AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA INFORMATYKI

PRACA DYPLOMOWA MAGISTERSKA

Falls prediction with recurrent neural networks

Przewidywanie zasłabnięć z wykorzystaniem rekurencyjnych sieci neuronowych

Autor: *Marcin Radzio*
Kierunek studiów: *Informatyka*
Typ studiów: *Stacjonarne*
Opiekun pracy: *dr inż. Maciej Wielgosz*

Kraków, 2019

Oświadczenie studenta

Uprzedzony(-a) o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2018 r. poz. 1191 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony(-a) o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

……

(czytelny podpis studenta)

Contents

Abstract

Recurrent neural networks are widely used nowadays to analyze time series of different origins for a variety of reasons and purposes. However medicine does not implement these technologies often enough. Even though it could greatly benefit by deploying them, for instance as decision support or disease detection systems. The goal of this work is to explore efficiency of applying the state of the art GRU architecture in medical field. This research primarily focuses on a problem related to temporal data analysis, namely forecasting occurrences of syncope while using real-life cardiological time series. Nevertheless the devised methodology and architecture may be successfully applied to different types of tasks that involve signal processing or signal classification.

1. Introduction

This chapter is meant to explain why fainting among patients hospitalized for a long time is a serious health hazard. It provides the motivation behind predicting and early-preventing the collapse. Also the core of this work which is the development of a new method of syncope detection based on classification of cardiological time series with recurrent neural networks is going to be touched upon. At last the current state of the art will be summarized briefly to introduce the foundation of this research.

1.1. Motivation

Patients residing in hospitals are exposed to being weakened over time. Frequently they require further rehabilitation before they are strong and confident enough to leave. Whether the time has come is decided during the examination, during which patients tend to collapse. Oftentimes what brings them to this state is low physical activity and restrictive diet. It seems to be very common among the elders. Moreover upon fainting patients are exposed to stress which then results in lack of confidence and trust in the rehabilitation process. This mental strain causes the phenomenon of vicious cycle where patients exposed to experiencing syncope tend to stay in hospitals longer than needed or never recover at all. Often it could be avoided by simply predicting whether the examined person is likely to faint. In case of an early-detection the examination could be ended beforehand. Such an individual could be commissioned to further rehabilitation while not being exposed to stress associated with the collapse.

1.2. Goal

The purpose of this work is to develop a model based on recurrent neural networks that would be capable of forecasting occurrences of syncope by using real-life cardiological time series.

This project is realized in collaboration with Medical University of Graz and professor Nandu Gosowami, who provided the medical data used to train and evaluate future models. The designed methodology and system is supposed to be used at Medical University of Graz.

Created models should prove to be more accurate than methods based on naive classification approaches (SVMs, decision trees, etc.) or solutions dedicated to time series analysis (e.g. ARIMA). Moreover the provided real-life data ought to be processed minimally in order to guarantee that the model could consistently predict fainting during experiments performed live. The collapse ideally should be predicted ahead of time allowing the technician to interrupt the examination. Models should also be highly sensitive – patients who are certainly going to faint must be classified correctly, even at the cost of the overall accuracy.

1.3. State of the art

1.3.1. Recurrent neural networks

A recurrent neural network is a specific type of the neural networks that found its niche in problems where the input data has sequential nature (signals, time series, text flow etc.) - where the formerly introduced values provide the context for the later inputs. The main feature distinguishing it from a plain neural network is the existence of the hidden state which somewhat makes the network aware of its previous inputs.

A typical example of such a recurrent architecture is the Elman network [1]. It may be easily described in three mathematical equations:

$$h^{(0)} = 0$$

$$h^{(t)} = \sigma_h(W_h x^{(t)} + U_h h^{(t-1)} + b_h)$$

$$y^{(t)} = \sigma_y(W_y h^{(t)} + b_y)$$

where:
x – input vector
h – hidden vector (hidden state)
y – output vector
σ_h, σ_y – activation function
W_h, U_h, W_y, U_y – weight matrices
b_h, b_y – bias vectors

Nevertheless these networks are prone to the phenomenon of the vanishing and exploding gradient [2]. Because of it they are reluctant to be properly trained and oftentimes they may perform poorly unless these issues are addressed. Fortunately there exist architectures like LSTM (Long-Short Term Memory) [3] or GRU (ang. Gated Recurrent Unit) [4] that not only resolve the mentioned difficulties but also achieve better results in comparison to plain RNNs.

The GRU features simpler structure than the LSTM, yet they perform equally well [5]. Therefore it is the preferred architecture, for the reason alone that it has less parameters and as a result models may be trained more

efficiently.

A GRU network can be described as follows:

$$h^{(0)} = 0$$

$$z^{(t)} = \sigma(W_z x^{(t)} + U_z h^{(t-1)} + b_z)$$
$$r^{(t)} = \sigma(W_r x^{(t)} + U_r h^{(t-1)} + b_r)$$
$$s^{(t)} = tanh(W_s x^{(t)} + r^{(t)} \odot U_s h^{(t-1)} + b_s)$$
$$h^{(t)} = z^{(t)} \odot h^{(t-1)} + (1 - z^{(t)}) \odot s^{(t)}$$

where:
x – input vector
h – hidden vector (hidden state)
z – update gate state vector
r – reset gate state vector
s – current state vector
σ – sigmoid function
$tanh$ – hyperbolic tangent
$\odot$ – Hadamard product
$W_z, U_z, W_r, U_r, W_s, U_s$ – weight matrices
b_z, b_r, b_s – bias vectors

For many problems a single GRU cell may be insufficient and in response the need of deepening the network arises. There are a few types of deep RNNs [6], but the most popular of them all is uncompromisingly the stacked variant. It relies on vertical joining of the RNN cells, where every consecutive cell acts as a layer. This approach is especially important because the medical data may prove to be to hard for a shallow recurrent model.

The areas where RNN-based solutions are regularly applied:

- **text analysis:**
  - language modeling,
  - authorship verification,
  - sentiment analysis,
  - automatic translation;
- **sound analysis:**
  - music generation,
  - rhythm detection,
  - speech recognition;
- **image analysis:**

- gesture recognition;

- **time series analysis.**

The especially important scope of RNNs usage is the time series analysis in medical applications. Data like probed blood-pressure, heart rate or medical history have strictly sequential nature. In that case the RNNs can be used for forecasting or predicting their future state, which tightly complies with the research goals of this work.

1.3.2. Signal processing and restoration

Regardless of the selected architecture there always exists the problem of data representation. Signals that were not processed beforehand may cause models to perform poorly. This is especially relevant in case of medical time series that tend to be fuzzy or incomplete. These problems arise from the imperfections of the measuring devices, but also from human factor. Fortunately many methods of handling such a flawed data was developed over the years [7]. Various smoothing filters could be used in case of fuzzy or defective signals. In most cases median filter proves to be sufficient. Yet it still does not alleviate the problem of gaps and missing samples within the time series. This type of faulty signals can be often fixed with linear, polynomial or spline interpolation.

1.3.3. Small batch strategy

Whenever neural networks are trained using GPUs or CPUs there will always exist the trade-off of time and quality. In all cases training can be boosted thanks to usage of mini-batches. Nevertheless the recent studies prove that small mini-batch size can improve not only accuracy but also the generalization capabilities of trained models [8]. It seems that packing more than 32 inputs may be detrimental to overall quality of models. Whereas passing only 2 inputs at the same time seems to be optimal unless time limits are considered.

1.3.4. Bayesian optimization

Bayesian optimization is a stochastic approach of finding global optima of a black-box function [9]. It is strictly related to notion of uncertainty and uses Gaussian Processes to further model it. Basically it operates on two normal distributions: prior and posterior. While the former is the belief of the optimized function's shape, the later is the previous belief updated with new observations. The more observations are performed the less uncertain the function's shape happens to be. In order to avoid getting stuck in local optima an acquisition function is used. It trades off exploration and exploitation. Exploration allows drawing points from regions of high uncertainty whereas exploitation exploits the regions with the highest probability obtained from the posterior distribution.

The main loop of the Bayesian optimization algorithm can be summarized as follows:

1. Minimize acquisition function in order to obtain a candidate point x.
2. Evaluate $y = f(x)$.
3. Update current belief with (x, y) pair.

Neural networks usually need to be tuned in order to yield the best possible results. The most popular methods of hyperparameters searching involve either use of grid search or random search. The former is very expensive computationally and depending on the granularity it may produce nearly optimal tuning. On the other hand random

search is a cheap alternative, but there is no guarantee of obtaining the satisfactory results. Bayesian optimization introduces the factor of constant improvement. By storing and reusing observations it can efficiently locate potential points. Hence the computation time is highly reduced even though optimization yields satisfactory results.

1.3.5. Similar problems

The data provided for the sake of this research is quite unique and not publicly available. Therefore it is unlikely to find any other attempts of predicting collapse with machine learning in the general literature. Hence this work is meant to introduce a reliable approach to handling temporal medical data and provide a comprehensive overview of advantages resulting from using RNNs over other appropriate models like ARIMA (ang. Autoregressive Integrated Moving Average).

On the other hand neural networks have recently become a point of interest. Machine learning approaches proved to be extremely accurate in various prediction and classification tasks. Quite a few relevant researches exist involving use of medical temporal data in conjunction with neural network based models for automated diagnosis of medical conditions [10] and early detection of possible health complications [11]. But unlike this particular work they tend to use sparse time series.

2. Data analysis

This chapter describes the data, its features and the general methodology of processing it for later use. It also compares the variants of the undertaken procedures and mentions subtle difficulties that may arise while working with time series.

2.1. Overview

The data provided consisted of nearly 700 files. Each file was labeled either as "syncope" (indicating fainting during examination) or "no findings" which for simplicity will be addressed as "nosyncope" from now on. Also there were several measurements performed involving every patient. Upon further investigation it was demonstrated that most of them were strongly correlated and derivable from the remaining ones. In many cases various measurements were taken by the same devices as indicated by the identical gaps and pauses in the signals. After consulting these issues with the data donor it was decided to use only the two most important time series:

- `mBP` – mean blood pressure,
- `HR` – heart rate.

Selected data was still defective: incomplete, unbalanced and fuzzy. In a few cases data happened to be wrongly labeled or existed in "syncope" as well as in "nosyncope" class. These issues had to be resolved. The processed dataset then could be divided into training, test and validation sets. The specific steps of the data processing are explained below.

2.2. Data leakage

Even though the data was delivered in two separate archives, individual time series were wrongly classified or duplicated. For a long time this issue was not noticed. As a result the models were trained, evaluated and tested on erroneous data, which caused their low predictive capabilities and worse overall accuracy. The problem was eventually solved by fixing the labels and removing the redundant files.

2.3. Processing scheme

2.3.1. Signal trimming

First 500 and last 50 samples of every time series were trimmed. It is justified by the fact that most of the removed samples were associated with starting, tuning and turning off of the measuring devices. From the

perspective of this research they had no value and could be wasteful during the training phase.

After trimming was done, patients with signals shorter than 500 samples were dropped since at the later stages it would be hard to make any use of too short sequences. RNNs tend to improve with the window size hence if time series happened to be too short they could produce unsatisfactory results.

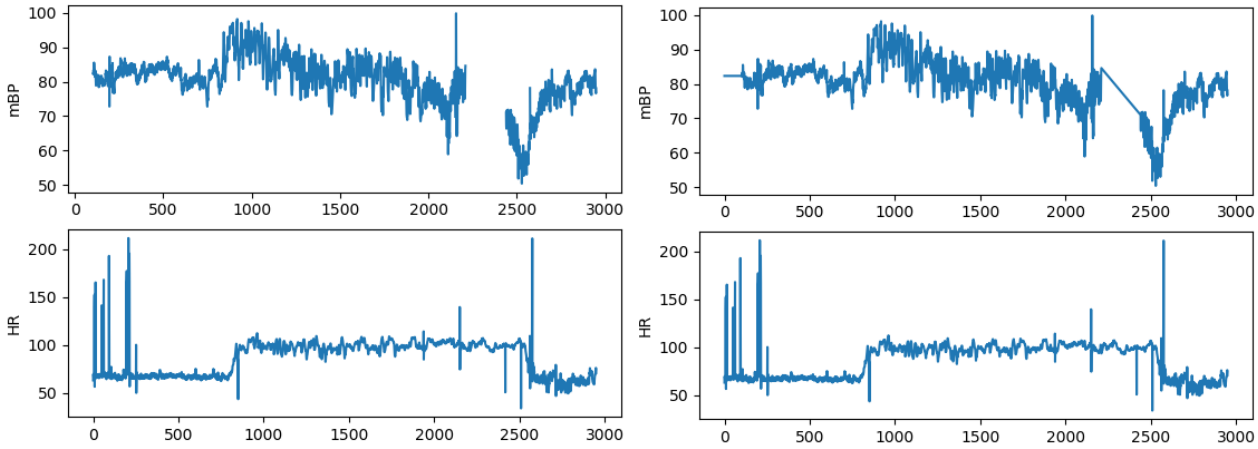

Figure 2.1: *Time series with visible discontinuities on the left and interpolated signals on the right.*

2.3.2. Missing samples restoration

The measurements were performed with various devices, whose working principle is not well-known. In many cases the times series were not continuous. Oftentimes individual samples or even whole regions were missing, indicating temporary failures of the measuring instruments. The gaps in the signals were also inconsistent. At this point every patient had two corresponding signals, which frequently either started or ended independently of one another.

In order to heal the faulty signals, they were treated as follows:

- the missing samples at the beginning of the time series were filled with the leftmost value, that is the first obtainable value of the signal,
- the missing samples at the end of the time series were filled with the rightmost value, that is the last obtainable value of the signal,
- the remaining missing samples were linearly interpolated by using the nearest values on the right and the left.

An example, where the time series start at different moments and a wide gap is visible, was shown in Figure 2.1. The previously mentioned figure also presents the same signals after the interpolation step.

2.3.3. Outliers removal

The obtained data in many cases contained multiple time series infested with outliers and overall fuzziness. These issues mostly stemmed from faulty measuring devices but also from the human factor involvement.

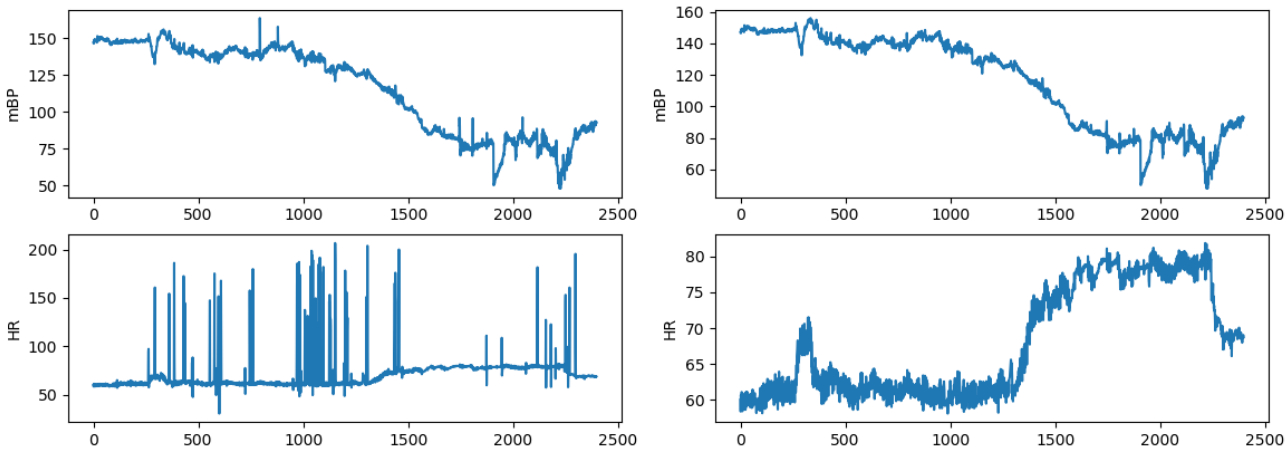

Figure 2.2: *Time series with visible outliers on the left. On the right the same signals after applying median filters.*

In order to improve the quality of the signals, while preserving their original character at the same time, no smoothing or averaging methods were applied. In most cases simple implementation of a median filter into processing flow was enough to get rid of the extreme outliers. Nevertheless such an approach happened to be insufficient in case of more complicated or noisy signals.

The final process used to combat the issues above involved performing multiple runs of a median filter on standardized draft copies of the data. The usage of standardization step allowed for enclosing the majority of initial time series in the range $[-3, 3]$. The exact formula used is presented below:

$$y = \frac{x - \mu}{\sigma}$$

where:
y – standardized sample
x – original sample
μ – average value of the signal
σ – standard deviation of the signal

The main idea was to perform multiple runs of the loop presented below with a limit that was harmonically decreased to detect smaller deviations with each iteration:

1. Copying the original signal.
2. Standardizing the copy.
3. Applying median window of size 31 to every sample in the copy.
4. Calculating the difference between the median signal and the copy.
5. Identifying the outliers by checking if the absolute value of the difference was greater than the limit.
6. Updating the original signal by removing all located outliers and interpolating it to fill the gaps.

Eventually the starting limit of 2 and 5 iterations were enough to treat even very noisy signals. A signal heavily infested with outliers is presented in Figure 2.2. The same figure also pictures the result of performing multiple runs of a median filter on the data.

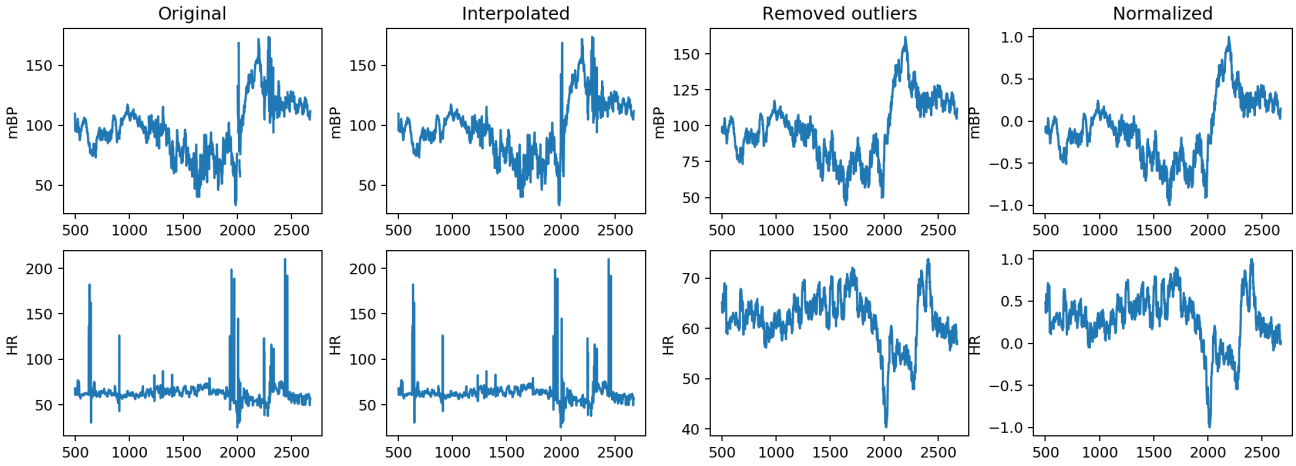

Figure 2.3: *The full data processing workflow using the normalization step.*

2.3.4. Range mapping

One of the important aspects of data preparation was an adequate mapping of signals to the $[-1, 1]$ range. This step was essential. In fact, as it is shown later, neural networks require such narrow range. Otherwise they tend to fail to learn or become unstable.

Experiments were performed by training and evaluating sample models for three different methods of data representation:

- without mapping – models failed to learn, resulting in low accuracy;
- normalization:

$$y^{(t)} = (\frac{x^{(t)} - min(x)}{max(x) - min(x)} - 0.5) \cdot 2$$

where:
x – original time series
y – normalized time series

This approach yielded models with maximum accuracy not higher than 80%. Most likely due to the fact that signals were stretched or squished horizontally. As a result information about mean value or standard deviation was irreversibly lost at cost of data calibration to the required range. In summary normalization was a wasteful approach even though the models reached relatively high accuracy. The whole process of data preparation while using the normalization step is shown in Figure 2.3;

- rescaling:

$$x_{min} = min(x_1 \cup x_2 \cup ... \cup x_N)$$

$$x_{max} = max(x_1 \cup x_2 \cup ... \cup x_N)$$

$$y^{(t)} = (\frac{x^{(t)} - x_{min}}{x_{max} - x_{min}} - 0.5) \cdot 2$$

where:
x_{min} – minimal value of all time series combined, global minimum of the dataset
x_{max} – maximal value of all time series combined, global maximum of the dataset
x – original time series
y – rescaled time series

In this case training was smooth whereas the models obtained displayed relatively very high accuracy. Most likely because of preserving the nature of the original signals. Due to its advantages this approach of data mapping was used in the final experiments. The whole process of data preparation while using the normalization step is shown in Figure 2.4

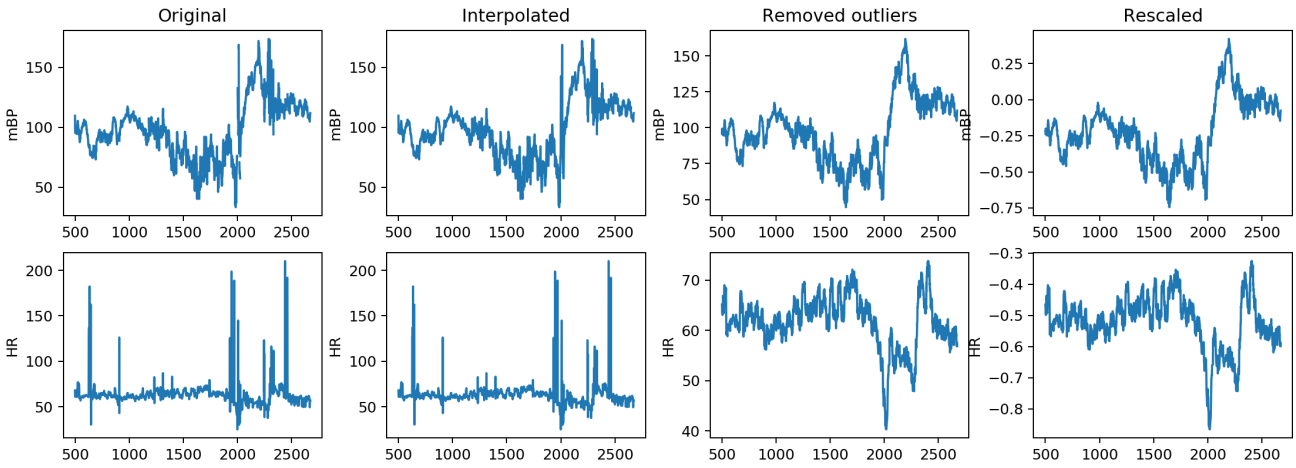

Figure 2.4: *The full data processing workflow using the rescaling step.*

2.4. Dataset balancing and division

The number ratio of "nosyncope" to "syncope" files in the dataset was almost 6:1. Unfortunately models naively trained using all available data showed bias towards classification as "nosynkope". In order to fix it balancing was applied. Generally all data of class "synkope" was used while the files of the second one were selected at random to generate a dataset with exactly equal number of files of each class.

The balanced data was then divided into three sets:

- trening set (70%) – used to train final models,
- test set (15%) – used to evaluate final models,
- validation (15%) – used to tune the hyperparameters of the final models as well as evaluate them.

3. Architecture

This chapter is meant to describe the general architecture of the models used in this research. It touches upon the structure of the final neural networks as well as how they were trained and evaluated. It also briefly explains what are the essential architectural hyperparameters and how they were determined.

3.1. Overview

The main goal of this research is to investigate how well the recurrent neural networks function while being trained and evaluated using medical, temporal data. Because of it the basic model taken under consideration should be the deep RNN. Simply due to the fact that multilayered networks in theory should be capable of modeling hard problems more accurately.

Another relevant aspect of this work is applying the classification based approach to the resultant models. Even though it is believed that models trained using the anomaly detection strategy could display equal or higher accuracy – due to the fact that the datasets were initially unbalanced.

The final architecture of the networks used in the experiments is shown in Figure 3.1. It is worth noting that the models accept two inputs although there is no reason that prevents them from using more time series.

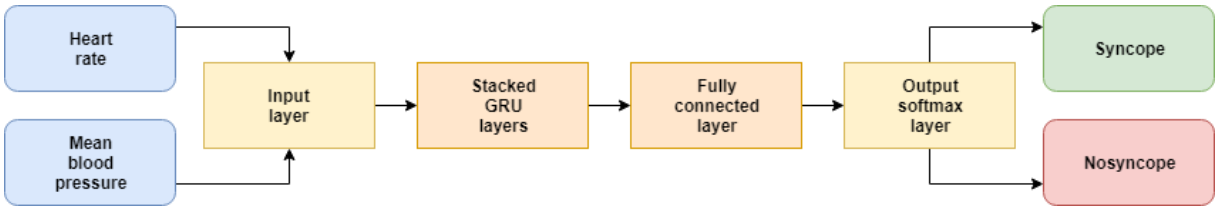

Figure 3.1: *Neural network architecture.*

3.2. Network structure

The models that turned out to give very optimistic results were a variation of a deep, stacked GRU network. This architecture involves using three layer groups. Each of them serves different roles. The input layer simply accepts the time series prepared beforehand. The hidden layer acts as the recurrent layer. It provides the context and implicitly stores the history of previous inputs. The output layer maps the resultant time series into range $[0, 1]$ in order to enable further analysis.

3.2.1. Input layer

The input layer accepts two signals. These signals are respectively: the mean blood pressure and the heart rate of a particular patient which at this point were already preprocessed and adapted to the requirements of neural networks. In the earlier phases of this research more inputs were tested, e.g. the average volume of blood pumped into a cardiac ventricle. Yet it did not improve the accuracy of the resulting models and needlessly increased their complexity and training time. Most likely because they could be derived from the first two time series they were introducing redundancy of inputs.

3.2.2. Hidden layer

The hidden layer consists of one or more recurrent layers stacked on top of each other. Instead of using the plain recurrent neural networks architecture it was deliberately decided to make use of the GRU cells. It is simply due to the fact that GRU tends to be better, more accurate and generally avoids the typical issues associated with plain RNNs. Namely the exploding and vanishing gradient phenomenon during training.

Instead of using only the vanilla GRU architecture it is also possible to exploit the bidirectional GRU cells. Bidirectional networks tend to be twice as big and feature two-way recursive connections. While performing the experiments both approaches were tested in order to investigate how efficient and accurate their respective models were in comparison with each other.

The optimal number of recurrent layers or depth of the network is unknown, potentially flexible. It is believed that increasing it may boost accuracy at the cost of memory and training time of the models. In practice during experiments depth never exceed 2 because of limited computational resources that could have been disposed.

3.2.3. Output layer

The output layer consists of a fully connected layer joined with the operator performing *softmax* calculations. As a result the output generates two signals – one per class. The sum of these signals is always equal to 1. In practice only the first signal is useful. It represents the confidence of the model that the patient is in presyncope state or how likely the patient is to faint at the given time.

In order to maximize the accuracy and the confidence of the models a concept of a *threshold* is introduced. Whenever the generated "syncope" classification time series exceeds the *threshold* it is acknowledged that the patient is either fainting or about to collapse. More thorough explanation on how this parameter was determined is presented in Chapter 4.1.

3.3. Procedures

3.3.1. Training

The undefeated loss function considering classification problems is still the cross entropy loss function [12]. It was used in this work whenever errors needed to be calculated. For two outputs the formula for calculating them can be notated as follows:

$$loss(z, c) = -\log(\frac{\exp(z_c)}{\exp(z_0) + \exp(z_1)})$$

where:
z_0 – "syncope" class output value generated by the model,
z_1 – "nosyncope" class output value generated by the model,
c – target class label, respectively 0 for "syncope" and 1 for "nosyncope".

The exact algorithm used to update the weights of the model is called ADADELTA. It proves to be better than a simple SGD (Stochastic Gradient Descent) or its other optimizers like Momentum or ADAGRAD [13].

During training the learning rate was decreased steadily. At the end of each epoch it was decayed according to the formula presented below:

$$r_e = r_{e-1} \cdot d = r_0 \cdot d^e$$

where:
r - learning rate,
d - decay factor,
e - epoch number.

In case of training the randomized batches of size 16 containing data from training dataset were fed to the networks. It allowed for faster convergence of the models. It is worth noting that the size was deliberately small in order to maximize the generalizational capabilities of networks [8]. More about how this particular value was established can be found in chapter 4.1.

3.3.2. Evaluation

During the evaluation stage samples of every patient were passed into models sequentially. At the time it turned out that the defined test set was too small and produced too optimistic results. Therefore in the final experiments the test dataset consisted of earlier defined test and validation sets. The scores obtained by default were worse but better reflected the generalizational properties of the created networks.

Data of every patient used during this step was fed into a network to produce a classification signal – only the first output of the networks were used. Then all signals were assembled and evaluated for various *thresholds* and analyzed in order to establish the one that balances accuracy and reaction time as well as tries to avoid producing false negatives by maximizing the sensitivity of the classifier.

The reaction time describes the capacity of a model to predict collapsing before it may actually be observed. It is calculated by subtracting the manual detection time noted by a technician from the time of the automated classification performed by a model. Naturally the lower the median of these differences the better the network is

at predicting syncope before it actually happens.

All remaining metrics used while evaluating the quality of a model can be derived from the entries of a confusion matrix [14]:

	Real positive	**Real negative**
Predicted positive	TP	FP
Predicted negative	FN	TN

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$sensitivity = \frac{TP}{TP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

$$F_1 = 2 \cdot \frac{precision \cdot sensitivity}{precision + sensitivity}$$

where:
TP – number of true positives (correctly classified as "syncope"),
TN – number of true negatives (correctly classified as "nosyncope"),
FP – number of false positives (falsely classified as "syncope"),
FN – number of false positives (falsely classified as "nosyncope"),
$accuracy$ – percentage of correctly classified series,
$sensitivity$ – how often the classifier classifies "synkope" signals correctly,
$precision$ – how often the classier is correct if it classifies the time series as "synkope",
F_1 – score that more accurately reflects the capability of the classifier to correctly classify series.

In practice F_1 and $sensitivity$ are a more accurate criteria of quality in this particular problem. It is due to the fact that it is important to avoid false negatives. Otherwise patients that are likely to faint would be incorrectly classified. In turn the examination would not be interrupted beforehand and the patient would collapse. It was explained in Chapter 1.1 why such a situation ought to be avoided at all costs.

3.3.3. Hyperparameters determination

The accuracy of a model can oftentimes be manipulated by adequately defining the parameters of the network, learning algorithm and the classification criterion used. In case of this study the hyperparameters that needed to be determined were consecutively:

- *hiddenSize* – number of neurons in every gate of a GRU cell,
- *depth* – number of GRU cells stacked on top of each other,
- *sequenceLength* – window length,
- *minibatchSize* – size of the batches of data,
- *lr* – learning rate,
- *lrd* – learning rate decay,
- *threshold* – classification criterion threshold.

The optimal hyperparameters were obtained by the use of the Bayesian optimization process. More information about it can be found in Chapter 4.1.

4. Experiments

This chapter presents the results of various experiments performed using models of the devised architecture as well as other classical methods in order to evaluate and compare their quality and efficiency. Firstly the tuning process of hyperparameters will be summarized, then two most potent variants of the final model are going to be thoroughly compared. Lastly scores for other popular algorithms will be discussed to highlight the advantages of the approach devised in this work.

4.1. Tuning

Tuning of hyperparameters consisted of two runs of various parameter ensembles of the Bayesian optimization algorithm. After each run the set of parameters was reduced, eventually it contained only two most vague but also relevant parameters. The whole idea of parameters tuning was used in order to reduce computational time by avoiding other methods of space searching, e.g. grid search. All runs were done on vanilla GRUs.

The first run was performed on seven different variables: $hiddenSize$, $depth$, $sequenceLength$, $minibatchSize$, lr, lrd, $threshold$ – their definitions can be found in Chapter 3.3.3. The validation dataset was used for training as well as testing purposes due to limited computational resources and because of the relatively high number of parameters to be optimized. Each iteration was only 3 epochs long for the same reason. On the other hand this run was done only to establish which hyperparameters visibly influence the classifier quality. The partial dependence plot yielded by the experiment is presented in Figure 4.1. It shows a few interesting facts and it leads to natural conclusions about this particular parameter ensemble:

- learning rate converges correctly to 1 for ADADELTA,
- relevance of depth decreases with increasing values of neurons in hidden layers,
- models tend to perform better whenever low sizes of mini-batches are being used,
- learning rate decay has no influence perhaps due to low epoch count,
- threshold should be not higher than 0.75.

The second run used a reduced set of parameters: $hiddenSize$, $depth$, $sequenceLength$. Which in fact were the most important of them all and should directly influence the neural network architecture, training and evaluating times, but also the memory consumption of the models. Similarly to the previous experiment the results are presented as the partial dependence plot. It is shown in Figure 4.2. It is clearly visible that low $sequenceLength$ tends to improve the accuracy of the classifier. It is worth noting that the optimal $hiddenSize$ seems to lie between 100 and 160, regardless of $depth$ factor.

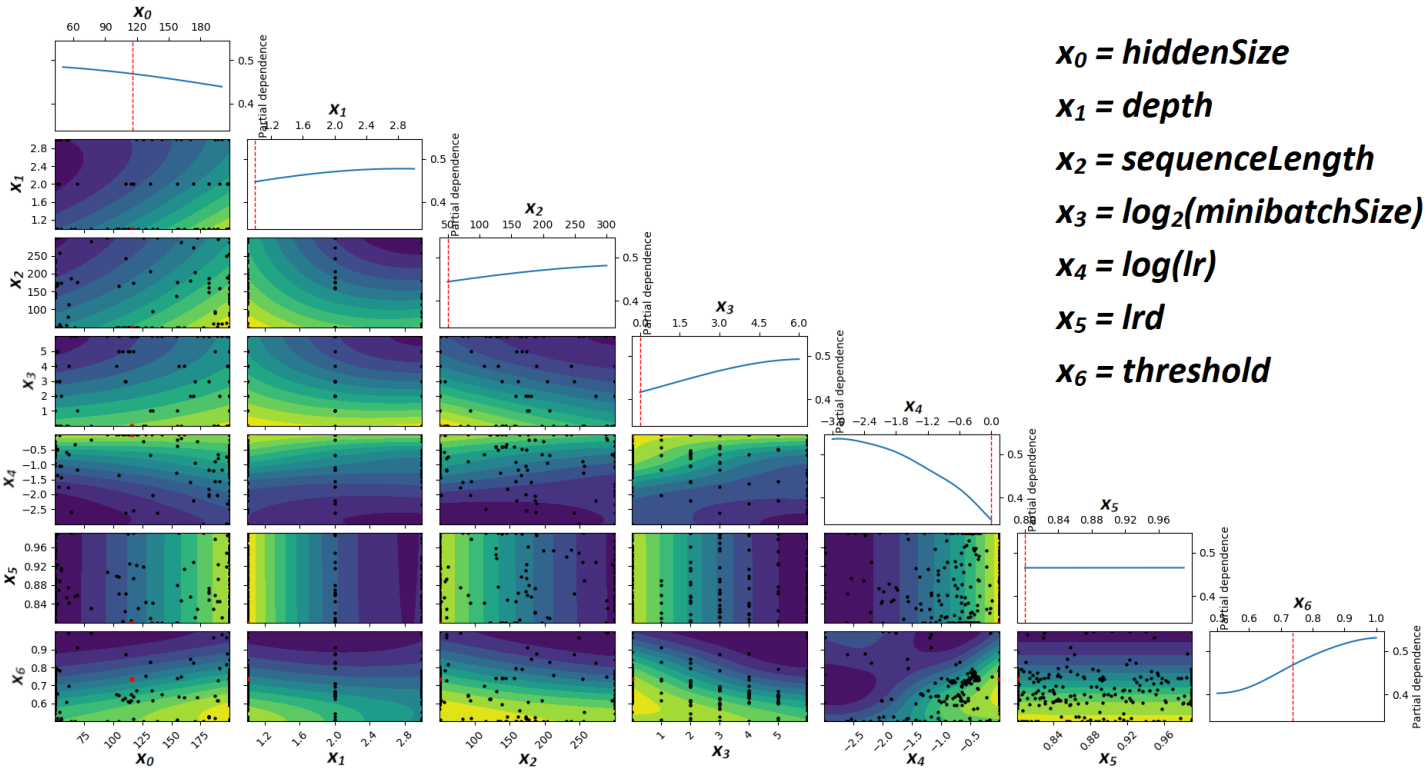

Figure 4.1: *Partial dependence of all parameters. It consists of the heat maps of influence of two specific parameters and line plots showing how on average the classification error of trained models changed for a given parameter.*

In summary the hyperparameter space was reduced thanks to the application of the Bayesian optimization algorithm. The values used in the training experiments described in the next chapter heavily depend on the estimations presented below:

$$hiddenSize \in [100, 160]$$

$$depth \in \{1, 2\}$$

$$sequenceLength = 300$$

$$minibatchSize = 16$$

$$lr = 1$$

$$lrd = 0.97$$

$$threshold < 0.75$$

4.2. Training

Various models were trained making use of the parameters found in the previous chapter. Two main architectures considered in this experiment involved the vanilla GRUs and their bidirectional variation. As it turns out later both demonstrate different properties when it comes to classification of temporal data. In order estimate their usefulness their predictions are going to be compared with the manual detection times – noted during an examination by a technician whenever a particular patient fainted.

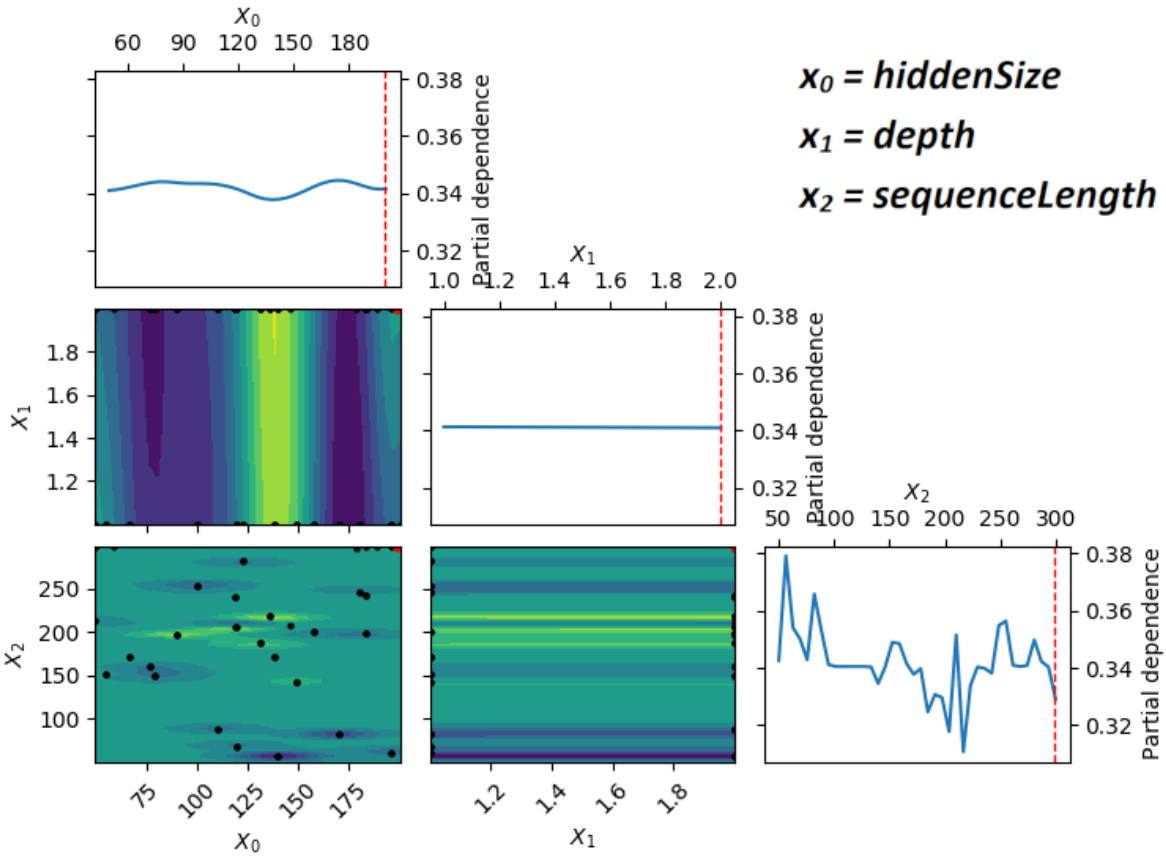

Figure 4.2: *Partial dependence of parameters used in the second run.*

4.2.1. Model comparison

At this stage multiple networks were trained and evaluated using predefined $threshold$ of 0.7 as well as the best found $threshold$ for a particular model. Each model was trained for approximately 72h using only CPU cores of the supercomputer Prometheus courtesy of PL-Grid infrastructure. Time constraints of a single computational task were some of the limitations of this particular research. It is believed that longer training time with steadily decreasing learning rate may improve the quality of the classifier. This tendency was observed in the worst models which achieved much higher accuracy as a result of retraining.

	hiddenSize = 100 depth = 1	**hiddenSize = 200 depth = 1**	**hiddenSize = 100 depth = 2**
vanilla	0.667 (0.526)	0.731 (0.632)	0.808 (0.763)
bidirectional	0.745 (0.658)	0.792 (0.737)	0.905 (0.895)

Table 4.1: *F_1 scores (accuracy in brackets) obtained for various networks and threshold = 0.7.*

Table 4.1 represents the F_1 scores obtained for various network setups by utilizing the predefined $threshold$. It is clearly visible that in this scenario the bidirectional models outrank their counterparts. These experiments also show that vertical expansion (depth) is superior to simple increasing the number of neurons in layers. The best model happened to be the deep stacked bidirectional variation and it is going to be discussed in detail in Chapter 4.2.3.

	hiddenSize = 100 depth = 1	**hiddenSize = 200 depth = 1**	**hiddenSize = 100 depth = 2**
vanilla	0.750 (0.789)	0.872 (0.868)	0.808 (0.763)
bidirectional	0.800 (0.763)	0.826 (0.789)	0.905 (0.895)

Table 4.2: *F_1 scores (accuracy in brackets) obtained for various networks and the best threshold found per model.*

Oftentimes using a predefined $threshold$ caused the models to not exhibit their real potential. The best way to solve it was to implement grid search in order to find the optimal value of this hyperparameter for each particular network. The same networks were evaluated once again this time for their optimal $threshold$ what can be seen in Table 4.2. Even though this approach was not impressive in case of bidirectional networks the vanilla models happened to achieve much higher scores. It can be shown that the later tend to act better when the overall number of neurons in layers is being increased rather than depth. The most potent vanilla network is explored in Chapter 4.2.2.

4.2.2. Vanilla GRU

The vanilla GRU networks turned out to be less efficient and harder to be modelled. Although they reached high accuracy, their reaction time and resilience to false negatives were unacceptable and had yet to be improved. All vanilla models yielded similar results, but the best of them so far turned out to be the shallow one with 200 neurons per hidden layer. Figure 4.3 presents signals generated by this network for all the time series from the test dataset. The most important chart lays on the right and shows the maximal values reached for various patients. In this case $threshold$ could be visualized as a horizontal line dividing these points into two groups where the ones above it belong to times series classified as "syncope". It is easy to see that the goal of the classifier is to get as few green dots and as many red dots below the $threshold$ as possible.

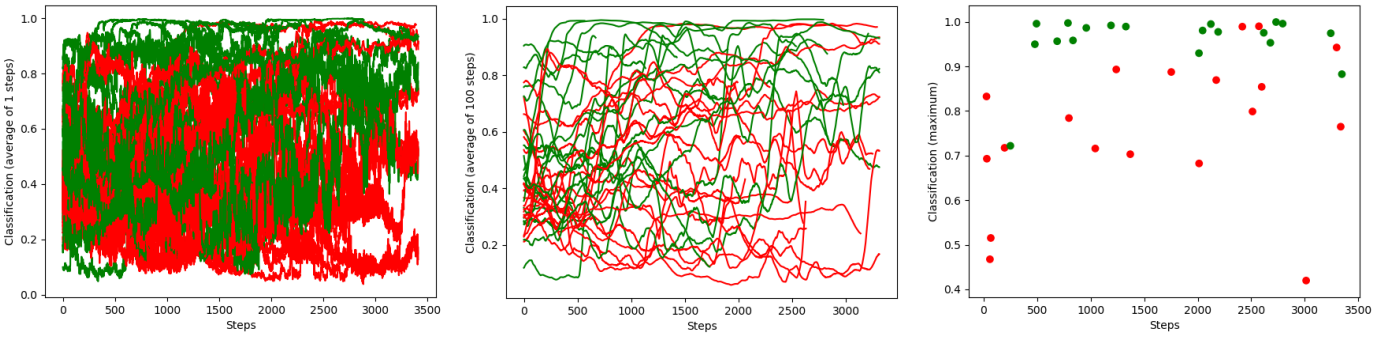

Figure 4.3: *Combined raw signals obtained from the vanilla network for the test dataset (red for "nosyncope" and green for "syncope") on the left. Smoothed signals in the center. Locations of maximal values of the signals on the right.*

Another useful details can be observed in the Figure 4.4. It shows the behaviour of the classifier for various values of $threshold$ as well as its reaction time. The main goal of this research is to devise a model that maximizes the sensitivity as well as accuracy but detects the fainting relatively early. It is quite obvious that the best $threshold$ equal to 0.895 produces the highest accuracy (0.868), but also insufficient sensitivity (0.895) and reaction time (31s after the manual detection). On the other hand the threshold of 0.72 provides lower accuracy (0.711), although the sensitivity is maximized and the reaction time is high as well (more than 3min before the

manual detection).

It is obvious that in this case the best $threshold$ turned out not to be the one that maximized the accuracy of the classifier. The most reliable value of this parameter happened to be lowering the accuracy at the cost of increasing the sensitivity of the model and improving its reaction time to symptoms of fainting.

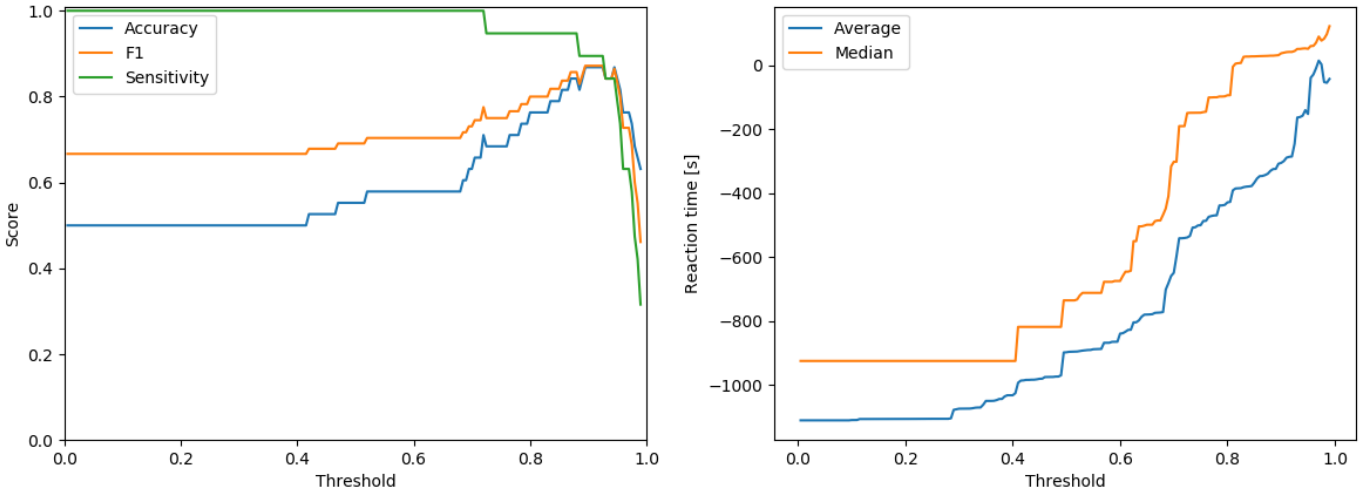

Figure 4.4: *Scores achieved by the vanilla network for various thresholds on the left. Difference between model's prediction and manual presyncope detection on the right.*

4.2.3. Bidirectional GRU

The bidirectional variations seemed to be more accurate, but also more confident while classifying data. It is clearly visible in Figure 4.5 where the points showing the maximum values of signals lie on the opposite ends of the chart. All the green points are located near 1 and most of the red points can be found close to 0. That implies that $threshold$ loses its importance drastically.

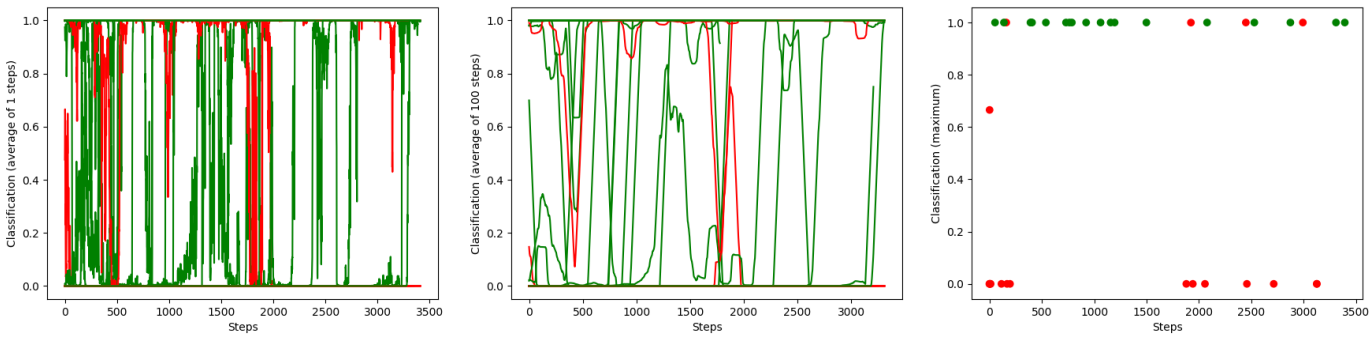

Figure 4.5: *Combined raw signals obtained from the bidirectional network for the test dataset (red for "nosyncope" and green for "syncope") on the left. Smoothed signals in the center. Locations of maximal values of the signals on the right.*

Figure 4.6 proves that the only thing that $threshold$ slightly influences in case of this classifier is its reaction time. It does not require further analysis due to the fact that the default $threshold$ of value 0.7 not only maximizes the sensitivity, accuracy and F_1, but also provides a satisfactory reaction time (more than 10min before manual

detection). This proves that the deep bidirectional variant tends to be not only more effective but also more reliable on the whole when it comes to predicting collapse. Therefore it should be the baseline approach for this particular problem.

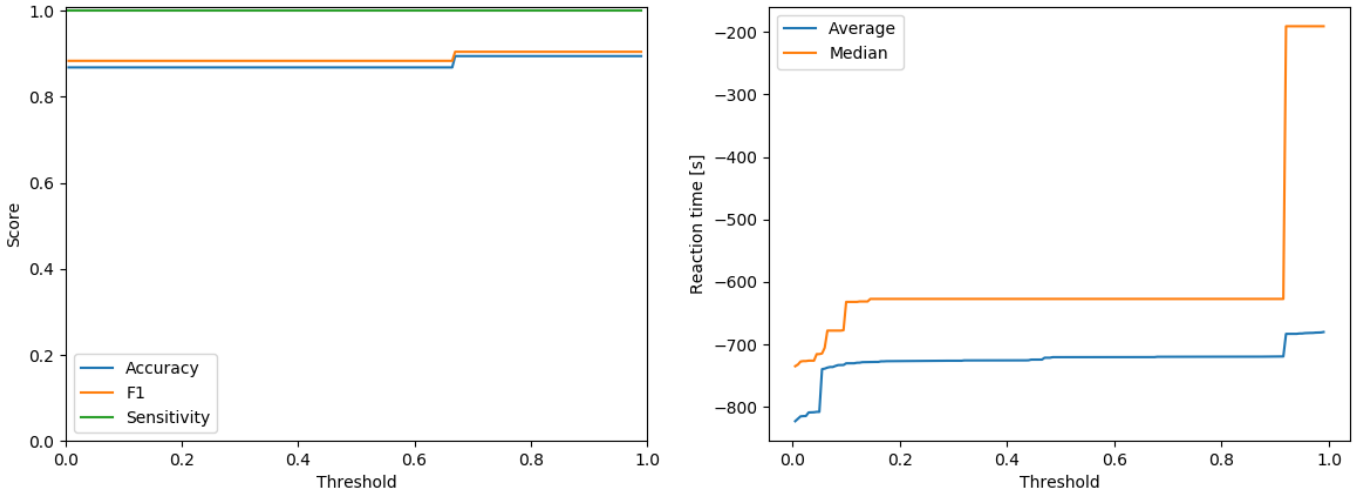

Figure 4.6: *Scores achieved by the bidirectional network for various thresholds on the left. Difference between model's prediction and manual presyncope detection on the right.*

4.3. Other methods

In order to compare the efficiency and accuracy of GRU architectures further experiments were performed using commonly used classification algorithms. All models were trained using the training set and evaluated on combined validation and test sets. Experiments were performed for nonsequential data as well as for summed and flattened signals. Whenever data was summed the two main signals were added to each other and then a window of size 300 was used to further infer the feature vectors. Whereas flattening relied on alternating the input signals and then applying similar window of size 600. Time series had to be either summed or flattened due to the fact that most of the algorithms applied are not sequence friendly and require single feature vector as input. The final results for all data representation approaches are presented in Table 4.3.

	Nonsequential	**Summed**	**Flattened**
Decision Tree	0.549	0.569	0.574
kNN(2001)	0.605	0.576	0.646
Gaussian Naive Bayes	0.580	0.507	0.570
Random Forest(101)	0.568	0.616	0.600
SVM	0.623	0.514	0.640
ARIMA	-	<0.7	-

Table 4.3: *Ensemble of accuracy scores obtained for various classification algorithms and different data representation methods.*

It can be seen that the methods that utilize decision trees tend to improve slightly upon introduction of sequential data. Remaining methods show a severe discrepancy between scores obtained for summed and flattened signals. The main reason for this may be that summing the input signals is in fact lossy and not an optimal

approach to the problem.

Nevertheless the results are not satisfactory and nowhere as compelling as the ones obtained from the architecture devised in this work. It is worth noting that in case of standard classification methods only accuracy was tested. Other important factors like high sensitivity or reasonable reaction time were not even taken under consideration due to the fact that the earlier mentioned accuracy was already relatively low and unsatisfactory.

4.4. Summary

The devised time series processing scheme proves to be an effective technique of handling medical temporal data. It solves frequent problems arising while transforming flawed time series. This research also provides other variants of data preparation as well as restoration and highlights their advantages and disadvantages.

Networks created for the sake of this work exhibit high accuracy. In fact the most potent architecture turns out to be the deep stacked bidirectional GRU, which not only outranks other classical algorithms and methods of time series analysis but also fulfills all the requirements set by this research. Because it has low tolerance for false negatives, high accuracy and detects fainting long time before it actually happens.

The results show that neural network based models can be efficiently used in medicine. Whether to augment the examination process or support the decision making of a doctor. The data processing scheme and the architecture devised for the sake of this research may potentially be taken advantage of in the future not only to predict the cases of fainting, but also in other similar problems.

4.4.1. Future works

Even though the results achieved in this work were satisfactory, there is still space for improvement. From the very beginning this research focused on applying classification approach while using neural networks. Yet it is highly possible that networks with similar architecture, but implementing anomaly detection methodology could achieve equal or even better results due to the fact that the data provided by the donor was not balanced to begin with.

5. Bibliography

[1] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[2] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. *arXiv e-prints*, page arXiv:1211.5063, November 2012.

[3] Sepp Hochreiter and JÃijrgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[4] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv e-prints*, page arXiv:1409.1259, September 2014.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv e-prints*, page arXiv:1412.3555, December 2014.

[6] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to Construct Deep Recurrent Neural Networks. *arXiv e-prints*, page arXiv:1312.6026, December 2013.

[7] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent Neural Networks for Multivariate Time Series with Missing Values. *ArXiv e-prints*, page arXiv:1606.01865, June 2016.

[8] Dominic Masters and Carlo Luschi. Revisiting Small Batch Training for Deep Neural Networks. *ArXiv e-prints*, page arXiv:1804.07612, April 2018.

[9] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization.

[10] Lei Lin, Beilei Xu, Wencheng Wu, Trevor Richardson, and Edgar A. Bernal. Medical Time Series Classification with Hierarchical Attention-based Temporal Convolutional Networks: A Case Study of Myotonic Dystrophy Diagnosis. *arXiv e-prints*, page arXiv:1903.11748, Mar 2019.

[11] Anahita Hosseini and Majid Sarrafzadeh. Unsupervised Prediction of Negative Health Events Ahead of Time. *arXiv e-prints*, page arXiv:1901.11168, Jan 2019.

[12] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 1 2005.

[13] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv e-prints*, page arXiv:1212.5701, Dec 2012.

[14] Tom Fawcett. An introduction to ROC analysis. `https://people.inf.elte.hu/kiss/11dwhdm/roc.pdf`, Dec 2005.