\mathbf{0.720 \pm 0.016}$	0.053 ± 0.002
LSTM	$\mathbf{0.737 \pm 0.011}$	$\mathbf{0.056 \pm 0.002}$
BRITS	$\mathbf{0.733 \pm 0.037}$	$\mathbf{0.070 \pm 0.014}$
GRUD	0.663 ± 0.010	0.040 ± 0.003

Table F.6: Performance metrics



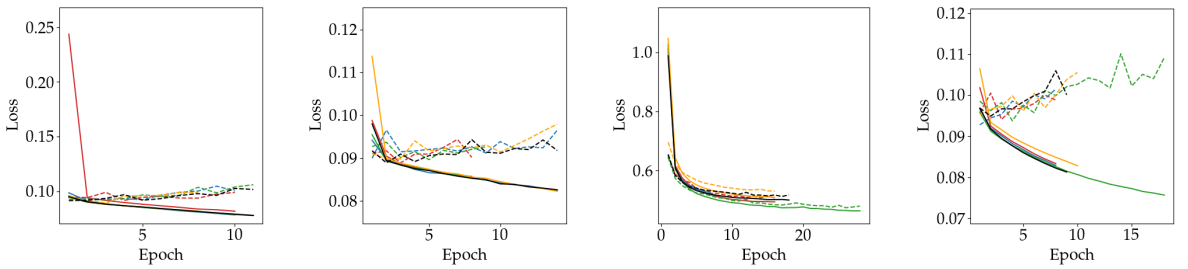
(a) reliability-diagram-dp-demographics-TCN **(b)** reliability-diagram-dp-demographics-LSTM **(c)** reliability-diagram-dp-demographics-BRITS **(d)** reliability-diagram-dp-demographics-GRUD

Figure F.11: Reliability diagrams for the experiment.



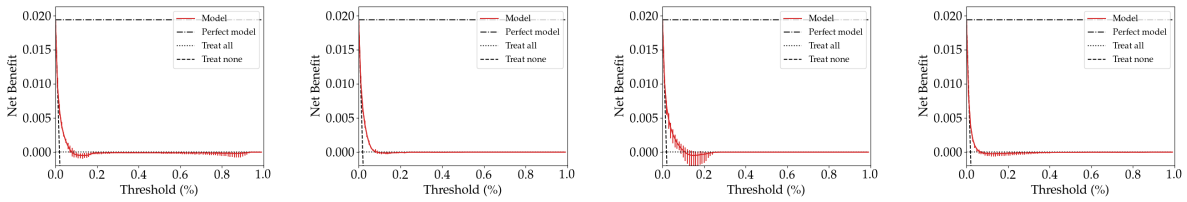
(a) sample-confidence-distribution-graph-dp-demographics-TCN **(b)** sample-confidence-distribution-graph-dp-demographics-LSTM **(c)** sample-confidence-distribution-graph-dp-demographics-BRITS **(d)** sample-confidence-distribution-graph-dp-demographics-GRUD

Figure F.12: Sample confidence distribution graph for the experiment.



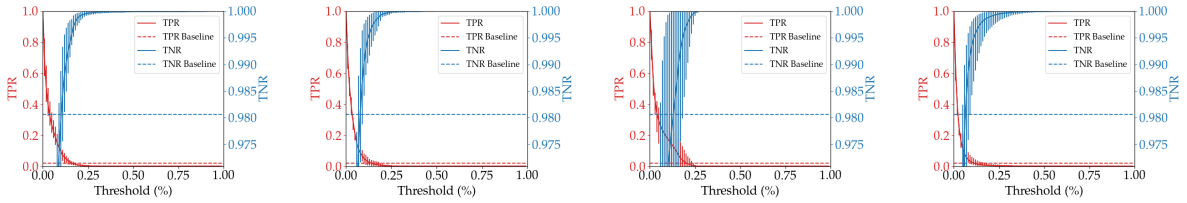
(a) history-graph-dp-demographics-TCN **(b)** history-graph-dp-demographics-LSTM **(c)** history-graph-dp-demographics-BRITS **(d)** history-graph-dp-demographics-GRUD

Figure F.13: Training and validation loss for each epoch for the experiment.



(a) dc-graph-dp-demographics-TCN **(b)** dc-graph-dp-demographics-LSTM **(c)** dc-graph-dp-demographics-BRITS **(d)** dc-graph-dp-demographics-GRUD

Figure F.14: DC.



(a) TPRTNR-graph-dp-demographics-TCN **(b)** TPRTNR-graph-dp-demographics-LSTM **(c)** TPRTNR-graph-dp-demographics-BRITS **(d)** TPRTNR-graph-dp-demographics-GRUD

Figure F.15: TPR and TNR.

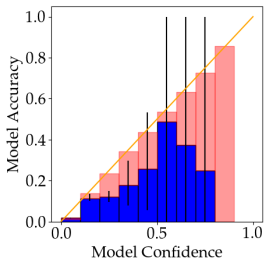
F.4 Missingness Data Representation Experiment

missingness-representation	ECE	ACE	MCE
TCN	**0.010 ± 0.003**	**0.008 ± 0.002**	**0.586 ± 0.322**
LSTM	**0.008 ± 0.002**	**0.007 ± 0.002**	**0.286 ± 0.225**

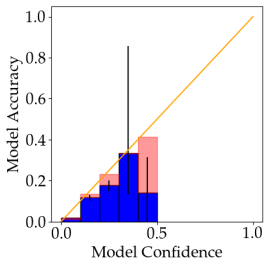
Table F.7: Calibration metrics

missingness-representation	AUROC	AUPRC
TCN	**0.772 ± 0.016**	**0.087 ± 0.005**
LSTM	**0.779 ± 0.007**	**0.091 ± 0.003**

Table F.8: Performance metrics

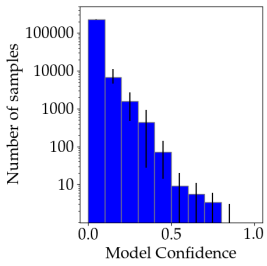


(a) reliability-diagram-dp-missingness-representation-TCN

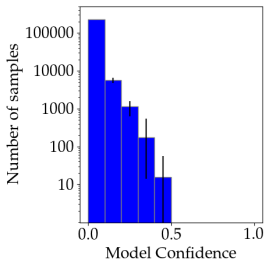


(b) reliability-diagram-dp-missingness-representation-LSTM

Figure F.16: Reliability diagrams for the experiment.

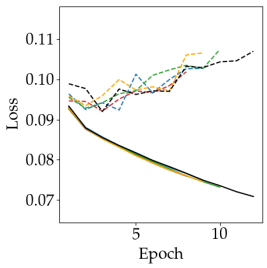


(a) sample-confidence-distribution-graph-dp-missingness-representation-TCN

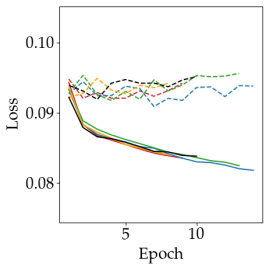


(b) sample-confidence-distribution-graph-dp-missingness-representation-LSTM

Figure F.17: Sample confidence distribution graph for the experiment.

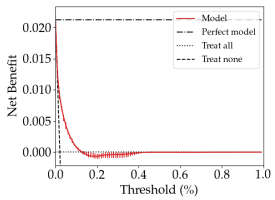


(a) history-graph-dp-missingness-representation-TCN

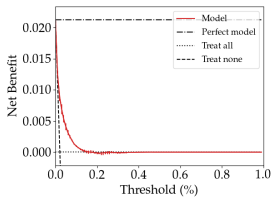


(b) history-graph-dp-missingness-representation-LSTM

Figure F.18: Training and validation loss for each epoch for the experiment.

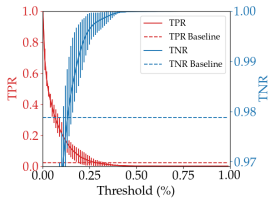


(a) dc-graph-dp-missingness-representation-TCN

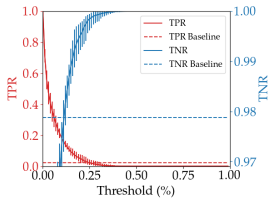


(b) dc-graph-dp-missingness-representation-LSTM

Figure F.19: DC.



(a) TPRTNR-graph-dp-missingness-representation-TCN



(b) TPRTNR-graph-dp-missingness-representation-LSTM

Figure F.20: TPR and TNR.

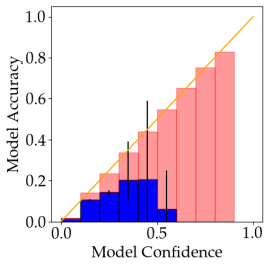
F.5 Observation Rate Experiment

observation-rate	ECE	ACE	MCE
TCN	$\mathbf{0.009 \pm 0.001}$	$\mathbf{0.009 \pm 0.001}$	0.772 ± 0.106
LSTM	$\mathbf{0.008 \pm 0.001}$	$\mathbf{0.008 \pm 0.001}$	$\mathbf{0.418 \pm 0.258}$
BRITS	$\mathbf{0.008 \pm 0.003}$	$\mathbf{0.008 \pm 0.003}$	$\mathbf{0.198 \pm 0.335}$
GRUD	$\mathbf{0.008 \pm 0.004}$	$\mathbf{0.007 \pm 0.003}$	$\mathbf{0.580 \pm 0.302}$
XGBOOST	$\mathbf{0.009 \pm 0.000}$	$\mathbf{0.008 \pm 0.000}$	0.834 ± 0.000

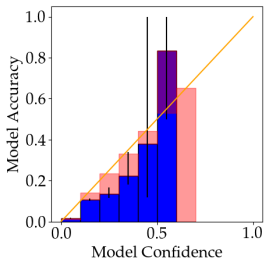
Table F.9: Calibration metrics

observation-rate	AUROC	AUPRC
TCN	0.763 ± 0.008	0.083 ± 0.001
LSTM	0.775 ± 0.005	0.084 ± 0.003
BRITS	0.759 ± 0.007	0.097 ± 0.005
GRUD	0.767 ± 0.007	0.084 ± 0.003
XGBOOST	$\mathbf{0.817 \pm 0.000}$	$\mathbf{0.110 \pm 0.000}$

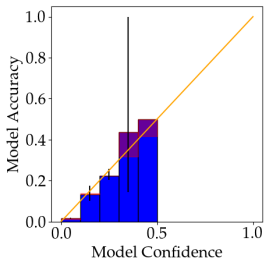
Table F.10: Performance metrics



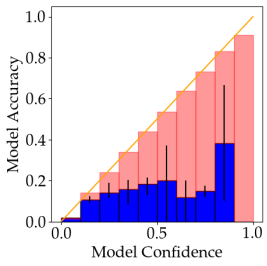
(a) reliability-diagram-dp-observation-rate-TCN



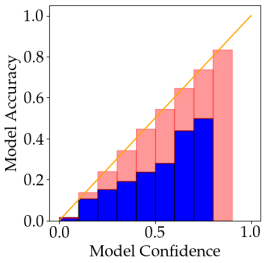
(b) reliability-diagram-dp-observation-rate-LSTM



(c) reliability-diagram-dp-observation-rate-BRITS

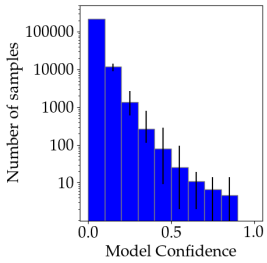


(d) reliability-diagram-dp-observation-rate-GRUD

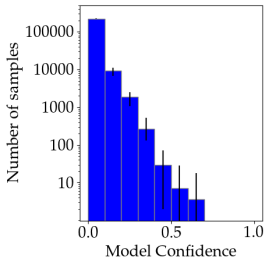


(e) reliability-diagram-dp-observation-rate-XGBOOST

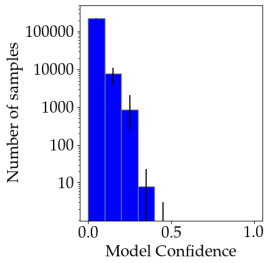
Figure F.21: Reliability diagrams for the experiment.



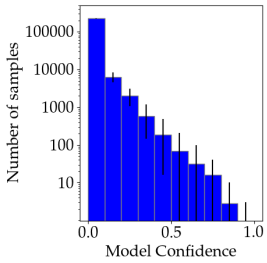
(a) sample-confidence-distribution-graph-dp-observation-rate-TCN



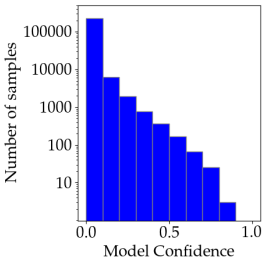
(b) sample-confidence-distribution-graph-dp-observation-rate-LSTM



(c) sample-confidence-distribution-graph-dp-observation-rate-BRITS

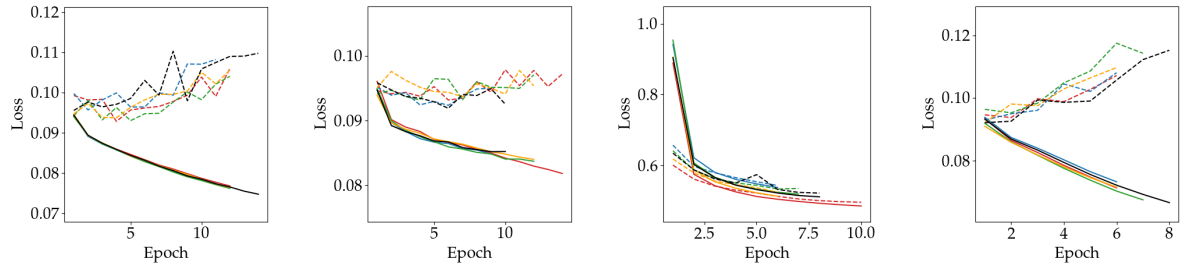


(d) sample-confidence-distribution-graph-dp-observation-rate-GRUD



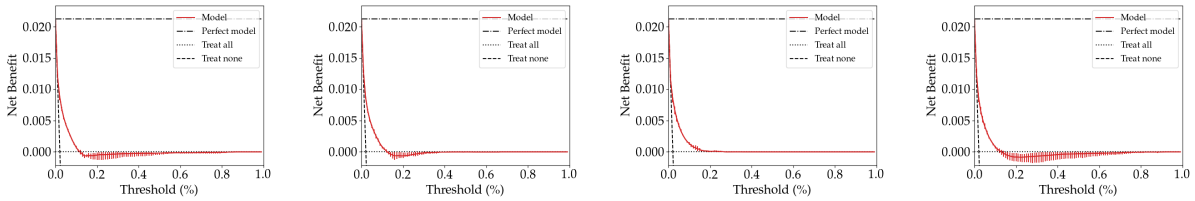
(e) sample-confidence-distribution-graph-dp-observation-rate-XGBOOST

Figure F.22: Sample confidence distribution graph for the experiment.

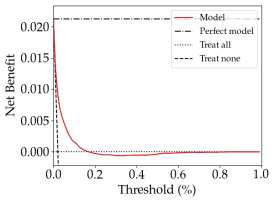


(a) history-graph-dp-observation-rate-TCN **(b)** history-graph-dp-observation-rate-LSTM **(c)** history-graph-dp-observation-rate-BRITS **(d)** history-graph-dp-observation-rate-GRUD

Figure F.23: Training and validation loss for each epoch for the experiment.

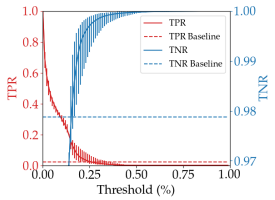


(a) dc-graph-dp-observation-rate-TCN **(b)** dc-graph-dp-observation-rate-LSTM **(c)** dc-graph-dp-observation-rate-BRITS **(d)** dc-graph-dp-observation-rate-GRUD

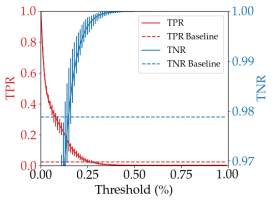


(e) dc-graph-dp-observation-rate-XGBOOST

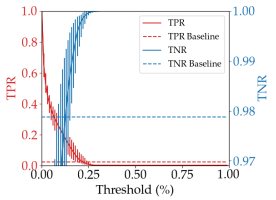
Figure F.24: DC.



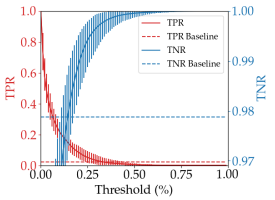
(a) TPRTNR-graph-dp-observation-rate-TCN



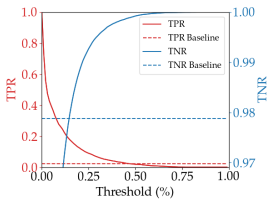
(b) TPRTNR-graph-dp-observation-rate-LSTM



(c) TPRTNR-graph-dp-observation-rate-BRITS



(d) TPRTNR-graph-dp-observation-rate-GRUD



(e) TPRTNR-graph-dp-observation-rate-XGBOOST

Figure F.25: TPR and TNR.

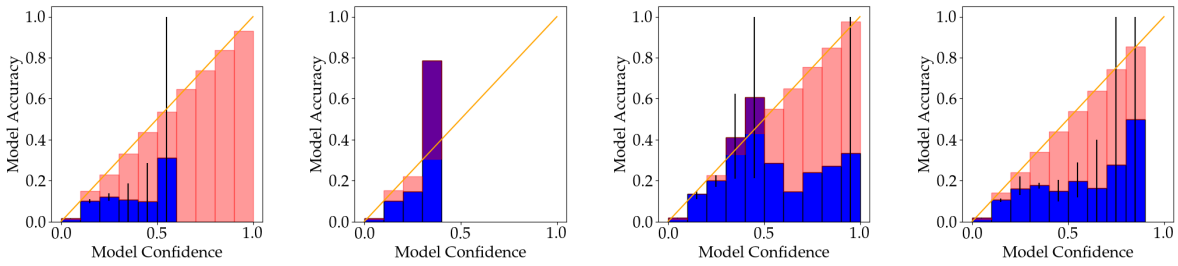
F.6 Delta Experiment

delta	ECE	ACE	MCE
TCN	**0.008 ± 0.002**	0.009 ± 0.001	**0.702 ± 0.240**
LSTM	0.009 ± 0.000	**0.007 ± 0.000**	**0.480 ± 0.000**
BRITS	**0.008 ± 0.001**	**0.008 ± 0.001**	**0.638 ± 0.495**
GRUD	**0.009 ± 0.001**	**0.008 ± 0.001**	**0.667 ± 0.255**

Table F.11: Calibration metrics

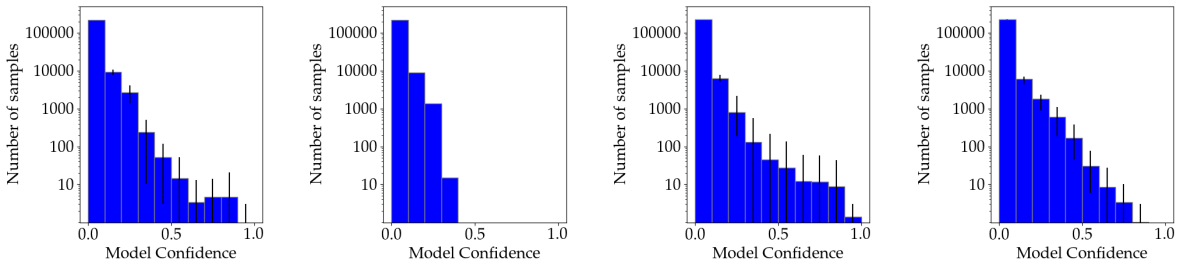
delta	AUROC	AUPRC
TCN	**0.755 ± 0.012**	0.073 ± 0.003
LSTM	0.748 ± 0.000	0.078 ± 0.000
BRITS	**0.768 ± 0.009**	**0.096 ± 0.003**
GRUD	**0.767 ± 0.009**	0.085 ± 0.003

Table F.12: Performance metrics



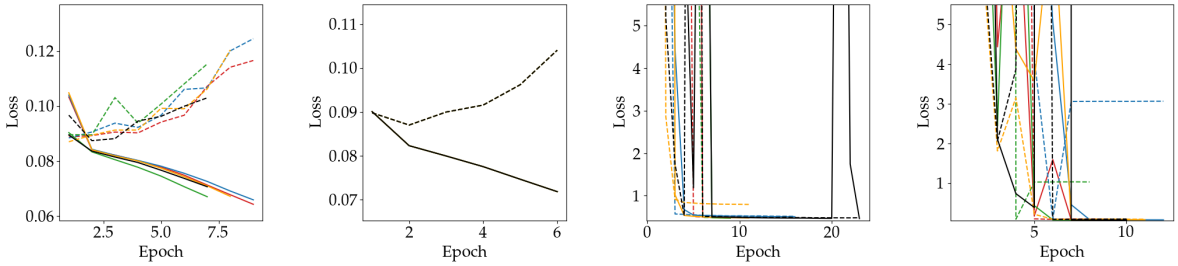
(a) reliability-diagram-dp-delta-TCN **(b)** reliability-diagram-dp-delta-LSTM **(c)** reliability-diagram-dp-delta-BRITS **(d)** reliability-diagram-dp-delta-GRUD

Figure F.26: Reliability diagrams for the experiment.



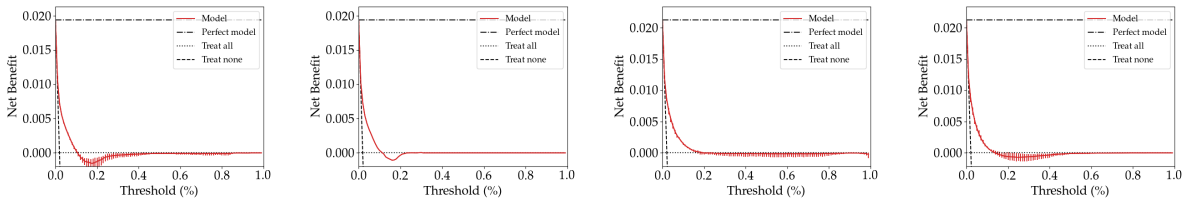
(a) sample-confidence-distribution-graph-dp-delta-TCN **(b)** sample-confidence-distribution-graph-dp-delta-LSTM **(c)** sample-confidence-distribution-graph-dp-delta-BRITS **(d)** sample-confidence-distribution-graph-dp-delta-GRUD

Figure F.27: Sample confidence distribution graph for the experiment.



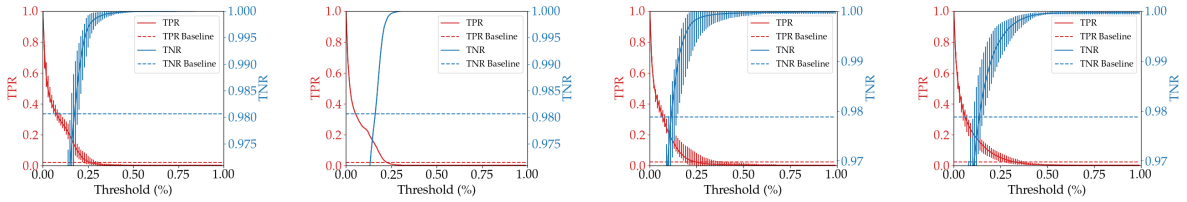
(a) history-graph-dp-delta-TCN **(b)** history-graph-dp-delta-LSTM **(c)** history-graph-dp-delta-BRITS **(d)** history-graph-dp-delta-GRUD

Figure F.28: Training and validation loss for each epoch for the experiment.



(a) dc-graph-dp-delta-TCN **(b)** dc-graph-dp-delta-LSTM **(c)** dc-graph-dp-delta-BRITS **(d)** dc-graph-dp-delta-GRUD

Figure F.29: DC.



(a) TPRTNR-graph-dp-delta-TCN **(b)** TPRTNR-graph-dp-delta-LSTM **(c)** TPRTNR-graph-dp-delta-BRITS **(d)** TPRTNR-graph-dp-delta-GRUD

Figure F.30: TPR and TNR.

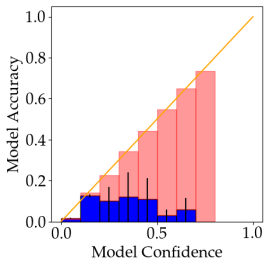
F.7 Final Experiment

final	ECE	ACE	MCE
TCN	**0.009 ± 0.003**	0.008 ± 0.002	**0.445 ± 0.333**
LSTM	**0.010 ± 0.002**	0.009 ± 0.002	**0.492 ± 0.289**
BRITS	**0.008 ± 0.002**	**0.007 ± 0.002**	**0.276 ± 0.379**
GRUD	**0.010 ± 0.002**	0.009 ± 0.002	**0.556 ± 0.315**
XGBOOST	**0.008 ± 0.000**	**0.005 ± 0.000**	**0.524 ± 0.000**

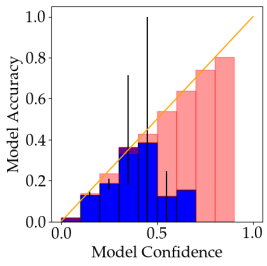
Table F.13: Calibration metrics

final	AUROC	AUPRC
TCN	0.754 ± 0.024	0.081 ± 0.009
LSTM	0.774 ± 0.010	0.098 ± 0.003
BRITS	0.777 ± 0.007	0.113 ± 0.007
GRUD	0.761 ± 0.011	0.093 ± 0.006
XGBOOST	**0.818 ± 0.000**	**0.122 ± 0.000**

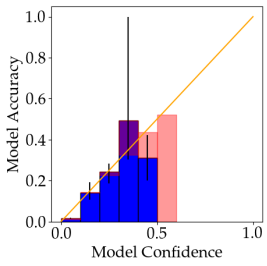
Table F.14: Performance metrics



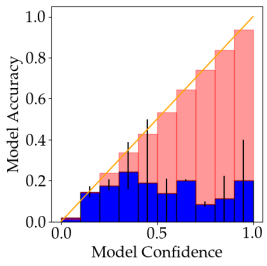
(a) reliability-diagram-dp-final-TCN



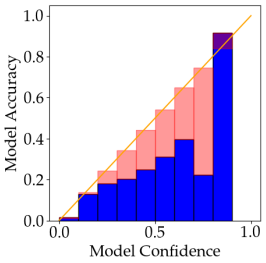
(b) reliability-diagram-dp-final-LSTM



(c) reliability-diagram-dp-final-BRITS

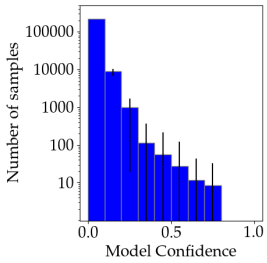


(d) reliability-diagram-dp-final-GRUD

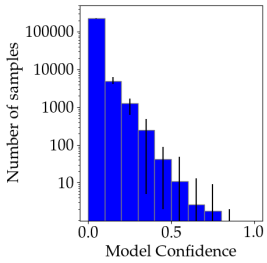


(e) reliability-diagram-dp-final-XGBOOST

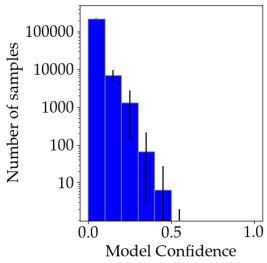
Figure F.31: Reliability diagrams for the experiment.



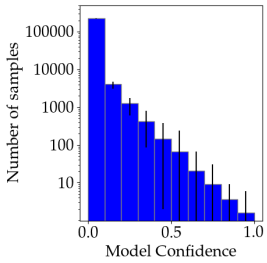
(a) sample-confidence-distribution-graph-dp-final-TCN



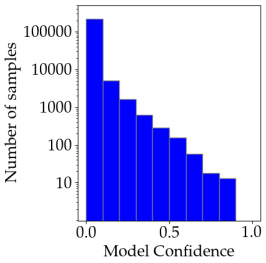
(b) sample-confidence-distribution-graph-dp-final-LSTM



(c) sample-confidence-distribution-graph-dp-final-BRITS

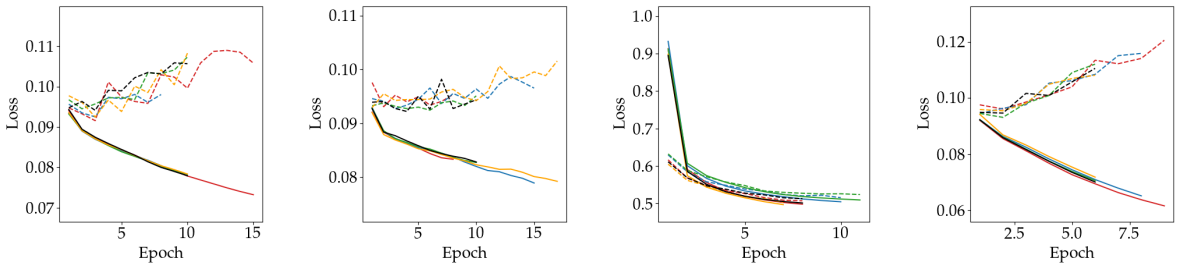


(d) sample-confidence-distribution-graph-dp-final-GRUD



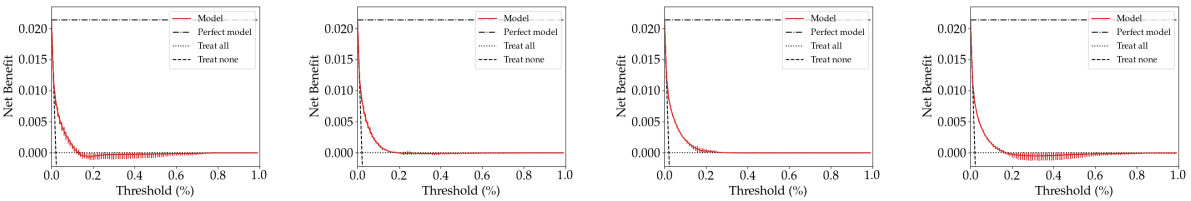
(e) sample-confidence-distribution-graph-dp-final-XGBOOST

Figure F.32: Sample confidence distribution graph for the experiment.

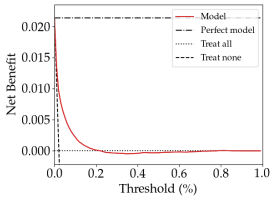


(a) history-graph-dp-final-TCN **(b)** history-graph-dp-final-LSTM **(c)** history-graph-dp-final-BRITS **(d)** history-graph-dp-final-GRUD

Figure F.33: Training and validation loss for each epoch for the experiment.

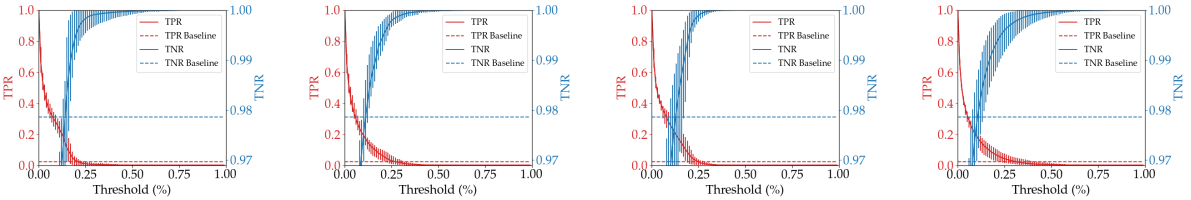


(a) dc-graph-dp-final-TCN **(b)** dc-graph-dp-final-LSTM **(c)** dc-graph-dp-final-BRITS **(d)** dc-graph-dp-final-GRUD

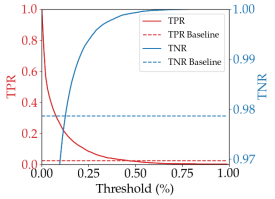


(e) dc-graph-dp-final-XGBOOST

Figure F.34: DC.



(a) TPRTNR-graph-dp-final-TCN **(b)** TPRTNR-graph-dp-final-LSTM **(c)** TPRTNR-graph-dp-final-BRITS **(d)** TPRTNR-graph-dp-final-GRUD



(e) TPRTNR-graph-dp-final-XGBOOST

Figure F.35: TPR and TNR.

Appendix G

Processed CROSS-TRACKS dataset

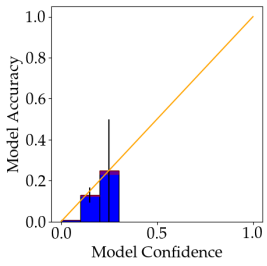
G.1 Baseline

baseline	ECE	ACE	MCE
TCN	**0.001 ± 0.000**	**0.001 ± 0.000**	**0.120 ± 0.155**
LSTM	**0.001 ± 0.001**	**0.001 ± 0.001**	**0.209 ± 0.475**
BRITS	**0.001 ± 0.001**	**0.002 ± 0.001**	**0.001 ± 0.001**
GRUD	0.001 ± 0.001	**0.002 ± 0.000**	0.408 ± 0.262
XGBOOST	**0.001 ± 0.000**	0.002 ± 0.000	0.565 ± 0.000

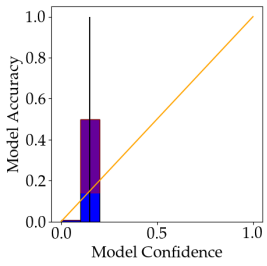
Table G.1: Calibration metrics

baseline	AUROC	AUPRC
TCN	0.750 ± 0.007	0.032 ± 0.006
LSTM	0.746 ± 0.011	0.029 ± 0.004
BRITS	0.687 ± 0.052	0.020 ± 0.002
GRUD	0.741 ± 0.011	0.025 ± 0.002
XGBOOST	**0.831 ± 0.000**	**0.140 ± 0.000**

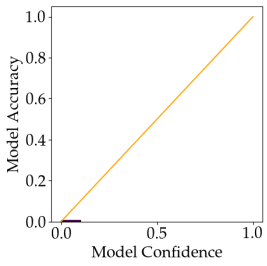
Table G.2: Performance metrics



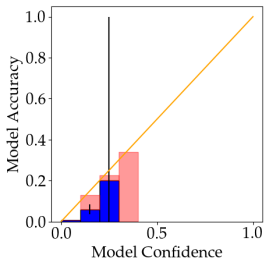
(a) reliability-diagram-dct-baseline-TCN



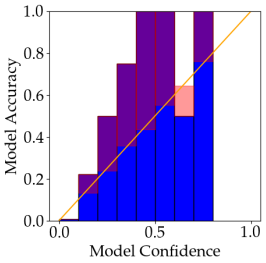
(b) reliability-diagram-dct-baseline-LSTM



(c) reliability-diagram-dct-baseline-BRITS

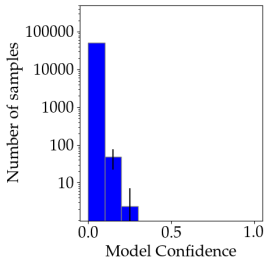


(d) reliability-diagram-dct-baseline-GRUD

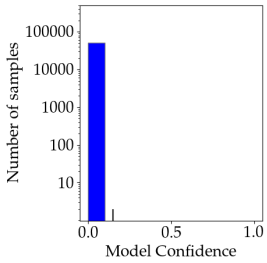


(e) reliability-diagram-dct-baseline-XGBOOST

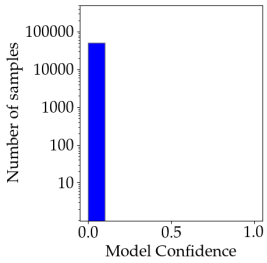
Figure G.1: Reliability diagrams for the experiment.



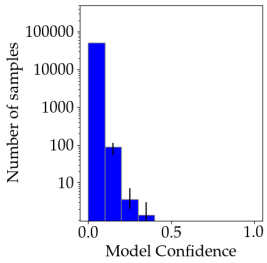
(a) sample-confidence-distribution-graph-dct-baseline-TCN



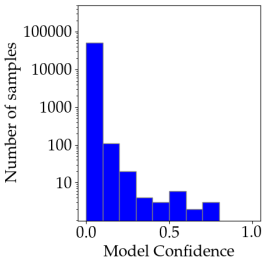
(b) sample-confidence-distribution-graph-dct-baseline-LSTM



(c) sample-confidence-distribution-graph-dct-baseline-BRITS

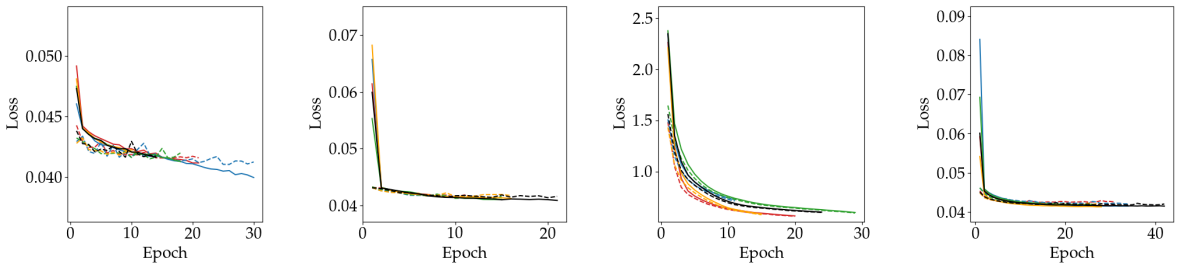


(d) sample-confidence-distribution-graph-dct-baseline-GRUD



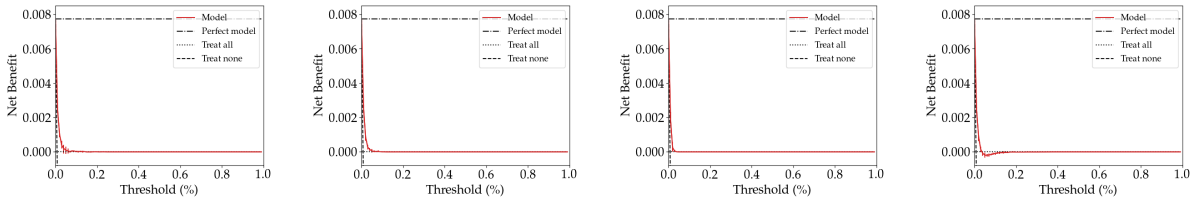
(e) sample-confidence-distribution-graph-dct-baseline-XGBOOST

Figure G.2: Sample confidence distribution graph for the experiment.

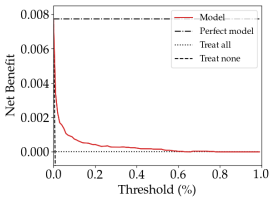


(a) history-graph-dct-baseline-TCN **(b)** history-graph-dct-baseline-LSTM **(c)** history-graph-dct-baseline-BRITS **(d)** history-graph-dct-baseline-GRUD

Figure G.3: Training and validation loss for each epoch for the experiment.

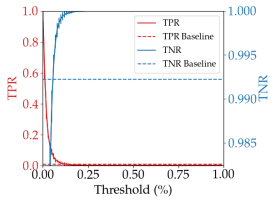


(a) dc-graph-dct-baseline-TCN **(b)** dc-graph-dct-baseline-LSTM **(c)** dc-graph-dct-baseline-BRITS **(d)** dc-graph-dct-baseline-GRUD

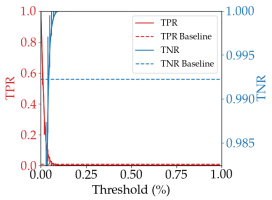


(e) dc-graph-dct-baseline-XGBOOST

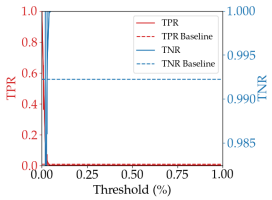
Figure G.4: DC.



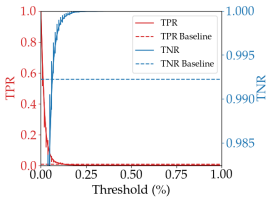
(a) TPRTNR-graph-dct-baseline-TCN



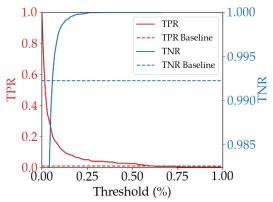
(b) TPRTNR-graph-dct-baseline-LSTM



(c) TPRTNR-graph-dct-baseline-BRITS



(d) TPRTNR-graph-dct-baseline-GRUD



(e) TPRTNR-graph-dct-baseline-XGBOOST

Figure G.5: TPR and TNR.

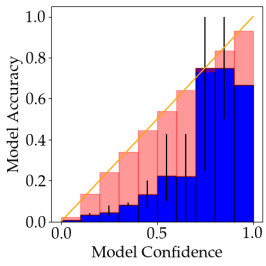
G.2 Class Weight Experiment

class-weight	ECE	ACE	MCE
4-TCN	0.020 ± 0.002	0.020 ± 0.002	0.449 ± 0.181
4-LSTM	0.023 ± 0.004	0.023 ± 0.004	$\mathbf{0.198 \pm 0.107}$
4-BRITS	0.024 ± 0.003	0.024 ± 0.003	$\mathbf{0.133 \pm 0.075}$
4-GRUD	0.023 ± 0.006	0.022 ± 0.006	0.588 ± 0.189
4-XGBOOST	$\mathbf{0.015 \pm 0.000}$	$\mathbf{0.015 \pm 0.000}$	0.246 ± 0.000
16-TCN	0.099 ± 0.010	0.099 ± 0.010	0.679 ± 0.172
16-LSTM	0.096 ± 0.011	0.096 ± 0.011	0.509 ± 0.091
16-BRITS	0.105 ± 0.030	0.105 ± 0.030	0.496 ± 0.054
16-GRUD	0.092 ± 0.009	0.092 ± 0.009	0.795 ± 0.132
16-XGBOOST	0.054 ± 0.000	0.054 ± 0.000	0.265 ± 0.000
128-TCN	0.394 ± 0.073	0.394 ± 0.073	0.881 ± 0.021
128-LSTM	0.390 ± 0.028	0.390 ± 0.028	0.809 ± 0.028
128-BRITS	0.403 ± 0.086	0.403 ± 0.086	0.842 ± 0.032
128-GRUD	0.367 ± 0.039	0.367 ± 0.039	0.891 ± 0.004
128-XGBOOST	0.216 ± 0.000	0.216 ± 0.000	0.678 ± 0.000

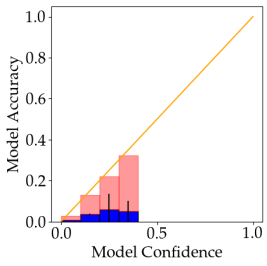
Table G.3: Calibration metrics

class-weight	AUROC	AUPRC
4-TCN	0.773 ± 0.003	0.047 ± 0.020
4-LSTM	0.763 ± 0.007	0.028 ± 0.003
4-BRITS	0.750 ± 0.029	0.023 ± 0.002
4-GRUD	0.759 ± 0.003	0.027 ± 0.001
4-XGBOOST	$\mathbf{0.844 \pm 0.000}$	0.191 ± 0.000
16-TCN	0.770 ± 0.014	0.033 ± 0.004
16-LSTM	0.768 ± 0.007	0.030 ± 0.001
16-BRITS	0.756 ± 0.004	0.024 ± 0.002
16-GRUD	0.767 ± 0.010	0.027 ± 0.002
16-XGBOOST	0.834 ± 0.000	$\mathbf{0.201 \pm 0.000}$
128-TCN	0.765 ± 0.012	0.029 ± 0.004
128-LSTM	0.771 ± 0.007	0.027 ± 0.001
128-BRITS	0.776 ± 0.006	0.030 ± 0.002
128-GRUD	0.768 ± 0.009	0.024 ± 0.003
128-XGBOOST	0.801 ± 0.000	0.136 ± 0.000

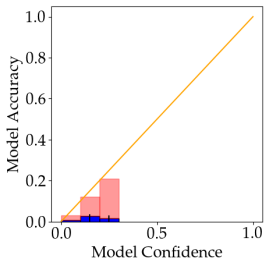
Table G.4: Performance metrics



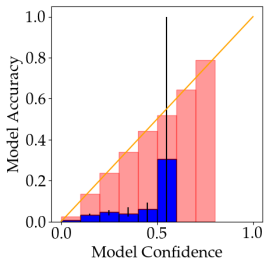
(a) reliability-diagram-dct-class-ratio-4-TCN



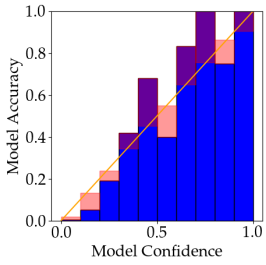
(b) reliability-diagram-dct-class-ratio-4-LSTM



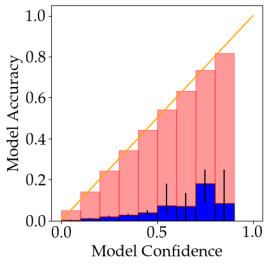
(c) reliability-diagram-dct-class-ratio-4-BRITS



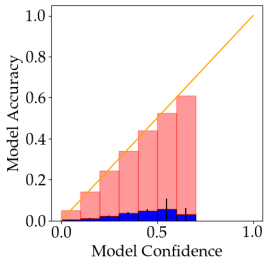
(d) reliability-diagram-dct-class-ratio-4-GRUD



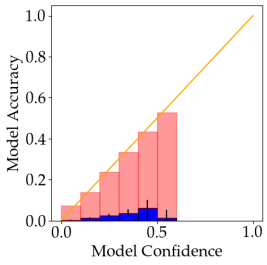
(e) reliability-diagram-dct-class-ratio-4-XGBOOST



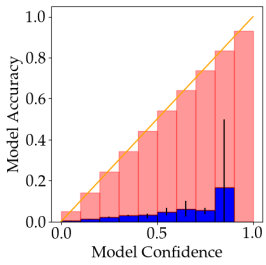
(f) reliability-diagram-dct-class-ratio-16-TCN



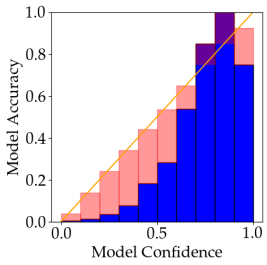
(g) reliability-diagram-dct-class-ratio-16-LSTM



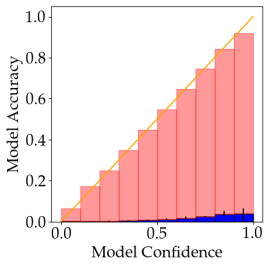
(h) reliability-diagram-dct-class-ratio-16-BRITS



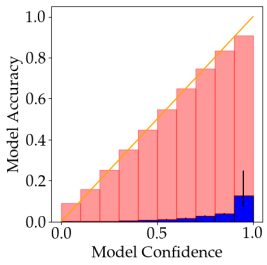
(i) reliability-diagram-dct-class-ratio-16-GRUD



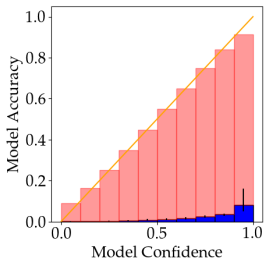
(j) reliability-diagram-dct-class-ratio-16-XGBOOST



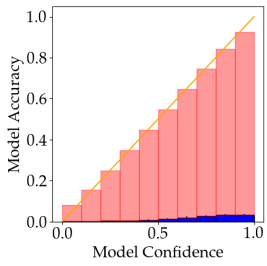
(k) reliability-diagram-dct-class-ratio-128-TCN



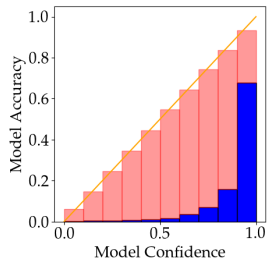
(l) reliability-diagram-dct-class-ratio-128-LSTM



(m) reliability-diagram-dct-class-ratio-128-BRITS

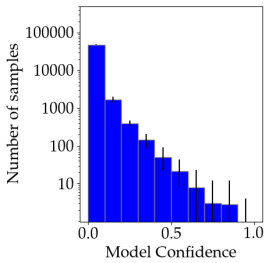


(n) reliability-diagram-dct-class-ratio-128-GRUD

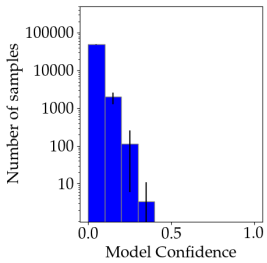


(o) reliability-diagram-dct-class-ratio-128-XGBOOST

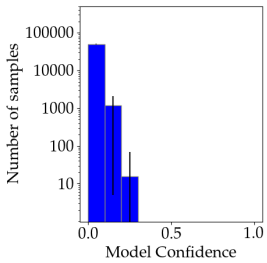
Figure G.6: Reliability diagrams for the experiment.



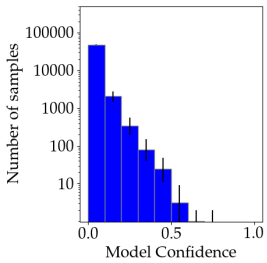
(a) sample-confidence-distribution-graph-dct-class-ratio-4-TCN



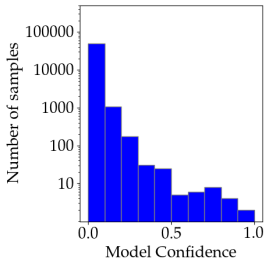
(b) sample-confidence-distribution-graph-dct-class-ratio-4-LSTM



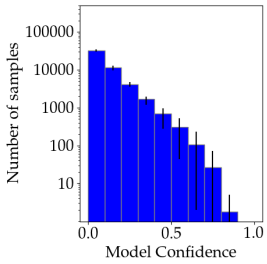
(c) sample-confidence-distribution-graph-dct-class-ratio-4-BRITS



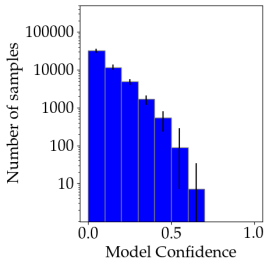
(d) sample-confidence-distribution-graph-dct-class-ratio-4-GRUD



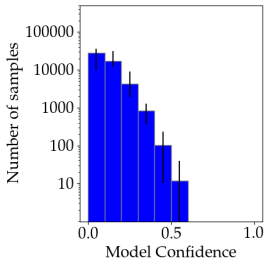
(e) sample-confidence-distribution-graph-dct-class-ratio-4-XGBOOST



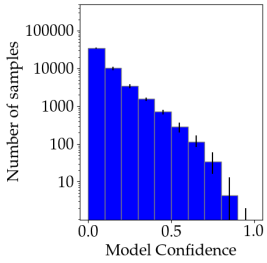
(f) sample-confidence-distribution-graph-dct-class-ratio-16-TCN



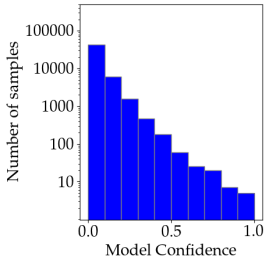
(g) sample-confidence-distribution-graph-dct-class-ratio-16-LSTM



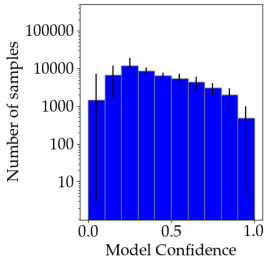
(h) sample-confidence-distribution-graph-dct-class-ratio-16-BRITS



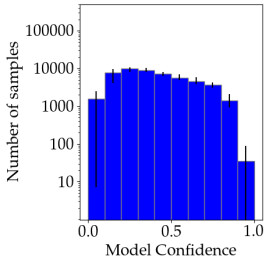
(i) sample-confidence-distribution-graph-dct-class-ratio-16-GRUD



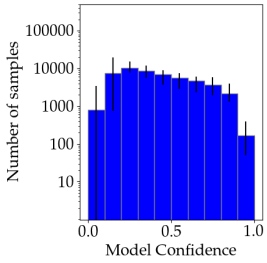
(j) sample-confidence-distribution-graph-dct-class-ratio-16-XGBOOST



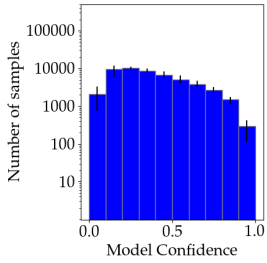
(k) sample-confidence-distribution-graph-dct-class-ratio-128-TCN



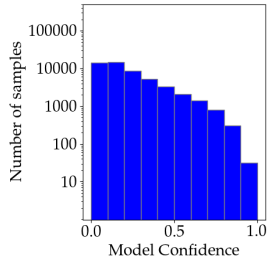
(l) sample-confidence-distribution-graph-dct-class-ratio-128-LSTM



(m) sample-confidence-distribution-graph-dct-class-ratio-128-BRITS

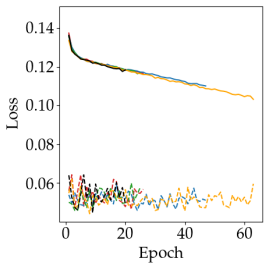


(n) sample-confidence-distribution-graph-dct-class-ratio-128-GRUD

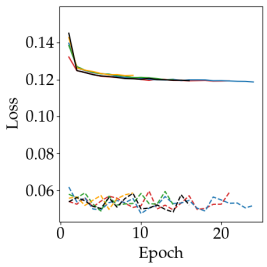


(o) sample-confidence-distribution-graph-dct-class-ratio-128-XGBOOST

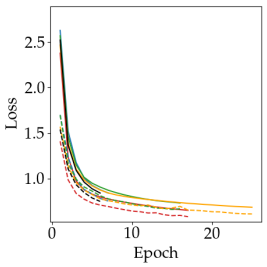
Figure G.7: Sample confidence distribution graph for the experiment.



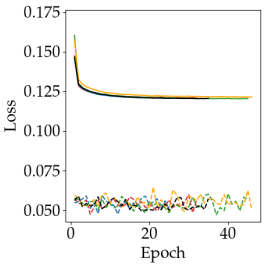
(a) history-graph-dct-class-ratio-4-TCN



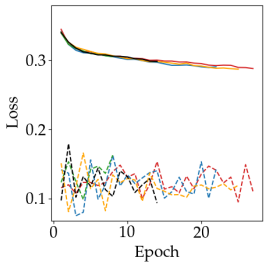
(b) history-graph-dct-class-ratio-4-LSTM



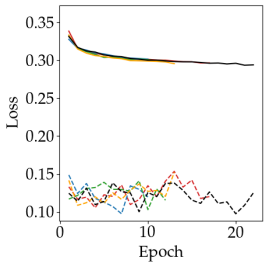
(c) history-graph-dct-class-ratio-4-BRITS



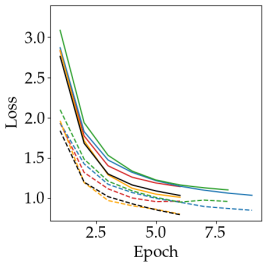
(d) history-graph-dct-class-ratio-4-GRUD



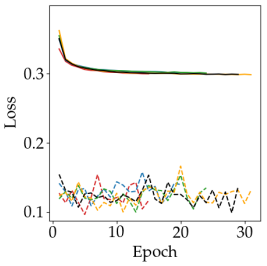
(e) history-graph-dct-class-ratio-16-TCN



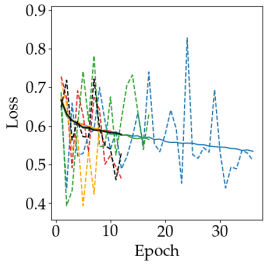
(f) history-graph-dct-class-ratio-16-LSTM



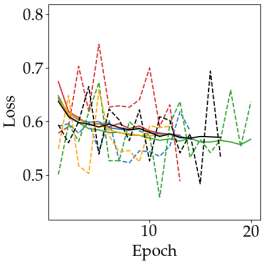
(g) history-graph-dct-class-ratio-16-BRITS



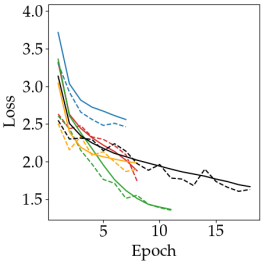
(h) history-graph-dct-class-ratio-16-GRUD



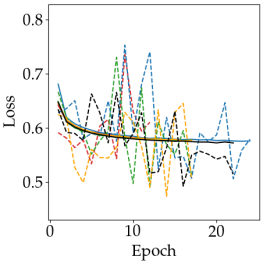
(i) history-graph-dct-class-ratio-128-TCN



(j) history-graph-dct-class-ratio-128-LSTM

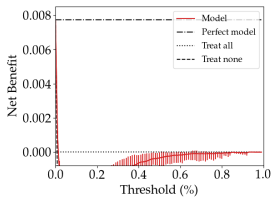


(k) history-graph-dct-class-ratio-128-BRITS

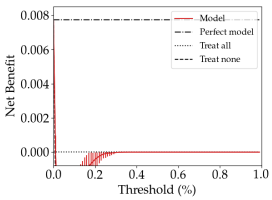


(l) history-graph-dct-class-ratio-128-GRUD

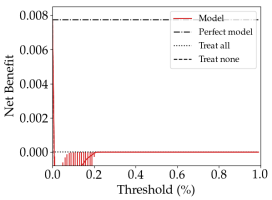
Figure G.8: Training and validation loss for each epoch for the experiment.



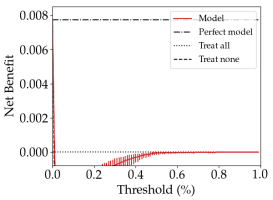
(a) dc-graph-dct-class-ratio-4-TCN



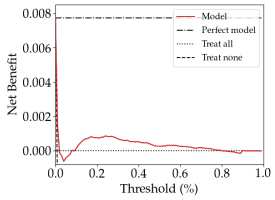
(b) dc-graph-dct-class-ratio-4-LSTM



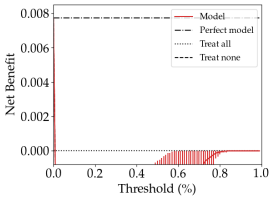
(c) dc-graph-dct-class-ratio-4-BRITS



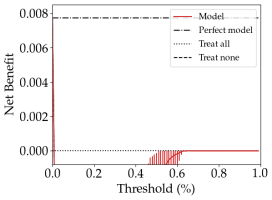
(d) dc-graph-dct-class-ratio-4-GRUD



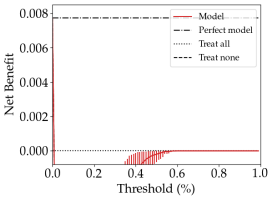
(e) dc-graph-dct-class-ratio-4-XGBOOST



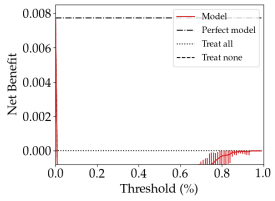
(f) dc-graph-dct-class-ratio-16-TCN



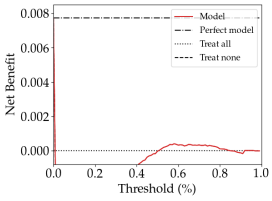
(g) dc-graph-dct-class-ratio-16-LSTM



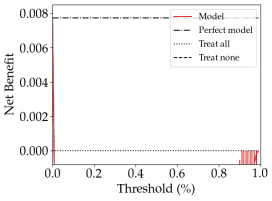
(h) dc-graph-dct-class-ratio-16-BRITS



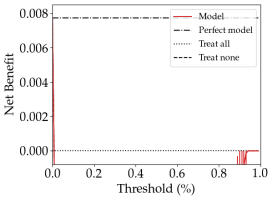
(i) dc-graph-dct-class-ratio-16-GRUD



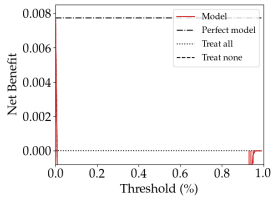
(j) dc-graph-dct-class-ratio-16-XGBOOST



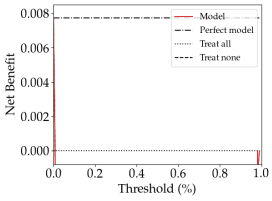
(k) dc-graph-dct-class-ratio-128-TCN



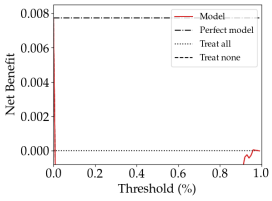
(l) dc-graph-dct-class-ratio-128-LSTM



(m) dc-graph-dct-class-ratio-128-BRITS

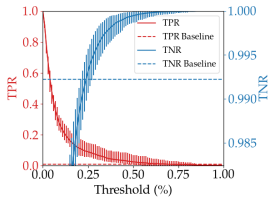


(n) dc-graph-dct-class-ratio-128-GRUD

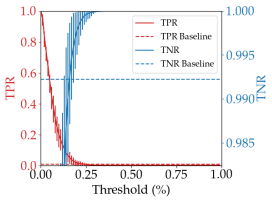


(o) dc-graph-dct-class-ratio-128-XGBOOST

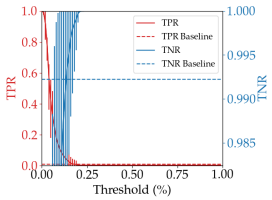
Figure G.9: DC.



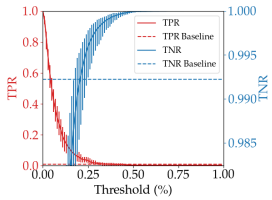
(a) TPRTNR-graph-dct-class-ratio-4-TCN



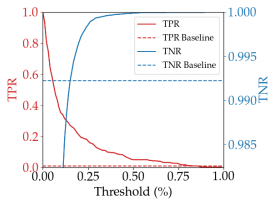
(b) TPRTNR-graph-dct-class-ratio-4-LSTM



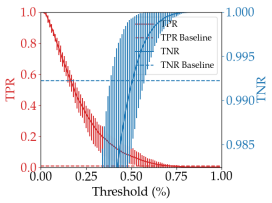
(c) TPRTNR-graph-dct-class-ratio-4-BRITS



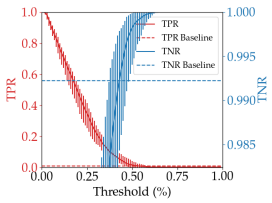
(d) TPRTNR-graph-dct-class-ratio-4-GRUD



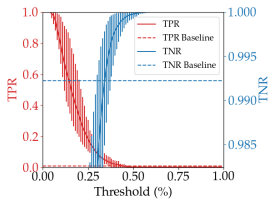
(e) TPRTNR-graph-dct-class-ratio-4-XGBOOST



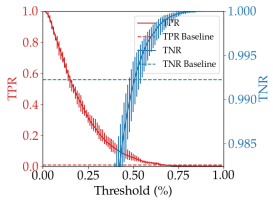
(f) TPRTNR-graph-dct-class-ratio-16-TCN



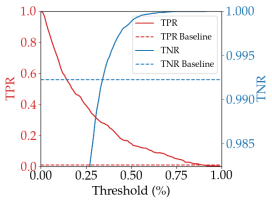
(g) TPRTNR-graph-dct-class-ratio-16-LSTM



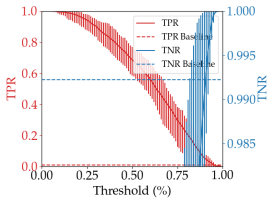
(h) TPRTNR-graph-dct-class-ratio-16-BRITS



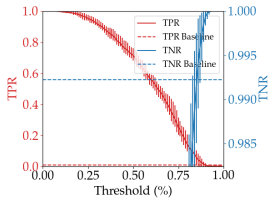
(i) TPRTNR-graph-dct-class-ratio-16-GRUD



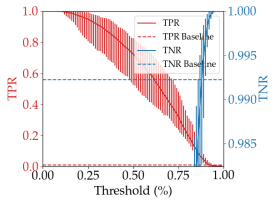
(j) TPRTNR-graph-dct-class-ratio-16-XGBOOST



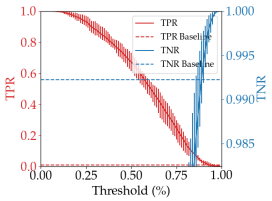
(k) TPRTNR-graph-dct-class-ratio-128-TCN



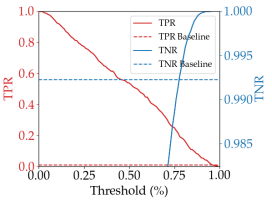
(l) TPRTNR-graph-dct-class-ratio-128-LSTM



(m) TPRTNR-graph-dct-class-ratio-128-BRITS



(n) TPRTNR-graph-dct-class-ratio-128-GRUD



(o) TPRTNR-graph-dct-class-ratio-128-XGBOOST

Figure G.10: TPR and TNR.

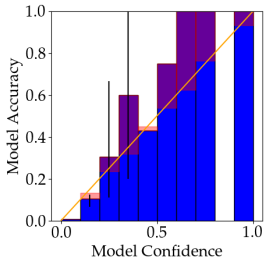
G.3 Missingness Data Representation Experiment

missingness-representation	ECE	ACE	MCE
TCN	$\mathbf{0.001 \pm 0.001}$	$\mathbf{0.001 \pm 0.001}$	$\mathbf{0.300 \pm 0.372}$
LSTM	$\mathbf{0.000 \pm 0.000}$	$\mathbf{0.001 \pm 0.000}$	$\mathbf{0.260 \pm 0.449}$

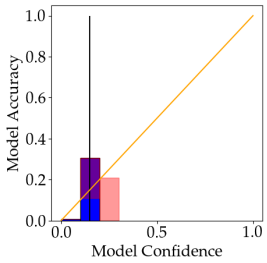
Table G.5: Calibration metrics

missingness-representation	AUROC	AUPRC
TCN	$\mathbf{0.747 \pm 0.014}$	$\mathbf{0.040 \pm 0.020}$
LSTM	$\mathbf{0.758 \pm 0.006}$	$\mathbf{0.031 \pm 0.003}$

Table G.6: Performance metrics

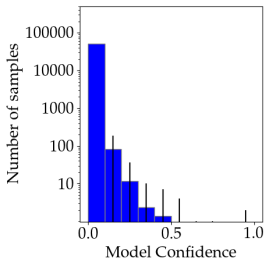


(a) reliability-diagram-dct-missingness-representation-TCN

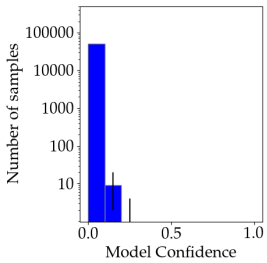


(b) reliability-diagram-dct-missingness-representation-LSTM

Figure G.11: Reliability diagrams for the experiment.

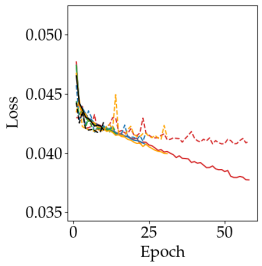


(a) sample-confidence-distribution-graph-dct-missingness-representation-TCN

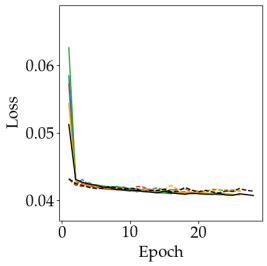


(b) sample-confidence-distribution-graph-dct-missingness-representation-LSTM

Figure G.12: Sample confidence distribution graph for the experiment.

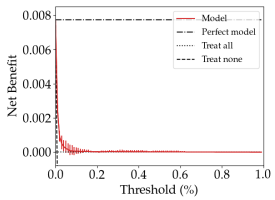


(a) history-graph-dct-missingness-representation-TCN

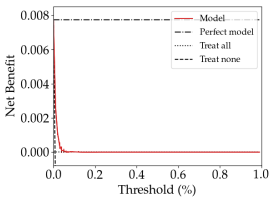


(b) history-graph-dct-missingness-representation-LSTM

Figure G.13: Training and validation loss for each epoch for the experiment.

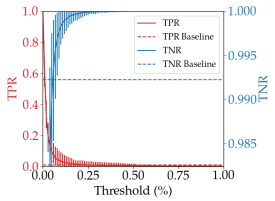


(a) dc-graph-dct-missingness-representation-TCN

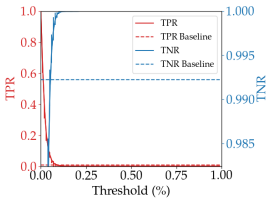


(b) dc-graph-dct-missingness-representation-LSTM

Figure G.14: DC.



(a) TPRTNR-graph-dct-missingness-representation-TCN



(b) TPRTNR-graph-dct-missingness-representation-LSTM

Figure G.15: TPR and TNR.

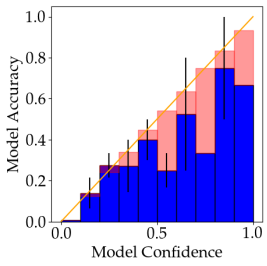
G.4 Observation Rate Experiment

observation-rate	ECE	ACE	MCE
TCN	**0.001 ± 0.001**	**0.001 ± 0.000**	**0.168 ± 0.189**
LSTM	**0.001 ± 0.001**	**0.001 ± 0.001**	**0.229 ± 0.411**
BRITS	0.003 ± 0.002	**0.003 ± 0.002**	**0.299 ± 0.139**
GRUD	0.002 ± 0.001	0.004 ± 0.001	**0.347 ± 0.112**
XGBOOST	**0.000 ± 0.000**	**0.001 ± 0.000**	0.441 ± 0.000

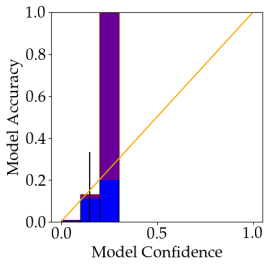
Table G.7: Calibration metrics

observation-rate	AUROC	AUPRC
TCN	0.745 ± 0.014	0.047 ± 0.031
LSTM	0.746 ± 0.008	0.030 ± 0.002
BRITS	0.733 ± 0.007	0.034 ± 0.007
GRUD	0.735 ± 0.016	0.126 ± 0.012
XGBOOST	**0.829 ± 0.000**	**0.142 ± 0.000**

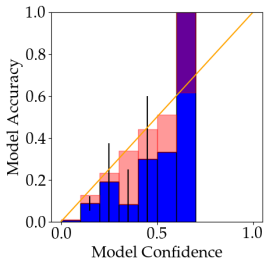
Table G.8: Performance metrics



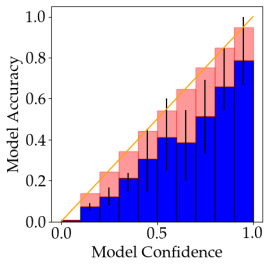
(a) reliability-diagram-dct-observation-rate-TCN



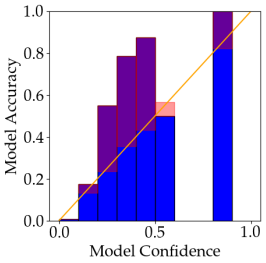
(b) reliability-diagram-dct-observation-rate-LSTM



(c) reliability-diagram-dct-observation-rate-BRITS

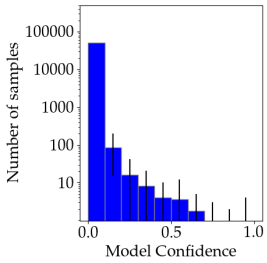


(d) reliability-diagram-dct-observation-rate-GRUD

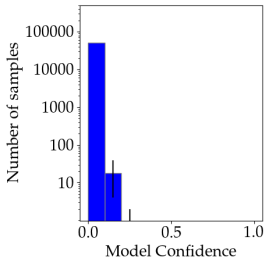


(e) reliability-diagram-dct-observation-rate-XGBOOST

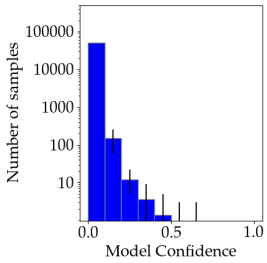
Figure G.16: Reliability diagrams for the experiment.



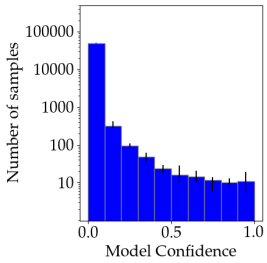
(a) sample-confidence-distribution-graph-dct-observation-rate-TCN



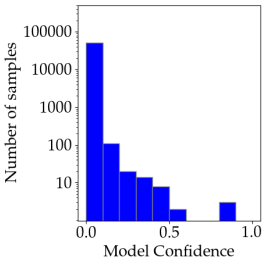
(b) sample-confidence-distribution-graph-dct-observation-rate-LSTM



(c) sample-confidence-distribution-graph-dct-observation-rate-BRITS

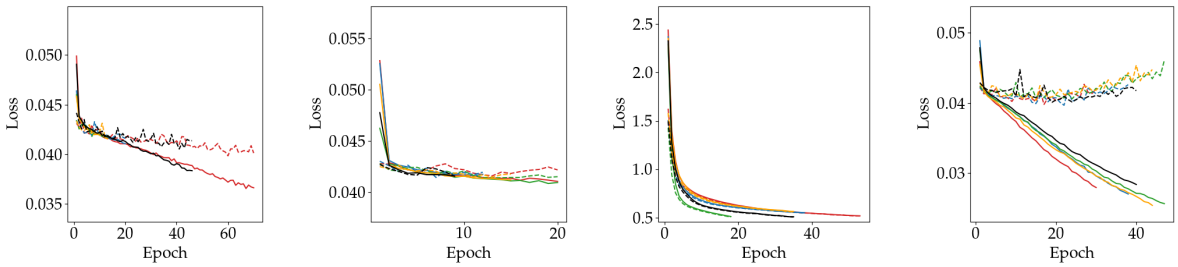


(d) sample-confidence-distribution-graph-dct-observation-rate-GRUD



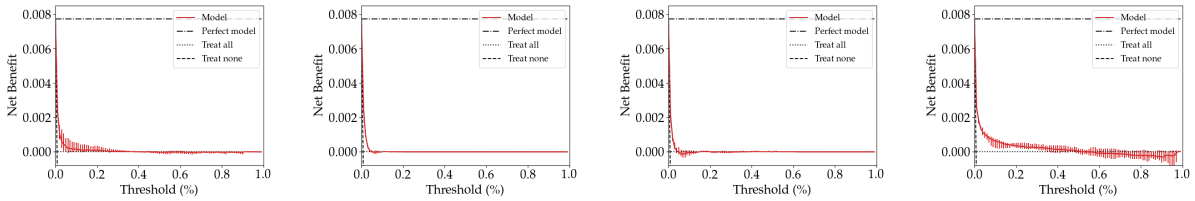
(e) sample-confidence-distribution-graph-dct-observation-rate-XGBOOST

Figure G.17: Sample confidence distribution graph for the experiment.

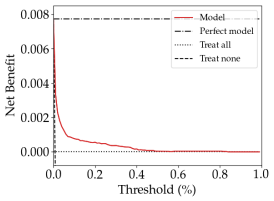


(a) history-graph-dct-observation-rate-TCN **(b)** history-graph-dct-observation-rate-LSTM **(c)** history-graph-dct-observation-rate-BRITS **(d)** history-graph-dct-observation-rate-GRUD

Figure G.18: Training and validation loss for each epoch for the experiment.

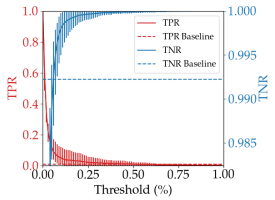


(a) dc-graph-dct-observation-rate-TCN **(b)** dc-graph-dct-observation-rate-LSTM **(c)** dc-graph-dct-observation-rate-BRITS **(d)** dc-graph-dct-observation-rate-GRUD

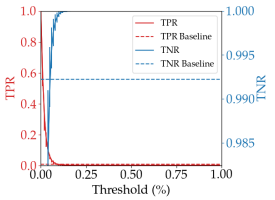


(e) dc-graph-dct-observation-rate-XGBOOST

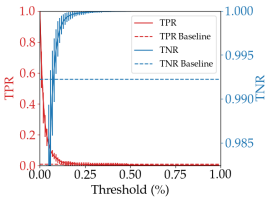
Figure G.19: DC.



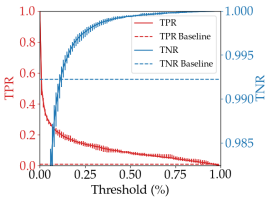
(a) TPRTNR-graph-dct-observation-rate-TCN



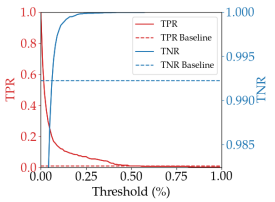
(b) TPRTNR-graph-dct-observation-rate-LSTM



(c) TPRTNR-graph-dct-observation-rate-BRITS



(d) TPRTNR-graph-dct-observation-rate-GRUD



(e) TPRTNR-graph-dct-observation-rate-XGBOOST

Figure G.20: TPR and TNR.

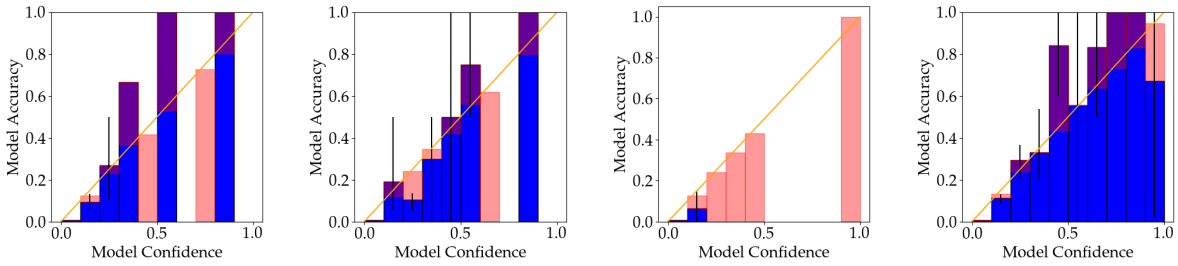
G.5 Delta Experiment

delta	ECE	ACE	MCE
TCN	**0.001 ± 0.001**	**0.002 ± 0.001**	**0.215 ± 0.379**
LSTM	**0.002 ± 0.002**	**0.002 ± 0.002**	**0.431 ± 0.219**
BRITS	**0.001 ± 0.001**	**0.001 ± 0.001**	**0.448 ± 0.396**
GRUD	**0.008 ± 0.021**	**0.009 ± 0.020**	**0.577 ± 0.293**

Table G.9: Calibration metrics

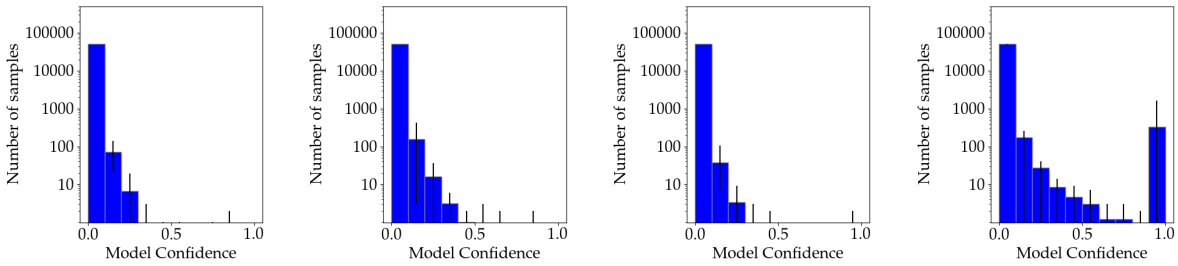
delta	AUROC	AUPRC
TCN	**0.740 ± 0.018**	**0.033 ± 0.004**
LSTM	**0.741 ± 0.014**	**0.034 ± 0.007**
BRITS	**0.726 ± 0.026**	**0.024 ± 0.004**
GRUD	**0.704 ± 0.127**	**0.078 ± 0.058**

Table G.10: Performance metrics



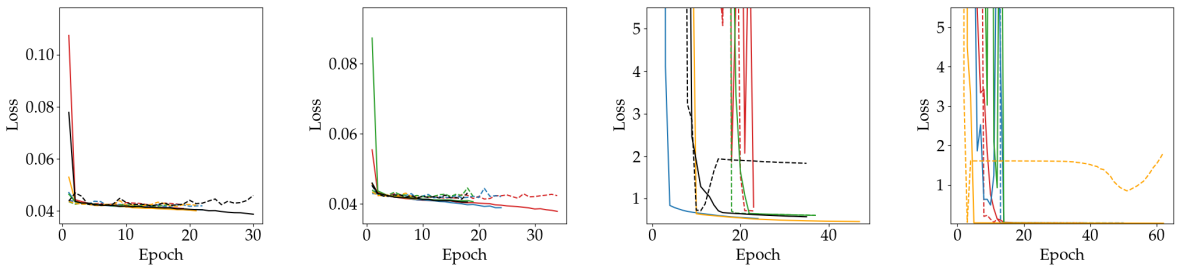
(a) reliability-diagram-dct-delta-TCN **(b)** reliability-diagram-dct-delta-LSTM **(c)** reliability-diagram-dct-delta-BRITS **(d)** reliability-diagram-dct-delta-GRUD

Figure G.21: Reliability diagrams for the experiment.



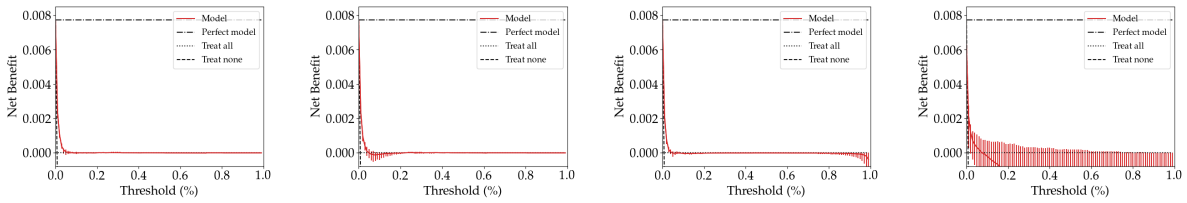
(a) sample-confidence-distribution-graph-dct-delta-TCN **(b)** sample-confidence-distribution-graph-dct-delta-LSTM **(c)** sample-confidence-distribution-graph-dct-delta-BRITS **(d)** sample-confidence-distribution-graph-dct-delta-GRUD

Figure G.22: Sample confidence distribution graph for the experiment.



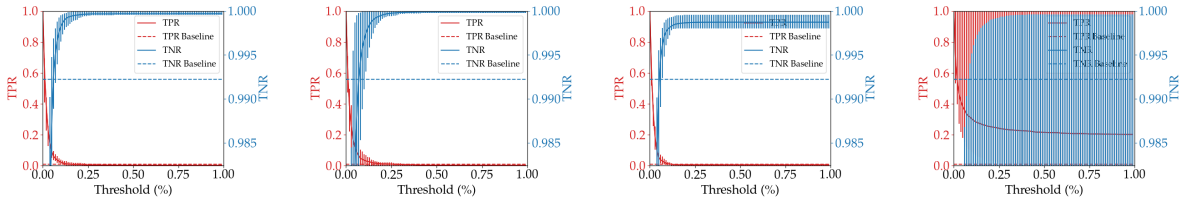
(a) history-graph-dct-delta-TCN **(b)** history-graph-dct-delta-LSTM **(c)** history-graph-dct-delta-BRITS **(d)** history-graph-dct-delta-GRUD

Figure G.23: Training and validation loss for each epoch for the experiment.



(a) dc-graph-dct-delta-TCN **(b)** dc-graph-dct-delta-LSTM **(c)** dc-graph-dct-delta-BRITS **(d)** dc-graph-dct-delta-GRUD

Figure G.24: DC.



(a) TPRTNR-graph-dct-delta-TCN **(b)** TPRTNR-graph-dct-delta-LSTM **(c)** TPRTNR-graph-dct-delta-BRITS **(d)** TPRTNR-graph-dct-delta-GRUD

Figure G.25: TPR and TNR.

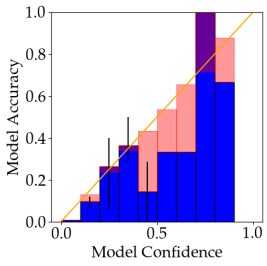
G.6 Final Experiment

final	ECE	ACE	MCE
TCN	**0.002 ± 0.002**	0.002 ± 0.001	**0.222 ± 0.183**
LSTM	**0.001 ± 0.001**	**0.001 ± 0.001**	**0.122 ± 0.103**
BRITS	**0.001 ± 0.001**	0.001 ± 0.000	**0.165 ± 0.159**
GRUD	0.002 ± 0.001	0.004 ± 0.001	0.356 ± 0.093
XGBOOST	**0.001 ± 0.000**	**0.001 ± 0.000**	0.654 ± 0.000

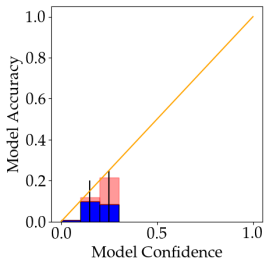
Table G.11: Calibration metrics

final	AUROC	AUPRC
TCN	0.751 ± 0.014	0.040 ± 0.014
LSTM	0.743 ± 0.010	0.031 ± 0.006
BRITS	0.748 ± 0.009	0.028 ± 0.001
GRUD	0.730 ± 0.010	0.102 ± 0.008
XGBOOST	**0.820 ± 0.000**	**0.131 ± 0.000**

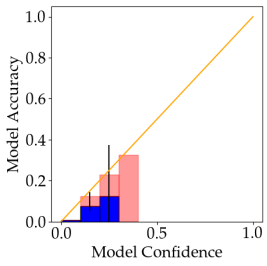
Table G.12: Performance metrics



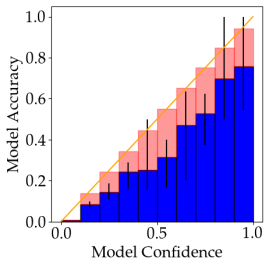
(a) reliability-diagram-dct-final-TCN



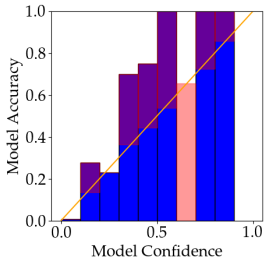
(b) reliability-diagram-dct-final-LSTM



(c) reliability-diagram-dct-final-BRITS

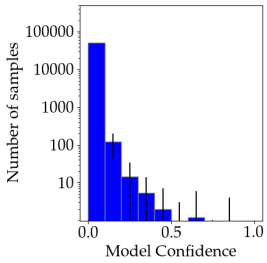


(d) reliability-diagram-dct-final-GRUD

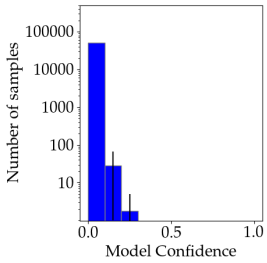


(e) reliability-diagram-dct-final-XGBOOST

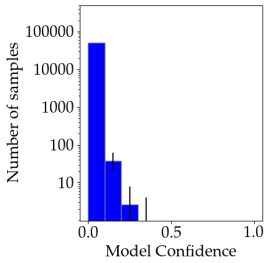
Figure G.26: Reliability diagrams for the experiment.



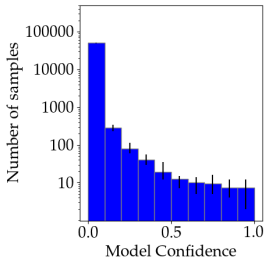
(a) sample-confidence-distribution-graph-dct-final-TCN



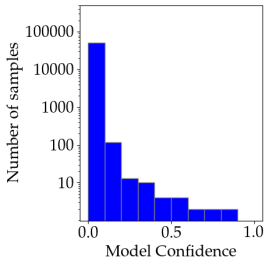
(b) sample-confidence-distribution-graph-dct-final-LSTM



(c) sample-confidence-distribution-graph-dct-final-BRITS

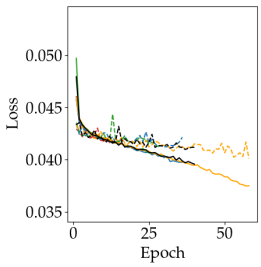


(d) sample-confidence-distribution-graph-dct-final-GRUD

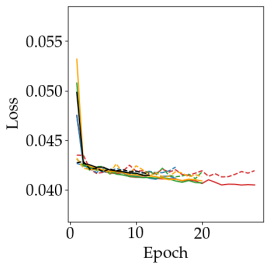


(e) sample-confidence-distribution-graph-dct-final-XGBOOST

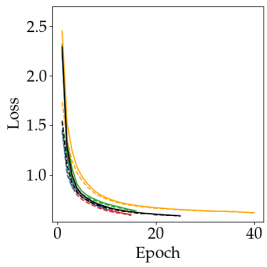
Figure G.27: Sample confidence distribution graph for the experiment.



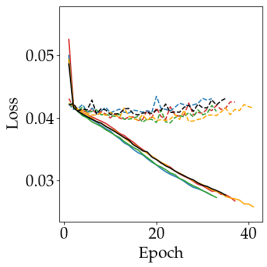
(a) history-graph-dct-final-TCN



(b) history-graph-dct-final-LSTM

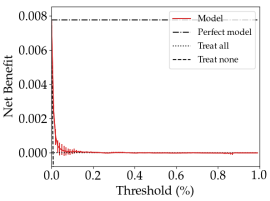


(c) history-graph-dct-final-BRITS

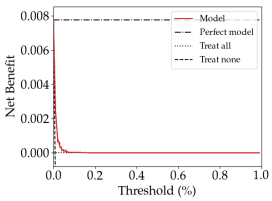


(d) history-graph-dct-final-GRUD

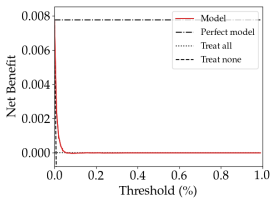
Figure G.28: Training and validation loss for each epoch for the experiment.



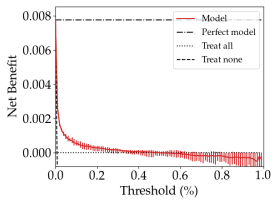
(a) dc-graph-dct-final-TCN



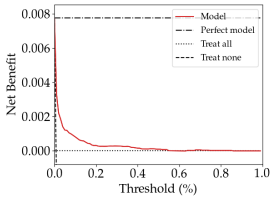
(b) dc-graph-dct-final-LSTM



(c) dc-graph-dct-final-BRITS

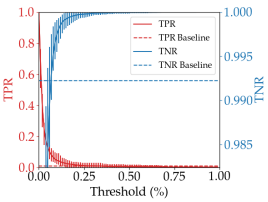


(d) dc-graph-dct-final-GRUD

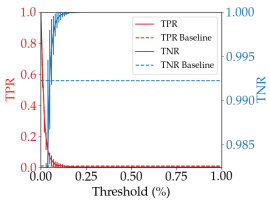


(e) dc-graph-dct-final-XGBOOST

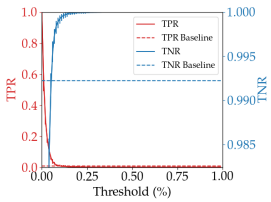
Figure G.29: DC.



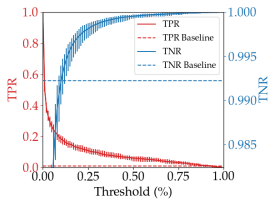
(a) TPRTNR-graph-dct-final-TCN



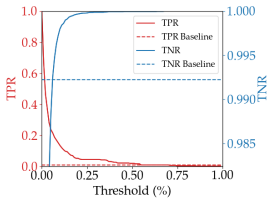
(b) TPRTNR-graph-dct-final-LSTM



(c) TPRTNR-graph-dct-final-BRITS



(d) TPRTNR-graph-dct-final-GRUD



(e) TPRTNR-graph-dct-final-XGBOOST

Figure G.30: TPR and TNR.