EEG-Based Neurological Outcome Prediction after Cardiac Arrest with an LSTM RNN

Can We Predict Life after the Heart Stops Beating?

Laura van Poppel
4287142

Delft University of Technology
31-12-2020

Supervisors: Dr. ir. A. C. Schouten, TU Delft
Dr. W. V. Potters Amsterdam UMC

TU Delft

EEG-based Neurological Outcome Prediction after Cardiac Arrest with an LSTM RNN

Laura M. van Poppel

Supervisors: Dr. Wouter V. Potters, Dr. Alfred C. Schouten

Delft University of Technology 31-12-2020

Abstract

Objective: Prediction of comatose patients' neurological outcome after cardiac arrest (CA) is essential to prevent unnecessary continuation of treatment (with expensive and scarce resources) and prevent emotional burdens on families. Electroencephalography (EEG) can aid the prediction of outcome after CA. Visual EEG analysis is subjective, time-consuming, and might miss important information. Existing EEG-based machine learning models are more reliable but lack interpretation of temporal characteristics of EEG. Long-short-term-memory networks (LSTMs) do take advantage of such temporal characteristics. Therefore, I hypothesized that an LSTM would outperform time-insensitive models.
Methods: Patients' neurological outcome at six months after CA was classified as good (no/mild neurological damage) or poor (severe neurological damage/vegetative state/death). Twelve quantitative EEG features were extracted from five-minute EEG epochs recorded 12 (n=78) or 24 (n=176) hours after CA. A time-insensitive baseline logistic regression model (LR) and an LSTM were developed and trained. The performance was evaluated using the area under the receiver operator curve (AUC) and the sensitivity at 100% specificity (SeSp100) for poor outcome prediction. The LSTM was compared to the current LR and previously published models.
Results and conclusion: The LSTM predicted poor outcome with AUC=0.90 and SeSp100=0.66, meaning it did not significantly improve performance over LR (AUC=0.89, SeSp100=0.67). However, the LSTM and LR outperformed almost all previously reported models, likely due to the features' high prognostic power. The SeSp100 was even higher for the LSTM (0.79) and LR (0.78) when using only epochs at 12 hours after CA, suggesting that earlier EEG might further improve prognostication.

1 Introduction

Approximately half of the patients who suffer from a postanoxic coma (PAC) after cardiac arrest (CA) never recover to consciousness [1, 2, 3]. Only 10% to 30% of the patients reach a meaningful recovery [4]. Reliable and early identification of (un)salvageable patients is of significant importance, as the uncertainty of outcome causes a large emotional burden on families. Moreover, the prediction could support decisions concerning life-supporting treatment [5]. Premature withdrawal of this treatment should always be prevented. However, the continuation of care when no meaningful recovery can be achieved should also be avoided for multiple reasons. First of all, to spare relatives a long, emotionally burdensome process and potentially unrealistic expectations. Furthermore, limited resources for life-supporting treatment are available, and the healthcare costs are very high [6, 7], making an unnecessary continuation of treatment undesirable.

A poor neurological outcome, defined as severe disability, vegetative state, or (brain)death, is currently predicted with a combination of several clinical examinations [5]. Although these include predictors with high specificity, like the absence of somatosensory evoked potential responses or pupil reflexes, their sensitivity is low [8, 9]. Approximately 20% of the patients with a poor outcome can be identified accordingly [3, 10]. Therefore, clinical examinations are not sufficient in the prediction of poor outcome.

Used in combination with the predictive clinical examinations, monitoring brain activity can aid in prognostication [11]. Visual analysis of the electroencephalogram (EEG) can reliably predict poor or good outcomes in around 50% of the patients [12, 13, 14, 15, 16]. Specifically, EEG recordings within 24 hours after CA showed the strongest predictive power, while not limited by post-resuscitation treatment, according to current literature [12, 16, 17, 18, 19, 20, 21]. Specific EEG patterns were associated with a poor outcome: an isoelectric EEG, a low voltage ($<$ 20 µV) EEG, and a burst-suppression EEG, especially if the bursts are synchronous or identical. Continuous EEG activity was associated with a good outcome [14, 15, 17]. However, other EEG patterns that showed no consistent association with either poor or good outcomes remain [11]. Appendix A provides background information about the mechanism of EEG and the physiological EEG signals. Appendix B describes EEG signals following CA and resuscitation.

Unfortunately, visual analysis of the EEG has its limitations. Visual analysis suffers from inter-observer variability and subjectivity, as the expertise of the neurologist influences the interpretation. Another limitation is the lengthy specialised training required to educate neurologists in visual EEG analysis. Moreover, studying the EEG itself is time-consuming [22, 23, 24, 25, 26, 27]. Finally, only the discussed patterns with an established re-

lationship with a poor or good outcome are used in prognostication by visual EEG analysis. Important prognostic information could be present in the remaining EEG patterns. As visual analysis discards remaining patterns, this information is lost[18, 20].

Automatic analysis of the signal could overcome these drawbacks [20, 28, 29, 30]. The use of quantitative EEG (qEEG) features derived from the EEG, such as the power in the frequency bands or the signals' entropy, might offer several advantages. QEEG results in a more precise classification between outcomes and better detection of features that are not easily visible with human analysis. Moreover, it offers real-time prognosis and decreased healthcare costs [6, 19, 31, 32]. Consequently, research on the use of qEEG features for comatose patients' outcome prediction after CA has increased in the last years [33]. Tjepkema-Cloostermans et al. developed a cerebral recovery index (CRI), which integrated several features into one index [19]. Over the years, researchers enhanced [34], optimised [35], and revised [18] this index, including increasingly more qEEG features and advanced machine learning (ML) algorithms. The most recent version offers "the most sensitive, reliable predictor of neurological outcome of comatose patients after cardiac arrest published so far", according to the authors [18]. Recently, some researchers investigated artificial neural networks (NN) for outcome prediction [6, 20, 21]. The investigators' choice for NN was motivated by these models' ability to extract features from the data automatically. The model itself decides on the classification features and might discover relevant information in the signal. Specifically, these investigators created convolutional neural networks (CNNs), which have proven effectiveness in image analysis [6, 20, 21]. Appendix C discusses various EEG-based ML outcome prediction models, including their characteristics and performances.

A possible drawback of currently published ML outcome prediction models is their inability to preserve the input's temporal characteristics. As the brain produces non-linear and dynamic time-series data, temporal properties of the EEG signals could add intrinsic information about the brain's activity [36, 37, 38, 39, 40]. A recurrent neural network (RNN) is an example of an ML model that does capture such temporal characteristics [38, 39, 41, 42]. However, RNNs struggle with vanishing gradients due to their design, making it difficult for them to learn long-term dependencies (10 timesteps or more) [42]. The long short-term memory RNN (LSTM), a specific type of RNN, overcomes this problem and can retain information across many timesteps [41, 42, 43]. LSTMs performed very well in numerous EEG-based regression or classification tasks [38, 39, 44, 45, 46]. Moreover, comparisons between ML models in these EEG-based tasks showed that LSTMs outperformed more traditional ML models, like logistic regression (LR) [46, 47], support vector machines [39, 44, 45, 46, 47, 48, 49], or random forest classifiers [45, 46], and also feedforward NN [47, 50], deep belief networks [40, 48], and even CNNs [40, 45, 46, 48]. Appendix D provides more details about studies using EEG-based LSTM models.

The previously reported EEG-based outcome prediction models used five-minute EEG recordings for prediction, without accounting for the EEG signals' changes within these five minutes. The current study's goal was to investigate whether accounting for the EEG signal changes in the five-minute recordings using an LSTM could achieve higher prognostic performance. Therefore, I created and compared an LSTM and a baseline time-insensitive LR. Both models were trained and tested using the same dataset. The research question of the current study was stated as: 'Does an LSTM achieve better performance in neurological outcome prediction for patients suffering from a postanoxic coma after cardiac arrest using five-minute EEG recordings than a logistic regression model using equal recordings?'

Additionally, both models were compared to previously reported EEG-based PAC prognostication models. As the available dataset (n=254) is relatively small for training neural networks, I mainly aimed to investigate the effectiveness of LSTMs for EEG-based outcome prediction and formulate future directions. I stated two hypotheses. First of all, that the LSTM would outperform the LR model. Secondly, that the LSTM would achieve better performance than the previously reported EEG-based outcome predictors for patients suffering from a PAC after CA.

2 Methods

2.1 Patients and treatment

The data used in this study resulted from a previous cardiac arrest (CA) outcome prediction study [51]. The data were obtained from consecutive adult comatose CA patients admitted at the intensive care unit (ICU) of Amsterdam University Medical Centres – location AMC. Patients were eligible if continuous EEG recording had started within 24 hours after CA. Exclusion criteria included traumatic brain injury, acute stroke, progressive neurodegenerative disease, prearrest modified Rankin scale $\geq$4, or prearrest life expectancy $\leq$6 months based on comorbidity. Continuous EEG recording was part of the routine care in patients suffering from a postanoxic coma (PAC) after CA. Therefore, the institutional review board of the Amsterdam University Medical Centres, the Netherlands, approved the study protocol and waived the need for written informed consent. The treatment included targeted temperature management for 24 hours. The patients were sedated with propofol and rarely with additional midazolam. Physicians based decisions about the withdrawal of life-supporting treatment on the Dutch recommendations for prognostication in PAC. Table I summarises the patient characteristics. Appendix E describes the cause and treatment of CA.

Table I: Patient characteristics. CA=cardiac arrest. ECG= Electrocardiography. Std=standard deviation.

		Good outcome (n=76)	**Poor outcome (n=105)**
Age	mean (+/- std)	59.5 (+/- 14.8)	61.6 (+/- 15.5)
Sex	Male	59	77
	Female	17	28
Location of resuscitation	Out-of-hospital	69	87
	In-hospital	7	18
Presumed cause of CA	Cardiac	59	53
	Non-cardiac	5	35
	Unknown	12	17
Initial ECG rhythm	Asystole	4	25
	Bradycardia	0	2
	Pulseless electrical activity	3	18
	Pulseless ventricular tachycardia	1	2
	Sinusbradycardia	1	0
	Unshockable	0	2
	Ventricular fibrillation	66	51
	Unknown	1	4
	Else	0	1
Patient's recording time	Only at 12 h after CA	2	3
	Only at 24 h after CA	35	68
	Both 12 h and 24 h after CA	39	34

2.2 Data acquisition

Continuous EEG recording was initiated as soon as possible after admittance to the ICU. Recording generally started within 24 hours after CA and lasted for 72 hours, or up until the moment life-supporting treatment was withdrawn, or patients recovered to consciousness. Recordings were made using nine electrodes, placed according to the international 10-20 system (Fig. 1). Additionally, a ground and reference electrode were placed in the midline. The recorded channels included: [Fp1-REF, Fp2-REF, T3-REF, T4-REF, C3-REF, C4-REF, Cz-REF, O1-REF, O2-REF]. The EEG was recorded with a sample rate of 1000 Hz. As a result, physicians recorded continuous EEG of 181 comatose patients admitted to the hospital. For this study, artefact-free five-minute EEG epochs (n=254) were available, recorded approximately 12 hours (n_12=78) and 24 hours (n_24=176) after CA. An artefact-detection algorithm, described in a previous study [19], selected artefact-free epochs.

2.3 Outcome assessment

As the primary outcome measure, the neurological outcome at six months after CA was used, classified by the Cerebral Performance Category (CPC). The outcome was dichotomised as good or poor, consistent with previously reported outcome prediction models [6, 16, 18, 19, 20, 21, 29, 32, 34, 52]. Good outcomes had a CPC score of 1 or 2, indicating no or mild neurological damage, respectively. Poor outcomes had a CPC score of 3, 4 or 5, meaning severe neurological damage, vegetative state, or (brain)death respectively [4, 5]. Appendix F includes a table of CPC scores and their characteristics. Furthermore, Appendix G shows EEG fragments of patients with different outcomes.

2.4 Preprocessing

The nine-channel EEG epochs were re-referenced to a longitudinal bipolar montage to reduce the noise in the signal. Subsequently, a bandpass filter of 0.5-30 Hz was applied, reducing the influence of frequencies outside the brain's main power spectrum. Additionally, a notch filter of 50 Hz was applied to remove any powerline artefacts. Finally, the signals were downsampled to 128 Hz to reduce computational time. Appendix H states the preprocessing pipeline in detail. The raw data were preprocessed with Brainstorm [53].

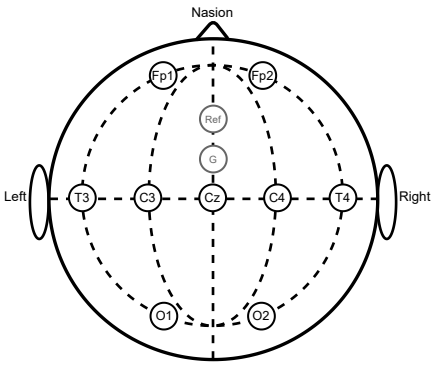


Figure 1: Visualisation of electrode placement according to the international 10–20 system with nine electrodes and an additional reference and ground electrode in the midline. The electrodes are visualised as circles with a letter that identifies the lobe: Fp=prefrontal, T=temporal, C=central, O=occipital, and a number that indicates the left side of the head (uneven numbers), the right side of the head (even numbers), or the midline (z). Ref=reference. G=ground.

2.5 Feature extraction and input selection

Extracted quantitative EEG (qEEG) features from the EEG signal were used as input, not the EEG signal itself. Extracting features provides the model with important hidden information within the signal and significantly reduces the dimensionality [39]. Moreover, various studies using EEG-based LSTMs showed that features outperform the pure signal as input [38, 39, 44]. Figure 2 shows the feature extraction and input selection process.

Feature extraction

The five-minute preprocessed EEG epochs were segmented into 30 non-overlapping 10-second time-fragments. 19 QEEG were extracted features per channel per time-fragment. The features were averaged across all channels and scaled between 0 and 1 with respect to the features of all epochs. For the LR, features were averaged across all time-fragments per epoch, resulting in a matrix of features by epochs. The required input shape to the LSTM was a three-dimensional tensor of features by time-fragments by epochs, thereby accounting for the change of the features over time within the epoch. Appendix I detailly describes the extraction process.

Table II summarises the 19 extracted qEEG features. The features were categorised into the three domains defined by Ghasemmi et al. [52]. Features in the complexity domain indicated the extent to which the signal is random or irregular. The category features were related to the EEG patterns observed in PAC patients. The features in the connectivity domain quantified relations between the EEG channels. The 19 features were selected based on results of other outcome prediction studies, where they showed notable contribution to prognostication at 12 or 24 hours after CA [18, 35, 52, 54]. As EEG patterns could change between 12 and 24 hours after CA [10, 28], only features that showed similar correlation with the neurological outcome at both timepoints were included. A detailed description of all features, including the calculations and the relation to the EEG patterns observed in PAC or to cerebral functioning, is discussed in Appendix J. Appendix Z.1 offers the MATLAB code used for feature extraction, which was based on Ghassemi et al., 2019 [52].

Feature input selection

From the 19 extracted features, the features with low multicollinearity were selected to use as input to the models. This features set was termed the final feature set (FFS). Multicollinearity was calculated with the variance inflation factor (VIF). The input to an LR model must never contain highly correlated inputs [55]; otherwise, the model will be unable to establish a correct feature-outcome relationship, which causes bad generalisation [55, 56, 57]. Generalisation is defined as a model's performance on unseen data [42, 58, 59]. Moreover, highly correlated inputs could also negatively impact the learning process of an LSTM, even though a neural network is capable of approximating any feature-outcome relationships. The high correlation in the input causes redundancies which lead to unnecessary complexities. These complexities could increase the number of local minima [60]. Table III states the FFS resulting from feature selection. The VIF calculation and the feature input selection process are described in Appendix K. Appendix Z.2 states the MATLAB script used for this process.

Additional to the FFS, multiple other feature sets were created, termed additional feature sets (AFSs) (Appendix L). These AFSs were formed using features from Table II and the clinical features age and sex. It was evaluated if AFSs showed better performance than the FFS when used as input to the final models. Appendix Z.3 and Z.4 include the MATLAB scripts used for creating the FFS and AFS for LR and the LSTM, respectively.

2.6 Performance metrics

Consistent with previously reported outcome prediction models, the final model performances was evaluated using three metrics retrieved from the receiver operating characteristic (ROC) curve: i) the area under this curve (AUC), and the sensitivities for ii) poor and iii) good outcome prediction at a predefined specificity threshold [6, 18, 19, 20, 21, 29, 32, 34, 35, 52, 54]. High specificity for poor outcome prediction is crucial to avoid withdrawal of life-supporting treatment in patients with a viable outcome [6, 18, 20, 52]. Therefore, the sensitivity at 100% specificity (SeSp100) for poor outcome prediction was used. Falsely predicting good outcome has a less disastrous consequence; hence the sensitivity at 95% specificity (SeSp95) was used. Appendix M states the computation of the performance metrics.

The models were optimised specifically for the metric SeSp100 during the hyperparameter optimisation process because a sensitive predictor of poor outcome is the most useful clinical application. Reliably prediction of poor outcome could prevent unnecessary use of the limited and expensive resources in the ICU and prevent long-term burdens on families [6, 7]. Secondary to the SeSp100, the AUC was considered an important metric, as the model should accurately predict poor and good outcomes to be clinically applicable [18]. The SeSp95 for good outcome prediction was calculated for comparability to other models. Achieving a high SeSp95 was not the focus of the current study.

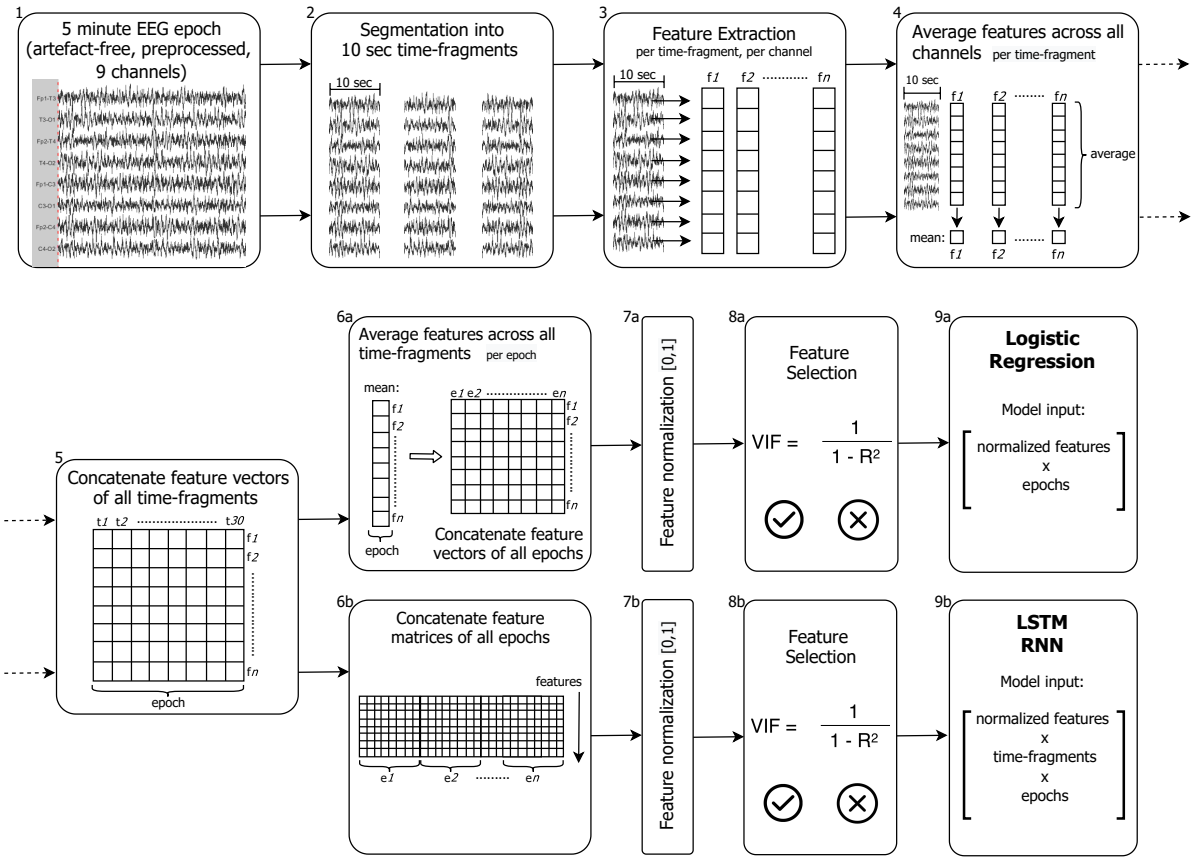


Figure 2: The feature extraction and input preparation process for the LR and LSTM models. The five-minute preprocessed EEG epochs (box 1) were segmented into 30 non-overlapping 10-second time-fragments (box 2). 19 QEEG features were extracted per channel from each 10-second fragment (box 3). The extracted features were averaged across all channels (box 4), resulting in a matrix per epoch of features by time-fragments (box 5). Subsequently, the process divided into two paths. Path a, for the input to the LR model: the mean values were averaged for the five-minute epoch, resulting in 19 qEEG features per epoch and the features of all epochs were concatenated (box 6a). Hereafter, features were scaled between 0 and 1 with respect to the features (box 7a). The features were selected for different feature sets to be used as input through various selection methods (box 8a).
Path b, for the input to the LSTM: the 19 qEEG features from the 30 time-fragments per epoch were used to account for temporal information within the epoch. The features from all time-fragments of all epochs were concatenated (box 6b). The features were scaled between 0 and 1 with respect to the features of all epochs (box 7b) and selected for different feature sets to be used as input (box 8b). The features were reshaped into the required input shape for the LSTM (box 9b). LSTM (RNN)=long short-term memory recurrent neural network. LR=logistic regression. (q)EEG=(quantitative) electroencephalography. VIF=variance inflation factor.

2.7 Logistic Regression

An LR model was created as baseline model for this study. The performance of the LSTM was compared to LR to evaluate if an LSTM model performed better than a simpler, time-insensitive model. LR was selected for the known ease in use, efficiency, and robustness to noise. LR forms a popular method for statistical modelling of binary outcomes [75]. One can regard LR as a type of feedforward neural network (NN)[58]. Appendix N offers background information about NNs.

Model information

In LR, the logarithm of the probability of odds, termed the logit, is equalled to a linear regression function.

$$ln\left(\frac{p_i}{1-p_i}\right) = W \cdot X + b \quad (1)$$

In this equation, p_i denotes the predicted value, which is the probability the output equals 1 (the probability of a poor neurological outcome). The linear regression consists of a dot product between the input (the features), X, and the corresponding weights, W. Moreover, it includes a bias term, b. To compute the predicted value,

Table II: Extracted qEEG features and descriptions. AR=autoregressive. Coeff=coefficient. (q)EEG=(quantitative) electroencephalography. Ref=reference.

Domain		Feature	Description	Ref.
Complexity	1	Shannon entropy	A measure to quantify the uncertainty of a stochastic signal	[61]
	2	Tsallis entropy	A measure to quantify the uncertainty of a stochastic signal in a nonextensive manner	[62] [63]
	3	Cepstrum	Both coefficients are a measure to quantify the rate of change in	[64]
	4	coefficients	different spectrum bands	[65]
	5	Hjorth mobility	The ratio of the variance of the first derivative of the signal and the signal itself, indication a proportion of the variance of the power spectrum	[66] [67]
	6	Hjorth complexity	A measure that quantifies how much similarity the shape of the signal has with that of a pure sine wave	
	7	False nearest neighbours	A measure to quantify the degree of stochasticity in a signal by estimating the embedding dimension, indicating its constancy and smoothness	[68] [69]
	8	AR coefficient 1	Estimated nonseasonal autoregressive term coefficients at t-1 (coeff. 1) and	[70]
	9	AR coefficient 2	t-2 (coeff. 2) given the EEG signal of a second order autoregressive model	[64]
Category	10	Normalised delta power	Delta power (0.5-4 Hz) divided by the total power (0.5-30 Hz)	[71]
	11	Normalised theta power	Theta power (4-7 Hz) divided by the total power (0.5-30 Hz)	[72]
	12	Normalised alpha power	Alpha power (8-13 Hz) divided by the total power (0.5-30 Hz)	[73]
	13	Normalised beta power	Beta power (14-30 Hz) divided by the total power (0.5-30 Hz)	
	14	Signal power	The total power in the frequency range of interest (0.5-30 Hz)	
	15	Regularity	A measure to quantify regularity in amplitude of the signal	[19]
	16	Epileptic spikes	The number of epileptic form spikes in the EEG	[52]
	17	Burst suppression ratio	The ratio of the duration of an EEG signal with an amplitude equal to or lower than 5 microvolts to the duration of the entire signal	[18]
Connectivity	18	Delta coherence	A measure to quantify the degree of similarity in the delta band	[19]
	19	Phase lag index	A measure to quantify phase synchronisation, indicating the level of asymmetry between two signals	[74]

Table III: Final feature set used as input to the LR and LSTM model. AR=autoregressive. LR=logistic regression. LSTM=long short-term memory recurrent neural network.

	Final feature set
1.	Tsallis entropy
2.	False nearest neighbours
3.	AR coefficient 2
4.	Normalised theta power
5.	Normalised alpha power
6.	Normalised beta power
7.	Signal power
8.	Regularity
9.	Number of epileptic spikes
10.	Burst suppression ratio
11.	Delta coherence
12.	Phase lag index

the logit function needs to be solved for p_i.

$$p_i = \frac{1}{1 + e^{-(W \cdot X + b)}} \tag{2}$$

To obtain a predictive model, the weights and bias terms, labelled the model parameters, need to be estimated [41, 58, 59, 76].

Estimation of the model parameters was done with an iterative process, compromising several steps. In the first iteration, random values for the model parameters were used. Per batch of samples, the predicted outcome was obtained through equation 2 and 3, termed the forward propagation. Subsequently, the error between the model's predicted outcome and the true outcome, denoted by the loss, was computed using a loss function. Thirdly, the backward propagation generated the gradients of the loss function with respect to the model parameters. An optimisation algorithm used the gradients to update the model parameters in the gradients' opposite direction to minimise the next iteration loss. Finally, the optimisation process was repeated for the defined number of epochs [41, 42, 58, 59, 77, 78]. Appendix O states more details about LR and the estimation process of the model parameters.

Hyperparameters

The LR model was built with Keras 2.3.1 upon Tensorflow 2.3.0 backend in Python 3.7. Figure 3 illustrates the model's architecture. LR was modelled with one unit in a fully connected layer, also termed dense layer, and a sigmoid activation function. Dense in combination with a sigmoid function implements the following operation [79].

$$p = \sigma\left(W \cdot X + b\right) \tag{3}$$

The sigmoid activation function maps the linear operation within dense between 0 and 1, obtaining the probability the outcome is equal to 1 [79].

$$\sigma\left(z\right) = \frac{1}{1 + e^{-z}} \tag{4}$$

Training the LR required the specification of several hyperparameters. The design of the LR defined the majority of hyperparameters that concerned building the model. Several choices for hyperparameters concerning compiling and fitting the model were made based

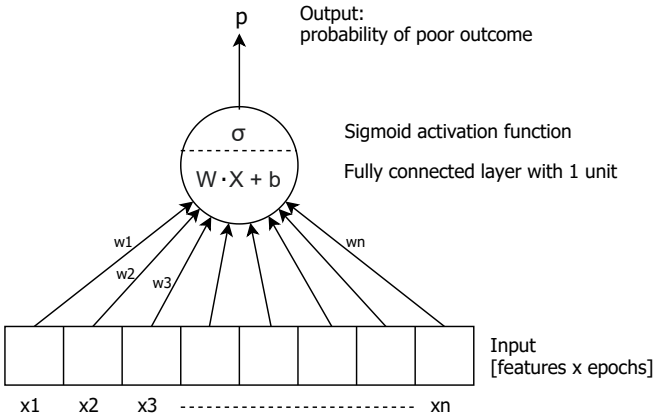


Figure 3: Logistic regression model architecture. The input followed a dense operation and was mapped between 0 and 1 by a sigmoid function. The output denoted the probability of a poor neurological outcome. X=input, W=weights, b=bias, σ=sigmoid activation function, p=predicted outcome.

on references in literature or defaults of the used framework, which are generally good [59]. As the performance of an ML model is highly dependent on the learning rate and the number of epochs with which it is trained [77], these hyperparameters were optimised. Appendix P detailly discusses the hyperparameters implemented in the model. The input to the LR during optimisation were the features in the FFS extracted from all epochs. During optimisation, the hyperparameter configurations were judged based on i) the performance metrics on unseen data (the SeSp100 for poor outcome prediction and in a lesser amount the AUC), ii) the robustness of the models, and iii) the stability the learning process near the end of the training. The latter quantified if, and in what degree, underfitting or overfitting occurred. Underfitting results from an oversimplified model that has not (yet) learned the relationship between the features and outcome, resulting in a high loss. An overfitted model adjusts the model parameters perfectly to the data used for training. Consequently, the loss computed on this dataset is minimised. However, the model will not generalise well to data it has never seen before [58, 59]. A reliable and robust estimation of the configurations' performances was obtained with five repetitions of 10-fold cross-validation (CV) [58, 59]. The hyperparameter optimisation and process of 10-fold CV is detailly explained in Appendix Q.

2.8 Long Short-Term Memory Recurrent Neural Network

Recurrent neural networks (RNNs) can learn temporal dependencies from time-series data, like EEG signals. Whereas traditional NNs only use input from the current timestep, RNNs additionally use information from the previous timesteps through connections between their layers with respect to time [43, 46, 58]. LSTMs form a particular category of the RNN, first introduced by Hochreiter and Schmidhuber in 1997 [80], and modified to how they are known today by Gers et al. in 1999 [81]. Their advantage over standard RNNs is their good performance in learning long-term dependencies (10 timesteps or more), without struggling with vanishing gradients like standard RNNs [41, 42, 43]. Consequently, an LSTM can extract information from the EEG features' change during the 30 time-steps to make an outcome prediction.

Model information

LSTM networks are comprised of a special type of cell (Fig. 4), which computes different operations than a regular NN unit or RNN unit. An LSTM layer includes as many cells as the number of timesteps used as input. These cells connect to each other with respect to time. One can view an LSTM cell as a network on its own; it includes four NN layers composed of several units. Appendix N provides information about units in NNs. The following equations summarise the mathematics within a single LSTM cell.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{5}$$

$$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right) \tag{6}$$

$$i_t = \sigma\left(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i\right) \tag{7}$$

$$\widetilde{C_t} = tanh\left(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c\right) \tag{8}$$

$$h_t = o_t * tanh\left(C_t\right) \tag{9}$$

$$o_t = \sigma\left(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o\right) \tag{10}$$

LSTMs store information in what is termed the cell state, C_t. The cell receives the cell state from the previous timestep and passes it on to the next one. The cell can modify the state's information, based on the received input from current timestep, x_t, and the output from the previous timestep, h_{t-1}, using structures termed gates (5). The gates consist of a NN layer (f_t, i_t, o_t) and an element-wise multiplication, denoted by $*$ (the Hadamard product). The NN layers themselves are composed of several units using a sigmoid activation function (4), which outputs a value between 0 and 1. Multiplication by 0 causes complete removal of information and multiplication by 1 results in preserving all information. The cell can remove information from the state via the forget gate layer, f_t (6). Additionally, it can add new information to the cell state, via the input gate layer, i_t (7), and the candidate elements, $\widetilde{C_t}$ (8). The candidate elements are obtained by an NN layer that uses the hyperbolic tangent (tanh) activation function (11), which outputs values between -1 and 1. Thereby, the tanh function provides the possibility to either increase or decrease elements in the cell state.

$$tanh\,(z) = \frac{(e^{z} - e^{-z})}{(e^{z} + e^{-z})} \tag{11}$$

In this way, under careful regulation, the state is updated in each cell and evolves through time. Furthermore, the cell controls which parts of the cell state it generates as output, h_t (9), via the output gate layer o_t (10) [41, 43, 45, 46, 59].

Although hyperparameters, determined through optimisation, define the LSTM network, a general architecture can be sketched in advance. An LSTM cell can include one or more units, which refers to the number of units in the four NN layers within the cell. LSTM networks can be composed of one or more LSTM cell layers, which are the cells "unrolled" over the timesteps. Following the last LSTM layer, it is appropriate for binary classification of a NN to include a final dense layer composed of one unit with a sigmoid activation function [58, 59, 77].

$$\hat{y} = \sigma\,(W_{FC} \cdot h_t + b_{FC}) \tag{12}$$

In this equation, ht denotes the output from the final LSTM layer, and $\hat{y}$ the probability of a poor neurological outcome. All the W, U, and b terms denote the weights matrices and bias vectors, respectively, and form the model parameters. The model parameter estimation compromised of the following steps. The output was predicted for a batch of samples through equations in the LSTM cells and dense unit in the forward propagation. One sample included one epoch with the sequence of features of 30 time-fragments. For the first iteration, the model parameters were randomly initialised. Subsequently, the loss of this batch was computed between the model's predicted outcome and the true outcome for every time-fragment. The binary cross-entropy loss function computed the loss, as it is the most appropriate function for binary classification of NN [58, 59, 77]. Backward propagation through time (BPTT) computed the gradients of the model parameters across all timesteps. An optimisation algorithm adjusted the model parameters using the gradients. Finally, this process was repeated per batch for the defined number of epochs [41, 59, 77, 82, 83]. Appendix R offers a step-by-step explanation of the operations within the LSTM cell, the learning process, and the calculations of the BPTT.

Hyperparameters

Keras 2.3.1 upon Tensorflow 2.3.0 backend in Python 3.7. was used to build the LSTM network. LSTMs require the specification of many hyperparameters, including ones that concern building, compiling, and training the model. In general, hyperparameters achieving the highest performance, cannot be defined in advance [58, 59, 77]. Moreover, as no LSTMs were used for outcome prediction, specific hyperparameter choices could not directly be derived from literature (i.e. no transfer learning). It was technically not feasible to try all possible hyperparameters and select the best ones. Fortunately, several choices could be made with standard tricks suggested in the literature [77]. Moreover, the used framework generally provides good default options [59] which were used if no conclusive literature about the hyperparameter was found. Appendix S offers an in-depth description of all hyperparameters and defines a set of most promising options.

A random search was performed to find the best combination of hyperparameters from this promising set [59, 77]. The search was performed for both a one-layer and two-layer LSTM model. The input to the LSTM during the search were the features in the FFS extracted from all epochs. The search explored 25% of the full hyperparameter space defined by the promising set. In the search, the data were split into 80% training set and 20%

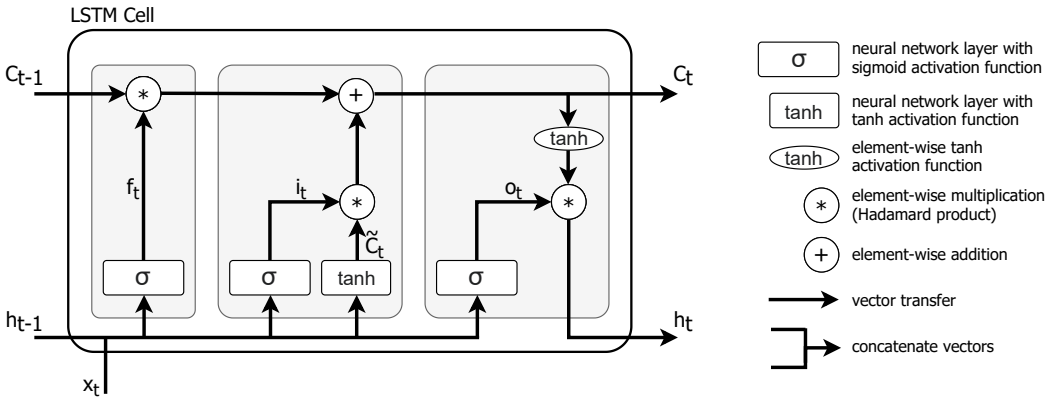


Figure 4: Schematic overview of an LSTM cell at one specific timestep t. C=cell state. $\widetilde{C_t}$=candidate state. f=forget gate layer. h=output. i=input gate layer. LSTM=long short-term memory recurrent neural network. o=output gate layer. σ=sigmoid activation function. tanh=hyperbolic tangent. x=input. The figure is inspired by figures from "Understanding LSTM Networks" by Olah, C., 2015, (https://colah.github.io/posts/2015-08-Understanding-LSTMs/)

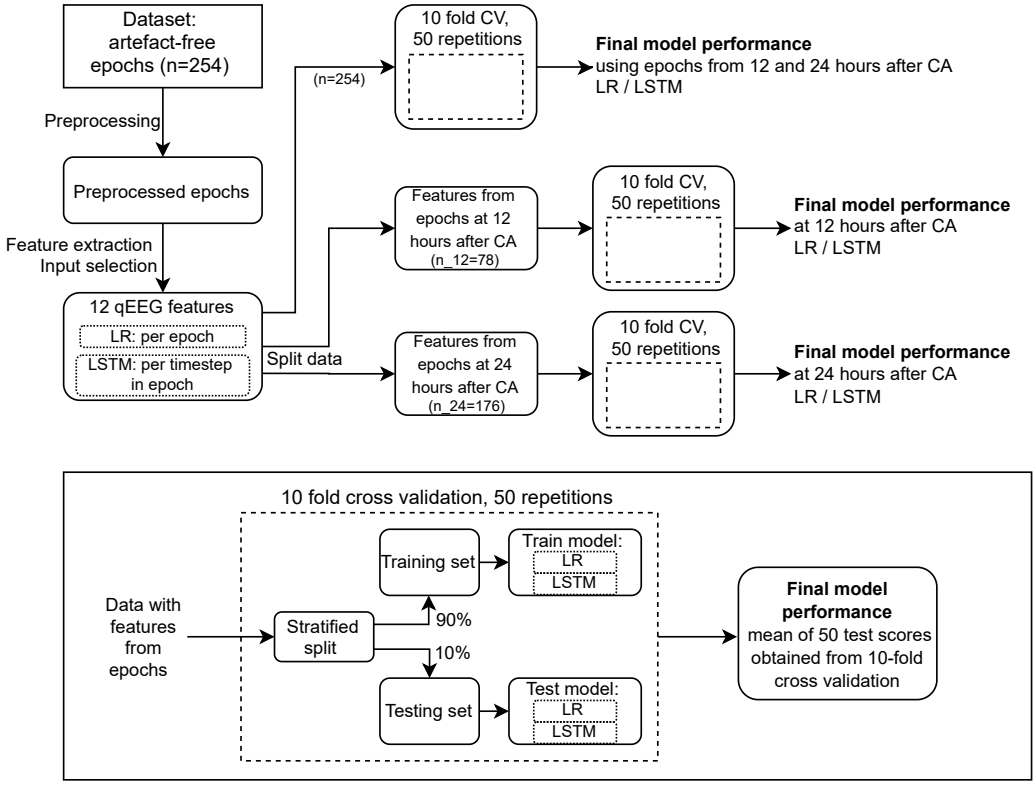


Figure 5: Full flowchart of data from raw dataset to the final model performance using the final feature set features. The 10-fold cross-validation is displayed in more detail in the rectangular box below the flowchart. CA=cardiac arrest. CV=cross validation. LR=logistic regression. LSTM=long short-term memory recurrent neural network. qEEG=quantitative electroencephalography.

validation set. Both sets consisted of a balanced number of epochs with good and poor outcomes. A probabilistic reduction was used to remove configurations that were likely to give a validation AUC lower than 0.80 to speed up the search. To obtain a more robust and reliable result from the hyperparameter optimisation, the random searches were repeated ten times. Each time the data were shuffled before split into training and validation data. The results from the ten searches were merged.

The hyperparameters' performance on the validation set was evaluated by inspecting boxplots of all hyperparameter options against the SeSp100 and AUC. The options for the hyperparameters that consistently obtained a high SeSp100 for all configurations were selected. Therefore, the selection was judged based on a high median and a small distribution, the latter was reflected in the plots by small boxes, short whiskers and little outliers. If the hyperparameters with the highest SeSp100 did not give a low AUC, they were used in the final model. After that, it was manually evaluated how layer weight regularisation enhanced performance. Finally, the best performing one-layer and two-layer configurations were compared, and the most optimal model was selected. Appendix T describes the optimisation more detailed.

2.9 Final performance evaluations and statistical comparisons

The LR and the LSTM were retrained with the optimal hyperparameter configuration to evaluate the final models' performance. The performance was quantified using the metrics in Section 2.6. The 10-fold stratified CV process was repeated 50 times to obtain a reliable and robust estimate of the model's performance [84]. For both the LR and the LSTM, separate models were trained and evaluated with features in the FFS using i) all epochs at both timepoints, ii) epochs recorded 12 hours after CA, and iii) epochs recorded 24 hours after CA. The models were trained with all epochs at both timepoints, as this offered a larger dataset than when using the separate recording times. As neural networks require a large dataset to reach optimal performance, training with epochs at both timepoints yielded the most reliable result. The reason for separately training the models at 12 hours and 24 hours after CA was twofold. Firstly, misleadingly high performances on the test set could have occurred when both recording times were used as input. These high performances could result from similar EEG epochs from the same patient recorded at 12 and 24 hours after CA present in both the training and test set. Secondly, training at both timepoints separately allowed for a fairer comparison

to published outcome prediction models that followed a similar approach. However, one should note that the small number of epochs available at 24 hours and the even smaller number at 12 hours after CA made it difficult to draw reliable conclusions about the performance at separate timepoints. Figure 5 visualises the full flow of data from raw epoch to the final model performance using FFS features.

Furthermore, separate models were trained and evaluated with features in the FFS and features in the AFSs using epochs at both timepoints, for both the LR and the LSTM. It was evaluated if AFSs showed better performance than the FFS when used as input to the final models.

The differences between the model's performance metrics were compared using an unpaired two-sample students t-test. The difference was considered statistically significant if $p<0.05$.

3 Results

3.1 Logistic Regression

Table IV shows the optimal hyperparameter configuration for LR. Appendix U detailly describes the results of the hyperparameter optimisation. Appendix Z.6 includes the LR Python script.

Table VI compares the LR model's performance separately trained and evaluated at both timepoints (12 and 24 hours), 12 hours, and 24 hours after cardiac arrest (CA). The input to the models were the features from the final feature set (FFS) (Table III). The area under the receiver operating characteristic curves (AUCs) were similar, being 0.89 at both timepoints, 0.90 at 12 hours, and 0.88 at 24 hours. The sensitivity at 100% specificity (SeSp100) for poor outcome prediction at 12 hours of 0.78 was significantly higher than that at 24 hours or both timepoints, which was 0.67 for both models. On the contrary, the sensitivity at 95% specificity (SeSp95) for good outcome prediction at 12 hours of 0.30 was significantly lower than that at 24 hours (0.56) or both timepoints (0.60). The model's performance at 24 hours was comparable to the performance at both timepoints after CA.

The additional feature set (AFS) adding clinical features age and sex to the FFS did not show statistically significant differences in performance metrics. Models trained with features from the AFSs did not outperform the model trained on the FFS on the SeSp100 or AUC. However, an AFS achieved higher performance on SeSp95 for good outcome prediction, but at the cost of a lower AUC, which was undesirable as the AUC was considered more important. Appendix V compares the performance of the LR models more detailly.

Table IV: Hyperparameters used for the logistic regression model. Keras 2.3.1 upon Tensorflow 2.3.0. Sequential class () [79]. NAG=Nesterov accelerated gradient. SGD=stochastic gradient descent.

Hyperparameter	
Layer (units)	Dense (1)
Activation function	Sigmoid
Kernel initialiser	Glorot uniform
Bias initialiser	Zeros
Loss function	Binary cross-entropy
Optimiser	SGD with NAG
Learning rate	0.01
Momentum term	0.9
Number of epochs	1500
Batch size	32

3.2 Long Short-Term Memory Recurrent Neural Network

Table V shows the optimal hyperparameter configuration for the LSTM. Appendix W detailly describes the results of the hyperparameter optimisation. Appendix Z.7 and Z.8 include the hyperparameter search and analysis Python scripts, respectively. Appendix Z.9 includes the final LSTM Python script.

Table VI compares the performance metrics of the LSTMs separately trained and evaluated at both timepoints (12 and 24 hours), 12 hours, and 24 hours after CA. The input to the models were the features from

Table V: Hyperparameters used for the LSTM model. Keras 2.3.1 upon Tensorflow 2.3.0. Sequential class () [79]. Tanh=hyperbolic tangent.

Hyperparameter	
LSTM layers	1
LSTM units	16
LSTM activation	Tanh
LSTM recurrent activation	Sigmoid
LSTM kernel initialiser	Glorot uniform
LSTM recurrent initialiser	Orthogonal
LSTM bias initialiser	Zeros
LSTM kernel regulariser	L1= 0.001, L2=0.001
LSTM recurrent regulariser	L1= 0.001, L2=0.001
LSTM bias regulariser	L1= 0.001, L2=0.001
Dropout rate	0.5
Recurrent dropout rate	0
Dense layers	1
Dense units	1
Dense activation	Sigmoid
Dense kernel initialiser	Glorot uniform
Dense bias initialiser	Zeros
Loss function	Binary cross-entropy
Optimiser	Adam
Adam - Learning rate	0.001
Adam – decay rates	$\beta_1 = 0.9$ $\beta_2 = 0.999$
Adam - Epsilon	1e-7
Number of epochs	120
Batch size	32

Table VI: Comparison of performance metrics of the LSTM and LR models separately trained and evaluated at both 12 and 24, 12 hours, and 24 hours after cardiac arrest. Both models were trained and evaluated with features from the final feature set. The following differences were statistically significant ($p<0.05$): For both models, i) between the sensitivities of the models trained at both timepoints and at 12 hours, and ii) between the sensitivities of models trained at 12 hours and 24 hours. Furthermore, for the LSTM model, iii) between the SeSp95 of models trained at both timepoints and at 24 hours. Finally, iv) between the SeSp95 at 24 hours of the LR and the LSTM. AUC=area under the receiver operating curve. LR=logistic regression. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity. SeSp95=sensitivity at 95% specificity.

	AUC mean (95% CI)	**SeSp100 (poor outcome prediction)** mean (95% CI)	**SeSp95 (good outcome prediction)** mean (95% CI)
LSTM – 12 h + 24 h	0.898 (0.893-0.903)	0.662 (0.648-0.676)	0.574 (0.551-0.597)
LR – 12 h + 24 h	0.893 (0.887-0.898)	0.669 (0.655-0.683)	0.599 (0.576-0.621)
LSTM – 12 h	0.901 (0.891-0.912)	0.785 (0.764-0.805)	0.303 (0.275-0.332)
LR – 12 h	0.901 (0.890-0.911)	0.783 (0.763-0.803)	0.295 (0.266-0.323)
LSTM – 24 h	0.901 (0.895-0.908)	0.681 (0.664-0.697)	0.439 (0.415-0.463)
LR – 24 h	0.883 (0.877-0.890)	0.671 (0.655-0.687)	0.557 (0.532-0.581)

the final feature set (Table III). The AUCs (0.90) were approximately equal. The SeSp100 at 12 hours of 0.79 was significantly higher than that at 24 hours or both timepoints, which were 0.66 and 0.68, respectively. On the contrary, The SeSp95 at 12 hours of 0.30 was significantly lower than that at 24 hours (0.57) or both timepoints (0.44). The SeSp100 of the model trained at 24 hours was comparable to both timepoints. The SeSp95 at 24 hours was significantly lower than that at both timepoints. Figure 6 shows the receiver operating characteristic curve with the SeSp100 and SeSp95 of the LSTM at both timepoints.

Using AFS that added the clinical features age and sex to the FFS or used all 19 extracted qEEG features (with clinical features) did not show statistically significant differences in the performance metrics. Models trained on features from the AFSs did not outperform the model trained on the FFS regarding the SeSp100 or AUC. Like the LR, higher performance on SeSp95 could be achieved using AFSs but at the cost of a lower AUC, which was undesirable as the AUC was considered more important. Appendix X compares the performance of the LSTMs more detailly.

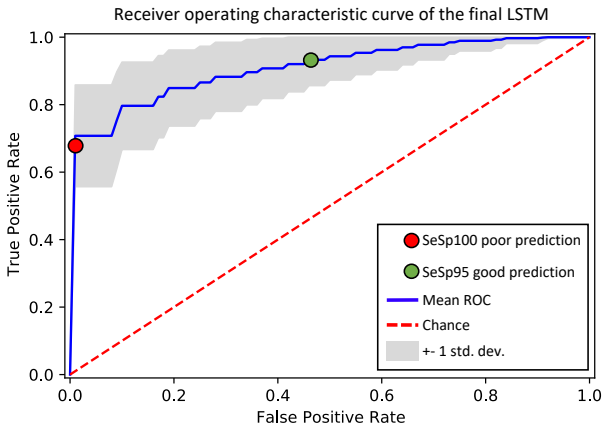


Figure 6: The receiver operating characteristic (ROC) curve of the LSTM model at both timepoints using the features from the final feature set. The red dot denotes the SeSp100 for poor outcome prediction, which was 0.662. The green dot denotes the SeSp95 for good outcome prediction, which was 0.574. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity. SeSp95=sensitivity at 95% specificity. std. dev.= standard deviation

4 Discussion

Using the same five-minute EEG recordings, the LSTM model achieved similar performance compared to the LR model in outcome prediction for patients suffering from a postanoxic coma (PAC) after cardiac arrest (CA). When statistically comparing the LSTM and the LR performances, the only significant difference was between the sensitivity at 95% specificity (SeSp95) for good outcome prediction at 24 hours after CA, where the LR model (SeSp95=0.56) outperformed the LSTM (SeSp95=0.44). Therefore, the hypothesis that the LSTM would outperform the LR was rejected. The high sensitivity at 100% specificity (SeSp100) for poor outcome prediction at 12 hours after CA for both models was the most remarkable finding. It would be very valuable in clinical settings if this SeSp100 is validated with more data. Sensitive prediction of poor outcome at 12 hours would lead to the reliable withdrawal of treatment of many unsolvable patients in a period smaller than a day.

The LR and LSTM achieved similar performance despite the LSTM's ability to account for the temporal characteristics within the epoch and map more complex feature-outcome relationships than an LR. As the similarity between both models was the same set of quantitative EEG (qEEG) features used for prediction, the following can be concluded. The predictive power of the LSTM was the result of the carefully selected features and most likely not from its ability to learn time-dependencies, nor its superior feature-outcome mapping,

as the LR did not have these abilities. Given that a relatively simple LR model could establish the feature-outcome relationship indicated a high degree of predictive power in the 12 qEEG features. The similar results between both models emphasised the necessity of good feature engineering, which might be more important than the choice of the model itself. Rubin made similar conclusions, stating that often no significant difference was found in performance between complex and simple classifiers when meaningful features were used [85].

A time frame of five minutes was likely too small to have provided additional prognostic information for outcome prediction, even though LSTMs using EEG from small timeframes aided in prediction of other classification or regression tasks [38, 39, 44, 45, 46, 86]. That the LSTM could not extract prognostic information from the temporal dynamics in the five-minute EEG, was assumably due to the gradual change in a time range of hours of EEG waves in comatose patients [10, 28]. Therefore, using recordings over a larger time frame might have increased predictive power of the model, but at the cost of less reliable comparability to previously published outcome prediction models.

When trained and evaluated at 12 hours after CA, the LSTM obtained a similar area under the receiver operating characteristic curve (AUC), a significantly increased SeSp100 for poor outcome prediction, and a significantly decreased SeSp95 for good outcome prediction, compared the models trained at both timepoints and 24 hours after CA. So, at an earlier timepoint, the model could predict a poor outcome with higher sensitivity, at the cost of predicting good outcome with lower sensitivity. The evolution of the EEG patterns could explain this sensitivity difference between timepoints. The cortical activity of poor outcome patients often changes to a somewhat more continuous pattern over time. In general, continuous EEG activity is associated with a good outcome. As a result, the poor outcome EEG patterns at later timepoints display more resemblance with EEG patterns associated with good outcome [10, 12, 14, 16, 18, 87]. Therefore, prognostication could be better at 12 hours than at 24 hours after CA.

Similarly, the decreased sensitivity for good outcome prediction at 12 hours can be explained. For some patients with a good outcome, the continuous cortical activity has not yet remerged at the early timepoint after CA, either due to targeted temperature management, sedation, or the brain's physiological recovery. Some of the good outcome EEG patterns will have evolved to a continuous pattern at 24 hours, increasing the sensitivity at that timepoint. The difference in the model's performance at 12 and 24 hours was not the result of a difference in treatment, as this was equal at both timepoints.

Independent of the timepoints after CA, SeSp100 for poor outcome prediction significantly outperformed SeSp95 for good outcome prediction. As SeSp100 was considered more important, this was a favourable result. The difference in the sensitivities was most likely the result of optimising the model for SeSp100 without considering the SeSp95. The hyperparameters, as well as the feature sets, were selected to reach an optimal SeSp100. The influence of the choice of qEEG features on the sensitivities of the models is evident when comparing the performance of the final feature set (FFS) with the additional feature sets (AFSs) (Appendix X). One set of features (AFS 5) reached a SeSp95 of 0.72 using epochs at both timepoints, which was significantly higher than the features in the FFS reached (SeSp95=0.57). However, as the AUC of this AFS was significantly lower, it was deliberately not used for further analysis. It should be noted that patients with good outcome received a higher mean dose of the sedative propofol than poor outcome patients. Although propofol influences the EEG, it was unlikely that it influenced the model's sensitivity for good outcome prediction, as it does not negatively affect the prognostic value of the background pattern [18, 19].

4.1 Comparison to other studies

The models developed in this study predicted poor outcome with notably higher sensitivity compared to the visual analysis of the EEG, which offers a reliable prediction of poor outcome in only half of the patients [12, 14, 15, 16]. Table VII compares the LR and LSTM to previously reported outcome prediction models. The table only includes the most recent cerebral recovery index (CRI), the revised CRI (rCRI), as it outperformed all previous versions [18]. The AUCs of the LR and LSTM were equal to or higher than most other models. Furthermore, the LR and LSTM achieved the highest sensitivity for poor outcome prediction of all models. Other models outperformed the LR and LSTM at specific timepoints regarding the sensitivity for good outcome prediction. As this sensitivity was not the focus of this study, it was not further considered it in this comparison. The LSTM achieved the highest performance on all metrics 24 hours after CA compared to the current literature. As PAC patients' EEG recording often starts between 12 and 24 hours after CA, more epochs are generally available at 24 hours. Therefore, the LSTM offers the most reliable outcome prediction at the most convenient timepoint. However, the LR and LSTM achieved remarkably high SeSp100 at 12 hours for poor outcome prediction compared to the other timepoints and compared to all other published models. Moreover, the majority of the other models also showed more accurate performance at 12 hours. Therefore, it could be argued that the standard treatment routine in the intensive care unit (ICU) should change and EEG recording should always start 12 hours after CA.

Table VII: Comparison of the performance of the outcome prediction models on the test sets from multiple studies. *The model was evaluated on two different (in.=internal, ex.=external) independent test sets. AUC=area under the receiver operating curve. CA=cardiac arrest. CI=confidence interval. CNN=convolutional neural network. LR=logistic regression. LSTM=long short-term memory recurrent neural network. RFC=random forest classifier. Se (at Sp)=sensitivity at specificity. Std=standard deviation.

	Recording time after CA	**Sample size at time**	**AUC** mean (95% CI or std)	**Se (at Sp) for poor outcome prediction** mean (95% CI or std)	**Se (at Sp) for good outcome prediction** mean (95% CI or std)
12 features current LR	12 h + 24 h	254	0.89 (0.89-0.90)	0.67 (0.66-0.68) (100%)	0.60 (0.58-0.62) (95%)
	12 h	78	0.90 (0.89-0.91)	0.78 (0.76-0.80) (100%)	0.30 (0.27-0.32) (95%)
	24 h	178	0.88 (0.88-0.89)	0.67 (0.66-0.69) (100%)	0.56 (0.53-0.58) (95%)
12 features current LSTM	12 h + 24 h	254	0.90 (0.89-0.90)	0.66 (0.65-0.68) (100%)	0.57 (0.55-0.60) (95%)
	12 h	78	0.90 (0.89-0.91)	0.79 (0.76-0.81) (100%)	0.30 (0.28-0.33) (95%)
	24 h	178	0.90 (0.90-0.91)	0.68 (0.66-0.70) (100%)	0.44 (0.42-0.46) (95%)
8 Features Bayes classifier [32]	<24 h	94 in total	0.81	0.54 (100%)	
2 Features [29]	12 h	316	0.86 (0.82-0.90)	0.50 (0.42-0.57) (100%)	0.52 (0.44-0.59) (90%)
	24 h	464	0.87 (0.83-0.91)	0.42 (0.35-0.48) (100%)	0.57 (0.48-0.67) (90%)
44 Features RFC [18]	12 h	335	0.94 (0.83–0.99)	0.66 (0.65–0.78) (100%)	0.72 (0.61–0.85) (95%)
	24 h	480	0.88 (0.78–0.93)	0.60 (0.51–0.75) (100%)	0.40 (0.30–0.51) (95%)
56 Features LR [52]	1-12 h	438 in total	0.71 (± 0.05)	-	∼0.43 (95%)
	13-24 h		0.70 (± 0.06)	-	∼0.41 (95%)
	Overall (1-72 h)		0.83 (± 0.08)	-	0.46 (± 18) (95%)
EEG signal CNN [21]	12 h	287	0.89	0.58 (100%)	0.58 (97%)
	24 h	399	0.76	∼0.30 (100%)	-
EEG signal CNN [20]*	12 h in. test	374	0.87 (0.87-0.88)	0.42 (0.36-0.48) (100%)	0.48 (0.45-0.51) (95%)
	24 h in. test	534	0.90 (0.90-0.91)	0.57 (0.54-0.60) (100%)	0.33 (0.30-0.36) (95%)
	12 h ex. test	167	0.92 (0.90-0.94)	0.58 (0.51-0.65) (100%)	0.48 (0.45-0.51) (95%)
	24 h ex. test	239	0.88 (0.86-0.90)	0.51 (0.49-0.53) (100%)	0.22 (0.20-0.25) (95%)
EEG signal CNN [6]	20.2 (+-6.1)	267 in total	0.89 (0.78-0.96)	0.78 (89%)	-

The hypothesis that the LSTM would achieve better performance than the previously reported EEG-based outcome predictors was accepted. However, the superior performance of the LSTM compared to the models was not the result of accounting for the EEG signals' changes within the five minutes epochs on which this hypothesis was based, as the LR performed equal to the LSTM.

The superior performance of the models in the current study over the Bayes classifier created by Zubler et al. [32], was likely due to the higher predictive power of the by me selected qEEG features and the fact that a Bayes classifier is generally outperformed by an LR [88, 89] and by neural networks [59].

The models developed in this study outperformed the model of Ruijter et al. [29], assumably because they only used two features and no machine learning (ML) algorithm. To date, Nagaraj et al. developed the best performing outcome prediction model (rCRI) [18]. The rCRI was the only published model that outperformed the LSTM on specific metrics. Specifically, the rCRI achieved a higher AUC (0.94 vs. 0.90) and SeSp95 for good outcome prediction (0.72 vs. 0.30), but a lower SeSp100 for poor outcome prediction (0.66 vs. 0.79) at 12 hours after CA. At 24 hours, the LR and LSTM outperformed the rCRI on all metrics. For all performance metrics, the 95% confidence interval was notably larger for the rCRI than for the LR and LSTM, indicating the latter's more stable behaviour. However, while comparing these performances, the significantly larger sample size used for the rCRI should be considered, making its results more reliable. The superior SeSp100 for poor outcome and the partially inferior SeSp95 for good outcome of the LR and LSTM compared to the rCRI could not be explained by the fact that the models in this study were optimised for SeSp100 for poor outcome prediction, as the rCRI was also optimised for this metric. The superior SeSp95 of the rCRI at 12 hours could result from its significantly larger available dataset for training, as ML models' performances generally increase with more data [59]. The rCRI might also have used more features associated with good outcome at that timepoint. As the rCRI used 44 qEEG features, it is assumable that some of these aided in predicting good outcome. On the other hand, although a selection was made by an algorithm during training, this large number of features might have resulted in the inferior performance on specific metrics compared to the LR and LSTM. As these features were not selected based on proved predictive power, as has been done in the current study, they might not have been ideal for PAC outcome prediction and could have led the model in the wrong direction. Moreover, there is a good chance that multiple features used by Nagaraj et al. were highly correlated, causing bad generalisation [56, 57]. No evidence was found to support that either an LR or a random forest classifier, used for the rCRI, would achieve the highest performance. Multiple stud-

ies compared the two models, and the better performing model differed per study [90].

Likely for similar reasons concerning the usage of a large number of features, the LR and LSTM outperformed the model developed by Ghassemi et al., which made use of 56 features and an LR with elastic net regularisation [52].

The superior performance of the LR and LSTM over the convolutional neural networks (CNNs) [6, 20, 21] on most metrics could result from the fact that I used features as input to my models as opposed to raw data. Extracting features provides the model with important hidden information within the signal [39]. The EEG signal itself might contain much noise, making it harder for the CNNs to extract the relevant information for feature prediction.

4.2 Limitations and future directions

A limitation was the small dataset available. Resultingly, the performance of the LSTM in this study might deviate from the performance obtained when using more data. Moreover, the small dataset led to the necessity to optimise, train and evaluate models using epochs at both timepoints simultaneously. As the EEG patterns change over time [10, 28], this formed a limitation. More accurate models have been obtained if they were separately developed for either 12 or 24 hours after CA. For example, different feature sets used for both timepoints could have increased performance, as Ghassemi et al. showed that accounting for the change over time of the association between qEEG features and neurological outcome improved their outcome prediction model [52]. This study's results and most other outcome prediction studies showed more accurate models at 12 hours after CA. The LR's and LSTM's very promising SeSp100 for poor outcome prediction at 12 hours should encourage future research to validate and optimise these models with a larger dataset. For example, one could perform the hyperparameter optimisation with only data at 12 hours or a feature set could be designed explicitly for poor outcome prediction at that timepoint.

The comparison of other outcome prediction models to the models developed in this study has its limitations due to i) the significant differences in the datasets, both in terms of quantity and quality, ii) the different methods used for validation, and iii) the different metrics used to optimise the models. A fairer comparison could be made by evaluating the LR's and LSTM's performance with the datasets used in other studies and evaluating the performance of the models developed in other studies with this dataset.

Future research should investigate if LSTMs achieve better performance if provided with the temporal evolution of the EEG over several hours or days instead of five-minute recordings. Furthermore, research could look into the variability of the features within the five-minute epoch. If the variability is small, it will strengthen the assumption that the time frame was too small to provide prognostic information and strengthen the conclusion that the similar performance achieved by both models was the result of the specific set of features.

Using features as input to the LSTM model might have limited its predictive power if additional prognostic information was present in the EEG signal, which could have been learned by the LSTM. Future research should explore using the raw EEG signal as input to the LSTM.

During the feature extraction process, the features were averaged across all channels. It was assumed that no information was lost by averaging, as PAC affects the whole brain and not specific localised regions [52]. This assumption was adapted from previously reported outcome prediction studies [18, 20, 52, 54]. Considering most developed models were reliable outcome predictors, this assumption might be valid. However, no study compared the input from all channels to input averaged over all channels. Therefore, prognostic information might be lost in averaging over all channels, decreasing the model's predictive accuracy. Future research is needed to exclude this possibility.

The limited computational resources available formed another limitation, which mostly affected the random search space dimension. Future research should incorporate more hyperparameters in the search space to find the most optimal configuration. In Appendix Y discusses recommendations for improved hyperparameter optimisation in detail.

An artefact-detection algorithm automatically selected the artefact-free epochs. However, the possibility that artefacts were still present in the epochs cannot be excluded. Consequently, the features might be influenced by the remaining artefacts. Ideally, an expert visually inspects the EEG epochs to ensure artefact free data. The patients received targeted temperature management, and sedative medication, of which the former barely affects EEG in general [91] and the latter does not influence the prognostic value of the background EEG pattern [18, 19].

This study focused on outcome prediction based on qEEG features. Integrating features from other modalities might offer more accurate outcome prediction, but this was outside of the scope of this study. Further research could investigate if a multimodal approach improves the performance of an outcome prediction model. For example, information from the clinical examination with high sensitivity could be incorporated, like the absence of somatosensory evoked potential responses or pupil reflexes [8, 9]. Furthermore, Wennervirta et al. and Stammet et al. both reported improved accuracy of outcome prediction by combining qEEG features with biochemical markers [30, 92].

This study used data from only one centre, which precluded the models' generalisability to other centres. External validation is required to validate the LR and

LSTM further.

As with all other outcome prediction after CA studies, the risk of self-fulfilling prophecy exists, because treating physicians were not blinded for the EEG [18, 19, 20, 29, 32, 34, 35, 36, 52]. However, physicians followed the Dutch recommendations for prognostication in PAC for decisions regarding withdrawal of life-supporting treatment, which excludes the EEG within 72 hours after CA. Moreover, this study was performed offline, so the results of the outcome prediction models were not available to physicians. For these reasons, the risk of self-fulfilling prophecy was minimal.

4.3 Key points

The LR and LSTM developed in this study predicted patients' neurological outcome six months after CA equally good while using the same set of quantitative EEG features extracted from five-minute epochs recorded at 12 or 24 hours after CA. The prognostic performance of the LR did not increase using the LSTM, which accounts for the temporal characteristics within the epoch. Future research should investigate LSTMs with features (or raw EEG data) over larger timeframes for outcome prediction. Careful feature selection is more crucial than the variation of these features within a five-minute epoch, as emphasised by the high predictive performance of the LR. The highest AUC and sensitivity at 100% specificity for poor outcome prediction were achieved at 12 hours after CA. Therefore, future research should investigate how EEG measurements at earlier timepoints can further improve prognostication. Prediction of poor outcome at 12 hours after CA is clinically more valuable than prediction at 24 hours: it could lead to the earlier withdrawal of unnecessary treatment of many unsalvageable patients. Consequently, resource use in the ICU and the associated healthcare cost could be decreased. Moreover, early decision making would spare the family of the patient.

Acknowledgements

Throughout the writing of this thesis, I have received a great deal of support. First of all, my gratitude goes to my daily supervisor Wouter Potters. Wouter, your expertise and enthusiasm have been a driving force during these past months. Thank you so much for always being available for assistance every day of the week and all the effort you put in our collaboration. Moreover, I would like to thank my TU Delft supervisor, Alfred Schouten, for the time he took to help me the past year and for his useful feedback. I felt like I could always go by your office for any question. Furthermore, I would like to give my thanks to Winfred Mugge, for his feedback, which brought my work to a higher level. Finally, I would like to thank my parents and friends for their support along the way.

Appendices

A. Background information about electroencephalography

Mechanism of EEG

In short, EEG measures the electrical activity produced by neurons in the brain at the scalp surface with electrodes. Due to their excitability neurons can generate extracellular currents. The activity of large populations of neurons summed together can create electric fields that are large enough to be picked up at a distant site from the current source, like the electrodes placed on the scalp. These electrodes connect to a machine that displays the voltage difference between two electrodes as a signal in microvolts in time. Ohm's law can explain the simplified mechanism: the voltage difference between two electrodes is dependent on the current through a conductor and the resistance. In the brain, the current is created by is the summation of neuronal activity. The layers of tissue in the head and material between the scalp and the electrodes form the resistance [93, 94, 95, 96]. The recorded activity is filtered, resulting in the brain activity of interest, and amplified. The activity is then per electrode visualised as a signal in a specific channel. In a bipolar montage, the channel presents the voltage differences between neighbouring electrodes. The monopolar montage uses a common reference electrode for all channels. Therefore, the signal presents the voltage difference between an electrode and the common reference electrode [97].

Brain activity

The brain activity recorded with EEG in the absence of a stimulus is termed spontaneous brain activity. This activity indicates that the brain generates information independently of external factors and thus not only processes stimuli. Therefore, it is also called self-generated, self-organised activity, or background activity. There is little understanding of what exactly drives the brain to generate these activities [98].

The brain network can support activity in different distinguishing oscillatory behaviours, determining individual neurons' firing patterns [98]. These behaviours are mostly sinusoidal waves with amplitudes varying from 20 to 100 microvolt. The waves are unique for each individual [71, 72, 95]. With a spectral analysis, the EEG signal is quantitatively analysed. The power of a frequency band and its spatial distribution are primarily studied. The power spectral density, or short power, shows how much of a frequency band is present in the EEG signal. Power is expressed in the amplitude squared per Hertz, so often in microvolts^2 per Hertz. The brain patterns are categorised into five major frequency bands [71, 72, 73, 95, 98, 99, 100].

- Delta (0-4 Hz). Delta waves form the slowest waves with the highest amplitude. Delta appears during deep sleep, and the activity source is thalamocortical or cortical.
- Theta (4-7 Hz). Theta waves are more irregular and occur during sleep or sometimes during deep concentration or mediation. Theta activity is uncommon in wake adult but appears in children. The source of the activity is cortical or hippocampal.
- Alpha (8-13 Hz). The alpha waves show a regular rhythm, with a moderate amplitude of approximately 50 microvolts. Alpha is present all wakeful states, but its power increases during relaxation and closure of the eyes. Hypotheses of the genesis of alpha are discussed shortly.
- Beta (14-30 Hz). Beta waves are rhythmic and semi-regular, with a low amplitude and dominate during a wakeful state with intense mental activity. The generator of the beta frequency is cortical.
- Gamma (30-80 Hz). Gamma waves are the fastest in this empirical classification; it is associated with parallel information processing from different brain areas. The cortex generates the gamma waves.

Aside from this empirical classification, a sixth category of frequency bands can be distinguished: high-frequency oscillations [101]. These subdivide into high gamma (50-125 Hz), ripple (125-250 Hz) and fast ripple (250-500 Hz) frequency bands. High-frequency oscillations relate to memory processing and originate from distributed areas in the cortex and limbic brain [102]. Recent studies show high-frequency oscillations as a biomarker of epilepsy [100, 103].

B. Electroencephalography following cardiac arrest

EEG activity after cardiac arrest

The human brain's electrophysiology from the onset of CA and the time immediately after that is not systemically studied [104]. Consequently, little information is available about the EEG during CA. However, various animal models studied neurophysiology after CA with EEG recordings [28]. Borjigin et al. induced CA in rats and recorded brain activity [104]. In the early period after CA, four sequential states were distinguished in all rats (Fig. A.1). The last heartbeat immediately initiated CA state one. A distinct decrease in the EEG amplitude and increased power in the gamma frequency band around 130 Hz in all channels characterised this state. The loss of oxygenated blood pulse marked the end of this state. CA state two showed increased theta

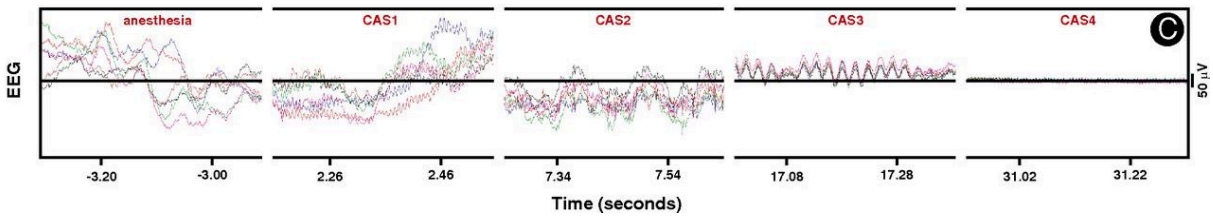


Figure A.1: Six EEG channels from a single rat's brain during the four states following CA. The six brain regions included: right frontal, left frontal, right parietal, left parietal, right occipital and left occipital areas. Time 0 seconds indicated the start of CA, induced by an injection into the heart of potassium chloride. The immediate period after CA divided into four different states. CAS1 (CA state 1): this state started with the last heartbeat and ended with the loss of oxygenated blood pulse. A distinct decrease in the EEG signal amplitude and increased power in the gamma frequency band around 130 Hz in all channels characterised CAS1. CAS2 (CA state 2): showed increased theta and high-frequency gamma power and terminated with bursts of delta waves. CAS3 (CA state 3): the amplitude of the signals dropped below ten μV. Low-frequency synchronous gamma (35-50 Hz) waves were seen, coupled to theta waves. CAS4 (CA state 4): the signal remained below ten μV. The frequency range consisted mostly of very high-frequency signals of 300 Hz, which continued without changing characteristics for the rest of the recording duration. CA=cardiac arrest. EEG=electroencephalogram. Adapted from "Surge of neurophysiological coherence and connectivity in the dying brain" by Borjigin et al., 2013, Proceedings of the National Academy of Science, 110(35), p. 14433 © 2013 by the Author(s) [104].

and high-frequency gamma power. The state terminated with bursts of delta waves. In the third state, the amplitude of the signals reduced to below ten microvolts. Mostly power in the low-frequency gamma band (35-50 Hz) was seen. These waves appeared synchronous and coupled to theta oscillations. In the final state, CA state four, the signal remained below ten microvolts. The frequency range consisted mostly of very high-frequency signals of 300 Hz. The low voltage high-frequency waves continued without changing characteristics for the rest of the recording duration [104].

Some electrophysiologic data of humans is available during CA. An EEG recording of a patient in two-minute asystole during carotid endarterectomy showed suppression of activity on all channels. Electrical cerebral silence reflected in the EEG within 10 seconds of asystole onset. EEG activity reappeared 15-20 seconds after the start of chest compressions. The waves were initially of high frequency and low voltage. Progressively, the physiological EEG waves returned [105]. In another case, where the patient suffered from 27-second lasting asystole, physiological EEG activity reemerged instantly when spontaneous circulation was returned [106]. In a different study, EEG recordings were available during a controlled period of CA necessary for a specific type of heart operation. Slowing to delta frequency and attenuation or complete loss of all faster activity was observed in most patients. Changes in the EEG occurred within a mean of 10 seconds after CA. Physiological EEG activity reemerged as soon as cardioversion was initiated [107]. These studies concluded that cerebral activity disappeared rapidly after CA, resulting in an isoelectric EEG within 10 seconds. Moreover, when blood circulation was established, EEG activity reappeared and returned to physiological waves. The abolishment of electrical activity reflects the large-scale failure in cortical synaptic transmission [108].

EEG activity after resuscitation

Jørgensen and colleagues studied the evolvement of the continuously monitored EEG after resuscitation. They stated the prognostic significance of the appearance of particular EEG patterns within a certain period after CA [109, 110, 111]. In 1998, Jørgensen and Holm defined noticeable features of postanoxic cerebral recovery during patients' unconscious state. They categorised the features into four phases. During the first phase, they only observed cranial nerve reflexes. The return of cephalic reactivity characterised the second phase. In these two phases, patients displayed electrical cerebral silence, which reflected in an isoelectric EEG. Hereafter a phase of intermittent cortical activity, also termed burst-suppression, appeared on the EEG. In the final phase, the burst-suppressions progressed to continuous cortical activity. Subsequently, the patient could regain consciousness [109]. Jørgensen and colleagues observed patients who had a good neurological outcome. The EEG activity reemerged after ten minutes to 8 hours after the first phase. In the majority of the patients, a burst-suppression period preceded continuous cortical activity. Some cases showed an immediate appearance of continuous EEG activity. In all patients, the EEG activity showed a progressively developing increase in amplitude and frequency [110]. Furthermore, Jørgensen and colleagues observed patients who had a poor neurological outcome. They found that the time-to-appearance of any initial EEG activity was significantly longer than in the patients with a good neurological outcome. This time-to-appearance ranged from 15 minutes to 124 hours after resuscitation. Moreover, in the majority of the poor-outcome patients, continuous cortical activity did not reemerge. If continuous cortical activity appeared on the EEG, the preceding period with a burst-suppression pattern was more prolonged

than in patients with a good neurological outcome [111]. However, at the time of Jørgensen and colleagues' reports, post-resuscitation care did not include targeted temperature management (Appendix E). As cooling the patients reduces the brain's oxygen demand, brain activity recorded nowadays could deviate from the patterns described in the 1980s. However, similar to Jørgensen and colleagues, more recent studies also emphasised the importance of the EEG's evolution towards continuous activity for a good outcome. If continuous physiological activity reappeared in the first 12 hours after resuscitation, the neurological outcome was most likely good. On the other hand, if no evolvement towards continuous activity appeared within the first 24 hours, a poor outcome is inevitable [10, 12, 14, 16, 87]. Combining results of different studies and using standardised terminology [10, 25, 87, 112], Hofmeijer and van Putten distinguished six categories of patterns in postanoxic patients [17]:

1. Iso-electric EEG
2. Low voltage EEG ($< 20\ \mu$V)
3. Burst-suppression EEG and the subcategory of burst-suppressions with identical bursts
4. Epileptiform EEG, including status epilepticus and generalised periodic discharges
5. EEG with continuous activity less than 8 Hz (diffused slow EEG)
6. EEG with continuous activity equal to or greater than 8 Hz ("normal" EEG)

Several studies showed that EEG patterns in the categories isoelectric, low voltage, and burst suppressions (especially identical bursts) were associated with poor neurological outcome. Continuous EEG patterns in either of the two categories were associated with good neurological outcome. Other EEG patterns, like status epilepticus or generalised periodic discharges, formed in-between patterns due to their inconsistent association with a specific outcome [5, 8, 10, 12, 16, 87, 113, 114, 115]. The EEG patterns are dynamic. The waves' changes are gradual and seen in a time range of hours [10, 28]. Figure A.2 shows EEG recordings at different times after CA from two patients suffering from a postanoxic coma.

C. EEG-based machine learning outcome prediction models

Various studies used machine learning models for outcome prediction of PAC with qEEG features as input. First of all, a series of studies created and improved the CRI. The CRI is a single index number that positively correlates with the probability of a good neurological outcome. The studies hourly extracted qEEG features from five-minute EEG epochs, on which they hourly predicted neurological outcome. Tjepkema-Cloostermans

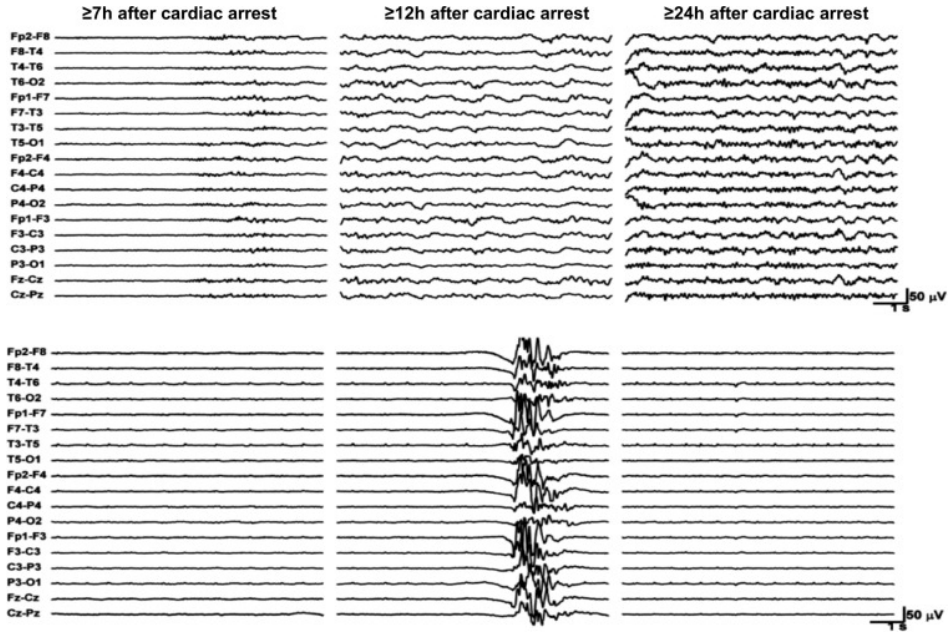


Figure A.2: EEG patterns from a patient with a good neurological outcome (upper panel) and poor neurological outcome (lower panel) recorded at 7 hours, 12 hours and 24 hours after cardiac arrest. The EEG of the patient with a good neurological outcome evolved to continuous physiological EEG activity within 12 hours after CA. The EEG of the patient with a poor neurological outcome evolved from isoelectric, to burst suppression and eventually showed a low voltage EEG pattern. CA=cardiac arrest. EEG=electroencephalography. Adapted from "EEG in a postanoxic coma: Prognostic and diagnostic value" by Hofmeijer, J. and van Putten, MJAM, 2016, Clinical Neurophysiology, 127, p. 2050, © 2016 International Federation of Clinical [17].

et al. created the first CRI in 2013. They calculated five qEEG features and combined them with equal weights to form one index value [19]. The performance of the original CRI was optimised using an increased number of qEEG features and an LR model [34] or a random forest classifier [35]. Most recently, Nagaraj et al. presented the revised CRI, which extracted 44 qEEG features and used the least absolute shrinkage and selection operator (LASSO) to select the most optimal features as input for a random forest classifier. Additionally, the features were extracted per 10-second fragment within the five-minute epoch and averaged across all 30 fragments. With the prediction of poor neurological outcome, the revised CRI reached an AUC of 0.94 and 0.88 at 12 and 24 hours after CA, respectively. Moreover, it could predict a poor neurological outcome at 100% specificity with a sensitivity of 0.66 and 0.60 at 12 and 24 hours, respectively. To date, the revised CRI is the best performing qEEG-based ML model for outcome prediction in PAC [18]. Next to the CRI series, other qEEG-based studies used different methods for outcome prediction. One study extracted eight qEEG features from five-minute hourly epochs for the input to a Bayes classifier [32]. Another study fed two features to a self-designed prediction model [29]. Lee et al. studied outcome prediction in children. They also extracted qEEG features hourly from five-minute epochs but took a different approach and averaged those over an early time interval (0-17 hours) and a late one (> 18 hours). They trained three different models with this data: an LR, a support vector machine, and a random forest classifier. The latter model and the early time interval showed the best performance [54]. Ghassemi et al. explored the time dependencies of qEEG features. They extracted 56 features per five-minute epoch per hour and used an elastic net to identify predictive features per 12-hour time interval. Subsequently, they trained LRs every 12-hour time interval using the current and preceding time intervals' feature information. This time-sensitive model outperformed one that used time-independent features, indicating that the prognostic importance of qEEG features alters over time [52]. Another study compared parametric and nonparametric models using clinical features combined with one qEEG feature extracted per-hourly epoch [36]. To a lesser extent, deep neural networks are studied in outcome prediction after CA. Two groups fed the raw EEG data of five-minute epochs, divided into 10-second fragments, to an CNN. The probability of poor or good neurological outcome was based on the mean probability of all 10-second fragments. The authors trained several models, using data from 12 or 24 hours after CA [20, 21] or by combining both recording times as input [20]. Furthermore, a study used raw data with a mean latency of 20.3 (+- 6.1) hours after CA for a one-dimensional CNN. The input data consisted of five-minute epochs divided into fragments [6]. To date, Tjepkema-Cloostermans et al. present the best performing CNN: predicting a poor outcome had an AUC of 0.87-0.92 and 0.86-0.90 at 12 and 24 hours after CA, respectively. Moreover, the model predicted poor outcome with a sensitivity at 100% specificity of 0.58 and 0.51 at 12 and 24 hours after CA, respectively [20]. Table A.I summarises the characteristics of the mentioned studies.

D. EEG-based LSTMs

Over the last years, deep learning networks were used for EEG classification tasks, including seizure detection, sleep stage scoring, emotion recognition, and classification of motor (imagery) tasks [50]. Various studies showed that LSTMs outperform other models like decision trees [39, 46], support vector machines [39, 44, 45, 46, 47, 48, 49], logistic regressions [46, 47], random forest classifiers [45, 46], naïve Bayes [46], feed-forward neural networks [47, 50], deep belief networks [40, 48], and even CNNs [40, 45, 46, 48]. The superior performance in EEG classification of LSTMs over other models is likely due to their ability to account for time dependencies. As EEG is time-series data, preserving temporal characteristics might significantly improve the model's accuracy [37, 38, 39, 40]. The majority of the architectures consisted of one or two LSTM layers, followed by one or two fully connected layers. Input to the LSTM compromised mostly features extracted from EEG signals. However, the signal itself and images were also used [50]. Several EEG-based studies compared the use of features to the raw EEG signal as input for the LSTM layer. Usage of the signal itself consistently and massively underperformed in these comparisons [38, 39, 44]. Table A.II summarises the characteristics of the mentioned.

Table A.I: Characteristics of outcome prediction models. Continued on next page.

	[20]	[21]	[18]	[35]	[34]	[19]
Time of EEG	12 h 24 h	12 h 24 h	Start a.s.a.p Hourly up to 72 h	Start a.s.a.p Hourly up to 72 h	Unknown	Hourly up to 48 h Every 2 hours up to 72 h
EEG fragment length	5 min Artefact free selected	5 min Artefact free selected	5 min Artefact free selected	5 min Artefact free selected per hour	5 min Artefact free selected	5 min Artefact free selected
Sample size	895	686	551	283	167	109
Input	5 min epoch divided into 30 non-overlapping 10 sec fragments	5 min epoch divided into 30 non-overlapping 10 sec fragments	5 min epoch divided into 30 non-overlapping 10 sec fragments	5 min epoch	5 min epoch	5 min epoch divided into 30 non-overlapping 10 sec fragments
Input type	Raw data	Raw data	44 qEEG features from each fragment averaged across epoch	9 qEEG features per epoch	10 qEEG features per epoch	5 qEEG features from each fragment averaged across epoch
Model	DL CNN trained at 1. t=12 h 2. t=24 h	DL CNN trained at 1. t=12 h 2. t=24 h	RFC LASSO trained at each hour after CA	RFC	LR	Features combined with equal weights
Output	Probability of good outcome mean of each 10 sec fragment Good=CPC 1-2 Poor=CPC 3-5	Good / poor outcome Good=CPC 1-2 Poor=CPC 3-5	Probability of good outcome mean of each 10 sec fragment Good=CPC 1-2 Poor=CPC 3-5	Probability of good outcome Good=CPC 1-2 Poor=CPC 3-5	CRI (correlates with probability of good outcome) Good=CPC 1-2 Poor=CPC 3-5	CRI (correlates with probability of good outcome) Good=CPC 1-2 Poor=CPC 3-5
Time of outcome	6 months	6 months	6 months	6 months	Hospital discharge	6 months
Train/Val	- 80% for train/val with 10-fold CV - 20% test - extra test set	- 80% train - 20% val	- 80% for train/val with 10-fold CV - 20% test	- 50% train - 50% val	Leave one out CV	- 50% train - 50% val
AUC	Poor t12: 0.92 Poor t24: 0.88	Poor t12: 0.89 Poor t24: 0.76	Poor t12: 0.94 Poor t24: 0.88	t12: 0.92 t24: 0.90	overal 0.0075 higher than tc2013	t12: 0.74 t24: 0.87
Se (at Sp)	Poor 12: 0.58 (100%) Poor 24: 0.51 (100%) Good 12: 0.48 (95%) Good 24: 0.22 (95%)	Poor t12: 0.58 (100%) Good t12: 0.58 (97%)	Poor t12: 0.66 (100%) Poor t24: 0.60 (100%) Good t12: 0.72 (95%) Good t24: 0.20 (95%)	Poor t12: 0.56 (100%) Poor t24: 0.65 (94%) Good t12: 0.63 (94%) Good t24: 0.58 (93%)	Unknown	Poor t12: 0.13 (100%) Poor t24: 0.55 (100%) Good t12: 0 (100%) Good t24: 0.25 (100%)

Table A.I: Comparison table with all studied recently published (< 5 years) EEG-based post anoxic coma prognostication models and the original CRI [19] (> 5 years). CA=cardiac arrest. CNN=convolutional neural network. CRI=cerebral recovery index. (P)CPC=(pediatric) cerebral performance category. CV=cross validation. GBTM=group-based trajectory modelling. Grad-CAM=gradient-weighted class activation mapping. ICU=intensive care unit. LR=logistic regression. qEEG=quantitative electroencephalography. RFC=random forest classifier. TTM=targeted temperature management. SVM=support vector machine.

	[36]	*[52]*	*[54]*	*[6]*	*[29]*	*[32]*
Time of EEG	Start < 24h Continuous Until 48 h	Start a.s.a.p Hourly up to 72 h	Start a.s.a.p Hourly up to end	20.2 h (+-6.1 h)	Start a.s.a.p Hourly up to end	During TTM
EEG fragment length	> 6h	5 min Artefact free selected per hour	5 min Artefact free selected	5 min Artefact free selected	5 min Artefact free selected	5 min Artefact free selected
Sample size	1010	438	69 (children)	267	559	94
Input	Hourly epochs	5 min hourly epochs divided in time intervals every 12 h (e.g. 0-12, 13-24, 25-36)	1.Early set (0-17h) with 5 min/hour 2. Late set (18-end) with 5 min/hour	5 min epoch divided into 10 sec epochs with 75% overlap	5 min epoch	5 min epoch divided into 30 non-overlapping 10 sec fragments
Input type	1 qEEG feature per hour and clinical features	52 qEEG and 4 clinical features (each interval also uses previous features)	- 8 qEEG features from each epoch averaged across interval - 2 clinical features	1D image	2 qEEG features per epoch	8 qEEG features from each fragment averaged across epoch
Model	3 classes: 1. GBTM 2. Non-para -metric k-means 3. Bayesian regression	LR Elastic net trained at each interval	1. LR 2. SVM 3. RFC	1D CNN Grad-CAM to visualise features	Prediction model	Bayes classifier
Output	Poor outcome Good=CPC 1-3 Poor=CPC 2-5	Probability of good outcome Good=CPC 1-2 Poor=CPC 3-5	Predicting good / poor outcome Good= PCPC 1-3 Poor= PCPC 4-6	Probability of poor outcome mean of each 10 sec fragment	Probability of good outcome Good=CPC 1-2 Poor=CPC 3-5	Good vs. poor outcome
Time of outcome	Hospital discharge	6 months	Hospital discharge	3 months	6 months	3 months
Train/Val	Leave one out CV	- 90% train - 10% val 10 fold CV	5-fold CV	- 80% train/val (80% train 20% val) - 20% test	10 fold CV	- 2/3 train/val (leave one out CV) -1/3 test
AUC	Unkown	Good t12: 0.71 Good t72: 0.73	RFC early: 0.88 RFC late: 0.74	0.885	t12: 0.86 t24: 0.87	Poor: 0.81
Se (at Sp)	0.38 (>99%) (GBTM)	Good t12: 0.43 (95%) Good t72: 0.61 (95%)	RFC early: 0.84 (75%) RFC late: 0.76 (62%)	0.78 (89%)	Poor t12: 0.50 (100%) Poor t24: 0.42 (100%) Good t12: 0.52 (90%) Good t24: 0.57 (90%)	Poor: 0.54 (100%)

Table A.II: Characteristics of EEG-based LSTMs. Continued on next page. BCE=binary cross-entropy. BLSTM=bidirectional long short-term memory. CCE=categorical cross-entropy. CE=cross-entropy. CNN=convolutional neural network. CV=cross validation. EEG=electroencephalogram. Elu=exponential linear. FC=fully connected. HP=hyperparameter. ICA=independent component analysis. LSTM=long short-term memory. MSE=mean squared error. NN=neural network (feedforward). Norm=normalisation. Opt=optimiser. PC=principle components. Relu=rectified linear.

	[37]	*[116]*	*[39]*	*[86]*	*[45]*	*[38]*
Model	LSTM	LSTM	LSTM Compared to: Decision tree, Ripper, SVM	LSTM Compared to other papers:	Cascaded LSTM Compared to other papers: (SVM, RFC, CNN)	1. LSTM 2. BLSTM 3. Deep BLSTM -LSTM
Goal	Predict arousal state: - binary classification - continuous prediction	1. classify auditory or visual stimuli	Binary classification between preictal and interictal seizures	Classify high/ low states of arousal, valence, liking	classification of sleep stages: - 4 class model - 2 class model	- Age prediction - Gender prediction
Input	X: 1. 4 variants of EEG decomposition by SSD 2. SDD variant + SPoC Y: High and low arousal labels from questionaires	X: Components from ICA Y: Stimuli time and latency labels	X: 1. Extracted features from EEG 2. Raw EEG Y: Preictal or ictal label	X: Segmented raw EEG from music videos Y: High or low state label	X: 4 class: 11 selected features from feature extraction 2 class: 27 PC from 11 features Y: Sleep stage	X: 1. Alpha 2. Beta 3. Gamma 4. Delta 5. Theta 6. Raw EEG Y:Age, gender
Output	Binary: - High arousal - Low arousal	Binary: - Visual stimulus - Auditory stimulus	Binary: - Preictal - Interictal	Binary high/ low for: - Arousal - Valence - Liking	4 class: W, N1/REM, N2, N3 2 class: N1, REM	- Age (6 age) - Gender (M/F)
Tuning HP	Tuning of: - LSTM layers - LSTM hidden units - FC size - Activation - Adam learning rate - Weight regulation	No tuning discussed; choices not explained	3 architectures evaluated, variations in: - LSTM layers - Hidden units - Dropout rates	No tuning discussed; choices not explained	1000 evaluated. No tuning discussed; choices not explained. Variations in: - LSTM layers - Hidden units	Various models made, best models are discussed. No tuning discussed; choices not explained
Architecture and HPs	LSTM (10:100, relu/elu) Optional: LSTM (10:100, relu/elu) Optional: FC (tanh) FC (tanh) Loss=MSE	LSTM (100) Dropout (0.5) Dense (1, sigmoid) Loss=BCE Opt=RMSprop Batch size=16 Epochs=50	Best performing LSTM: LSTM (128) LSTM (128) FC (30, relu) Dense (2, softmax) Loss=CE Opt=Adam Batch size=10	LSTM (64, relu) Dropout (0.2) LSTM (32, sigmoid) Dense (1, sigmoid) Opt=RMSprop Epochs=30	Best 4 class: LSTM (101) FC (4) Activation (softmax) Best 2 class: LSTM (125) LSTM (98) FC (2) Activation (softmax) Loss=CE Opt=Adam	LSTM model: LSTM (128) Batch norm. LSTM (64) Batch norm. FC (32) FC (age:6 /gender:2) Age: loss=CCE Gender: loss=BCE
Train /Val	10-fold CV	25% training set 75% testing set	10-fold CV	4-fold CV	10-fold CV: - 80% training - 10% val - 10% test	10-fold CV: - 60% training - 30% test - 10% val

Table A.II: Characteristics of recently published (<3 years) EEG-based LSTMs. CSP=common special pattern. DBN=deep belief network. FFTEM=fast Fourier transform energy map. GAFRN=Gramian angular field using residual network. GAFTCNN=Gramian angular field using tiled convolutional neural network. K-NN=K-nearest neighbors. LR=logistic regression. MCDBN=multi-channel deep belief network. RNN=recurrent neural network. SAE=stacked auto-encoder. SGD=stochastic gradient descent. SPoC=Source Power Comodulation. SSD=Spatio Spectral Decomposition.

	[46]	[40]	[44]	[47]	[49]	[48]
Model	LSTM + attention Compared to: PLV, ANN, SVM, LR, decision tree, RFC, naïve bayes, CNN, LSTM	LSTM Compared to: CSP, MCDBN, FFTEM, GAFRN, GAFTCNN	LSTM Compared to: SVM	LSTM Compared to: improved NN, NN, LR, SVM	SAE+LSTM Compared to: SVM, LSTM, ICA-LSTM	BLSTM Compared to: RNN-LSTM, DBN, CNN, K-NN, SVM
Goal	Binary classification of hand movement: - left hand - right hand Cross-subject Intra-subject	Classify motor imagery tasks: 1. Left/right hand 2. Left hand/ both feet 3. Left hand/tongue 4. Right hand/ both feet 5. Right hand/tongue 6. Both feet/tongue	Classify seizure EEG: - ictal - preictal - interictal or Binary classification between 2 classes	Compare different models and parameters in EEG classification	Binary classification of EEG into high/low states of arousal and valence	Classify confusion
Input	X: Features from EEG signal Y: Left hand or right hand label	X: Channel weighted 1d-AX extracted matrices from EEG Y: Movement label	X: 1. Processed features 2. Features 3. Raw data Y: ictal label	X: 4 extracted features per channel from 20 channels	X: Feature sequence Y: High or low state label	X: Feature vector Y: Confusion label
Output	Binary: - Left hand movement - Right hand movement	Binary task dependent	Classification: - Preictal - Ictal - Interictal		Binary high/ low for: - Arousal - Valence	Binary: - Confused - Not confused
Tuning HP	Tuning of: - Recurrent depth - Batch size - Epochs - LSTM hidden units - Dropout rates - Weight regulation	Tuning of: - LSTM layers - LSTM hidden units	2 architectures evaluated, tuning of: - LSTM layers - Hidden units - Learning rate	Evaluation of different parameters: - Activation functions - Optimisers - Loss functions	No tuning discussed; choices not explained	No tuning discussed; choices not explained
Architecture and HPs	Cross/intra-subject: Dropout (0/0.7) LSTM (256) Dropout (0.2/0.2) LSTM (256) Dropout (0.1/0.1) LSTM (256) Dropout (0.2/0.1) Attention layer FC (sigmoid) Loss=BCE Opt=Adam Batch size=32/2 Epochs=100/10	LSTM (12) Dropout (0.6) Activation (softmax) Opt=Adam Batch size=64	Best performing model: LSTM (100) Dropout LSTM (100) Dropout Opt=Adam Batch size=150	LSTM (2 layers) Best HPs are different for each metric	LSTM (125) Dropout FC (125) Activation (sigmoid) Opt=Mini-batch SGD Loss=MSE	BLSTM (50) Activation (tanh) FC layer Activation (sigmoid) Batch size=20
Train /Val	10-fold CV	Hybrid data: 5x5 fold CV Subject specific: 1 training session 1 val session	- 80% training - 20% test		10-fold CV	5-fold CV

E. Cardiac arrest: cause and treatment

Cardiac arrest is defined as "the sudden cessation of cardiac activity so that the victim becomes unresponsive, with no normal breathing and no signs of circulation. If corrective measures are not taken rapidly, this condition progresses to sudden death. Cardiac arrest should be used to signify an event as described above that is reversed, usually by cardiopulmonary resuscitation (CPR) and/or defibrillation or cardioversion, or cardiac pacing. Sudden cardiac death should not be used to describe events that are not fatal" [117]. An underlying structural cardiac impairment, most frequently ischemic coronary disease, often causes the CA. However, CA also originates from non-cardiac causes. The aetiology differs per age and population. Generally, the arrests are sudden and unexpected, often resulting in a fatal outcome. Proper identification of CA is critical before initiating treatment. After correctly diagnosing CA, the first stage is basic life-supporting treatment, including CPR and automated external defibrillation. CPR should be executed until emergency responses are present. Hereafter, advanced life-supporting procedures are initiated, including medication. With these measures, the return of spontaneous circulation (ROSC) can be achieved. Hereafter, post-resuscitation care is initiated [118].

CA causes global ischemia in the brain. Therefore, oxygen and glucose decrease, which causes a rapid depletion of ATP. Consequently, a biochemical ischemic cascade is initiated, causing severe cerebral damage and neuronal death. The patient loses consciousness within seconds. After ROSC, the primary sign of cerebral damage is PAC. PAC occurs in 80% of CA victims occurring outside of the hospital [2, 4].

For most comatose patients, post-resuscitation care includes targeted temperature management (TTM) as part of the standard protocol upon arrival in the emergency care unit. TTM, previously known as therapeutic hypothermia, lowers the body temperature of a patient to a value between 33 (hypothermia) and 36 (normothermia) degrees Celsius [119, 120, 121]. TTM is neuroprotective. As patients are likely to continue to suffer from cerebral ischemia for hours after CA, cooling them reduces the brain's oxygen demand. Consequently, the cerebral damage decreases. TTM resulted in improved neurological outcomes and decreased in-hospital mortality [1, 4, 7, 119].

F. Outcome assessment

Generally, physicians use the CPC to classify neurological outcome (Table A.III). The CPC is derived from the Glasgow Outcome Scale, which classifies coma due to traumatic head injury. The CPC values link to the Glasgow Outcome Scale values in reversed order. A CPC score of 1 or 2 indicates no or mild neurological damage, respectively. A CPC score of 3, 4 or 5 means severe neurological damage, vegetative state, or (brain)death, respectively [4, 5]. For this study, the CPC at six months after cardiac arrest, scored by a researcher, was used as the primary outcome measure.

G. Examples of EEG recordings from patients with different outcomes

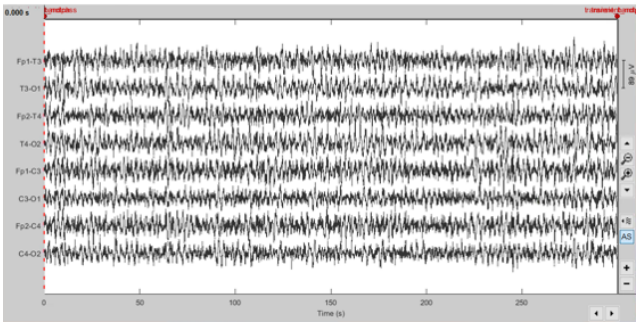


Figure A.3: Five-minute EEG recorded from a comatose patient with a good neurological outcome (CPC score of 1 at 6 months after cardiac arrest). Scale between channels = 89μV. CPC=cerebral performance category.

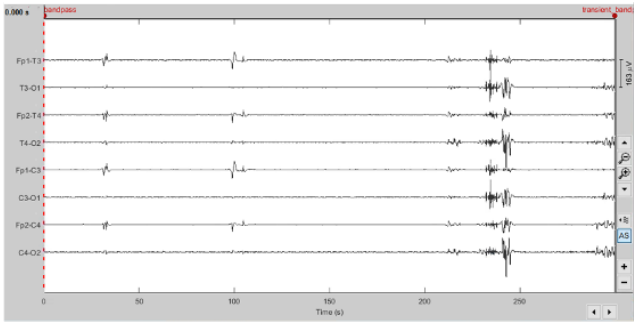


Figure A.4: Five-minute EEG recorded from a comatose patient with a poor neurological outcome (CPC score of 5 at 6 months after cardiac arrest). Scale between channels = 389μV. CPC=cerebral performance category.

H. Preprocessing

Preprocessing the raw EEG data compromised the following subsequent steps (Fig. A.5). First of all, to reduce the signal's noise, the EEG recordings were re-referenced to another montage. For better comparability to other studies, re-referencing should be done similarly as existing outcome prediction models [122]. From the studies that included the re-referencing method in their article, the majority re-referenced to a longitudinal bipolar montage [18, 20, 21, 29]. One used an average montage [52]. The average re-reference requires 64 electrodes at the minimum to limit the bias of the unevenly distributed electrodes on the scalp [73]. As the recordings in this study were made with nine electrodes,

Table A.III: Cerebral Performance Categories (CPC) and Glasgow Outcome Scales (GOS). Adapted from "Prognostication after cardiac arrest", by Sandroni, C., D'Arrigo, S., Nolan, J., 2018, Critical Care, 22(150), p.2, © 2018 by the Author(s) [5]

CPC	GOS	Disability	Conscious	Independent	Features
1	5	No, minor	Yes	Yes	Able to work and lead a normal life. May have mild dysphasia, non-incapacitating hemiparesis, or minor cranial nerve abnormalities
2	4	Moderate	Yes	Yes	Able to travel by public transport and work in sheltered environment. Independent in activities of daily life. May have hemiplegia, seizures, ataxia, dysarthria, or memory changes
3	3	Severe	Yes	No	Limited cognition, dementia, locked-in, minimally conscious. Usually in institution, but it may be looked after at home with exceptional family effort
4	2	Unconscious	No	No	Persistent vegetative state
5	1	Dead	-	-	Certified brain dead or dead by traditional criteria

I chose to re-reference to a longitudinal bipolar montage: [Fp1-T3, T3-O1, Fp2-T4, T4-O2, Fp1-C3, C3-O1, Fp2-C4, C4-O2]. Subsequently, to reduce the influence of frequencies outside the brain's power spectrum, the signal's bandwidth was limited. The lowest frequency of interest, the delta frequency, has a range of 0.5–4 Hz. The highest relevant frequency, the beta frequency, ranges from 16–30 Hz [73, 98]. I used a bandpass filter of 0.5–30 Hz to keep the signal within a significant range. Furthermore, a notch filter of 50 Hz removed any powerline artefacts. Finally, to decrease computational time, the recordings were downsampled to 128 Hz.

I. Feature extraction

Step-by-step process

The feature extraction and input preparation process compromised several steps. The five-minute preprocessed epochs were segmented into 30 non-overlapping 10-second time-fragments. Per time-fragment, I extracted 19 qEEG features from all channels. Following feature extraction, the feature matrix in R^{fxc} was obtained, where f denotes the number of features and c the number of channels. In the next step, the feature matrix was averaged across all channels, resulting in one feature vector in R^{fx1} for each time-fragment. Subsequently, the features vectors of the 30 time-fragments in an epoch were concatenated, resulting in one R^{fxt} feature matrix per epoch, where t denotes the time-fragments. Hereafter, the process was divided into two separate paths. One path was dedicated to the input preparation for the LR, the other for the LSTM, as these require different input forms. For the LR, I averaged the features across all time-fragments per epoch. Consequently, a feature matrix R^{fxe} was obtained, where e denotes the number of epochs. This matrix included one value per feature per epoch. For the LSTM input, the feature matrices R^{fxt} of all epochs were concatenated, creating a feature matrix R^{fxte} where te denotes the time-fragments of all epochs. This matrix included one value per feature per time-fragment and contained all the epochs. Hereafter, the features were normalised. Both matrices were scaled between 0 and 1 with respect to the features of all epochs. With various techniques, I selected specific features sets as input to the models. The input to the LR was a matrix of the normalised features by epochs. The input to the LSTM was reshaped to a three-dimensional tensor of the normalised features by time-fragments by epochs.

Motivation of choices

I extracted qEEG features from the EEG signal to use as input to the LSTM instead of the raw signal, which was used by the previously reported outcome prediction neural networks [20, 21, 52]. Extracting features provides the model with important hidden information within the signal and reduces the dimensionality [39]. Furthermore, not only the majority of EEG-based LSTMs were fed with features [50], but various studies using EEG-based LSTMs showed that features outperformed the pure signal as input [38, 39, 44].

To reduce noise, I extracted the features per time-fragment for the LR and averaged them for the five-minute epoch, instead of extracting the features per five-minute epoch.

The reason for averaging the features across all channels was twofold. Most importantly, averaging decreased the input dimension. Presumably, no information was lost by averaging as postanoxic coma affects the whole brain, not specific localised regions [34]. Secondly, I increased the comparability to previously reported outcome prediction models, as these followed this approach [18, 20, 52, 54].

Normalisation was applied, as the features were very diverse in magnitude. Normalisation speeds up learning and convergence. Also, it avoids creating a bias towards any specific data class during training [44, 123]. Normalising the features (rescaling between 0 and 1) had the preference over standardising (rescaling the distribution to 0 mean and a standard deviation of 1), as the

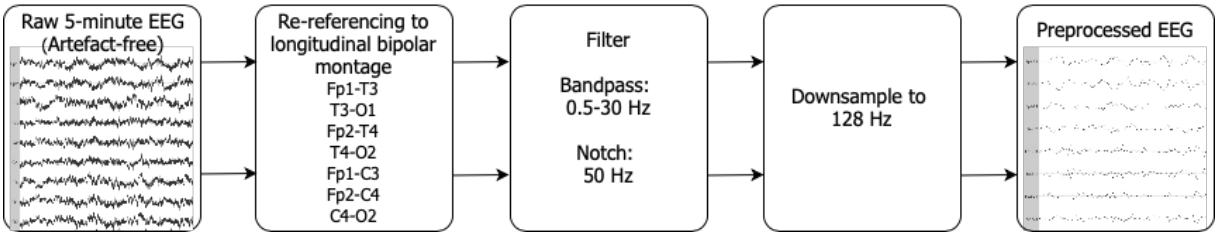


Figure A.5: The preprocessing pipeline. Firstly, the nine-channel EEG epochs were re-referenced to a longitudinal bipolar montage to reduce the signal's noise. Subsequently, a bandpass filter of 0.5-30 Hz was applied, reducing the influence of frequencies outside the brain's main power spectrum. Additionally, a notch filter of 50 Hz was applied to remove any powerline artefacts. Finally, the signals were downsampled to 128 Hz to reduce computational time. EEG=electroencephalography.

features did not fit a gaussian distribution. Therefore, standardisation could result in unreliable results [123]. The features were scaled with respect to the features of all patients' epochs, not within a single patient's epochs.

The 19 qEEG features were specifically selected based on their observed prognostic significance in other outcome prediction studies [18, 35, 52, 54]. Using features with previously proven predictive power increased the chance the models were fed with features that carry significant prognostic value. Furthermore, the input dimension space was efficiently used, as the other features previously used in outcome prediction studies, not explicitly showing predictive power, were excluded.

J. Feature descriptions and calculations

A description of the extracted features from the EEG signal is discussed per domain in this appendix. The features were specifically selected based on their observed prognostic significance in previous outcome prediction studies. Tjepkema-Cloostermans et al. showed the individual feature contributions to outcome prediction from (amongst others) 12 and 24 hours after CA in their study [35]. In the work of Nagaraj et al., the least absolute shrinkage and selection operator (LASSO) identified a set of features with high predictive power across all hours [18]. Ghassemi et al. showed which features significantly correlated with outcome and features the elastic net (LASSO) selected for prediction. They gave this information for all time intervals, including 0 to 12 hours and 13 to 24 hours [52]. Finally, Lee et al. studied the importance of individual features to outcome prediction using Gini importance and discussed those for the time interval 0 to 17 hours after CA [54]. I combined the highly predictive features from these different studies to one feature set and used features from this set as input to this study's models. All features were calculated per 10-second fragment, except for the false nearest neighbour feature, which I calculated for the entire five-minute EEG fragment.

Complexity features

The brain produces highly complex and stochastic signals, characterised by randomness and irregularity [33]. Presumably, this complexity correlates positively with physiological brain functionality [19]. A decrease in complexity might indicate brain injury and impairment of physiological functioning [124, 125]. The features in this domain quantified the extent to which the EEG signal is random or irregular.

Shannon entropy Shannon entropy is a measure to quantify the complexity of a stochastic signal. Shannon and Weaver defined it with the following formula [61].

$$SE = -\sum_{i=1}^{N} p\left(x_i\right) \log_2 p\left(x_i\right) \tag{A.1}$$

In this formula, x_i is the signal's amplitude, and p is the probability that this amplitude is present in that fragment of the signal. I calculated Shannon entropy similarly to Tjepkema-Cloostermans et al.: "The probability density function $p(x_i)$ was estimated by using the histogram method in which the amplitude range of the signal was linearly divided into bins" (Tjepkema-Cloostermans et al., 2013). The minimal and maximal bins were set to -200 μV and 200 μV, respectively. The bin width was set to 2 μV. Shannon entropy showed prognostic importance in multiple outcome prediction studies [18, 19, 52].

Tsallis entropy Tsallis entropy is a statistic to quantify a stochastic signal's complexity in a nonextensive manner [63]. Opposed to Shannon entropy, Tsallis entropy does not assume extensive or entropic additivity. This assumption might result in an overestimation of the signal's complexity due to the inherent properties of the EEG [62, 63, 126, 127]. Therefore, Tsallis entropy might be a more reliable indicator of the signal's complexity [33, 64]. Tsallis entropy is defined as

$$TE = \frac{1 - \sum_{i=1}^{M} p_i^q}{q - 1} \tag{A.2}$$

In this equation, p_i^q is the probability of the presence of the amplitude, and q refers to the nonextensivity degree, termed the entropic index. The index is typically chosen empirically, and no method for its determination exists [62]. The entropic index was set to 2, as Tsallis entropy with a q of 2 had significant prognostic power in EEGs recorded within the first 24 hours [52]. Tsallis entropy proved to have value for prognostication in studies with animals suffering from PAC. Ghassemi et al. used it for the first time in human outcome prediction [52].

Cepstrum coefficients Cepstrum coefficients provide information regarding the rate of change of the different frequency bands in the spectrum. It is defined as "the inverse of the Fourier transform of the log-magnitude of the spectrum of the signal" [33, 65]. The cepstrum coefficients offer a different way to quantify the complexity of a system than entropy. Cepstrum was used in animal studies to evaluate postanoxic brain injury, where its prognostic value outperformed other spectral features [64]. Moreover, it showed a significant contribution to outcome prediction within the first 24 hours of EEG recording in humans [52]. The calculation of the cepstrum coefficients is as follows [64, 65]:

1. Calculate the autoregressive (AR) coefficients and find the denominator of the spectrum of the EEG signal:
$$H\left(e^{j\omega}\right) = \frac{1}{|A\left(e^{j\omega}\right)|} \tag{A.3}$$
2. Calculate the log-magnitude of the spectrum:
$$C\left(e^{j\omega}\right) = \log\left(H\left(e^{j\omega}\right)\right) \tag{A.4}$$
3. Take the inverse fast Fourier transform of the logarithm of the spectrum.

Hjorth mobility and complexity Hjorth mobility and complexity are two categories of the Hjorth parameters. They provide information about the spectral frequencies in the EEG signal and quantify the statistical property of the EEG in the time domain [66, 67]. Hjorth mobility is defined as: "the square root of the ratio of the variance of the first derivative of the signal and that of the signal" [67]. The Hjorth complexity quantifies how much similarity the signal's shape has with that of a pure sine wave. With increasing similarity, it approaches a value of 1 [67].

$$Mobility = \sqrt{\frac{var\left(y'\left(t\right)\right)}{var\left(y\left(t\right)\right)}} \tag{A.5}$$

$$Complexity = \frac{mobility\left(y'\left(t\right)\right)}{mobility\left(y\left(t\right)\right)} \tag{A.6}$$

Here $y\left(t\right)$ is the signal and $y'\left(t\right)$ its first derivative. Hjorth mobility correlated with outcome in the first 24 hours after CA. The elastic net selected Hjorth complexity for outcome prediction in the first 24 hours after CA [52].

False nearest neighbour The false nearest neighbour algorithm is an algorithm to determine the embedding dimension [69]. It is another measure to indicate the signal's complexity by quantifying its constancy and smoothness [52]. I used an improved version of the algorithm by Hegger and Kantz, which can discriminate between deterministic and stochastic processes, indicating the signal's continuity [68]. The false nearest neighbour correlated notably with the neurological outcome and was selected by the elastic net for feature prediction [52], indicating it carried significant predictive value.

Autoregressive model coefficients An AR model can represent a random process. Therefore, one can use it to investigate the EEG signal's temporal characteristics [64]. Ghassemi et al. showed that the AR coefficients in their study had significant predictive value. Specifically, these were the non-seasonal AR term coefficients at t-1 and t-2 of a second-order AR model, estimated by maximum likelihood given the EEG signal [52]. The AR model with order p is defined as

$$AR = x\left[k\right]\sum_{i=1}^{p} a_i x\left[k-i\right] + e\left[k\right] \tag{A.7}$$

Here a_i is the estimated coefficient, and $e[k]$ the white noise in the time-series $x[k]$ [64, 70]. The MATLAB Econometrics toolbox is used to calculate the AR coefficients [128].

Category features

As discussed in Appendix B.2, six categories of patterns can be distinguished in postanoxic patients [17]:

1. Iso-electric EEG
2. Low voltage EEG ($< 20\ \mu$V)
3. Burst-suppression EEG and the subcategory of burst-suppressions with identical bursts
4. Epileptiform EEG, including status epilepticus and generalised periodic discharges
5. EEG with continuous activity less than 8 Hz (diffused slow EEG)
6. EEG with continuous activity equal to or greater than 8 Hz ("normal" EEG)

Frequency band powers were used as features because specific postanoxic EEG patterns, like the continuous EEG activity patterns, are characterised by specific frequencies. I calculated the power in a frequency band

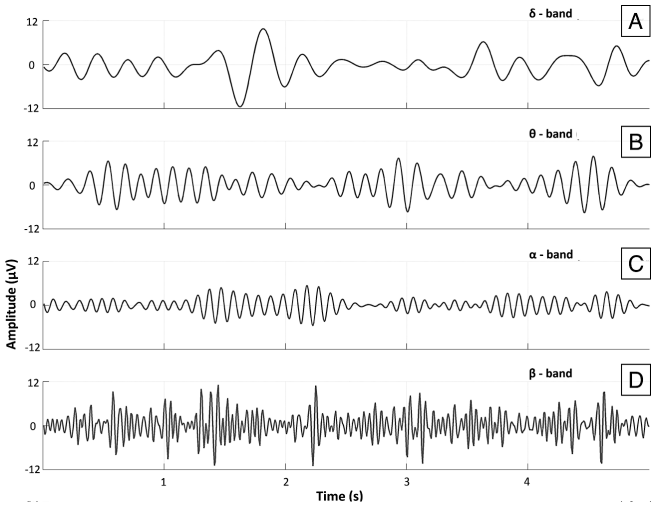


Figure A.6: Examples of EEG waves in the frequency domains that were used as features. A) delta frequency, B) theta frequency, C) alpha frequency, D) beta frequency. Adapted from "Towards the bio-personalisation of music recommendation systems: a single-sensor EEG biomarker of subjective music preference" by Adamos D, Dimitriadis S, Laskaris N. [130]

with MATLAB's band power function, which uses a modified periodogram to determine the average power in the frequency range [129]. The frequency bands' power was normalised to the total signal power in the range of interest (0.5-30 Hz), as this showed discriminative power between good and poor outcomes [18, 54].

Normalised delta power The delta band compromises frequencies between 0.5–4 Hz (Fig. A.6) [71, 72, 73]. In the study of Lee et al., delta frequency normalised to total power was one of the most important features for prediction in the time interval of 0-17h. A higher value was associated with good neurological outcomes [54].

Normalised theta power The theta band compromises frequencies between 4–7 Hz (Fig. A.6) [71, 72, 73]. Theta power/total power belonged to the top 10 features selected by LASSO in the model of Nagaraj et al. [18].

Normalised alpha power The alpha band compromises frequencies between 8–13 Hz (Fig. A.6) [71, 72, 73]. As with normalised theta power, LASSO selected normalised alpha power for outcome prediction [18].

Normalised beta power The beta band compromises frequencies between 14–30 Hz (Fig. A.6) [71, 72, 73]. Normalised beta power was important for prediction in the study of Lee et al. [54]. Moreover, as with the normalised theta and alpha power, beta power was selected by LASSO the study of in Nagaraj et al. [18], where it showed significant discriminative power between good and poor outcome. Higher values of beta power were associated with good outcome [18].

Signal power The signal power was defined as the total power in the frequency range of interest (0.5-30 Hz). Signal power showed to be one of the most important features for outcome prediction in the study of Tjepkema-Cloostermans et al. [35].

Regularity Tjepkema-Cloostermans et al. developed the regularity feature to discriminate between postanoxic EEG patterns with burst-suppressions and continuous activity. The latter is assumed to have a regular and constant amplitude [19]. The measure indicates the continuity of the signal. Regularity might give important information regarding the outcome, as burst suppressions are associated with poor outcome, whereas continuous activity indicates good outcome. The calculation of regularity included several steps. A moving average filter with a window of 0.5 seconds was applied to the squared EEG signal. The length of this window was chosen to average out the separate peaks in a single burst optimally (at least 500 milliseconds in duration [17]), without averaging out activity between successive suppressions and bursts [19]. The moving average filter gave a non-negative and smoothed signal. Hereafter, the values of this smoothed version were sorted in descending order. Subsequently, this signal's normalised standard deviation was calculated, obtaining the regularity feature of the EEG signal [19, 35]. The formula for regularity is

$$REG = \sqrt{\frac{\sum_{i=1}^{N} i^2 q\,(i)}{\frac{1}{3}N^2 \sum_{i=1}^{N} q\,(i)}} \tag{A.8}$$

N is the signal's length in samples and q the smoothed and sorted signal [19]. The value of the regularity lies between 0 and 1, where 1 indicates a constant amplitude. The regularity will approach zero if the number of bursts is small or their duration is short and long suppressions are present. If relatively long bursts or more bursts are present, the value increases [18, 19, 35]. Regularity showed a significant prognostic value in the study of Tjepekema-Cloostermans et al. and Ghassemi et al. [52, 19]. Higher values of regularity correlated positively with a good neurological outcome [52].

Number of epileptic spikes Epileptic spikes in the EEG are high-speed, high amplitude waves of 70 milliseconds or shorter [131]. They occur during the epileptiform EEG pattern in PAC. Ghassemi et al. found that the number of epileptic spikes present in the EEG had a significant positive correlation with poor outcome. Moreover, the number of spikes was included by the elastic net for outcome prediction within the first 24 hours

[52]. Therefore, I adopted the calculation of the number of epileptic spikes from the study of Ghassemi et al. Waves were counted as spikes when their amplitude reached a value of three standard deviations from the mean signal value and when their duration was equal to or shorter than 70 milliseconds [52].

Burst suppression ratio Burst suppression ratio/minute (BSR) was the most often selected feature by LASSO in the study of Nagaraj et al., indicating high predictive power for predicting the neurological outcome [18]. BSR is defined as:

$$BSR = \frac{duration\ the\ EEG\ is\ lower\ than\ 5\ \mu V}{total\ duration\ EEG} \tag{A.9}$$

The BSR is almost equal to the feature termed "low voltage EEG" in the work of Ghassemi et al., which divides the duration the EEG is lower than 5 μV by the total duration of the EEG. Low voltage showed a significant positive correlation with poor outcome in the first 24 hours [52]. The features BSR and low voltage EEG are related to the postanoxic low voltage EEG pattern, which is associated with poor neurological outcome.

Connectivity features

The brain's neural network architecture can be seen as a system with many communicating subsystems or brain areas [74]. The complex interactions between these different areas reflect healthy brain activity [132]. Features in this domain quantified the relationships between specific channels, which indicated the interaction between the different brain regions. The connectivity features were extracted between two channels and subsequently averaged across all channels.

Delta coherence I calculated the coherence in the delta frequency band (0.5–4 Hz) between all channel combinations with this feature. The coherence quantified the degree of similarity in the delta band. When a higher synchronisation level occurs, the brain activity is less random and diverse, indicating less healthy brain functioning [19, 35, 52]. For the calculation, a Hanning window of 4 seconds with 2 seconds overlap was used [19]. Delta coherence was an important feature in the studies of Tjepkema-Cloostermans et al. and Ghassemi et al. [35, 52].

Phase lag index Ghassemi et al. found that the phase lag index (PLI) had a significant positive correlation with poor outcome during the first 24 hours and was selected by LASSO during both the interval of 0-12 and 13-24 hours [52]. The PLI quantifies interactions between time series. Specifically, it is defined as "a measure of the asymmetry of the distribution of phase differences between two signals" [74]. PLI was calculated with the following formula.

$$PLI = |\langle sign\left[\Delta\Phi\left(t\right)\right]\rangle| \tag{A.10}$$
$$\Delta\Phi\left(t_k\right), k = 1 \ldots N$$

Here $\Delta\phi(t_k)$ is the time series of phase differences, and t_k are discrete time-steps and N the number of samples. The index ranges between 0 and 1. When asymmetry between two signals occurs, a constant non-zero phase difference between these two signals is present, called the lag. With stronger non-zero phase locking, the PLI approaches 1. If the phase-locking is perfect, with $\Delta\phi$ different from 0 mod π, the index is 1. When the signals are coupled with a $\Delta\phi$ centred around 0 mod π, or the signals are not coupled to each other, the PLI is 0 [74].

The motivation for using the PLI as feature originated from a study that found that poor outcome is associated with decreased EEG connectivity. The neural network architecture in patients with poor outcome was observed to be smaller and less connected. If neurons' activity is inadequately distributed and differentiated, impaired cerebral functioning might result [132]. The PLI indicates the degree of asymmetry, which reflects the connectivity. A good neurological outcome is correlated to a high level of connectivity, which is associated with a low level of asymmetry, thus a small PLI.

K. Feature input selection

The final feature set used as input was created by excluding features with high multicollinearity from the 19 extracted qEEG features.

Calculation of high multicollinearity

Multicollinearity was calculated with the variance inflation factor (VIF). The VIF indicates if features are correlated and contain similar information about the variance within the dataset. The VIF is calculated as follows.

$$VIF = \frac{1}{1 - R_i^2} \tag{A.11}$$

R^2 is the coefficient of determination for the regression of one feature on the other features. The R^2 is calculated for each feature i, resulting in R_i^2. The VIF indicates how much variance of one specific feature is inflated as a result of multicollinearity. Although multicollinearity does not influence the predictive power of a model, identifying its presence in a dataset is essential. If multiple features are highly correlated, the model will not be able to determine the relationship between one of those features and the outcome. Consequently, the statistical significance of a feature decreases. Therefore, the influence of one specific feature on the outcome might be incorrectly estimated. When new data is presented to the model, large errors may occur due to this incorrect feature-outcome relationship [56, 57]. I computed the VIF with the code of Vasilaky [133]. Vasilaky

eliminated the requirement to calculate multiple regressions by computing the VIF by retrieving the diagonal elements of the inversed correlation matrix [133, 134]. The correlation coefficients were calculated for the feature matrix with the averaged features across per epoch (matrix R^{fxe}, Appendix I). A feature's multicollinearity is often labelled high when the VIF is higher than 10 [135]. Therefore, features with a VIF higher than 10 were excluded to create the final feature set used as input. The exclusion process was as follows:

1. The VIFs of the features were calculated.
2. The feature with the highest VIF was removed.
3. Step one and two were repeated until the highest VIF had a value below 10.

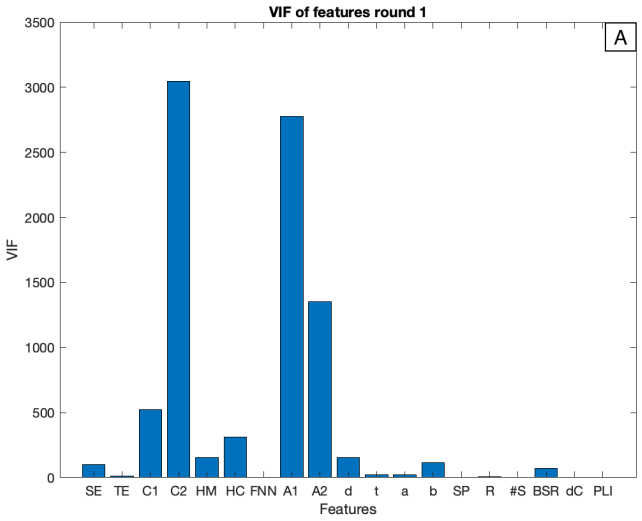
The VIFs were recalculated after each feature removal as the multicollinearity changed after every feature removal.

Results of the feature input selection

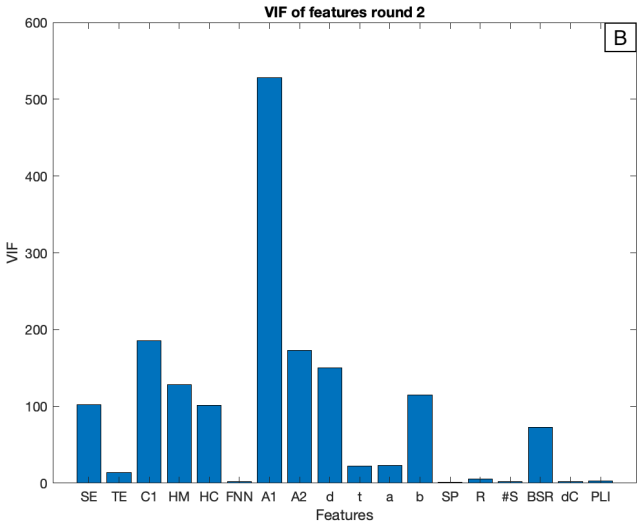
Figure A.7 displays the eight rounds of high VIF feature exclusion to create the final feature set. For each round, a subplot of the VIF scores is shown. The highest scoring feature was removed, and the VIFs were recalculated until the plot showed all VIF values below 10. Table A.IV states the final features included in the set.

Table A.IV: Final feature set used as input to the LR and LSTM model. This was the resulting set of features after the features with high multicollinearity were excluded from the 19 qEEG features. AR=autoregressive. LR=logistic regression. LSTM=long short-term memory recurrent neural network.

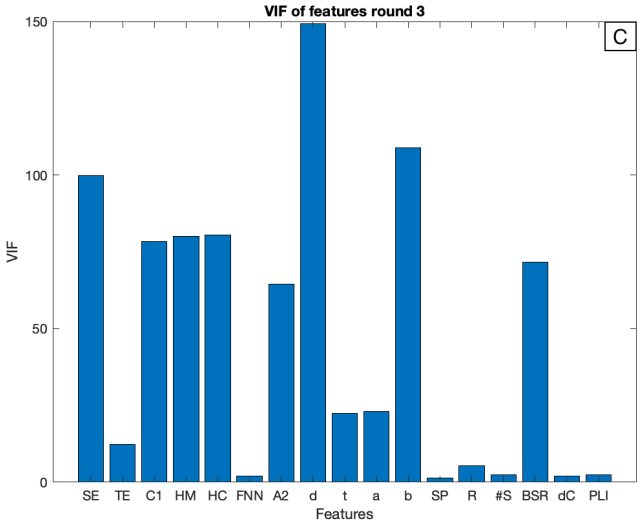
	Final feature set
1.	Tsallis entropy
2.	False nearest neighbours
3.	AR coefficient 2
4.	Normalised theta power
5.	Normalised alpha power
6.	Normalised beta power
7.	Signal power
8.	Regularity
9.	Number of epileptic spikes
10.	Burst suppression ratio
11.	Delta coherence
12.	Phase lag index



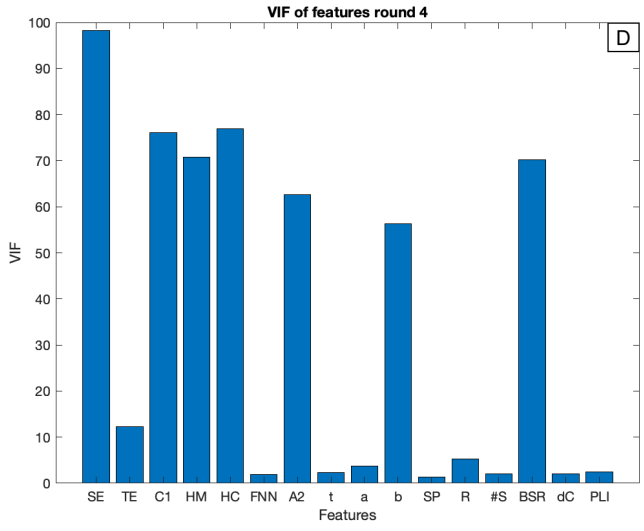
(a) Round 1: Cepstrum coeff. 2 had the highest VIF and was removed.



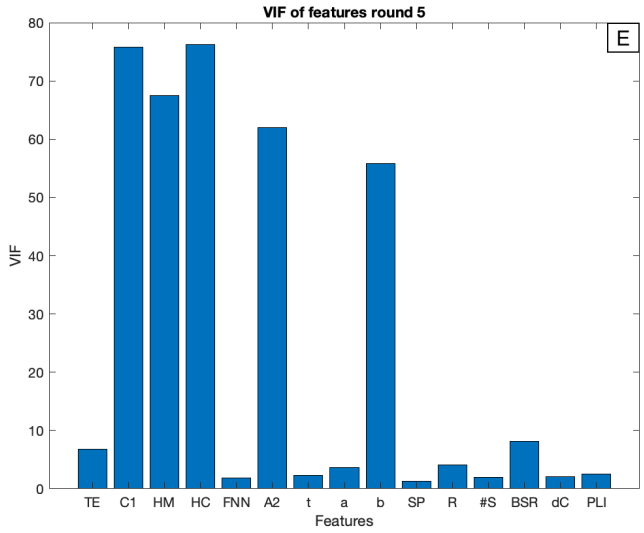
(b) Round 2: AR coeff. 1 had the highest VIF and was removed.



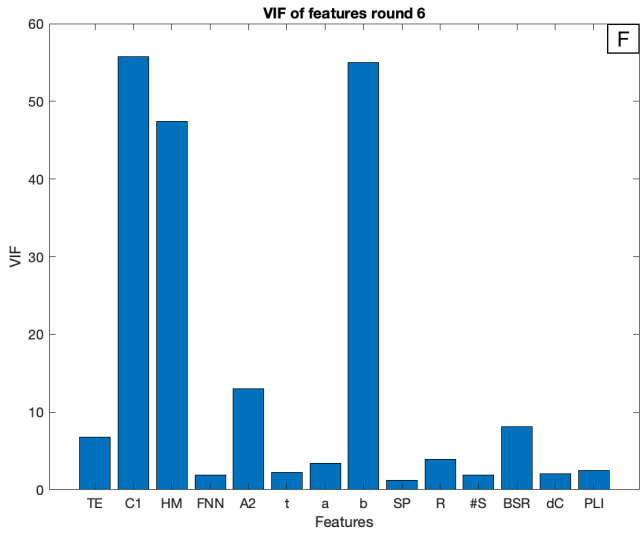
(c) Round 3: Normalised delta power had the highest VIF and was removed.



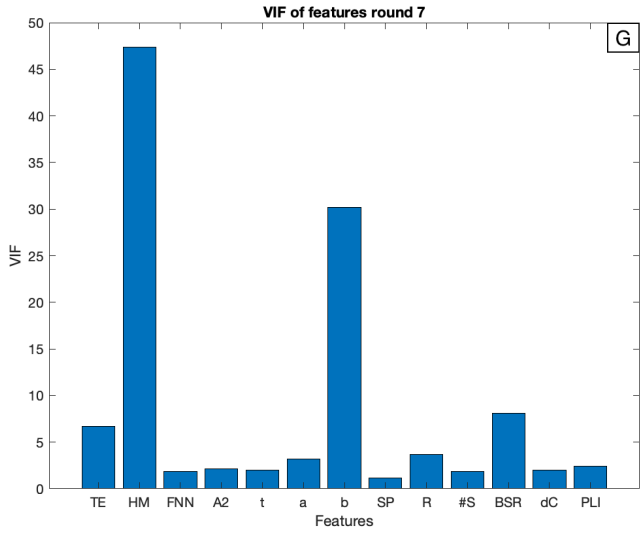
(d) Round 4: Shannon entropy had the highest VIF and was removed.



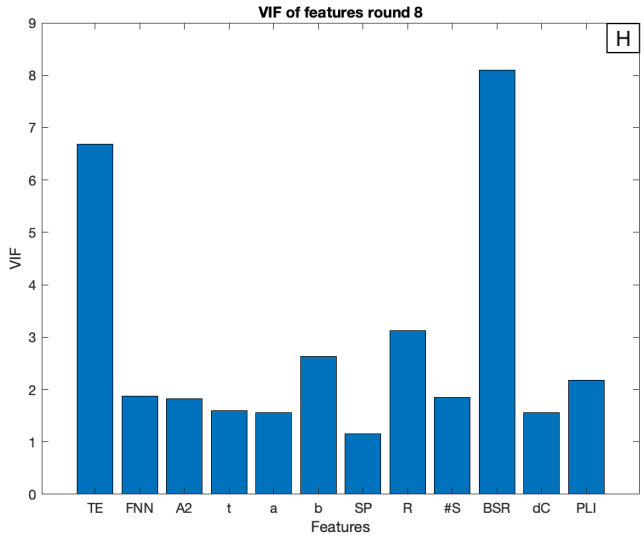
(e) Round 5: Hjorth compl. had the highest VIF and was removed.



(f) Round 6: Cepstrum coeff. 1 had the highest VIF and was removed.



(g) Round 7: Hjorth mob. had the highest VIF and was removed.



(h) Round 8: all features had a VIF below 10

Figure A.7: Bar graphs of the VIFs of all qEEG features. These graphs were used for the selection of the final feature set used as input to the models. Each round of feature exclusion, the VIFs were calculated and the highest scoring feature was removed until all features had a VIF below 10. Y-axis: VIF score. X-axis: SE=Shannon entropy, TE=Tsallis entropy, C1=cepstrum coefficient, C2=cepstrum coefficient 2, HM=Hjorth mobility, HC=Hjorth complexity, FNN=false nearest neighbor, d=normalised delta power, t=normalised theta power, a=normalised alpha power, b=normalised beta power, SP=signal power, R=regularity, #S=number of epileptic spikes, BSR=burst suppression ratio, dC=delta coherence, PLI=phase lag index. Coeff=coeffiecient. Compl=complexity. Mob=mobility. (q)EEG=(quantitative) electroencephalography. VIF=variance inflation factor.

L. Additional feature sets

I created five AFSs (Table A.V). AFS 1 included all qEEG features with low multicollinearity, equal to the FFS used as input (Table A.IV), and two clinical features: age and sex. Including clinical features like these showed slightly better performance than qEEG features alone in the work of Ghassemi et al., 2019 [52]. By comparing the model performance between AFS 1 and the FFS, I could evaluate if adding clinical features would increase the performance of the LR or the LSTM.

AFS 2 included all 19 extracted qEEG features. By comparing AFS 2 and the final feature set, I could evaluate if the LSTM model benefitted from excluding features with high multicollinearity. AFS 3 included all 19 extracted qEEG features and the clinical features age and sex, which again allowed for evaluating the influence of clinical features.

AFS 4 was created by excluding qEEG features that showed little discrimination between good and poor outcome. The averaged features across all time-fragments per epoch were used in this analysis (matrix R^{fxe}, Appendix I). I analysed the features with boxplots. A feature was included in AFS 4 if it satisfied either of the following two criteria (Fig. A.8). (1) A minimal of 25% of the epochs' feature values with poor neurological outcomes did not lie in the range of values from epochs with good neurological outcomes or vice versa. Outliers were excluded. In other words, 25% of the data in either of the outcomes classes was invariably associated with that outcome class. (2) The feature's median value from epochs with a poor neurological outcome lied outside the range between the 25 and 75 percentiles of the feature values from epochs with a good neurological outcome, and vice versa. Table A.VI states the results of the evaluation of both criteria for all features. Figure A.9 shows the boxplots of all features. The MATLAB script used to create the boxplots is placed in Appendix Z.5.

AFS 5 was created by excluding features with high multicollinearity from AFS 4 using the same process as discussed in Appendix K. Figure A.10 displays the five rounds of high VIF feature exclusion to create the final feature set. The MATLAB script used for the process of high VIF feature exclusion is placed in Appendix Z.2.

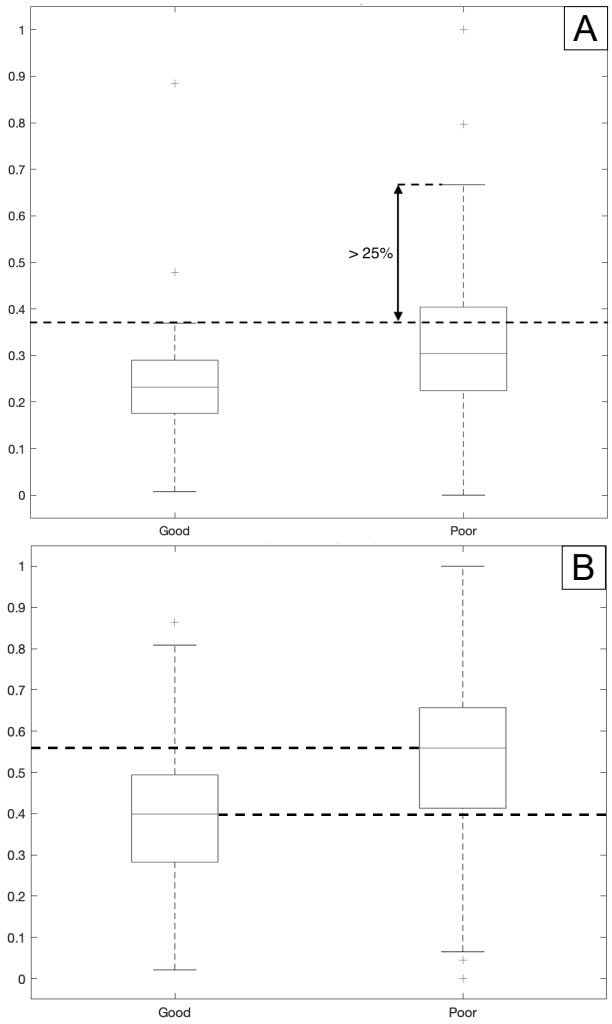


Figure A.8: Examples of feature boxplots that satisfy the criteria for feature inclusion in AFS 4. The x-axes denote good or poor neurological outcomes, the y-axes the feature value. The rectangle is the box, where the bottom edge denotes the 25th percentile and the top edge the 75th percentile. The central mark within the box indicates the median of the data. The whiskers extend to the most extreme data points, outliers excluded. The '+' symbol denotes individual outliers. A) Example boxplot of a feature that satisfies criterion 1: A minimal of 25% of the feature values from patients with a poor neurological outcome does not lie in the range of values from patients with a good neurological outcome or vice versa. B) Example boxplot of a feature that satisfies criterion 2: The feature's median value from patients with poor neurological outcome lies outside the range between the 25 and 75 percentiles of the feature values from patients with good neurological outcome and vice versa.

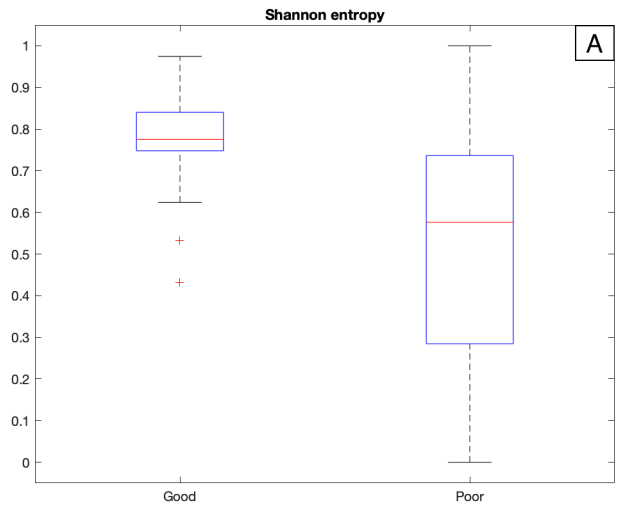
Table A.V: Additional feature sets. AR=autoregressive. BSR=burst suppression ratio. Coeff=coefficient. FNN=false nearest neighbour. PLI=phase lag index. qEEG=quantitative electroencephalography. VIF=variance inflation factor.

	AFS 1	*AFS 2*	*AFS 3*	*AFS 4*	*AFS 5*
Description	QEEG features with low multi-collinearity and clinical features	All 19 extracted qEEG features	All 19 extracted qEEG features and clinical features	QEEG features that show high discriminative power	QEEG features that show high discriminative power and low multicollinearity
Included features	1. Tsallis entropy 2. FNN 3. AR coefficient 4. normalised theta 5. normalised alpha 6. normalised beta 7. Signal power 8. Regularity 9. Epileptic spikes 10. BSR 11. Delta coherence 12. PLI 13. Age 14. Sex	1. Shannon entropy 2. Tsallis entropy 3. Cepstrum coeff. 1 4. Cepstrum coeff. 2 5. Hjorth mobility 6. Hjorth complexity 7. FNN 8. AR coeff. 1 9. AR coeff. 2 10. normalised delta 11. normalised theta 12. normalised alpha 13. normalised beta 14. Signal power 15. Regularity 16. Epileptic spikes 17. BSR 18. Delta coherence 19. PLI	1. Shannon entropy 2. Tsallis entropy 3. Cepstrum coeff. 1 4. Cepstrum coeff. 2 5. Hjorth mobility 6. Hjorth complexity 7. FNN 8. AR coeff. 1 9. AR coeff. 2 10. normalised delta 11. normalised theta 12. normalised alpha 13. normalised beta 14. Signal power 15. Regularity 16. Epileptic spikes 17. BSR 18. Delta coherence 19. PLI 20. Age 21. Sex	1. Shannon entropy 2. Tsallis entropy 3. Cepstrum coeff. 1 4. Cepstrum coeff. 2 5. Hjorth mobility 6. Hjorth complexity 7. FNN 8. normalised beta 9. Epileptic spikes 10. BSR 11. Delta coherence	1. Tsallis entropy 2. Hjorth complexity 3. FNN 4. normalised beta 5. Epileptic spikes 6. BSR 7. Delta coherence

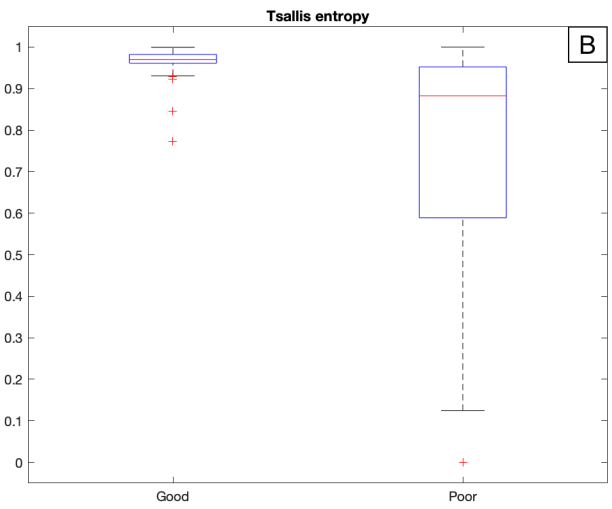
Table A.VI: Criteria evaluation for all qEEG features. If a feature satisfied either of the two criteria (visualised in Fig. A.8) it was included in AFS 4 (bold). AFS=additional feature set. AR=autoregressive. BSR=burst suppression ratio. Coeff=coefficient. FNN=false nearest neighbour. PLI=phase lag index. qEEG=quantitative electroencephalography.

Feature	Criterion 1	Criterion 2
Shannon entropy	Yes	Yes
Tsallis entropy	Yes	Yes
Cepstrum coeff. 1	Yes	No
Cepstrum coeff. 2	Yes	No
Hjorth mobility	Yes	No
Hjorth complexity	No	Yes
FNN	Yes	No
AR coefficient 1	No	No
AR coefficient 2	No	No
Normalised delta	No	No
Normalised theta	No	No
Normalised alpha	No	No
Normalised beta	Yes	No
Signal power	No	No
Regularity	No	No
Epileptic spikes	Yes	No
BSR	Yes	No
Delta coherence	Yes	No
PLI	No	No

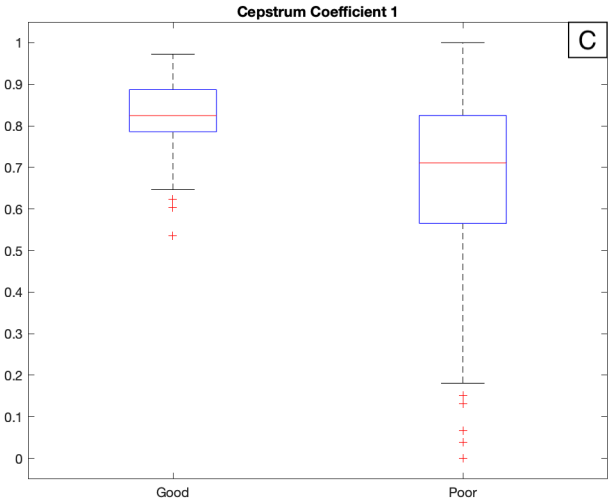
Figure A.9: Boxplots ((a) to (s)) of the feature values (y-axis) for a good outcome (left) and poor (right) neurological outcome (x-axis). The blue rectangle is termed the box, where the bottom edge denotes the 25 percentiles and the top edge the 75 percentiles. The red central mark within the box indicates de median of the data. The whiskers extend to the most extreme data points, outliers excluded. The red '+' symbol denotes individual outliers. Boxplots are visualised on the following pages.



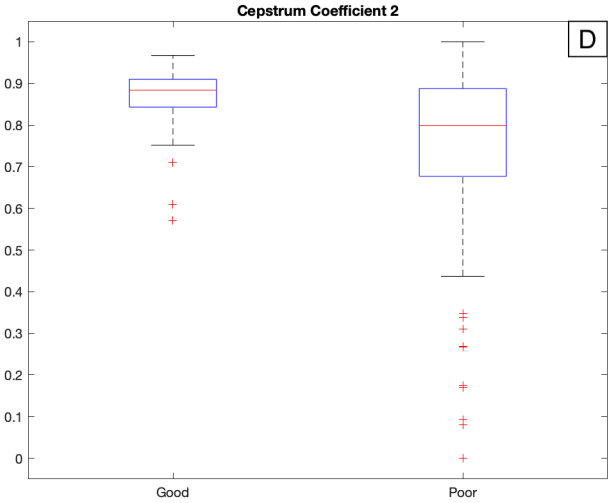
(a) Shannon Entropy



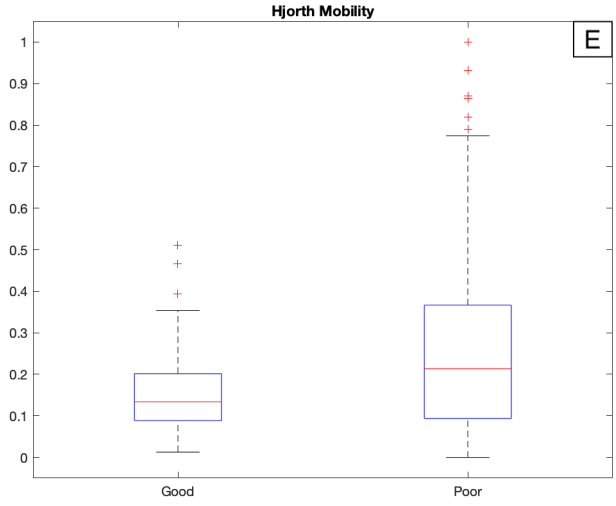
(b) Tsallis Entropy



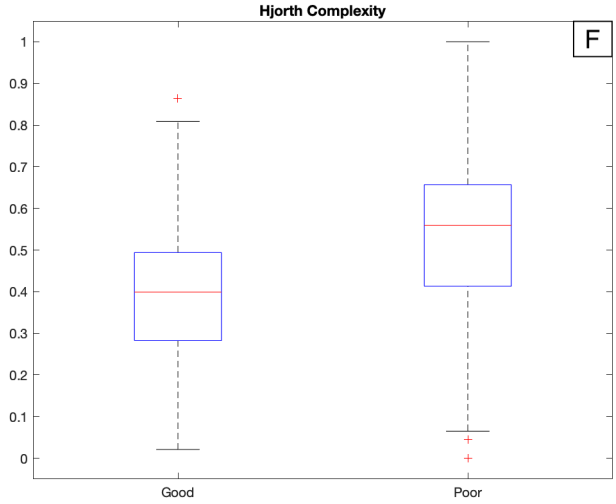
(c) Cepstrum Coefficient 1



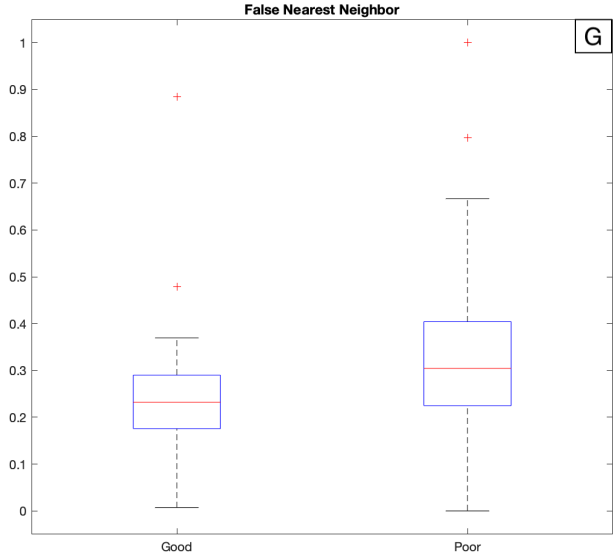
(d) Cepstrum Coefficient 2



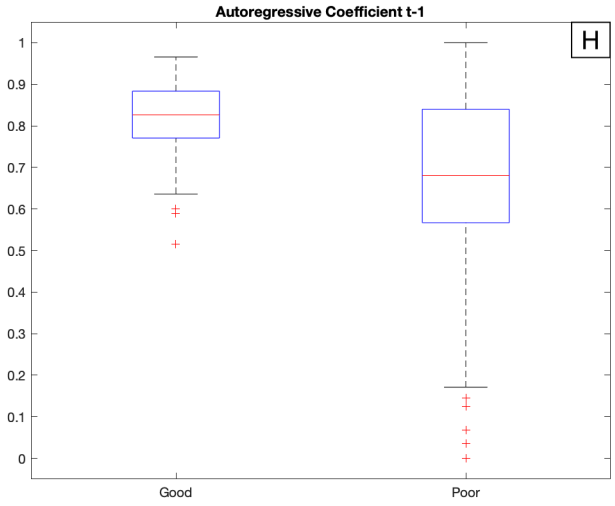
(e) Hjorth mobility



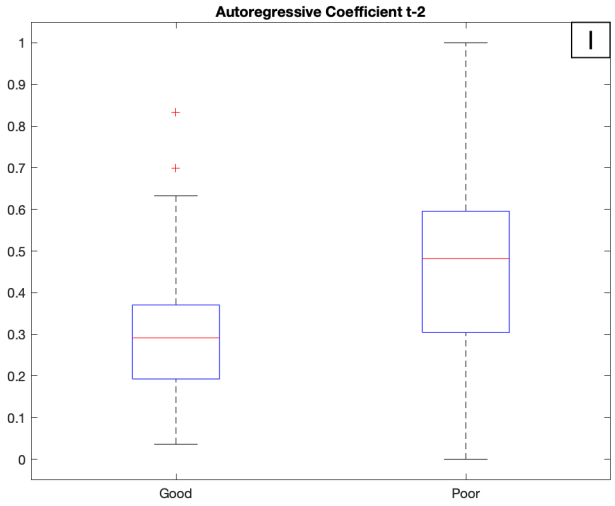
(f) Hjorth Complexity



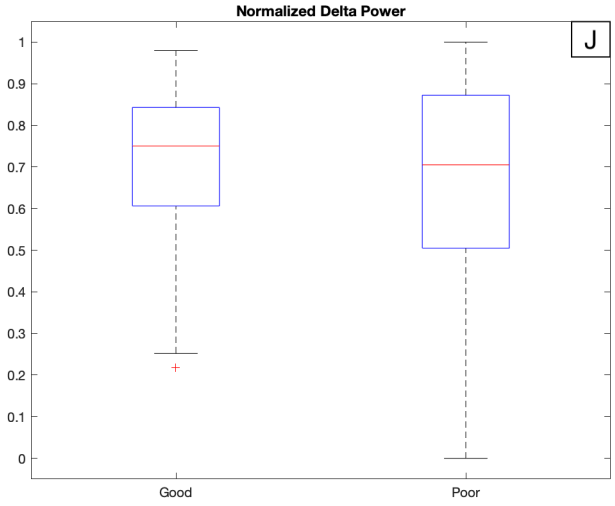
(g) False Nearest Neighbours



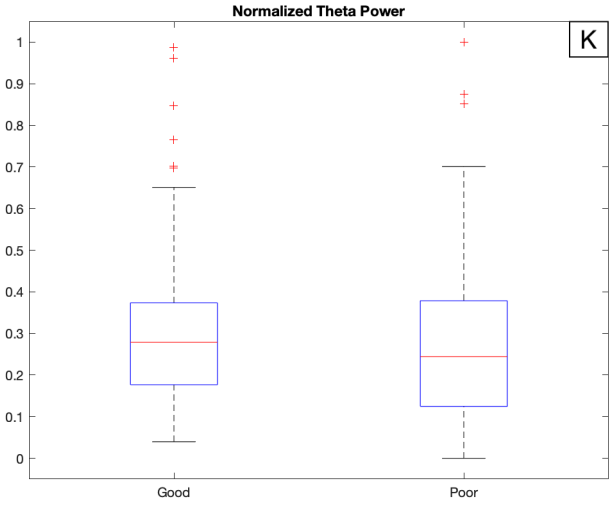
(h) Autoregressive Coefficient 1



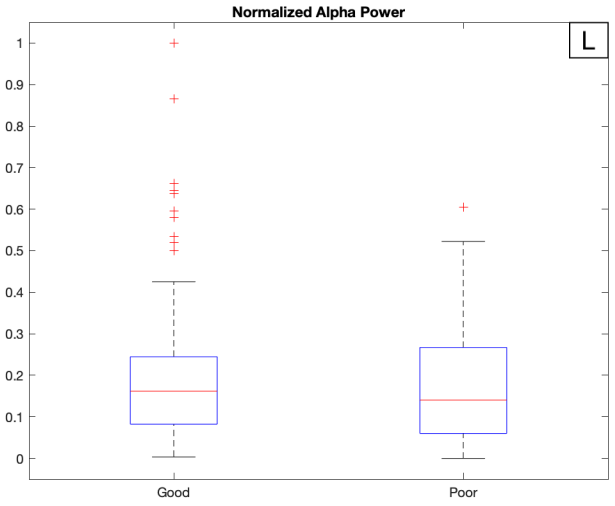
(i) Autoregressive Coefficient 2



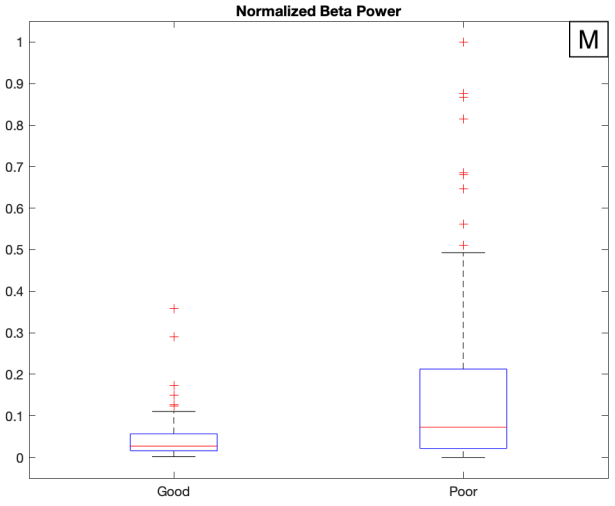
(j) Normalised Delta Power



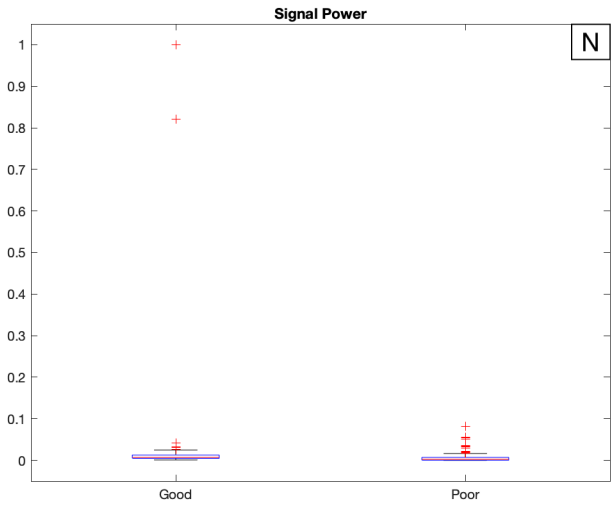
(k) Normalised Theta Power



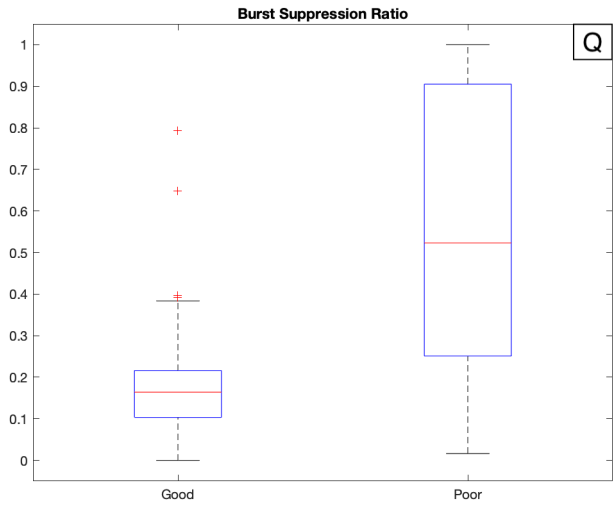
(l) Normalised Alpha Power



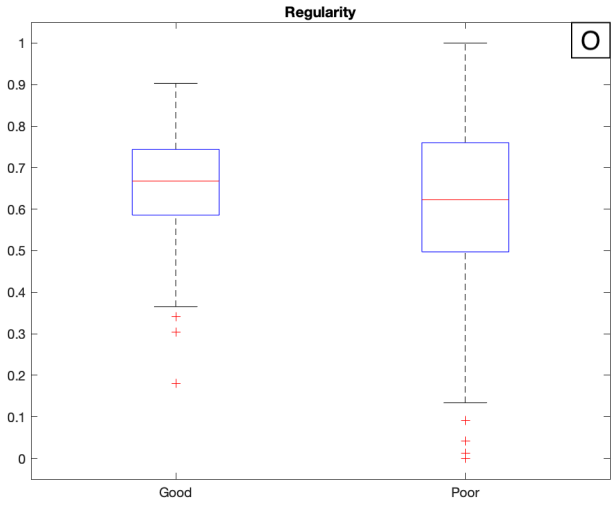
(m) Normalised Beta Power



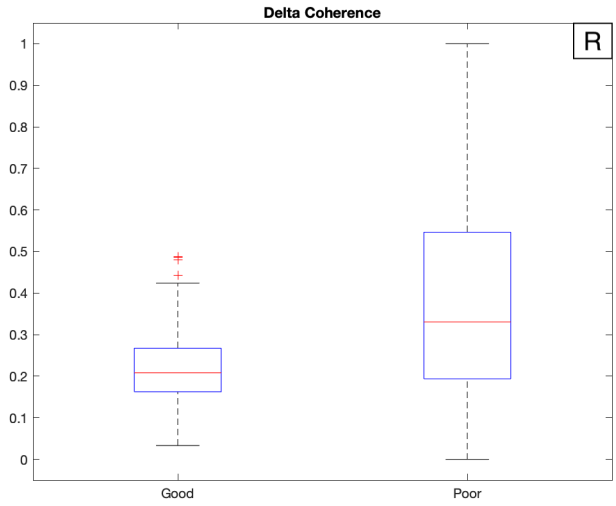
(n) Signal Power



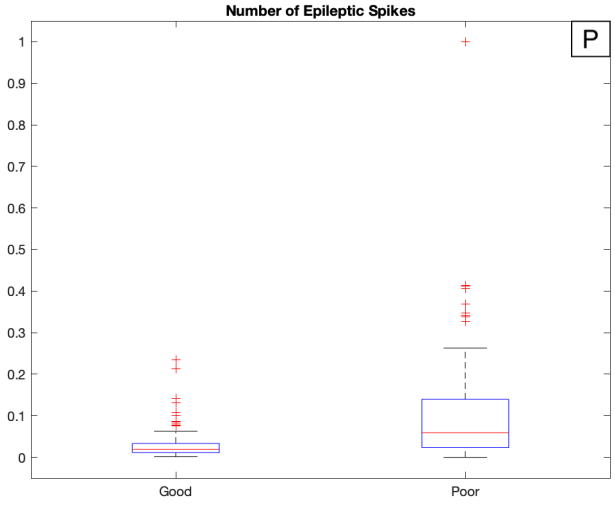
(q) Burst Suppression Ratio



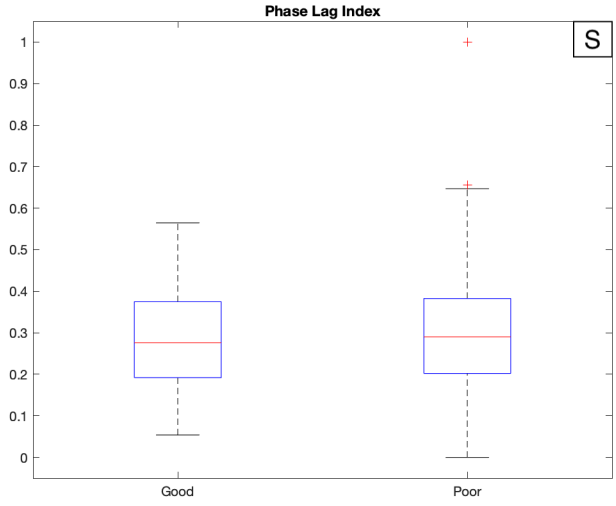
(o) Regularity



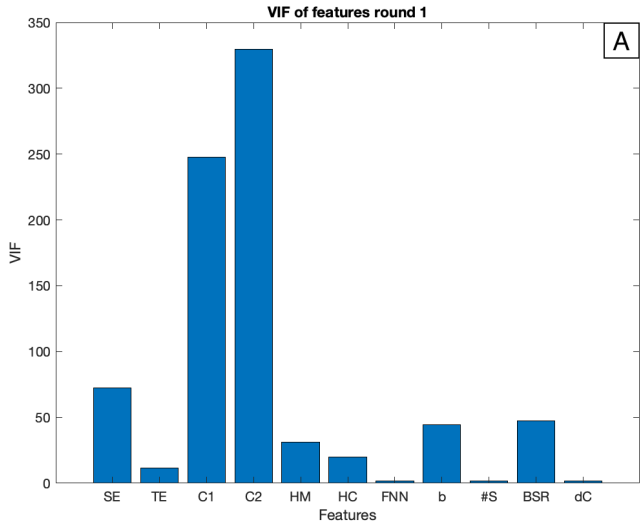
(r) Delta Coherence



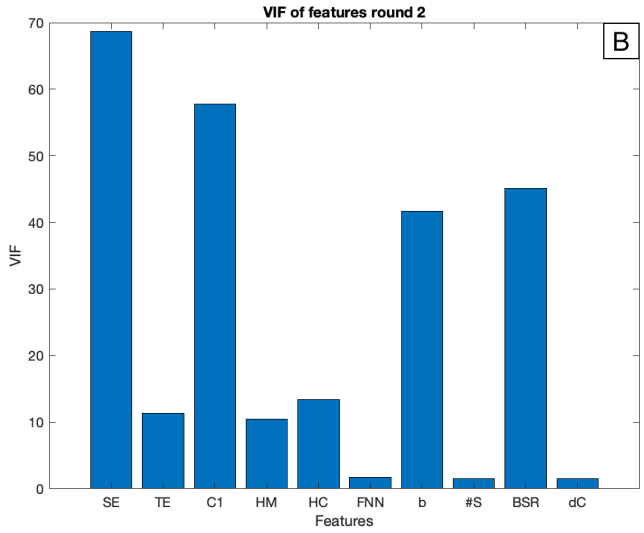
(p) Epileptic Spikes



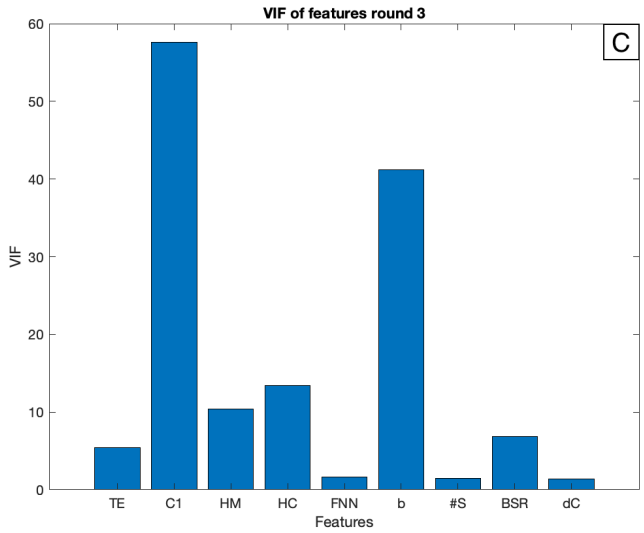
(s) Phase Lag Index



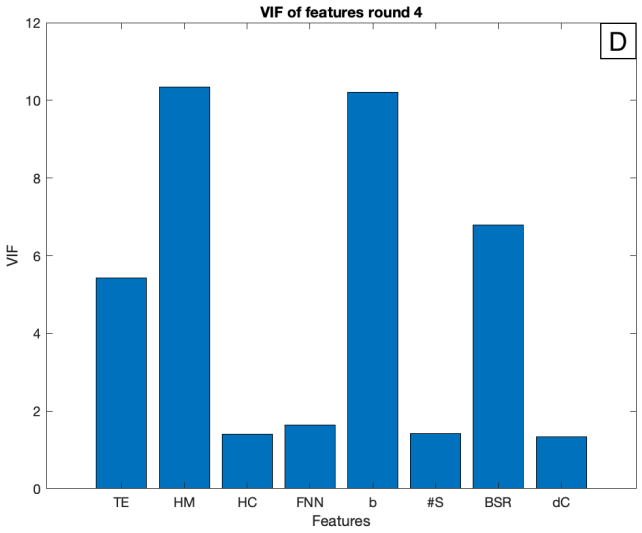
(a) Round 1: Cepstrum coeff. 2 had the highest VIF and was removed.



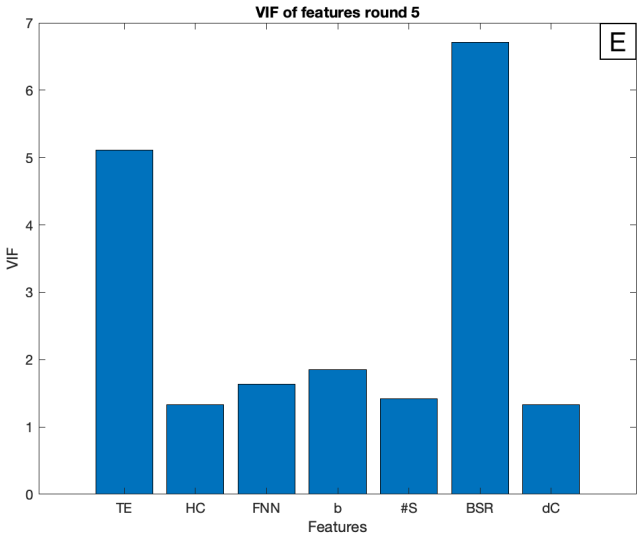
(b) Round 2: Shannon ent. had the highest VIF and was removed.



(c) Round 3: Cepstrum coeff. 1 had the highest VIF and was removed.



(d) Round 4: Hjorth mob. had the highest VIF and was removed.



(e) Round 5: all features had a VIF below 10.

Figure A.10: Bar graphs of the VIFs of the features from AFS 4. These graphs are used for feature selection for AFS 5. Each round of feature exclusion, the VIFs were calculated and the highest scoring feature was removed until all features had a VIF below 10. Y-axis: VIF score. X-axis: SE=Shannon entropy, TE=Tsallis entropy, C1=cepstrum coefficient, C2=cepstrum coefficient 2, HM=Hjorth mobility, HC=Hjorth complexity, FNN=false nearest neighbor, b=normalised beta power, #S=number of epileptic spikes, BSR=burst suppression ratio, dC=delta coherence. AFS=additional feature set. Coeff=coefficient. Ent= entropy. Mob=mobility. VIF=variance inflation factor.

M. Performance metrics

In this study, a poor neurological outcome was the positive class (1) and good neurological outcome the negative class (0). I define specific metrics to clarify the subsequent calculations of the performance metrics [58, 78, 136, 137]:

- True positive (TP) = the number of epochs correctly predicted as a poor outcome
- False positive (FP) = the number of epochs incorrectly predicted as a poor outcome
- True negative (TN) = the number of epochs correctly predicted as a good outcome
- False negative (FN) = the number of epochs incorrectly predicted as a good outcome
- True positive rate (TPR) or sensitivity

$$TPR = Sensitivity = \frac{TP}{TP + FN} \tag{A.12}$$

- True negative rate (TNR) or specificity

$$TNR = Specificity = \frac{TN}{TN + FP} \tag{A.13}$$

- False positive rate (FPR)

$$FPR = \frac{FP}{FP + TN} \tag{A.14}$$

- False negative rate (FNR)

$$FNR = \frac{FN}{FN + TP} \tag{A.15}$$

Area under the receiver operating characteristic curve

The receiver operating characteristic (ROC) curve shows the predictive ability of the model for all thresholds. The curve is made by plotting the TPR against the FPR. The area under this curve (AUC) indicates the discriminative power of the model. If the AUC approaches 1, the model can perfectly discriminate between poor and good neurological outcome. When the area is close to 0.5, the model's predictions are close to chance [54]. AUC is preferred over accuracy if the dataset is imbalanced, to avoid overfitting one specific class [138, 139].

Sensitivity at specificity

From the ROC, one can evaluate the model's sensitivity at a certain specificity threshold, as the two are plotted against each other. In this study, the model was evaluated on its sensitivity with 100% specificity for predicting poor outcome and its sensitivity with 95% to predict good outcome.

N. Background information about neural networks

The first algorithm of neural networks was published by Rosenblatt in 1957 and is termed Perceptron (Fig. A.11A) [140]. The Perceptron models a human neuron. The computations in the Perceptron algorithm are as follows.

$$y = f(W \cdot X + b) \tag{A.16}$$

$$f(a) = f \begin{cases} +1, a \geq 0 \\ -1, a \leq 0 \end{cases} \tag{A.17}$$

To make a prediction based on the input, the neuron calculates the weighted sum of the input, X, with their corresponding weights, W, (and optionally a bias, b). It activates this sum with an activation function, $f()$. This activation function is a step function. The output, y, is the result predicted by Perceptron. The weights and bias are the unknown model parameters which need to be determined with training data that includes the input and true output. During training, the neuron's predicted output is compared to the true output, and the error is fed back. With the use of the error, the model parameters are estimated. The Perceptron, being just one neuron in a single layer, is considered a prototype of neural networks. The neuron will henceforth be termed unit, consistent with the terminology often seen in the literature. The Perceptron unit can be combined to form multiple units within a layer. Moreover, multiple layers can be connected. Figure A.11B displays an example of a feedforward neural network. The units can employ other activation functions than a step function. With non-linear activation functions, the networks can map non-linear relationships between input and output. Deep learning networks refer to neural networks with many layers of units [41, 42, 58].

O. Logistic Regression – Model parameter estimation and mathematical background

LR can be classified as a neural network. LR regression is similar to the Perceptron but uses a different activation function (Fig A.12). LR predicts the probability that the response variable, in this study the neurological outcome, is equal to 1 based on a set of explanatory variables, in this study the features. The probability that the outcome has a value of 1 is termed the predicted value, p. As poor outcome defined the positive class, p equalled the probability of a poor neurological outcome. The relation between the predicted value and the features in LR is established in a three-step process. First of all, the relationship between the features and the linear predictor of the logistic model is defined as

$$z_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_n x_{in} + b \tag{A.18}$$

Or in vector notation

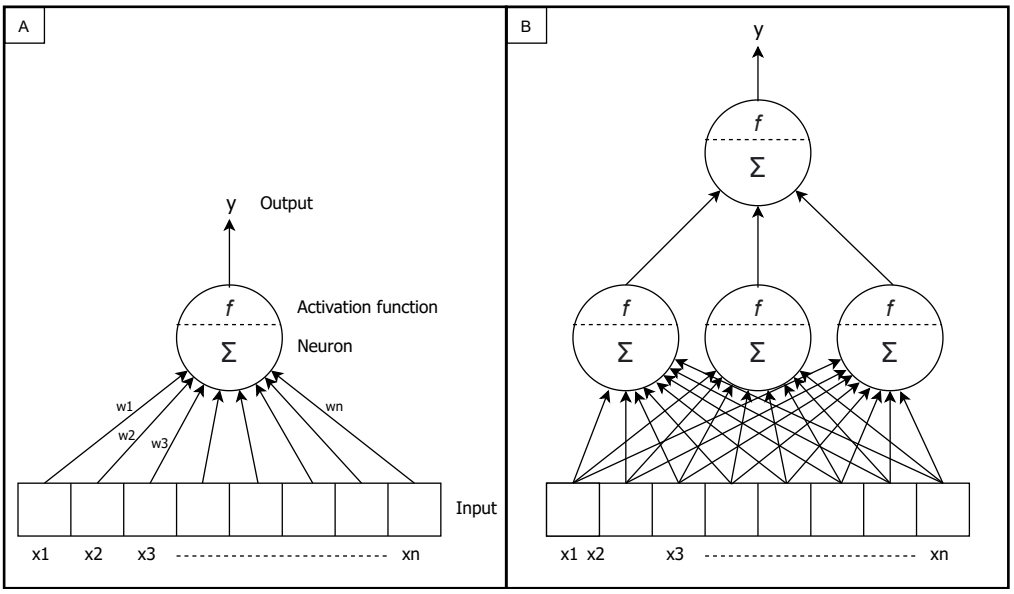


Figure A.11: A) A perceptron. This algorithm, consisting of just one unit in one layer, is considered a neural network prototype. B) Multiple perceptrons connected to form a feedforward neural network. In the figures, x shows the input signal, w the weights corresponding to each input, $\sum$ the sum of the weighted inputs, f the activation function, and y the output.

$$Z = W \cdot X + b \tag{A.19}$$

Where z_i denotes the linear predictor, which is the sum of the terms in the regression, b is the bias term, and $w_i x_i$ is the term that indicates the value of the input feature x_i and its coefficient w_i, also the weight coefficient. Secondly, the relationship between the linear predictor z_i and the predicted value p_i is

$$z_i = \ln\left(\frac{p_i}{1-p_i}\right) \tag{A.20}$$

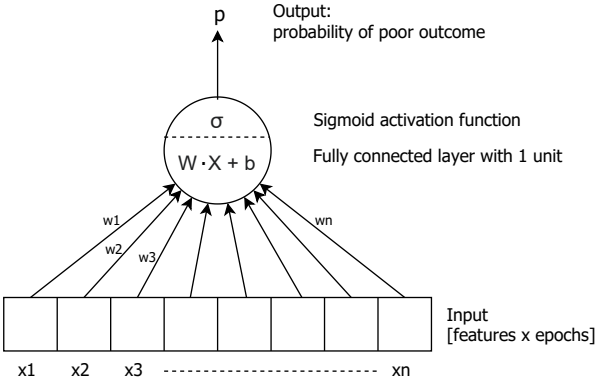


Figure A.12: Logistic regression model architecture. The input matrix was fed into a single unit which calculates the weighted sum of the input and the bias. This sum was fed into a sigmoid activation function, which mapped the value between 0 and 1. The output of the sigmoid function denoted the probability of a poor neurological outcome. X=input, W=weights, b=bias, σ=sigmoid activation function, p=predicted outcome.

Where $\left(\frac{p_i}{1-p_i}\right)$ is named the formula for odds and the log function of this formula is termed the logit function. If equation (A.18) and (A.20) are combined, the resulting relationship between the predicted value and the features is obtained:

$$\ln\left(\frac{p_i}{1-p_i}\right) = z_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_n x_{in} + b \tag{A.21}$$

Finally, to calculate the predicted value, the logit function needs to be solved for p_i [41, 58, 59, 76, 78].

$$p_i = \frac{e^{z_i}}{1+e^{z_i}} = \frac{1}{1+e^{-z_i}} \tag{A.22}$$

The LR model's parameters, the bias term and the weight vector have to be estimated. The estimation of the model parameter compromises six steps [41, 42, 59]:

1. Construct the model with an initial set of random model parameters
2. Forward propagation: for a batch of samples predict the outcome based on the features
3. Compute the loss, which is the difference between the predicted outcomes made by the model and true outcomes
4. Backward propagation: compute the gradients of the loss with respect to the model parameters and
5. Adjust the model parameters to minimise the loss using the gradients
6. Repeat the process from step 2, iteratively updating the model parameters

P. Logistic regression – Hyperparameters

Table A.VII summarises all hyperparameters implemented in the LR or explored in the hyperparameter optimization. Moreover, I discuss the hyperparameters more detailly per category:

1. Hyperparameters to build the model
2. Hyperparameters to compile the model
3. Hyperparameters to fit the model

Building hyperparameters

These hyperparameters concerned building the model and were mostly determined by the LR model's design (Fig A.12). One dense layer composed of one unit with a sigmoid activation function (Fig. A.13) was used to model the LR, as this results in a one-dimensional probability [58, 146]. For the first iteration in the learning process, the weights must be randomly initialised to small values to break symmetry during learning [60, 77]. By default, Keras uses the Glorot uniform initialiser for the kernel weights [79]. The Glorot uniform, also termed Xavier initialisation, samples from a random uniform distribution within the following limit.

$$limit = \sqrt{\frac{6}{fan_{in} + fan_{out}}} \tag{A.23}$$

Where fan_{in} and fan_{out} refer to the number of input units in the layer's weight tensor and the number of outgoing units from that layer, respectively. The limit results in a constant variance of activations and gradients through the network [141]. As the sigmoid and tanh activation functions require a constant variance of input, the Xavier initialisation is often used in combination with these activations [147]. Therefore, I used the default Xavier initialiser for the kernel initialiser in the LR. As biases are usually initialised to 0 [77], and Keras' default is the zeros class, I also initialised the biases in my model to 0.

Compiling hyperparameters

After the network was built, it was compiled. Compilation computes the matrix transformations of the defined network for the use of the processor. Compilation requires a specification of the loss function to evaluate the network and the optimiser algorithm. Furthermore, additional metrics to the loss can be specified, which are calculated while the model is fitted [79].

Loss function An often-used framework for estimation is the maximum likelihood estimation (MLE) [76, 149]. MLE attempts to find the optimum values for the model parameters from the provided training data. The training data in this study included the features and

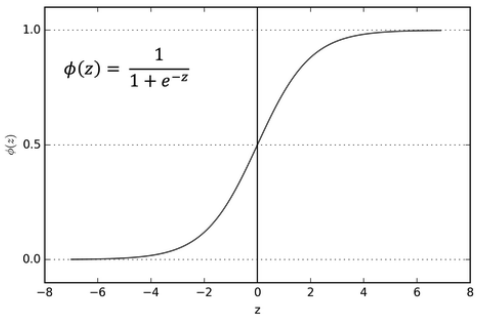


Figure A.13: A plot of the sigmoid function. Adapted from "Activation Functions in Neural Networks" by Sharma, S, 2017 (https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6) [148]

outcomes. Under the MLE framework, the difference between i) the probability distribution of the outcome predictions made by the LR based on the features in the training set, and ii) the probability distribution of the true outcomes in the training set, was measured using cross-entropy. For binary classification, a binary cross-entropy loss function is the best choice for computing the loss [41, 59]. Binary cross-entropy is defined as follows [146, 150].

$$BCE = -\frac{1}{m}\sum_{i=1}^{m} y \log(p) + (1-y)\log(1-p) \tag{A.24}$$

In this equation, m denotes the number of samples used (also termed the batch size), p the probability that the outcome is equal to 1 (the predicted value), and y the true outcome. The function is designed in such a way that false predictions are not only penalised, but the ones that are confident in their incorrect prediction result in a more severe penalty, thus in a larger loss, than less confident ones (Fig. A.14) [150].

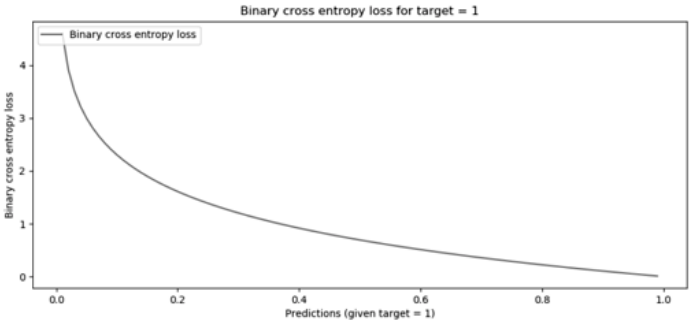


Figure A.14: A plot of the binary cross-entropy function. Adapted from "How to use binary & categorical cross-entropy with Keras?" by Chris, 2019 (https://www.machinecurve.com/index.php/2019/10/22/how-to-use-binary-categorical-crossentropy-with-keras/) [151])

Table A.VII: Logistic regression hyperparameters. AUC=area under the curve of the receiver operating curve. LR=logistic regression. ML=machine learning. NAG=Nesterov accelerated momentum. Ref=references. SeSp100=sensitivity at 100% specificity (for poor outcome prediction). SeSp95=sensitivity at 95% specificity (for good outcome prediction). SGD=stochastic gradient descent.

Category	**Hyperparameter**	**Option(s) implemented**	**Reason behind the implemented option(s)**	**Ref.**
Build	Type and number of layers	1 dense layer	Defined by design of the LR model.	[58] [59] [77]
	Number of units	1		
	Activation function	Sigmoid		
	Kernel initialiser	Glorot uniform (default)	Weights are required to be randomly initialised to small values to break symmetry during learning. As the sigmoid activation function was used, the Glorot uniform initialiser was the most appropriate for the kernel. As biases are typically initialised to 0, this is also done in this study.	[60] [77] [79] [141]
	Bias initialiser	Zeros (default)		
Compile	Loss function	Binary cross-entropy	Binary cross-entropy is the most often used and preferred loss function for binary classification problems.	[58] [59] [77]
	Optimiser	1) Mini batch SGD	SGD is the most commonly used optimiser in ML. It updates model parameters by subtracting a fraction of the gradient from the current model parameters.	[41] [58] [59] [77]
		2) Mini batch SGD using NAG	Applying NAG to SGD could result in more stable learning behavior and faster convergence.	[78] [142] [143]
	Learning rate	1) Mini batch SGD rate = 0.1	Determines the degree with which the model parameters are updated. For SGD a learning rate of 0.1 was used to achieve convergence in a reasonable amount of time. NAG allows for smaller learning rates. Therefore, a learning rate of 0.01 was used. Larger learning rate gives the advantage of less computational time.	[41] [142] [143]
		2) Mini batch SGD using NAG rate = 0.01		
	Metrics	AUC, SeSp100, SeSp95	See Section 2.6	
Fit	Number of epochs	SGD: 200 to 3000 with step size 200 SGD with NAG: 100 to 6000 with step size 500	Number of times the complete dataset is passed through the model once. Too few epochs result in underfitting, too many in overfitting. The search space was decided based on several initial runs.	[58] [59]
	Batch size	32 (default)	Determines the number of samples used to compute the gradient of the loss function and update the model parameters accordingly. It does not significantly influence performance. A default of 32 is confirmed to be appropriate and was thus used.	[77] [144] [145]

Optimiser and learning rate The applied optimisation method was the commonly used mini batch variant of Stochastic Gradient Descent (SGD), often referred to as just SGD [41, 58, 59]. The iterative process of SGD optimised the model parameters to minimise loss. During backwards propagation, the gradients of the loss function with respect to the parameters were calculated per stochastically chosen batch with size m.

$$\frac{\partial BCE}{\partial W} = \frac{1}{m}\sum_{i=1}^{m} F\,(p-y)^T \tag{A.25}$$

$$\frac{\partial BCE}{\partial b} = \frac{1}{m}\sum_{i=1}^{m} (p-y) \tag{A.26}$$

Subsequently, the model parameters were updated in the opposite direction of the gradients. Therefore, the gradients were multiplied with a learning rate and subtracted from the previous model parameter values.

$$W = W - \propto \bigtriangledown BCE\,(W) \tag{A.27}$$

$$b = b - \propto \bigtriangledown BCE\,(b) \tag{A.28}$$

Where $\propto \delta W$ and $\propto \delta b$ are termed the update vectors and $\propto$ denotes the learning rate. The learning rate determines how much the model parameters are updated due to the loss [41]. A larger learning rate results in big step sizes and rapidly reaches a solution. However, it might converge to a suboptimal solution, as the step sizes are too big to find the minimum loss. On the other hand, a small learning rate might also result in a suboptimal solution as it could get stuck in a local minimum. Moreover, a small learning rate could take too long to

converge [59, 152, 153]. To overcome these challenges, algorithms can be added to the SGD. As small learning rates can get stuck in local minima and are slow in their convergence, momentum is often added to accelerate the learning process in the appropriate direction [59, 142]. With momentum, a fraction of the update vector of the previous adjustment step is included in the current update vector [58, 142].

$$v_t = \beta v_{t-1} + \propto \bigtriangledown BCE\,(W) \tag{A.29}$$

$$W = W - v_t \tag{A.30}$$

$$v_t = \beta v_{t-1} + \propto \bigtriangledown BCE\,(b) \tag{A.31}$$

$$b = b - v_t \tag{A.32}$$

Where β is labelled the momentum term and sets the fraction of the previous update vector added. A momentum term of 0.9 is most commonly used [152, 154]. Momentum stimulates the SGD to update for dimensions of which the gradients point in the same direction, and discourages updates for dimensions of which the gradients change direction. Momentum results in significantly faster convergence and dampens oscillatory learning behaviour [142, 143]. Furthermore, Nesterov accelerated gradient (NAG) or also Nesterov momentum, is similar to momentum. NAG computes the gradient from the point the current momentum is directed to using an approximation of the parameters' future position. Therefore, it anticipates the future direction of the momentum and can adjust the inner parameters accordingly. As a result of this anticipatory behaviour, NAG shows more stable learning [143]. SGD updates with applied NAG are computed as follows.

$$v_t = \beta v_{t-1} + \propto \bigtriangledown BCE\left(W - \beta v_{t-1}\right) \tag{A.33}$$

$$W = W - v_t \tag{A.34}$$

$$v_t = \beta v_{t-1} + \propto \bigtriangledown BCE\left(b - \beta v_{t-1}\right) \tag{A.35}$$

$$b = b - v_t \tag{A.36}$$

The hyperparameter optimisation process explored both traditional SGD and SGD applying NAG. As Nesterov momentum allows for smaller learning rates, I used a learning rate of 0.01 for SGD applying NAG. For SGD without momentum, a larger learning rate of 0.1 was used to reach convergence in a reasonable number of epochs.

Metrics Additional to the loss, the metrics discussed in Section 2.6 were computed during fitting the model. These metrics included the AUC, the sensitivity at 100% specificity of poor outcome prediction, and the sensitivity at 95% specificity of good outcome prediction.

Fitting hyperparameters

Fitting the model refers to adjusting the model parameters to the training data. Fitting required the specification of a training dataset, which in my case included a feature matrix of [epochs, features] and an output matrix with neurological outcomes per epoch. Backpropagation is applied for a specified number of epochs, which was one of the hyperparameters. The optimiser adjusts the model parameters per batch, which required the second hyperparameter specification, namely the batch size [41, 58, 59].

Number of epochs For each learning rate, I tuned the optimal number of epochs. The number of epochs determines how many times the model parameters are adjusted based on the loss computed (not to be confused with the five-minute EEG recording) [42, 59]. It is crucial to set an appropriate number of epochs to train the model. If this value is set too small, the model might underfit. On the contrary, an overfit model can occur when trained for too many epochs [58, 59]. The number of epochs should compromise between underfitting and overfitting, where the chance of either one occurring is limited. For the learning rate of 0.01, the options for the number of epochs ranged from 100 to 6000. A step size of 500 was used (excluding the first increment from 100 to 500 epochs). The options for the number of epochs for the learning rate of 0.1 ranged from 200 to 3000 with a step size of 200. I did not exceed 6000 and 3000 epochs, as these models showed significant overfitting, of which the identification is discussed shortly. The step sizes were determined with trial and error.

Batch size The batch size determines the number of samples used to compute the gradient of the loss function and update the model parameters accordingly [144]. When more samples are used to update the model parameters, these are likely updated in the appropriate direction. The appropriate updates initially cause the model's loss to converge with large steps. With fewer samples, the gradient's computation will be noisy, as it is relatively sensitive to outliers in the batch. Consequently, smaller batches cause noisy model parameter updates. These noisy updates lead to slower convergence but offer an improved generalisation, and therefore often better performance [77, 144, 145]. Keras uses a default batch size of 32. Bengio and Masters & Luschi agree that 32 is an appropriate batch size, which generally results in stable training and adequate regularisation [77, 145]. Batch size does not influence the generalisation ability very much but affects training time [77]. Considering the limited effect on performance, the limited available computational resources and the confirmed good default value of 32, I used a batch size of 32 for my LSTM network.

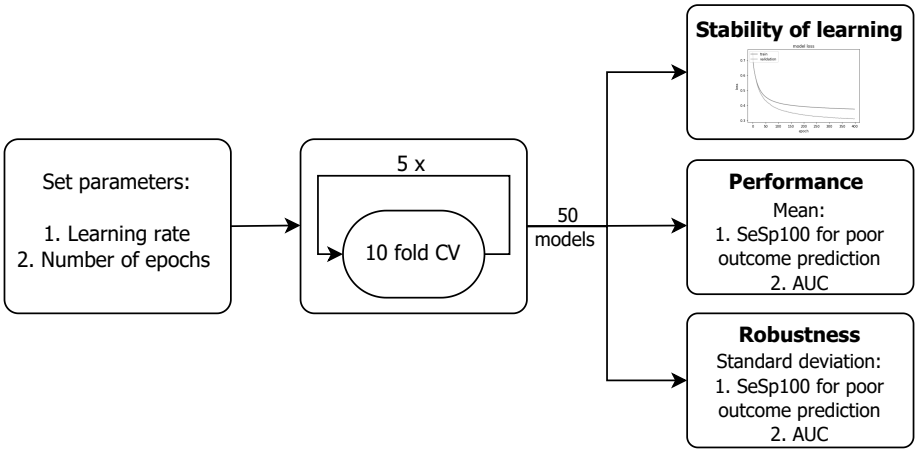


Figure A.15: The process to obtain the data with which the number of epochs with a specific learning rate were judged. First, the learning rate and the number of epochs were defined. Hereafter 10-fold stratified cross-validation was repeated five times. From the resulting 50 models, the stability of the learning behaviour near the end of training and the mean and the standard deviation of the performance metrics were obtained. AUC=area under the receiver operator curve. CV=cross-validation. SeSp100=sensitivity at 100% specificity (for poor outcome prediction)

Q. Logistic regression – Hyperparameter optimisation

Hyperparameter performance quantification

To quantify how well a hyperparameter configuration performed the model was judged based on i) the SeSp100 for poor outcome prediction and in a lesser amount the AUC (both on unseen data), ii) the robustness, and iii) the stability of the learning process near the end of training (Fig. A.15).

As the metrics were evaluated on unseen data, I studied the performance and the generalisability of the model. A dataset can be split into a training and a test set to evaluate the generalisability. The model parameters are constructed using the training set. The performance is evaluated using the unseen test set. Training a model on one training set and evaluating it on one test set is likely to give an unreliable estimate of the model's performance. This unreliable estimate is due to the possibility that outliers are present in the test set, or the test set's performance is good by chance [58]. I used 10-fold stratified cross-validation (CV) to obtain a more reliable estimate of the model's performance (Fig. A.16A). 10-fold CV split the dataset into ten folds. Each fold contained approximately the same number of poor and good outcomes. The model was trained on nine folds of the data, labelled the training set. During training, the model parameters were adjusted with every epoch, constructing the model. Following training, the model's performance was evaluated on the unseen fold that was held out, labelled the test set. Training and evaluating was repeated ten times until all folds were used exactly once as a test set. Between every repetition, the model parameters were reinitialised [58, 59, 155]. After all repetitions were completed, the scores from the test sets were averaged across the ten models. As a result, the mean of the metrics SeSp100 and AUC were obtained. For further increased reliability and a robust estimate of the model's performance, I repeated the process of 10-fold CV five times [84]. Subsequently, these five means were averaged, resulting in a grand mean.

To judge a model based on robustness, the standard deviation of the performance metrics of those repetitions was also calculated. A robust model should show little difference in performance across folds. Therefore, the preferred standard deviation was low.

The learning process's stability focused on the model's loss near the end of the number of epochs used for training. However, the loss computed on the training data after each epoch does not give insight into the model's generalisation ability. Therefore, the loss was additionally computed on unseen data after each epoch. Each repetition, 90% of the nine folds of training data, termed the true training data, was used for constructing the model parameters. The other 10% of the nine folds, termed the validation data, was used to evaluate the model's performance on data is had not seen before (Fig. A.16B). The validation set contained approximately the same proportions of poor and good outcomes. The model's loss on the validation data (the validation loss) was obtained after every epoch. For all five rounds of 10-fold CV, the model's training loss and validation loss were plotted against the number of epochs. These plots were classified into five groups, based on the validation loss near the end of training (Fig. A.18): 1) clearly decreasing validation loss, 2) slightly decreasing validation loss, 3) plateaued validation loss, 4) slightly increasing validation loss, and 5) clearly increasing validation loss. Clearly decreasing validation loss (group 1) indicated significant underfitting and was therefore not preferred. Clearly increasing validation loss (group

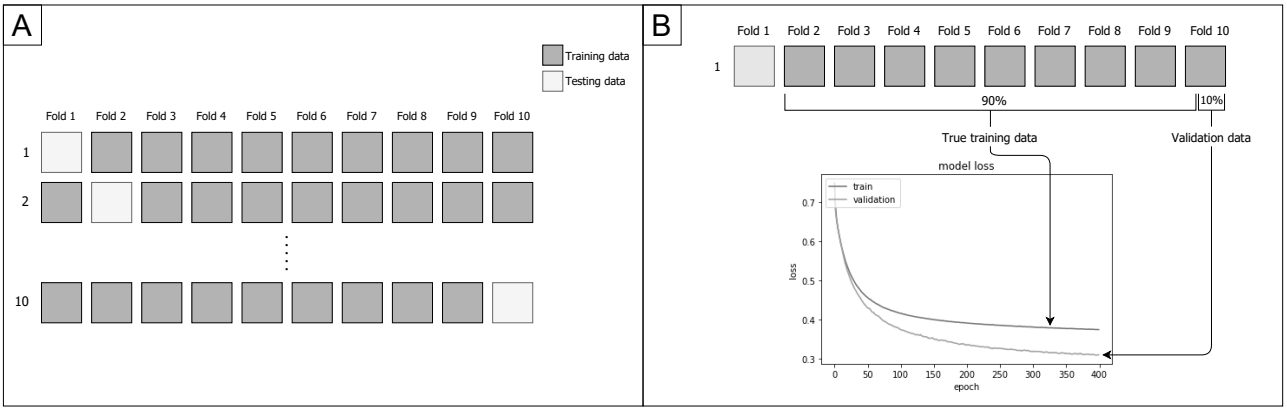
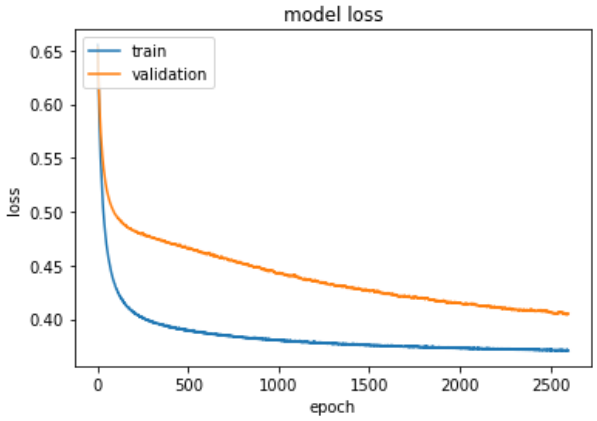
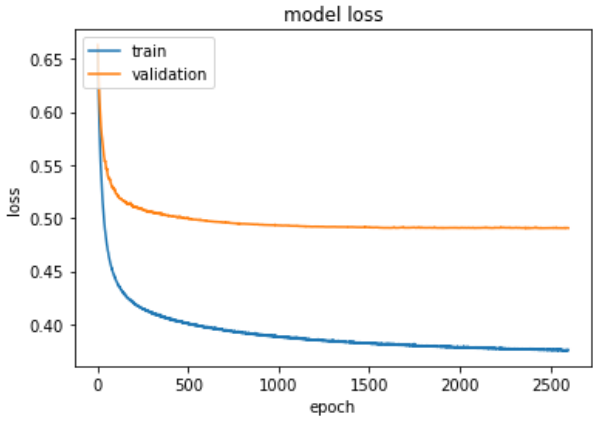


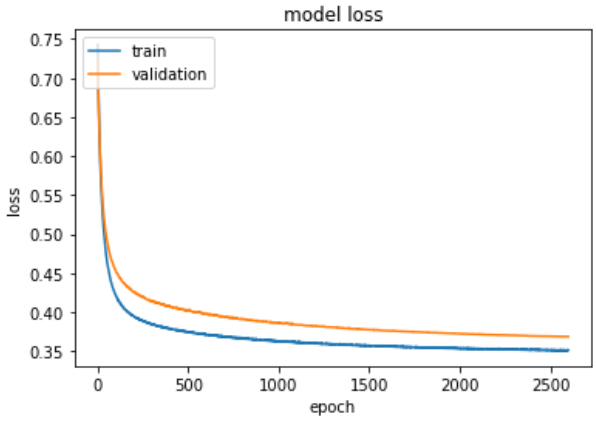
Figure A.16: A) A visualisation of 10-fold cross-validation. The dataset was split into ten folds. Nine of the ten folds were used as training data. The hold out fold was used to evaluate the model's performance. Training and evaluating was repeated ten times until all folds were used as test data exactly once. B) A visualisation of the division of the nine training folds into 90% true training data and 10% validation date. The loss on the training data and validation data was computed after every epoch. The losses on both the training data and validation data were plotted against the number of epochs trained to classify the learning behavior.The training/validation data split was done for all 10 repetitions of the 10-fold cross validation.



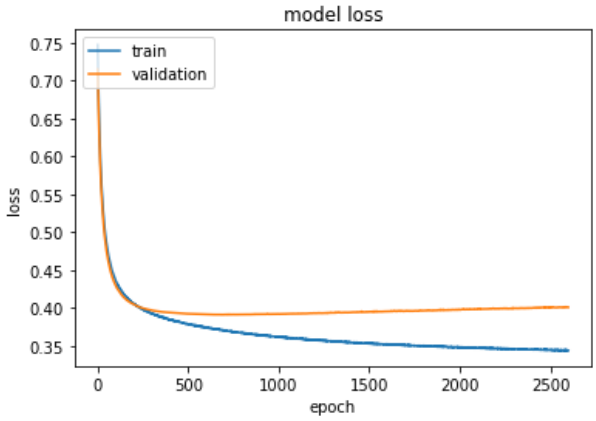
Group 1: clearly decreasing validation loss. The blue line denotes the training loss, the orange line the validation loss.



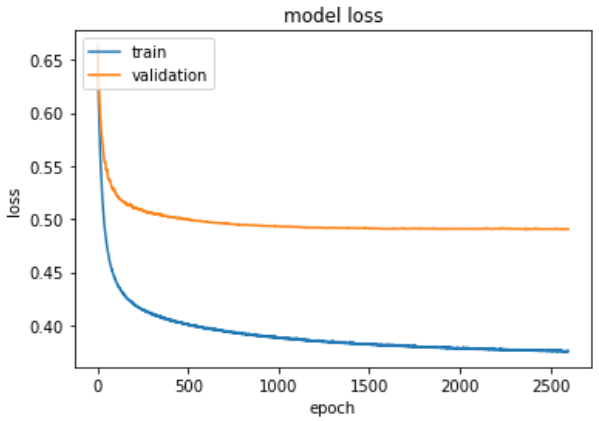
Group 3: plateaued validation loss. The blue line denotes the training loss, the orange line the validation loss.



Group 2: slightly decreasing validation loss. The blue line denotes the training loss, the orange line the validation loss.



Group 4: slightly increasing validation loss. The blue line denotes the training loss, the orange line the validation loss.



Group 5: clearly increasing validation loss. The blue line denotes the training loss, the orange line the validation loss.

Figure A.18: Plots illustrating the five groups of learning behaviour as categorised by their validation loss. The plots have the number of epochs on the x-axis and loss on the y-axis. The blue line indicates the training loss, the orange one the validation loss.

5) indicated significant overfitting and was therefore also not preferred. Ideally, the model had reached a constant validation loss (group 3). As the slightly in- or decreasing validation loss (group 2 and group 3, respectively) also resulted in a relatively stable model, these groups were also considered preferable.

Hyperparameter performance evaluation

After the 50 models per configuration were obtained using five rounds of 10-fold CV, and the configurations' performance was quantified, the best performing configuration was chosen through four steps (Fig. A.17). The same steps were completed for both learning rates. First of all, I considered the stability of learning behaviour. As severe under- and overfitting (group 1 and 5, respectively) should be avoided, the models trained for the number of epochs that often showed this behaviour were excluded. Specifically, I excluded the models that used a specific number of epochs for training from further analysis if more than 20% of them (10 or more of the 50) was classified in group 1 or 5. Secondly, for both the SeSp100 and AUC, the number of epochs producing the highest means were chosen. Thirdly, to account for the robustness, the metrics' means were divided by their standard deviation. I termed the resulting score the robust performance score. For both metrics, I listed the number of epochs with the top three highest robust performance scores. The number of epochs with the highest SeSp100 score while having a good AUC score was considered the optimal number of epochs for that learning rate. Finally, the hyperparameters of the models scoring the highest SeSp100 between the two learning rates were implemented in the final LR model.

Consequently, the learning rate and the number of epochs used for the final model gave a stable validation loss and showed good and robust performance. As stable learning behaviour was assured for the final model, no validation set was needed, and the nine folds in the 10-fold CV were used entirely for training. As a result, the model made use of the maximum data available for training.

R. LSTM – Model parameter estimation and mathematical background

Figure A.19 visualises the interactions within the LSTM cell. X_t denotes the input and h_t the output at timestep t. Figure 24 represents three equally structured LSTMs cells 'unrolled' over different timesteps. The information flow mathematics is discussed step-by-step [41, 43, 59, 83].

In Figure A.20 visualises the flow of the cell state C. The cell receives the cell state from *t-1* and can adjust this state through specific gates, including the forget, input, and output gate. The gates work like a filtering mechanism, regularising the flow of information. The gates consist of i) a NN layer with a specified number of units, implementing the operation in Equation (A.16), and applying a sigmoid activation function (Sec-

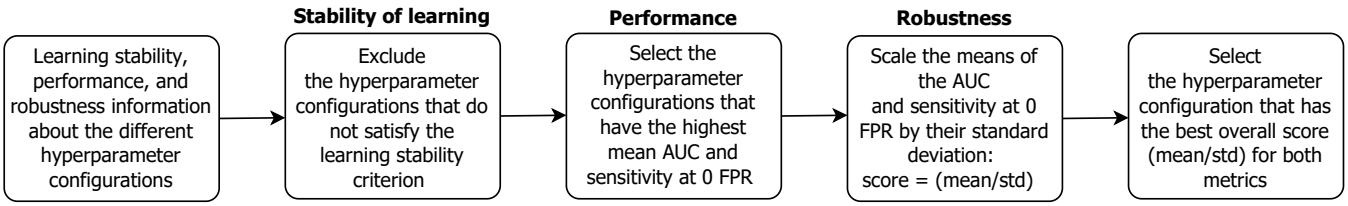


Figure A.17: The flowchart of selecting the optimal number of epochs for a specified learning rate. First of all, the numbers of epochs that resulted in 20% or more of the models classified in group 1 (clearly decreasing validation loss) or group 5 (clearly increasing validation loss) were excluded from further analysis. Subsequently, per metric the numbers of epochs that produced the highest mean were chosen. Hereafter, these means were divided by their standard deviation. The resulting score was termed the robust performance score. Finally, the optimal number of epochs was chosen based on the top 3 highest robust performance scores rankings.

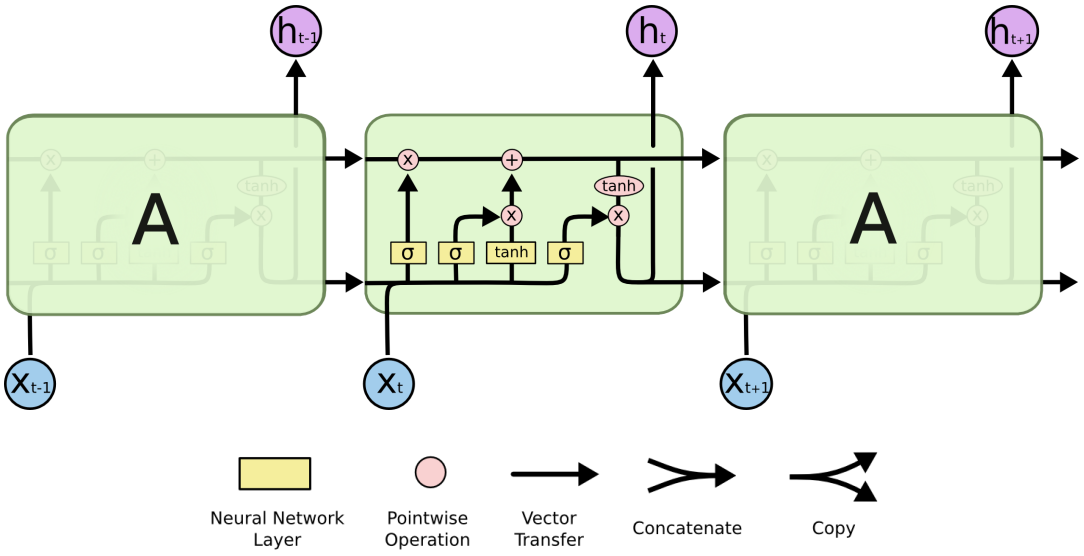


Figure A.19: LSTM cell at three consecutive timesteps. h_t=output at time-step t. σ=sigmoid function. Tanh=hyperbolic tangent function. X_t=input at timestep t. Adapted from "Understanding LSTM Networks" by Olah, C., 2015, (https://colah.github.io/posts/2015-08-Understanding-LSTMs/). LSTM=long short-term memory recurrent neural network [43]

tion 2.7, Eq. 4), and ii) an element-wise multiplication (the Hadamard product). The NN layer with the sigmoid function outputs a value between 0 and 1 for each number in the cell state. Subsequently, the gate uses this value for the multiplication. When a value of 0 is outputted, it completely closes that gate. As the element in the cell state is multiplied with 0, all information is removed. When the output is 1, the gate is fully open, and all information of that element in the cell state is passed through.

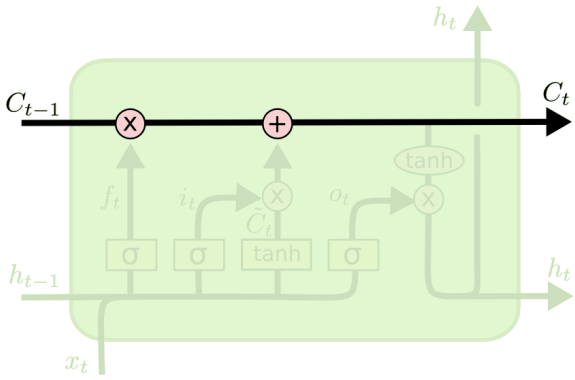


Figure A.20: The flow of the cell state through an LSTM cell. C_t=cell state at time-step t. $\widetilde{C}_t$=candidate state at time-step t. f_t=forget gate at timestep t. h_t=output at time-step t. i_t=input gate at timestep t. o_t=output gate at timestep t. σ=sigmoid function. Tanh=hyperbolic tangent function. x_t=input at timestep t. Adapted from "Understanding LSTM Networks" by Olah, C., 2015, (https://colah.github.io/posts/2015-08-Understanding-LSTMs/) [43]

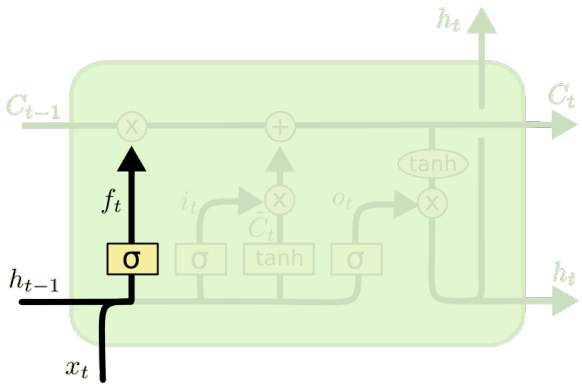


Figure A.21: Operation of the forget gate. C_t=cell state at time-step t. $\widetilde{C}_t$=candidate state at time-step t. f_t=forget gate at timestep t. h_t=output at time-step t. i_t=input gate at timestep t. o_t=output gate at timestep t. σ=sigmoid function. Tanh=hyperbolic tangent function. x_t=input at timestep t. Adapted from "Understanding LSTM Networks" by Olah, C., 2015, (https://colah.github.io/posts/2015-08-Understanding-LSTMs/) [43]

$$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1}, + b_f\right) \tag{A.37}$$

First of all, the cell decides what to remove and what to preserve from the previous cell state based on the previous output, $h_t - 1$, and the current input, x_t, with the forget gate layer f_t (A.37) (Fig. A.21). W_f and U_f denote the weight matrices and b_f the bias of the forget gate layer.

$$i_t = \sigma\left(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i\right) \tag{A.38}$$

$$\widetilde{C_t} = tanh\left(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c\right) \tag{A.39}$$

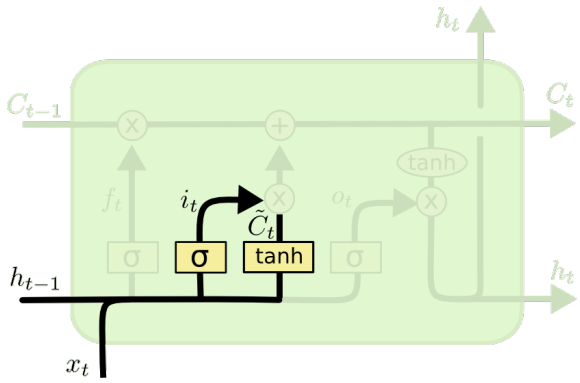


Figure A.22: Operation of the input gate and the candidate state. C_t=cell state at time-step t. $\widetilde{C_t}$=candidate state at time-step t. f_t=forget gate at timestep t. h_t=output at time-step t. i_t=input gate at timestep t. o_t=output gate at timestep t. σ=sigmoid function. Tanh=hyperbolic tangent function. x_t=input at timestep t. Adapted from "Understanding LSTM Networks" by Olah, C., 2015, (https://colah.github.io/posts/2015-08-Understanding-LSTMs/) [43]

The input gate also uses the concatenated $h_t - 1$ and x_t to determine what information should be added to the cell state (Fig. A.22). This process is composed of two parts. The input gate layer, i_t, decides which elements in the cell state should be updated (A.38). Meanwhile, the candidate state, $\widetilde{C_t}$, is created to decide what information should be added to the update (A.39). $\widetilde{C_t}$ is generated by passing $h_t - 1$ and x_t through a NN layer operating with Equation 16 and using a hyperbolic tangent (tanh) (A.40) (Fig. A.23). Tanh is used to obtain a value between -1 and +1, providing the possibility to either increase or decrease elements in the cell state. W_i, U_i, b_i and W_c, U_c, b_c represent the weight matrices and the biases.

$$tanh\,(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \tag{A.40}$$

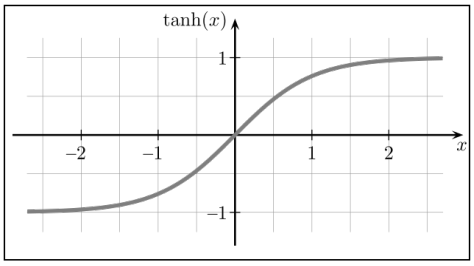


Figure A.23: Plot of the hyperbolic tangent function. Tanh=hyperbolic tangent. Adapted from "Hyperbolic Tangent" by Wikimedia, 2020, (https://commons.wikimedia.org/wiki/File:Hyperbolic_Tangent.svg) [156]

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{A.41}$$

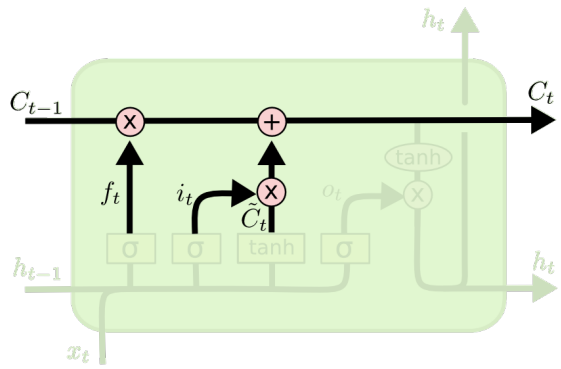


Figure A.24: Operation of the update of the cell state. C_t=cell state at time-step t. $\widetilde{C_t}$=candidate state at time-step t. f_t=forget gate at timestep t. h_t=output at time-step t. i_t=input gate at timestep t. o_t=output gate at timestep t. σ=sigmoid function. Tanh=hyperbolic tangent function. x_t=input at timestep t. Adapted from "Understanding LSTM Networks" by Olah, C., 2015, (https://colah.github.io/posts/2015-08-Understanding-LSTMs/) [43]

Subsequently, the candidate state is multiplied by the input gate and added to the previous cell state, creating a new cell state (A.41) (Fig. A.24).

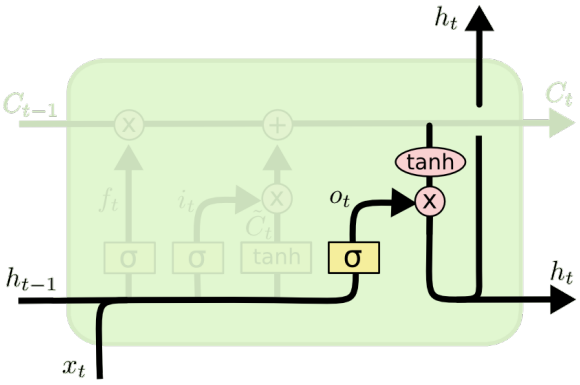


Figure A.25: Operation of the output gate. C_t=cell state at time-step t. $\widetilde{C_t}$=candidate state at time-step t. f_t=forget gate at timestep t. h_t=output at time-step t. i_t=input gate at timestep t. o_t=output gate at timestep t. σ=sigmoid function. Tanh=hyperbolic tangent function. x_t=input at timestep t. Adapted from "Understanding LSTM Networks" by Olah, C., 2015, (https://colah.github.io/posts/2015-08-Understanding-LSTMs/) [43]

$$o_t = \sigma\,(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o) \tag{A.42}$$

$$h_t = o_t * tanh\,(C_t) \tag{A.43}$$

The new state is passed on to the next timestep. The output generated by the cell, h_t, is controlled by a final gate: the output gate o_t (A.42) (Fig. A.25). The cell state is first mapped between -1 and 1 by a tanh function, after which it is multiplied by the output gate

(A.43). W_o, U_o and b_o represent the weight matrices and bias matrix of the output gate, respectively [41, 43, 59].

The output ht is passed onto the next layer, being another LSTM layer or a dense layer for classification. The dimension of ht is equal to the number of units in the NN layers within the cell. Figure A.26 more detailly visualises the operations within the layers and gates.

In this study, the dense layer was composed of one unit using a sigmoid activation function and computed the following operation [79].

$$\hat{y} = \sigma\left(W_{FC} h_t + b_{FC}\right) \tag{A.44}$$

Where h_t denotes the LSTM layer's output, σ the sigmoid activation function, W_{FC} and b_{FC} the weights and bias of the dense unit, respectively, and $\hat{y}$ the probability of a poor neurological outcome. All model parameters have to be estimated. The estimation of the model parameters, also the learning process, can be divided into six steps, similar to the LR [41, 59]:

1. Construct the model with an initial set of random model parameters
2. Forward propagation: for a batch of samples, predict the outcome based on the sequence of time-fragments of features
3. Compute the loss, which is the difference between the predicted outcomes made by the model and actual outcomes
4. Backpropagation through time (BPTT): unroll the network and calculate the gradients of the loss with respect to the model parameters across all time-fragments
5. Adjust the model parameters to minimise the loss using the gradients of all time-fragments
6. Repeat the process from step 2, iteratively updating the model parameters

The gradients were computed with BPTT through the following operations [83]. The gradient of output at time t, δ, h_t, was calculated using $\triangle_t$ and Δh_t. $\triangle_t$ denotes the output difference at time t, computed with the loss's derivative with respect to h_t. Δh_t denotes the output difference computed at *t+1*, which will be zero for the last time-fragment as no future time-fragments exist.

$$\delta h_t = \triangle_t + \Delta h_t \tag{A.45}$$

$$\delta C_t = \delta out_t * o_t * \left(1 - tanh^2\left(C_t\right)\right) + \delta C_{t+1} * f_{t+1} \tag{A.46}$$

The gradients of the gates at time t:

$$\delta \widetilde{C_t} = \delta C_t * i_t * \left(1 - \widetilde{C_t}^2\right) \tag{A.47}$$

$$\delta i_t = \delta C_t * \widetilde{C_t} * i_t * \left(1 - i_t\right) \tag{A.48}$$

$$\delta f_t = \delta C_t * C_{t-1} * f_t * \left(1 - f_t\right) \tag{A.49}$$

$$\delta o_t = \delta h_t * tanh\left(C_t\right) * o_t * \left(1 - o_t\right) \tag{A.50}$$

The gradients of the input of time t and the output of time *t-1*:

$$\delta x_t = W^T + \delta gates_t \tag{A.51}$$

$$\Delta h_{t-1} = U^T + \delta gates_t \tag{A.52}$$

The model parameters were updated as follows [83]. In this study T=30 time-fragments.

$$\delta W = \sum_{t=0}^{T} \delta gates_t * x_t \tag{A.53}$$

$$\delta U = \sum_{t=0}^{T-1} \delta gates_{t+1} * h_t \tag{A.54}$$

$$\delta b = \sum_{t=0}^{T} \delta gates_{t+1} \tag{A.55}$$

S. LSTM - Hyperparameters

I included the most promising options for hyperparameters in the search space of the random search (Table A.VIII). These options were obtained from Keras' defaults or derived from literature. The hyperparameters and the options are detailly discussed for three separate categories:

1. Hyperparameters to build the model
2. Hyperparameters to compile the model
3. Hyperparameters to fit the model

Building parameters

The definition of several hyperparameters is required to build an LSTM network. Furthermore, the used framework allows the specification of multiple optional hyperparameters. In Keras, LSTMs are built as a sequence of stacked layers. EEG-based classification or regression LSTMs generally compromised one or two LSTM layers, followed by one dense layer [50]. One hyperparameter must be specified, which is the number of units in the layer. Several hyperparameters have a default option used by the framework which can be changed, including the activation function and the initialisation of the model parameters. Furthermore, one can use several regularisation techniques, including regularises, constraints, and dropout.

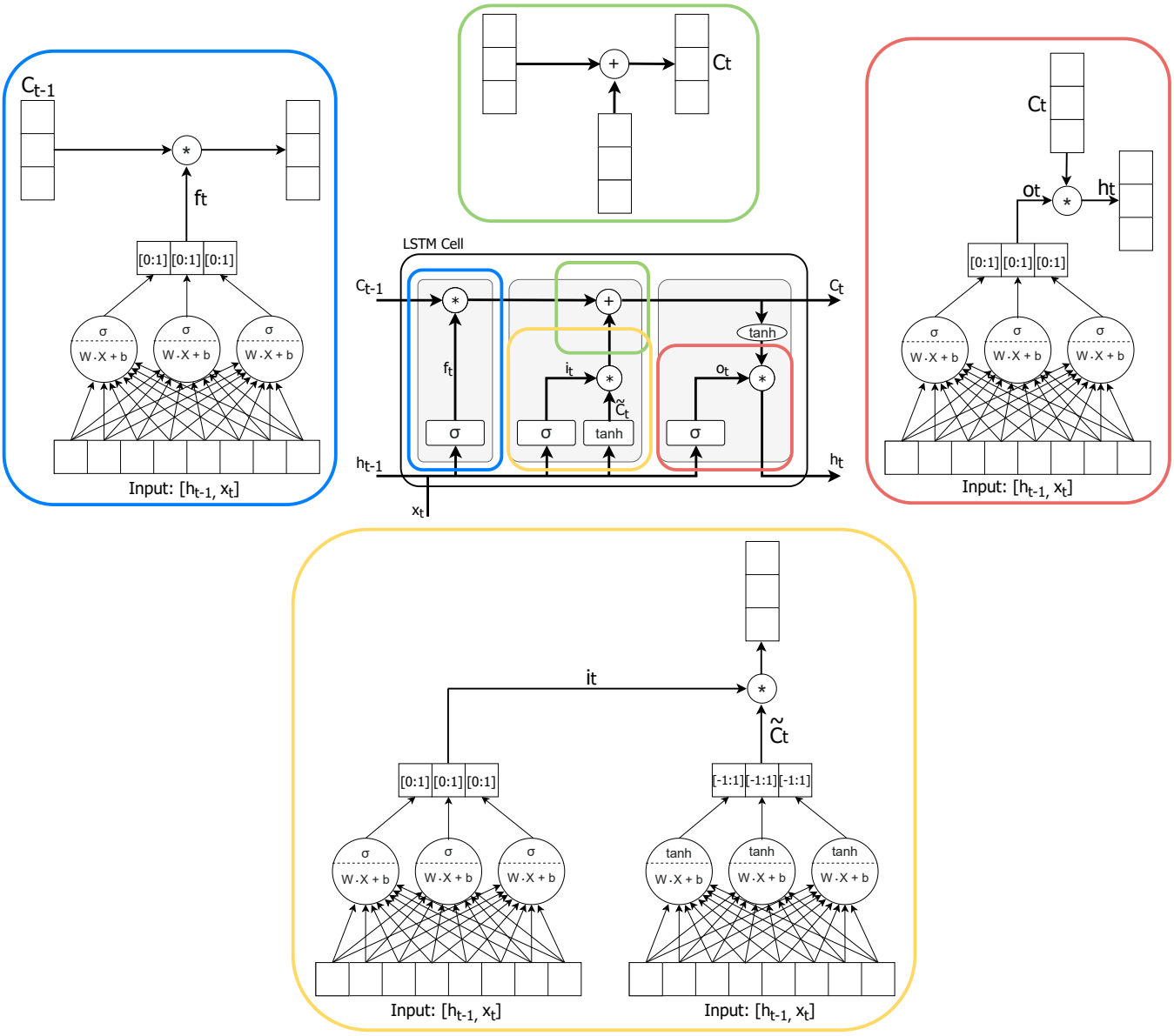


Figure A.26: Visualisation of neural network layers in an LSTM cell. In this example the number of units is 3, resulting in an output dimension on 3. C_t=cell state at time-step t. $\widetilde{C_t}$=candidate state at time-step t. f_t=forget gate at timestep t. h_t=output at time-step t. i_t=input gate at timestep t. o_t=output gate at timestep t. σ=sigmoid function. Tanh=hyperbolic tangent function. x_t=input at timestep t. LSTM=long short-term recurrent neural network.

LSTM related

The number of LSTM layers and units The number of layers and the number of units per layer determines the number of learnable model parameters and thereby the capacity. Too much capacity results in overfitting and requires a lot of computational resources. Not enough capacity leads to underfitting [59, 77, 157].

The optimal number of LSTM layers and cells varies per data and application [44]. These hyperparameters are generally determined empirically, as no formula exists to find the optimal values [58, 59, 158]. As for the number of LSTM layers, most EEG-based classification or regression LSTMs included 1 or 2 [50]. As for the number of units, Bengio found that using an equal number of units for all layers achieved higher or similar performance than using a different number of units for each layer. However, this might be data-dependent [77, 159]. Bengio reported that a first layer with more units than the input vector often worked better than one with units than the input vector. Moreover, he stated that a number of units larger than the optimal number generally does not significantly decrease the generalisation performance [77]. Heaton provided three rules of thumb for obtaining a starting point for the number of units. Furthermore, he suggested trying different num-

bers around these starting points. The three rules are as follows [157]: i) The number of units should be between the input layer's size and the output layer's size. ii) The number of units should be 2/3 the size of the sum of the input and output layer. iii) The number of units should not exceed the size of the two times the input layer.

These rules resulted in the following number of units using the 12 extracted qEEG features: i) between 1 and 12 units, ii) 9 units, and iii) a maximum of 24 units.

The search space included 1 or 2 layers with an equal number of units ranging from 2 to 24, with increments of 2, as these options followed the recommendations of Bengio, Craik et al. and Heaton [50, 77, 157].

Activation function of the LSTM layer(s) Activation functions add nonlinearity to the units that perform a linear operation (Eq. (A.16)), giving the network the ability to learn non-linear relationships between the input and output [41, 58]. In the original design of the LSTM cells [80, 81], the gate layers use a sigmoid activation function (Section 2.7, Eq. 4) and use a hyperbolic tangent (Eq. 40) for the activation of the cell state. As the gates are required to function as a filter mechanism, multiplying values in the (candidate) state vector with a value between 0 and 1, a sigmoid activation function is appropriate. Therefore, I used the original design with a sigmoid as recurrent activation in this study. Some EEG-based LSTM studies changed the tanh function to a rectified linear (ReLu) activation function (e.g. [37, 86]), as ReLus form the most popular activation function in NN at the moment [41]. ReLu (Fig. A.27) returns the provided input if this is larger than 0; otherwise, it returns 0 [58]. ReLu has the advantage of producing sparse representations, being computationally cheaper than tanh functions and being able to handle the vanishing gradient problem [160, 161]. Although the ReLu seems attractive, several disadvantages exist, especially concerning its use in this study. The sparse representations are the result of the inactivation of specific units whose input values are negative. This behaviour of "killing off" units in the network causes the requirement of twice as many units as a network using the tanh function would need [161]. An increased number of units increases the number of model parameters to be estimated [59]. Consequently, the computational resources needed are increased, which was not desired. Furthermore, the advantage of ReLu being capable of handling vanishing gradient does not have added value with an LSTM network, as LSTMs do not suffer from the vanishing gradient problem [80]. Moreover, the linear design of the ReLu in the positive domain causes unbounded behaviour of the activations. Additional regularisation techniques are required to prevent problems resulting from unbounded behaviour, adding complexity to the model [161]. Additionally, the mean of the ReLu is non-zero as the activation is non-negative. A non-zero mean causes a bias for the next layer, which could cause a bias shift for the following units [160]. A bias shift decreases learning effectiveness and speed [162]. For optimal learning speed, inputs with a non-zero mean should always be avoided [163]. The tanh does not induce a bias shift, as its mean is zero-centred. In my case, the disadvantages of using a ReLu outweigh the advantages. Consequently, I used a tanh activation function and did not change the original LSTM cell design.

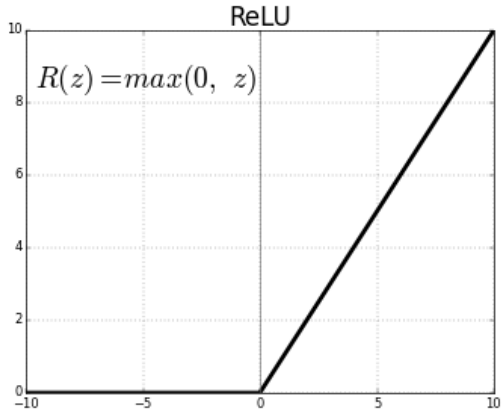


Figure A.27: Rectified Linear (ReLu) activation function. Adapted from "ReLU : Not a Differentiable Function: Why used in Gradient Based Optimisation? and Other Generalisations of ReLU." Sarkar K. 2018 (https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimisation-7fef3a4cecec) [164])

Weight initialisers of the LSTM layer(s) The layer weight initialisers include the kernel initialiser, used for the linear transformation of the inputs, the recurrent initialiser, used for the linear transformation of the recurrent states, and the bias initialiser, used to initialise the biases. As biases are usually initialised to 0 [77], and Keras' default is the zeros class, I also initialised my LSTM model's biases to 0.

The (recurrent) weights must be randomly initialised to small values and not to 0. The small random values are required to break the symmetry between the different units in an NN layer. Otherwise, these units will be updated the same way in every iteration, which will result in no better performance than a linear model [60, 77]. By default, Keras uses the Glorot uniform initialiser for the kernel weights [79]. The Glorot uniform, also termed Xavier initialisation, samples from a random uniform distribution within the following limit.

$$limit = \sqrt{\frac{6}{fan_{in} + fan_{out}}} \tag{A.56}$$

Where fan_{in} and fan_{out} refer to the number of input units in the layer's weight tensor and the number of outgoing units from that layer respectively. The limit results in a constant variance of activations and gradients

through the network [141]. As the sigmoid and tanh activation functions require a constant variance of input, the Xavier initialisation is often used in combination with these activations [147]. Therefore, I used the default Xavier initialiser for the kernel initialiser in the LSTM.

The gradients of recurrent weights in an LSTM are propagated over many timesteps, forming a deep network resulting from the temporal component. The initialiser of the recurrent weights is by Keras' default orthogonal. Assumably, this default was chosen because orthogonal initialisation provides reliable propagation of the gradients in deep non-linear networks [165]. I used the default, as I did not find contradictory literature about using orthogonal recurrent initialisers for LSTMs.

Dense related

Dense layer A dense layer followed the LSTM layer(s). Dense layers are used to convert the output from the LSTM cells to the predicted output. In general, for binary classification of a NN, it is appropriate to include a final dense layer composed of one unit with a sigmoid activation [58, 59, 77]. I implemented that structure, as it results in a one-dimensional probability, corresponding to the required output of this study (the probability of poor outcome).

Weight initialisers of the dense layer The layer weight initialisers for the dense layer include the kernel initialiser and the bias initialiser. As a result of Keras' default, the kernel initialises using the Glorot uniform initialiser, and the bias initialises to 0. The same arguments for using these defaults stated in the LSTM layer's weight initialisers apply for the dense layer. Consequently, the kernel was initialised using Glorot uniform and the bias was initialised with the zeros class.

Regularisation related

There is always the danger that the model overfits, which significantly decreases its performance on unseen data. The best solution to prevent overfitting is to train the model with more data [59, 77, 157]. However, training with more data is impossible due to the limited dataset available for this study. Other methods exist to decrease overfitting and increase generalisation ability. First of all, the model should not have the amount of capacity to be able to overfit on the training data. As previously stated, the number of layers and the number of units per layer determine the capacity. An appropriate number of layers and units were carefully determined using automated random search to reduce overfitting risk due to too much capacity. Another solution to overfitting is reducing the model's complexity by forcing the model parameters to be small [59]. Smaller values for the model parameters, result in a more regular distribution of model parameter values and a more stable model. Hence, methods that increase generalisation are termed regularisation methods [60]. Various regularisation methods are available. The most often and most effective regularisation technique in NN is dropout [59, 77, 166]. Finally, a batch normalisation (BN) layer can be included in the model's architecture, which provides a regularisation effect. However, BN mostly addresses learning speed [162].

Dropout Dropout is a regularisation technique to reduce overfitting, first introduced by Hinton et al., and improved by Srivastava et al. [167, 168]. With a predefined probability, also termed the dropout rate, the technique randomly drops units while training. Temporarily dropping specific units, helps to avoid too many co-adaptations between them. These co-adaptations might not generalise well to data the model has never seen before, resulting in overfitting [167]. Although applying dropout in feedforward NNs improves performance [58, 167], Chollet reported that applying dropout in RNN complicates learning rather than improves generalisation [59]. Gal and Ghahramani determined the correct application of dropout in RNN and came to the following conclusions. First of all, the same dropout mask should be used for the inputs of all timesteps. A randomly applied dropout mask at each timestep would hinder the propagation of the loss through time, thereby disrupting learning. Secondly, next to the connections between layer in RNN, the connection between recurrent units should also be regularised with dropout [169]. Therefore, a temporally constant (recurrent) dropout mask was made available for Keras to implement LSTM layers. The (recurrent) dropout rate is another hyperparameter to be determined. The dropout rate is typically set between 0.2 and 0.5 [59, 170]. For all LSTM layers, a dropout and recurrent dropout (with the same mask for all timesteps) were included in the search with a rate 0 (meaning no dropout), 0.2 and 0.5.

Weight regularisers A regulariser aims to push the model parameters to small values, which lead to a less complex model and therefore less chance of overfitting on [59]. The regulariser picks the smallest model parameters that predict the output, thereby suppressing irrelevant components of the model parameters [171]. Weight regularisation applies a penalty on the model parameters of the layer. The penalties are added to the loss function, which is minimised in the optimisation process. Consequently, the model parameters are also minimised [79, 77]. Three types of regularisers are available [59, 77, 79, 166].

- L1 regularisation, of which the penalty is computed by the sum of the absolute values of the weights and/or biases. L1 forces the model param-

eters that are not contributing to 0, encouraging a sparse model.

- L2 regularisation, of which the penalty is computed by the sum of the squared values of the weights and/or biases. L2 penalises larger model parameters more strongly but does not push them to exactly 0.
- L1_L2 regularisation, which applies both L1 and L2 penalties.

All penalties are multiplied with a regularisation factor to determine in what degree the penalty should influence the optimisation process. The factor has a value between 0 and 1, where higher factors cause a larger penalty. The regularisers can be applied to the layer's model parameters separately, including the model weights, the recurrent weights or the bias. Only penalising the biases is discouraged, as it might be compensated by the weights [77]. Furthermore, a regulariser can be applied to the layer's output, also termed activation, which encourages units to be sparse and give small outputs [79]. As I already aimed to limit overfitting by carefully choosing the number of LSTM layers and units and applying dropout and the limited computational resources, regularisers were not included in the original search. However, weight regularisation was manually explored with the best hyperparameter configuration obtained from the search, to evaluate if higher performance was achieved with L1, L2 or L1_L2 regularisation.

Weight constraints Weight regularisers apply penalties that encourage small model parameters [59], but they do not force small model parameters. Weight constraints can be applied to the model parameters during training to force the model parameters to be within a specific range [79]. If a gradient update violates a constraint on the model parameters, the model parameters are rescaled. Therefore, weight constraints are more aggressive and do not allow model parameters to become very large, regardless of the gradient update size [168]. Constraints can be applied on the weights and the biases. However, they are typically not applied on the bias [172]. Four types of constraints are available in Keras [79]:

1. Max-norm, which forces the model parameters to be a value less than or equal to the defined limit.
2. Min-max norm, which forces the model parameters to be between the defined lower and upper limit.
3. Non-negative norm, which forces the model parameters to be non-negative.
4. Unit-Norm, which forces the model parameters to be of a magnitude of 1.

I did not apply weight constraints in the search for the same reason as the weight regularisers were not applied. I already aimed to limit overfitting by carefully choosing the number of LSTM layers and units and applying dropout. Furthermore, the computational resources available were limited.

Batch normalisation A BN layer can be built in the architecture of the model. BN was introduced by Ioffe and Szegedy to reduce the internal covariate shift that complicates training deep NN, which is the shift in the distribution of the inputs to each layer due to the change in parameters of previous layers [162]. The internal covariate shift causes decreased learning speed and effectiveness, as is requires low learning rates and careful initialisation of parameters. BN normalises the layer inputs, which removes the influence of changing parameters of one layer on the following layers. In feedforward NNs, BN proved to accelerate learning, by allowing larger learning rates, and generalise better, by offering a form of regularisation of the model [162]. BN was applied by Laurent and Pereyra in RNN to the hidden-to-hidden transitions, referring to h_{t-1}, and the input-to-hidden transitions, referring to xt. They concluded that the hidden-to-hidden BN did not improve the training process. Furthermore, they observed that input-to-hidden transitions, which only influence the connections between layers and not the connections between timesteps within the recurrent layer, did not improve the generalisation performance [173]. Cooijmans et al. reported that BN, after careful initialisation of the BN parameters, of hidden-to-hidden transitions in LSTMs does improve training speed and generalisation [174]. However, it is not possible in Keras to apply BN between recurrent LSTM cells; it can only be applied between stacked layers. As BN between stacked layers did not improve the model's performance [173], BN was not included in the LSTM in this study.

Compiling hyperparameters

After the network was built, it was compiled. Compilation computes the matrix transformations of the defined network for the use of the processor. Compilation requires a specification of the loss function to evaluate the network and the optimiser algorithm. Furthermore, additional metrics can be specified, which are calculated while the model is fitted.

Loss function Typically, a binary cross-entropy loss function (Eq. (A.24)) is used in NN for binary classification [58, 59, 77]. Therefore, I used a binary cross-entropy loss function to compute the loss in this model.

Optimiser An optimiser algorithm uses the gradients, computed by the backpropagation through time, to

iteratively change the model parameters to decrease the loss [41]. Mini-batch stochastic gradient descent (SGD) and the variant using momentum or Nesterov accelerated gradient are explained in Appendix P. Until the last few years, mini-batch SGD formed the golden standard [77]. Especially when combined with (Nesterov) momentum, SGD is less impacted by the initially set learning rate and can better escape from suboptimal local minima [142, 143]. However, SGD (with (Nesterov) momentum) applies the same learning rate to all model parameters whilst updating. In the last decade, optimiser algorithms that adapt the learning rate to each individual model parameter whilst updating were developed and gained popularity. Therefore, these algorithms can update individual model parameters according to their importance [168, 175, 176, 177, 178].

Adagrad was the first developed algorithm capable of individually updating model parameters [176]. Adagrad updates model parameters that were infrequently updated with a larger amount than the model parameters that were frequently updated during training. Adagrad updates the model parameters by multiplying the current gradient with a learning rate scaled by the square root of the sum of the squared gradients from all previous timesteps. As this scaling of the learning rate determines the model parameters' updates, the impact of the initial learning rate on the learning process is significantly less than with SGD [176]. The disadvantage of Adagrad is the increasingly larger square root with which the learning rate is divided. The square root grows with every iteration, as the squared gradient from the previous iteration is added. Consequently, the scaled learning rate becomes increasingly smaller throughout the training. When this update vector decreases to an infinitely small number, the algorithm cannot learn anymore.

An extension of Adagrad was developed, namely Adadelta, to address this problem [177]. Adadelta does not divide the learning rate by a square root of the accumulation of the previously squared gradients, but by the square root of the decaying average of all previously squared gradients. The latter does not cause a continual decay of the scaled learning rate. The square root of the decaying average of all previously squared gradients is equal to the root mean squared (RMS) error criterion of the gradient. Furthermore, the learning rate is replaced by the square root of the decaying average of all previous squared parameter updates, eliminating the requirement of selecting an initial learning rate [177].

The RMSprop algorithm was also developed to address the problem of the increasingly smaller-scaled learning rate. RMSprop is similar to Adadelta; it divides the learning rate by the RMS error criterion of the gradient [168].

The following SGD-based algorithm is basically RMSprop with momentum: Adaptive Moment Estimation (Adam) [175]. Additional to calculating the decaying average of past squared gradients, Adam calculates the decaying average of past gradients themself, comparable to momentum. The decaying average of past squared gradients and the gradients are an estimation of the first-order moment and the second-order moment, respectively. The moment's estimates are corrected from occurring biases with two parameters included in the update rule. The authors also developed Adamax, which is slightly different from Adam and might result in better performance in models with embeddings [175]. The newest development is essentially Adam with Nesterov momentum, termed Nadam [178]. In Nadam, the bias-corrected estimate of the first-order moment from the previous timestep used in Adam is replaced by this estimate of the current timestep, providing the anticipatory behaviour [178]. The adaptive SGD-based algorithms are extensions or improvements of older versions. Nadam was developed most recently and showed increased performance over Adam, according to the author [178]. However, Adam is still the most popular and widely used optimiser over the past few years and generally used as default choice [44, 146, 150, 152, 179]. Furthermore, none of the evaluated EEG-based LSTM models used the Nadam optimiser, whereas the majority used Adam [37, 39, 40, 44, 45, 46]. Therefore, considering the limited computational resources available in this study, I choose to use Adam optimiser and not include other optimisers in the search. As Adam incorporates momentum and RMSprop, it inherits both algorithms' benefits: the momentum gives the appropriate direction for the descent and RMSprop adapts the degree in which the model updates its individual parameters in that direction [175]. According to the authors, Adam significantly outperforms the older adaptive algorithms, is computationally efficient, and achieves good performance for most problems, including ones with a lot of data or parameters, non-stationary objectives, noise and sparse gradients. Moreover, they state that the hyperparameters within the optimiser often need only little tuning [175], confirmed by the Deep Learning book by Goodfellow et al. [60]. The Adam optimiser updates the model parameters using the following equations.

$$\theta_{t+1} = \theta_t - \frac{\propto}{\sqrt{\widehat{v_t}} + \epsilon} \widehat{m_t} \tag{A.57}$$

$$\widehat{m_t} = \frac{\beta_1 m_{t-1} + (1 - \beta_1)\, g_t}{1 + \beta_1^t} \tag{A.58}$$

$$\widehat{v_t} = \frac{\beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2}{1 + \beta_1^t} \tag{A.59}$$

Where t denotes the timestep, θ all the model parameters, $\propto$ the learning rate, $\hat{m}$, the bias-corrected first-moment estimate, m, the first-moment estimate, which is the decaying average of past gradients, $\hat{v}$ the bias-corrected second-moment estimate, v, decaying average of past squared gradients, β_1 and β_2 the exponential decay rates for the first and second moment estimates respectively, and g the gradients of the loss function with

respect to the model parameters. The defaults suggested by the paper are 0.001 for the learning rate, 0.9 and 0.999 for β_1 and β_2, respectively [175]. As Adam is robust to these hyperparameters' choice, it is recommended to use default values for the decay rates. However, a better performing learning rate than default might be found [60]. Therefore, I used the default values for the decay rates and included several options for the learning rate in the search. Bengio stressed that the learning rate should always be tuned and recommended a rate smaller than 1 and larger than 1e-6 for NNs with standardised input. Moreover, he recommended sampling in the logarithmic domain as values in the linear domain are likely to give similar results [77]. I included a learning rate with the default value and a factor 10 larger and smaller than the default in the search space. Additionally, I included intermediate values as the differences between the learning rates are quite large. A learning rate search space of the following values was obtained: $1e^{-4}$, $5e^{-4}$, $1e^{-3}$, $5e^{-3}$, $1e^{-2}$.

Metrics Additional to the loss, the metrics discussed in Section 2.6 were computed while fitting the model: the AUC, the sensitivity at 100% specificity of poor neurological outcome prediction, and the sensitivity at 95% specificity of good neurological outcome prediction.

Fitting hyperparameters

Fitting the model refers to adjusting the model parameters to the training data. Fitting requires the specification of a training dataset, which in my case includes a feature matrix of [samples, time-fragments, features], and an output matrix with neurological outcomes per sample. Backpropagation through time is applied for a specified number of epochs. The number of epochs forms one of the hyperparameters in fitting the network. The optimiser adjusts the model parameters per batch. The batch size is another hyperparameter [41, 58, 59].

Number of epochs The number of epochs determines how many times the model parameters are adjusted based on the loss computed. Too few epochs cause underfitting, too many cause overfitting [58, 59]. The number of epochs should compromise between underfitting and overfitting, where the chance of either one occurring is limited. Several runs of 10-fold CV determined the range of options for the number of epochs. The options ranged from 20 to 200 epochs with intervals of 20 to precisely determine the right number between under- and overfitting.

Batch size The batch size determines the number of samples used to compute the gradient of the loss function and update the model parameters accordingly [144]. When more samples are used to update the model parameters, these are likely updated in the appropriate direction. The appropriate updates initially cause the model's loss to converge with large steps. With fewer samples, the gradient's computation will be noisy, as it is relatively sensitive to outliers in the batch. Consequently, smaller batches cause noisy model parameter updates. These noise updates lead to slower convergence but offer an improved generalisation, and therefore often better performance [77, 144, 145]. Keras uses a default batch size of 32. Bengio and Masters and Luschi agree that 32 is an appropriate batch size, which generally results in stable training and adequate regularisation [77, 145]. Batch size does not influence the generalisation ability very much but affects training time [77]. Considering the limited effect on performance, the limited available computational resources and the confirmed good default value of 32, I used a batch size of 32 for my LSTM network.

T. LSTM – Hyperparameter optimisation

Talos was used to perform the random search [180]. The random searches were done separately for a model with one-layer LSTM and two-layer LSTM, as differencing between one or two layers within a search could not be done with Talos. The method for both models was identical and was as follows. The random method used was "uniform mersenne" by Talos' default. A random 25% of the full hyperparameter space was searched. I chose to use 25% of the configurations as this number of permutations (approximately 1300) could be explored within 1 or 1.5 days for the model using one LSTM layer and two LSTM layers, respectively. Moreover, as the random search was repeated ten times, one could assume that most of the possible configurations would be covered (repeatedly), eliminating the necessity for a more extensive search space within the individual runs. A probabilistic reduction was used to remove configurations that were likely to give a validation AUC lower than 0.80 to speed up the search. The reducer stopped between every 20 permutations (by default) and evaluated the correlation between the validation AUC and the hyperparameters of the previous 50 permutations (by default) to remove the hyperparameters that were likely to give a validation AUC lower than 0.80.

After completing the searches, the correlation between the hyperparameters and the validation AUC and validation SeSp100 for poor outcome prediction of the individual searches were compared. A robust model should have approximately the same correlation between the hyperparameters and metrics in every search. Subsequently, the permutations of all ten searches were combined. For all hyperparameters that had a notable correlation with the performance metrics, boxplots of the validation SeSp100 and validation AUC for poor outcome prediction were created per hyperparameter option. These boxplots were judged based on the values

Table A.8: LSTM hyperparameters. Continued next page.

Category	Hyperparameter	Option(s) used	Reason behind the implemented option(s)	Ref.
Build	LSTM: Number of layers	1, 2	Determines the number of learnable model parameters, defining the capacity of the network. Too few learnable model parameters result in underfitting, too many in overfitting. EEG-based LSTM models are typically composed of 1 or 2 layers. Using references from literature and rules of thumb, a range of possible number of units was defined.	[44] [58] [59] [77] [79] [157] [158]
	LSTM: Number of units in the layer	2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24		
	LSTM: Activation function	Tanh (default)	Gives the NN the ability to learn non-linear relationships between the input and output. Original design of the LSTM cell used tanh activation and sigmoid recurrent activation. No advantage was found to change the activation functions of the original design.	[41] [58] [79] [80] [162] [160] [163] [161]
	LSTM: Recurrent activation function	Sigmoid (default)		
	LSTM: Kernel initialiser	Glorot uniform (default)	Weights are required to be randomly initialised to small values to break symmetry during learning. As the sigmoid activation function was used, the Glorot uniform initialiser was the most appropriate for the kernel. The orthogonal initialiser was the best choice for the recurrent weights, as it offers reliable propagation over many time-fragments. As biases are typically initialised to 0, this is also done in this study.	[60] [79] [134] [141] [165]
	LSTM: Recurrent kernel initialiser	Orthogonal (default)		
	LSTM: Bias initialiser	Zeros (default)		
	Dense: Number of layers	1	Dense layers are used to convert the output from the LSTM cells to the predicted output. Typically, a final dense layer with 1 unit using a sigmoid activation function is used for binary classification of a NN, as this outputs a one-dimensional probability. This typical architecture was implemented, as the required output was the probability of poor neurological outcome.	[77] [58] [59]
	Dense: Number of units in the layer	1		
	Dense: Activation function	Sigmoid		
	Dense: Kernel initialiser	Glorot uniform (default)	Weights are required to be randomly initialised to small values to break symmetry during learning. As the sigmoid activation function was used, the glorot uniform initialiser was the most appropriate for the kernel. As biases are typically initialised to 0, this is also done in this study.	[60] [77] [79] [141]
	Dense: Bias initialiser	Zeros (default)		
	Dropout	0, 0.2, 0.5	A regularisation technique that limits overfitting by preventing too much co-adaptations between (recurrent) units. The technique drops units from training. The probability a unit is dropped, is determined by the dropout rate. Typically, the dropout rate is set in the range between 0.2 and 0.5, if applied at all.	[58] [59] [167] [168] [169] [170]
	Recurrent dropout	0, 0.2, 0.5		
	Batch normalisation	Not included	Accelerates learning and offers better generalisation by normalising the layer's input in feedforward NN and normalising both input and recurrent inputs in RNN. Keras does not offer batch normalisation of recurrent cells, so batch normalisation could not be applied.	[162] [173] [174]
	Kernel regulariser	Not included	A regularisation technique by encouraging small model parameters. As overfitting was battled with determining the best number of layers and units, an with applying dropout, and the computational resources available were limited, regularisers were not included in the search.	[59] [77] [79] [166] [171]
	Recurrent regulariser			
	Bias regulariser			
	Activity regulariser			
	Kernel constraint	Not included	A regularisation technique by forcing small model parameters. As overfitting was battled with determining the best number of layers and units, and with applying dropout, and the computational resources available were limited, constraints were not included in the search.	[79] [168] [172]
	Recurrent constraint			
	Bias constraint			

Table A.VIII: LSTM hyperparameters. AUC=area under the receiver operating curve. EEG= electroencephalogram. LSTM=long short-term memory recurrent neural network. NN=neural network. SeSp100=sensitivity at 100% specificity (for poor outcome prediction). SeSp95=sensitivity at 95% specificity (for good outcome prediction). SGD=stochastic gradient descent.

Category	**Hyperparameter**	**Option(s) used**	**Reason behind the implemented option(s)**	**Ref.**
Compile	Loss function	Binary cross-entropy	Typically, a binary cross-entropy loss function is used in NN for binary classification.	[58, 59, 77]
	Optimiser	Adam	To date, Adam, a gradient descent based adaptive algorithm, is the most popular and widely used optimiser in NN.	[44, 59, 152, 154, 175, 179]
	Learning rate	0.0001, 0.0005, 0.001, 0.005, 0.01	Determines the degree with which the model parameters are updated. Adam's default is 0.001, which might be suboptimal and should therefore be tuned. As is recommended, options were sampled in the logarithmic domain. Intermediate values were included.	[60, 77, 175, 179]
	Metrics	AUC, SeSp100, SeSp95	See Section 2.6	
Fit	Number of epochs	20, 40, 60, 80, 100, 120, 140, 160, 180, 200	Number of times the complete dataset is passed through the model once. Too few epochs result in underfitting, too many in overfitting. The search space was decided based on several initial runs.	[58, 59]
	Batch size	32 (default)	Determines the number of samples used to compute the gradient of the loss function and update the model parameters accordingly. It does not significantly influence performance. A default of 32 is confirmed to be appropriate and was thus used.	[77, 144, 145]

of the SeSp100 and AUC and on their distribution. A small distribution is preferred as it indicates stable performance, reflected in the plots by small boxes, short whiskers and little outliers. The options scoring low in SeSp100 and AUC or resulting in a large distribution were removed from the merged search results. Consequently, all permutations with poorly performing hyperparameter options were eliminated. Again, with the remaining permutations, boxplots were created of the validation SeSp100 and validation AUC for poor outcome prediction per hyperparameter option. These boxplots were evaluated similarly as described above. The option with the highest validation SeSp100 for poor outcome prediction was selected, under the condition that this option did not result in a notably reduced AUC compared to the other options. A model using the best performing option per hyperparameter was trained using 10-fold CV (Fig. A.16A). A 10% split for validation data was used to check if underfitting or overfitting occurred (Fig. A.16B). Furthermore, this was manually tuned with weight regularisation to evaluate if the performance could be further enhanced. The best performing models with one LSTM layer and two LSTM layers were compared, and the best one was chosen as final model.

U. Results Logistic Regression – Hyperparameter optimisation

Results for a learning rate of 0.01

Figure A.28 shows the models' distribution over the five learning behaviour groups per specified number of epochs (Appendix Q, Fig. A.18). Figure A.29 visualises the number of epochs that satisfied the learning behaviour criterion. For these number of epochs, bar plots of the mean and standard deviation of the AUC and SeSp100 for poor outcome prediction on the test set are shown (Fig. A.30). Table A.IX and Table A.X show the number of epochs' robust performance scores with the highest mean test AUC and SeSp100, respectively. Table A.XI shows the top three number of epochs with which the models were trained that obtained the best robust performance score on both metrics. Training the model with 1500 epochs reached the highest robust performance score for the AUC and SeSp100. Consequently, the optimal number of epochs for a learning rate of 0.01 was set to 1500.

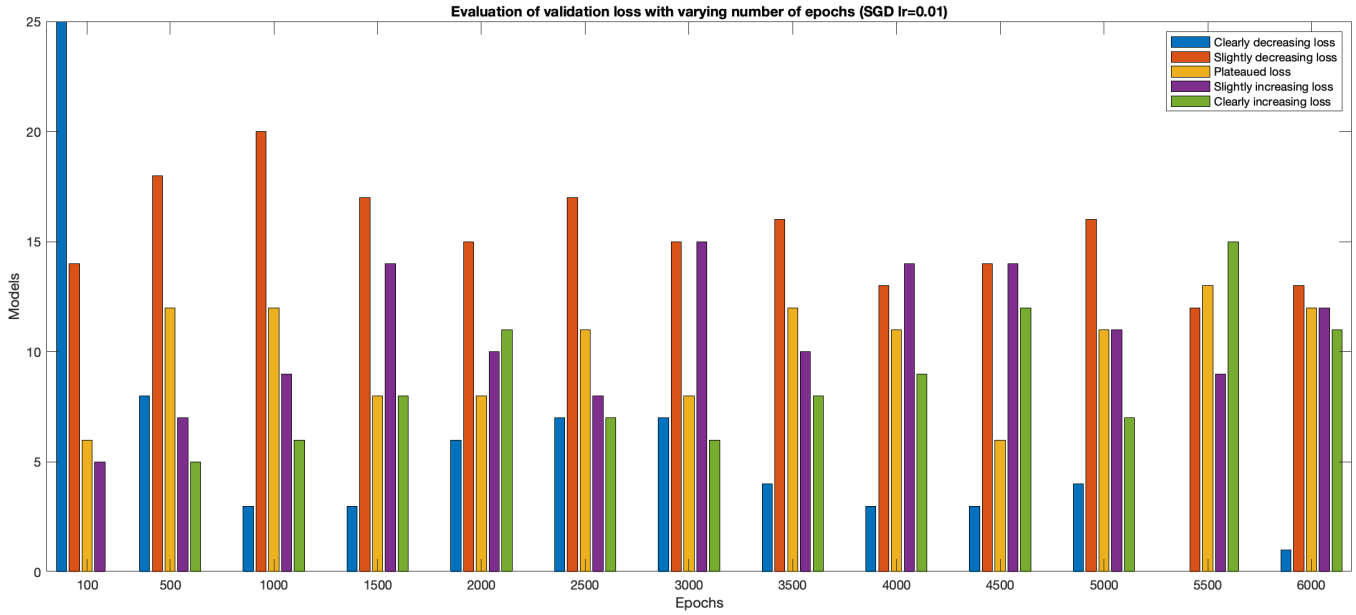


Figure A.28: Distribution of the models over the five learning behaviour groups per specified number of epochs (learning rate=0.01). Lr=learning rate. SGD=stochastic gradient descent.

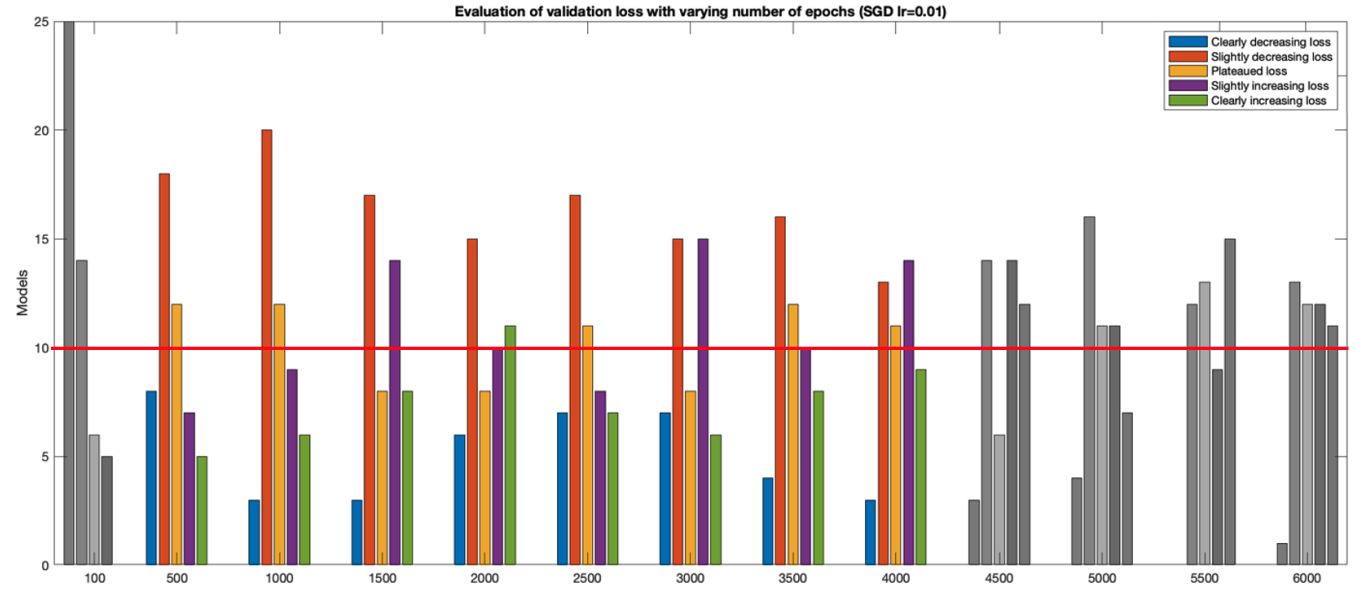


Figure A.29: Visualisation of the number of epochs that satisfied the learning stability criterion (shown in colour) (learning rate=0.01). Lr=learning rate. SGD=stochastic gradient descent.

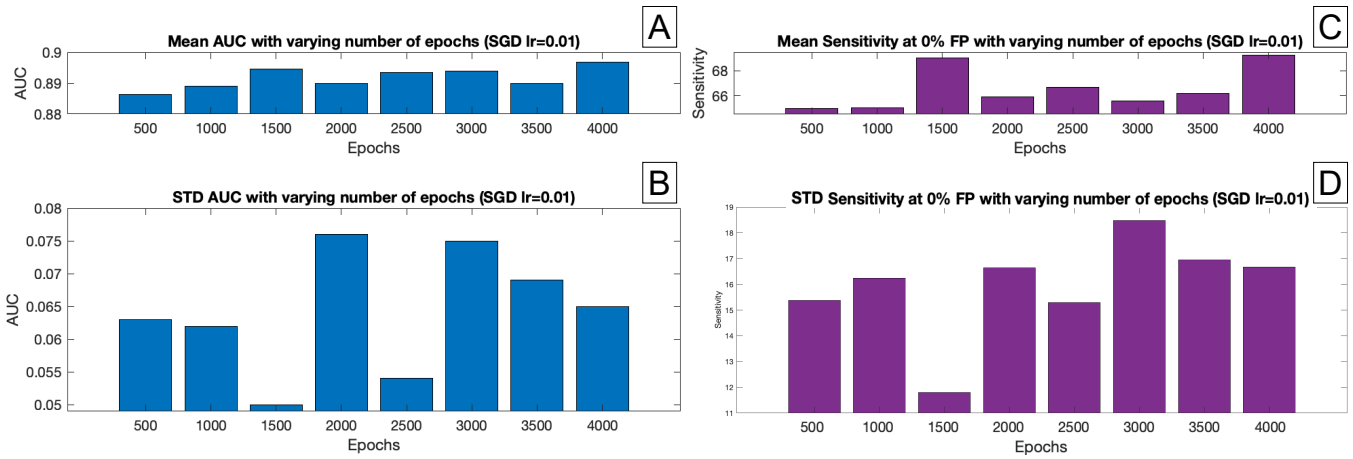


Figure A.30: Bar graphs of the mean and standard deviation of the AUC (A, B) and SeSp100 (C, D) on the test set per number of epochs, concerning the SGD with a learning rate of 0.01. AUC= area under the receiver operating curve. FP=false positives. Lr=learning rate. SeSp100=sensitivity at 100% specificity for poor outcome prediction. SGD=stochastic gradient descent. STD=standard deviation.

Table A.IX: Mean, standard deviation and robust performance scores of the number of epochs with the highest mean test AUC (learning rate=0.01). AUC=area under the receiver operating curve. std=standard deviation.

Number of epochs	**1500**	**2500**	**3000**	**4000**
Mean	0.8946	0.8934	0.8939	0.8968
Standard deviation	0.050	0.054	0.075	0.065
Robust performance score (= mean/std)	17.89	16.54	11.92	13.80

Table A.X: Mean, standard deviation and robust performance scores of the number of epochs with the highest mean test SeSp100 for poor outcome prediction (learning rate=0.01). SeSp100=sensitivity at 100% specificity. std=standard deviation.

Number of epochs	**1500**	**2500**	**3500**	**4000**
Mean	69.03	66.69	66.12	69.28
Standard deviation	11.80	15.30	16.95	16.67
Robust performance score (= mean/std)	5.85	4.36	3.90	4.16

Table A.XI: Top three robust performance scores of the models trained for a specific number of epochs for both metrics (learning rate=0.01). AUC=area under receiver operating curve. SeSp100=sensitivity at 100% specificity.

Top 3 highest robust performance scores	**AUC**	**SeSp100**
1	1500	1500
2	2500	2500
3	4000	4000

Results for a learning rate of 0.1

Figure A.31 shows the models' distribution over the five learning behaviour groups per specified number of epochs. Figure A.32 visualises which number of epochs satisfied the learning behaviour criterion. Furthermore, bar plots of the mean and standard deviation of the AUC and SeSp100 for poor outcome prediction on the test set are shown (Fig. A.33). Table A.XII and Table A.XIII show the number of epochs' robust performance scores with the highest mean test AUC and SeSp100, respectively. Table A.XIV shows the top three number of epochs with which the models were trained that obtained the best robust performance score on both metrics. Training the model with 2200 epochs reached the highest robust performance score for the AUC and the SeSp100. Consequently, the optimal number of epochs for a learning rate of 0.1 was set to 2200.

Best performing Logistic Regression model

The AUC and SeSp100 for poor outcome prediction of the best performing LR models for both learning rates are stated in Table A.XV. The model with a learning rate of 0.01, trained on 1500 epochs, achieved significantly higher SeSp100 for poor outcome prediction ($p<0.05$ with an unpaired two-sample t-test), while the AUC for both models was not significantly different. Therefore, I proceeded with the LR with a learning rate of 0.01 and 1500 epochs.

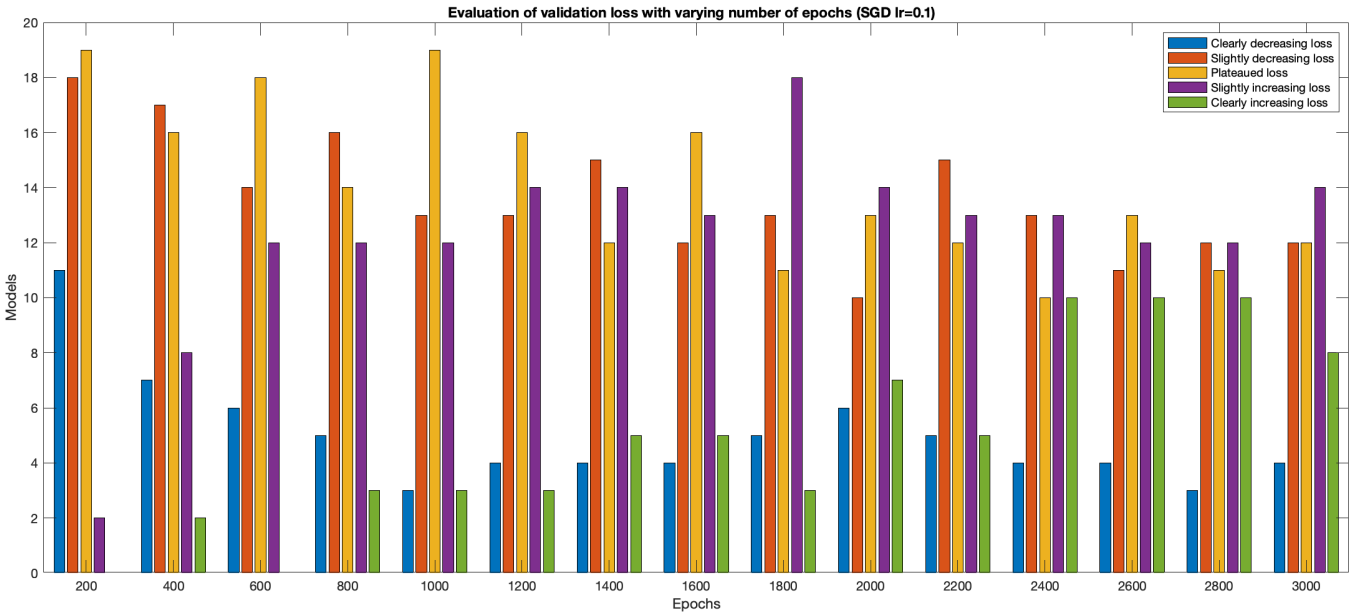


Figure A.31: Distribution of the models over the five learning behaviour groups per specified number of epochs (learning rate=0.1). Lr=learning rate. SGD=stochastic gradient descent.

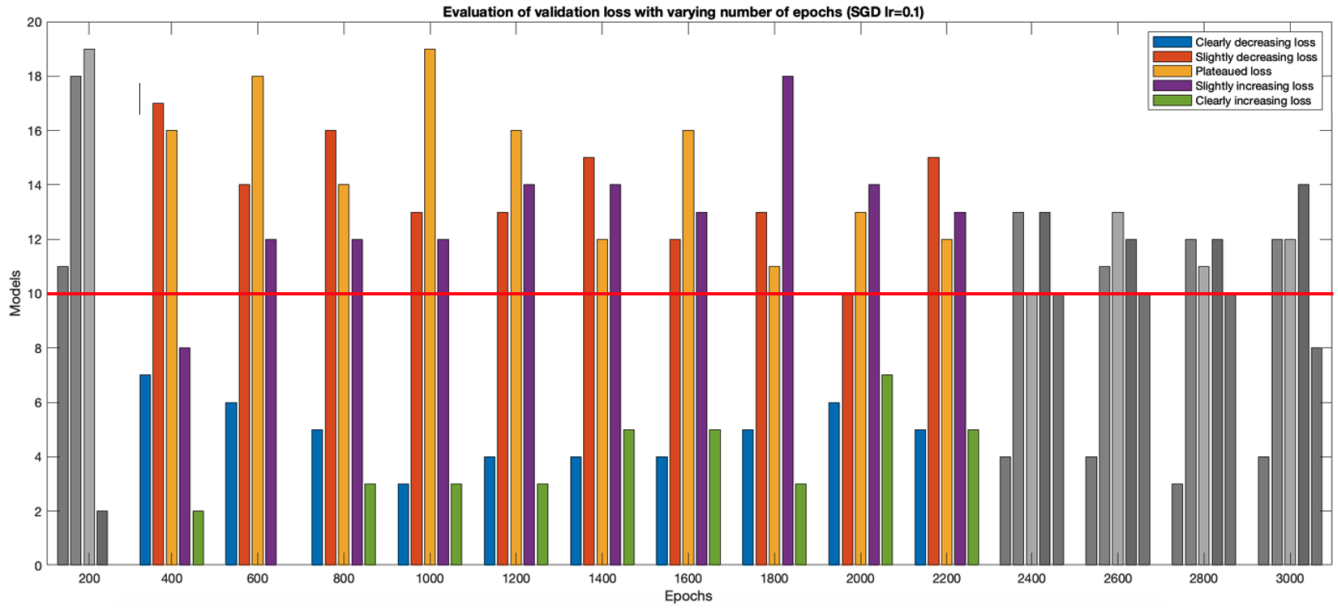


Figure A.32: Visualisation of the number of epochs that satisfied the learning stability criterion (shown in colour) (learning rate=0.1). Lr=learning rate. SGD=stochastic gradient descent.

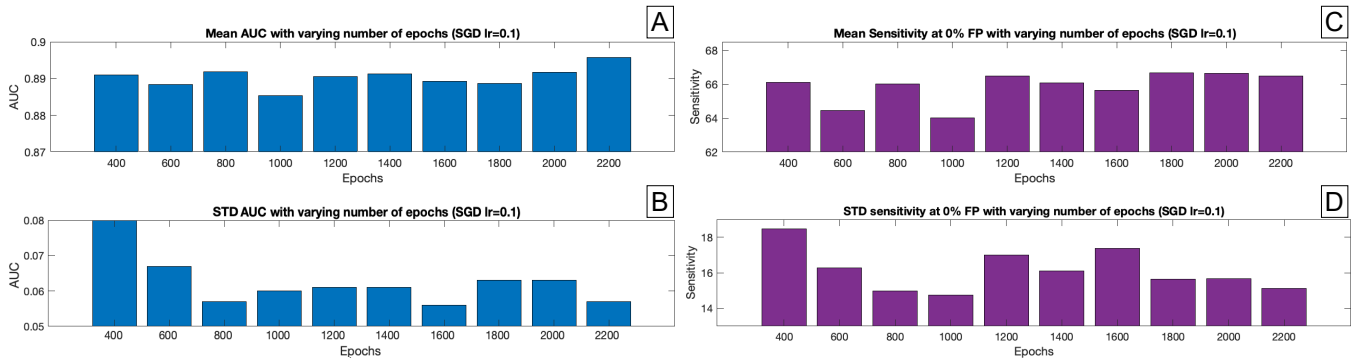


Figure A.33: Bar graphs of the mean and standard deviation of the AUC (A, B) and SeSp100 (C, D) on the test set per number of epochs, concerning the SGD with a learning rate of 0.1. AUC= area under the receiver operating curve. FP=false positives. Lr=learning rate. SeSp100=sensitivity at 100% specificity for poor outcome prediction. SGD=stochastic gradient descent. STD=standard deviation.

Table A.XII: Mean, standard deviation and robust performance scores of the number of epochs with the highest mean test AUC (learning rate=0.1). AUC=area under the receiver operating curve. std=standard deviation.

Number of epochs	**400**	**800**	**1400**	**2000**	**2200**
Mean	0.8910	0.8919	0.8913	0.8917	0.8958
Standard deviation	0.080	0.057	0.061	0.063	0.057
Robust performance score (=mean/std)	11.14	15.65	14.61	14.15	15.72

Table A.XIII: Mean, standard deviation and robust performance scores of the number of epochs with the highest mean test SeSp100 for poor outcome prediction (learning rate=0.1). SeSp100=sensitivity at 100% specificity. std=standard deviation.

Number of epochs	**1200**	**1800**	**2000**	**2200**
Mean	66.48	66.67	66.64	66.47
Standard deviation	17.00	15.65	15.67	15.12
Robust performance score (= mean/std)	3.91	4.26	4.25	4.40

Table A.XIV: Top three robust performance scores of the models trained for a specific number of epochs for both metrics (learning rate=0.1). AUC=area under receiver operating curve. SeSp100=sensitivity at 100% specificity.

Top 3 highest robust performance scores	**AUC**	**SeSp100**
1	2200	2200
2	800	1800
3	1400	2000

Table A.XV: Best performing logistic regression models with a learning rate of 0.01 and 0.1. AUC=area under the receiver operating curve. SeSp100=sensitivity at 100% specificity. std=standard deviation.

Learning rate	**Number of epochs**	**AUC (mean (std))**	**SeSp100 (mean (std))**
0.01	1500	81.26 (6.29)	69.03 (11.80)
0.1	2200	81.45 (6.65)	66.47 (15.12)

Table A.XVI: Comparison of the performance metrics on the test sets of the LR separately trained and evaluated at both 12 and 24, 12 hours, and 24 hours after cardiac arrest. The models were trained and evaluated with features from the final feature set. AUC=area under the curve, SeSp100=sensitivity at 100% specificity, SeSp95=sensitivity at 95% specificity.

	AUC Mean (95% CI)	**SeSp100 Mean (95% CI)**	**SeSp95 Mean (95% CI)**
LR – 12 h + 24 h	0.893 (0.887-0.898)	0.669 (0.655-0.683)	0.599 (0.576-0.621)
LR – 12 h	0.901 (0.890-0.911)	0.783 (0.763-0.803)	0.295 (0.266-0.323)
LR – 24 h	0.883 (0.877-0.890)	0.671 (0.655-0.687)	0.557 (0.532-0.581)

Table A.XVII: Comparison of the performance on the test set of the LR trained and evaluated with features from the FFS and AFSs using all epochs on both timepoints. *the difference of the performance metric was statistically significant from the performance metric obtained by using features from the final feature set. AUC=area under the receiver operating curve. AFS=additional feature set. FFS=final feature set. SeSp100=sensitivity at 100% specificity. SeSp95=sensitivity at 95% specificity.

	AUC Mean (95% CI)	**SeSp100 Mean (95% CI)**	**SeSp95 Mean (95% CI)**
FFS	0.893 (0.887-0.898)	0.669 (0.655-0.683)	0.599 (0.576-0.621)
AFS 1	0.893 (0.888-0.899)	0.659 (0.646-0.673)	0.585 (0.563-0.607)
AFS 5	0.862 (0.855-0.868)*	0.652 (0.636-0.668)	0.713 (0.692-0.734)*

V. Results Logistic Regression – Final model performances

Models trained at different timepoints

Table A.XVI compares the LR performance metrics separately trained and evaluated at both timepoints (12 and 24 hours), 12 hours, and 24 hours after CA. The models were trained and evaluated with features from the final feature set (Table A.IV). The AUCs were not significantly different from each other. The AUC at both timepoints was 0.893, at 12 hours it was 0.901, and at 24 it was 0.883. The SeSp100 at 12 hours after CA of 0.783 was significantly higher than at 24 hours (0.671) or both timepoints (0.669). On the contrary, the SeSp95 at 12 hours after CA of 0.295 was significantly lower than at 24 hours (0.557) or both timepoints (0.599). The performance of the model trained at 24 hours was not significantly different from the performance at both timepoints after CA.

Models trained with different feature sets

Table A.XVII compares the LR's performance metrics and evaluated on the FFS and the AFSs using all epochs at both timepoints. Only AFSs that included features with low multicollinearity were used (Table A.V).

The performance of AFS 1, which included the clinical features age and sex additional to the qEEG features in FFS, did not significantly differ from the FFS. Therefore, adding clinical features to the FFS did not result in significantly different performance. The AUC of AFS 5 (0.862) was significantly lower than the one of the FFS (0.893), whereas the SeSp95 was significantly higher for AFS 5 (0.713) than for the FFS (0.599). Therefore, excluding features that showed little discriminative power between a poor and good outcome in a boxplot significantly increased the sensitivity for good outcome prediction but at the cost of the AUC. As the SeSp100 and AUC were considered more important than SeSp95, AFS 5 was not considered to outperform the FFS.

W. Results LSTM – Hyperparameter optimisation

Results for a one-layer LSTM

The correlation between the hyperparameters and the validation AUC and SeSp100 of poor outcome prediction of all random searches is stated in Table A.XVIII and Table A.XIX respectively. The number of epochs, the number of units, and the learning rate showed a notable positive correlation with the validation AUC. Overall, the SeSp100 showed less consistent correlation over all searches than the AUC. However, the mean correlation of the number of units and the learning rate showed a notable positive correlation with the validation SeSp100. Boxplots of the number of units (Fig. A.34), the number of epochs (Fig. A.35), and the learning rate (Fig. A.36) for both metrics were evaluated to make the first selection of well-performing hyperparameter options. The SeSp100 and AUC were higher for i) the number of units equal to or higher than 10 (with the exemption of 14 units), ii) the number of epochs equal to or higher than 120, and iii) the learning rate equal to or higher than 0.001. Therefore, only permutations that used these hyperparameter options were used for further evaluation. Boxplots of validation SeSp100 and AUC that included only these permutations are shown for the number of units (Fig. A.37), the number of epochs (Fig. A.38), the learning rate (Fig. A.39), the dropout rate (Fig. A.40), and the recurrent dropout rate (Fig. A.41). The highest values with the smallest distribution for the validation SeSp100 were achieved by training the model for 120 epochs, using 16 units in the LSTM layer, a learning rate of 0.001, and applying a dropout rate of 0.5 and no recurrent dropout. Moreover, these hyperparameters also resulted in high values with a small distribution for the validation AUC. Therefore, I chose these hyperparameter values for the LSTM model composed of one LSTM layer. Subsequently, I checked if these hyperparameters caused overfitting, and I manually applied weight regularisation to increase the model's generalisation ability. Overfitting was prevented, and the highest validation AUC and SeSp100 of poor outcome prediction were achieved using L1_L2 regularisation of 0.001 on the bias and the recurrent kernel and kernel weights. Plots of the training and validation loss against the number of epochs for the model with and without weight regularisation are shown in Figure 49. Table A.XX summarises the best hyperparameter values for a one-layer LSTM. Table A.XXI states the performance achieved by this model using 50 repetitions of 10-fold CV.

Table A.XVIII: Correlation between the hyperparameters and the validation AUC of all random searches for a model with 1 LSTM layer. AUC= area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. Val=validation.

Correlation with val. AUC	Search 1	Search 2	Search 3	Search 4	Search 5	Search 6	Search 7	Search 8	Search 9	Search 10	Mean of all searches
Dropout rate	0.002	0.021	0.021	-0.011	0.004	-0.012	0.003	0.045	0.012	0.018	**0.01**
Number of epochs	0.125	0.13	0.227	0.212	0.246	0.196	0.159	0.147	0.173	0.201	**0.182**
Number of units	0.266	0.221	0.238	0.26	0.245	0.239	0.209	0.205	0.253	0.226	**0.236**
Learning rate	0.204	0.194	0.23	0.278	0.319	0.24	0.154	0.154	0.225	0.223	**0.222**
Recurrent dropout rate	0.023	0.017	-0.009	-0.002	-0.014	0.006	-0.026	0.053	-0.031	0.034	**0.005**

Table A.XIX: Correlation between the hyperparameters and the validation SeSp100 of poor outcome prediction of all random searches for a model with 1 LSTM layer. LSTM=long short-term memory recurrent neural network. SeSp100 = sensitivity at 100% specificity Val=validation.

Correlation with val. SeSp100	Search 1	Search 2	Search 3	Search 4	Search 5	Search 6	Search 7	Search 8	Search 9	Search 10	Mean of all searches
Dropout rate	-0.033	-0.009	0.096	-0.048	0.109	0.159	0.022	0.19	0.25	0.027	**0.076**
Number of epochs	0.144	0.157	0.057	0.205	0.199	-0.011	0.204	-0.201	-0.122	-0.031	**0.06**
Number of units	0.224	0.263	0.07	0.323	0.23	0.014	0.259	-0.048	0.072	0.009	**0.142**
Learning rate	0.124	0.212	0.126	0.407	0.38	0.038	0.206	-0.204	-0.145	0.011	**0.116**
Recurrent dropout rate	0.035	0.01	-0.06	-0.014	-0.083	-0.116	0.01	-0.026	-0.019	-0.084	**-0.035**

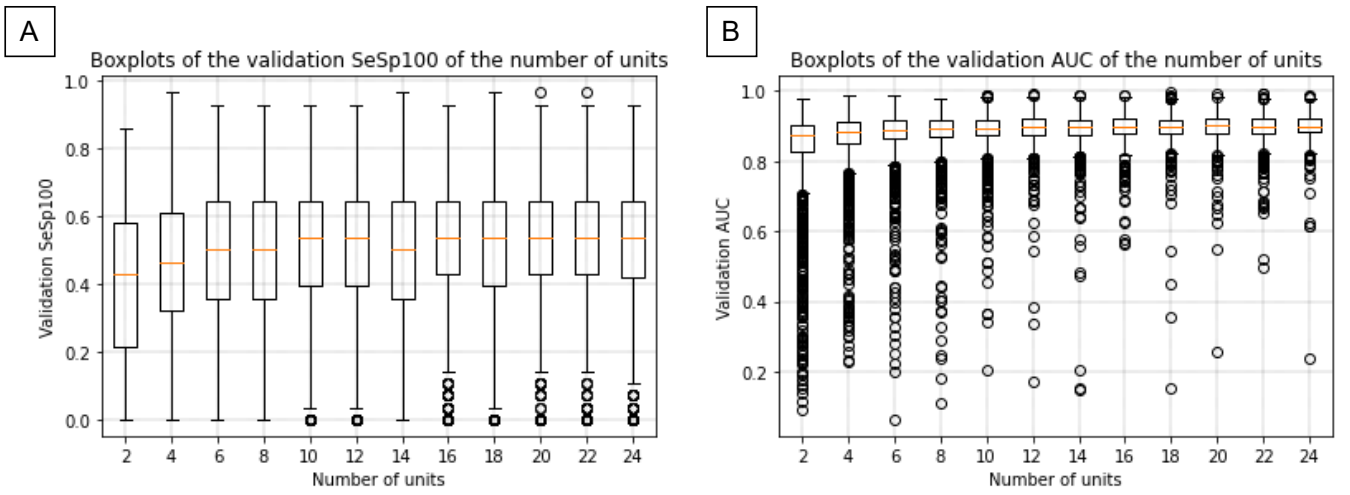


Figure A.34: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per number of units used in the LSTM network with one LSTM layer. The SeSp100 increased if the number of units was equal to or more than 10, except for 14 units. The AUC also increased with an increasing number of units. The AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100 = sensitivity at 100% specificity (for poor outcome prediction).

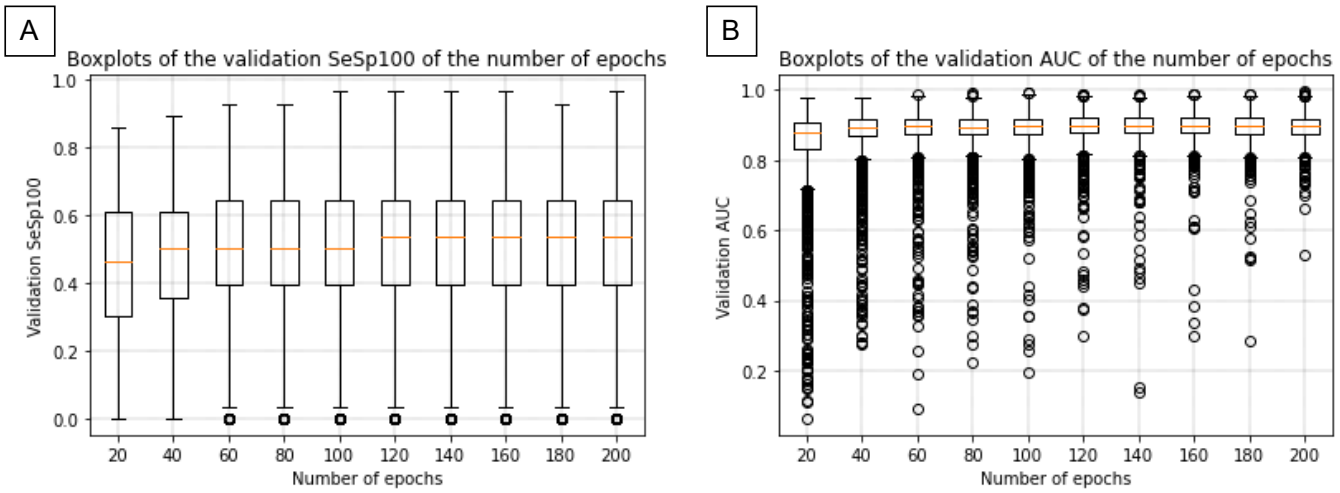


Figure A.35: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per number of epochs used to train the LSTM network with one LSTM layer. The SeSp100 increased if the number of epochs was equal to or more than 120. The AUC also increased with an increasing number of epochs. The AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100 = sensitivity at 100% specificity (for poor outcome prediction).

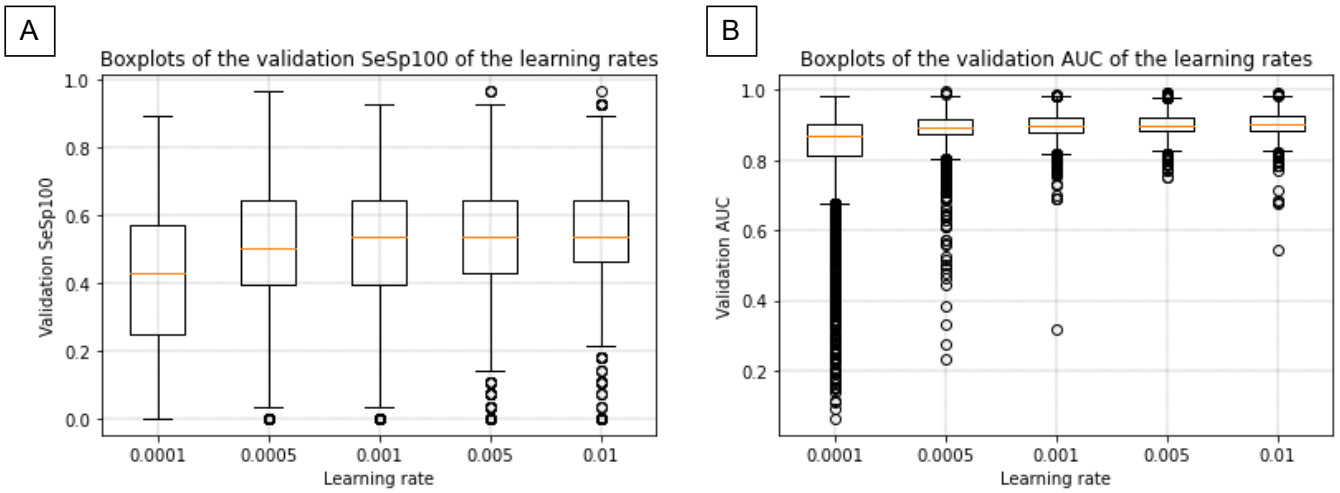


Figure A.36: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per learning rate in the LSTM network with one LSTM layer. The SeSp100 was the highest for the learning rates of 0.001, 0.005, and 0.01. The AUC of the learning rates of 0.001, 0.005, and 0.01 gave the highest performance with the least number of outliers. The AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100 = sensitivity at 100% specificity (for poor outcome prediction).

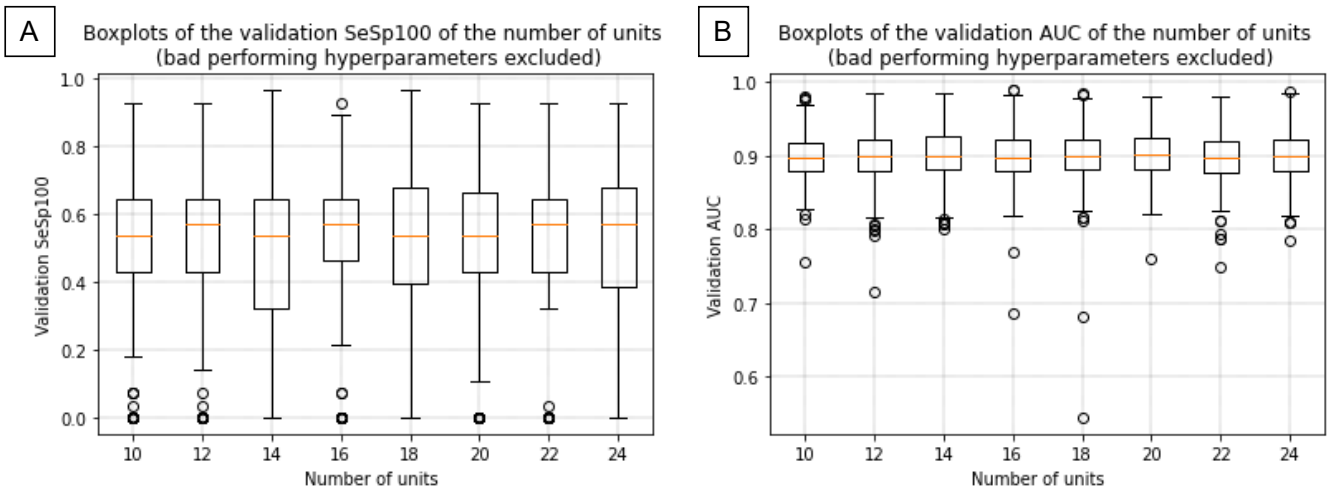


Figure A.37: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per number of units in the LSTM network with one LSTM layer. These boxplots only include the configurations with hyperparameters that obtained high performance on both metrics. An LSTM layer composed of 16 units showed the highest SeSp100 most consistently, as the box and whiskers were small. The difference in AUC between the number of units was hardly noticeable. AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100 = sensitivity at 100% specificity (for poor outcome prediction)

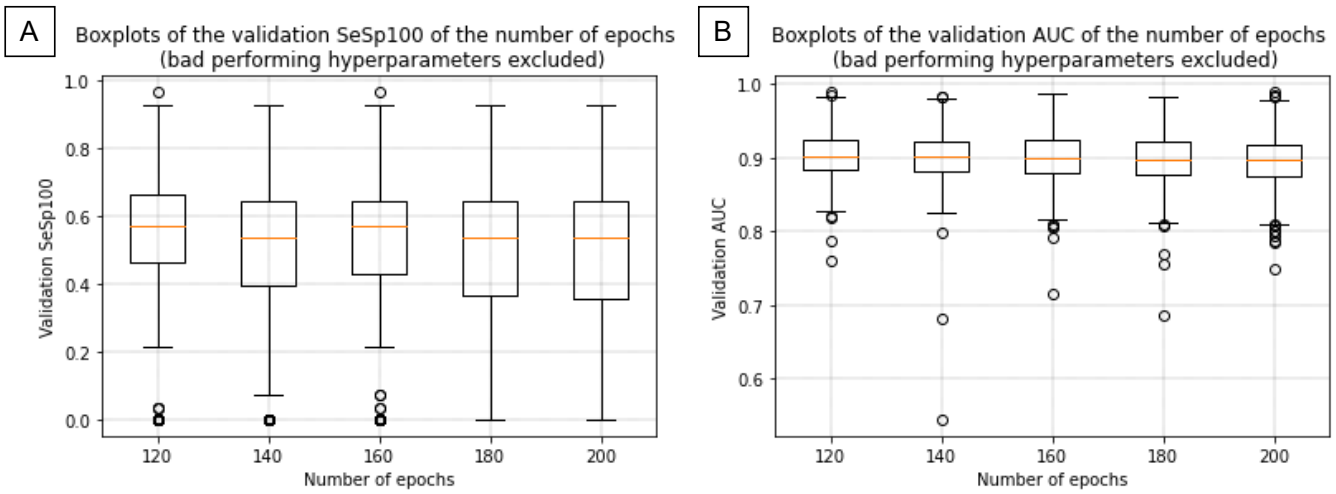


Figure A.38: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per number of epochs used to train the LSTM network with one LSTM layer. These boxplots only include the configurations with hyperparameters that obtained high performance on both metrics. 120 number of epochs showed the highest SeSp100 with the smallest distribution. The difference in AUC between the number of epochs was hardly noticeable, only slightly decreasing with an increased number of epochs. AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100 = sensitivity at 100% specificity (for poor outcome prediction).

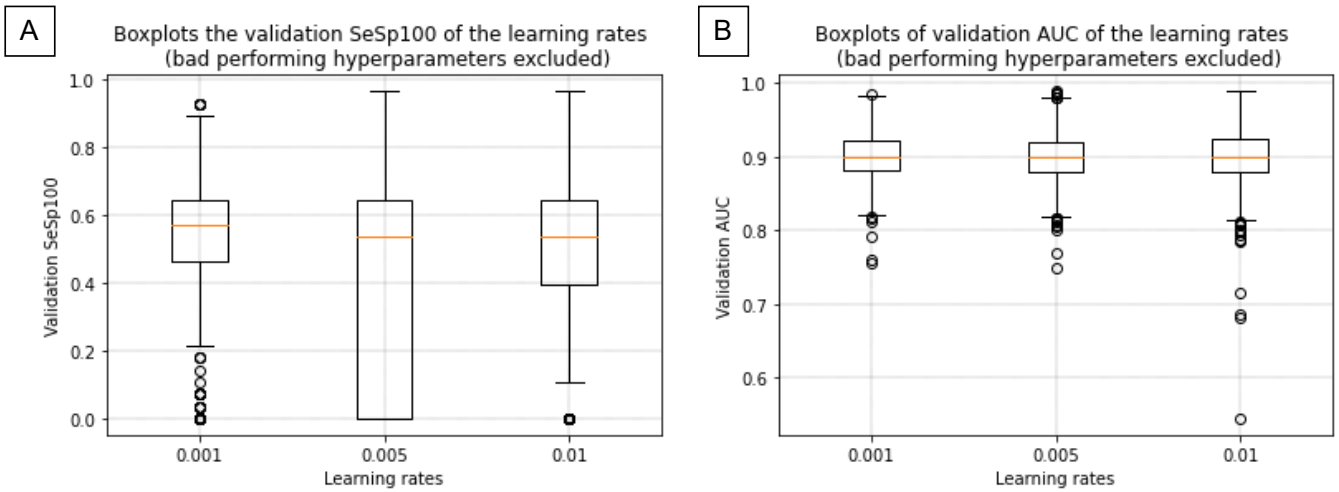


Figure A.39: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per learning rate in the LSTM network with one LSTM layer. These boxplots only include the configurations with hyperparameters that obtained high performance on both metrics. The models using a learning rate of 0.001 clearly achieved a higher SeSp100 and gave the smallest distribution. The difference in AUC between the number of units was hardly noticeable, but the learning rate of 0.01 resulted in a little more outliers. AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity (for poor outcome prediction).

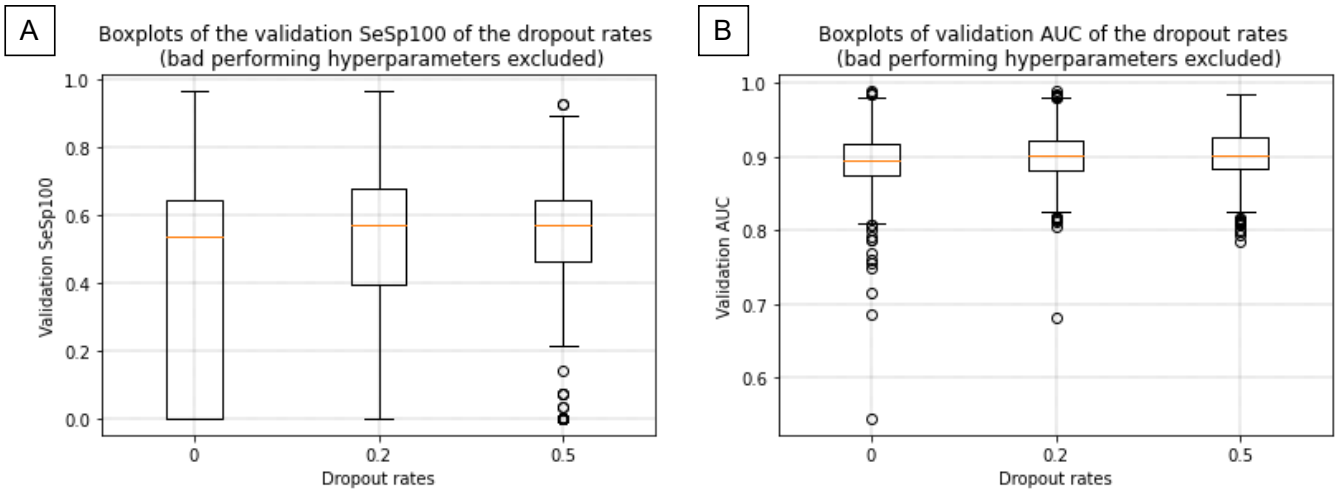


Figure A.40: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per dropout rate in the LSTM network with one LSTM layer. These boxplots only include the configurations with hyperparameters that obtained high performance on both metrics. The models using a dropout rate of 0.5 clearly were the most consistent in their SeSp100. Furthermore, the median SeSp100 for the dropout rate of 0.2 and 0.5 were approximately the same and higher than for no dropout. The difference in AUC between the dropout rates was little, where no dropout performed slightly worse than dropout. AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity (for poor outcome prediction).

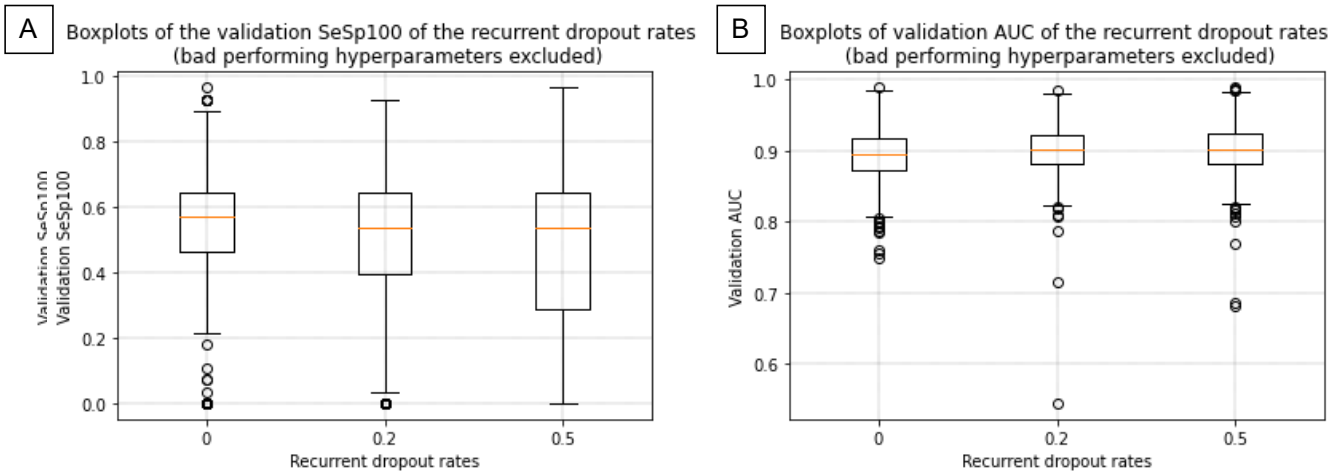


Figure A.41: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per recurrent dropout rate in the LSTM network with one LSTM layer. These boxplots only include the configurations with hyperparameters that obtained high performance on both metrics. The models using a recurrent dropout rate of 0 clearly showed consistently high SeSp100. The difference in AUC between the recurrent dropout rates was little. AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity (for poor outcome prediction).

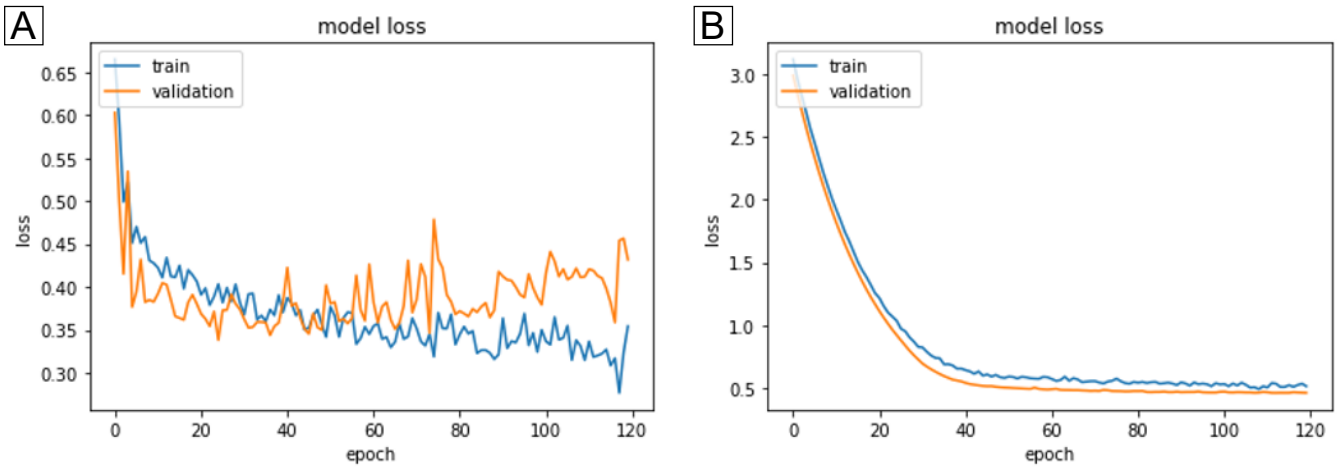


Figure A.42: Plots of the training and validation loss against the number of epochs for a one-layer LSTM model with no weight regularisation (A) and a one-layer LSTM model with weight regularisation (B). LSTM=long short-term memory recurrent neural network.

Table A.XX: Best performing hyperparameter values for a one-layer LSTM model. LSTM=long short-term memory recurrent neural network. Reg=regularisation.

	Number of epochs	Number of units	Learning rate	Dropout rate	Recurrent dropout rate	Kernel reg.	Recurrent kernel reg.	Bias reg.
Value	120	16	0.001	0.5	0	L1_L2 0.001	L1_L2 0.001	L1_L2 0.001

Table A.XXI: Performance metrics of the one-layer LSTM model using the best performing hyperparameter values. AUC=area under the curve. CI=confidence interval. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity, SeSp95=sensitivity at 95% specificity.

AUC Mean (95% CI)	SeSp100 of poor outcome prediction Mean (95% CI)	SeSp95 of good outcome prediction Mean (95% CI)
0.8981 (0.8930-0.9032)	0.6621 (0.6478-0.6764)	0.5739 (0.5511-0.5967)

Results for a two-layer LSTM

The correlation between the hyperparameters and the validation AUC and the SeSp100 of all random searches is stated in Table A.XXII and Table A.XXIII, respectively. The number of epochs, the number of units, and the learning rate showed a notable positive correlation with the validation AUC. Overall, the SeSp100 showed less consistent correlation over all searches than the AUC. However, the number of epochs correlated positively with the SeSp100. Boxplots of the number of units (Fig. A.43), the number of epochs (Fig. A.44), and the learning rate (Fig. A.45) for both metrics were evaluated to make the first selection of well-performing hyperparameter options. The SeSp100 was the highest for the number of epochs between 80 and 140, without compromising the AUC. The number of units equal to or higher than 16, and a learning rate of 0.005, obtained the best scores for both metrics. Therefore, only permutations that used these hyperparameter values were used for further evaluation. Boxplots of the SeSp100 and AUCthat included only these permutations were evaluated for the number of units (Fig. A.46), the number of epochs (Fig. A.47), the dropout rate (Fig. A.48), and the recurrent dropout rate (Fig. A.49). I achieved the highest values with the smallest distribution for the SeSp100 by training the model for 80 epochs, 20 units in the LSTM layers, a learning rate of 0.005, and applying a dropout rate of 0.5 and no recurrent dropout. Moreover, these hyperparameters also resulted in high values with a small distribution for the AUC. Therefore, I chose these hyperparameters values for the LSTM model that was composed of two LSTM layers. Subsequently, I checked if these hyperparameters caused overfitting, and I manually applied weight regularisation to increase the model's generalisation ability. Overfitting was prevented, and the highest validation AUC and Se100 of poor outcome prediction were achieved using L1_L2 regularisation of 0.001 on the bias and the recurrent kernel and kernel weights. Plots of the training and validation loss against the number of epochs for the model with and without weight regularisation are shown in Figure A.50. Table A.XXIV summarises the best hyperparameter values for a two-layer LSTM. Table A.XXV states the performances achieved by this model using 50 repetitions of 10-fold CV.

Best performing LSTM

Table A.XXVI states the final model performances of the one-layer and two-layer LSTMs. The differences between the performance metrics were not statistically significant. The number of trainable model parameters was 1873 and 5941 for the one-layer LSTM and two-layer LSTM, respectively. The more model parameters are included in the network, the longer the computational time. As the performance differences were not statistically significant, and the number model parameters differed by a factor 3, I chose the one-layer model as final LSTM.

Table A.XXII: Correlation between the hyperparameters and the validation AUC of all random searches for a model with 2 LSTM layers. AUC= area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. Val=validation.

Correlation with val. AUC	Search 1	Search 2	Search 3	Search 4	Search 5	Search 6	Search 7	Search 8	Search 9	Search 10	Mean of all searches
Dropout rate	0.062	0.073	0.079	0.022	0.07	0.059	0.011	0.097	0.009	-0.016	**0.047**
Number of epochs	0.101	0.151	0.129	0.126	0.08	-0.075	0.069	0.187	0.263	0.079	**0.111**
Number of units	0.237	0.221	0.237	0.212	0.274	0.154	0.198	0.242	0.257	0.22	**0.225**
Learning rate	0.112	0.285	0.197	0.069	0.103	-0.044	0.064	0.265	0.224	0.109	**0.139**
Recurrent dropout rate	0.065	0.016	0.012	-0.012	0.02	0.078	-0.021	0.012	0.03	-0.003	**0.02**

Table A.XXIII: Correlation between the hyperparameters and the validation SeSp100 of poor outcome prediction of all random searches for a model with 2 LSTM layers. LSTM=long short-term memory recurrent neural network. SeSp100 = sensitivity at 100% specificity Val=validation.

Correlation with val. SeSp100	Search 1	Search 2	Search 3	Search 4	Search 5	Search 6	Search 7	Search 8	Search 9	Search 10	Mean of all searches
Dropout rate	0.04	0.118	0.197	0.07	0.194	-0.029	-0.078	0.295	0.202	0.073	**0.108**
Number of epochs	0.214	0.12	0.015	-0.043	-0.063	-0.035	0.223	-0.182	-0.038	0.157	**0.037**
Number of units	0.356	0.271	0.29	0.096	0.224	0.148	0.344	0.032	0.16	0.285	**0.221**
Learning rate	0.282	0.142	0.068	-0.148	0.025	-0.141	0.321	-0.131	-0.018	0.197	**0.06**
Recurrent dropout rate	-0.005	0.03	-0.022	-0.108	-0.073	0.104	0.037	-0.065	0.047	0.034	**-0.002**

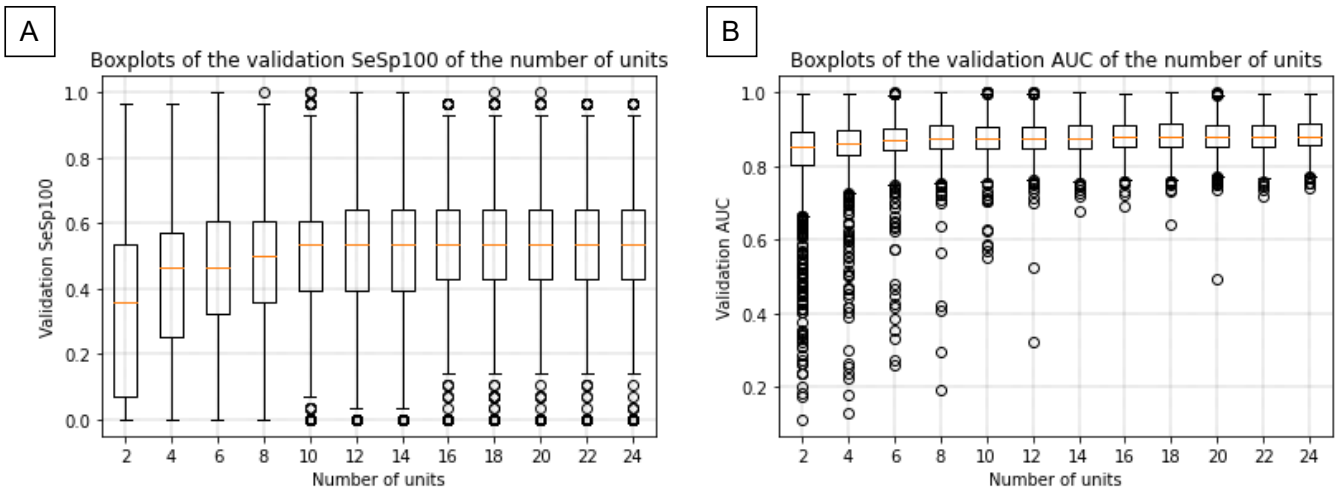


Figure A.43: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per number of units used in the LSTM network with two LSTM layers. The SeSp100 increased, and the distribution decreased if the number of units was equal to or more than 16. The AUC also increased with an increasing number of units, where less than 12 units result in relatively more outliers. The AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity.

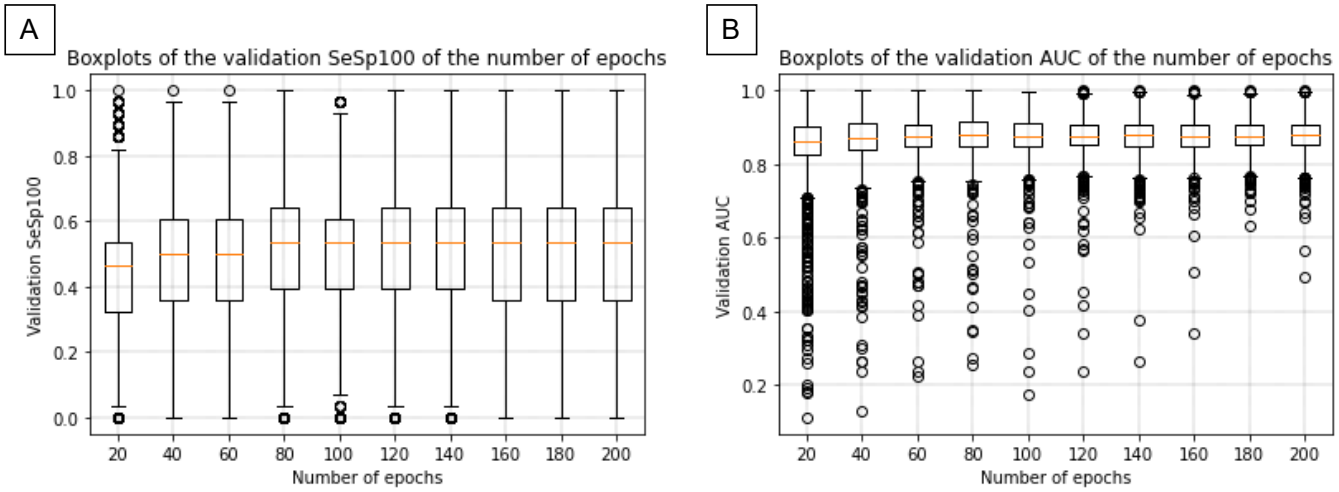


Figure A.44: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per number of epochs used to train the LSTM network with two LSTM layers. The SeSp100 was the highest and showed the smallest distribution if the epochs were between 80 and 140. The number of outliers for the AUC decreased with an increasing number of epochs. The AUC=area under the receiver operating curve. SeSp100=sensitivity at 100% specificity.

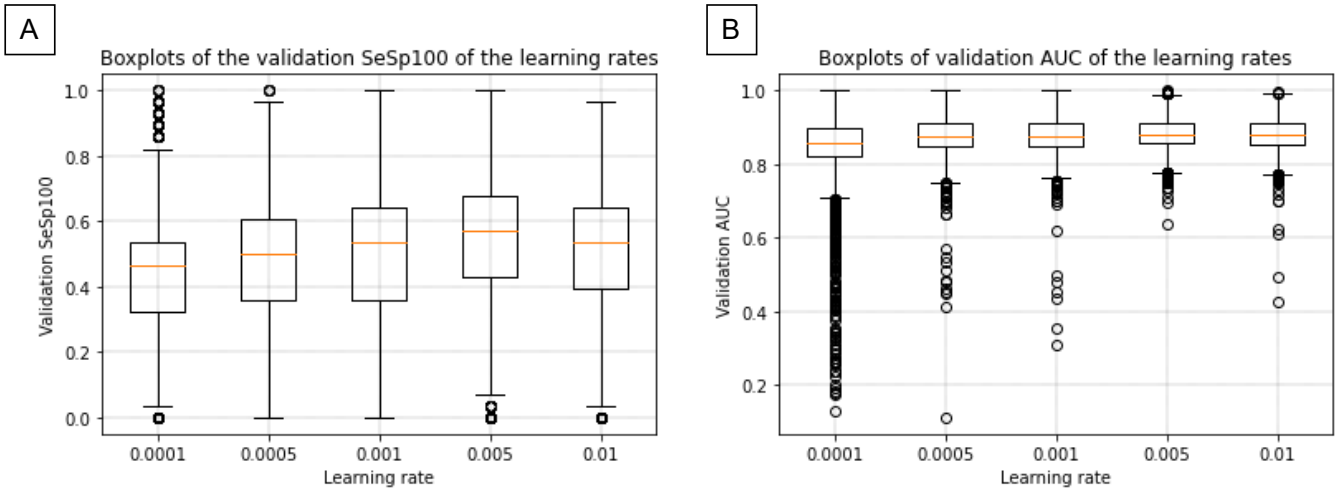


Figure A.45: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per learning rate in the LSTM network with two LSTM layers. The SeSp100 was the highest for the learning rate of 0.005. The learning rate of 0.005 also gave the least number of outliers and the highest values in the AUC. The AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity.

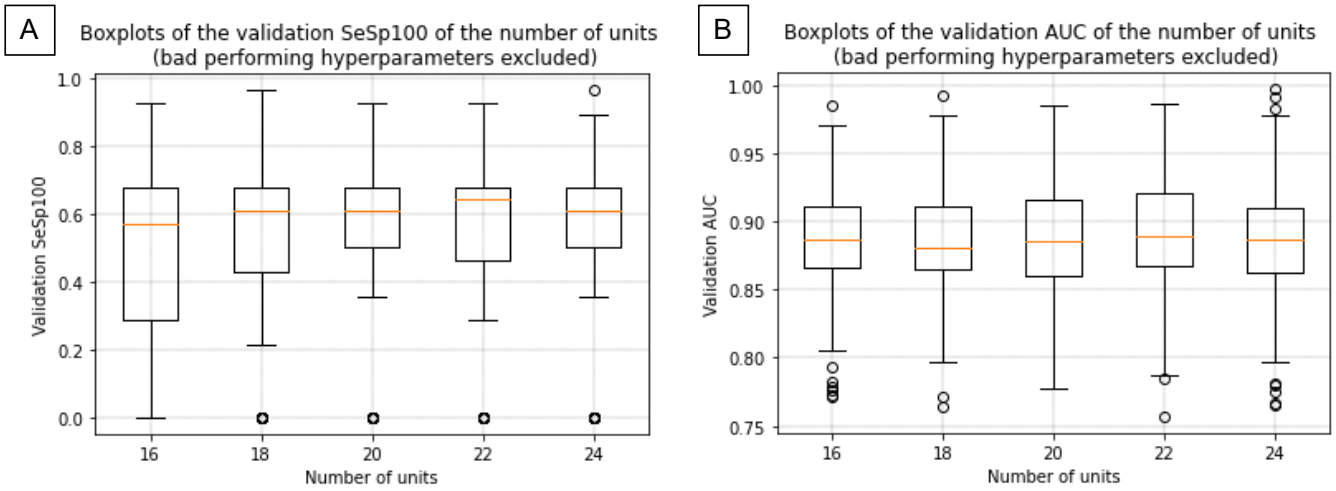


Figure A.46: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per number of units in the LSTM network with two LSTM layers. These boxplots only include the configurations with hyperparameters that obtained high performance on both metrics. The LSTM layers composed of 20 units showed a relatively high SeSp100 while having a small distribution. The median SeSp100 of 22 units was a little higher, but also included more models scoring a lower SeSp100 than models composed of 20 units. The difference in the median of the AUC between the number of units was hardly noticeable. Models composed of 16, 18 and 24 units gave somewhat more consistent results for AUC. AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity.

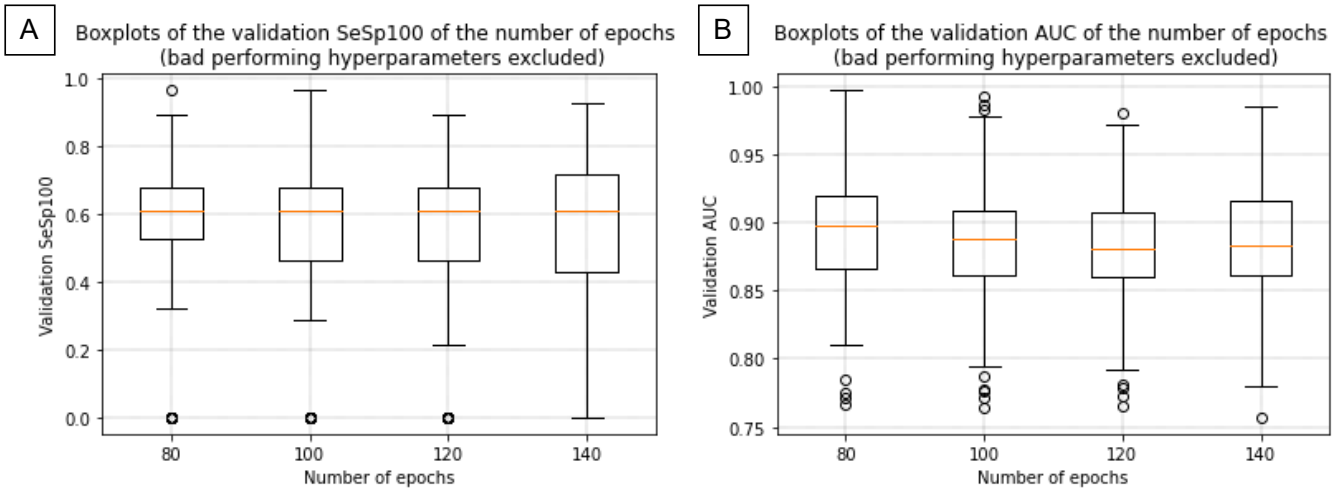


Figure A.47: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per number of epochs used to train the LSTM network using two LSTM layers. These boxplots only include the configurations with hyperparameters that obtained high performance on both metrics. Training the model 80 epochs showed consistently high values for SeSp100. The best AUC was also achieved with training for 80 epochs. AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity.

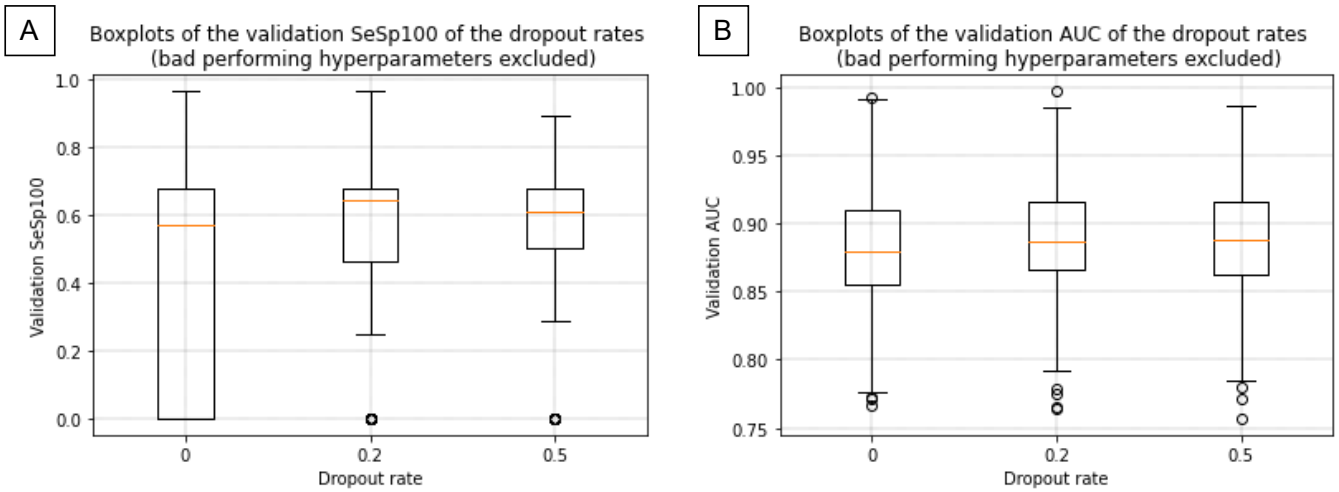


Figure A.48: Boxplots of the validation the SeSp100 (A) and the validation AUC (B) of poor outcome prediction per dropout rate in the LSTM network with two LSTM layers. These boxplots only include the configurations with hyperparameters that obtained high performance on both metrics. The models using no dropout clearly scored a lower Se100. The dropout rates of 0.2 and 0.5 achieved similar SeSp100. Models using a rate of 0.2 scored a higher median SeSp100 but had a larger distribution. Therefore, it included models that scored higher but also lower SeSp100 than modes using a rate of 0.5. A rate of 0.5 gave more consistent results and still scored high SeSp100. The difference in AUC between the dropout rates was little, where no dropout performed slightly worse than dropout. AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity.

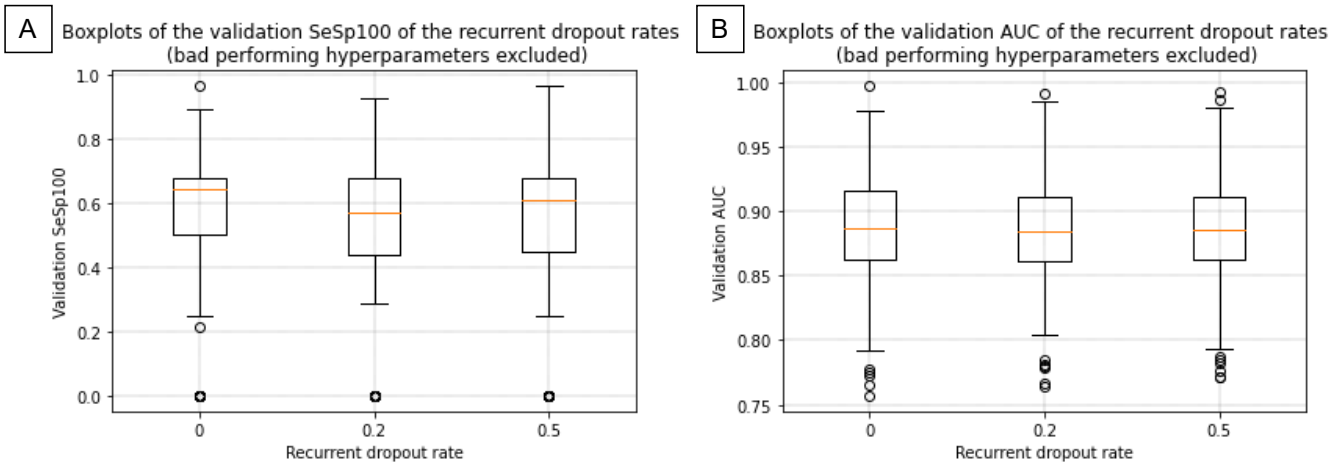


Figure A.49: Boxplots of the validation the Se100 (A) and the validation AUC (B) of poor outcome prediction per recurrent dropout rate in the LSTM network with two LSTM layers. These boxplots only include the configurations with hyperparameters that obtained high performance on both metrics. The models using a recurrent dropout rate of 0 were clearly the most consistent and highest in their SeSp100. The difference in AUC between the recurrent dropout rates was little, where no recurrent dropout performed slightly better than a recurrent dropout. AUC=area under the receiver operating curve. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity.

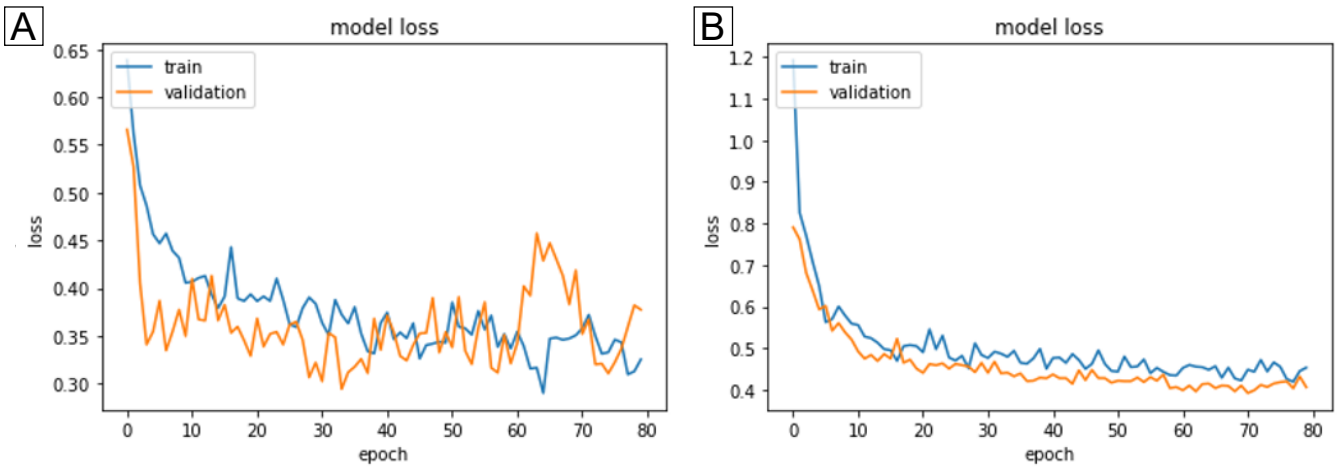


Figure A.50: Plots of the training and validation loss against the number of epochs for a two-layer LSTM model with no weight regularisation (A) and a model with weight regularisation (B). LSTM=long short-term memory recurrent neural network.

Table A.XXIV: Best performing hyperparameter values for a two-layer LSTM model. LSTM=long short-term memory recurrent neural network. Reg=regularisation.

	Number of epochs	**Number of units**	**Learning rate**	**Dropout rate**	**Recurrent dropout rate**	**Kernel reg.**	**Recurrent kernel reg.**	**Bias reg.**
Value	80	20	0.005	0.5	0	L1_L2 0.001	L1_L2 0.001	L1_L2 0.001

Table A.XXV: Performance metrics of the two-layer LSTM model using the best performing hyperparameter values. AUC=area under the curve. CI=confidence interval. LSTM=long short-term memory recurrent neural network. SeSp100=sensitivity at 100% specificity, SeSp95=sensitivity at 95% specificity.

AUC Mean (95% CI)	**SeSp100 of poor outcome prediction Mean (95% CI)**	**SeSp95 of good outcome prediction Mean (95% CI)**
0.8952 (0.8900-0.9005)	0.6570 (0.6421-0.6718)	0.5760 (0.5537-0.5984)

Table A.XXVI: Performance metrics of the one-layer and two-layer LSTMs using the best performing hyperparameters. AUC=area under the curve, SeSp100=sensitivity at 100% specificity, SeSp95=sensitivity at 95% specificity. LSTM=long short-term memory recurrent neural network. CI=confidence interval.

	AUC Mean (95% CI)	**Se100 of poor outcome prediction Mean (95% CI)**	**Se95 of good outcome prediction Mean (95% CI)**
One-layer LSTM	0.8981 (0.8930-0.9032)	0.6621 (0.6478-0.6764)	0.5739 (0.5511-0.5967)
Two-layer LSTM	0.8952 (0.8900-0.9005)	0.6570 (0.6421-0.6718)	0.5760 (0.5537-0.5984)

X. Results LSTM – Final model performances

Models trained at different timepoints

Table A.XXVII compares the performance metrics of the LSTMs separately trained and evaluated at both timepoints (12 and 24 hours), 12 hours, and 24 hours after CA. The models were trained and evaluated with features from the final feature set (Table A.IV). The AUCs, all of which were rounded to 0.90, were not significantly different from each other. The SeSp100 at 12 hours of 0.785 was significantly higher than that at 24 hours (0.662) or both timepoints (0.681). On the contrary, the SeSp95 at 12 hours of 0.303 was significantly lower than that at 24 hours (0.574) or both timepoints (0.439). The SeSp100 of the model trained at 24 hours was not significantly different those at both timepoints. The SeSp95 at 24 hours was significantly lower than that at both timepoints.

Models trained with different feature sets

Table A.XXVIII compares the performance metrics of the LSTMs trained and evaluated using features from the FFS and the AFSs using all epochs at both timepoints. Adding the clinical features age and sex to the FFS (AFS1) or using all 19 extracted qEEG features (AFS 2) (with additional clinical features (AFS 3) did not result in statistically significant different performance for any of the metrics. Excluding features that showed little discriminative power between poor and good outcome (AFS 4) and additionally excluding the features with high multicollinearity (AFS 5) significantly increased the SeSp95 from 0.574 (FFS) to 0.702 (AFS 4) and 0.719 (AFS 5). However, these AFS showed significantly decreased AUC of 0.867 (AFS 5) and nearly significantly (p=0.06) decreased AUC of 0.874 (AFS 4), compared to the FFS with an AUC of 0.898. As the SeSp100 and the AUC were considered more important than SeSp95, none of the AFS outperformed the FFS.

Y. Hyperparameter recommendations

I recommend exploring different optimiser algorithms. Although Adam is widely used and considered a right default choice [150, 152, 154, 179], other options might achieve better prognostic performance. The Nadam optimiser should be investigated, as it applies NAG, which is often superior to momentum. Furthermore, the author claims significant performance improvement of Nadam over Adam [178]. Although adaptive optimisers like Adam and Nadam dampen[175, 178], they have their challenges. Adaptive optimisers could be unsuccessful in converging to the minimum loss [181, 182]. Moreover, adaptive optimisers show worse generalisation than traditional SGD optimisers [183]. For these reasons, SGD should be included in the hyperparameter search as well. An interesting option to explore is the recently developed optimiser AdaBound, which is claimed to be "as fast as Adam and as good as SGD" [181]. Adabound incorporates the fast convergence seen in adaptive optimiser like Adam and the reliable generalisation ability seen in SGD [181]). Concerning the learning rate, I advise implementing a grid search on a logarithmic scale to determine which order the optimiser's learning rate should be. Subsequently, a grid search around the best order should result in the most optimal rate. One could also include the other hyperparameters within the adaptive optimisers in the search space, like the decay rates and constant numerical stability. As Keras argues that the latter's default value might not be a good default in general [79], different values for this hyperparameter could be worth exploring. On the other hand, the decay rates suggested by the paper are good default values which are rarely changed. Moreover, the optimisers are robust to these hyperparameters' choice [154, 175]. Therefore, I doubt changing the values of the decay rates would make a difference in the performance.

As the manually tuned weight regularisation notably increased generalisation, future research should incorporate it into the search space. Ideally, weight regularisation on the kernel weight, recurrent weight, biases and activity are individually evaluated. Moreover, multiple values for the regularisation factor should be explored for L1, L2, and L1_L2 regularisation. I recommend a full grid search to find the optimal regularisation factor on a logarithmic scale from 0 and 0.1 [184].

Weight constraints could also be worth exploring, where different constraints should be evaluated for the input and recurrent connections [172]. Furthermore, I especially recommend trying max-norm regularisation if dropout is applied. The dropout algorithm developers reported increased performance when dropout and max-norm regularisation were combined instead of just dropout ([167].

Table A.XXVII: Comparison of the performance metrics on the test sets of the LSTM separately trained and evaluated at both 12 and 24, 12 hours, and 24 hours after cardiac arrest. The models were trained and evaluated with features from the final feature set. AUC=area under the curve, SeSp100=sensitivity at 100% specificity, SeSp95=sensitivity at 95% specificity.

	AUC Mean (95% CI)	**SeSp100 Mean (95% CI)**	**SeSp95 Mean (95% CI)**
LSTM – 12 h + 24 h	0.898 (0.893-0.903)	0.662 (0.648-0.676)	0.574 (0.551-0.597)
LSTM – 12 h	0.901 (0.891-0.912)	0.785 (0.764-0.805)	0.303 (0.275-0.332)
LSTM – 24 h	0.901 (0.895-0.908)	0.681 (0.664-0.697)	0.439 (0.415-0.463)

Table A.XXVIII: Comparison of the performance on the test set of the LSTM models trained and evaluated with features from the final feature set and additional feature set using all epochs on both timepoints. *The difference of the performance metric was statistically significant from the performance metric obtained by using features from the final feature set. AFS=additional feature set. AUC=area under the curve. CI=confidence interval. FFS=final feature set. SeSp100=sensitivity at 100% specificity. SeSp95=sensitivity at 95% specificity.

	AUC - Mean (95% CI)	**SeSp100 - Mean (95% CI)**	**SeSp95 - Mean (95% CI)**
FFS	0.898 (0.893-0.903)	0.662 (0.648-0.676)	0.574 (0.551-0.597)
AFS 1	0.889 (0.884-0.894)	0.648 (0.634-0.662)	0.604 (0.583-0.626)
AFS 2	0.897 (0.892-0.902)	0.651 (0.634-0.667)	0.577 (0.555-0.598)
AFS 3	0.887 (0.882-0.892)	0.649 (0.634-0.663)	0.617 (0.596-0.637)
AFS 4	0.874 (0.868-0.880)	0.643 (0.625-0.661)	0.702 (0.681-0.724)*
AFS 5	0.867 (0.860-0.873)*	0.655 (0.638-0.672)	0.719 (0.697-0.740)*

Some EEG-based LSTM studies obtained good performance with the ReLu activation function (e.g. [37, 86]), so it is worth including this in the hyperparameter optimisation process if computational power allows it.

The same hyperparameters values for both LSTM layers were used to reduce the computational power: all configurations used the same number of units and the same (recurrent) dropout rate. As these hyperparameters' optimal values could be different for the individual layers, these hyperparameters should be tuned separately. Finally, although the batch size does not influence the generalisation ability very much, it does affect the training time [77]). Different batch sizes should be evaluated to optimise the training time.

Z. MATLAB and Python scripts

All scripts are attached to this thesis after the references. All scripts include:

- Z.1 Feature extraction (MATLAB)
- Z.2 High VIF feature exclusion (MATLAB)
- Z.3 Input preparation for logistic regression (MATLAB)
- Z.4 Input preparation for LSTM (MATLAB)
- Z.5 Feature boxplots (MATLAB)
- Z.6 Final Logistic Regression (Python)
- Z.7 Hyperparameter search LSTM (Python)
- Z.8 Hyperparameter search analysis LSTM (Python)
- Z.9 Final LSTM (Python)

References

[1] S. A. Bernard, T. W. Gray, M. D. Buist, B. M. Jones, W. Silvester, G. Gutteridge, and K. Smith, "Treatment of comatose survivors of out-of-hospital cardiac arrest with induced hypothermia," N Engl J Med, vol. 346, 2002.

[2] A. Thomassen and M. Wernberc, "Prevalence and prognostic significance of coma after cardiac arrest outside intensive care and coronary units," Acta anaesth, vol. 23, pp. 143–148, 1979.

[3] E. G. Zandbergen, A. Hijdra, J. H. Koelman, A. A. Hart, P. E. Vos, M. M. Verbeek, R. J. de Haan, and P. S. Group, "Prediction of outcome in anoxic-ischaemic coma," 2006.

[4] C. M. Booth, R. H. Boone, G. Tomlinson, A. S. Detsky, C. Author, D. L. Simel, D. Rennie, and D. Edi-tor, "Is this patient dead, vegetative, or severely neurologically impaired? assessing outcome for comatose survivors of cardiac arrest the rational clinical examination section editors," JAMA, vol. 291, pp. 870–879, 2004.

[5] C. Sandroni, S. D'Arrigo, and J. P. Nolan, "Prognostication after cardiac arrest," Critical Care, vol. 22, 6 2018.

[6] S. Jonas, A. O. Rossetti, M. Oddo, S. Jenni, P. Favaro, and F. Zubler, "Eeg-based outcome prediction after cardiac arrest with convolutional neural networks: Performance and visualization of discriminative features," Human Brain Mapping, vol. 40, pp. 4606–4617, 11 2019.

[7] E. A. Samaniego, S. Persoon, and C. A. Wijman, "Prognosis after cardiac arrest and hypothermia: A new paradigm," Current Neurology and Neuroscience Reports, vol. 11, pp. 111–119, 2 2011.

[8] C. Sandroni, F. Cavallaro, C. W. Callaway, T. Sanna, S. D'Arrigo, M. Kuiper, G. D. Marca, and J. P. Nolan, "Predictors of poor neurological outcome in adult comatose survivors of cardiac arrest: A systematic review and meta-analysis. part 1: Patients not treated with therapeutic hypothermia," Resuscitation, vol. 84, pp. 1310–1323, 10 2013.

[9] M. Oddo and A. O. Rossetti, "Early multimodal outcome prediction after cardiac arrest in pa-

tients treated with hypothermia," Critical Care Medicine, vol. 42, pp. 1340–1347, 2014.

[10] M. C. Cloostermans, F. B. V. Meulen, C. J. Eertman, H. W. Hom, and M. J. V. Putten, "Continuous electroencephalography monitoring for early prediction of neurological outcome in postanoxic patients after cardiac arrest: A prospective cohort study," Critical Care Medicine, vol. 40, pp. 2867–2875, 10 2012.

[11] M. Dougherty, "The utility of eeg in prognosis post-cardiac arrest," Practical Neurology, pp. 26–27, 2017.

[12] J. Hofmeijer, T. M. Beernink, F. H. Bosch, A. Beishuizen, M. C. Tjepkema-Cloostermans, M. J. van Putten, and F. J. C. Neurophysiology, "Early eeg contributes to multimodal outcome prediction of postanoxic coma," Neurology, vol. 85, pp. 137–143, 2015.

[13] A. O. Rossetti, A. A. Rabinstein, and M. Oddo, "Neurological prognostication of outcome in patients in coma after cardiac arrest," The Lancet Neurology, vol. 15, pp. 597–609, 5 2016.

[14] A. Sivaraju, E. J. Gilmore, C. R. Wira, A. Stevens, N. Rampal, J. J. Moeller, D. M. Greer, L. J. Hirsch, and N. Gaspard, "Prognostication of post-cardiac arrest coma: early clinical and electroencephalographic predictors of outcome," Intensive Care Medicine, vol. 41, pp. 1264–1272, 7 2015.

[15] M. Spalletti, R. Carrai, M. Scarpino, C. Cossu, A. Ammannati, M. Ciapetti, L. T. Buoninsegni, A. Peris, S. Valente, A. Grippo, and A. Amantini, "Single electroencephalographic patterns as specific and time-dependent indicators of good and poor outcome after cardiac arrest," Clinical Neurophysiology, vol. 127, pp. 2610–2617, 7 2016.

[16] M. C. Tjepkema-Cloostermans, J. Hofmeijer, R. J. Trof, M. J. Blans, A. Beishuizen, and M. J. V. Putten, "Electroencephalogram predicts outcome in patients with postanoxic coma during mild therapeutic hypothermia," Critical Care Medicine, vol. 43, pp. 159–167, 1 2015.

[17] J. Hofmeijer and M. van Putten, "Eeg in postanoxic coma: Prognostic and diagnostic value," Clinical Neurophysiology, vol. 127, pp. 2047–2055, 4 2016.

[18] S. B. Nagaraj, M. C. Tjepkema-Cloostermans, B. J. Ruijter, J. Hofmeijer, and M. J. van Putten, "The revised cerebral recovery index improves predictions of neurological outcome after cardiac arrest," Clinical Neurophysiology, vol. 129, pp. 2557–2566, 12 2018.

[19] M. C. Tjepkema-Cloostermans, F. B. van Meulen, G. Meinsma, and M. J. van Putten, "A cerebral recovery index (cri) for early prognosis in patients after cardiac arrest," Critical Care, vol. 17, 10 2013.

[20] M. C. Tjepkema-Cloostermans, C. da Silva Lourenço, B. J. Ruijter, S. C. Tromp, G. Drost, F. H. Kornips, A. Beishuizen, F. H. Bosch, J. Hofmeijer, and M. J. van Putten, "Outcome prediction in postanoxic coma with deep learning," Critical care medicine, vol. 47, pp. 1424–1432, 10 2019.

[21] M. J. van Putten, J. Hofmeijer, B. J. Ruijter, and M. C. Tjepkema-Cloostermans, "Deep learning for outcome prediction of postanoxic coma," IFMBE Proceedings, vol. 65, pp. 506–509, 2017.

[22] R. Agarwal, J. Gotman, D. Flanagan, and B. Rosenblatt, "Automatic eeg analysis during long-term monitoring in the icu," Electroencephalography and clinical Neurophysiology, vol. 107, pp. 44–58, 1998.

[23] R. P. Brenner, "How useful is eeg and eeg monitoring in the acutely ill and how to interpret it?," Epilepsia, vol. 50, pp. 34–37, 12 2009.

[24] B. Foreman and J. Claassen, "Quantitative eeg for the detection of brain ischemia," Critical care, vol. 16, pp. 216–225, 2012.

[25] N. Gaspard, "Acns critical care eeg terminology: Value, limitations, and perspectives," Clincal Neurophysiology, vol. 32, pp. 452–455, 2015.

[26] H. Moshirvaziri, N. Ramezan-Arab, and S. Asgari, "Prediction of the outcome in cardiac arrest patients undergoing hypothermia using eeg wavelet entropy," 2016.

[27] M. J. A. M. V. Putten, "The colorful brain: Visualization of eeg background patterns," J Clin Neurophysiol, vol. 25, pp. 63–68, 2008.

[28] M. A. Koenig, P. W. Kaplan, and N. V. Thakor, "Clinical neurophysiologic monitoring and brain injury from cardiac arrest," Neurologic Clinics, vol. 24, pp. 89–106, 2 2006.

[29] B. J. Ruijter, J. Hofmeijer, M. C. Tjepkema-Cloostermans, and M. J. van Putten, "The prognostic value of discontinuous eeg patterns in postanoxic coma," Clinical Neurophysiology, vol. 129, pp. 1534–1543, 8 2018.

[30] J. E. Wennervirta, M. J. Ermes, S. M. Tiainen, T. K. Salmi, M. S. Hynninen, M. O. Särkelä, M. J. Hynynen, U. H. Stenman, H. E. Viertiö-Oja, K. P. Saastamoinen, V. Y. Pettilä, and A. P. Vakkuri, "Hypothermia-treated cardiac arrest patients with

good neurological outcome differ early in quantitative variables of eeg suppression and epileptiform activity," Critical Care Medicine, vol. 37, pp. 2427–2435, 2009.

[31] C. B. Swisher and S. R. Sinha, "Utilization of quantitative eeg trends for critical care continuous eeg monitoring: A survey of neurophysiologists," Journal of Clinical Neurophysiology, vol. 33, pp. 538–544, 12 2016.

[32] F. Zubler, M. Bandarabadi, R. Kurmann, A. Steimer, H. Gast, and K. A. Schindler, "Quantitative eeg in the intensive care unit," Epileptologie, vol. 33, pp. 166–172, 2016.

[33] S. Asgari, H. Moshirvaziri, F. Scalzo, and N. Ramezan-Arab, "Quantitative measures of eeg for prediction of outcome in cardiac arrest subjects treated with hypothermia: a literature review," Journal of Clinical Monitoring and Computing, vol. 32, pp. 977–992, 12 2018.

[34] M. M. Ghassemi, E. Amorim, S. B. Pati, R. G. Mark, E. N. Brown, P. L. Purdon, and M. B. Westover, "An enhanced cerebral recovery index for coma prognostication following cardiac arrest," Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS, vol. 2015-November, pp. 534–537, 11 2015.

[35] M. C. Tjepkema-Cloostermans, J. Hofmeijer, A. Beishuizen, H. W. Hom, M. J. Blans, F. H. Bosch, and M. J. V. Putten, "Cerebral recovery index: Reliable help for prediction of neurologic outcome after cardiac arrest," Critical Care Medicine, vol. 45, pp. e789–e797, 8 2017.

[36] J. Elmer, B. L. Jones, and D. S. Nagin, "Comparison of parametric and nonparametric methods for outcome prediction using longitudinal data after cardiac arrest," Resuscitation, vol. 148, pp. 152–160, 3 2020.

[37] S. Hofmann, M. Gaebler, and H. S. Scholte, "Finding traces of neurophenomenology predicting self-reported arousal trajectories using lstm recurrent neural networks on ssd extracted alpha-components of the eeg signal recorded during virtual roller coaster rides research report," 2018.

[38] P. Kaushik, A. Gupta, P. P. Roy, and D. P. Dogra, "Eeg-based age and gender prediction using deep blstm-lstm network model," IEEE Sensors Journal, vol. 19, pp. 2634–2641, 4 2019.

[39] K. Tsiouris, V. C. Pezoulas, M. Zervakis, S. Konitsiotis, D. D. Koutsouris, and D. I. Fotiadis, "A long short-term memory deep learning network for the prediction of epileptic seizures using eeg signals," Computers in Biology and Medicine, vol. 99, pp. 24–37, 8 2018.

[40] P. Wang, A. Jiang, X. Liu, J. Shang, and L. Zhang, "Lstm-based eeg classification in motor imagery tasks," IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 26, pp. 2086–2095, 11 2018.

[41] J. D. Kelleher, Deep Learning. The MIT Press, 2019.

[42] P. Rivas, Deep Learning for Beginners. Packt Publishing, 2020.

[43] C. Olah, "Understanding lstm networks." https://colah.github.io/posts/2015-08-Understanding-LSTMs/, Aug 2017.

[44] M. U. Abbasi, A. Rashad, A. Basalamah, and M. Tariq, "Detection of epilepsy seizures in neonatal eeg using lstm architecture," IEEE Access, vol. 7, pp. 179074–179085, 2019.

[45] N. Michielli, U. R. Acharya, and F. Molinari, "Cascaded lstm recurrent neural network for automated sleep stage classification using single-channel eeg signals," Computers in Biology and Medicine, vol. 106, pp. 71–81, 3 2019.

[46] G. Zhang, V. Davoodnia, A. Sepas-Moghaddam, Y. Zhang, and A. Etemad, "Classification of hand movements from eeg using a deep attention-based lstm network," IEEE Sensors Journal, vol. 20, pp. 3113–3122, 3 2020.

[47] P. Nagabushanam, S. T. George, and S. Radha, "Eeg signal classification using lstm and improved neural network algorithms," Soft Computing, 2019.

[48] Z. Ni, A. C. Yuksel, X. Ni, M. I. Mandel, and L. Xie, "Confused or not confused?: Disentangling brain activity from eeg data using bidirectional lstm recurrent neural networks," ACM-BCB 2017 - Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 241–246, 8 2017.

[49] X. Xing, Z. Li, T. Xu, L. Shu, B. Hu, and X. Xu, "Sae+lstm: A new framework for emotion recognition from multi-channel eeg," Frontiers in Neurorobotics, vol. 13, 2019.

[50] A. Craik, Y. He, and J. L. Contreras-Vidal, "Deep learning for electroencephalogram (eeg) classification tasks: A review," Journal of Neural Engineering, vol. 16, 2019.

[51] M. M. Admiraal, A. F. van Rootselaar, J. Hofmeijer, C. W. Hoedemaekers, C. R. van Kaam, H. M. Keijzer, M. J. van Putten, M. J. Schultz, and J. Horn, "Electroencephalographic reactivity as predictor of neurological outcome in postanoxic coma: A multicenter prospective cohort study," Annals of Neurology, vol. 86, pp. 17–27, 7 2019.

[52] M. M. Ghassemi, E. Amorim, T. Alhanai, J. W. Lee, S. T. Herman, A. Sivaraju, N. Gaspard, L. J. Hirsch, B. M. Scirica, S. Biswal, V. M. Junior, S. S. Cash, E. N. Brown, R. G. Mark, and M. B. Westover, "Quantitative electroencephalogram trends predict recovery in hypoxic-ischemic encephalopathy," Critical care medicine, vol. 47, pp. 1416–1423, 10 2019.

[53] F. Tadel, S. Baillet, J. C. Mosher, D. Pantazis, and R. M. Leahy, "Brainstorm: A user-friendly application for meg/eeg analysis," Computational Intelligence and Neuroscience, vol. 2011, 2011.

[54] S. Lee, X. Zhao, K. A. Davis, A. A. Topjian, B. Litt, and N. S. Abend, "Quantitative eeg predicts outcomes in children after cardiac arrest," Neurology, vol. 92, pp. E2329–E2338, 5 2019.

[55] J. Brownlee, "Logistic regression for machine learning." https://machinelearningmastery.com/logistic-regression-for-machine-learning/, Aug 2020.

[56] S. Glen, "Variance inflation factor." https://www.statisticshowto.com/variance-inflation-factor/, Dec 2020.

[57] I. Staff, "Variance inflation factor (vif)." https://www.investopedia.com/terms/v/variance-inflation-factor.asp, Mar 2018.

[58] Y. Sugomori, K. Boštjan, S. F. M., and A. M. F. Souza, Deep learning: practical neural networks with Java: build and run intelligent applications by leveraging key Java machine learning libraries. Packt Publishing, 2017.

[59] F. Chollet, Deep learning with Pyhton. Manning Publications Co., 2018.

[60] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. The MIT Press, 2017.

[61] C. E. Shannon and W. Weaver, "The mathematical theory of communication," Bell Syst Tech J, vol. 27, pp. 379–423, 1948.

[62] N. V. Thakor and S. Tong, "Advances in quantitative electroencephalogram analysis methods," Annual Review of Biomedical Engineering, vol. 6, pp. 453–495, 2004.

[63] C. Tsallis, "Possible generalization of boltzmann-gibbs statistics," Journal of Statistical Physics, vol. 52, pp. 479–487, 1988.

[64] R. G. Geocadin, T. Muthuswamy, D. L. Sherman, V. Thakor, and D. F. Hanley, "Early electrophysiological and histologic changes after global cerebral ischemia in rats," Movement Disorders, vol. 15, pp. 14–21, 2000.

[65] J. G. Proakis and D. G. Manolakis, Digital Signal Processing. Pearson Prentice Hall, 4 ed., 2007.

[66] B. Hjorth, "Technical contribution an on-line transformation of eeg scalp potentials into orthogonal source derivations," Electroencephalography and Clinical Neurophysiology, vol. 39, pp. 526–530, 1975.

[67] S.-H. Oh, Y.-R. Lee, and H.-N. Kim, "A novel eeg feature extraction method using hjorth parameter," International Journal of Electronics and Electrical Engineering, pp. 106–110, 2014.

[68] R. Hegger and H. Kantz, "Improved false nearest neighbor method to detect determinism in time series data," Physical review, vol. 60, pp. 4970–4973, 1999.

[69] M. B. Kennel, R. Brown, and H. D. I. Abarbanel, "Determining embedding dimension for phase-space reconstruction using a geometrical construction," PHYSICAL REVIEW A, vol. 45, pp. 3403–3411, 1992.

[70] J. E. Box, G. Jenkins, G. Reisel, and G. Ljung, Autoregressive processes. John Wiley & Sons, 5 ed., 2016.

[71] S. Mack, E. Kandel, J. Schwartz, T. Jessell, J. Schwartz, S. Siegelbaum, and A. Hudspeth, Principles of neural science. McGraw-Hill, 5 ed., 2013.

[72] E. N. Marieb and K. Hoehn, Human anatomy & physiology. Pearson, 9 ed., 2019.

[73] P. L. Nunez and R. Srinivasan, Electric Fields of the Brain The Neurophysics of EEG. Oxford University Press, 2 ed., 2006.

[74] C. J. Stam, G. Nolte, and A. Daffertshofer, "Phase lag index: Assessment of functional connectivity from multi channel eeg and meg with diminished bias from common sources," Human Brain Mapping, vol. 28, pp. 1178–1193, 11 2007.

[75] J. V. Tu, "Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes," J CLIN EPIDEMIOL, vol. 49, p. 12251231, 1996.

[76] J. M. Hilbe, Practical Guide to Logistic Regression. Chapman and Hall, 2015.

[77] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," 6 2012.

[78] J. Patterson and A. Gibson, Deep learning: a practitioners approach. OReilly, 2017.

[79] K. Team, "Keras documentation: Keras api reference." https://keras.io/api/, journal=Keras.

[80] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, pp. 1735–1780, 1997.

[81] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning t o forget: Continual prediction with lstm," Artificial Neural Networks, vol. 7-10, pp. 850–855, 1999.

[82] A. Gomez, "Backpropogating an lstm: A numerical example." https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9, Sep 2018.

[83] A. Graves, Supervised sequence labelling with recurrent neural networks. Springer, 2014.

[84] J. Brownlee, "How to evaluate the skill of deep learning models." https://machinelearningmastery.com/evaluate-skill-deep-learning-models/, May 2017.

[85] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," Nature Machine Intelligence, vol. 1, pp. 206–215, 5 2019.

[86] S. Alhagry, A. A. Fahmy, and R. A. El-Khoribi, "Emotion recognition based on eeg using lstm recurrent neural network," IJACSA) International Journal of Advanced Computer Science and Applications, vol. 8, 2017.

[87] J. Hofmeijer, M. C. Tjepkema-Cloostermans, and M. J. van Putten, "Burst-suppression with identical bursts: A distinct eeg pattern with poor outcome in postanoxic coma," Clinical Neurophysiology, vol. 125, pp. 947–954, 2014.

[88] L. M. Gladence, M. Karthi, and V. M. Anu, "A statistical comparison of logistic regression and different bayes classification methods for machine learning," ARPN Journal of Engineering and Applied Sciences, vol. 10, pp. 5947–5953, 2015.

[89] A. Ng and M. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in Advances in Neural Information Processing Systems (T. Dietterich, S. Becker, and Z. Ghahramani, eds.), vol. 14, pp. 841–848, MIT Press, 2002.

[90] K. Kirasich, T. . Smith, and B. Sadler, "Random forest vs logistic regression: Binary classification for heterogeneous datasets," Recommended Citation Kirasich, vol. 1, p. 9, 2018.

[91] D. L. Schomer and L. d. S. F. Henrique, Niedermeyers electroencephalography: basic principles, clinical applications, and related fields. Lippincott William & Wilkins, 2011.

[92] P. Stammet, D. R. Wagner, G. Gilson, and Y. Devaux, "Modeling serum level of s100β and bispectral index to predict outcome after cardiac arrest," Journal of the American College of Cardiology, vol. 62, pp. 851–858, 8 2013.

[93] M. R. Borich, K. E. Brown, B. Lakhani, and L. A. Boyd, "Applications of electroencephalography to characterize brain activity: Perspectives in stroke," Journal of Neurologic Physical Therapy, vol. 39, pp. 43–51, 1 2015.

[94] F. L. D. Silva, EEG: Origin and measurement. Springer Berlin Heidelberg, 2010.

[95] M. Teplan, "Fundamental of eeg measurement," MEASUREMENT SCIENCE REVIEW, vol. 2, 2002.

[96] M. J. A. M. V. Putten and J. Hofmeijer, "Eeg monitoring in cerebral ischemia: Basic concepts and clinical applications," Clinical Neurophysiology, vol. 33, pp. 203–210, 2016.

[97] J. W. Britton and D. Greer, "Eeg and cardiac arrest," Neurology, vol. 86, pp. 1470–1471, 4 2016.

[98] G. Buzsáki, Rhythms of the Brain. 2006.

[99] C.-H. Im, Computational EEG Analysis Methods and Applications. Springer Nature, 2018.

[100] E. Boran, J. Sarnthein, N. Krayenbühl, G. Ramantani, and T. Fedele, "High-frequency oscillations in scalp eeg mirror seizure frequency in pediatric focal epilepsy," Scientific Reports, vol. 9, 12 2019.

[101] M. T. Kucewicz, J. Cimbalnik, J. Y. Matsumoto, B. H. Brinkmann, M. R. Bower, V. Vasoli, V. Sulc, F. Meyer, W. R. Marsh, S. M. Stead, and G. A. Worrell, "High frequency oscillations are associated with cognitive processing in human recognition memory," Brain, vol. 137, pp. 2231–2244, 2014.

[102] G. Buzsáki and F. L. da Silva, "High frequency oscillations in the intact brain," Progress in Neurobiology, vol. 98, pp. 241–249, 9 2012.

[103] M. Zijlmans, P. Jiruska, R. Zelmann, F. S. Leijten, J. G. Jefferys, and J. Gotman, "High-frequency oscillations as a new biomarker in epilepsy," Annals of Neurology, vol. 71, pp. 169–178, 2 2012.

[104] J. Borjigin, U. C. Lee, T. Liu, D. Pal, S. Huff, D. Klarr, J. Sloboda, J. Hernandez, M. M. Wang, and G. A. Mashour, "Surge of neurophysiological coherence and connectivity in the dying brain," Proceedings of the National Academy of Sciences of the United States of America, vol. 110, pp. 14432–14437, 8 2013.

[105] T. J. Losasso, D. A. Muzzi, F. B. Meyer, and F. W. Sharbrough, "Electroencephalographic monitoring of cerebral function during asystole and successful cardiopulmonary resuscitation," Anesth Analg, vol. 75, p. 10214, 1992.

[106] J. Moss and M. Rockoff, "Eeg monitoring during cardiac arrest and resuscitation," JAMA, vol. 244, pp. 2750–2751, 1980.

[107] H. Clute and W. Levy, "Electroencephalographic changes during brief cardiac arrest in humans.anesthesiology," Anestesiology, vol. 73, pp. 821–825, 1990.

[108] J. Hofmeijer and M. J. A. M. V. Putten, "Ischemic cerebral damage an appraisal of synaptic failure," Stroke . 2012, vol. 43, pp. 607–615, 2012.

[109] E. O. Jørgensen and S. Holm, "The natural course of neurological recovery following cardiopulmonary resuscitation," Resuscitation, vol. 36, pp. 111–122, 1998.

[110] E. O. Jørgensen and A. Malchow-Møller, "Natural history of global and critical brain ischaemia part i: Eeg and neurological signs during the first year after cardiopulmonary resuscitation in patients subsequently regaining consciousness," Resuscitation, vol. 9, pp. 133–153, 1981.

[111] E. O. Jørgensen and A. Malchow-Mujller, "Natural history of global and critical brain ischaemia part ii: Eeg and neurological signs in patients remaining unconscious after cardiopulmonary resuscitation," Resuscitation, vol. 9, pp. 155–174, 1981.

[112] L. J. Hirsch, R. P. Brenner, F. W. Drislane, E. So, P. W. Kaplan, K. G. Jordan, S. T. Herman, S. M. Laroche, B. Young, T. P. Bleck, M. L. Scheuer, and R. G. Emerson, "The acns subcommittee on research terminology for continuous eeg monitoring: Proposed standardized terminology for rhythmic and periodic eeg patterns encountered in critically ill patients," Journal of Clinical Neurophysiology, vol. 22, pp. 128–135, 2005.

[113] C. Sandroni, A. Cariou, F. Cavallaro, T. Cronberg, H. Friberg, C. Hoedemaekers, J. Horn, J. P. Nolan, A. O. Rossetti, J. Soar, C. Sandroni, F. Cavallaro, A. Cariou, H. Friberg, J. Horn, J. P. Nolan, and A. O. Rossetti, "Resuscitation and intensive care medicine. originally published in resuscitation," Intensive Care Med, vol. 40, pp. 1816–1831, 2014.

[114] L. Sondag, B. J. Ruijter, M. C. Tjepkema-Cloostermans, A. Beishuizen, F. H. Bosch, J. A. van Til, M. J. van Putten, and J. Hofmeijer, "Early eeg for outcome prediction of postanoxic coma: Prospective cohort study with cost-minimization analysis," Critical Care, vol. 21, 5 2017.

[115] E. F. Wijdicks, A. Hijdra, G. B. Young, C. L. Bassetti, and S. Wiebe, "Practice parameter: Prediction of outcome in comatose survivors after cardiopulmonary resuscitation (an evidence-based review). report of the quality standards subcommittee of the american academy of neurology," Neurology, vol. 67, pp. 203–210, 2006.

[116] T. Sweet and T. Rallings, "Recurrent neural network-based approach for classifying audio/visual stimuli eeg data."

[117] L. H. Kuller, "Cardiovascular diseases sudden death-definition and epidemiologic considerations," Progress in cardiovascular diseases, vol. 13, 1980.

[118] K. Patel and J. Hipskind, "Cardiac arrest," StatPearls, 2020.

[119] N. Boulé-Laghzali, L. D. Pérez, K. Dyrda, J. F. Tanguay, M. Chabot-Blanchet, Y. Lamarche, D. Parent, A. F. Dupriez, A. Deschamps, and A. Ducharme, "Targeted temperature management after cardiac arrest: The montreal heart institute experience," CJC Open, vol. 1, pp. 238–244, 9 2019.

[120] C. A.Z., A. Rabinstein, J. Fugate, J. Mandrekar, E. Wijdicks, R. White, and J. Britton, "Continuous eeg in therapeutichypothermia after cardiac arrest," Neurology, vol. 80, pp. 339–344, 2013.

[121] J. Dankiewicz, T. Cronberg, G. Lilja, J. C. Jakobsen, J. Bělohlávek, C. Callaway, A. Cariou, G. Eastwood, D. Erlinge, J. Hovdenes, M. Joannidis, H. Kirkegaard, M. Kuiper, H. Levin, M. P. Morgan, A. D. Nichol, P. Nordberg, M. Oddo, P. Pelosi, C. Rylander, M. Saxena, C. Storm, F. Taccone, S. Ullén, M. P. Wise, P. Young, H. Friberg, and N. Nielsen, "Targeted hypothermia versus targeted normothermia after out-of-hospital cardiac arrest (ttm2): A randomized clin-

ical trial—rationale and design," American Heart Journal, vol. 217, pp. 23–31, 11 2019.

[122] L. Leuchs, "Press release choosing your reference- and why it matters," Brain Products, 2019.

[123] J. Brownlee, "How to scale data for long short-term memory networks in python." https://machinelearningmastery.com, Jul 2017.

[124] A. J. Seely and P. T. MacKlem, "Complex systems and the technology of variability analysis," Critical Care, vol. 8, pp. R367–R384, 12 2004.

[125] S. Tong, A. Bezerianos, J. Paul, Y. Zhu, and N. Thakor, "Nonextensive entropy measure of eeg following brain injury from cardiac arrest," Physica A, vol. 305, pp. 619–628, 2002.

[126] D. Contreras, A. Destexhe, T. J. Sejnowski, and M. Steriade, "Spatiotemporal patterns of spindle oscillations in cortex and thalamus," The Journal of Neuroscience, vol. 17, pp. 1179–1196, 1997.

[127] T. M. Cover and J. A. Thomas, Elements of Information Theory. John Wiley and Sonse Interscience, 1991.

[128] MathWorks, "Matlab and econometrics toolbox," 2020a.

[129] MathWorks, "Matlab," 2020a.

[130] D. A. Adamos, S. I. Dimitriadis, and N. A. Laskaris, "Towards the bio-personalization of music recommendation systems: a single-sensor eeg biomarker of subjective music preference."

[131] H. Jaseja, "Eeg spike versus eeg sharp wave: Differential clinical significance in epilepsy," Epilepsy and Behavior, vol. 25, p. 137, 9 2012.

[132] M. Beudel, M. C. Tjepkema-Cloostermans, J. H. Boersma, and M. J. A. M. V. Putten, "Small-world characteristics of eeg patterns in postanoxic encephalopathy neurocritical and neurohospitalist care," Frontiers in neurology, vol. 5, 2014.

[133] V. Daniel, "vif(x)." https://www.mathworks.com/matlabcentral/fileexchange/60551-vif-x, Dec 2016.

[134] D. A. Belsley and E. Kuh, Regression diagnostics. Wiley, 1980.

[135] M. H. Kutner, C. J. Nachtsheim, and J. Never, Applied linear regression models. McGraw-Hill/Irwin, 4 ed., 2004.

[136] A. Baratloo, M. Hosseini, A. Negida, and G. E. Ashal, "Part 1: Simple definition and calculation of accuracy, sensitivity and specificity," Emergency, vol. 3, pp. 48–49, 2015.

[137] Y. Zhao and Y. Cen, Data mining applications with R. Academic Press, 2014.

[138] J. Czakon, "F1 score vs roc auc vs accuracy vs pr auc: Which evaluation metric should you choose?." https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc#5, Dec 2020.

[139] J. Brownlee, "Failure of classification accuracy for imbalanced class distributions." https://machinelearningmastery.com, Jan 2020.

[140] F. Rosenblatt, "The perceptron: A perceiving and recognizing automaton (project para)," 1 1957.

[141] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," JMLR, vol. 9, pp. 249–256, 2010.

[142] N. Qian, "On the momentum term in gradient descent learning algorithms," Neural Networks, vol. 12, pp. 145–151, 1999.

[143] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," JMLR: W&CP, 2013.

[144] J. Brownlee, "How to control the stability of training neural networks with the batch size." https://machinelearningmastery.com, Jan 2019.

[145] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," 4 2018.

[146] D. Bakhuis, "A logistic regression from scratch." https://towardsdatascience.com, 2017.

[147] A. Ng, "Neural networks and deep learning." https://www.coursera.org/learn/neural-networks-deep-learning, 2018.

[148] S. Sharma, "Activation functions in neural networks." https://towardsdatascience.com, Feb 2019.

[149] J. Brownlee, "A gentle introduction to logistic regression with maximum likelihood estimation." https://machinelearningmastery.com, Oct 2019.

[150] C. Versloot, "How to use binary & categorical crossentropy with keras?." https://www.machinecurve.com/index.php/2019/10/22, Oct 2019.

[151] C. Versloot, "Extensions to gradient descent: from momentum to adabound." https://www.machinecurve.com/index.php/2019/11/03, Nov 2019.

[152] S. Ruder, "An overview of gradient descent optimization algorithms," 9 2016.

[153] J. Brownlee, "Understand the impact of learning rate on neural network performance," Jan 2019.

[154] V. Bushaev, "Stochastic gradient descent with momentum." https://towardsdatascience.com, Dec 2017.

[155] Y. Jung, "Multiple predicting k-fold cross-validation for model selection," Journal of Nonparametric Statistics, vol. 30, pp. 197–215, 1 2018.

[156] Wikimedia, "Hyperbolic tangent." https://commons.wikimedia.org/wiki/File:Hyperbolic_Tangent.svg, 2020.

[157] J. Heaton, Introduction to Neural Networks for Java. Heaton Research, 2 ed., 2008.

[158] D. Stathakis, "How many hidden layers and nodes?," International Journal of Remote Sensing, vol. 30, pp. 2133–2147, 4 2009.

[159] H. Larochelle, Y. Bengio, J. Louradour, and L. U. Ca, "Exploring strategies for training deep neural networks pascal lamblin," Journal of Machine Learning Research, vol. 1, pp. 1–40, 2009.

[160] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," 11 2016.

[161] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," JMLR: W&CP, vol. 15, pp. 315–321, 2011.

[162] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2 2015.

[163] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, pp. 436–444, 5 2015.

[164] K. Sarkar, "Relu : Not a differentiable function: Why used in gradient based optimization?." https://medium.com/@kanchansarkar, May 2018.

[165] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," 12 2014.

[166] J. Mueller and L. Massaron, Deep learning for dummies. John Wiley & Sons, Inc., 2019.

[167] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," Journal of Machine Learning Research, vol. 15, pp. 1929–1958, 2014.

[168] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini--batch gradient descent."

[169] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," 12 2015.

[170] J. Brownlee, "Dropout regularization in deep learning models with keras." https://machinelearningmastery.com, Jun 2016.

[171] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," pp. 950–957, 1992.

[172] J. Brownlee, "A gentle introduction to weight constraints in deep learning." https://machinelearningmastery.com, Nov 2018.

[173] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks," 10 2015.

[174] T. Cooijmans, N. Ballas, C. Laurent, Çağlar Gülçehre, and A. Courville, "Recurrent batch normalization," 3 2017.

[175] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 12 2014.

[176] J. Duchi and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," Journal of Machine Learning Research, vol. 12, pp. 2121–2159, 2011.

[177] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 12 2012.

[178] T. Dozat, "Incorporating nesterov momentum into adam," 2015.

[179] A. Karpathy, "Cs231n: Convolutional neural networks for visual recognition." http://cs231n.stanford.edu/, 2020.

[180] A. Talos, "Hyperparameter optimization with talos." https://github.com/autonomio/talos, 2019.

[181] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," 2 2019.

[182] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," 4 2018.

[183] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to sgd," 12 2017.

[184] M. uhn and K. Johnson, Applied Predictive Modeling. Springer Science and Business, 2013.

Appendix Z.1 Feature extraction

Contents

- Feature extraction all subjects
- Split EEG epoch into fragments of 10 sec
- Create empty matrices to store features
- FEATURES %%
- Features calculated for whole EEG epoch (not per fragment)
- Feature #7 - False Nearest Neighbors
- Features calculated per fragment %%
- Feature #1 - Shannon Entropy (complexity)
- Feature #2 - Tsallis Entropy (q=2) (complexity)
- Feature #3 and #4 - Cepstrum coefficients (complexity)
- Feature #5 #6 - Hjorth Parameters of mobility and complexity (complexity)
- Feature #7 - False Nearest Neighbors (complexity)
- Feauture #8 #9 - Autoregressive coefficients (complexity)
- Feature #14 - Total signal power (category)
- Feature #10 Delta Band power / total signal power (category)
- Feature #11 Theta Band power / total signal power (category)
- Feature #12 Alpha Band power / total signal power (category)
- Feature #13 Beta Band power / total signal power (category)
- Feature #15 Regularity (category)
- Feature #16 - Number of epilepticform spikes (category)
- Feature #17 - Burst Supression Ratio (category)
- Feature #18 - Coherence in the delta band (connectivity)
- Feature #19 - Phase Lag Index (connectivity)
- Feature matrix per fragment %%
- GENERATE FEATURE MATRIX PER EPOCH %%
- Table of feature matrix per epoch
- Save feature matrix per subject
- Store features averaged across all fragements per subject
- OPTIONAL PLOTS %%
- Plot a selected EEG fragment (1) or specific time period (2)
- Boxplot features of all fragments
- Plots to check feature changes throughout an epoch
- Boxplots to check variability of features between channels

Author: L.M. van Poppel 12/2020

```
% Feature extraction code:
    % Specify EEG recoring time post CA (12 or 24)
    % LOOP PER SUBJECT:
    % 1. Load preprocessed data from a subject
    % 2. Split 5 minute epoch in 30 time fragments of 10 seconds
    % 3. Extract features per channel that are extracted from 5 minute epoch
    % 4. Average features across all channels
    %   LOOP PER FRAGMENT
    %   1. Extract features per channel that are extracted from fragments
    %   2. Average features from across all channels
    %   3. Stack features matrices per fragment
    % 5. Create matrix with all features x all fragments
    % 6. Save feature matrix of subject

% plots at the end of the script used to analyze data
% supporting functions are found at
% https://github.com/deskool/ComaPrognosticanUsingEEG/tree/master/Supporting%20Functions

clear all; close all; clc;
```

Feature extraction all subjects

```
% variables
Fs = 128;       % sample rate
channels = 8;   % longitudinal bipolar
features = 22;  % 21 features and 1 outcome
fragments = 30; % 30 fragments of 10 sec

% choose EEG time
EEGtime = "_24";

% load excel with clinical characteristics
excelpac = "Clinical";
excelfile = ".xlsx";
excel_id = strcat(excelpac,EEGtime,excelfile);
clinical = readtable(excel_id);

all_subjects = height(clinical);
```

```
features_all_subjects = zeros(all_subjects,features);

% loop through all subjects in time frame
for s = 1:all_subjects
```

```
    subjectnumber = clinical.StudyID(s,1);

% define strings
subject = strcat(subjectnumber,EEGtime); % AMC000_time
MLfile = ".mat";
file_subject = strcat(subject,MLfile);   % AMC000_time.mat
load (file_subject);

% find clinical features of specific subject
subjectrow = find(strcmp(clinical.StudyID, subjectnumber)==1);

% EEG epoch 5 min (8 channels, 5min x 128Hz)
% avg = loaded matrix in workspace
datasubject = avg;

duration = (1:size(avg,2)); %datapoints in datasubject
signal = 1000000*avg';      %volts to microvolts
dp10 = Fs*10;               %number of datapoints in a 10 sec fragment

% strings used to save resulting feature matrix
subject_features = "_features";
subID = strcat(subject,subject_features); %AMC000_time_features
```

Split EEG epoch into fragments of 10 sec

```
% split the number of columns by 30 and replicate 30 times
coldist = size(datasubject, 2) / fragments * ones(1, fragments);

% 30 fragments of 8x1280 matrices
splitmats = mat2cell(datasubject, channels, coldist);
```

Create empty matrices to store features

```
% empty feature matrix to store all features of all channels per fragment
features_fragment = zeros(features,channels);
% empty feature matrix of all fragments
features_all = zeros(features, channels, fragments);

% features are averaged across all channels to obtain new feature matrix
% feature matrix per fragment contains averaged feature value

% create empty feature matrix of all features of all fragments
% features matrix contains averaged features of all fragments
features_epoch = zeros(features,fragments);
```

FEATURES %%

```
%%%%%%%%%%%%%%
```

Features calculated for whole EEG epoch (not per fragment)

```
% Clinical features
%age
age = clinical(subjectrow,5);
age = age{1,1};

%sex 1=male ; 2=female
sex = clinical(subjectrow,4);
sex = sex{1,1};

% CPC
cpc = clinical(subjectrow, 3);
cpc = cpc{1,1};
```

Feature #7 - False Nearest Neighbors

modified from code Ghassemi et al., 2019

```
 FNN_matrix = zeros(1,channels);
for i = 1:channels
    npts = 1280;        %10sec * Fs
    maxdims = 50;
    max_delay = 256;    %2sec * Fs
    distance_thresh = 0.5;

    %tt = ami(x(:,i),npts,max_delay);
    tt = ami(signal(:,i),npts,max_delay);
    [~,idx] = min(tt);
    %nn = false_neighbors_kd(x(:,i), idx, maxdims, npts, 1, 0, 0, 0);
    nn = false_neighbors_kd(signal(:,i), idx, maxdims, npts, 1, 0, 0, 0);
    mindim = find((mean(nn.z./nn.d > distance_thresh) < 0.1) == 1);
    FNN = mindim(1);
```

```
    FNN_matrix(1,i) = FNN;
end
```

Features calculated per fragment %%

```
% repeat the feature extraction for all fragments in the epoch
for f = 1:fragments
```

```
% extract fragment matrices
data_10sec = splitmats(f);
data_10sec = data_10sec{1,1};
data_10sec = 1000000*data_10sec;    % from volts to microvolts
x = data_10sec.';                   % 1280 samples x 8 channels input matrix

% minimum and maximum EEG amplitude
min_ampl_frag = min(data_10sec,[],'all'); %microvolts
max_ampl_frag = max(data_10sec,[],'all'); %microvolts
```

Feature #1 - Shannon Entropy (complexity)

A measure to quantify the uncertainty of a stochastic signal defined by Shannon and Weaver, 1949 modified from code Ghassemi et al., 2019

```
% empty matrix for fragment
Shannon_matrix = zeros(1,channels);

for i = 1:channels

    bin_min = -200; bin_max = 200; binWidth = 2;
    Shannon_entropy = CRI_ShannonEntropy(x(:,i), bin_min, bin_max, binWidth);

    % create matrix to store entropy from every channel
    Shannon_matrix(1,i) = Shannon_entropy;
end
```

Feature #2 - Tsallis Entropy (q=2) (complexity)

A measure to quantify the uncertainty of a stochastic signal in a nonextensive manner modified from code Ghassemi et al., 2019

```
Tsallis_matrix = zeros(1,channels);

for i = 1 : channels

    bin_min = -200; bin_max = 200; binWidth = 2; q = 2;
    [~, prob] = CRI_ShannonEntropy(x(:,i), bin_min, bin_max, binWidth);
    tsallis = Tsallis_entro(prob',(q + 0.01));
    Tsallis_matrix(1,i) = tsallis;

end
```

Feature #3 and #4 - Cepstrum coefficients (complexity)

A measure to quantify the rate of change in different spectrum bands modified from code Ghassemi et al., 2019

```
cep_matrix = zeros(2,channels);

%Cepstrum coefficient 1
for i = 1 : channels
    % The number of components
    num_lcp = 2;

    %Take the autocorrelation
    hac = dsp.Autocorrelator;
    hac.MaximumLagSource = 'Property';
    hac.MaximumLag = num_lcp;           % Compute autocorrelation lags between [0:9]
    a = step(hac, x(:,i));

    % Run levinson solver to find LPC coefficeints.
    hlevinson = dsp.LevinsonSolver;
    hlevinson.AOutputPort = true;   % Output polynomial coefficients
    LPC = step(hlevinson, a);       % Compute LPC coefficients

    %Now convert to cepstral coefficents.
    hlpc2cc = dsp.LPCToCepstral('CepstrumLength',round(1.5*num_lcp));

    CC{i} = step(hlpc2cc, LPC); % Convert LPC to CC.
    cep1 = CC{i}(2);

    cep_matrix(1,i) = cep1;
end

%Cepstrum coefficient 2
for i = 1 : channels
    num_lcp = 2;

    %Take the autocorrelation
    hac = dsp.Autocorrelator;
    hac.MaximumLagSource = 'Property';
    hac.MaximumLag = num_lcp;           % Compute autocorrelation lags between [0:9]
```

```
    a = step(hac, x(:,i));

    % Run levinson solver to find LPC coefficeints.
    hlevinson = dsp.LevinsonSolver;
    hlevinson.AOutputPort = true;   % Output polynomial coefficients
    LPC = step(hlevinson, a);       % Compute LPC coefficients

    %Now convert to cepstral coefficents.
    hlpc2cc = dsp.LPCToCepstral('CepstrumLength',round(1.5*num_lcp));

    CC{i} = step(hlpc2cc, LPC); % Convert LPC to CC.
    cep2 = CC{i}(3);

    cep_matrix(2,i) = cep2;
end
```

Feature #5 #6 - Hjorth Parameters of mobility and complexity (complexity)

mobility (2nd Hjorth parameter) = indication of the proportion of the variance of the power spectrum complexity (3nd Hjorth parameter) = a measure that quantifies how much similarity between the shape of the signal and a pure sine wave modified from code Ghassemi et al., 2019

```
Hjorth_matrix = zeros(2,channels);

for i = 1:channels

    [mobility,complexity] = HjorthParameters(x(:,i));
    Hjorth_matrix(1,i) = mobility;
    Hjorth_matrix(2,i) = complexity;

end
```

Feature #7 - False Nearest Neighbors (complexity)

% modified from code Ghassemi et al., 2019 calculated for entire epoch

Feauture #8 #9 - Autoregressive coefficients (complexity)

Estimated coefficients at t-1 and t-2 of a 2nd order AR model given the EEG signal modified from code Ghassemi et al., 2019 use the Econometrics Toolbox

```
arma_matrix = zeros(2,channels);

for i = 1:channels

    mod{i} = arima(2,0,0);
    arma_mod{i} = estimate(mod{i},x(:,i));
    arma1 = arma_mod{i}.AR{1};

    mod{i} = arima(2,0,0);
    arma_mod{i} = estimate(mod{i},x(:,i));
    arma2 = arma_mod{i}.AR{2};

    arma_matrix(1,i) = arma1;
    arma_matrix(2,i) = arma2;

end
```

Feature #14 - Total signal power (category)

```
power_matrix = zeros(1,channels);

for i = 1:channels
    power = bandpower(x(:,i),Fs,[0.5,30]);
    power_matrix(1,i) = power;
end
```

Feature #10 Delta Band power / total signal power (category)

compromises frequencies between 0.5 – 4 Hz modified from code Ghassemi et al., 2019

```
delta_matrix = zeros(1,channels);

for i = 1:channels
    delta = bandpower(x(:,i),Fs,[0.5,4]);
    delta_matrix(1,i) = delta;
end

% normalize to total power
delta_norm = delta_matrix./power_matrix;
```

Feature #11 Theta Band power / total signal power (category)

compromises frequencies between 4 – 7 Hz modified from code Ghassemi et al., 2019

```
theta_matrix = zeros(1,channels);

for i = 1:channels
    theta = bandpower(x(:,i),Fs,[4,7]);
    theta_matrix(1,i) = theta;
```

```
end

% normalize to total power
theta_norm = theta_matrix./power_matrix;
```

Feature #12 Alpha Band power / total signal power (category)

compromises frequencies between 8 – 13 Hz modified from code Ghassemi et al., 2019

```
alpha_matrix = zeros(1,channels);

for i = 1:channels
    alpha = bandpower(x(:,i),Fs,[8,13]);
    alpha_matrix(1,i) = alpha;
end

% normalize to total power
alpha_norm = alpha_matrix./power_matrix;
```

Feature #13 Beta Band power / total signal power (category)

compromises frequencies between 14 – 30 Hz modified from code Ghassemi et al., 2019

```
beta_matrix = zeros(1,channels);

for i = 1:channels
    beta = bandpower(x(:,i),Fs,[14,30]);
    beta_matrix(1,i) = beta;
end

% normalize to total power
beta_norm = beta_matrix./power_matrix;
```

Feature #15 Regularity (category)

A measure to quantify regularity in amplitude of the signal Formula from Tjepkema-Cloostermans et al. 2013 modified from code Ghassemi et al., 2019

```
reg_matrix = zeros(1,channels);

for i = 1:channels
    %square the signal
    in_x = x(:,i).^2;
    %find the filter length in samples (0.5 sec)
    num_wts = Fs/2;

    a = 1;
    wts = ones(1,num_wts)/num_wts;
    q = filter(wts,a,in_x);
    %sort the values of the smoothed signal in descending order
    q = sort(q,'descend');
    N = length(q);
    u = 1:N;

    %compute the regularity
    %var(q)
    regularity = sqrt(sum(u.^2.* q')/(sum(q)*N^2*1/3));
    reg_matrix(1,i) = regularity;

end
```

Feature #16 - Number of epilepticform spikes (category)

Count the number of epileptic form spikes A feature related to the epileptiform EEG pattern in post-anoxic coma modified from code Ghassemi et al., 2019

```
spike_matrix = zeros(1, channels);

for i = 1 : channels

    stds_away = 3;
    dealta_jumps = 2;
    %remove the mean from the signal and look for the spikes
    %Find a spikes lasting lt 70 ms = 9 samples
    %margin around minimal number of samples: 9 x 2 = 18
    [pks, locs]=findpeaks(x(:,i)-mean(x(:,i)),'MaxPeakWidth',18,'SortStr', 'descend');

    %make sure that it is more than 3 standard deviations from the median
    pk_index = locs(find(stds_away * std(x(:,i)) < pks));
    NUM_SPIKE = length(pk_index);
    spike_matrix(1,i) = NUM_SPIKE;

end
```

Feature #17 - Burst Supression Ratio (category)

The ratio between length of the fragment the that signal is below 5 microvolts and the total length of the fragment A feature related to the low voltage EEG pattern in post-anoxic coma formula obtained from paper Nagaraj et al. 2018

```
BSR_matrix = zeros (1, channels);

for i = 1 : channels
    supression_threshold = 5; %microvolts
    EEG = abs(x(:,i));
    supressions = EEG(EEG <= supression_threshold);

    BSR = (length(supressions) / length(EEG))*100;
    BSR_matrix(1,i) = BSR;

end
```

Feature #18 - Coherence in the delta band (connectivity)

Measure to quantify the degree of similarity in the delta band paramters used from Tjepkema-Cloostermans et al. 2013 modified from code Ghassemi et al., 2019

```
% empty matrix for coherence of channels with all other channels
dco_allchannels = zeros(channels,channels);

for i = 1:channels
   for i2 = 1:channels
   % coherence between all channel combinations

   % exclude coherence between same channels
   if    i==i2
         dco_allchannels(i,i2) = NaN;
   else

         overlap = 2 * Fs;  %data points
         window_length = 4; %seconds
         nfft = window_length * Fs; %data points
         [Cxy F] = mscohere(x(:,i),x(:,i2),hanning(nfft),overlap,nfft,Fs);
         delta_coherence = mean(Cxy(find(F >= 0.5 & F<=4)));

         dco_allchannels(i,i2) = delta_coherence;
   end
   end
end

% remove NaN
dco_nonan = zeros(7,channels);
for i = 1 : channels
    dco = dco_allchannels(:,i);
    dco = dco(~isnan(dco));
    dco_nonan(:,i) = dco;
end

% average over all channels
% returns a row vector containing the mean of each column
dco_matrix = mean(dco_nonan);
```

Feature #19 - Phase Lag Index (connectivity)

modified from code Ghassemi et al., 2019

```
pli_allchannels = zeros(channels, channels);

for i = 1 : channels
    for i2 = 1:channels

    % exclude coherence between same channels
    if    i==i2
          pli_allchannels(i,i2) = NaN;
    else

    hxi = hilbert(x(:,i));
    hxj = hilbert(x(:,i2));

    % calculating the INSTANTANEOUS PHASE
    inst_phasei = atan(angle(hxi));
    inst_phasej = atan(angle(hxj));

    pli_out = abs(mean(sign(inst_phasej - inst_phasei)));

    pli_allchannels(i,i2) = pli_out;
    end

    end
end

% remove NaN
pli_nonan = zeros(7,channels);
for i = 1 : channels
    pli = pli_allchannels(:,i);
    pli = pli(~isnan(pli));
    pli_nonan(:,i) = pli;
end

% average over all channels
% returns a row vector containing the mean of each column
pli_matrix = mean(pli_nonan);
```

Feature matrix per fragment %%

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% fill feature matrix with features
    % feature matrix consist of [features x channels] per fragment
% calculate the mean of a feature across all channels
% obtain vector of mean features across all channels per fragment

% fill feature matrix with shannon entropy
features_fragment(1,:) = Shannon_matrix;
meanf1 = mean(Shannon_matrix);

% fill feature matrix with tsallis entropy
features_fragment(2,:) = Tsallis_matrix;
meanf2 = mean(Tsallis_matrix);

% fill feature matrix with cepstrum coefficients
features_fragment(3:4,:) = cep_matrix;
meanf3 = mean(cep_matrix(1,:));
meanf4 = mean(cep_matrix(2,:));

% fill feature matrix with Hjorth mobility and complexity
features_fragment(5:6,:) = Hjorth_matrix;
meanf5 = mean(Hjorth_matrix(1,:)); %mobility
meanf6 = mean(Hjorth_matrix(2,:)); %complexity

% fill feature matrix with FNN
features_fragment(7,:) = FNN_matrix;
meanf7 = mean(FNN_matrix);

% fill feature matrix with ARMA
features_fragment(8:9,:) = arma_matrix;
meanf8 = mean(arma_matrix(1,:));
meanf9 = mean(arma_matrix(2,:));

% fill feature matrix with delta normalized power
features_fragment(10,:) = delta_norm;
meanf10 = mean(delta_norm);

% fill feature matrix with theta normalized power
features_fragment(11,:) = theta_norm;
meanf11 = mean(theta_norm);

% fill feature matrix with alpha normalized power
features_fragment(12,:) = alpha_norm;
meanf12 = mean(alpha_norm);

% fill feature matrix with beta power normalized power
features_fragment(13,:) = beta_norm;
meanf13 = mean(beta_norm);

% fill feature matrix with signal power
features_fragment(14,:) = power_matrix;
meanf14 = mean(power_matrix);

% fill feature matrix with regularity
features_fragment(15,:) = reg_matrix;
meanf15 = mean(reg_matrix);

% fill feature matrix with epileptic spikes
features_fragment(16,:) = spike_matrix;
meanf16 = mean(spike_matrix);

% fill feature matrix with BSR
features_fragment(17,:) = BSR_matrix;
meanf17 = mean(BSR_matrix);

% fill feature matrix with delta coherence
features_fragment(18,:) =  dco_matrix;
meanf18 = mean( dco_matrix);

% fill feature matrix with phase lag index
features_fragment(19,:) =  pli_matrix;
meanf19 = mean(pli_matrix);

% fill feature matrix with age
features_fragment(20,:) = age * ones(1,channels);

% fill feature matrix with sex
features_fragment(21,:) = sex * ones(1,channels);

% fill feature matrix with CPC
features_fragment(22,:) = cpc * ones(1,channels);


%TABLE of feature matrix (features)x(channels)
%row names
feature = {'Shannon Entropy','Tsallis Entropy','Ceptrum 1','Ceptrum 2',...
           'Hjorth Mobility','Hjorth Complexity','FNN', 'Arma1', 'Arma2',...
           'Delta','Theta','Alpha','Beta','Signal Power','Regularity',...
           'Number Epileptic Spikes','Burst Suppression Ratio',...
           'Delta Coherence','Phase Lag Index','Age','Sex','CPC'};
%column names
channelnumber = {'Fp1T3','T3O1','Fp2T4','T4O2','Fp1C3','C3O1','Fp2C4','C4O2'};
%table features per channel
features_channels_10sec = array2table(features_fragment,'RowNames',feature,'VariableNames',channelnumber);
```

```
%TABLE of mean feature matrix
%(mean features across all channels)x(1) of 1 fragment
%column name
channelname = {'allchannels'};
%complete vector
meanfeatures = [meanf1; meanf2; meanf3; meanf4; meanf5; meanf6; meanf7; meanf8; meanf9; meanf10; meanf11; meanf12; meanf13; meanf14; meanf15; mean
%table features averaged across all channels
meanfeatures_10sec = array2table(meanfeatures,'RowNames',feature,'VariableNames',channelname);

%3D structure of all feature_fragment matrices of all fragments
%(features)x(channels)x(fragments)
features_all(:,:,f) = features_fragment;
```

GENERATE FEATURE MATRIX PER EPOCH %%

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% store the mean features across all channels per fragment in a matrix
% [meanfeatures x fragments]
% gives feature matrix of an epoch
features_epoch(:,f) = meanfeatures;
```

```
end
```

Table of feature matrix per epoch

```
fragmentnumber = {'fragment1', 'fragment2', 'fragment3', 'fragment4', 'fragment5',...
                  'fragment6', 'fragment7', 'fragment8', 'fragment9', 'fragment10',...
                  'fragment11', 'fragment12', 'fragment13', 'fragment14', 'fragment15',...
                  'fragment16', 'fragment17', 'fragment18', 'fragment19', 'fragment20',...
                  'fragment21', 'fragment22', 'fragment23', 'fragment24', 'fragment25',...
                  'fragment26', 'fragment27', 'fragment28', 'fragment29', 'fragment30'};
features_epoch_table = array2table(features_epoch, 'RowNames', feature,'VariableNames',fragmentnumber);
```

Save feature matrix per subject

save as "AMC0000_time_features.mat"

```
save(subID, 'features_epoch')
```

Store features averaged across all fragements per subject

gives matrix of averaged features of epoch of all subjects [meanfeatures across fragment x subjects] used for feature selection and logistic regression

```
features_all_subjects(s,:)= mean(features_epoch.');
save('features_all_subjects')
```

```
end
```

OPTIONAL PLOTS %%

```
%%%%%%%%%%%%%%%%%%%%%%
```

Plot a selected EEG fragment (1) or specific time period (2)

```
% minimal and maximal amplitude
min_ampl_epoch = min(signal,[],'all'); %microvolts
max_ampl_epoch = max(signal,[],'all'); %microvolts

% option 1: plot EEG fragment
%define fragment
eeg_fragment = 10;
%define time to plot
start_frag = ((eeg_fragment-1)*dp10)+1;
end_frag = (eeg_fragment)*dp10;
frgm = start_frag : end_frag;

% option 2: plot specific time period
%define start and end time
%tstart = 10;   %sec
%tend = 35;     %sec

% plot signal of all channels
for i = 1:channels
```

```
   figure(1)
    % option 1: plot specific fragment
    subplot(channels,1,i);
    plot(duration(frgm), signal(frgm,i), 'b');

    % option 1: plot time period
    %plot(duration(tstart*Fs:tend*Fs), signal(tstart*Fs:tend*Fs,i), 'b');

    if abs(min_ampl_epoch) > abs(max_ampl_epoch)
    ylim ([min_ampl_epoch abs(min_ampl_epoch)])
    else
    ylim ([-max_ampl_epoch max_ampl_epoch])
    end

    ylabel ('Voltage (uV)')
    xlabel 'time * sample rate'

end
```

Boxplot features of all fragments

```
boxepoch = features_epoch.';
figure (2)
title ('Variability of features between fragments')

subplot(2,9,1)
boxplot(boxepoch(:,1))
xlabel('Shannon Entropy')

subplot(2,9,2)
boxplot(boxepoch(:,2))
xlabel('Tsallis Entropy')

subplot(2,9,3)
boxplot(boxepoch(:,3))
xlabel('Ceptrum 1')

subplot(2,9,4)
boxplot(boxepoch(:,4))
xlabel('Ceptrum 2')

subplot(2,9,5)
boxplot(boxepoch(:,5))
xlabel('Hjorth Mobility')

subplot(2,9,6)
boxplot(boxepoch(:,6))
xlabel('Hjorth Complexity')

subplot(2,9,7)
boxplot(boxepoch(:,7))
xlabel('FNN')

subplot(2,9,8)
boxplot(boxepoch(:,10))
xlabel('Delta')

subplot(2,9,9)
boxplot(boxepoch(:,11))
xlabel('Theta')

subplot(2,9,10)
boxplot(boxepoch(:,12))
xlabel('Alpha')

subplot(2,9,11)
boxplot(boxepoch(:,13))
xlabel('Beta')

subplot(2,9,12)
boxplot(boxepoch(:,14))
xlabel('Signal Power')

subplot(2,9,13)
boxplot(boxepoch(:,15))
xlabel('Regularity')

subplot(2,9,14)
boxplot(boxepoch(:,16))
xlabel('Spikes')

subplot(2,9,15)
boxplot(boxepoch(:,17))
xlabel('BSR')

subplot(2,9,16)
boxplot(boxepoch(:,18))
xlabel('Delta Coherence')

subplot(2,9,17)
boxplot(boxepoch(:,19))
xlabel('PLI')
```

Plots to check feature changes throughout an epoch

```
figure(3)
title ('Change of features through epoch')

subplot(2,9,1)
plot(boxepoch(:,1))
xlabel('Shannon Entropy')

subplot(2,9,2)
plot(boxepoch(:,2))
xlabel('Tsallis Entropy')

subplot(2,9,3)
plot(boxepoch(:,3))
xlabel('Ceptrum 1')

subplot(2,9,4)
plot(boxepoch(:,4))
xlabel('Ceptrum 2')

subplot(2,9,5)
plot(boxepoch(:,5))
xlabel('Hjorth Mobility')

subplot(2,9,6)
plot(boxepoch(:,6))
xlabel('Hjorth Complexity')

subplot(2,9,7)
plot(boxepoch(:,7))
xlabel('FNN')

subplot(2,9,8)
plot(boxepoch(:,10))
xlabel('Delta')

subplot(2,9,9)
plot(boxepoch(:,11))
xlabel('Theta')

subplot(2,9,10)
plot(boxepoch(:,12))
xlabel('Alpha')

subplot(2,9,11)
plot(boxepoch(:,13))
xlabel('Beta')

subplot(2,9,12)
plot(boxepoch(:,14))
xlabel('Signal Power')

subplot(2,9,13)
plot(boxepoch(:,15))
xlabel('Regularity')

subplot(2,9,14)
plot(boxepoch(:,16))
xlabel('Spikes')

subplot(2,9,15)
plot(boxepoch(:,17))
xlabel('BSR')

subplot(2,9,16)
plot(boxepoch(:,18))
xlabel('Delta Coherence')

subplot(2,9,17)
plot(boxepoch(:,19))
xlabel('PLI')
```

Boxplots to check variability of features between channels

within a fragment

```
boxfragment = specific_feature_fragment.';
figure(4)
title('Variability of features between channels');

subplot(2,9,1)
boxplot(boxfragment(:,1))
xlabel('Shannon Entropy')

subplot(2,9,2)
boxplot(boxfragment(:,2))
xlabel('Tsallis Entropy')

subplot(2,9,3)
boxplot(boxfragment(:,3))
xlabel('Ceptrum 1')

subplot(2,9,4)
```

```
boxplot(boxfragment(:,4))
xlabel('Ceptrum 2')

subplot(2,9,5)
boxplot(boxfragment(:,5))
xlabel('Hjorth Mobility')

subplot(2,9,6)
boxplot(boxfragment(:,6))
xlabel('Hjorth Complexity')

subplot(2,9,7)
boxplot(boxfragment(:,7))
xlabel('FNN')

subplot(2,9,8)
boxplot(boxfragment(:,10))
xlabel('Delta')

subplot(2,9,9)
boxplot(boxfragment(:,11))
xlabel('Theta')

subplot(2,9,10)
boxplot(boxfragment(:,12))
xlabel('Alpha')

subplot(2,9,11)
boxplot(boxfragment(:,13))
xlabel('Beta')

subplot(2,9,12)
boxplot(boxfragment(:,14))
xlabel('Signal Power')

subplot(2,9,13)
boxplot(boxfragment(:,15))
xlabel('Regularity')

subplot(2,9,14)
boxplot(boxfragment(:,16))
xlabel('Spikes')

subplot(2,9,15)
boxplot(boxfragment(:,17))
xlabel('BSR')

subplot(2,9,16)
boxplot(boxfragment(:,18))
xlabel('Delta Coherence')

subplot(2,9,17)
boxplot(boxfragment(:,19))
xlabel('PLI')
```

Published with MATLAB® R2020a

Appendix Z.2 High VIF feature exclusion

Contents

- PART 1 - Feature input selection
- Initialization
- Load Data
- Create binary classification of outcomes
- Normalize data in the range 0:1
- Include only qEEG features
- VIF round 1
- Remove highest VIF round 1
- VIF round 2
- Remove highest VIF round 2
- VIF round 3
- Remove highest VIF round 3
- VIF round 4
- Remove highest VIF round 4
- VIF round 5
- Remove highest VIF round 5
- VIF round 6
- Remove highest VIF round 6
- VIF round 7
- Remove highest VIF round 7
- VIF round 8
- PART 2 - Additional feature set 5
- Initialization
- Load Data
- create binary classification of outcomes
- normalize data in the range 0:1
- include only qEEG features with high predictive value
- VIF round 1
- Remove highest VIF round 1
- VIF round 2
- Remove highest VIF round 2
- VIF round 3
- Remove highest VIF round 3
- VIF round 4
- Remove highest VIF round 4
- VIF round 5

Author: L.M. van Poppel 12/2020

```
% VIF correction code: used to remove features with high variance inflation
% factor (>10). Features subsequently removed until all VIFs < 10

%VIFs are also the diagonal elements of the inverse of the correlation matrix [1]
%a convenient result that eliminates the need to set up the various regressions
%[1] Belsley, D. A., E. Kuh, and R. E. Welsch. Regression Diagnostics. Hoboken, NJ: John Wiley & Sons, 1980.
```

PART 1 - Feature input selection

create final feature set of qEEG features with low multicollinearity

Initialization

```
clear all; close all; clc
```

Load Data

load data t=12

```
data12 = load('Feature_Subjects_12.mat');
data12 = data12.features_all_subjects_12;
% load data t=24
data24 = load('Feature_Subjects_24.mat');
data24 = data24.features_all_subjects_24;

alldata = [data12; data24]; %combine data sets
```

Create binary classification of outcomes

0 = good outcome (CPC 1-2) 1 = poor outcome (CPC 3-5)

```
alldata(:,22) = alldata(:,22)>2;
```

Normalize data in the range 0:1

normalize operates on each column of data separately 'range' scales between 0:1

```
alldata_norm = normalize(alldata,'range');
```

Include only qEEG features

exclude the extracted clinical features: age(ft. 20) and sex(ft. 21)

```
excludeClinicalFeatures = [20 21];
alldata_norm(:,excludeClinicalFeatures) = [];
```

VIF round 1

```
X = alldata_norm(:,1:19);
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'SE';'TE';'C1';'C2';'HM';'HC';'FNN';'A1';'A2';'d';'t';...
            'a';'b';'SP';'R';'#S';'BSR';'dC';'PLI'};

% plot VIF scores
bar(VIF)
title('VIF of features round 1')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 1

```
X(:,featurenumber)=[];
% Cepstrum 2
```

VIF round 2

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'SE';'TE';'C1';'HM';'HC';'FNN';'A1';'A2';'d';'t';...
```

```
            'a';'b';'SP';'R';'#S';'BSR';'dC';'PLI'};

% plot VIF scores
bar(VIF)
title('VIF of features round 2')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 2

```
X(:,featurenumber)=[];
% ARMA 1
```

VIF round 3

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'SE';'TE';'C1';'HM';'HC';'FNN';'A2';'d';'t';...
            'a';'b';'SP';'R';'#S';'BSR';'dC';'PLI'};

% plot VIF scores
bar(VIF)
title('VIF of features round 3')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 3

```
X(:,featurenumber)=[];
% Delta power
```

VIF round 4

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'SE';'TE';'C1';'HM';'HC';'FNN';'A2';'t';...
            'a';'b';'SP';'R';'#S';'BSR';'dC';'PLI'};

% plot VIF scores
bar(VIF)
title('VIF of features round 4')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 4

```
X(:,featurenumber)=[];
% Shannon Entropy
```

VIF round 5

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'TE';'C1';'HM';'HC';'FNN';'A2';'t';...
            'a';'b';'SP';'R';'#S';'BSR';'dC';'PLI'};

% plot VIF scores
bar(VIF)
title('VIF of features round 5')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 5

```
X(:,featurenumber)=[];
% Hjorth Complexity
```

VIF round 6

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'TE';'C1';'HM';'FNN';'A2';'t';...
            'a';'b';'SP';'R';'#S';'BSR';'dC';'PLI'};

% plot VIF scores
bar(VIF)
title('VIF of features round 6')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 6

```
X(:,featurenumber)=[];
% Cepstrum 1
```

VIF round 7

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'TE';'HM';'FNN';'A2';'t';...
            'a';'b';'SP';'R';'#S';'BSR';'dC';'PLI'};

% plot VIF scores
bar(VIF)
title('VIF of features round 7')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 7

```
X(:,featurenumber)=[];
% Hjorth Mobility
```

VIF round 8

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'TE';'FNN';'A2';'t';...
            'a';'b';'SP';'R';'#S';'BSR';'dC';'PLI'};

% plot VIF scores
bar(VIF)
title('VIF of features round 8')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)

% all features have a score below 10
% excluded features:
% Shannon entropy (1)
% Cepstrum 1 (3)
% Cepstrum 2 (4)
% Hjorth mobility (5)
% Hjorth complexity (6)
% ARMA 1 (8)
% Normalized delta (10)
```

PART 2 - Additional feature set 5

create additional feature set 5 with qEEG features from additional feature set 4 with low multicollinearity

Initialization

```
clear all; close all; clc
```

Load Data

load data t=12

```
data12 = load('Feature_Subjects_12.mat');
data12 = data12.features_all_subjects_12;
% load data t=24
data24 = load('Feature_Subjects_24.mat');
data24 = data24.features_all_subjects_24;

alldata = [data12; data24]; %combine data sets
```

create binary classification of outcomes

0 = good outcome (CPC 1-2) 1 = poor outcome (CPC 3-5)

```
alldata(:,22) = alldata(:,22)>2;
```

normalize data in the range 0:1

normalize operates on each column of data separately 'range' scales between 0:1

```
alldata_norm = normalize(alldata,'range');
```

include only qEEG features with high predictive value

qEEG features excluded from the all features to create additional feature set 4:

```
        %8      ARMA 1
        %9      ARMA 2
        %10     Delta
        %11     Theta
        %12     Alpha
        %14     Signal power
        %15     Regularity
        %19     PLI
        %20     Age
        %21     Sex

AFS4 = alldata_norm;
excludefeatures = [8 9 10 11 12 14 15 19 20 21];
AFS4(:,excludefeatures) = [];
```

VIF round 1

```
X = AFS4(:,1:11);
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'SE';'TE';'C1';'C2';'HM';'HC';'FNN';'b';'#S';'BSR';'dC'};

% plot VIF scores
bar(VIF)
title('VIF of features round 1')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 1

```
X(:,featurenumber)=[];
% Cepstrum 2
```

VIF round 2

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'SE';'TE';'C1';'HM';'HC';'FNN';'b';'#S';'BSR';'dC'};

% plot VIF scores
bar(VIF)
title('VIF of features round 2')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 2

```
X(:,featurenumber)=[];
% Shannon entropy
```

VIF round 3

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'TE';'C1';'HM';'HC';'FNN';'b';'#S';'BSR';'dC'};

% plot VIF scores
bar(VIF)
title('VIF of features round 3')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 3

```
X(:,featurenumber)=[];
% Cepstrum 1
```

VIF round 4

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);

features = {'TE';'HM';'HC';'FNN';'b';'#S';'BSR';'dC'};

% plot VIF scores
bar(VIF)
title('VIF of features round 4')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)
```

Remove highest VIF round 4

```
X(:,featurenumber)=[];
% Hjorth mobility
```

VIF round 5

```
R0 = corrcoef(X);           % correlation matrix
VIF=diag(inv(R0))';
[score,featurenumber] = max(VIF);
```

```
features = {'TE';'HC';'FNN';'b';'#S';'BSR';'dC'};

% plot VIF scores
bar(VIF)
title('VIF of features round 5')
xlabel('Features')
ylabel('VIF')
set(gca,'XTick',1:length(VIF),'XTickLabel',features)

% all features have a score below 10
```

Published with MATLAB® R2020a

Appendix Z.3 Input preparation for logistic regression

Contents

- Initialize
- 1. Load and prepare data
- 2. Create binary classification of outcomes
- 3. Normalize data in the range 0:1
- Create final feature set (FFS)
- Create additional feature sets (AFSs)
- AFS 1
- AFS 5

Author: L.M. van Poppel 12/2020

```
% code to prepare input data for Logistic Regression
% 1) create final feature set
% 2) create additional feature sets (AFS)

% input = [features x epochs]

% CODE
    % 1. Load features LR (averaged features x epochs)
    % 2. Change CPC to binary
            % 0 = good outcome (CPC 1-2)
            % 1 = poor outcome (CPC 3-5)
    % 3. Normalize feature matrix to scale [0-1]

    % 4. Create final feature set:
    %       Exclude high VIF and clinical features:
    %       1 3 4 5 6 8 10 20 21

    % 5. Create AFS with features with low multicollinearity
    %   AFS 1. Include clinical features (20 21) to final feature set
    %   AFS 5. Exclude high VIF features from AFS 4
            % AFS 4 = qEEG features that show high predicitve power
```

Initialize

```
close all; clear all; clc
```

1. Load and prepare data

load data t=12

```
data12 = load('Feature_Subjects_12.mat');
data12 = data12.features_all_subjects_12;
% load data t=24
data24 = load('Feature_Subjects_24.mat');
data24 = data24.features_all_subjects_24;

alldata = [data24; data24]; %combine data sets
```

2. Create binary classification of outcomes

0 = good outcome (CPC 1-2) 1 = poor outcome (CPC 3-5)

```
alldata(:,22) = alldata(:,22)>2;

% data 12h and 24h
data12(:,22) = data12(:,22)>2;
data24(:,22) = data24(:,22)>2;
```

3. Normalize data in the range 0:1

normalize operates on each column of data separately 'range' scales between 0:1

```
alldata_norm = normalize(alldata,'range');

% data 12h and 24h
data12_norm = normalize(data12,'range');
data24_norm = normalize(data24,'range');
```

Create final feature set (FFS)

exclude high VIF features and clinical features from extracted features

```
FFS = alldata_norm;
exludeClinicalVIFFeatures = [1 3 4 5 6 8 10 20 21];
FFS(:,exludeClinicalVIFFeatures) = [];

% data 12h and 24h
FFS_12 = data12_norm;
FFS_12(:,exludeClinicalVIFFeatures) = [];
FFS_24 = data24_norm;
FFS_24(:,exludeClinicalVIFFeatures) = [];
```

Create additional feature sets (AFSs)

only feature sets with low multicollinearity are included

AFS 1

exclude high VIF features from extracted features equal to FFS + clinical features High VIF: 1 3 4 5 6 8 10

```
AFS1 = alldata_norm;
exludeVIFFeatures = [1 3 4 5 6 8 10];
AFS1(:,exludeVIFFeatures) = [];
```

AFS 5

exclude high VIF features from AFS 4

```
AFS5 = alldata_norm;
exludeLPVIFFeatures = [1 3 4 5 8 9 10 11 12 14 15 19 20 21];
AFS5(:,exludeLPVIFFeatures)=[];
```

Published with MATLAB® R2020a

Appendix Z.4 Input preparation for LSTM

Contents

- Initialization
- 1. Load data
- 2. Create binary classification of outcomes
- Store data
- 3. Normalize data in the range 0:1
- Save stacked data matrices
- Create final feature set (FFS)
- Create additional feature sets (AFSs)
- AFS 1
- AFS 2
- AFS 3
- AFS 4
- AFS 5

Author: L.M. van Poppel 12/2020

```
% code to prepare input data for LSTM
% 1) create final feature set (FFS)
% 2) create additional feature sets (AFS)

% Specify t=12 and t=24
% Per patient:
    % 1. Load data
    % 2. Change CPC to binary
         % 0 = good outcome (CPC 1-2)
         % 1 = poor outcome (CPC 3-5)
    % 3. Normalize normalize(A,'range') to scale [0-1]
    % Result: subject_timestep x features
    % Run for t=12 and t=24 and save matrices
    % Combine data12 and data24 for complete subject matrix and save

% CODE
    % 1. Load features LR (averaged features x epochs)
    % 2. Change CPC to binary
            % 0 = good outcome (CPC 1-2)
            % 1 = poor outcome (CPC 3-5)
    % 3. Normalize feature matrix to scale [0-1]

    % 4. Create final feature set:
    %       Exclude high VIF and clinical features:
    %       1 3 4 5 6 8 10 20 21

    % 5. Create AFS
    %   AFS 1. Include clinical features (20 21) to final feature set
    %   AFS 2. Exclude clinical features: 20 21 from extraction matrix
    %   AFS 3. Save matrix from feature extraction (includes all qEEG and
    %                                               clinical features)
    %   AFS 4. Exclude qEEG features that show low predicitve power
    %   AFS 5. Exclude high VIF features from AFS 4
```

Initialization

```
clear all; close all; clc;
```

1. Load data

```
% choose EEG time
EEGtime = "_24";

% load excel with subject IDs
excelpac = "Clinical";
excelfile = ".xlsx";
excel_id = strcat(excelpac,EEGtime,excelfile);
subject_number = readtable(excel_id);

% create empty matrix for all features
all_subjects = height(subject_number);
feature_count = 22;
fragments = 30;
all_timesteps = all_subjects*fragments;
features_allsubjects = zeros(all_timesteps,feature_count);

for i = 1:all_subjects
```

```
% define strings
subjectID = subject_number.StudyID(i,1);
subject = strcat(subjectID,EEGtime); % AMC000_time
features = "_features";
MLfile = ".mat";
file_subject = strcat(subject,features,MLfile);   % AMC000_time_features.mat

% load data
load (file_subject);
alldata = features_epoch.';
```

2. Create binary classification of outcomes

0 = good outcome (CPC 1-2) 1 = poor outcome (CPC 3-5)

```
alldata(:,22) = alldata(:,22)>2;
```

Store data

```
epochs = (1:30:(all_subjects*30)+30);
features_allsubjects(epochs(i):(epochs(i+1)-1),:)=alldata;
```

```
end
```

3. Normalize data in the range 0:1

normalize operates on each column of data separately 'range' scales between 0:1

```
alldata_norm = features_allsubjects;
alldata_norm(:,1:21) = normalize(alldata_norm(:,1:21),'range');
```

Save stacked data matrices

save matrix for t=12 and t=24 with appropriate names allfeatures_data12 = alldata_norm; allfeatures_data24 = alldata_norm; LSTM_allfeatures = [allfeatures_data12 ; allfeatures_data24]; Save LSTM_allfeatures and use for preparation of inputs

Create final feature set (FFS)

exclude high VIF features and clinical features from extracted features

```
LSTM_FFS = LSTM_allfeatures;
excludeClinicalVIFFeatures = [1 3 4 5 6 8 10 20 21];
LSTM_FFS(:,excludeClinicalVIFFeatures) = [];

% seperate input for 12 and 24 hours after cardiac arrest
epoch12 = 78*30;

LSTM_FFS_12h = LSTM_FFS(1:epoch12,:);
LSTM_FFS_24h = LSTM_FFS;
LSTM_FFS_24h(1:epoch12,:) = [];
```

Create additional feature sets (AFSs)

AFS 1

exclude high VIF features from extracted features equal to FFS + clinical features High VIF: 1 3 4 5 6 8 10

```
LSTM_AFS1 = LSTM_allfeatures;
excludeVIFFeatures = [1 3 4 5 6 8 10];
LSTM_AFS1(:,excludeVIFFeatures) = [];
```

AFS 2

exclude clinical features from extracted features includes all 19 qEEG features

```
LSTM_AFS2 = LSTM_allfeatures;
excludeClinicalFeatures = [20 21];
LSTM_AFS2(:,excludeClinicalFeatures) = [];
```

AFS 3

exclude no features from extracted features includes all 19 qEEG features + clinical features

```
LSTM_AFS3 = LSTM_allfeatures;
```

AFS 4

exclude qEEG features that show low predicitve power

```
LSTM_ASF4 = LSTM_allfeatures;
excludeLPFeatures = [8 9 10 11 12 14 15 19 20 21];
```

```
LSTM_ASF4(:,exludeLPFeatures)=[];
```

AFS 5

exclude high VIF features from AFS 4

```
LSTM_AFS5 = LSTM_allfeatures;
exludeLPVIFFeatures = [1 3 4 5 8 9 10 11 12 14 15 19 20 21];
LSTM_AFS5(:,exludeLPVIFFeatures)=[];
```

Published with MATLAB® R2020a

Appendix Z.5 Feature boxplots

Contents

- Initialize
- Load and prepare data
- create binary classification of outcomes
- normalize data in the range 0:1
- Choose feature
- Scatter plot
- Box plot

Author: L.M. van Poppel 12/2020

```
% Code to plot of features vs. outcome
```

Initialize

```
close all
clear all
clc
```

Load and prepare data

load data t=12

```
data12 = load('Feature_Subjects_12.mat');
data12 = data12.features_all_subjects_12;
% load data t=24
data24 = load('Feature_Subjects_24.mat');
data24 = data24.features_all_subjects_24;

alldata = [data12; data24]; %combine data sets
```

create binary classification of outcomes

0 = good outcome (CPC 1-2) 1 = poor outcome (CPC 3-5)

```
alldata(:,22) = alldata(:,22)>2;
```

normalize data in the range 0:1

normalize operates on each column of data separately 'range' scales between 0:1

```
alldata_norm = normalize(alldata,'range');
```

Choose feature

```
%1      Shannon entropy
%2      Tsallis entropy
%3      Cepstrum 1
%4      Cepstrum 2
%5      Hjorth mobility
%6      Hjorth complexity
%7      FNN
```

```
%8      AR 1
%9      AR 2
%10     Delta
%11     Theta
%12     Alpha
%13     Beta
%14     Signal power
%15     Regularity
%16     Number of epileptic spikes
%17     BSR
%18     Delta coherence
%19     PLI
%20     Age
%21     Sex

% manually set feature name and number
% Feature name
ft = 'Burst Suppression Ratio';
% Feature number
feature = alldata_norm(:,17);
```

Scatter plot

```
outcome = alldata_norm(:,22);
%outcome = corrected_norm_data(:,14);

myColors = zeros(size(feature,1),3);
% Set colors
rowsToSetRed = outcome == 1;     %red is poor outcome
rowsToSetBlue = outcome == 0;    %green is good outcome

% Set colormap to red for the red rows.[RGB]
myColors(rowsToSetRed, 1) = 1; %red
myColors(rowsToSetRed, 2) = 0;
myColors(rowsToSetRed, 3) = 0;
% Set colormap to blue for the blue rows.[RGB]
myColors(rowsToSetBlue, 1) = 0;
myColors(rowsToSetBlue, 2) = 1;
myColors(rowsToSetBlue, 3) = 0; %blue

%scatterplot
figure (1)
scatter(outcome, feature, 80, myColors);
grid on;
title(ft)
xlabel('Outcome: good (green) vs. poor (red)')
ylabel(ft)
```

Box plot

```
poor = find(outcome==1);
good = find(outcome==0);
ftpoor = feature(poor);
ftgood = feature(good);
fts = [ftgood;ftpoor];

g1 = repmat({'Good'},size(ftgood));
g2 = repmat({'Poor'},size(ftpoor));
g = [g1;g2];

%boxplot
figure (2)
boxplot (fts,g)
title(ft)
```

Published with MATLAB® R2020a

Z.6 Final Logistic Regression

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on 31-12 2020

@author: lauravanpoppel
"""

import pandas as pd
import numpy as np
import matplotlib
from matplotlib import pyplot
from scipy.io import loadmat
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold
import statsmodels.stats.api as sms


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.regularizers import l1, l2, l1_l2
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.metrics import Accuracy, AUC, Precision, SensitivityAtSpecificity, Speci
ficityAtSensitivity
from keras_adabound import AdaBound

# Prepare data #
# choose input
dataMatlab = loadmat('input4.mat');
dataML = dataMatlab.get('input4');

# define features and targets
# change according to input used
x = dataML[:,:12];
y = dataML[:,12]; # 1 is poor outcome

# repeat process 50 times
repeats = 50
test_loss_per_repeat = []
test_acc_per_repeat = []
test_auc_per_repeat = []
test_pre_per_repeat = []
test_se_per_repeat = []
test_sp_per_repeat = []

test_loss_repeat = []
test_acc_repeat = []
test_auc_repeat = []
test_pre_repeat = []
test_se_repeat = []
test_sp_repeat = []

for i in range(repeats):

    # use 10-fold cross validation
    num_folds = 10;
    fold_no = 1

    # Define the K-fold Cross Validator
    skf = StratifiedKFold(n_splits=num_folds, shuffle=True)

    # test metrics for CV
    test_loss_per_fold = [];
    test_acc_per_fold = [];
```

```
    test_auc_per_fold = [];
    test_pre_per_fold = [];
    test_se_per_fold = [];
    test_sp_per_fold = [];

    # constant paramters
    inputdim = int((x.shape[1:])[0])
    outputdim = 1
    act = 'sigmoid'
    opt = SGD(learning_rate=0.01,
              momentum=0.9,
              nesterov=True)
    loss_fun = 'binary_crossentropy'
    nb_epoch = 1500
    metric1 = 'accuracy'
    metric2 = AUC(name='auc')
    metric3 = Precision(name='precision')
    metric4 = SensitivityAtSpecificity(1, name='sensitivity_at_specificity')
    metric5 = SpecificityAtSensitivity(0.95, name='specificity_at_sensitivity')

    for train, test in skf.split(x, y):

        # build model
        model = Sequential()
        model.add(Dense(outputdim,
                        activation=act,
                        input_dim=inputdim))

        # compile model
        model.compile(optimizer=opt,
                      loss=loss_fun,
                      metrics=[metric1, metric2, metric3, metric4, metric5]
                      )

         # Generate a print
        print('------------------------------------------------------------------------')
        print(f'Training for fold {fold_no} ...')


        # fit model
        history = model.fit(x[train],
                            y[train],
                            epochs=nb_epoch,
                            verbose=0)

        # evaluate model
        scores = model.evaluate(x[test], y[test], verbose=0)
        # print(f'Test score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]};\
        #                                        {model.metrics_names[1]} of {scores[1]*100}%
    ;\
        #                                        {model.metrics_names[2]} of {scores[2]};\
        #                                        {model.metrics_names[3]} of {scores[3]};\
        #                                        {model.metrics_names[4]} of {scores[4]};')
        test_loss_per_fold.append(scores[0])
        test_acc_per_fold.append(scores[1]*100)
        test_auc_per_fold.append(scores[2])
        test_pre_per_fold.append(scores[3])
        test_se_per_fold.append(scores[4])
        test_sp_per_fold.append(scores[5])

        # Increase fold number
        fold_no = fold_no + 1

       # == Provide average scores ==
    print('------------------------------------------------------------------------')
    print('Score per fold')
```

```
    for i in range(0, len(test_acc_per_fold)):
      print('------------------------------------------------------------------------')
      print(f'> Fold test {i+1} - Loss: {test_loss_per_fold[i]} - Accuracy: {test_acc_per_fold[i]}% - AUC: {test_auc_per_fold[i]} - Precision: {test_pre_per_fold[i]} - Poor Se100: {test_se_per_fold[i]} - Good Se95: {test_sp_per_fold[i]}')
    print('------------------------------------------------------------------------')
    print('Average test scores for all folds:')
    print(f'> Accuracy: {np.mean(test_acc_per_fold)} (+- {np.std(test_acc_per_fold)})')
    print(f'> Loss: {np.mean(test_loss_per_fold)} (+- {np.std(test_loss_per_fold)})')
    print(f'> AUC: {np.mean(test_auc_per_fold)} (+- {np.std(test_auc_per_fold)})')
    print(f'> Precision: {np.mean(test_pre_per_fold)} (+- {np.std(test_pre_per_fold)})')
    print(f'> Poor Se100: {np.mean(test_se_per_fold)} (+- {np.std(test_se_per_fold)})')
    print(f'> Good Se95: {np.mean(test_sp_per_fold)} (+- {np.std(test_sp_per_fold)})')
    print('------------------------------------------------------------------------')

    # append metrics for all repeats
    test_loss_per_repeat.append(test_loss_per_fold)
    test_acc_per_repeat.append(test_acc_per_fold)
    test_auc_per_repeat.append(test_auc_per_fold)
    test_pre_per_repeat.append(test_pre_per_fold)
    test_se_per_repeat.append(test_se_per_fold)
    test_sp_per_repeat.append(test_sp_per_fold)

    # calculate 95% CI
    test_loss_repeat.extend(test_loss_per_fold)
    ci_loss = sms.DescrStatsW(test_loss_repeat).tconfint_mean()
    test_acc_repeat.extend(test_acc_per_fold)
    ci_acc = sms.DescrStatsW(test_acc_repeat).tconfint_mean()
    test_auc_repeat.extend(test_auc_per_fold)
    ci_auc = sms.DescrStatsW(test_auc_repeat).tconfint_mean()
    test_pre_repeat.extend(test_pre_per_fold)
    ci_pre = sms.DescrStatsW(test_pre_repeat).tconfint_mean()
    test_se_repeat.extend(test_se_per_fold)
    ci_se = sms.DescrStatsW(test_se_repeat).tconfint_mean()
    test_sp_repeat.extend(test_sp_per_fold)
    ci_sp = sms.DescrStatsW(test_sp_repeat).tconfint_mean()

print('------------------------------------------------------------------------')
print('Average test scores for all repeats:')
print(f'> Accuracy: {np.mean(test_acc_per_repeat)} std:(+- {np.std(test_acc_per_repeat)}) 95%CI:{ci_acc}')
print(f'> Loss: {np.mean(test_loss_per_repeat)} std:(+- {np.std(test_loss_per_repeat)}) 95%CI:{ci_loss}')
print(f'> AUC: {np.mean(test_auc_per_repeat)} std:(+- {np.std(test_auc_per_repeat)}) 95%CI:{ci_auc}')
print(f'> Precision: {np.mean(test_pre_per_repeat)} std:(+- {np.std(test_pre_per_repeat)}) 95%CI:{ci_pre}')
print(f'> Poor Se100: {np.mean(test_se_per_repeat)} std:(+- {np.std(test_se_per_repeat)}) 95%CI:{ci_se}')
print(f'> Good Se95: {np.mean(test_sp_per_repeat)} std:(+- {np.std(test_sp_per_repeat)}) 95%CI:{ci_sp}')
print('------------------------------------------------------------------------')
```

Z.7 Hyperparameter search LSTM

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on 31-12 2020

@author: lauravanpoppel
"""

import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Bidirectional, BatchNormalization, A
ctivation
from tensorflow.keras.constraints import max_norm
from tensorflow.keras.regularizers import l1, l2, l1_l2
from tensorflow.keras.optimizers import Adam, Nadam, SGD, Adamax
from keras_adabound import AdaBound
from tensorflow.keras.metrics import Accuracy, AUC, Precision, SensitivityAtSpecificity, Speci
ficityAtSensitivity
from tensorflow.keras.metrics import TruePositives, TrueNegatives, FalsePositives, FalseNegati
ves


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.stats.api as sms

from sklearn.model_selection import train_test_split, KFold, StratifiedKFold
from sklearn.metrics import roc_curve, auc
from scipy.io import loadmat
import talos
from talos.utils import lr_normalizer

# import 2D structure
dataMatlab = loadmat('LSTM_input4.mat');
data = dataMatlab.get('LSTM_input4');

# define features and targets
features = data[:,:12];
outcomes = data[:,12]; # 1 is poor outcome

# reshape to input matrices and outcome vector
timesteps = 30;
predictors = 12;
subjects = int(features.shape[0]/timesteps);

x = features.reshape(subjects,timesteps,predictors);
y = outcomes.reshape(subjects,timesteps);
y = y[:,0];


# divide data into train and validation sets
x_tr,x_val,y_tr,y_val=train_test_split(x,y,test_size=0.2,stratify=y);


def LSTM_model(x_tr, y_tr, x_val, y_val, params):

    ######################
    # Define parameters #
    ######################
    # first LSTM layer
    input_LSTM = (x.shape[1:])
```

```
61.
62.     # fully connected layer
63.     activation_function_FC = 'sigmoid'
64.     hidden_units_FC = 1
65.
66.     # compile
67.     opt = Adam(learning_rate=params['lr'])
68.     loss_function = 'binary_crossentropy'
69.     metric1 = 'accuracy'
70.     metric2 = AUC(name='auc')
71.     metric3 = SensitivityAtSpecificity(1, name='se100_poor_outcome')
72.     metric4 = SpecificityAtSensitivity(0.95, name='se95_good_outcome')
73.
74.     # fit
75.     batchsize = 32
76.
77.     ################
78.     # Build model #
79.     ################
80.     model = Sequential()
81.     model.add(LSTM(params['hidden_units_LSTM'],
82.                    input_shape=input_LSTM,
83.                    dropout=params['dropout'],
84.                    recurrent_dropout=params['recurrent_dropout'],
85.                    return_sequences=True
86.                    ))
87.
88.     #For 2-layer model, uncomment followin4 lines and set return_sequences=True
89.     model.add(LSTM(params['hidden_units_LSTM'],
90.                dropout=params['dropout'],
91.                recurrent_dropout=params['recurrent_dropout']
92.                ))
93.
94.     model.add(Dense(hidden_units_FC,
95.                     activation=activation_function_FC))
96.
97.     ##################
98.     # Compile model #
99.     ##################
100.    model.compile(loss=loss_function,
101.                  optimizer=opt,
102.                  metrics=[metric1, metric2, metric3, metric4])
103.
104.
105.    ##############
106.    # Fit model #
107.    ##############
108.    history = model.fit(x_tr,
109.                        y_tr,
110.                        validation_data=(x_val,y_val),
111.                        batch_size=batchsize,
112.                        epochs=params['epochs'],
113.                        verbose=0)
114.
115.   # history object and model are returned
116.    return history, model
117.
118.
119.###############
120.# Parameters #
121.###############
122.
123.p = {'hidden_units_LSTM':[2,4,6,8,10,12,14,16,18,20,22,24],
124.     'dropout': [0, 0.2, 0.5],
125.     'recurrent_dropout':[0, 0.2, 0.5],
126.     'lr': [0.0001, 0.0005, 0.001, 0.005, 0.01],
```

```
    'epochs': [20,40,60,80,100,120,140,160,180,200],
    }

########
# Scan #
########
scan_object = talos.Scan(x=x_tr,
                         y=y_tr,
                         x_val=x_val,
                         y_val=y_val,
                         model=LSTM_model,
                         params=p,
                         experiment_name='Final search 2 layers',
                         reduction_method='correlation',
                         reduction_metric='val_auc',
                         reduction_threshold = 0.8,
                         fraction_limit=0.25
                         )
```

Z.8 Hyperparameter search analysis LSTM

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on 31-12 2020

@author: lauravanpoppel
"""


import pandas as pd
import numpy as np
from numpy import mean, median
import matplotlib.pyplot as plt
import statsmodels.stats.api as sms
import talos


# Analyze result of 10 random searches
analyze1=talos.Analyze('112620222042.csv')
df1 = analyze1.data
analyze2=talos.Analyze('112620222045.csv')
df2 = analyze2.data
analyze3=talos.Analyze('112620222049.csv')
df3 = analyze3.data
analyze4=talos.Analyze('112620222057.csv')
df4 = analyze4.data
analyze5=talos.Analyze('112620222100.csv')
df5 = analyze5.data
analyze6=talos.Analyze('112620222104.csv')
df6 = analyze6.data
analyze7=talos.Analyze('112620222107.csv')
df7 = analyze7.data
analyze8=talos.Analyze('112620222111.csv')
df8 = analyze8.data
analyze9=talos.Analyze('112620222114.csv')
df9 = analyze9.data
analyze10=talos.Analyze('112620222118.csv')
df10 = analyze10.data

merge = [df1, df2, df3, df4, df5, df6, df7, df8, df9, df10]
dfm = pd.concat(merge)

corr1=analyze1.correlate('val_auc', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good
_outcome','val_accuracy', 'val_loss','val_se100_poor_outcome','val_se95_good_outcome'])
corr2=analyze2.correlate('val_auc', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good
_outcome','val_accuracy', 'val_loss','val_se100_poor_outcome','val_se95_good_outcome'])
corr3=analyze3.correlate('val_auc', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good
_outcome','val_accuracy', 'val_loss','val_se100_poor_outcome','val_se95_good_outcome'])
corr4=analyze4.correlate('val_auc', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good
_outcome','val_accuracy', 'val_loss','val_se100_poor_outcome','val_se95_good_outcome'])
corr5=analyze5.correlate('val_auc', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good
_outcome','val_accuracy', 'val_loss','val_se100_poor_outcome','val_se95_good_outcome'])
corr6=analyze6.correlate('val_auc', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good
_outcome','val_accuracy', 'val_loss','val_se100_poor_outcome','val_se95_good_outcome'])
corr7=analyze7.correlate('val_auc', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good
_outcome','val_accuracy', 'val_loss','val_se100_poor_outcome','val_se95_good_outcome'])
corr8=analyze8.correlate('val_auc', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good
_outcome','val_accuracy', 'val_loss','val_se100_poor_outcome','val_se95_good_outcome'])
corr9=analyze9.correlate('val_auc', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good
_outcome','val_accuracy', 'val_loss','val_se100_poor_outcome','val_se95_good_outcome'])
corr10=analyze10.correlate('val_auc', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_go
od_outcome','val_accuracy', 'val_loss','val_se100_poor_outcome','val_se95_good_outcome'])
```

```
corrall_auc = pd.concat([corr1,corr2, corr3, corr4, corr5, corr6, corr7, corr8, corr9, corr10], axis=1)
corrall_auc = round(corrall_auc,3)
corr_mean_auc = corrall_auc.mean(1)
corr_mean_auc = round(corr_mean_auc,3)

corr1=analyze1.correlate('val_se100_poor_outcome', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good_outcome','val_accuracy', 'val_loss','val_auc','val_se95_good_outcome'])
corr2=analyze2.correlate('val_se100_poor_outcome', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good_outcome','val_accuracy', 'val_loss','val_auc','val_se95_good_outcome'])
corr3=analyze3.correlate('val_se100_poor_outcome', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good_outcome','val_accuracy', 'val_loss','val_auc','val_se95_good_outcome'])
corr4=analyze4.correlate('val_se100_poor_outcome', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good_outcome','val_accuracy', 'val_loss','val_auc','val_se95_good_outcome'])
corr5=analyze5.correlate('val_se100_poor_outcome', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good_outcome','val_accuracy', 'val_loss','val_auc','val_se95_good_outcome'])
corr6=analyze6.correlate('val_se100_poor_outcome', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good_outcome','val_accuracy', 'val_loss','val_auc','val_se95_good_outcome'])
corr7=analyze7.correlate('val_se100_poor_outcome', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good_outcome','val_accuracy', 'val_loss','val_auc','val_se95_good_outcome'])
corr8=analyze8.correlate('val_se100_poor_outcome', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good_outcome','val_accuracy', 'val_loss','val_auc','val_se95_good_outcome'])
corr9=analyze9.correlate('val_se100_poor_outcome', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good_outcome','val_accuracy', 'val_loss','val_auc','val_se95_good_outcome'])
corr10=analyze10.correlate('val_se100_poor_outcome', ['accuracy', 'loss', 'auc','se100_poor_outcome','se95_good_outcome','val_accuracy', 'val_loss','val_auc','val_se95_good_outcome'])
corrall_se100 = pd.concat([corr1,corr2, corr3, corr4, corr5, corr6, corr7, corr8, corr9, corr10], axis=1)
corrall_se100 = round(corrall_se100,3)
corr_mean_se100 = corrall_se100.mean(1)
corr_mean_se100 = round(corr_mean_se100,3)


##########
# Epochs # [20,40,60,80,100,120,140,160,180,200]
##########
e = dfm[['val_auc','val_se100_poor_outcome','epochs']]
e_names = ['20','40','60','80','100','120','140','160','180','200']


e20 = e[e['epochs']==20]
e20_auc = e20['val_auc']
e20_auc = e20_auc.tolist()
e20_se100 = e20['val_se100_poor_outcome']
e20_se100 = e20_se100.tolist()

e40 = e[e['epochs']==40]
e40_auc = e40['val_auc']
e40_auc = e40_auc.tolist()
e40_se100 = e40['val_se100_poor_outcome']
e40_se100 = e40_se100.tolist()

e60 = e[e['epochs']==60]
e60_auc = e60['val_auc']
e60_auc = e60_auc.tolist()
e60_se100 = e60['val_se100_poor_outcome']
e60_se100 = e60_se100.tolist()

e80 = e[e['epochs']==80]
e80_auc = e80['val_auc']
e80_auc = e80_auc.tolist()
e80_se100 = e80['val_se100_poor_outcome']
e80_se100 = e80_se100.tolist()

e100 = e[e['epochs']==100]
e100_auc = e100['val_auc']
```

```
e100_auc = e100_auc.tolist()
e100_se100 = e100['val_se100_poor_outcome']
e100_se100 = e100_se100.tolist()

e120 = e[e['epochs']==120]
e120_auc = e120['val_auc']
e120_auc = e120_auc.tolist()
e120_se100 = e120['val_se100_poor_outcome']
e120_se100 = e120_se100.tolist()

e140 = e[e['epochs']==140]
e140_auc = e140['val_auc']
e140_auc = e140_auc.tolist()
e140_se100 = e140['val_se100_poor_outcome']
e140_se100 = e140_se100.tolist()

e160 = e[e['epochs']==160]
e160_auc = e160['val_auc']
e160_auc = e160_auc.tolist()
e160_se100 = e160['val_se100_poor_outcome']
e160_se100 = e160_se100.tolist()

e180 = e[e['epochs']==180]
e180_auc = e180['val_auc']
e180_auc = e180_auc.tolist()
e180_se100 = e180['val_se100_poor_outcome']
e180_se100 = e180_se100.tolist()

e200 = e[e['epochs']==200]
e200_auc = e200['val_auc']
e200_auc = e200_auc.tolist()
e200_se100 = e200['val_se100_poor_outcome']
e200_se100 = e200_se100.tolist()

## AUC ##

# Boxplot
e_auc = [e20_auc, e40_auc, e60_auc, e80_auc, e100_auc,
         e120_auc, e140_auc, e160_auc, e180_auc, e200_auc]
fig_e_auc, ax_e_auc = plt.subplots()
ax_e_auc.set_title('Boxplots of the validation AUC of the number of epochs')
ax_e_auc.boxplot(e_auc)
plt.xlabel('Number of epochs')
plt.ylabel('Validation AUC')
plt.xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], e_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

# Correlation
corr_e_auc = dfm[['val_auc','epochs']]
corr_e_auc =  corr_e_auc.corr(method='pearson')

# Mean
mean_e20_auc = mean(e20_auc)
mean_e40_auc = mean(e40_auc)
mean_e60_auc = mean(e60_auc)
mean_e80_auc = mean(e80_auc)
mean_e100_auc = mean(e100_auc)
mean_e120_auc = mean(e120_auc)
mean_e140_auc = mean(e140_auc)
mean_e160_auc = mean(e160_auc)
mean_e180_auc = mean(e180_auc)
mean_e200_auc = mean(e200_auc)
#mean_e_auc = mean(e_auc,1)
mean_e_auc = [mean_e20_auc, mean_e40_auc, mean_e60_auc, mean_e80_auc, mean_e100_auc,
              mean_e120_auc, mean_e140_auc, mean_e160_auc, mean_e180_auc, mean_e200_auc]
fig_emean_auc, ax_emean_auc = plt.subplots()
```

```
ax_emean_auc.set_title('Mean validation AUC per the number of epochs')
ax_emean_auc.plot(e_names,mean_e_auc)
plt.grid(color='black', linestyle='--', linewidth=0.2)

# Median
med_e20_auc = median(e20_auc)
med_e40_auc = median(e40_auc)
med_e60_auc = median(e60_auc)
med_e80_auc = median(e80_auc)
med_e100_auc = median(e100_auc)
med_e120_auc = median(e120_auc)
med_e140_auc = median(e140_auc)
med_e160_auc = median(e160_auc)
med_e180_auc = median(e180_auc)
med_e200_auc = median(e200_auc)
#median_e_auc = np.median(e_auc,1)
median_e_auc = [med_e20_auc, med_e40_auc, med_e60_auc, med_e80_auc, med_e100_auc,
                med_e120_auc, med_e140_auc, med_e160_auc, med_e180_auc, med_e200_auc]
fig_emed_auc, ax_emed_auc = plt.subplots()
ax_emed_auc.set_title('Median validation AUC per the number of epochs')
ax_emed_auc.plot(e_names,median_e_auc)
plt.grid(color='black', linestyle='--', linewidth=0.2)

## SE100 ##

# Boxplot
e_se100 = [e20_se100, e40_se100, e60_se100, e80_se100, e100_se100,
           e120_se100, e140_se100, e160_se100, e180_se100, e200_se100]
fig_e_se100, ax_e_se100 = plt.subplots()
ax_e_se100.set_title('Boxplots of the validation SeSp100 of the number of epochs')
ax_e_se100.boxplot(e_se100)
plt.xlabel('Number of epochs')
plt.ylabel('Validation SeSp100')
plt.xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], e_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

corr_e_se100 = dfm[['val_se100_poor_outcome','epochs']]
corr_e_se100 =  corr_e_se100.corr(method='pearson')

# Mean
mean_e20_se100 = mean(e20_se100)
mean_e40_se100 = mean(e40_se100)
mean_e60_se100 = mean(e60_se100)
mean_e80_se100 = mean(e80_se100)
mean_e100_se100 = mean(e100_se100)
mean_e120_se100 = mean(e120_se100)
mean_e140_se100 = mean(e140_se100)
mean_e160_se100 = mean(e160_se100)
mean_e180_se100 = mean(e180_se100)
mean_e200_se100 = mean(e200_se100)
#mean_e_se100 = mean(e_se100,1)
mean_e_se100 = [mean_e20_se100, mean_e40_se100, mean_e60_se100, mean_e80_se100, mean_e100_se100,
                mean_e120_se100, mean_e140_se100, mean_e160_se100, mean_e180_se100, mean_e200_se100]
fig_emean_se100, ax_emean_se100 = plt.subplots()
ax_emean_se100.set_title('Mean validation Se100 of poor outcome prediction per number of epochs')
ax_emean_se100.plot(e_names,mean_e_se100)
plt.grid(color='black', linestyle='--', linewidth=0.2)

# Median
med_e20_se100 = median(e20_se100)
med_e40_se100 = median(e40_se100)
med_e60_se100 = median(e60_se100)
med_e80_se100 = median(e80_se100)
```

```
med_e100_se100 = median(e100_se100)
med_e120_se100 = median(e120_se100)
med_e140_se100 = median(e140_se100)
med_e160_se100 = median(e160_se100)
med_e180_se100 = median(e180_se100)
med_e200_se100 = median(e200_se100)
#median_e_se100 = np.median(e_se100,1)
median_e_se100 = [med_e20_se100, med_e40_se100, med_e60_se100, med_e80_se100, med_e100_se100,

                med_e120_se100, med_e140_se100, med_e160_se100, med_e180_se100, med_e200_se100
    ]
fig_emed_se100, ax_emed_se100 = plt.subplots()
ax_emed_se100.set_title('Median validation Se100 of poor outcome prediction per number of epoc
    hs')
ax_emed_se100.plot(e_names,median_e_se100)
plt.grid(color='black', linestyle='--', linewidth=0.2)


# Mean both metrics in 1
fig_e, ax_e = plt.subplots()
ax_e.set_title('validation AUC and Se100 of poor outcome prediction per number of epochs')
ax_e.plot(e_names,mean_e_auc,'g-',label='mean AUC', linewidth=1)
ax_e.plot(e_names,mean_e_se100,'g--',label='mean Se100', linewidth=1)
ax_e.plot(e_names,median_e_auc,'y-',label='median AUC', linewidth=1)
ax_e.plot(e_names,median_e_se100,'y--',label='median Se100', linewidth=1)
ax_e.legend(loc='center right')
plt.grid(color='black', linestyle='--', linewidth=0.2)


#########
# Units # [2,4,6,8,10,12,14,16,18,20,22,24]
#########
hu = dfm[['val_auc','val_se100_poor_outcome','hidden_units_LSTM']]
hu_names = ['2','4','6','8','10','12','14','16','18','20','22','24']

hu1 = hu[hu['hidden_units_LSTM']==2]
hu1_auc = hu1['val_auc']
hu1_auc = hu1_auc.tolist()
hu1_se100 = hu1['val_se100_poor_outcome']
hu1_se100 = hu1_se100.tolist()

hu2 = hu[hu['hidden_units_LSTM']==4]
hu2_auc = hu2['val_auc']
hu2_auc = hu2_auc.tolist()
hu2_se100 = hu2['val_se100_poor_outcome']
hu2_se100 = hu2_se100.tolist()

hu3 = hu[hu['hidden_units_LSTM']==6]
hu3_auc = hu3['val_auc']
hu3_auc = hu3_auc.tolist()
hu3_se100 = hu3['val_se100_poor_outcome']
hu3_se100 = hu3_se100.tolist()

hu4 = hu[hu['hidden_units_LSTM']==8]
hu4_auc = hu4['val_auc']
hu4_auc = hu4_auc.tolist()
hu4_se100 = hu4['val_se100_poor_outcome']
hu4_se100 = hu4_se100.tolist()

hu5 = hu[hu['hidden_units_LSTM']==10]
hu5_auc = hu5['val_auc']
hu5_auc = hu5_auc.tolist()
hu5_se100 = hu5['val_se100_poor_outcome']
hu5_se100 = hu5_se100.tolist()

```

```
hu6 = hu[hu['hidden_units_LSTM']==12]
hu6_auc = hu6['val_auc']
hu6_auc = hu6_auc.tolist()
hu6_se100 = hu6['val_se100_poor_outcome']
hu6_se100 = hu6_se100.tolist()

hu7 = hu[hu['hidden_units_LSTM']==14]
hu7_auc = hu7['val_auc']
hu7_auc = hu7_auc.tolist()
hu7_se100 = hu7['val_se100_poor_outcome']
hu7_se100 = hu7_se100.tolist()

hu8 = hu[hu['hidden_units_LSTM']==16]
hu8_auc = hu8['val_auc']
hu8_auc = hu8_auc.tolist()
hu8_se100 = hu8['val_se100_poor_outcome']
hu8_se100 = hu8_se100.tolist()

hu9 = hu[hu['hidden_units_LSTM']==18]
hu9_auc = hu9['val_auc']
hu9_auc = hu9_auc.tolist()
hu9_se100 = hu9['val_se100_poor_outcome']
hu9_se100 = hu9_se100.tolist()

hu10 = hu[hu['hidden_units_LSTM']==20]
hu10_auc = hu10['val_auc']
hu10_auc = hu10_auc.tolist()
hu10_se100 = hu10['val_se100_poor_outcome']
hu10_se100 = hu10_se100.tolist()

hu11 = hu[hu['hidden_units_LSTM']==22]
hu11_auc = hu11['val_auc']
hu11_auc = hu11_auc.tolist()
hu11_se100 = hu11['val_se100_poor_outcome']
hu11_se100 = hu11_se100.tolist()

hu12 = hu[hu['hidden_units_LSTM']==24]
hu12_auc = hu12['val_auc']
hu12_auc = hu12_auc.tolist()
hu12_se100 = hu12['val_se100_poor_outcome']
hu12_se100 = hu12_se100.tolist()

## AUC ##

# Boxplot
hu_auc = [hu1_auc, hu2_auc, hu3_auc, hu4_auc,
          hu5_auc, hu6_auc, hu7_auc, hu8_auc,
          hu9_auc, hu10_auc,hu11_auc, hu12_auc]
fig_hu_auc, ax_hu_auc = plt.subplots()
ax_hu_auc.set_title('Boxplots of the validation AUC of the number of units')
ax_hu_auc.boxplot(hu_auc)
plt.xlabel('Number of units')
plt.ylabel('Validation AUC')
plt.xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], hu_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

corr_hu_auc = dfm[['val_auc','hidden_units_LSTM']]
corr_hu_auc =  corr_hu_auc.corr(method='pearson')

# Mean
mean_hu1_auc = mean(hu1_auc)
mean_hu2_auc = mean(hu2_auc)
mean_hu3_auc = mean(hu3_auc)
mean_hu4_auc = mean(hu4_auc)
mean_hu5_auc = mean(hu5_auc)
mean_hu6_auc = mean(hu6_auc)
```

```
mean_hu7_auc = mean(hu7_auc)
mean_hu8_auc = mean(hu8_auc)
mean_hu9_auc = mean(hu9_auc)
mean_hu10_auc = mean(hu10_auc)
mean_hu11_auc = mean(hu11_auc)
mean_hu12_auc = mean(hu12_auc)
#mean_hu_auc = mean(hu_auc,1)
mean_hu_auc = [mean_hu1_auc, mean_hu2_auc, mean_hu3_auc, mean_hu4_auc,
               mean_hu5_auc, mean_hu6_auc, mean_hu7_auc, mean_hu8_auc,
               mean_hu9_auc, mean_hu10_auc, mean_hu11_auc, mean_hu12_auc]
fig_humean_auc, ax_humean_auc = plt.subplots()
ax_humean_auc.set_title('Mean validation AUC per number of units')
ax_humean_auc.plot(hu_names,mean_hu_auc)
plt.grid(color='black', linestyle='--', linewidth=0.2)

# Median
med_hu1_auc = median(hu1_auc)
med_hu2_auc = median(hu2_auc)
med_hu3_auc = median(hu3_auc)
med_hu4_auc = median(hu4_auc)
med_hu5_auc = median(hu5_auc)
med_hu6_auc = median(hu6_auc)
med_hu7_auc = median(hu7_auc)
med_hu8_auc = median(hu8_auc)
med_hu9_auc = median(hu9_auc)
med_hu10_auc = median(hu10_auc)
med_hu11_auc = median(hu11_auc)
med_hu12_auc = median(hu12_auc)
#median_hu_auc = np.median(hu_auc,1)
median_hu_auc = [med_hu1_auc, med_hu2_auc, med_hu3_auc, med_hu4_auc,
                 med_hu5_auc, med_hu6_auc, med_hu7_auc, med_hu8_auc,
                 med_hu9_auc, med_hu10_auc, med_hu11_auc, med_hu12_auc]
fig_humed_auc, ax_humed_auc = plt.subplots()
ax_humed_auc.set_title('Median validation AUC per number of units')
ax_humed_auc.plot(hu_names,median_hu_auc)
plt.grid(color='black', linestyle='--', linewidth=0.2)

## SE100 ##
hu_se100 = [hu1_se100, hu2_se100, hu3_se100, hu4_se100,
            hu5_se100, hu6_se100, hu7_se100, hu8_se100,
            hu9_se100, hu10_se100, hu11_se100, hu12_se100]
fig_hu_se100, ax_hu_se100 = plt.subplots()
ax_hu_se100.set_title('Boxplots of the validation SeSp100 of the number of units')
ax_hu_se100.boxplot(hu_se100)
plt.xlabel('Number of units')
plt.ylabel('Validation SeSp100')
plt.xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], hu_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

corr_hu_se100 = dfm[['val_se100_poor_outcome','hidden_units_LSTM']]
corr_hu_se100 =  corr_hu_se100.corr(method='pearson')

# Mean
mean_hu1_se100 = mean(hu1_se100)
mean_hu2_se100 = mean(hu2_se100)
mean_hu3_se100 = mean(hu3_se100)
mean_hu4_se100 = mean(hu4_se100)
mean_hu5_se100 = mean(hu5_se100)
mean_hu6_se100 = mean(hu6_se100)
mean_hu7_se100 = mean(hu7_se100)
mean_hu8_se100 = mean(hu8_se100)
mean_hu9_se100 = mean(hu9_se100)
mean_hu10_se100 = mean(hu10_se100)
mean_hu11_se100 = mean(hu11_se100)
mean_hu12_se100 = mean(hu12_se100)
# mean_hu_se100 = mean(hu_se100,1)
```

```
mean_hu_se100 = [mean_hu1_se100, mean_hu2_se100, mean_hu3_se100, mean_hu4_se100,
                 mean_hu5_se100, mean_hu6_se100, mean_hu7_se100, mean_hu8_se100,
                 mean_hu9_se100, mean_hu10_se100, mean_hu11_se100, mean_hu12_se100]
fig_humean_se100, ax_humean_se100 = plt.subplots()
ax_humean_se100.set_title('Mean validation Se100 of poor outcome prediction per number of units')
ax_humean_se100.plot(hu_names,mean_hu_se100)
plt.grid(color='black', linestyle='--', linewidth=0.2)

# Median
med_hu1_se100 = median(hu1_se100)
med_hu2_se100 = median(hu2_se100)
med_hu3_se100 = median(hu3_se100)
med_hu4_se100 = median(hu4_se100)
med_hu5_se100 = median(hu5_se100)
med_hu6_se100 = median(hu6_se100)
med_hu7_se100 = median(hu7_se100)
med_hu8_se100 = median(hu8_se100)
med_hu9_se100 = median(hu9_se100)
med_hu10_se100 = median(hu10_se100)
med_hu11_se100 = median(hu11_se100)
med_hu12_se100 = median(hu12_se100)
#median_hu_se100 = np.median(hu_se100,1)
median_hu_se100 = [med_hu1_se100, med_hu2_se100, med_hu3_se100, med_hu4_se100,
                 med_hu5_se100, med_hu6_se100, med_hu7_se100, med_hu8_se100,
                 med_hu9_se100, med_hu10_se100, med_hu11_se100, med_hu12_se100]
fig_humed_se100, ax_humed_se100 = plt.subplots()
ax_humed_se100.set_title('Median validation Se100 of poor outcome prediction per number of units')
ax_humed_se100.plot(hu_names,median_hu_se100)
plt.grid(color='black', linestyle='--', linewidth=0.2)


# Mean both metrics in 1
fig_hu, ax_hu = plt.subplots()
ax_hu.set_title('validation AUC and Se100 of poor outcome prediction per number of units')
ax_hu.plot(hu_names,mean_hu_auc,'g-',label='mean AUC', linewidth=1)
ax_hu.plot(hu_names,mean_hu_se100,'g--',label='mean Se100', linewidth=1)
ax_hu.plot(hu_names,median_hu_auc,'y-',label='median AUC', linewidth=1)
ax_hu.plot(hu_names,median_hu_se100,'y--',label='median Se100', linewidth=1)
ax_hu.legend(loc='center right')
plt.grid(color='black', linestyle='--', linewidth=0.2)


######
# LR # [0.0001, 0.0005, 0.001, 0.005, 0.01],
######

lr = dfm[['val_auc','val_se100_poor_outcome','lr']]
lr_names = ['0.0001', '0.0005', '0.001','0.005','0.01']

lr1 = lr [lr['lr']==0.0001]
lr1_auc = lr1['val_auc']
lr1_auc = lr1_auc.tolist()
lr1_se100 = lr1['val_se100_poor_outcome']
lr1_se100 = lr1_se100.tolist()

lr2 = lr [lr['lr']==0.0005]
lr2_auc = lr2['val_auc']
lr2_auc = lr2_auc.tolist()
lr2_se100 = lr2['val_se100_poor_outcome']
lr2_se100 = lr2_se100.tolist()

lr3 = lr [lr['lr']==0.001]
```

```
lr3_auc = lr3['val_auc']
lr3_auc = lr3_auc.tolist()
lr3_se100 = lr3['val_se100_poor_outcome']
lr3_se100 = lr3_se100.tolist()

lr4 = lr [lr['lr']==0.005]
lr4_auc = lr4['val_auc']
lr4_auc = lr4_auc.tolist()
lr4_se100 = lr4['val_se100_poor_outcome']
lr4_se100 = lr4_se100.tolist()

lr5 = lr [lr['lr']==0.01]
lr5_auc = lr5['val_auc']
lr5_auc = lr5_auc.tolist()
lr5_se100 = lr5['val_se100_poor_outcome']
lr5_se100 = lr5_se100.tolist()

## AUC ##

# Boxplot
lr_auc = [lr1_auc, lr2_auc, lr3_auc, lr4_auc, lr5_auc]
fig_lr_auc, ax_lr_auc = plt.subplots()
ax_lr_auc.set_title('Boxplots of validation AUC of the learning rates')
ax_lr_auc.boxplot(lr_auc)
plt.xlabel('Learning rate')
plt.ylabel('Validation AUC')
plt.xticks([1, 2, 3, 4, 5], lr_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

corr_lr_auc = dfm[['val_auc','lr']]
corr_lr_auc =  corr_lr_auc.corr(method='pearson')

# Mean
mean_lr1_auc = mean(lr1_auc)
mean_lr2_auc = mean(lr2_auc)
mean_lr3_auc = mean(lr3_auc)
mean_lr4_auc = mean(lr4_auc)
mean_lr5_auc = mean(lr5_auc)
#mean_lr_auc = mean(lr_auc,1)
mean_lr_auc = [mean_lr1_auc, mean_lr2_auc, mean_lr3_auc, mean_lr4_auc, mean_lr5_auc]
fig_lrmean_auc, ax_lrmean_auc = plt.subplots()
ax_lrmean_auc.set_title('Mean validation AUC per learning rate')
ax_lrmean_auc.plot(lr_names,mean_lr_auc)
plt.grid(color='black', linestyle='--', linewidth=0.2)

# Median
med_lr1_auc = median(lr1_auc)
med_lr2_auc = median(lr2_auc)
med_lr3_auc = median(lr3_auc)
med_lr4_auc = median(lr4_auc)
med_lr5_auc = median(lr5_auc)
#median_lr_auc = np.median(lr_auc,1)
median_lr_auc = [med_lr1_auc, med_lr2_auc, med_lr3_auc, med_lr4_auc, med_lr5_auc]
fig_lrmed_auc, ax_lrmed_auc = plt.subplots()
ax_lrmed_auc.set_title('Median validation AUC per learning rate')
ax_lrmed_auc.plot(lr_names,median_lr_auc)
plt.grid(color='black', linestyle='--', linewidth=0.2)

## SE100 ##

# Boxplot
lr_se100 = [lr1_se100, lr2_se100, lr3_se100, lr4_se100, lr5_se100]
fig_lr_se100, ax_lr_se100 = plt.subplots()
ax_lr_se100.set_title('Boxplots of the validation SeSp100 of the learning rates')
ax_lr_se100.boxplot(lr_se100)
plt.xlabel('Learning rate')
```

```
plt.ylabel('Validation SeSp100')
plt.xticks([1, 2, 3, 4, 5], lr_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

corr_lr_se100 = dfm[['val_se100_poor_outcome','lr']]
corr_lr_se100 =  corr_lr_se100.corr(method='pearson')

# Mean
mean_lr1_se100 = mean(lr1_se100)
mean_lr2_se100 = mean(lr2_se100)
mean_lr3_se100 = mean(lr3_se100)
mean_lr4_se100 = mean(lr4_se100)
mean_lr5_se100 = mean(lr5_se100)
# mean_lr_se100 = mean(lr_se100,1)
mean_lr_se100 = [mean_lr1_se100, mean_lr2_se100, mean_lr3_se100, mean_lr4_se100, mean_lr5_se100]
fig_lrmean_se100, ax_lrmean_se100 = plt.subplots()
ax_lrmean_se100.set_title('Mean validation Se100 of poor outcome prediction per learning rate')
ax_lrmean_se100.plot(lr_names,mean_lr_se100)
plt.grid(color='black', linestyle='--', linewidth=0.2)

# Median
med_lr1_se100 = median(lr1_se100)
med_lr2_se100 = median(lr2_se100)
med_lr3_se100 = median(lr3_se100)
med_lr4_se100 = median(lr4_se100)
med_lr5_se100 = median(lr5_se100)
#median_lr_se100 = np.median(lr_se100,1)
median_lr_se100 = [med_lr1_se100, med_lr2_se100, med_lr3_se100, med_lr4_se100, med_lr5_se100]

fig_lrmed_se100, ax_lrmed_se100 = plt.subplots()
ax_lrmed_se100.set_title('Median validation Se100 of poor outcome prediction per learning rate')
ax_lrmed_se100.plot(lr_names,median_lr_se100)
plt.grid(color='black', linestyle='--', linewidth=0.2)


# Mean both metrics in 1
fig_lr, ax_lr = plt.subplots()
ax_lr.set_title('validation AUC and Se100 of poor outcome prediction per learning rate')
ax_lr.plot(lr_names,mean_lr_auc,'g-',label='mean AUC', linewidth=1)
ax_lr.plot(lr_names,mean_lr_se100,'g--',label='mean Se100', linewidth=1)
ax_lr.plot(lr_names,median_lr_auc,'y-',label='median AUC', linewidth=1)
ax_lr.plot(lr_names,median_lr_se100,'y--',label='median Se100', linewidth=1)
ax_lr.legend(loc='center right')
plt.grid(color='black', linestyle='--', linewidth=0.2)


##########################################
##########################################
# eliminate bad results from boxplots #
##########################################
##########################################


# Results 1 layer LSTM analysis:
df = dfm[['val_auc','val_se100_poor_outcome','dropout1','recurrent_dropout1','hidden_units_LSTM1','epochs','lr']]
df = df[(df['hidden_units_LSTM1']>=10) & (df['epochs']>=120) & (df['lr']>=0.001)]

# Results 2 layer LSTM analysis:
df = dfm[['val_auc','val_se100_poor_outcome','dropout','recurrent_dropout','hidden_units_LSTM','epochs','lr']]
```

```
df = df[ (df['hidden_units_LSTM']>=16) & (df['epochs']>=80) & (df['epochs']<=140)& (df['lr']==0.005)]

# ADJUST CODE TO 1 OR 2 LAYER ANALYSIS
# below shows results for 2 layer LSTM

# epochs
e = df[['val_auc','val_se100_poor_outcome','epochs']]
e_names = ['80','100','120','140']

e80 = e[e['epochs']==80]
e80_auc = e80['val_auc']
e80_auc = e80_auc.tolist()
e80_se100 = e80['val_se100_poor_outcome']
e80_se100 = e80_se100.tolist()

e100 = e[e['epochs']==100]
e100_auc = e100['val_auc']
e100_auc = e100_auc.tolist()
e100_se100 = e100['val_se100_poor_outcome']
e100_se100 = e100_se100.tolist()

e120 = e[e['epochs']==120]
e120_auc = e120['val_auc']
e120_auc = e120_auc.tolist()
e120_se100 = e120['val_se100_poor_outcome']
e120_se100 = e120_se100.tolist()

e140 = e[e['epochs']==140]
e140_auc = e140['val_auc']
e140_auc = e140_auc.tolist()
e140_se100 = e140['val_se100_poor_outcome']
e140_se100 = e140_se100.tolist()


e_auc = [e80_auc, e100_auc, e120_auc, e140_auc]
fig_e_auc, ax_e_auc = plt.subplots()
ax_e_auc.set_title('Boxplots of the validation AUC of the number of epochs \n (bad performing hyperparameters excluded)')
ax_e_auc.boxplot(e_auc)
plt.xlabel('Number of epochs')
plt.ylabel('Validation AUC')
plt.xticks([1, 2, 3, 4], e_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

e_se100 = [e80_se100, e100_se100, e120_se100, e140_se100]
fig_e_se100, ax_e_se100 = plt.subplots()
ax_e_se100.set_title('Boxplots of the validation SeSp100 of the number of epochs \n (bad performing hyperparameters excluded)')
ax_e_se100.boxplot(e_se100)
plt.xlabel('Number of epochs')
plt.ylabel('Validation SeSp100')
plt.xticks([1, 2, 3, 4], e_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

# units
hu = df[['val_auc','val_se100_poor_outcome','hidden_units_LSTM']]
hu_names = ['16','18','20','22','24']

hu8 = hu[hu['hidden_units_LSTM']==16]
hu8_auc = hu8['val_auc']
hu8_auc = hu8_auc.tolist()
hu8_se100 = hu8['val_se100_poor_outcome']
hu8_se100 = hu8_se100.tolist()

hu9 = hu[hu['hidden_units_LSTM']==18]
```

```
hu9_auc = hu9['val_auc']
hu9_auc = hu9_auc.tolist()
hu9_se100 = hu9['val_se100_poor_outcome']
hu9_se100 = hu9_se100.tolist()

hu10 = hu[hu['hidden_units_LSTM']==20]
hu10_auc = hu10['val_auc']
hu10_auc = hu10_auc.tolist()
hu10_se100 = hu10['val_se100_poor_outcome']
hu10_se100 = hu10_se100.tolist()

hu11 = hu[hu['hidden_units_LSTM']==22]
hu11_auc = hu11['val_auc']
hu11_auc = hu11_auc.tolist()
hu11_se100 = hu11['val_se100_poor_outcome']
hu11_se100 = hu11_se100.tolist()

hu12 = hu[hu['hidden_units_LSTM']==24]
hu12_auc = hu12['val_auc']
hu12_auc = hu12_auc.tolist()
hu12_se100 = hu12['val_se100_poor_outcome']
hu12_se100 = hu12_se100.tolist()

hu_auc = [hu8_auc, hu9_auc, hu10_auc,hu11_auc, hu12_auc]
fig_hu_auc, ax_hu_auc = plt.subplots()
ax_hu_auc.set_title('Boxplots of the validation AUC of the number of units \n (bad performing
    hyperparameters excluded)')
ax_hu_auc.boxplot(hu_auc)
plt.xlabel('Number of units')
plt.ylabel('Validation AUC')
plt.xticks([1, 2, 3, 4, 5], hu_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

hu_se100 = [hu8_se100, hu9_se100, hu10_se100, hu11_se100, hu12_se100]
fig_hu_se100, ax_hu_se100 = plt.subplots()
ax_hu_se100.set_title('Boxplots of the validation SeSp100 of the number of units \n (bad perfo
    rming hyperparameters excluded)')
ax_hu_se100.boxplot(hu_se100)
plt.xlabel('Number of units')
plt.ylabel('Validation SeSp100')
plt.xticks([1, 2, 3, 4, 5], hu_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)


## DO & RDO
hu20_e80_lr2 = df[(df['hidden_units_LSTM']==20) & (df['epochs']==80) & (df['lr']==0.005)]
best_1 = hu20_e80_lr2[['val_auc','val_se100_poor_outcome','dropout','recurrent_dropout']]
corr_b1 = best_1.corr(method='pearson')

# dropout
do = df[['val_auc','val_se100_poor_outcome','dropout']]
do_names = ['0','0.2','0.5']

do0 = do [do['dropout']==0]
do0_auc = do0['val_auc']
do0_auc = do0_auc.tolist()
do0_se100 = do0['val_se100_poor_outcome']
do0_se100 = do0_se100.tolist()

do02 = do [do['dropout']==0.2]
do02_auc = do02['val_auc']
do02_auc = do02_auc.tolist()
do02_se100 = do02['val_se100_poor_outcome']
do02_se100 = do02_se100.tolist()

do05 = do [do['dropout']==0.5]
```

```
do05_auc = do05['val_auc']
do05_auc = do05_auc.tolist()
do05_se100 = do05['val_se100_poor_outcome']
do05_se100 = do05_se100.tolist()

do_auc = [do0_auc, do02_auc, do05_auc]
fig_do_auc, ax_do_auc = plt.subplots()
ax_do_auc.set_title('Boxplots of the validation AUC of the dropout rates \n (bad performing hyperparameters excluded)')
ax_do_auc.boxplot(do_auc)
plt.xlabel('Dropout rate')
plt.ylabel('Validation AUC')
plt.xticks([1, 2, 3], do_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

do_se100 = [do0_se100, do02_se100, do05_se100]
fig_do_se100, ax_do_se100 = plt.subplots()
ax_do_se100.set_title('Boxplots of the validation SeSp100 of the dropout rates \n (bad performing hyperparameters excluded)')
ax_do_se100.boxplot(do_se100)
plt.xlabel('Dropout rate')
plt.ylabel('Validation SeSp100')
plt.xticks([1, 2, 3], do_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

# recurrent dropout
rdo = df[['val_auc','val_se100_poor_outcome','recurrent_dropout']]
rdo_names = ['0','0.2','0.5']

rdo0 = rdo [rdo['recurrent_dropout']==0]
rdo0_auc = rdo0['val_auc']
rdo0_auc = rdo0_auc.tolist()
rdo0_se100 = rdo0['val_se100_poor_outcome']
rdo0_se100 = rdo0_se100.tolist()

rdo02 = rdo [rdo['recurrent_dropout']==0.2]
rdo02_auc = rdo02['val_auc']
rdo02_auc = rdo02_auc.tolist()
rdo02_se100 = rdo02['val_se100_poor_outcome']
rdo02_se100 = rdo02_se100.tolist()

rdo05 = rdo [rdo['recurrent_dropout']==0.5]
rdo05_auc = rdo05['val_auc']
rdo05_auc = rdo05_auc.tolist()
rdo05_se100 = rdo05['val_se100_poor_outcome']
rdo05_se100 = rdo05_se100.tolist()

rdo_auc = [rdo0_auc, rdo02_auc, rdo05_auc]
fig_rdo_auc, ax_rdo_auc = plt.subplots()
ax_rdo_auc.set_title('Boxplots of the validation AUC of the recurrent dropout rates \n (bad performing hyperparameters excluded)')
ax_rdo_auc.boxplot(rdo_auc)
plt.xlabel('Recurrent dropout rate')
plt.ylabel('Validation AUC')
plt.xticks([1, 2, 3], rdo_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)

rdo_se100 = [rdo0_se100, rdo02_se100, rdo05_se100]
fig_rdo_se100, ax_rdo_se100 = plt.subplots()
ax_rdo_se100.set_title('Boxplots of the validation SeSp100 of the recurrent dropout rates \n (bad performing hyperparameters excluded)')
ax_rdo_se100.boxplot(rdo_se100)
plt.xlabel('Recurrent dropout rate')
plt.ylabel('Validation SeSp100')
plt.xticks([1, 2, 3], rdo_names)
plt.grid(color='black', linestyle='--', linewidth=0.2)
```

Z.9 Final LSTM

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on 31-12 2020

@author: lauravanpoppel
"""

import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Bidirectional, BatchNormalization, A
ctivation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l1_l2
from tensorflow.keras.metrics import Accuracy, AUC, Precision, SensitivityAtSpecificity, Speci
ficityAtSensitivity
from tensorflow.keras.metrics import TruePositives, TrueNegatives, FalsePositives, FalseNegati
ves

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.stats.api as sms

from sklearn.model_selection import train_test_split, KFold, StratifiedKFold
from sklearn.metrics import roc_curve, auc
from scipy.io import loadmat

# import 2D structure
dataMatlab = loadmat('LSTM_input4.mat');
data = dataMatlab.get('LSTM_input4');

# define features and targets
features = data[:,:12];
outcomes = data[:,12]; # 1 is poor outcome

# reshape to input matrices and outcome vector
timesteps = 30;
predictors = 12;
subjects = int(features.shape[0]/timesteps);

x = features.reshape(subjects,timesteps,predictors);
y = outcomes.reshape(subjects,timesteps);
y = y[:,0];

# repeat process 50 times
repeats = 50
test_loss_per_repeat = []
test_acc_per_repeat = []
test_auc_per_repeat = []
test_se100_per_repeat = []
test_se95_per_repeat = []


test_loss_repeat = []
test_acc_repeat = []
test_auc_repeat = []
test_se100_repeat = []
test_se95_repeat = []


for i in range(repeats):

```

```
    # use 10-fold cross validation
    num_folds = 10;
    fold_no = 1

    # Define per-fold score containers
    test_loss_per_fold = [];
    test_acc_per_fold = [];
    test_auc_per_fold = [];
    test_se100_per_fold = [];
    test_se95_per_fold = [];
    test_fpr_per_fold = [];
    test_tpr_per_fold = [];

    ######################
    # Define parameters #
    ######################
    # first LSTM layer
    hidden_units_LSTM1 = 16
    input_LSTM = (x.shape[1:])
    drop_out1 = 0.5
    r_drop_out1= 0

    k_reg=l1_l2(l1=0.001, l2=0.001)
    r_reg=l1_l2(l1=0.001, l2=0.001)
    b_reg=l1_l2(l1=0.001, l2=0.001)

    # fully connected layer
    activation_function_FC = 'sigmoid'
    hidden_units_FC = 1

    # compile
    opt = Adam(learning_rate=0.001)
    loss_function = 'binary_crossentropy'
    metric1 = 'accuracy'
    metric2 = AUC(name='auc')
    metric3 = SensitivityAtSpecificity(1, name='se100_poor_outcome')
    metric4 = SpecificityAtSensitivity(0.95, name='se95_good_outcome')

    # fit
    epoch_number = 120
    batch_size = 32

    # Define the K-fold Cross Validator
    skf = StratifiedKFold(n_splits=num_folds, shuffle=True)

    for train, test in skf.split(x, y):


        ################
        # Build model #
        ################
        model = Sequential()
        model.add(LSTM(hidden_units_LSTM1,
                       input_shape=input_LSTM,
                       kernel_regularizer=k_reg,
                       recurrent_regularizer=r_reg,
                       bias_regularizer=b_reg,
                       dropout=drop_out1,
                       recurrent_dropout=r_drop_out1,
                       ))

        model.add(Dense(hidden_units_FC,
                        activation=activation_function_FC))

        ##################
        # Compile model #
```

```
        ##################
        model.compile(loss=loss_function,
                      optimizer=opt,
                      metrics=[metric1, metric2, metric3, metric4])

        # Generate a print
        print('------------------------------------------------------------------------')
        print(f'Training for fold {fold_no} ...')

        ##############
        # Fit model #
        ##############
        history = model.fit(x[train],
                            y[train],
                            epochs=epoch_number,
                            batch_size = batch_size,
                            verbose=0)

        # Generate generalization metrics
        scores = model.evaluate(x[test], y[test], verbose=0)
        test_loss_per_fold.append(scores[0])
        test_acc_per_fold.append(scores[1]*100)
        test_auc_per_fold.append(scores[2])
        test_se100_per_fold.append(scores[3])
        test_se95_per_fold.append(1-(scores[4]))

        # Metrics for ROC
        y_predict = model.predict(x[test])
        fpr , tpr , thresholds = roc_curve (y[test],y_predict)


        plt.plot(fpr,tpr)
        plt.axis([0,1,0,1])
        plt.title('ROC curve')
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')

        # Increase fold number
        fold_no = fold_no + 1

    plt.show()

    # == Provide average scores ==
    print('------------------------------------------------------------------------')
    print('Average test scores for all folds:')
    print(f'> Loss: {np.mean(test_loss_per_fold)} (+- {np.std(test_loss_per_fold)})')
    print(f'> AUC: {np.mean(test_auc_per_fold)} (+- {np.std(test_auc_per_fold)})')
    print(f'> Se100 poor outcome: {np.mean(test_se100_per_fold)} (+- {np.std(test_se100_per_fold)})')
    print(f'> Se95 good outcome: {np.mean(test_se95_per_fold)} (+- {np.std(test_se95_per_fold)})')
    print('------------------------------------------------------------------------')


    # append metrics for all repeats
    test_loss_per_repeat.append(test_loss_per_fold)
    test_acc_per_repeat.append(test_acc_per_fold)
    test_auc_per_repeat.append(test_auc_per_fold)
    test_se100_per_repeat.append(test_se100_per_fold)
    test_se95_per_repeat.append(test_se95_per_fold)

    # calculate 95% CI
    test_loss_repeat.extend(test_loss_per_fold)
    ci_loss = sms.DescrStatsW(test_loss_repeat).tconfint_mean()
    test_acc_repeat.extend(test_acc_per_fold)
    ci_acc = sms.DescrStatsW(test_acc_repeat).tconfint_mean()
```

```
    test_auc_repeat.extend(test_auc_per_fold)
    ci_auc = sms.DescrStatsW(test_auc_repeat).tconfint_mean()
    test_se100_repeat.extend(test_se100_per_fold)
    ci_se100 = sms.DescrStatsW(test_se100_repeat).tconfint_mean()
    test_se95_repeat.extend(test_se95_per_fold)
    ci_se95 = sms.DescrStatsW(test_se95_repeat).tconfint_mean()


print('------------------------------------------------------------------------')
print('Average test scores for all repeats:')
print(f'> Loss: {np.mean(test_loss_per_repeat)} std:(+- {np.std(test_loss_per_repeat)}) 95%CI:{ci_loss}')
print(f'> AUC: {np.mean(test_auc_per_repeat)} std:(+- {np.std(test_auc_per_repeat)}) 95%CI:{ci_auc}')
print(f'> Poor Se100: {np.mean(test_se100_per_repeat)} std:(+- {np.std(test_se100_per_repeat)}) 95%CI:{ci_se100}')
print(f'> Good Se95: {np.mean(test_se95_per_repeat)} std:(+- {np.std(test_se95_per_repeat)}) 95%CI:{ci_se95}')
print('------------------------------------------------------------------------')
```