Imputation of missing time series values using statistical and mathematical strategies

Tomas Rakvåg Ulriksborg

Thesis submitted for the degree of
Master in Informatics: Programming and System Architecture
60 credits

Department of Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2022

Imputation of missing time series values using statistical and mathematical strategies

Tomas Rakvåg Ulriksborg

Imputation of missing time series values using statistical and mathematical strategies

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

In this thesis, we looked at different approaches for imputation of missing values in time series. We carried out experiments and compared how these approaches affected machine learning models from two different machine learning libraries. After selecting and comparing models across both libraries we tested these models up against both multivariate and univariate time series regression for predicting readiness to play in athletes. The experimentation saw different stages, finding the optimal parameters for the models, testing to find the best performing combination of imputation approaches, models and training method, and a final testing on multi-step forecasting. After testing we compared and discussed the results based on the accuracy score of the models along with looking at how the models behaved differently on the imputation approaches and training methods. Finally, we tried to draw some conclusions based on the results gotten as well as what would be some interesting ideas for future research.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Problem statement 2

2 Background and related work **4**

2.1 PmSys - smartphone-based athlete monitoring system 4

2.1.1 Introduction 4

2.1.2 Users 5

2.1.3 Functionality 5

2.1.4 Architecture 6

2.1.5 Parameters 7

2.1.6 The flow of PmSys 8

2.2 Technologies 9

2.2.1 Machine learning 9

2.2.2 Supervised learning 9

2.2.3 Neural networks 11

2.2.4 Time Series 11

2.2.5 Data analysis 12

2.2.6 Model training 13

2.2.7 TSAI library 15
2.2.8 Keras 15
2.2.9 Models used 16
2.3 Related studies 21
2.3.1 LSTM for peak readiness 22
2.3.2 Case study on self-reporting symptoms on triathletes 23
2.4 Summary 24

3 Methodology **27**
3.1 System specification 27
3.2 Dataset 28
3.2.1 NaN handling filling with zero values 29
3.2.2 NaN handling filling with mean values 29
3.2.3 NaN handling filling with interpolate values 29
3.2.4 Univariate 30
3.2.5 Multivariate 30
3.2.6 Timesteps and prediction clarification 31
3.2.7 Multi-step forecasting 33
3.2.8 Data splits for training, testing and validation sets . . 35
3.3 Keras and Tsai Models 35

4 Experiments and results **37**
4.1 Experiments overview 37
4.2 Initial experiments 39
4.2.1 Overview 39
4.2.2 Results for Tsai 41

4.2.3 Results for Keras 44
4.3 Discussion of initial experimentation 47
4.4 Filling Nan's with zero/mean/interpolate Tsai 49
4.4.1 Overview 49
4.4.2 Multivariate training on all players 50
4.4.3 Multivariate training on one player's self report data 53
4.4.4 Univariate training 56
4.4.5 Comparing the results from training methods 58
4.5 Filling Nan's with zero/mean/interpolate Keras 63
4.5.1 Overview 63
4.5.2 Multivariate training on all players 65
4.5.3 Multivariate training on one player's self report data 67
4.5.4 Univariate training 68
4.6 Discussion of imputation results 69
4.7 Experiment with univariate multi-step prediction 71
4.7.1 Overview 71
4.7.2 NaN handling with interpolated values 72
4.7.3 NaN handling with mean values 73
4.7.4 NaN handling with zero values 75
4.8 Discussion of multi-step forecasting 76
4.9 Summary 78

5 Conclusion and future work **79**

List of Figures

2.1 Wellness timeline with one player highlighted[20] 5

2.2 Body silhouette taken from mobile app 8

2.3 Injury summary taken from mobile app 8

2.4 Example of typical time series graph showing both seasonal shifts and predictable trends [2] 12

2.5 Graph showing features (Readiness, Sleep Duration, Sleep Quality, Stress, Fatigue and Session RPE divided by 100). Useful graph for understanding the relation between all features on a numerical level 13

2.6 What the features in an array of 7 does to the accuracy of the model. Features seen as var 0-6 shown with their represented accuracy change of removed 16

2.7 Figure shows an example of semantic segmentation where Fully Convolutional Networks can learn by upsampling the output size when wanted. Figure source: [15] 18

2.8 Figure shows an example of a multivariate time series problem and how the inside of Inception treats it, one of the modules in InceptionTime.[5] The steps of Inception goes as follows: First the input data is sent to a bottleneck which in this example is a bottleneck with the dimension of 1. The bottleneck transforms the time series to a multivariate time series with M dimensions, 1 in this case. This reduces the complexity and dimension of the model and in turn will give some prevention for overfitting. After this the output of the bottleneck is filtered with three convolutions in this example and concatenated with a parallel MaxPooling operation from the same time series which has gone through a bottleneck as well. 19

2.9 The structure of LSTM cells Figure source: [16] 21

2.10 The layers inside an LSTM cell, figure source: [8] 21

2.11 Figure shows one single GRU cell. Red circle indicating sigmoid functions and blue tanh. figure source: [13] 22

3.1 Figure shows what is estimated in linear interpolation (Interpolated point P) and the formula for this calculation, figure sources [11] [19] 30

3.2 Difference of filling NaNs with zero, mean value and interpolate values. The index of this dataset is dates, but to get a visual on the dataset it is indexed in an observational order 31

3.3 Figure shows multivariate dataset with Readiness, Sleep, Fatigue, Stress, SleepQuality and PE(Perceived Exertion). This particular set is also a set used for multivariate training on one player and predicting the Readiness level on that player. 32

3.4 Figure shows how strafe and window length looks like on a series of N length. This particular figure has a window of 4 and trafe of 1. Feeding this to a configured model will use the 4 values from x_0 to x_3 for predicting the value of x_4. Next prediction will use the values x_1 to x_4 for prediction of the next value. These two prediction will use some of the same values since the window is of size 4 and strafe is only 1. There are many different ways of configuring these types of models, having a strafe of 1 being fairly common. Figure source: [18] 33

3.5 Figure illustrates multi-step prediction/forecasting. In this example the window size is 9 and the last white box is the day to be predicted. When that day is predicted the window shifts one day ahead and puts the predicted value in to this window. This means the model will use the predicted value to further predict what values comes next. Figure source: [12] 34

3.6 Figure shows the split distribution of training, validation and test set 35

4.1 Figure shows initial experimentation results from training on all players readiness levels and predicting readiness of one player 42

4.2 Figure shows initial experimentation results from training on one players self report data and predicting on the same player 43

4.3 Figure shows initial experimentation results from univariate training 44

4.4 Figure shows how we designed the models used in experimentation for Keras. Each blue box represents a layer with the arrows pointing to the next layer. From each layer, the input is processed and passed on as output to the next layer. Once it reaches the dense layer the output will be narrowed down to a single output for each datapoint fed to the model. Every model has a dropout dividing the middle layer with the same dropout rate: 0.2. 45

4.5 Figure shows what the features in an array of 7 does to the accuracy of the model. Lower score results in a higher model accuracy 47

4.6 Figure shows how the models generally performed during initial experimentation 49

4.7 Figure shows results from training on all players with interpolated values as replacements with the best window size of 3 51

4.8 Figure shows results from training on all players with mean value NaN replacement 53

4.9 Figure shows results from training on all players with zero value replacement 54

4.10 Figure shows results from training on one player with interpolated values replacement dataset 55

4.11 Figure shows results from training on one player with mean value replacement dataset 55

4.12 Figure shows results from training on one player with zero value replacement dataset 56

4.13 Figure shows results from univariate training with interpolated values as replacements dataset 57

4.14 Figure shows results from univariate training with mean value replacements dataset 58

4.15 Figure shows results from univariate training with zero value replacements dataset 59

4.16 Best and worst models from interpolation replacement on multivariate training on all players 60

4.17 Best and worst models from mean value replacement on multivariate training on all players 60

4.18 Best and worst models from zero value replacement on multivariate training on all players 60

4.19 Best and worst models from interpolation replacement on multivariate training on one player 62

4.20 Best and worst models from mean value replacement on multivariate training on one player 62

4.21 Best and worst models from zero value replacement on multivariate training on one player 62

4.22 Best and worst models from interpolation replacement on univariate training 64

4.23 Best and worst models from mean value replacement on univariate training 64

4.24 Best and worst models from zero value replacement on univariate training 64

4.25 LSTM, GRU and RNN graphs from multi-step forecasting on interpolated dataset with a window size of 3 days 75

4.26 LSTM, GRU and RNN graphs from multi-step forecasting on mean value dataset with a window size of 3 days 76

4.27 LSTM, GRU and RNN graphs from multi-step forecasting on zero value dataset with a window size of 3 days 77

List of Tables

2.1 Self report parameters. 7

3.1 System specifications. 28

4.1 Table shows the configurations of hyperparameters and other relevant information used for initial experimentation. . 40

4.2 Table shows results from multivariate training on all predicting on one in Keras 45

4.3 Table shows results from multivariate training on one predicting on one in Keras 46

4.4 Table shows results from univariate training in Keras 47

4.5 Table shows updated overview of hyperparameters and other useful information used in further experimentation . . 50

4.6 Table shows the best models when training on all players. . 59

4.7 Table shows the best models when training on one player. . 61

4.8 Table shows the best models when training with univariate set. 63

4.9 Table shows results from **LSTM** training on three datasets, training on all players(all), training on one players self reported data(one) and training on the univariate dataset(Uni). Training was done on the three different NaN handling methods, **Interpolation**, **Mean value** and **Zero values**. 65

4.10 Table shows results from **GRU** training on three datasets, training on all players(all), training on one players self reported data(one) and training on the univariate dataset(Uni). Training was done on the three different NaN handling methods, **Interpolation**, **Mean value** and **Zero values**. 66

4.11 Table shows results from **RNN** training on three datasets, training on all players(all), training on one players self reported data(one) and training on the univariate dataset(Uni). Training was done on the three different NaN handling methods, **Interpolation**, **Mean value** and **Zero values**. 66

4.12 Table shows the best models for multivariate training on all players with Keras. 67

4.13 Table shows the best models for multivariate training on one player with Keras. 68

4.14 Table shows the best models for univariate training with Keras. 69

4.15 Table consists of the MSPE scores gotten from training on univariate time series with interpolated values for NaN replacement, divided into the three models ran on multi-step forecasting 73

4.16 Table consists of the MSPE scores gotten from training on univariate time series with mean values for NaN replacement, divided into the three models ran on multi-step forecasting 74

4.17 Table consists of the MSPE scores gotten from training on univariate time series with zero values for NaN replacement, divided into the three models ran on multi-step forecasting 74

Chapter 1

Introduction

In this day and age everything is stores as data. Technologies and companies capture and store everything which might have the slightest beneficial factor. From all this data new techniques of handling data is discovered and put to the test. Storing data in a timely matter can be considered to be time series. Time series is interesting for many reasons, one of them being the potential of looking into the future. Machine learning is something that has gotten a lot of praise in the media lately for how it can use time series to look in to the future. How well this actually works is something that has peaked the interest of many industries one of them being the sports industry.

1.1 Motivation

With technologies advancing and becoming a part of our daily routine, it has also become a helping hand in driving people's abilities ever further. In sport science and sport performance, technology has become a huge part in helping athletes becoming better versions of themselves. It has the ability to see even more than what humans see with the naked eye. Getting new and precise information about performance in both training and competition can be analyzed and used for improvement. For instance, in the world of soccer, analytic systems have shown to have a great impact on the game, with for example, coach Joachim Low calling a substitute of Mario Götze to arguably win the world cup final in Brazil 2014. The Germany based software company called SAP played a huge role in Germany's win. They were the ones providing the data and analysis of matches and players of the German national soccer team. Capturing and analyzing data from video cameras around the field, combining it with

player's speed, distance travelled and positioning, made up the dataset for what was needed to improve the soccer team's success.

There are many reasons being able to predict readiness of players can be profitable. Going inn to a match, players need to be at their absolute best in order to perform their absolute best, if a player is weakened or injured by heavy workload from training this will impact their performance in the match. Making a good workout/training plan is about finding the threshold an athlete can work at for performance increase with respect to how long or how well the athlete's body or mind can work at that pace/workload. In order to find an athlete's threshold, we need to be collecting and analyzing data from their performance in match days and training days. This is exactly what PMSys intend to do, closing the gap between coaches and athletes in an efficient way introducing a platform for data visualization, statistics and prediction of how the athletes states are. PMSys is an athlete monitoring system developed by Simula Research Laboratory along with University of Tromsø and ForzaSys. With technologies advancing, machine learning has become useful when looking at athletes performance. When using machine learning to predicting the states of athletes, consistency is a driving force. Having data scattered randomly across a time frame will not unsure the best predictions because of the irregularities in the data. In a system where data is captured by the users themselves, it will be prone to a few issues along the lines of data consistency. Some days of data might be missing which in turn might decrease potential accuracy of these predictions. Addressing inconsistency in the form of missing data can a yield boosting capabilities for machine learning, thus sparking the questions of how to deal with missing data.

1.2 Problem statement

As discussed earlier, when working with users you are prone to run into some inconsistencies. Since users are a part of PMSys, the athlete monitoring system we described in the prior chapter, the system om PMSys is prone to running into the same issues. Simula Research Laboratory and partners encounter the problem of users not registering data for some days, resulting in a time series with days consisting of no values. If this time series is to be used in any beneficial way these missing data points needs to be dealt with. PMSys is looking to give the users of the system even more functionality, introducing machine learning to predict readiness to play in athletes of a team. Combining machine learning with time series introduces a combination of new potential analysis which can be beneficial for teams and athletes wanting that extra insight in their per-

formance. Getting that extra insight into their performance might lend a helping hand in helping the athletes better plan their workouts or helping them realise some things the eyes might miss, like figuring out how their bodies might respond to exertion when training. One way technology might give a handy insight into this is by using the time series data to predict the readiness values of athletes the future days. Machine learning and time series can be combined to achieve just this, but how machine learning might respond to the missing data is something which PMSys has not been experimenting on quite yet. This raises the question for this thesis:

How efficient can statistical and mathematical strategies be used to impute missing values in time series data and how will it affect performance in machine learning?

In the following chapters we will be looking at researching different types of ways of imputing missing values from the world of mathematics and statistics. We will be using the different imputing approaches on time series data from PMSys and testing how efficient they work on different machine learning approaches and models. We will also give an interpretation of the results, how the different imputation approaches affected performance from both the perspective of machine learning and PMSys.

Chapter 2

Background and related work

This chapter will introduce PMSys, what it does, how it operates and why. PMSys is a big system of many moving parts which will be described, everything from the back of the system to what can be seen by the users. Since PMSys operates with large amount of data and data collection, the steps from collecting to using the data will be described. Every data collected and processed is related to athletes and how their performance might fluctuate over time, this will make the data time series which will be the main part of this thesis. Along with describing how time series work, previous work related to PMsys will be described.

2.1 PmSys - smartphone-based athlete monitoring system

2.1.1 Introduction

There are many ways of collecting and analysing data related to athletes performances. Depending on which type of sport the context is, it is important to find what data should be focused on. For example, triathletes and distance athletes need to know what affects their ability to sustain muscle fatigue. One of the many ways this can be measured is by testing the blood lactate levels. This is often done in the middle and/or after a workout focusing on raising the lactate level. This data could be plotted in somewhere to be further analyzed by coaches and team members in order to help the athlete better respond and prepare workouts improving this measurement. Seeing what relates to this lactate level and how it trends over a period of time can be done to understand the athlete's performance.

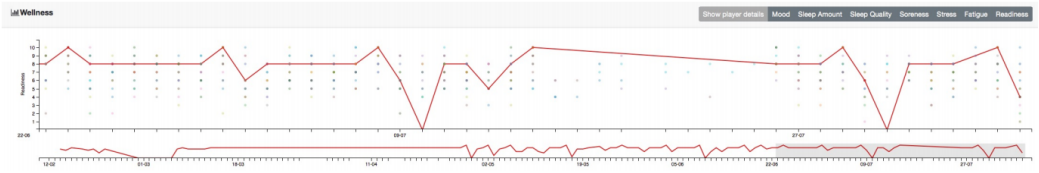

Figure 2.1: Wellness timeline with one player highlighted[20]

PmSys focuses on athletes in soccer. Pål and colleagues have made a self-reporting system customized with inputs focused on wellness, injury and illness for athletes.

2.1.2 Users

PMSys is generally made for organization, institutes and sport teams across Norway. From this there will be a variety of users involved. The system is both made for coaches and players and the intention is to bring insight and functionality to both parties. From PMSys' own website they showcase some of the teams and institutes using the system; The Norwegian International Football Team, Viking FK, MFK, Norges Toppidressgymnas, RBK and many more. For the athletes using this system they will get to see their own reported data, keeping the data logged and accessible for review and easily communicated to coaches and team members. Coaches has their own page for monitoring team players with graphs, stats and customizable data shown in a informational way.

2.1.3 Functionality

Logging data relevant for analysis and use in measuring performance, wellness, injury and illness for athletes. A smartphone-based application where the athletes on a daily base self-report their parameters for research. Designed on the principle that these subjective reports are captured in real-time with minimal effort, while the data is fresh and relevant. Team personnel, like coaches and physicians, got their own portal which displays the data submitted as can be seen in figure 2.1. The data is displayed and divided into two different pages, team centered and single-player centered. For the team view, the data can be arranged to visualize for instance injuries, illnesses and session participation. The data can be sorted to daily and weekly load, acute load, chronic load, acute chronic workload ratio, monotony and strain. The single player view is showing

all relevant data from each individual player. The portal provides push messages to players from team personnel, which has increased participation in the project.

2.1.4 Architecture

In order for PMSys to work, there are a few building blocks to support the load of athletes self reports and analysis. PMSys's architecture can be divided into 4 different layers:

- Mobile app
- Data Storage Unit
- Policy Server Unit
- PMSys trainer

The smartphone application is both available on iOS and Android. Made easy to download and set up for use. In the app you get a clear overview of reports for a specific time period, and you can easily make a report of every parameter used by PMSys. The report is made with a simple questionnaire with already defined values to choose from, both making it more consistent and easy to use.

Every player report is stored in one or more Data Storage Units. PMSys uses their own Open mHealth compliant Data Storage Units (DSU). These DSUs are running with Amazon's AWS cloud service, which both is secure, reliable and easily manageable. Player reports can be divided into multiple DSU servers if the level of isolation or replication requires it to be. Players names and identification is not stored on DSU this is done by a separate unit.

Policy Server Unit (PSU) is a component in charge of setting and following policies separate from DSU. When creating a team inside the application, the team owner will centrally manage the data and own that data. The PSU will then be in charge of controlling who has access to this team, for how long and performs the aggregation functions needed for the data. Pseudonymous player data is shared in real time.

	5	4	3	2	1
Sleep Quality	Well rested	Good	Normal	Restless sleep	Insomnia
Fatigue	Very fresh	Good	Normal	Little tired	Very tired
Soreness	Not sore	Feeling good	Normal	Sore	Very sore
Mood	Very positive mood	Generally good mood	Less interested in others and/or activities than usual	Snappiness at teammates or close ones	Highly annoyed/ irritable/ down
Stress	Very relaxed	Relaxed	Normal	Feeling stressed	Highly stressed

Table 2.1: Self report parameters.

2.1.5 Parameters

Data collected by PMSys's mobile questionnaire solution is presented on a scale on either 1-5, 0-10 or 0-100. Some examples can be seen in Table 2.1. In newest version of PMSys's mobile app for iOS the wellness parameters is set up as the following questions and gradation:

- How ready are you to train? 0-10 (Readiness)
- How fatigued do you feel? 1-5 (Fatigue)
- How much did you sleep last night? Own choosing (Sleep duration)
- How well did you sleep? 1-5 (Sleep quality)
- How sore are your muscles? 1-5 (Soreness)
- How stressed are you? 1-5 (Stress)
- What is your mood? 1-5 (Mood)

Apart from wellness reports the newest version also has sessionRPE reporting. In this tab the player registers length and time of completion along with type of session (Competition/Individual session/Team session), with classification of strength/endurance/soccer or other. Session RPE is assigned to this session on a scale of 1-10, 1 being Very light activity and 10 being max effort.

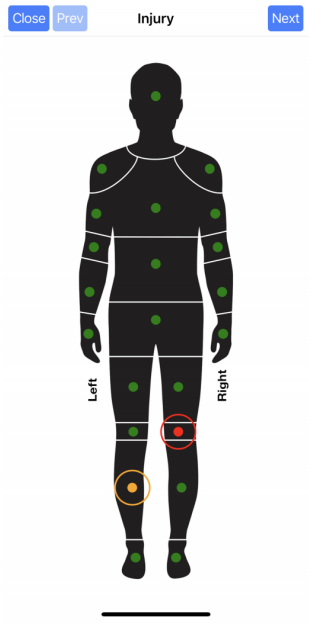

Figure 2.2: Body silhouette taken from mobile app

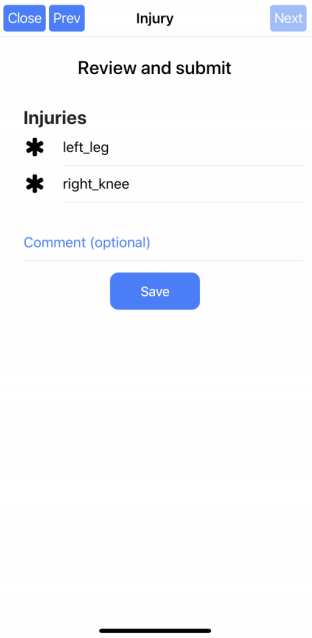

Figure 2.3: Injury summary taken from mobile app

2.1.6 The flow of PmSys

The flow of PmSys can be divided into different sections. The system is designed for athletes and coaches/team personnel to be able to communicate with each other in a more efficient way with data. Coaches will be able to fetch the data needed to assess the players. We can divide the system in to these sections:

Injury reports: Through a couple of iterations, this report method is made to resemble a figure as seen in figure 2.2 of the human body in where the body is divided into different parts/joints covering the whole body for accuracy. The process of injury reporting is minimized to about 12-15 seconds per report. This gives both a general overview for the coaches to keep track on and it works as a parameter for other functionalities like machine learning.

Coach/team personnel portal: An organized place for coaches and team personnel to see and manage relevant data from training and matches. Sorted in a way for the team to view the different parameters from the self-reported data given by the players themselves. Example being how much sleep individuals get, their overall wellness and readiness to play over a timeline specified in the interface.

Communication between coaches and players: Communication between coaches and players is important. Not only do the coaches have access to the stored and fixed data off the players parameters, they also can directly communicate to players in the app. If a particular player lacks parameters from that day, the coaches can notify the player to register their self-reports.

AI/Machine learning: Machine learning/AI is being used to predict peak performance and the future health and fitness of athletes. Using the data from self-reporting this model can predict the players readiness to train or if the player has a positive or negative readiness peak.

2.2 Technologies

In this section we will look at what make up the technology in this thesis. Everything technologies relevant for the operations and experimenting done will be described in the following sub sections.

2.2.1 Machine learning

Machine learning works by feeding a model data it needs in order to predict an outcome/result. Neural networks are what most think of when hearing machine learning. They are great for doing tasks which need a lot of computing and might not be as straight forward as typical mathematical problems might be. Problems like text to speech and speech to text are good examples of complex technical problems where machine learning shines. The data needed to solve these problems can be massive and the context needed for a program to solve it are difficult to do without machine learning. Machine learning stems from how our brains are made up and how they work. Since out brains are quite complex and intelligent there is much to get inspiration from and development of machine learning has been done for many years and is still being worked on.

2.2.2 Supervised learning

Supervised learning is usually done with a training set containing correct responses or the correct answers. Based on this training set the algorithm will learn to respond correctly on the given input and correct response. If we look at the example from Pål's research, this training set will be created

out of what the athletes sets as their readiness values. Inputs the algorithm uses is the self-reported data from the athletes, and based on previous collected data the algorithm will adapt the answers in order to get the most accurate results. The predictions from the algorithms are usually tested up against a test set which is unbiased, and from this we can craft an accuracy rating which will be an indicator on how good the model is. The input of a machine learning model/algorithm can be divided into many features. In the case of Pål's research they used the players self-reported "variables mood, stress, sleep quality, fatigue, and soreness" as inputs, and tried to predict the output value of the players readiness to train the next day. There are multiple ways of further modeling supervised learning, for example after looking at the data you can either pick classification or regression predictive modeling.

Classification: In a classification model you are to map a function (f) from features/inputs (x) in order to decide the most likely output values (y). The outputs are often called classes, labels or categories. Based on the previous observation in the values gives as features/inputs the model is to predict which label most likely fits. In the case of PMSys's readiness prediction you can divide the output into 10 classes. Each class deriving from the scale 0-10. The classification models will then use what data is provided, either it is just the univariate inputs of one self report parameter or multiple to predict which class at that particular date should be. Other examples of classification predictive modeling is predicting whether or not an email should be considered spam. This output is typically only split into "spam" or "not spam". If the email spam probability is set to be 0.9 the email would then be predicted as spam. The probability of a classification problem is the likelihood of the inputs belonging to a certain class. Only having two output classes is typically called a binary classification problem. Having more than two classes such as in PMSys or any voice or text recognition is called a multi-class classification problem.

Regression: Regression works with a similar mapping of a function (f) with inputs (x) to predict a variable (y), although in typical regression problems the output is continuous and outputs often represents a real-value for example being integer/floating point quantities such as size, prize or amounts. An example of regression based machine learning problem is predicting rain based on climatology related inputs such as temperature, sea level and other factors. A typical and often used example of regression prediction is linear regression. Linear regression shows the linear correlation between inputs and is good for simple regression problems.

2.2.3 Neural networks

The typical neural network is made up of a couple of layers. The first layer is the input layer, second is one or more hidden layers and the last layer is the output layer. The research gone in to this network is immense and there are many different methods and algorithms to chose from when deciding how the network should work, some might be better than others depending on what the network is designed for. If we start with the input, this can be represented in many different ways, vectors being one of the most used. The length of the vector is depending on how many features the inputs have. One feature might for example be in our case, how much sleep one athlete got that night. This input is apart of what makes the input layer. Weights will be what decides the importance of any given variable. The weights assigned to each input is then multiplied and added up. This output is then sent to an activation function, deciding whether the data should be transferred to the next node for further use. The activation functions uses a threshold in order to decide this, and the threshold can be modified for better results. Passing of data on to the next node is what makes this neural network a feedforward network [3]. This means data flows in one direction, from input to output. One other way to form a neural network is by using backpropagation. This is where you do the opposite, going from output to input. Backpropagation gives a detailed insight into how changing parts like the weight in the model might change the overall behaviour of the network. [4]

2.2.4 Time Series

Since PMSys's data both have a value and date, the data sets created can be considered a time series. Time series data can be a variety of data sets, daily closing values of stocks, ocean tides, customer count, it can all be categorized as time series if indexed in a time order. Therefor different kinds of analysis, algorithms, regression and classification that can be used to solve or answer any problem related to the time series. Time series forecasting is used to predict future values. In forecasting different methods and models are used depending of which type of data the problem consists of. Using previous observed values and other existing values related to the problem with seasonality as seen in figure 2.4 can be helpful for creating a accurate model. In the dataset gathered by self reports from PMSys there are multiple values and dates. Reports from 2016 to 2019 is all gathered in the dataset used in this thesis.

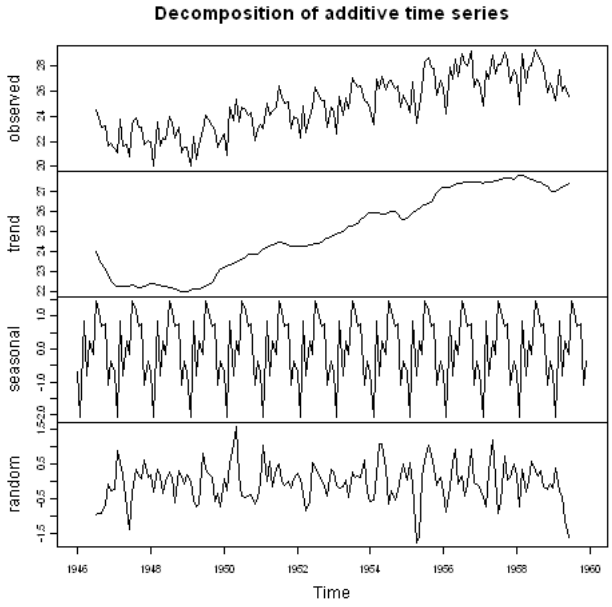

Figure 2.4: Example of typical time series graph showing both seasonal shifts and predictable trends [2]

2.2.5 Data analysis

Athletes participation lead to great data sets. Some of the data sets unfortunately contains some holes which can be expected with data being collected in such a big timeline. Some days the athletes might forget or simply not have the ability to log the required parameters. Because of this, testing of how the data sets should be prepared is important. With data missing there are some steps to take in order for the models to more accurately predict, either deleting NaNs completely from the data, replacing them with 0 or an average of the past observations or use a function to determine what values would fit in these holes. Where data is missing for the day, a NaN(Not a number) constant will be put in to maintain the timeline, this constant don't have any value. Deleting NaNs completely will lead to dates missing when feeding it to ML models which if not impacting the model can confuse results.

Feature engineering

The goal of feature engineering is to create features from variables that is the most favorable to the model and will give the best results considering the problem in mind. In this project we have a many parameters from self reports, some parameters hold greater value for readiness prediction than

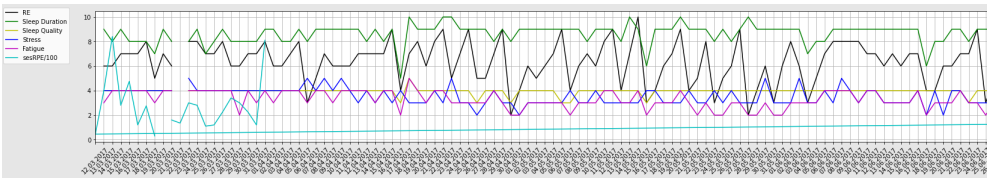

Figure 2.5: Graph showing features (Readiness, Sleep Duration, Sleep Quality, Stress, Fatigue and Session RPE divided by 100). Useful graph for understanding the relation between all features on a numerical level

others. One could use every parameter and bet that is the best features for the model, and one could also test the dataset for which have the closest relation. Figure 2.5 shows a graph over every potential feature from data collection. The relation between every parameter is hard to state but there are some parameters who are more closely following each other and showing signs of seasonality which could be in good use as features in a multivariate time series prediction. To test the relation between the possible features we looked at all of them in a single graph as seen in figure 3.2. Some parameters from the self reports make sense compared next to each other, for instance sleep duration (measured in hours) and session RPE (performance based value self given from a workout that day). In this graph i divided session RPE by 100 in order for it to more closely fit in the graph on the scale of 0-10. Sleep duration and session RPE might relate on a logical/bodily sense, a harder workout would make the body more tired and should theoretically result in a longer sleep duration. In the graph session RPE can be seen peaking one day with sleep duration dipping at the same time. This relation might not be as persistent as one would want it to be in order for it to be considered in the model as a feature. On the other hand, fatigue, readiness and sleep duration more closely relate to each other. Dips and peaks in these three parameters consistently happen at the same time or within a day in the future which logically makes sense as well.

2.2.6 Model training

In order to find the best way for training a model, we need to define what is a good model. There are many values to a model we can use to determine the overall score of a model. We need to consider both the results of testing, prediction and the behaviour of the model. Efficiency of models is also something to keep in mind, since you would want a model to be training fast and not take up too much of the computer's power. Every unit, node and layer of the model affects both time and efficiency. Having multiple layers with multiple nodes might make the

model more accurate but will have a significant loss of efficiency, so we will be keeping this in mind when designing models. When training a model we need to assess how the model trains through time/epochs. The models chosen in this thesis are pre-made when it comes to training them and predicting with them, but we still need a way of checking how well the models perform.

Loss

Loss in machine learning is an indicator for how good/bad a models' predictions was on a particular moment. The further prediction is from the actual values, the higher loss will be. Having a loss of zero means the model's prediction is perfect. When training a model having a low loss will be one of the goals. There are different functions for determining loss, mean square error(MSE) being one of the most used. In this thesis we are using MSE for checking loss both in training and when predicting on unseen and unbiased test sets. MSE works by taking the difference in the models' predictions and the actual values, square it and average it out for every step in question. The formula for MSE is defined as: $\frac{1}{N}\sum_{i=1}^{D}(x_i - y_i)^2$ MSE will be used for calculating the loss of the epoch you are at when training and machine learning libraries often project it to the user for debugging and making the models easier to improve. In addition to this, we will be using what is called MSPE (Mean Square Prediction Error). MSPE is just a regular MSE done on an unbiased and unseen test set or whatever is to be predicted after the model is trained.

Univariate time series

Using only one feature for training is commonly called univariate time series. It is defined as using only one observation over a time period for training and prediction.

Multivariate time series

Using multiple features for training is commonly called multivariate time series. It is defined by using more than one time-dependent variable as in our instance more than one self report parameter/feature.

2.2.7 TSAI library

Tsai is a Deep Learning library actively being developed by timeseriesAI. In the documentation it is stated it is a open-source deep learning package built on top of a couple of other Deep Learning libraries, Pytorch and fastai. Description read from its own documentation page "State-of-the-art Deep Learning library for Time Series and Sequences" [17]. Implementing models and neural networks for time series prediction is both complicated and time consuming, having an easy to use and efficient library to work with goes a long way. Tsai's functions and options provides a good deal of different specification, variables and parameters to test. Along with all the visuals created when training and testing and the models and training being efficient, all of this result in a great library for this thesis. Example of ready-made models in this library relevant for this thesis:

- ResNet
- ResCNN
- FCN
- InceptionTime
- Xceptiontime
- LSTM_FCN
- LSTM
- LSTM Bidirectional

Analysis of training, models, features and prediction in Tsai is a plus point for this library. Testing some of these features provided by Tsai gave good insights in which of the features and inputs comes in a favor or not. For example, the library provides a feature importance tool which is seen in figure 2.6. This tool uses permutation importance to help the user better understand what the features does for accuracy or other metrics like mean squared error in the model.

2.2.8 Keras

From keras.io, keras's own webpage[10] it claims to be the most used deep learning framework among the top 5 winning teams on Kaggle. It is a solid framework for machine learning and deep learning. Used by both

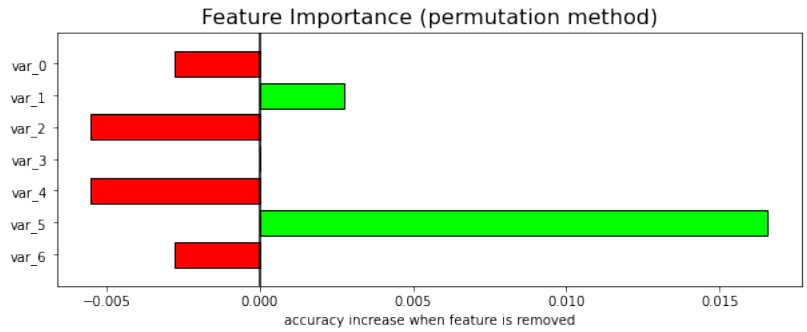

Figure 2.6: What the features in an array of 7 does to the accuracy of the model. Features seen as var 0-6 shown with their represented accuracy change of removed

CERN and NASA it has to be reliable and industry strong, meaning it is holding its weight in scaling and efficiency. Built on top of TensorFlow 2, a well known open source platform for machine learning. Keras focuses its neural networks on what they call layers, their building blocks. A model can contain many different layers each providing a little tweak in structure making up divers use cases. Since Keras is made with the intention of being highly scalable and easily deployable, many companies uses it for this purpose. With it being made for a borderline commercial but also research use in mind it provides a good debug-able environment. Utilities for debugging and visualising connections, layers and operations makes it great for people just diving in to deep learning, and a good tool for this thesis' experimentation.

2.2.9 Models used

Below we will give a short insight in how the different models used in this thesis operate.

ResNet

ResNet short for Residual neural network is an artificial neural network derived from the first very deep feedforward neural network Highway Network. Highway network is known for being the first neural network that could handle hundreds of layers and going much deeper than the other neural networks at that time. Working with a gated way to organise and operate on information flow this network saw good results when it came to preventing the vanishing gradient problem.[9] Where ResNet is

different from HighwayNet is being gateless. Being gateless this provides different options across layers, for ResNet networks if there is need for it, information might skip a couple of layers. This is a good way for preventing vanishing gradients and increases the speed of training. For more information about ResNet the paper can be found here.[7]

ResCNN

From Tsai's documentation there is not much to find about ResCNN. After some research we found a paper by Sarosij Bose where they compared CNN to the newly introduced ResilientCNN also called ResCNN.[1] From the little information we could gather about ResCNN, it uses image processing techniques like Singular Value Decomposition(SVD) instead of the typical convolution we can find in CNNs. Singular value decomposition is factorization of complex matrices which this network is based off, convolution implemented as a matrix matrix operation. Some qualities of this model is learning with both bigger batch sizes and bigger learning rates without sacrificing accuracy.

FCN

Fully Convolutional Networks(FCN) are typically used in semantic segmentation but can be used for other problems as well. In FCN's they typically do not use any Dense layers which makes them have less parameters. Dense layers are deeply connected layers taking many inputs and operating on them before sending the information elsewhere.[15] The dense layer often consists of many neurons which all receive input from the neurons from the preceding layer, making up a ton of parameters. In FCN's there are no Dense layers, so they only use locally connected layers, like convolution or upscaling as seen with an example of a picture in figure 2.7. This in turn makes the model faster to train because of the less parameters throughout all layers.

Inceptiontime

Based on both convolutional neural networks and Resnet, Inception-V4 was made to achieve good performance at a low computational cost. InceptionTime is a variation of Inception-V4 with multiple Inceptions built in to the network. InceptionTime takes five Inception modules with the intention of applying multiple filter at the same time to an input.[5] The

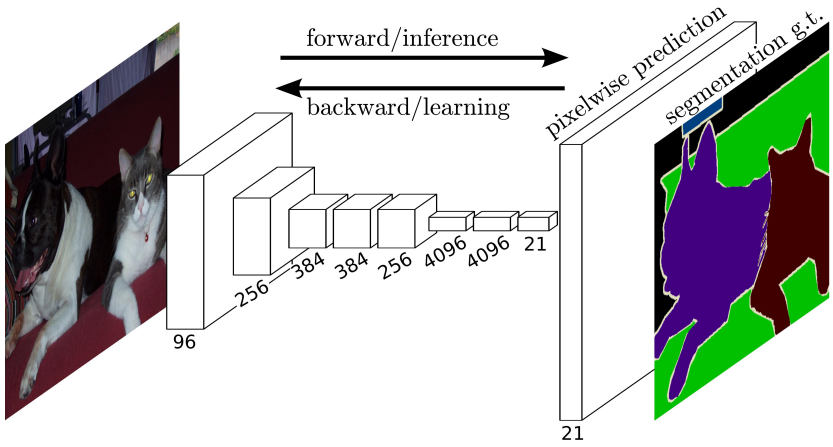

Figure 2.7: Figure shows an example of semantic segmentation where Fully Convolutional Networks can learn by upsampling the output size when wanted. Figure source: [15]

power of InceptionTime comes in the ability to extract data from both long and short time series. The flow of one Inception out of the five used in InceptionTime can be seen in figure 2.8

XceptionTime

XceptionTime is heavily inspired by InceptionTime, having a quick glance over the main parts of the network it is hard to spot any difference. For XceptionTime it has been tested better performance results on larger image classification tasks than with Inception V3 even though they are so similar in structure. The key part in what separates XceptionTime versus InceptionTime is the Convolution made. XceptionTime is implemented with depth-wise separable convolutions in the layer between the bottleneck and output. This in turn will mitigate the number of required parameters for the network.[14]

RNN

Neural networks are made to reflect the behavior of the human brain. Simply put, it is made up of nodes with weights in between them. The weights are used to change and decide which of the nodes are better used for the solution in question. Neural networks are great at recognizing patterns and solve common AI and machine learning problems. Traditional neural networks don't usually have persistence like our brains does, the human brain don't throw away every memory

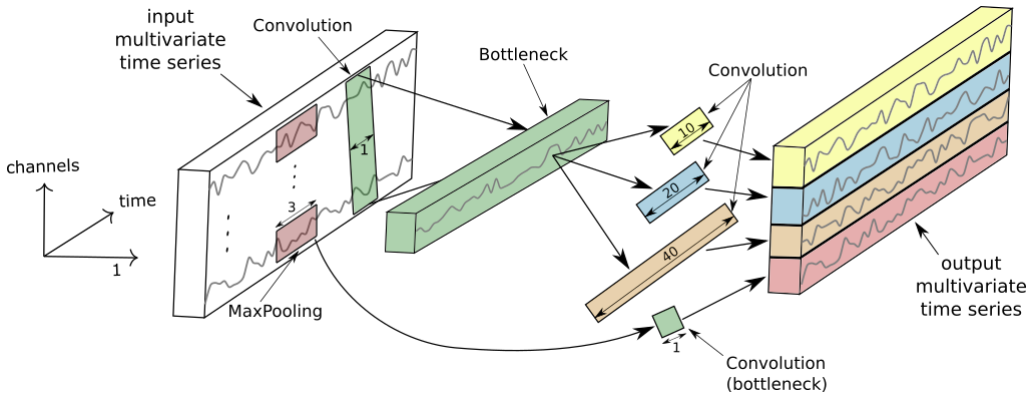

Figure 2.8: Figure shows an example of a multivariate time series problem and how the inside of Inception treats it, one of the modules in InceptionTime.[5] The steps of Inception goes as follows: First the input data is sent to a bottleneck which in this example is a bottleneck with the dimension of 1. The bottleneck transforms the time series to a multivariate time series with M dimensions, 1 in this case. This reduces the complexity and dimension of the model and in turn will give some prevention for overfitting. After this the output of the bottleneck is filtered with three convolutions in this example and concatenated with a parallel MaxPooling operation from the same time series which has gone through a bottleneck as well.

for each thought. Recurrent neural networks addresses this, looping in this network allows information to persist. Each chunk in the network allows information to be passed along to the next[3]. Uses of this type of neural network can be seen many places, language modeling, image captioning, speech recognition and Pål and colleagues work with performance prediction. Where some networks use just a normal backpropagation for determining the gradients, RNN uses backpropagation though time(BPTT). The difference here lies in when the errors for adjusting weights are calculated. In normal backpropagation it is usually done from output layer to the input layer, but in RNN errors are summed at each time step through training. One downside to doing it this way is the probability of running into vanishing gradients which essentially makes the model stop learning at a certain point if the gradient is too small.

LSTM

LSTM is based upon a concept of RNN, but has added some new technology to the common RNN network. LSTM has "cells" in the hidden layer with three different gates functions, input, output and forget [4]. With

these gates this network can more accurately decide the results based on context stated before. The cells can communicate prior context and information to the next cell and the cells can also decide what to forget. Information on the cell is sent to a sort of conveyor belt where if any information should be changed before outputting it, it can be done with these conveyor belts. For PMSys, patterns from the self-reported data can be found and used to predict if an injury could possible be in the horizon and which player is ready for a match, this can all be done in a nice and agile way with LSTM.

A single LSTM cell includes a couple components as shown in figure 2.9. In this figure red circles indicate operands being done from left to right depending on the decisions of the model. The yellow squares represent the functions Sigmoid and Tahn being used as activation functions. Each cell takes inn two inputs, one cell state and general input, it gives a output and sends the input and a cell state further to the next cell. For simplicity i will call the top running line both working as input and output trough the cells, the cell state. The two different activation functions both got their own role in this structure. Sigmoid functions will output from 0 to 1, making it known as the gatekeeper for a forget gate. For instance, if the value equals 0 in the first Sigmoid operation the input will be forgotten, and otherwise if the output is equal to 1, the input will be considered in future operations. Tanh outputs from -1 to 1, this is crucial to overcome the vanishing gradient problem where tanh will sustain for a longer range and not giving the model such a close value to 0. Tanh and Relu are both valuable options and the debate of which is best for LSTM is still active. A combination of Tanh and Sigmoid is used in what can be called the input gate and output gate which. Input gate being right after the forget gate multiplication/Sigmoid operation. Taking in the raw input from the cell t-1 running it through a Sigmoid actiavtion and multiplying it to the result of a Tanh activation. This is then added to what came from the forget gate. The output gate will take the raw input from cell t-1 running it with a Sigmoid activation and multiplying it with what comes from taking a Tanh activation on the cell state pipeline.

Sigmoid activation:

$$h_\theta(x) = \frac{1}{1 + e^{-}\theta^T x}$$

Tanh activation:

$$\tanh\left(\frac{x - g}{h}\right)$$

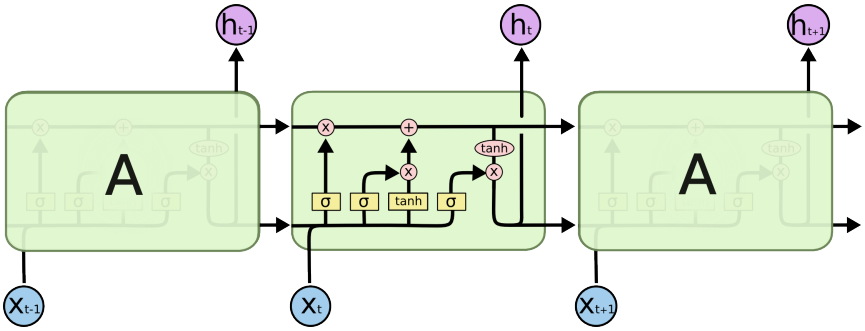

Figure 2.9: The structure of LSTM cells Figure source: [16]

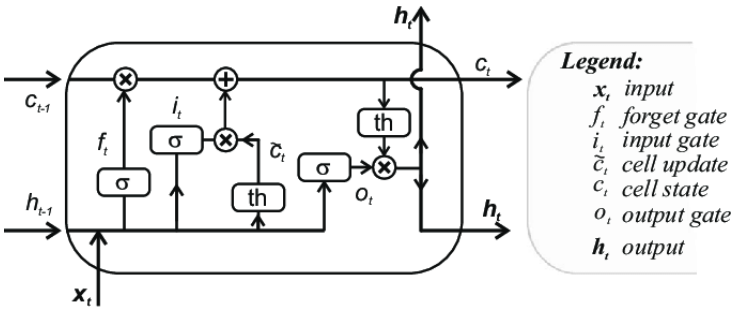

Figure 2.10: The layers inside an LSTM cell, figure source: [8]

GRU

Similar to LSTM, GRU is a runner up for solving RNN's problems. Where RNN under performs in short-term memory, GRU along with LSTM is made to address these problems. Made of one gate less than LSTM, GRU goes around the cell state from LSTM with only having an update gate and a reset gate. For the update gate, it decides what information to keep or ignore/throw away. The reset gate works in the same way on another level it assesses what part of the past information passed in it wants to forget for further computation. Since GRU has one less gate, the network will train somewhat faster than LSTM. Which of LSTM and GRU is the definite best of the two is up to the problem being solved, both LSTM and GRU can work great depending on the problem.

2.3 Related studies

There are some studies done which related to some degree to what we are researching. In this section we will introduce some of the problems relevant to this thesis which earlier has been addressed. These studies has

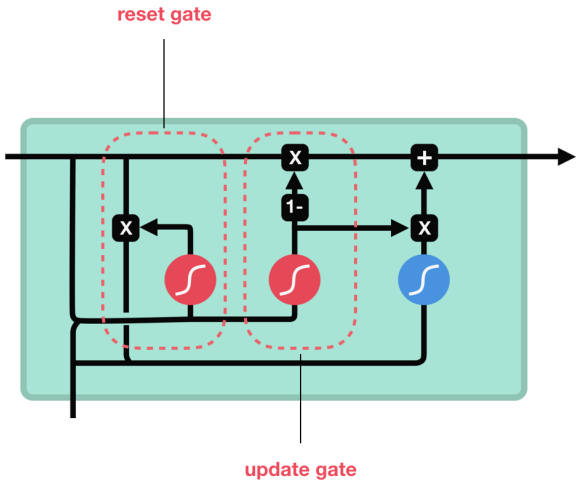

Figure 2.11: Figure shows one single GRU cell. Red circle indicating sigmoid functions and blue tanh. figure source: [13]

influenced what was decided to experimented on in this thesis.

2.3.1 LSTM for peak readiness

With the purpose of finding out if machine-learning methods could be applied and predict the future health and fitness of athletes Pål set up an experiment[20]. Pål used LSTM to train their model. The model took in players reported readiness values and outputted a score being the predicted readiness value. This all operated on a day to day basis, using one day's values to predict the next with the possibility of adding more data in the future which might help predicting more accurate readiness values. For training and validation, two different methods were used: "First, training on all other players on the team, then predicting the readiness of the chosen player. Second, use most of a player's data to train and then predict on the rest." [20]. The first method had a scarce amount of data to train and predict on since the model only trained and predicted on the same person, and only having 100 to 200 time steps to train on. The method performed quite good with respect to the limitations it had. The accuracy could have been much better, but it was mostly clear where the general peaks would have been. Training using the dataset of all players except one and predicting on that one player saw the best results out of the two methods. One drawback with this method could be that the other players might have different behaviors in their patterns, but the trade for this is that this method gets more data than the other method. The second method had some inaccuracy but overall it could

more accurately predict a clear peak and much earlier in the experiment than the other method. With the two methods having some inaccuracy possibly caused by the lack of training data, Pål extended their experiment to detect positive and negative peaks. This will come in handy for fitting a workout plan or knowing before a match which player is most ready and has the most potential for optimal performance. From observing the data outputted from the model, it is clear that the model could distinguish high and low values quite well. This made it possible to define a positive peak to values in readiness from 8 and up to 10. One small issue with the model was distinguishing higher values from each other. If a player had an higher average of values the model could distinguish the values better than with players with lower average readiness to train values. Negative peaks are a good indicator for finding out which players should not train or play. With a readiness value under 3 the players are classified as having a negative peak in readiness. In this evaluation the model could more easily distinguish between the negative peaks. Overall the experiment got good results on predicting peaks in readiness.

2.3.2 Case study on self-reporting symptoms on triathletes

Three case studies conducted by J. R. Grove and colleagues looked at triathletes performance stress measured with a training scale called Training Distress Scale (TDS).[6] This scale includes 19 items of measurement related to distress and performance readiness. Two of the case studies performed different interval training to validate TDS score over time. The third group were a group of swimmers examined in relation to TDS over a 2 week period before a swim competition. Training distress measured related to TDS include various symptoms in relation to "emotionality, general fatigue, concentration difficulties, physical discomfort, sleep disturbance, and appetite changes." [6]. Some examples from this includes "lack of energy", "muscle soreness" and "loss of appetite". Input from these symptoms where collected on a 0-5 scale on to what extent the participants experienced the symptoms the last two days. Both the first and second case study found that TDS demonstrated a link between performance and the self-reported data. All three cases concluded that using TDS as a measurement of training distress could successfully indicate peak readiness to perform. A high score in TDS related to a decrease in performance. All though, there are limitations to the research both in the limited sample size and variability of self-reporting with participants' possibility of not assessing the same scores compared to each other, the results are still promising.

2.4 Summary

This chapter gave an quick introduction to the background knowledge needed for this thesis. In section 2.1, we took a look at PMSys and how the platform is made for monitoring and improving athletes and is a good link between staff and athletes in an organization. Furthermore we described how the system uses self reporting data from the athletes and portrays it to a portal for coaches, team personnel and athletes. We went through the steps for the athletes when reporting their data and looked at the functionality of the system both with the mobile app and coach portal. We described every parameter the athletes could enter and send as reports to the system and how they are graded on a scale. At the end of this section we introduced how machine learning is used for readiness prediction and how it will be used in the system.
In section 2.2 we introduced the technologies used in this thesis. This section started with a description of how machine learning works and typical problems related to machine learning. From machine learning we moved on to a short introduction of supervised learning and how the data is created so that the machine learning models use it optimally. This subsection described the difference between classification and regression and how showed examples of how the two differ from each other. From this we moved on to neural networks, and how it uses layers and weights along with the supervised dataset. We also discussed the use of backpropagation and how it can improve the behaviour of models. Since PMSys stores their data along with dates, it can be considered time series. Here we looked at different examples of time series and how PMSys uses data from 2016 to 2019. With all this data stored it needs to be analysed and processed. Here, we discussed how the data is analysed with regards to missing values(NaNs) and how the dataset is split up in features. Feature engineering is a big part of making a model perform as well as we want it, so we discussed how the features gives the model potential. We discussed how features relate to one another and selecting the right ones for the problem in question is key for performance. Next we moved on to how model training works and how defining what you want out of a model is important. Along with this we described the way we measure the performance of a model with a mean square error(MSE) on the unbiased and unseen test set making it mean square prediction error(MSPE). Before we headed in to defining the two main machine learning libraries used in this thesis we briefly described the difference between univariate time series and multivariate time series. The two main machine learning libraries used in this thesis is Tsai and Keras. To finish this section we looked at what they are, where they are used and what they are good for.
Section 2.3 showed and explained the relevant models from machine

learning libraries explained just over. The models introduced where:

- ResNet
- ResCNN
- FCN
- InceptionTime
- XceptionTime
- RNN
- LSTM
- GRU

All of these models explain the main theory behind every model used in this thesis, since the rest of the models build upon the logic of these 8 models explained here. In this section we saw what defined the models, what makes them differ from each other and what typically makes them excel over each other.
For the last section 2.4 we took a look at prior work done on the relevant technology and system. We saw how LSTM had been used earlier by Pål and colleagues to predict the readiness values of players. In this case we explained how the models had a hard time distinguishing higher values from each other, how a the models did not always predict the actual highest value only catching the upwards trend most of the time. After this, we introduces a case done by J. R. Grove on how effective RPE(Rate of Percieved Exertion) self reporting had been seen in triathletes. This case brought to life the link between actual performance and what the athletes self reported on the TDS(Training Distress Scale).

Self-reporting is known by now to be a viable way for athletes to keep track of their health and is a good measurement used for team improvement which is seen in both Pål and Grove, J.R[20] [6]. Collecting this into a well developed system running on smartphones makes it even more efficient and opens up possibilities for even more functionality and research. For coaches and team personnel to be able to analyze and further work with this self-reported data improves the old way of keeping track of the athletes. Communicating and notifying athletes in the app also makes this much more efficient. Machine learning fitted around the relevant data for peak readiness has seen use in sports analytics. Neural networks with a long short term memory found the best use in predicting readiness in athletes with self-reporting as input. The prediction can be

helpful in both fitting a training plan, and finding which players are most ready for a match. Finding parameters and functionality which will give a better prediction in performance readiness can be done on the basis which already is available by Pål's work abd PMSys' data collection.

Chapter 3

Methodology

In this chapter, we will look at the where our experiments are run and what choices we made for the configuration of both the models and data used. We will look at the baseline of everything namely the system specifications of the computer used for running the experiments. Along with this we will look at different libraries used as the building blocks of how we are able to compute and run everything. Tsai and Keras are two of the main libraries used for the machine learning part of this thesis with some extra tools being used in order to make these main libraries work seamlessly. In addition to this, we will discuss the choices made inside these libraries when it comes to models, parameters for the models and functionality these models provides. The data fed to these models are also influential, so there are some decisions going into what separates the different datasets that we will look at as well.

3.1 System specification

Table 8.1 is a list of each component along with their respective version under the software tab. In the hardware tab you will see the most important hardware components along with its specifications. Everything in this list is ran with Windows on top. There is no specific reasoning for this other than it was the easiest to configure since it was already the running operating system on the computer used for testing. Running each the component on Windows did not come with any advantage or disadvantage, at least none experienced in this thesis. Python was used as the primary and only programming language used for implementation. Python is often used when it comes to implementation and running

Type	Name	Version	Description
Software	Windows Pro	10.0.19044	OS
	Python	3.8.5	Programming language
	Tsai	0.3.0 + extra	Machine Learning library
	Fastai	2.5.3	Machine Learning library
	Fastcore	1.3.27	Python extension
	Torch	1.10.0+cpu	Machine Learning library
	Pandas	1.3.4	Data analysis library
Hardware	Memory	16 GB	-
	Drive	Kingston 1TB M.2 SSD 2200 MBps / 2000 MBps	-
	CPU	AMD Ryzen 5 3600, 6-core	-

Table 3.1: System specifications.

of machine learning. it is easy to use with lots of compatibility for the libraries to be used. In table 8.1 the libraries used are only the core and base libraries needed in order to experiment with Keras or Tsai. To make implementation easier a couple of time-saver libraries were used. Libraries such as sklearn.metrics for MSE and confusion matrix, IPython.display for clearing output are some examples of the types of libraries not mentioned in the table. 'Keras' and 'Tsai' are the main libraries used for machine learning both being compatible with the components used and fairly easy to set up. For handling the data used fed to Keras and Tsai we used pandas, a data analysis library for python. This library made the typical time-consuming job of arranging and configuring the time series data much faster and easier.

3.2 Dataset

Since we aim to predict readiness to play, the shape of the different datasets plays a big role for optimal training. The data gotten from

PMSys' self reporting system unfortunatly got some days where players did not report anything to the system. These values needs to be addressed in order for any prediction to be done. Finding the optimal way of handling these missing data points(NaN) could make a difference for both training and prediction. We decided to replace the missing datapoints with three different approaches; filling NaNs with zeros, a mean value and an interpolate value. From this, there will be 9 different configurations of datasets used in experimentation; two multivariate and one univariate. Each of these will be used to test the three different NaN handling methods.

3.2.1 NaN handling filling with zero values

Replacing NaN values with 0 is not that difficult of a process. With panda's built-in function for filling NaN we were able to simply state what value we want to use for replacing and do it all in one go. Compared to the other methods for filling, this one should not need much explanation. For every point where there are missing data this function replaces it with the value 0.

3.2.2 NaN handling filling with mean values

Filling NaNs with mean values is a typical statistical way of imputing. What we believe will be the benefit from filling with mean values is that compared to fillig with zero values, mean values should be closer to what the athlete actual was going to report that day. This all relies on that the athletes are consistent in the way their body feels, which might not be the case every time. When filling NaNs with the mean value we first calculate the mean value of the whole dataset. This fairly simple, and after that we find every NaN in order for it to be replaced with this value we just computed.

3.2.3 NaN handling filling with interpolate values

Interpolation or more specifically linear interpolation as we will use, is a simple method for estimating any missing or unknown values/value of a function between known values. The interpolated point/target value is a point between two known values. The calculation of this point uses the known values from both side, take a look at figure 3.1 for visuals. In this

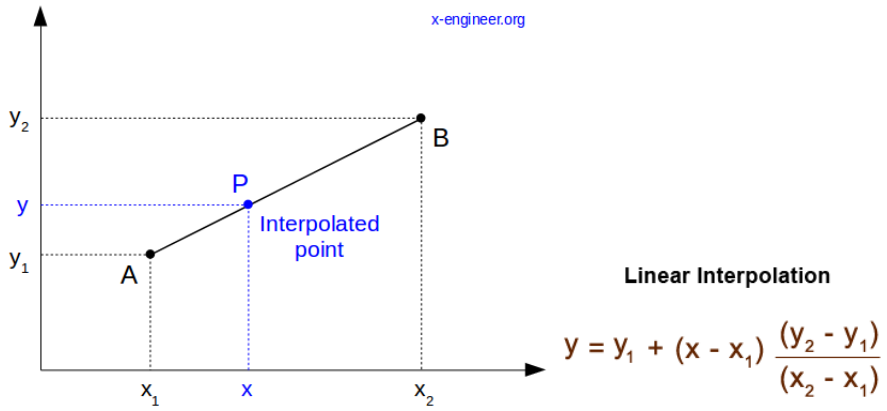

Figure 3.1: Figure shows what is estimated in linear interpolation (Interpolated point P) and the formula for this calculation, figure sources [11] [19]

example we take the biggest value of y from point B and subtract it with the y-value of point A. This is then divided with the x-value of B minus the x-value of A. All of this is then multiplied with (x-x_1) and added with the y-value of point A. Since interpolation requires two known values there are cases where some NaNs might not have two known values in front of behind itself. In these cases we decided to fill the rest of NaNs with zeros. Typically this would happen at the start or at the end of a dataset, and all the machine learning models used do not support NaN as data for training.

3.2.4 Univariate

For univariate training we will only use one features which is the same feature as we are to predict; Readiness to play. As we discussed earlier in this chapter, univariate training will be further divided into 3 datasets. These consists of NaNs filed with zeros, mean value and interpolate values. Training will happen on one of the three chosen datasets at a time, we will not use all three sets at once this will be done in the multivariate part of experimentation.

3.2.5 Multivariate

For multivariate datasets we have two different configurations for sets:

- One for training on every player's readiness level(30 in total) and

	Interpolate	Zero	Mean
0	0	0	6
1	4	4	4
2	4	0	6
3	4	0	6
4	4	0	6
..	...	...	...
912	6	6	6
913	6	0	6
914	7	0	6
915	8	8	8
916	8	0	6

Figure 3.2: Difference of filling NaNs with zero, mean value and interpolate values. The index of this dataset is dates, but to get a visual on the dataset it is indexed in an observational order

predicting one particular player Readiness level

- One for training on Readiness, Sleep, Fatigue, Stress, Sleep quality and PE (Percieved Exertion) levels on one player and predicting Readiness on the same player

Figure 3.2 and 3.3 shows the two configurations of multivariate datasets. Figure 3.2 is part of the dataset for multivariate training with training on all player's Readiness data(figure showing the three different NaN imputes for one player) and figure 8.3 is multivariate training on a subset of one player's self report data. For experimentation on the multivariate dataset training on all players there are a lot of missing datapoints. After looking at the entire dataset we found that some players have upwards of 800 days where they did not record readiness values. This could turn into an issue for the models when training so we decided to cut out the players with the most NaNs in their section of the dataset. For players with more than 750 missing NaNs we will cut them out the training, validation and test set. After cutting these players we are left with 15 in total with less than 750 NaNs throughout the period of self reporting. We will still call it training on all players even though we cut players and technically it is not "all" players.

3.2.6 Timesteps and prediction clarification

Timesteps or sometimes called lag or window size is a method in machine learning where you shift time series data so that previous observations will affect the prediction of models. Sliding window is also the same thing which some people will call it. Browsing the web you can find some

	Readiness	Sleep	Fatigue	Stress	SleepQuality	PE
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
..	...	...	...	...	...	..
912	7	9	4	4	5	4
913	6	10	3	4	4	4
914	5	9	3	4	3	4
915	5	9	3	4	3	4
916	5	9	3	4	3	4

Figure 3.3: Figure shows multivariate dataset with Readiness, Sleep, Fatigue, Stress, SleepQuality and PE(Perceived Exertion). This particular set is also a set used for multivariate training on one player and predicting the Readiness level on that player.

diversity in what people call timesteps, we decided to define timesteps as the number of previous observations used in making a prediction. In addition to timesteps when making the datasets for timeseries you can define the strafe the window will take when setting up predictions. Having a strafe of 1 is the equivalent to taking the x (timesteps) amount of days prior to the day you want to predict and after predicting the next prediction will use two of the same values used in the prediction prior. Finding the right balance in timesteps is important if you want to see some good results. We decided to try a couple of different timesteps when experimenting to see what gave the best average score. Looking at how machine learning would work in the area PMSys designed it to be implemented in, there are some things to take into consideration. One might get the best results by having a certain amount of timesteps, but typically football teams have their training and match days in a structured week. They might have their games on saturday or sunday using weekdays as training days. In this scenario you might see some seasonality in the data which might give a timestep of 6-7 days an advantage over just one or two days. This premise is not based on the fact that some days are missing data, so we will be testing different timesteps also called window size. When it comes to predicting just one day or a sequence of days ahead the results may vary. For practical reasons a team might want to predict 6 or 7 days ahead of a game in order to better plan what they should have the players do that week. If they find a player who is predicted to not be as ready to play as they would like, they can now adjust this in order for the player to the most ready to play. This way of predicting is called multi-step forecasting or multi-step prediction.

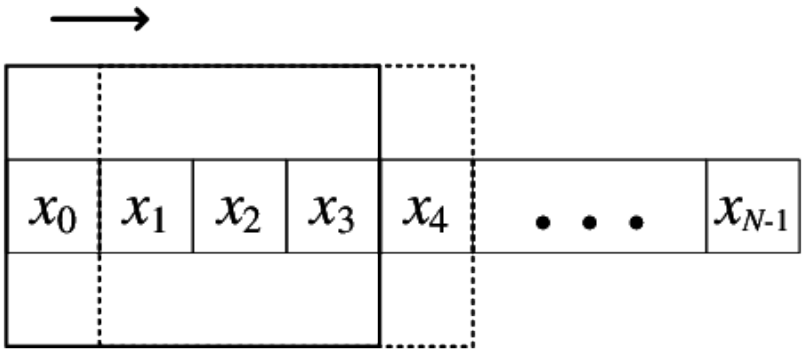

Figure 3.4: Figure shows how strafe and window length looks like on a series of N length. This particular figure has a window of 4 and trafe of 1. Feeding this to a configured model will use the 4 values from x_0 to x_3 for predicting the value of x_4. Next prediction will use the values x_1 to x_4 for prediction of the next value. These two prediction will use some of the same values since the window is of size 4 and strafe is only 1. There are many different ways of configuring these types of models, having a strafe of 1 being fairly common. Figure source: [18]

3.2.7 Multi-step forecasting

Multi-step prediction is predicting multiple days ahead of time. For example, you can train your models on 3 months worth of data and with multi-step prediction/forecasting you can predict the next 2 weeks of data. In the case of PMSys, multi-step prediction can be useful for teams that want to see how their players' readiness to play will be heading the next week. If they have pushed a week full of training and want to see how their readiness will be the next week, the models can predict day by day how the readiness might go. Something to be noted, the longer you want to predict into the future the less accurate the prediction will be. If we set the models to predict the next 10 days after self reported data stops, it will continue to top off days for prediction based on what is already predicted. Figure 3.5 illustrates how multi-step forecasting works. This example uses a sliding window length of 9 days and hops to the next day after one prediction. Once a prediction is made the first value is no longer used in prediction since the window length is only 9 so the newly predicted value will be used. This can go on as long as one wants, only downside is accuracy will decrease since the data used for prediction is not based on any facts/observations.

For our experiment with multi-step forecasting we will start by using a window length of 1 and a test set of 20 days. This means at the end of the test set there are no longer any observational data left for the model to use when predicting. We are going to see how accurate predictions will be and

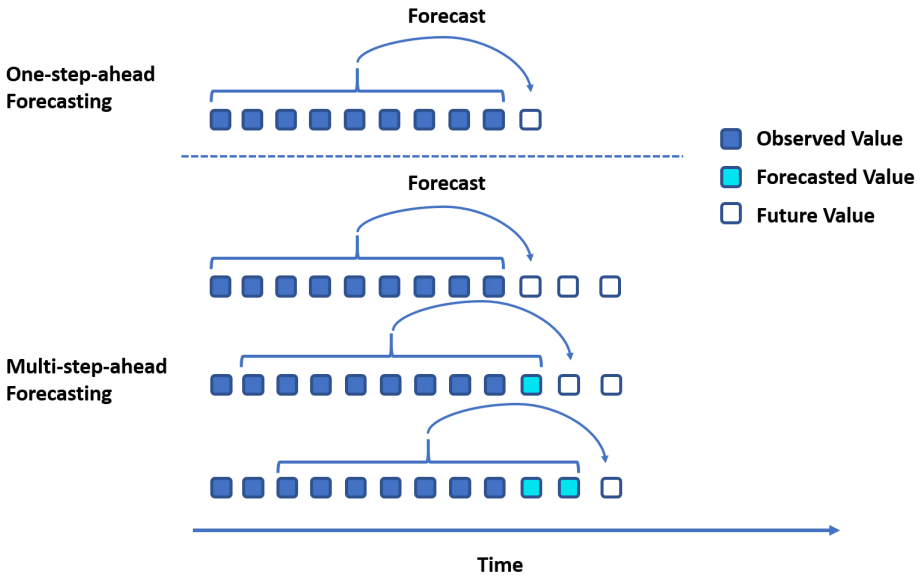

Figure 3.5: Figure illustrates multi-step prediction/forecasting. In this example the window size is 9 and the last white box is the day to be predicted. When that day is predicted the window shifts one day ahead and puts the predicted value in to this window. This means the model will use the predicted value to further predict what values comes next. Figure source: [12]

we will test what works best, replacing NaNs with interpolation, zero or mean values. Due to time constraints we will use Keras with univariate training for testing this. Tsai might have given better results but we did not find any functionality resembling this in their library and it seemed to be too complicated to try modify or implement this with Tsai. For Keras they did not bring any functionality for this but their predict method comes in handy, so we made a recursive multi-step forecasting solution ourselves. A fully working multi-step forecasting method might be coming soon to Keras so better solutions are probably out there in a moment of time.

For the experiments with multi-step forecasting we will test how well this works in Keras with the same three different models tested on before; LSTM, GRU and RNN. The models will be trained on the three different NaN handling datasets. When predicting we plan to use different amount of days as test set. The first prediction of these days will use actual observational data to predict and after this, we feed the model its own prediction for it to further predict.

Figure 3.6: Figure shows the split distribution of training, validation and test set

3.2.8 Data splits for training, testing and validation sets

Data splits are important for model training and having the perfect balance of training, testing and validation data will result in the best results. From PMSys' self-reporting system we got 915 days worth of data. Figure 3.6 shows the exact splits used in experimentation. We decided to go for a training size of about 80 percent, with validation set having 20 percent and test data will go on 10 days. For the test set we decided to go with the days not including zeros which was the case if we cut off the last 10 days.

- Training split: 714
- Validation split: 180
- Test split: 10

NaN handling as we discussed earlier in this chapter will be done on the entirety of the dataset, meaning it will not just be done on the training set but on validation as well. The test set will not be affected since we went for a test set without zeros in it.

3.3 Keras and Tsai Models

In order to compare the results between Keras and Tsai, we decided to try and find the best performing models from a subset of models chosen for both Keras and Tsai. Since Keras do not use the same pre-built models as Tsai uses, we were limited to making our own models out of what they provide with their layers architecture. The layers in each Keras model are created the same with only the defining part of the model (LSTM/GRU/RNN) being what is changed between them. We landed on 3 different RNN based models:

- LSTM (Long Short Term Memory)

- GRU (Gated Recurrent Unit)
- SimpleRNN (A simple version of a Recurrent Neural Network)

For the models chosen for Tsai we came up with the list of:

- FCN
- InceptionTime
- LSTM
- LSTM_FCN
- XceptionTime
- ResNet
- XResNet
- ResCNN

Chapter 4

Experiments and results

This chapter will contain an overview of the experiments done, how they were done and what the results looked like. Experimentation went through different stages, the start focused on testing and learning how the different libraries worked and behaved on our data. From experiment to experiment we use the knowledge learned along the way in order to find the optimal results. We will be testing what models works the best, and for what NaN handling method they combine best with, and for what training method they all work the best with. In addition to this, we will test to see if how the models and configurations we find the best will do on predicting readiness for players both a day ahead and a couple of days in the future. To finish this chapter we will present and discuss the overall results obtained.

4.1 Experiments overview

After learning and understanding how both keras and Tsai implements we decided to find the models with the best results for further experimentation. From Tsai we chose 14 models we found in Tsai's documentation and for Keras we 3 models who were easy to set up. Experiments are also made to compare what NaN handling methods work the best for these models. We have chosen three different NaN handling methods; Replacing NaNs with zeros, replacing with the mean value of the entire dataset and lastly, replacing with an interpolated value. On top of all this we will try to check what window size works the best when predicting. From prior work done with PMSys we chose to check with two sliding window sizes; 1 day window and 3 day window. Testing will be done on the same test set through all experiments in the beginning. This test set is a set off 10 days with no

NaNs.

The following experiments will use the same set of self report data for the whole of experimentation. Prediction will always be done on readiness to play and on the same player for every experiment, both for multivariate and univariate training methods. Multivariate models will use two different approaches. One of the approaches is training on multiple players in the team, and predicting on the values of a single player withing the same dataset. This approach will only use the parameter Readiness for training and prediction and the features for all models will be the different readiness to play taken from the dataset in question. The other approach is training with multiple features from the self reporting data on the same player as is being predicted on. The features for this training method will be: Readiness, fatigue, stress, sleep duration, sleep quality and perceived exertion. Univariate models will train on one player with one feature, Readiness.

When determining how well models performed we will look at the MSPE (Mean Square Prediction Error) of the models. This score is calculated based on an unbiased test set of 10 days going for the same set through all experiments. Training loss and validation loss from training will also be assessed in the initial part of experimentation in order to determine how to adjust parameters for models used in further experimentation. Graphs from prediction on test sets will also be used to determine how the models behave on the different datasets. Everything we learn from each step of experimentation will be used in the next step if possible, changes in parameters might not work universally on all models or datasets created, so some hyperparameters might be different inside each experimentation.

After initial testing is done, we intend of using the information we got on a different set of data. This time we will be testing different ways of handling NaNs and seeing what needs to be changed from initial experimentation on to this intermediate experimentation. In addition to this, we will be experimenting with different window sizes of 1 day or 3 days. For this middle part of experimentation we will compare all the models, the results they got and how they behaived when predicting on the test set with different datasets for training. After we have determined what worked the best, we will continue to a final experiment. This experiment is about predicting x number of days into the future. We will use Keras for this experimentation and check whether the models can hold up the same results in this last experiment compared to the intermediate experiments. We will use different window sizes and predict different amount of days into the future to check what works best and how well the models can predict into the future.

4.2 Initial experiments

4.2.1 Overview

The initial experimentation was done to establish what capacity and baseline results each model had in order for the models to improve and be compared later on. For the initial experimentation we opted for handling NaNs by filling them with zeros, for all sets including training and validation set. The beginning thoughts for this thesis was to compare results from Keras and Tsai' LSTM models. After some research and discussion we opted for testing what Tsai has to offer over Keras which is their many premade models and utilities easily usable. For initial experimentation we take a look at what Keras and Tsai has to offer and how they compare when it comes to results. Along with this we will take a look at which of the 3 training methods we use generate the best results. We will also track what hyperparameters work for the different setups, something we will need for further experimentation down the line.

Tsai

Tsai provides a tutorial notebook with different assembles for machine learning in their environment. From this notebook they provide a test you can run with their create_model to compare different models' metrics and time. We tested in total 14 different models with this test:

- ResNet
- ResCNN
- LSTM_FCN
- LSTM_FCN ('shuffle': false)
- FCN
- XceptionTime
- InceptionTime
- LSTM ('n_layers': 3, 'bidirectional': True)
- LSTM ('n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'mm_dropout': 0.2)

Type	Parameter	Description
Hyper-parameters	Number of epochs	50
	Batch size	32
	Learning rate	0.001
	Optimizer	Adam
Other	NaN handling	Fill with '0'
	Training method	Training on all players, training on one player multivariate and training on univariate
	Window size	1
	Prediction	Readiness

Table 4.1: Table shows the configurations of hyperparameters and other relevant information used for initial experimentation.

- LSTM ('n_layers': 2, 'bidirectional': True)
- LSTM ('n_layers': 2, 'bidirectional': False)
- LSTM ('n_layers': 3, 'bidirectional': False)
- LSTM ('n_layers': 1, 'bidirectional': True)
- xresnet1d34

Since we are experimenting on LSTM configuration in Keras as well, we added 6 different LSTM configurations in Tsai to be able to compare them with LSTM in Keras. When running every model listed above, we set the number of epochs to 50, learning rate to 0.001 and used Adam as optimizer which comes as default optimizer. For Tsai we treated this machine learning problem as a regression problem with a window size of 1.

Keras

Initial experimentation with Keras is aimed at showing us what keras has to offer and how it compares to Tsai. Keras thrives in flexibility

and make it easy for users to make changes and design their own model with their layered architecture. This is what we decided to experiment on in the initial stages of experimentation for Keras. For the models we decided to go for Keras' recurrent neural network layers found in their documentation. Models we will use in Keras are LSTM, GRU and SimpleRNN which is a basic RNN. These models will be setup in the same way dataset and training wise as Tsai by training and testing it on the two multivariate datasets and one univariate dataset as discussed before. Just as with Tsai we will be doing this with the handling of replacing NaNs with zero values. Because of the structure of which Keras is based on, completely copying Tsai's models with with Keras is hard. We will try to make the models as simple and default as we can based on what we know of the models chosen for testing with Tsai. When it comes to the hypterparameters for the models in Keras we chose the same as for Tsai, learning rate of 0.001, 50 number of epochs, batch size of 128 and optimizer as Adam and a window size of 1.

4.2.2 Results for Tsai

Experimentation for Tsai was done by testing 14 different models on the same dataset. The dataset was various self report data with '0' replacing NaNs. We ran three different tests on this dataset as seen in table 9.1 with the same number of epochs, batch size and learning rate. These tests are regression based test with the use of Tsai's TSRegression specified as a tfms parameter. This parameter as fastai defines it in their documentation, is a transformer and it is where you can specify how you want the models to work, classification or regression in our case. Prediction through the entirety of initial experimentation was done on the same player for every case in order to ensure continuity through testing. The following shows all results from initial experimentation.

Results from multivariate training on all players readiness levels

When training on all players, features was set to be every player's readiness level. In total this gave us a feature count of 15. After running the models listed previously on the configuration seen in table 9.1, we set it up to be compared as seen in figure 4.1. This figure is sorted in an descending order based on MSPE (Mean Square Prediction Error). At first we thought of measuring performance on the loss of the validation set. This gave us the initial thoughts that LSTM was the better choice for predicing on this dataset, having the lowest validation loss. After checking how the models

	arch	hyperparams	MSPE	train loss	valid loss	accuracy	time
0	ResNet	{}	2.1	0.432642	9.441711	0.541436	45
1	LSTM_FCN	{}	2.8	0.485278	7.884901	0.541436	60
2	LSTM	{'n_layers': 3, 'bidirectional': False}	2.9	1.220988	8.769525	0.541436	16
3	XceptionTime	{}	2.9	1.317182	8.706959	0.541436	181
4	InceptionTime	{}	3.2	0.492314	8.274734	0.541436	76
5	LSTM_FCN	{'shuffle': False}	3.3	0.494075	6.974593	0.541436	21
6	ResCNN	{}	3.8	0.483740	8.612706	0.541436	25
7	LSTM	{'n_layers': 2, 'bidirectional': False}	5.0	1.399889	8.634437	0.541436	12
8	FCN	{}	5.1	0.486839	8.301810	0.541436	16
9	LSTM	{'n_layers': 2, 'bidirectional': True}	5.6	1.272058	7.893655	0.541436	22
10	LSTM	{'n_layers': 1, 'bidirectional': True}	6.1	1.702367	7.608468	0.541436	11
11	LSTM	{'n_layers': 3, 'bidirectional': True}	7.2	0.959816	7.702870	0.541436	31
12	xresnet1d34	{}	7.3	0.460153	8.366636	0.541436	266
13	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	8.7	2.259892	7.772307	0.541436	9

Figure 4.1: Figure shows initial experimentation results from training on all players readiness levels and predicting readiness of one player

did on the test set with MSPE(MSE on test set) we came to another conclusion. MSE on the validation set is not the same as MSE on the test set, so calculating the MSPE (Mean squared prediction error) should be done in further experimentation to better show the true results.
Now knowing this, the results are looking different. ResNet, XceptionTime and InceptionTime did better than we expected when looking at MSPE score instead of validation loss. The top 5 models are now: ResNet, LSTM_FCN(shuffled), LSTM(3 layers), XceptionTime, InceptionTime and LSTM_FCN(no shuffle). When reflecting on the results we got, we can see the models tend to have a much lower training loss than validation loss. This is something we do not want to see when evaluating the models. Having such a low training loss compared to validation loss might indicate that the model is overfitting. Overfitting is when the training of models end up corresponding to close to the training set provided, resulting in worse performance when testing on other sets like the validation or test sets we use.

Concluding what we have discovered from this test, we have figured out ResNet seems to work the best on training on all so far. With LSTM_FCN, LSTM (3 layers) XceptionTime and InceptionTime coming in at the top 5 models. The models seems to be getting a low training loss but a high validation loss which might be caused by overfitting.

	arch	hyperparams	MSPE	train loss	valid loss	accuracy	time
0	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	8.5	7.203163	14.392179	0.59116	11
1	LSTM	{'n_layers': 1, 'bidirectional': True}	8.5	6.897178	14.284664	0.59116	14
2	LSTM	{'n_layers': 2, 'bidirectional': False}	9.2	6.566360	14.797298	0.59116	15
3	LSTM	{'n_layers': 3, 'bidirectional': False}	9.7	6.505358	14.506917	0.59116	20
4	ResNet	{}	11.0	4.385601	14.939713	0.59116	60
5	xresnet1d34	{}	11.1	4.379058	15.800803	0.59116	328
6	LSTM	{'n_layers': 3, 'bidirectional': True}	11.1	4.804148	15.544205	0.59116	40
7	LSTM	{'n_layers': 2, 'bidirectional': True}	11.5	5.704144	14.775404	0.59116	27
8	FCN	{}	11.7	4.521991	15.085564	0.59116	19
9	ResCNN	{}	11.8	4.457909	15.959368	0.59116	35
10	InceptionTime	{}	11.8	4.568074	14.953370	0.59116	81
11	LSTM_FCN	{}	12.2	4.545523	14.950475	0.59116	42
12	LSTM_FCN	{'shuffle': False}	12.2	4.517331	15.521642	0.59116	24
13	XceptionTime	{}	12.3	6.513585	17.068829	0.59116	257

Figure 4.2: Figure shows initial experimentation results from training on one players self report data and predicting on the same player

Results from multivariate training on one players self report data

As stated before, when training on self reported data on one player we used the following parameters as features: Readiness to play, Sleep Duration, Stress Level, Sleep quality, Fatigue, Soreness and Perceived Exertion. This time we got some worse results compared to the last experiment. The overall MSPE score is way up, now best coming in at 8.5 and training seems to go worse as well, figure 4.2 for reference. Validation loss was over 14 for every model with training loss lower but still much higher than experiments with training on all players. There are 4 models standing out in this test which are all different configurations of LSTM, after these the MSPE score tend to stay pretty close for the rest of the models. Even though this test indicated some sort of overfitting we still have something to work with for the next experiments with the same multivariate training method.

Results from univariate training

Results from training on an univariate dataset did at the start see some worse results than seen in Figure 4.3. The models were consistently getting MSPE scores around 16+ with both training and validation loss seen as high enough to explain the bad results. Both training and validation loss seemed to start high and end high throughout epochs. After testing different configurations of the base model parameters like learning rate, batch size, number of epochs and even training/validation

	arch	hyperparams	MSPE	train loss	valid loss	lr	time
0	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	2.8	7.072172	7.587441	0.001	23
1	LSTM	{'n_layers': 2, 'bidirectional': False}	2.8	6.720851	7.296099	0.001	31
2	LSTM	{'n_layers': 1, 'bidirectional': True}	2.8	6.862066	7.344058	0.001	29
3	LSTM_FCN	{}	3.3	5.675133	8.294541	0.001	51
4	LSTM	{'n_layers': 3, 'bidirectional': False}	3.4	5.655604	8.622453	0.001	41
5	LSTM	{'n_layers': 2, 'bidirectional': True}	3.4	5.782430	8.471456	0.001	51
6	LSTM_FCN	{'shuffle': False}	3.5	5.741974	8.360905	0.001	51
7	xresnet1d34	{}	3.6	5.529563	8.763271	0.001	772
8	LSTM	{'n_layers': 3, 'bidirectional': True}	3.6	5.436589	8.639433	0.001	72
9	ResCNN	{}	3.7	5.596797	8.388968	0.001	69
10	InceptionTime	{}	3.7	5.601092	8.324301	0.001	186
11	FCN	{}	3.8	5.715428	8.313446	0.001	44
12	ResNet	{}	3.8	5.579552	8.561468	0.001	111
13	XceptionTime	{}	4.0	8.028326	7.032271	0.001	487

Figure 4.3: Figure shows initial experimentation results from univariate training

splits we figured out the models did not get enough data to perform better. This lead to us changing the number of days needed for an prediction. Changing the number of days used in prediction to 3 days seemed to be a good fit for our univariate testing. After this change we got the results seen in figure 4.3. Now we had all models performing under 5 in MSPE score which was good to see. Running univariate training just like multivariate training having the same number of models, epochs and learning rate turned out not to be working that great. We will se in further experimentation if we need to run univariate training on its own set of hyperparameters.

4.2.3 Results for Keras

For experimentation with Keras we decided to first try running every model on the same parameters as we did for Tsai, see Table 9.1 for reference. Although Keras might react differently with the same parameters such as number of epochs and learning rate we decided to test how the results might vary from Tsai to Keras. Prediction will still be done on the readiness to play levels of the player selected. The player selected is also the same as in Tsai. In Keras we needed to "build" our own model with their layer building blocks. We tried not complicating the models in Keras too much for initial experimentation only having a 2 dropouts, 2 layers of chosen neural networks and 1 dense layer. We ran 3 different neural networks with the configuration seen in figure 4.4: LSTM, GRU and RNN. These models run on the same dataset with the same training, validation and test split so the data is completely the same in Keras and

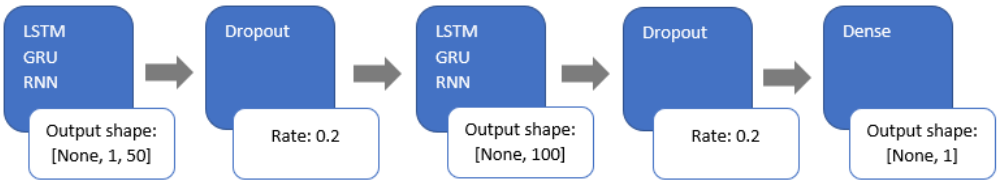

Figure 4.4: Figure shows how we designed the models used in experimentation for Keras. Each blue box represents a layer with the arrows pointing to the next layer. From each layer, the input is processed and passed on as output to the next layer. Once it reaches the dense layer the output will be narrowed down to a single output for each datapoint fed to the model. Every model has a dropout dividing the middle layer with the same dropout rate: 0.2.

Model	Train loss	Valid loss	MSPE	Time
LSTM	4.4500	4.2860	8.54	5.1
GRU	4.2453	4.5471	9.48	5
RNN	4.3890	4.2206	9.16	3.7

Table 4.2: Table shows results from multivariate training on all predicting on one in Keras

Tsai. For comparing we look at the MSPE score of the models, this is MSE for prediction data. This means doing a normal MSE validation on the predicted values and actual values from test set.

Results from multivariate training on all predicting on one

After running tests with Keras we got the results shown in table 4.2. When discussing how Keras is built and how it is intended to be used, we believed the running time and results from Keras would be slower and worse than Tsai' results. After this first experiment with training on all players and predicting on one, we were surprised to see the runtime so low at around 5 seconds with the batch size of 32 and 50 epochs. Training loss and validation loss looked promising when considering overfitting, no indication of any overfitting only a model with not the best accuracy score. Overall, the result from testing here had LSTM ahead of the other models and we will be considering this for further experimentation. All in all, LSTM got a score of 8.54 MSPE following with RNN at 9.16 and GRU at 9.48 MSPE score. Thing to note here is RNN is running a bit faster than the other models.

Model	Train loss	Valid loss	MSPE	Time
LSTM	8.1947	6.2341	12.92	4.9
GRU	8.0403	6.2185	13.1	5
RNN	8.0838	6.1591	13.08	3.4

Table 4.3: Table shows results from multivariate training on one predicting on one in Keras

Results from multivariate training on one predicting on one

Results of training on one player's self reported data is shown in table 4.3. From running tests on the different self reported data stated earlier we got some worse results compared to the first initial experiment ran with Keras. MSPE was looking way higher than anticipated with a training loss of almost double of the first Keras experiment. Increasing the epochs seemed to do nothing to the results, looking at the training and validation loss graph it was clear after 10-15 epochs the loss of both training and validation just bottomed out. We tried changing the optimizer, the learning rate, batch size and the results were still the same bottoming out around the same epoch. After adding shuffling to the data we saw some change in the MSPE score for the better, we therefore ran the model 10 times and took the average MSPE all runs. It went from a score of 16.34 on RNN to an average score of 8.45. When looking at a graph produces in one of the runs with a test size of 50 found in figure 4.5 we see the difference slightly. The left graph shows how the model more closely can predict when the readiness spikes compared to the other graph where the prediction seems to lag a bit more. The model without shuffling seems to need a day after readiness ascents or descents to predict it is going up or down. To conclude the results gotten, LSTM got the better of the 3 models even if it is only with a slight margin and training loss compared to the MSPE looks promising if we only could make the models more exploratory.

Results from univariate training

As expected testing with the univariate dataset saw worse results than multivariate testing. Overall higher in everything, training loss, validation loss and MSPE. These results were done with 1 day lag and predicting the day after and it revealed the model was not exploratory enough, table 4.4 shows the MSPE score of this test. It seemed the model was going kind off safe and not predicting volatile movements in readiness. If the actual

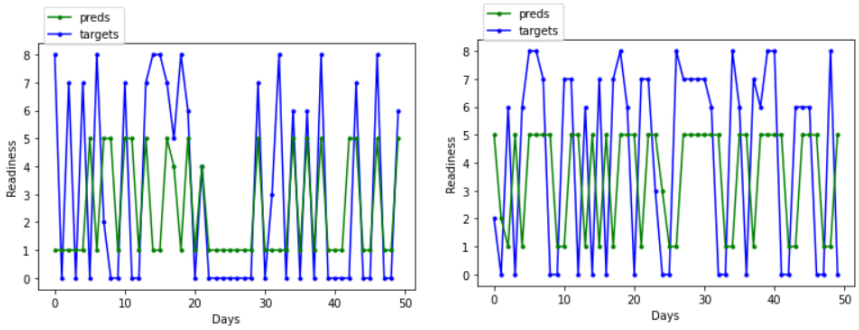

Figure 4.5: Figure shows what the features in an array of 7 does to the accuracy of the model. Lower score results in a higher model accuracy

Model	Train loss	Valid loss	MSPE	Time
LSTM	8.6053	7.3473	15.66	5
GRU	8.5192	7.3034	16.24	5.3
RNN	8.6929	7.2623	16.34	3.5

Table 4.4: Table shows results from univariate training in Keras

values were going all over the place between 1-5, the model seemed to safe it by predicting the readiness was a 3 on every point. To make the model more exploratory we decided to change the number of days used in prediction, namely the days used for prediction/window size. When changing the model to use 2-5 past days for predicting, the model seemed to go to the more extreme values. The MSPE did change for the worse, but the model seemed to catch the trends of higher and lower values and not only predict the values in between target values. This caught our eye and is something we might continue testing after initial experimentation is done. Even though the results varied the different setups with a bigger window size for prediction might be something useful for later and should be looked at further down in the experimentation. From the results here we can see the 3 different models are quite alike, LSTM only having a slight advantage with half a point better MSPE score.

4.3 Discussion of initial experimentation

After doing the initial experimentation we collected a valuable amount of knowledge we can use for further experimentation. First off for Keras, the result from all three experiments showed us training on all players had the best results both in the form of better fit when training (training and

validation loss) and when testing on the unbiased test set. LSTM coming in at the best model almost 1 MSPE score better than the other models. An ongoing theme in these experiments is the low exploratory factor the models had. The training loss and validation loss was consistently high and we found it hard to make the loss better. Changing number of epochs, learning rate, batch size, activation functions and optimizers did little to nothing for the loss and MSPE score. It was not until we tried changing the window size before we saw some change in the behaviour of the models. This was certainly noticeable on the univariate dataset. The univariate dataset has less data for the models to use, so changing the dataset by adding more days for prediction seemed help the models perform better.

When it comes to Tsai, the result on all three dataset tended to be a little better than keras' results. Training on all players gave the best results out off all three training methods.Having MSPE score all the way down at 2.1 with the top 5 models coming close at around or under 3 in score. Even though training loss and validation loss was seen having a higher score than the prediction itself the models seemed to perform quite well. Univariate training saw the next best results with a LSTM scoring 2.8 and the top 5 models going around the same with a max MSPE of 3.4 Training on 7 different self reported data gave the worst result out off the three, generally doing worse in both loss and MSPE score. For this training the LSTMs were the best so that is something to note. Lowering the MSPE score and loss was difficult and weirdly did not really go down after the testing we did. When moving on to the next experimentation we will take the note of the best performing models from each dataset we got during initial experimentation. The next experiments will be focused on how performance of models change when using new NaN handling methods.

What we have learned overall from initial experimentation is Tsai's univariate training with zero as NaN replacement saw the best results. From this experiment we can conclude the models needs to be more exploratory in order to catch more exact where trends in readiness shift. In figure 4.6 we see how the typical models performed. They did not reach far enough to catch the tops and bottoms of readiness. This might be caused by having such a low learning rate, low batch size and not enough training. We thought the models were overfitting, but what if the models never actually were learning good enough since they had such a high training loss as well as high validation loss. For further experimentation we will increase the learning rate and batch size since those two go hand in hand. We will also try increasing the number of epochs. Shuffling when training saw some good results in Keras when training on one player multivariate dataset, this is something to keep in mind when moving on with experimentation.

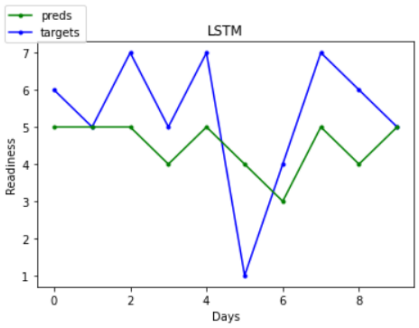

Figure 4.6: Figure shows how the models generally performed during initial experimentation

4.4 Filling Nan's with zero/mean/interpolate Tsai

4.4.1 Overview

In order to find the best predictive model, the dataset needs to be set up with the highest likelihood of helping training. There are multiple ways to prepare a dataset for training. Due to our natural way of collecting data with PMSys' self report system, missing data exists. Missing data will be referred to as NaN (Not a Number) in this thesis and is a commonly used word for missing data points. Models used in this thesis are not designed for handling NaNs, so we needed to find which was the best way of handling them. You can either delete or replace NaNs, both could work perfectly fine depending on what use the models are for. A conseqence of deleting NaNs is consistency regarding patterns of seasonality (e.g athletes doing the same workout every Monday reporting the same values). Replacing NaNs will maintain the integrity of the data and will be used in the experiments going forwards. For NaN handling we will further experiment with replacing with mean values and interpolate values. After that we will compare the results and discuss what models worked the best for each NaN handling method. Testing will happen on the same test set as used for every experiment up to this point. This test set has NaNs filled with zeros and we will test the models both on a 1 day and 3 days prediction basis to see what performs best. Going in to this experiment we decided from the last experiment on zero values that we needed more exploration. We will therefore also increase the learning rate to 0.02, increase the batch size to 128 and number of epochs to 75-100.

We decided to test the newly updated model hyperparameters on the same dataset used in initial experimentation with zero values as NaN replacements. Since the hyperparameters learning rate, batch size, number of epochs and window size changed, the models will produce

Type	Parameter	Description
Hyper-parameters	Number of epochs	75-100
	Batch size	128
	Learning rate	0.002
	Optimizer	Adam
Other	NaN handling	Fill with interpolate values, mean values and '0'
	Training method	Training on all players, training on one player multivariate and training on univariate
	Window size	1 and 3
	Prediction	Readiness

Table 4.5: Table shows updated overview of hyperparameters and other useful information used in further experimentation

different results on Zero value NaN replacement. So, this time we will test how the models perform on the new hyperparameters: learning rate 0.002, batch size 128, 100 number of epochs and 1 vs 3 day window size.

4.4.2 Multivariate training on all players

Interpolate

Interpolation uses values next to the datapoint which are to be replaced. The NaNs will be replaced with the result of the interpolate function described in the prior section. If there are any NaNs in the beginning or end of the dataset which might not have any values next to them to use for interpolation, these values will be replaced with zero values. Fortunately, there are only a few scenarios where this will happen, for example there are some player's dataset that start with a few NaNs and end with NaNs as well, here they will instead start with a few zeros in order for the dataset to keep its integrity. When training on all players we initially got promising

	arch	hyperparams	MSPE	train loss	valid loss	lr	time
0	InceptionTime	{}	2.0	0.059487	2.178447	0.002	52
1	xresnet1d34	{}	2.1	0.051372	3.003471	0.002	237
2	ResNet	{}	3.0	0.034754	1.954914	0.002	40
3	LSTM_FCN	{}	3.1	0.072175	1.465953	0.002	52
4	LSTM_FCN	{'shuffle': False}	3.3	0.075445	2.585382	0.002	17
5	FCN	{}	3.8	0.076544	2.269284	0.002	14
6	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	3.9	2.358707	3.420925	0.002	9
7	ResCNN	{}	4.1	0.068601	2.075466	0.002	20
8	LSTM	{'n_layers': 1, 'bidirectional': True}	4.2	1.822145	3.538503	0.002	11
9	LSTM	{'n_layers': 2, 'bidirectional': True}	4.9	1.343420	1.992840	0.002	19
10	LSTM	{'n_layers': 2, 'bidirectional': False}	5.0	1.453991	2.252820	0.002	11
11	LSTM	{'n_layers': 3, 'bidirectional': False}	5.0	1.331216	2.479043	0.002	15
12	LSTM	{'n_layers': 3, 'bidirectional': True}	5.3	1.057172	2.188600	0.002	27
13	XceptionTime	{}	13.9	11.290114	16.590834	0.002	151

Figure 4.7: Figure shows results from training on all players with interpolated values as replacements with the best window size of 3

results. When testing the models up against multivariate training on all we first tried running the models on a higher lr and batch size on 50 number of epochs. In addition to this we tested whether having 1, 3 or more days as window size would have an impact on performance with these models hyperparameters. Testing with one day window size seemed to give good results in the beginning, giving an MSPE of just under 3 for the top 5 performing models. From these test we saw the training loss and validation loss could have been better. Training loss saw a steady decrease over time, but validation loss saw an initial decrease followed by a flattening around 30 epochs and even an increase in some models. We thought the model might be overfitting, so we tried running some more test with a higher number of epochs and a lower learning rate to test this theory out. This seemed to only make our model weaker and less accurate, so we tried increasing the learning rate instead since the model did not seem to overfit as much as we thought. After these changes we got the results seen in figure 4.7. Validation loss decreased over time and training loss stagnated and flattened around where they lie in figure 4.7.

Concluding the changes made; they did absolutely give a boost to the models. In addition, interpolation seems to be doing the models a solid for performance as well. InceptionTime did the best followed closely by xresnet1d34, ResNet and the two LSTM_FC configurations.

Mean

When testing with mean values as replacement for NaN we described earlier how the mean value of the dataset is calculated and this value will be the replacement for every NaN. In the beginning of experimentation on this dataset we tested with number of epochs set to 50, batch size of 128 and a learning rate of 0.001. After running all the model through testing and prediction on the test set, we saw the models did not seem to be training as well as we would want them to be. Following the same changes as from interpolation we went to test with a higher learning rate and higher number of epochs. This immediately gave us better results and the models was training better, giving more promising results. In order to test where the boundary was for how high the learning rate can go before we caught a glimpse of overfitting we continued to lower the learning rate. From this we saw how the models operated on different hyperparameters. Some models such as InceptionTime worked better than LSTM on a lower learning rate and XceptionTime seemed to not be training optimally on this dataset at all. When reaching a learning rate of 0.002 we saw the best results yet, not extreme overfitting and we got some good MSPE scores from this learning rate aswell. In addition to increasing the learning rate we tried increasing the number of epochs to exaggerate overfitting. It seemed for the majority of the models, having around 100 epochs did the best for results. Running this training with a window length of 3 saw better results compared to a length of 1. Results can be seen in figure 4.8.

Concluding the results we got from testing with mean value replacement, we saw a better results with the changes made in number of epochs with around 100 to be the best. Batch size of 128 with a learning rate of 0.002 did also good for the models' performance. The top performing model was LSTM_FCN (no shuffling) followed by LSTM (3 layers), FCN, LSTM_FCN (shuffeled) and InceptionTime. Overall interpolation did better than mean value replacement on this training method.

Zero

After testing with the same dataset from initial experimentation for zero value replacements we go the results seen in figure 4.9. This time around we got a worse performance, so the changes in hyperparameters and window size of 3 only worked for the new NaN handlings tested. With the best model LSTM with 3.7 MSPE we can compare it to the best model from initial experiment where we saw the best model ResNet with a 2.1 MSPE. Concluding training on all players with Zero as NaN handling, window size of 1 saw the best results from the initial experimentation. ResNet got

	arch	hyperparams	MSPE	train loss	valid loss	lr	time
0	LSTM_FCN	{'shuffle': False}	2.4	0.110679	1.738391	0.002	48
1	LSTM	{'n_layers': 3, 'bidirectional': False}	2.6	0.232191	2.204723	0.002	59
2	FCN	{}	3.3	0.107799	1.881077	0.002	30
3	LSTM_FCN	{}	3.5	0.111575	1.825967	0.002	105
4	InceptionTime	{}	3.5	0.093699	1.640557	0.002	138
5	ResCNN	{}	3.7	0.101849	1.620358	0.002	60
6	ResNet	{}	3.9	0.089478	1.688112	0.002	85
7	LSTM	{'n_layers': 2, 'bidirectional': False}	4.0	0.243820	2.694526	0.002	42
8	xresnet1d34	{}	4.6	0.096220	1.939691	0.002	433
9	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	5.3	0.740426	3.605971	0.002	25
10	LSTM	{'n_layers': 2, 'bidirectional': True}	5.4	0.224897	2.342279	0.002	80
11	LSTM	{'n_layers': 3, 'bidirectional': True}	5.7	0.172745	2.318212	0.002	114
12	LSTM	{'n_layers': 1, 'bidirectional': True}	6.2	0.498333	2.965720	0.002	39
13	XceptionTime	{}	9.2	5.918781	4.562889	0.002	313

Figure 4.8: Figure shows results from training on all players with mean value NaN replacement

the best results on this combination of training method and NaN handling on this dataset with a 2.1 MSPE.

4.4.3 Multivariate training on one player's self report data

Interpolate

When testing the models with interpolate NaN handling we decided to first check how well the models worked on the same model hyperparameters (learning rate, batch size and such) as being changed from initial experimentation. The results achieved from these parameters looked promising and can be seen in figure 4.10. One model not suited for this dataset seemed to be XceptionTime, doing the worst out off all models with a MSPE score of 23.2 and very high training and validation loss. This time we can see the LSTMs did not perform as well as the other models such as ResNet, InceptionTime and FCN. Bidirectional LSTM with two layers and the FCN versions of LSTM on the other hand did perform better than stock LSTM scoring a 1.8 and 2.4. Training loss this time around were the lowest seen so far indicating that the models are well trained if not leaning on the overfitting side. Validation loss saw a steady decrease to a certain epoch where they stopped and stayed still for the remainder of epochs. To try and make the models even better we decided to try make the models more resistant against overfitting. This saw an increase in MSPE score and a much higher training/validation loss. Since the validation loss stopped decreasing, increasing the number of epochs only worsened the performance of the models. Having a number of epochs of 50 got the best res-

	arch	hyperparams	MSPE	train loss	valid loss	lr	time
0	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	3.7	1.225896	7.763178	0.002	43
1	LSTM	{'n_layers': 2, 'bidirectional': True}	4.6	0.578034	13.855516	0.002	90
2	LSTM	{'n_layers': 3, 'bidirectional': False}	5.4	0.588404	11.429290	0.002	66
3	LSTM	{'n_layers': 2, 'bidirectional': False}	7.1	0.767638	9.559166	0.002	47
4	xresnet1d34	{}	7.5	0.520288	11.960086	0.002	553
5	LSTM	{'n_layers': 3, 'bidirectional': True}	8.3	0.536142	13.512471	0.002	149
6	ResCNN	{}	8.9	0.511391	8.824788	0.002	59
7	InceptionTime	{}	11.0	0.508765	10.251256	0.002	239
8	FCN	{}	11.1	0.530064	8.064639	0.002	53
9	LSTM_FCN	{'shuffle': False}	11.1	0.517963	8.093196	0.002	75
10	ResNet	{}	11.8	0.486806	9.055149	0.002	100
11	LSTM_FCN	{}	12.1	0.530712	8.459752	0.002	162
12	LSTM	{'n_layers': 1, 'bidirectional': True}	14.0	1.117885	9.331154	0.002	45
13	XceptionTime	{}	16.5	2.943991	8.031458	0.002	457

Figure 4.9: Figure shows results from training on all players with zero value replacement

ults. The top 5 models from this test was LSTM (2 layers bidirectional), LSTM_FCN(shuffeled), FCN, ResNet and LSTM_FCN.

Mean

For training on one player's self reported data we tested with the newest findings from experimentation with interpolation on the same dataset. In addition to this we tried running the models on with different number of days for prediction. Among the best prediction windows was 1 or 3 both doing good on their own hyperparameters. Testing the different hyperparameters with the different prediction windows showed us how having a prediction window of 3 overfitted the models more easily and in turn needed adjustment to reach optimized result. After going through different sets of epochs, having around 100 number of epochs seemed to be the best. When reaching 150 epochs the models very overfitting and getting a worse result than on 50 epochs. It seemed most models had the best results with just under 100 epochs while the different LSTM configurations trained slower and had the lowest loss around 140 epochs. The best overall scores took place on 75 epochs with a learning rate of 0.002 and batch size of 128. Results can be seen in figure 4.11.

Zero

We also tested to see whether or not the new hyperparameters would work on the dataset used in initial experimentation with zero values as NaN

	arch	hyperparams	MSPE	train loss	valid loss	lr	time
0	LSTM	{'n_layers': 2, 'bidirectional': True}	1.8	1.623857	2.060045	0.002	26
1	LSTM_FCN	{}	2.4	0.078557	1.986495	0.002	22
2	FCN	{}	2.7	0.058293	1.805477	0.002	10
3	ResNet	{}	2.7	0.030852	1.428579	0.002	23
4	LSTM_FCN	{'shuffle': False}	2.7	0.074915	1.911883	0.002	15
5	InceptionTime	{}	2.9	0.037494	1.370284	0.002	45
6	LSTM	{'n_layers': 3, 'bidirectional': True}	3.1	1.587822	2.461262	0.002	38
7	ResCNN	{}	4.3	0.053183	1.618805	0.002	16
8	xresnet1d34	{}	5.0	0.042916	1.656421	0.002	151
9	LSTM	{'n_layers': 3, 'bidirectional': False}	5.2	3.916703	2.560075	0.002	19
10	LSTM	{'n_layers': 2, 'bidirectional': False}	5.6	2.291973	2.600882	0.002	13
11	LSTM	{'n_layers': 1, 'bidirectional': True}	5.7	2.242867	2.465302	0.002	13
12	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	6.0	2.493602	2.727535	0.002	8
13	XceptionTime	{}	23.2	15.050129	17.847687	0.002	107

Figure 4.10: Figure shows results from training on one player with interpolated values replacement dataset

	arch	hyperparams	MSPE	train loss	valid loss	lr	time
0	LSTM	{'n_layers': 2, 'bidirectional': False}	1.6	1.292834	2.637736	0.002	44
1	ResNet	{}	1.8	0.122189	1.413353	0.002	95
2	LSTM	{'n_layers': 2, 'bidirectional': True}	1.8	0.904589	2.409209	0.002	84
3	LSTM	{'n_layers': 3, 'bidirectional': True}	2.0	0.685066	2.328626	0.002	124
4	xresnet1d34	{}	2.4	0.122228	1.596390	0.002	509
5	LSTM	{'n_layers': 3, 'bidirectional': False}	2.4	0.787063	2.604806	0.002	63
6	LSTM	{'n_layers': 1, 'bidirectional': True}	2.8	1.558845	2.241021	0.002	41
7	ResCNN	{}	3.0	0.140854	1.531299	0.002	59
8	LSTM_FCN	{}	3.0	0.144477	1.782284	0.002	72
9	LSTM_FCN	{'shuffle': False}	3.1	0.152265	1.577629	0.002	51
10	InceptionTime	{}	3.2	0.124173	1.462434	0.002	172
11	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	3.5	2.011983	2.237561	0.002	27
12	FCN	{}	3.9	0.147376	1.715837	0.002	33
13	XceptionTime	{}	10.2	5.812951	4.665380	0.002	401

Figure 4.11: Figure shows results from training on one player with mean value replacement dataset

	arch	hyperparams	MSPE	train loss	valid loss	lr	time
0	LSTM	{'n_layers': 1, 'bidirectional': True}	4.5	2.498794	8.735361	0.002	51
1	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	4.7	2.836312	8.626534	0.002	32
2	LSTM	{'n_layers': 2, 'bidirectional': True}	6.9	2.079684	8.881829	0.002	93
3	LSTM	{'n_layers': 3, 'bidirectional': True}	7.4	1.705331	8.658513	0.002	141
4	LSTM	{'n_layers': 2, 'bidirectional': False}	7.7	1.980997	8.514883	0.002	50
5	LSTM	{'n_layers': 3, 'bidirectional': False}	8.1	1.721189	8.216838	0.002	80
6	ResNet	{}	8.9	0.883664	7.866220	0.002	111
7	LSTM_FCN	{'shuffle': False}	9.4	0.930324	7.222911	0.002	57
8	LSTM_FCN	{}	10.6	0.917823	7.181130	0.002	81
9	xresnet1d34	{}	11.3	0.890685	8.079105	0.002	541
10	ResCNN	{}	11.7	0.894884	6.905819	0.002	57
11	FCN	{}	12.0	0.906821	7.268569	0.002	42
12	InceptionTime	{}	12.3	0.890763	7.000956	0.002	177
13	XceptionTime	{}	14.0	4.374053	7.033852	0.002	394

Figure 4.12: Figure shows results from training on one player with zero value replacement dataset

replacement. This time we got some better results, having a much lower overall MSPE of all the models. This run, all the LSTM configurations was doing the best for all models with 1 layered bidirectional LSTM doing the best at 4.5 MSPE. Comparing this to the three other NaN handling methods, this one did the worst by a good margin. One note to take with us from this test is that the changes made in hyperparamters made the results overall better than from the initial experimentation. Comparing the best model from this run we got a 4.5 MSPE and from inital experimantation we got 8.5 on the same model which also performed the best that time as well. Loss from training saw an overall decrease which is a good thing along with MSPE score doing better. So to conclude the results from multivariate training on one player with the zero dataset, this hyperparameter configuration saw the best results even though it did perfrom the worst out off all three NaN handling methods from this training method. Results can be seen in figure 4.12.

4.4.4 Univariate training

Interpolate

Interpolate on univariate training did not seem to go any good. From the results seen in figure 4.13 it is clear that the models did not perform well on interpolate compared to the other NaN handling methods. We tried tweaking the hyperparameters for the models only to worsen the results. From running on the same learning rate of 0.002 with 200 epochs we saw the training loss steadily go down while the validation loss was

	arch	hyperparams	MSPE	train loss	valid loss	lr	time
0	FCN	{}	4.4	1.369777	2.491647	0.002	25
1	ResNet	{}	4.5	1.299925	2.521891	0.002	85
2	InceptionTime	{}	4.5	1.311959	2.480267	0.002	123
3	LSTM_FCN	{'shuffle': False}	4.6	1.388479	2.508346	0.002	41
4	xresnet1d34	{}	4.7	1.313275	2.484847	0.002	339
5	LSTM_FCN	{}	4.7	1.380093	2.503235	0.002	33
6	LSTM	{'n_layers': 2, 'bidirectional': True}	4.8	1.835499	2.003026	0.002	64
7	ResCNN	{}	5.3	1.333592	2.496067	0.002	62
8	LSTM	{'n_layers': 3, 'bidirectional': True}	5.5	1.634104	2.424113	0.002	97
9	LSTM	{'n_layers': 1, 'bidirectional': True}	5.7	3.364956	1.730281	0.002	32
10	LSTM	{'n_layers': 3, 'bidirectional': False}	5.9	2.692317	1.938431	0.002	47
11	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	6.3	3.581283	1.779340	0.002	21
12	LSTM	{'n_layers': 2, 'bidirectional': False}	6.8	2.909328	2.160264	0.002	34
13	XceptionTime	{}	13.9	8.364777	11.253026	0.002	261

Figure 4.13: Figure shows results from univariate training with interpolated values as replacements dataset

seen increasing after around 100 epochs. This was a sign of overfitting, so we tried running the model on a lower learning rate 0.02 and 0.006 with a lower number of epochs of 50, only to see the results once again for the worse. This was the optimal hyperparameters to run on univariate training with interpolation. Among the top 5 model we have FCN at 4.4 MSPE followed by ResNet, InceptionTime, LSTM_FCN and xresnet1d34. Compared to the other results gotten from earlier experiments univariate training with interpolation is not something to keep experimenting on with Tsai, Keras might see some other results but as far as this goes, we have seen better results.

Mean

In figure 9.12 we see results of training and predicting with the mean value NaN handling dataset. Compared to interpolation, the models seem to respond better with mean values as fillers. Loss from training and validation seems to be going at the same rate as for interpolation, only this time the MSPE is about half. We tried running these models on an even increasing learning rate only to have the models overfitting, even after increasing the batch size. Increasing batch size and learning rate did nothing do the eventual overfitting which seemed to be happening around the same epoch of 50 and up. From what time we had this was the best results we got from each model as seen in figure 4.14. Concluding the top 5 models from univariate training on mean NaN replacement: LSTM(3 layers and bidirectional) at 1.8 MSPE, LSTM(1 layer with dropout), FCN,

	arch	hyperparams	MSPE	train loss	valid loss	lr	time
0	LSTM	{'n_layers': 3, 'bidirectional': True}	1.8	0.996747	2.576930	0.02	130
1	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	2.0	1.607714	2.752922	0.02	26
2	FCN	{}	2.0	1.079658	2.652040	0.02	32
3	LSTM_FCN	{}	2.4	1.081106	2.641725	0.02	41
4	ResNet	{}	2.5	1.019115	2.579996	0.02	114
5	LSTM	{'n_layers': 2, 'bidirectional': True}	2.5	0.998384	2.585927	0.02	84
6	LSTM_FCN	{'shuffle': False}	2.5	1.089422	2.733039	0.02	50
7	LSTM	{'n_layers': 3, 'bidirectional': False}	2.7	1.001184	2.847146	0.02	64
8	xresnet1d34	{}	2.8	1.063172	2.679579	0.02	469
9	ResCNN	{}	2.8	1.034423	2.739732	0.02	77
10	LSTM	{'n_layers': 2, 'bidirectional': False}	2.9	1.008767	2.780128	0.02	44
11	LSTM	{'n_layers': 1, 'bidirectional': True}	3.2	1.265196	2.928819	0.02	42
12	InceptionTime	{}	3.4	1.031957	2.675532	0.02	170
13	XceptionTime	{}	3.5	1.091665	2.972295	0.02	364

Figure 4.14: Figure shows results from univariate training with mean value replacements dataset

LSTM_FCN and ResNet.

Zero

Replacing NaN values with zero when training on the univariate training set showed the best results MSPE wise. Every model we see in figure 4.15 is trained with a window size of 3. ResCNN showed the most promising the results with a score of 1.1. Even though the training loss and validation loss did not seem to agree with how well the model performed, it was ran through multiple times and showed good results each time. A score of 1.1 is the best score gotten so far in this thesis and following ResCNN the rest of the models performed great as well. InceptionTime scoring 1.3, ResNet 1.6, Xceptiontime 1.6 and FCN on 2.0.

4.4.5 Comparing the results from training methods

In this section we will compare the graphs of the best and worst models predicted above to see how they behave on the different datasets and NaN handling methods. We will look at how the behaviour differs in how they predict the trends of readiness, along with how accurate they are. We chose to use the best models and worst models because the models in between are typically a combination of either the best of the worst. The other models can be excepted to be doing worse than the best models with some exceptions which will be pointed out.

	arch	hyperparams	MSPE	train loss	valid loss	lr	time
0	ResCNN	{}	1.1	2.107155	8.164522	0.02	52
1	InceptionTime	{}	1.3	2.101861	8.050300	0.02	112
2	ResNet	{}	1.6	2.093576	8.095953	0.02	75
3	XceptionTime	{}	1.6	2.126707	8.254872	0.02	232
4	FCN	{}	2.0	2.188152	8.319682	0.02	20
5	LSTM_FCN	{}	2.1	2.167641	8.201663	0.02	26
6	LSTM_FCN	{'shuffle': False}	2.1	2.182993	8.163590	0.02	32
7	LSTM	{'n_layers': 2, 'bidirectional': False}	2.5	2.133295	7.963003	0.02	28
8	LSTM	{'n_layers': 3, 'bidirectional': False}	2.8	2.069256	8.175592	0.02	40
9	LSTM	{'n_layers': 3, 'bidirectional': True}	3.0	2.024731	8.106848	0.02	80
10	LSTM	{'n_layers': 2, 'bidirectional': True}	3.3	2.028742	8.117386	0.02	53
11	LSTM	{'n_layers': 1, 'bidirectional': False, 'fc_dropout': 0.2, 'rnn_dropout': 0.2}	3.7	2.747152	7.440007	0.02	16
12	LSTM	{'n_layers': 1, 'bidirectional': True}	4.0	2.473462	7.552545	0.02	26
13	xresnet1d34	{}	4.1	2.147679	8.290970	0.02	300

Figure 4.15: Figure shows results from univariate training with zero value replacements dataset

NaN handling	Model	MSPE
Interpolated	InceptionTime	2.0
Mean	LSTM_FCN (Shuffle: false)	2.4
Zero	LSTM(1 layers with dropout)	3.7

Table 4.6: Table shows the best models when training on all players.

Multivariate training on all players

From training on all players readiness levels we see the behaviour from the three NaN handling methods differ to some extent. Look at figures 4.16-4.18 for reference. When looking at the interpolation we see the best model dipping and reaching the bottom quite well, even though it is a day behind in prediction the model seems to got lower than the two other NaN handling methods. From experimentation interpolation saw the best results when it comes to MSPE score which makes sense when looking at the graph seeing it following the ups and down of target values more closely. The mean value NaN handling seems to be rounding the prediction more than the other, only going as low as 3 for a missed prediction. The LSTM from zero value NaN handling did also see a 3 as the lowest prediction aswell, indicating that interpolation has an edge when it comes to more precisely predicting the actual high and low values of readiness. When it comes to the worst models for the three, we have XceptionTime. This model did not seem to do very well on these datasets with a MSPE score of around 15 on average across the three.

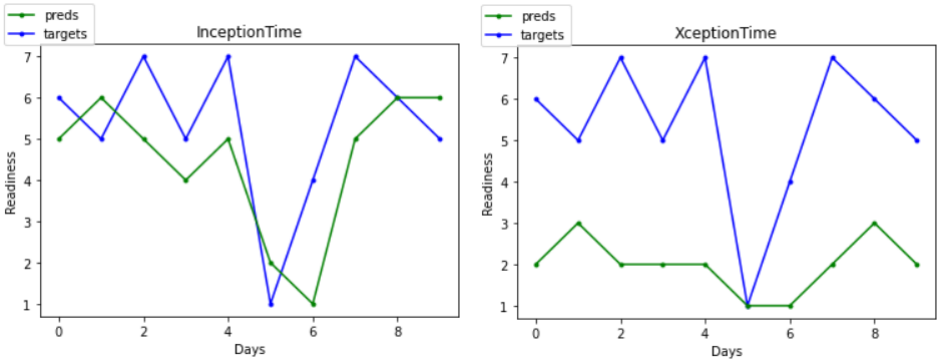

Figure 4.16: Best and worst models from interpolation replacement on multivariate training on all players

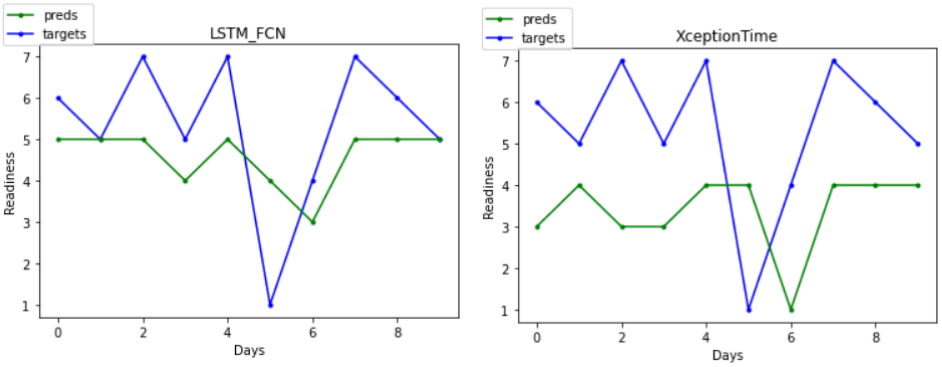

Figure 4.17: Best and worst models from mean value replacement on multivariate training on all players

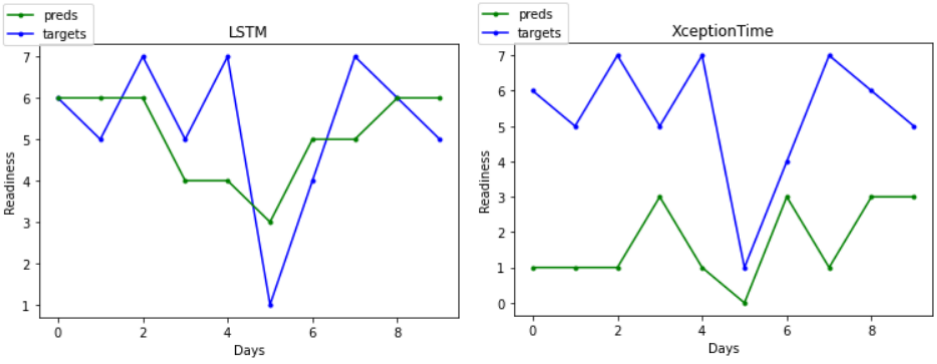

Figure 4.18: Best and worst models from zero value replacement on multivariate training on all players

NaN handling	Model	MSPE
Interpolated	LSTM (2 layers, bidirectional)	1.8
Mean	LSTM(2 layers)	1.6
Zero	LSTM (1 layer, bidirectional)	4.5

Table 4.7: Table shows the best models when training on one player.

Multivariate training on one player

Multivariate training on one player's self reported data showed much more promising results. From the figures 4.19-4.21 we can see the results in the graphs from this training. Interpolation here did not do as well as in training on all players, the graph shows how the model did not catch the lowest point of targets, but this time it predicted the highest point, only one day after it happened. This can be explained by the randomness that comes with machine learning models at times. For the mean value predictions we saw the LSTM perform the best so far, catching both the dip in readiness and the peak after that and on the exact day it happened as well. The model did not do that great in the beginning if we are focusing on how the MSPE would look, the prediction is off by one or two readiness levels, but the model can detect that indeed there are fluctuations. For the zero value NaN handling we see some same results compared to the mean value NaN handling. The model predict the trends quite well, and on the exact day it is peaking or dropping. From this NaN handling we see the prediction are a bit more extreme compared to mean value, going on a bigger range at the start. This decreases of course decreases the MSPE score of the model and is not the best if any teams are to decide what the readiness is of a player. When it comes to the worst models of this experiment we see XceptionTime this time as well. Coming in at the same performance as for the prior training method.

Univariate training

Univariate training saw the best overall MSPE scores so far through all of experimentation, we will now take look at the best performing models from each univariate Nan handling methods seen in figures 4.22-4.24. Interpolation on the univariate set saw the worst results out of the three with the best MSPE of 4.4 on FCN. This model as seen in figure 4.22

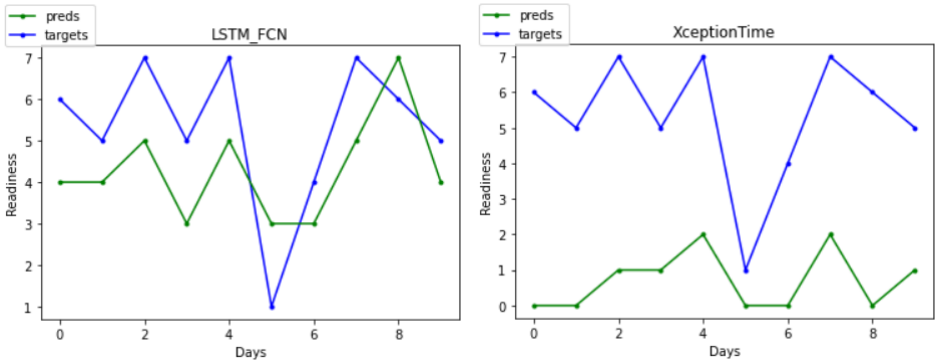

Figure 4.19: Best and worst models from interpolation replacement on multivariate training on one player

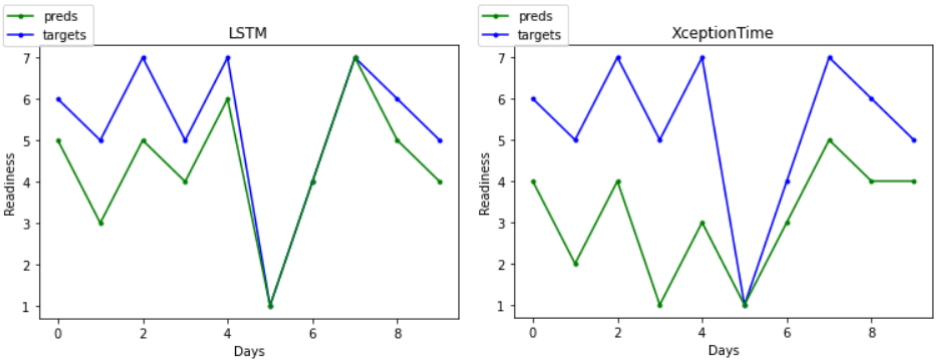

Figure 4.20: Best and worst models from mean value replacement on multivariate training on one player

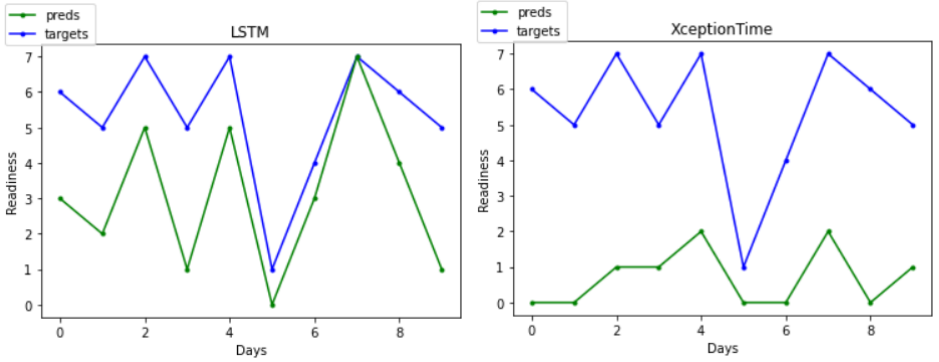

Figure 4.21: Best and worst models from zero value replacement on multivariate training on one player

NaN handling	Model	MSPE
Interpolated	FCN	4.4
Mean	LSTM (3 layers, bidirectional)	1.8
Zero	ResCNN	1.1

Table 4.8: Table shows the best models when training with univariate set.

just missed too many target values making the MSPE score that low and overall the models from univariate interpolation did not cooperate well together. It did not hit the bottoming trend of the target values but it almost hit the peaking trend only off by one readiness level. When it comes to univariate training with mean value NaN handling we see a much better result. Even though the model misses a few targets it seemed to get the trends correct, predicting lower and higher at the right time only missing by a couple of readiness levels. Zero value replacement saw the best results and as seen in figure 4.26 ResCNN closely followed the target values, atleast when it came to the lower target values. For the first 5 days it did not capture the top of readiness, but for all the dips in readiness this model hit every target value on point. When comparing the results of ResCNN to xresnet1d34 which also is a ResNet, we can see how xresned1d34 was more volatile. This volatility turned out to be costly for the overall MSPE score but having a model predicting both peaking and dropping in readiness should hold some value as well.

4.5 Filling Nan's with zero/mean/interpolate Keras

4.5.1 Overview

Experimenting with Keras will use knowledge gotten from all the runs with Tsai as well as prior results with Keras. Hyperparameters such as learning rate and number of epochs are tweaked from last experimentation with the zero values NaN handling dataset. Since Keras seemed to be able to run with a higher number of epochs we decided to increase the learning to 0.02 and epochs have increased to 500 for the first runs. Batch size is also increased to 128 in order to counteract overfitting for this experiment. We are now experimenting on a bigger scale by training and predicting 10 times over and taking the average results from these runs. We do this with Keras because the runtime of Keras is so low

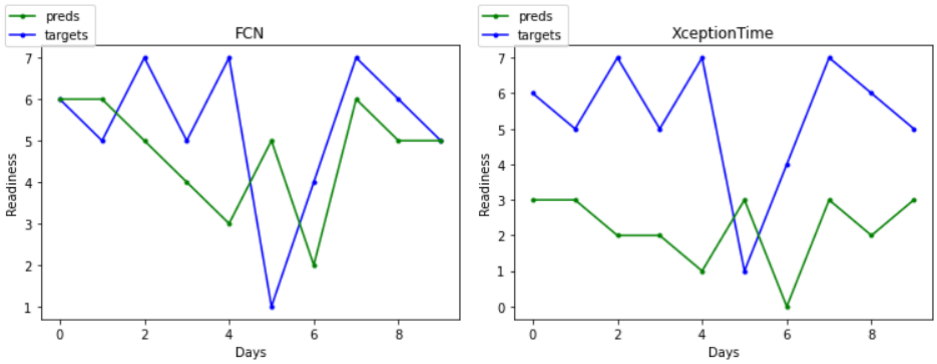

Figure 4.22: Best and worst models from interpolation replacement on univariate training

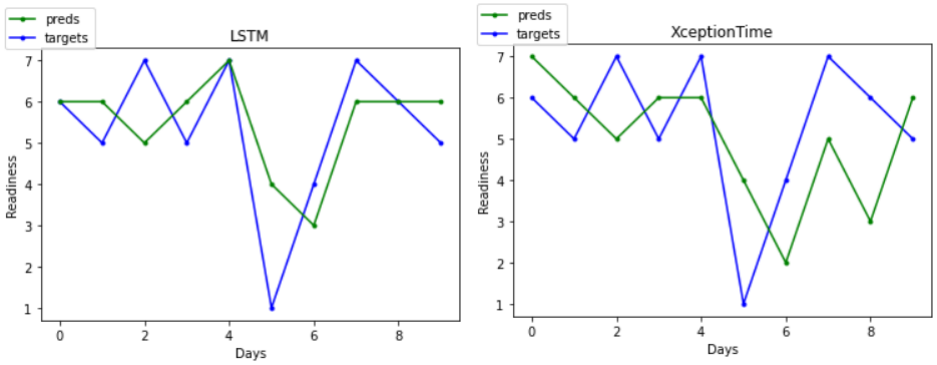

Figure 4.23: Best and worst models from mean value replacement on univariate training

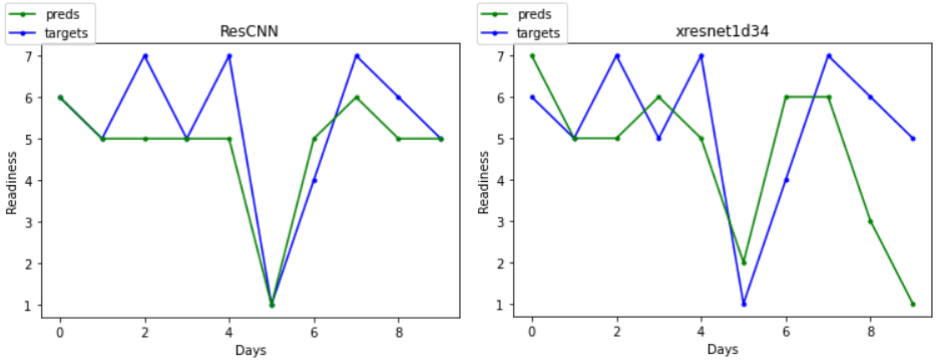

Figure 4.24: Best and worst models from zero value replacement on univariate training

NaN handling	Window	All	One	Uni
Interpolate	**1**	4.15	5.91	4.67
	3	3.79	4.59	5.41
Mean	**1**	3.63	4.21	3.30
	3	3.23	5.07	4.79
Zero	**1**	3.70	7.08	5.80
	3	2.95	9.53	8.50

Table 4.9: Table shows results from **LSTM** training on three datasets, training on all players(all), training on one players self reported data(one) and training on the univariate dataset(Uni). Training was done on the three different NaN handling methods, **Interpolation**, **Mean value** and **Zero values**.

compared to Tsai, running 500 epochs 10 times for each model with Keras takes less time than runnning all models on 50-100 epochs with Tsai. For updated experimentation with Keras we use the same configurations of the datasets as before; training on all players, training on one players self reported data and an univariate dataset on readiness levels. We will take a look at the results on training on all players, training on one player's self report data and univariate training. In addition to this, we will test what works best of window length 1 or 3 days.

4.5.2 Multivariate training on all players

Interpolate

For interpolate RNN saw the best results out of the three models, with a prediction window of 3 days it got an average MSPE score of 3.16 through the 10 runs we did. All three models was close in score fluctuating with under 1 score between all results from interpolation.

Mean

Going with 3 as prediction window saw the best overall results amongst all three models tested with mean value NaN handling on this dataset. 3 as prediction window gave a average MSPE score of 3.29 while going with 1 gave 3.74, so not that much of a difference but still better. When it came

NaN handling	Window	All	One	Uni
Interpolate	**1**	3.49	6.59	4.79
	3	3.50	5.63	5.12
Mean	**1**	3.94	3.90	3.30
	3	3.44	5.20	4.79
Zero	**1**	3.70	6.59	5.68
	3	3.47	9.49	8.16

Table 4.10: Table shows results from **GRU** training on three datasets, training on all players(all), training on one players self reported data(one) and training on the univariate dataset(Uni). Training was done on the three different NaN handling methods, **Interpolation**, **Mean value** and **Zero values**.

NaN handling	Window	All	One	Uni
Interpolate	**1**	3.50	5.38	4.77
	3	3.16	5.81	5.41
Mean	**1**	3.65	4.19	4.78
	3	3.20	5.04	4.77
Zero	**1**	3.73	6.00	5.88
	3	2.94	9.85	8.36

Table 4.11: Table shows results from **RNN** training on three datasets, training on all players(all), training on one players self reported data(one) and training on the univariate dataset(Uni). Training was done on the three different NaN handling methods, **Interpolation**, **Mean value** and **Zero values**.

NaN handling	Model	MSPE
Interpolated	RNN	3.16
Mean	RNN	3.20
Zero	RNN	2.94

Table 4.12: Table shows the best models for multivariate training on all players with Keras.

to which models did the best, both RNN and LSTM did better than GRU. LSTM and RNN got the same score oddly enough with 10 runs each and GRU got 0.3 worse score.

Zero

Replacing with zero values on a window of 3 saw the best results from all three NaN handling approaches in this training method. With an average score of 3.12 compared too 3.71 for 1 day window this results was pretty good. LSTM performed the best cutting close with RNN by only 0.01 in average difference.

All in all when looking over the results gotten from multivariate training on all players, the Zero value NaN handling method saw the best results. Having a window size of 3 days was clearly working the best for all the three models when training on all players scoring about 0.4 points better than with a window size of 1. RNN gave the best results as seen in table 4.12. RNN and LSTM gave the best MSPE score on the zero value dataset with 0.01 points difference between the two.

4.5.3 Multivariate training on one player's self report data

Interpolate

Interpolated NaN handling combined with training on one player's self reported data saw the worst results out off the three interpolation tests we did with Keras. With an average MSPE score from all 60 tests on 5.6. 3 days window came in a slight lead by a 0.5 difference compared to a 1 day window. As for the best model, LSTM performed the best.

NaN handling	Model	MSPE
Interpolated	LSTM	4.59
Mean	GRU	3.90
Zero	RNN	6

Table 4.13: Table shows the best models for multivariate training on one player with Keras.

Mean

Mean NaN handling on this training method did also see the worst results so far, averaging at 4,6 overall. This time having a 1 day window saw the best results at 4.1 MSPE score compared to 5.1 at 3 day window. For the models there was a slight deviation at around 0.1 favoring GRU.

Zero

Zero value replacement gave the worst results from this entire intermediate experimentation with Keras. Even though the results were better for a 1 day window the MSPE score was on average 6.55 across all models. For the tests on 3 day window the average score was 9.6, the highest seen from this experiment. RNN seemed to be working the best with Zero value NaN handling, but not far away from GRU 0.1 away.

Multivariate training on one player's self report data showed the best results with GRU on the mean value replacement dataset. See table 4.13 for the summary of the best models based on NaN handling. LSTM did the best for interpolated NaN handling, and RNN performed over the rest of the models on zero value NaN handling.

4.5.4 Univariate training

Interpolate

When it came to Univariate training, window size of 1 had the best results. With interpolation we saw a 4.74 on a 1 day window compared to 5.31 with a 3 day window. The models performed close to the same when it comes to the difference in results within the same window size. GRU saw the better results with 0.1 score gap between LSTM.

NaN handling	Model	MSPE
Interpolated	LSTM	4.67
Mean	GRU/LSTM	3.30
Zero	GRU	5.68

Table 4.14: Table shows the best models for univariate training with Keras.

Mean

3.79 is the best average score across models to be seen on univariate training and was done with mean value NaN replacement with a 1 day window. Average score acorss models on a 3 day window was 4.7 a whole point away from 1 day window. Again we saw similar results between the three models this time a split 1st place between LSTM and GRU with RNN 0.7 MSPE away.

Zero

Once more we see zero value replacement coming with a high MSPE score. Having the best score at a 1 day window size with the score of 5.78 compared to 8.34 with a 3 day window. Here we see GRU getting the best performance only by 0.1.

For univariate training both GRU and LSTM got the same MSPE score of 3.30, table 4.14 for reference. Mean NaN handling with GRU and LSTM got the best results from this training method doing over 1 MSPE over interpolated NaN handling coming in at second best. Zero value Nan handlin saw the worst results with GRU at 5.68 MSPE.

4.6 Discussion of imputation results

In this section of experiments we have taken a look at how replacing NaN with different values looked like all while predicting on the same test set. For Tsai we experimented with 14 different models and for Keras we experimentet with 3. We executed these experiments with the intention of predicting readiness to play based on the data received from self reporting in PMSys. Since this data had missing values on some days we modified the data to replace NaNs with zeros, the mean value of the entire dataset

and interpolated values calculated from the values closest. The test set was a series of readiness values without missing numbers so the test set has not been touched and is unseen and unbiased. When it comes to performance from Tsai compared to performance from Keras, Tsai has seen some better results from some of the 14 models we tested.

For Keras, the best results we got was from training on all players' readiness score with RNN and a window size of 3. This combination got a MSPE score 2.94. Coming close with only 0.01 score higher was training on all players with LSTM with a MSPE of 2.95. Window size in Keras favored having 1 day as window when training with one player, either it was multivariate training on one player or if it was univariate training on one player. For training on all players having a window size of 3 saw the best results overall. In addition to this, training with all players in general saw better results with Keras with an MSPE within 2.95 -:- 4.15 for LSTM, 3.47 -:- 3.94 for GRU and 2.94 -:- 3.73 for RNN.

When it comes to Tsai, as mentioned this library generally saw the best results. The best combination of training method and NaN handling method was univariate training with zero values replacing NaNs. Having a window size of 3 gave the models the best MSPE score seen through all of experimenting. We saw ResCNN on top with an MSPE score of 1.1 following close is InceptionTime with 1.3 in score, ResNet and XceptionTime with 1.6. The next best combination was training on one player's self reported data with mean values as NaN replacements. This proved to be a good fit with a window size of 3 and had the best MSPE score of 1.6 on LSTM with 2 layers. The scond best model from this set was ResNet once again coming with a low score of 1.8. Bidirectional LSTM with 2 and 3 layers saw some great results as well with a score respectively of 1.8 and 2.0. When training on all players, replacing NaNs with interpolated values gave the best results.

Something to note when discussing the results of experimentation on the different ways of handling NaNs is how the athletes values might look like from day to day. Since we saw the best results on the datasets where we replaced NaNs with zero, there need to be some explanation for this. The way we rate our model's performance, MSE focuses on the big mistakes rather than the small ones. This can tie in how the results came to be what they are. With what we know about how MSE is calculated we can combine that with what the athletes report. Normal readiness is typically on the upper side on the scale as we can see in figure 3.2 where the mean value is 6. Players are more often than not ready to train or ready to play

compared to not ready at all. This will in turn make the average predicted readiness value along that same average value. When the models predict they are trained on the prior data and for the mean value and interpolation value this typically is not 0. This might make the models trained on these two datasets more prone to predict a value higher on average. For the datasets with zero as replacement for NaNs they will be trained on having observing lower values more often. Then when the actual value to be predicted is a low value this favours the models trained on zero value replacement. Again looking at how MSE is computed the bigger misses are weighted heavier than small misses, so this will make the models not accurately predicting dips in readiness have a lower MSE. This works for the opposite also when the readiness values are peaking with a rapid change from for example 3 to 9 in readiness. Mean value and interpolation datasets might already be prone to predict higher so they might be hitting more often the tops than models trained on zero value replacement. At the same time, models trained on zero value replacement are trained on volatile data so they might predict somewhat when values peak in such a high value. Looking at the graphs in figures 4.16-4.24 we can visually see how the different NaN handling methods tends to behave with the zero replacements often predicting lower values than the rest of the methods. Although classifying the behaviour of the three NaN handling methods is hard when there is a element of randomness in machine learning and from having just this small sample of graphs, we can not draw any strong conclusions on behaviour.

4.7 Experiment with univariate multi-step prediction

4.7.1 Overview

Teams might want to predict X amount of days in to the future in order to see how their athletes might develop in readiness to play. We decided to use the knowledge gathered up to this point in order to see how well some of the models might predict into the future. This type of prediction is called multi-step forecasting and when forecasting x amount of days into the future we do not use any actual observations for predicting the next days. After the first day the models will recursively use the already predicted values to forecast the x amount of days we want. When thinking logically about how many days a team might want to predict, we thought about testing for a couple of days to see the performance of the models. Teams might want to predict how the week goes from for example a

Monday. Then the model will use the last reported data available in this case the Monday and predict how the readiness might go the next 7 days. When forecasting multi-steps the performance depends on the window size/timesteps used. We decided to experiment with 3 and 5 days as window sizes and predict the next 7 days recursively using predicted values. Here is an overview of what we will be using when multi-step forecasting:

- Univariate training on readiness
- Test set of max 10 days configured the same as in prior experiments
- Forecasting the next 7 days

This experiment will also have some variables we will compare the results of in order to find what performs the best:

- Timesteps/Window size: 3 and 5
- NaN handling with interpolated values
- NaN handling with mean values
- NaN handling with zero values

For these variables we will set them up with the three different models used with Keras in prior experimentation:

- LSTM
- GRU
- SimpleRNN

4.7.2 NaN handling with interpolated values

In table 4.15 we see the results of multi-step forecasting on the interpolated dataset. We tested with a window size of 3 and 5 days and from for all three models having a window size of 3 gave the best results. Figure 4.25 shows us the difference in the predicted versus actual value of the models tested on the interpolated dataset. Top left we have LSTM which got an MSPE score of 4.11 in this graph, showing how it can predict the trends in a very subtle way but still predicting the trends nonetheless. Top right we

Model	Window	MSPE
LSTM	**3**	4.11
	5	5.20
GRU	**3**	3.79
	5	4.61
SimpleRNN	**3**	4.03
	5	4.24

Table 4.15: Table consists of the MSPE scores gotten from training on univariate time series with interpolated values for NaN replacement, divided into the three models ran on multi-step forecasting

have GRU which got an MSPE score of 3.79 in this graph. The predicted values are seen closer to the actual values and as we discussed earlier this will give a better MSE score. GRU was not able to predict the trends in the graph as well as LSTM but still got the better score. Bottom right we see RNN with a score of 4.03. RNN seems to be able to follow the trends a bit better than GRU but not as much as LSTM.

4.7.3 NaN handling with mean values

For the results on training with mean values as NaN replacements we saw overall worse results compared to training with interpolated data. Looking at the results in table 4.16 we can see the two models LSTM and GRU on a window size of 3 performed worse when comparing MSPE score to the last test with interpolated data. SimpleRNN on the other hand saw some slight improvements with a score of 3.85 on window size 3. All three models seems to be predicting lower values on mean value imputation compared to interpolation. Once again we see LSTM predicting in a more volatile fashion with a higher gap between the predicted values following the movements of the test set. When it comes to the performance of having a 5 day window the score was a bit worse than with a window size of 3. About a 0.5 difference in score in general comparing 3 days to 5 days.

Model	Window	MSPE
LSTM	**3**	4.35
	5	4.70
GRU	**3**	4.63
	5	5.06
SimpleRNN	**3**	3.85
	5	4.34

Table 4.16: Table consists of the MSPE scores gotten from training on univariate time series with mean values for NaN replacement, divided into the three models ran on multi-step forecasting

Model	Window	MSPE
LSTM	**3**	4.76
	5	11.06
GRU	**3**	5.09
	5	10.43
SimpleRNN	**3**	5.18
	5	9.17

Table 4.17: Table consists of the MSPE scores gotten from training on univariate time series with zero values for NaN replacement, divided into the three models ran on multi-step forecasting

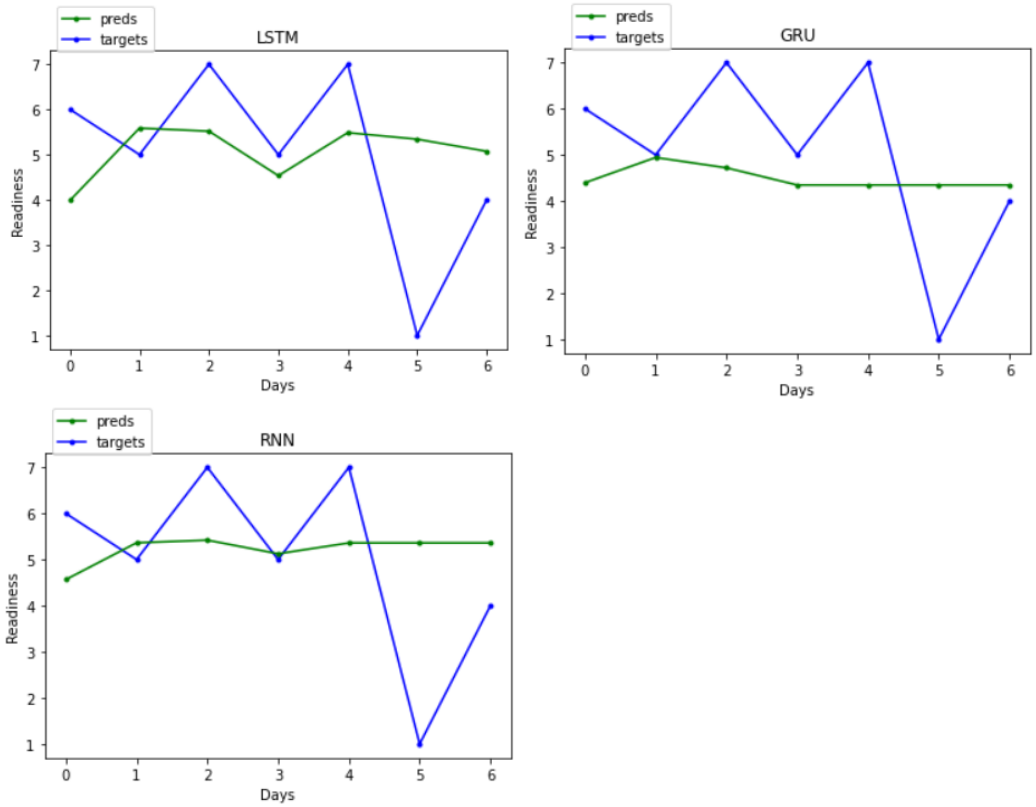

Figure 4.25: LSTM, GRU and RNN graphs from multi-step forecasting on interpolated dataset with a window size of 3 days

4.7.4 NaN handling with zero values

Replacing NaNs with zero values in this experiment with multi-step forecasting saw the worst results out off all three NaN handling methods. As seen in table 4.17 we now saw MSPE scores as high as 11.06 on LSTM with a window size of 5. It is clear that having a window size of 5 did not do any good for the MSPE scores with zero values as NaN replacements. Having a window size of 5 days saw on average 5,21 increase in MSPE score when calculating the average of all three models. The best results was LSTM with a window size of 3 getting a score of 4.76. Looking at the scores for all models we see that every score gotten is the worst out of the three ways of handling NaNs we so far have tested with multi-step forecasting. When looking at the graphs generated from the results, we can see how all three models have predicted a much lower value than the target value of that day. This is what makes the MSPE so bad for this NaN handling. If we focuse on how well the models could have potentially followed the flow of target values in the graph we see that LSTM is following the closest. It is somewhat catching the peak of day 2 followed by a small drop to day 3 and a slight increase from day 3 to

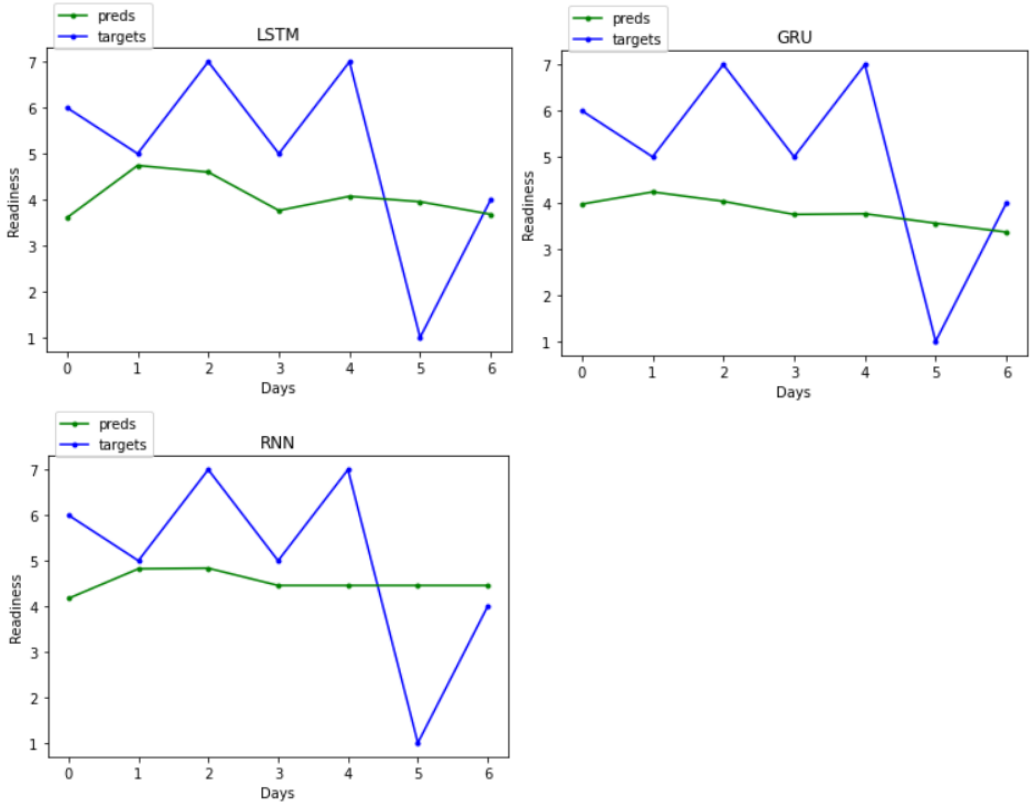

Figure 4.26: LSTM, GRU and RNN graphs from multi-step forecasting on mean value dataset with a window size of 3 days

day 4 once again. After day 4 the models prediction do not vary that much anymore. This is because the predicted values from prior days all slowly go towards the same number for every prediction. When the model just keeps predicting closer and closer to the value 4 which we see in LSTM after day 4, the model will continue to predict and shove the same predictions with a smaller and smaller variation until the predictions seem to bottom out. This bottoming out seems to happen much faster in the Models of GRU and RNN compared to LSTM.

4.8 Discussion of multi-step forecasting

The intention of experimenting with multi-step forecasting was first off all to the possibilities of predicting X amount of days in to the future. From the results we achieved we can say multi-step forecasting did not see the best of accuracy. Considering this being the first implementation and the time limitation we saw when making our own version of recursive multi-step forecasting, we can say the results reflect on this. Results can certainly

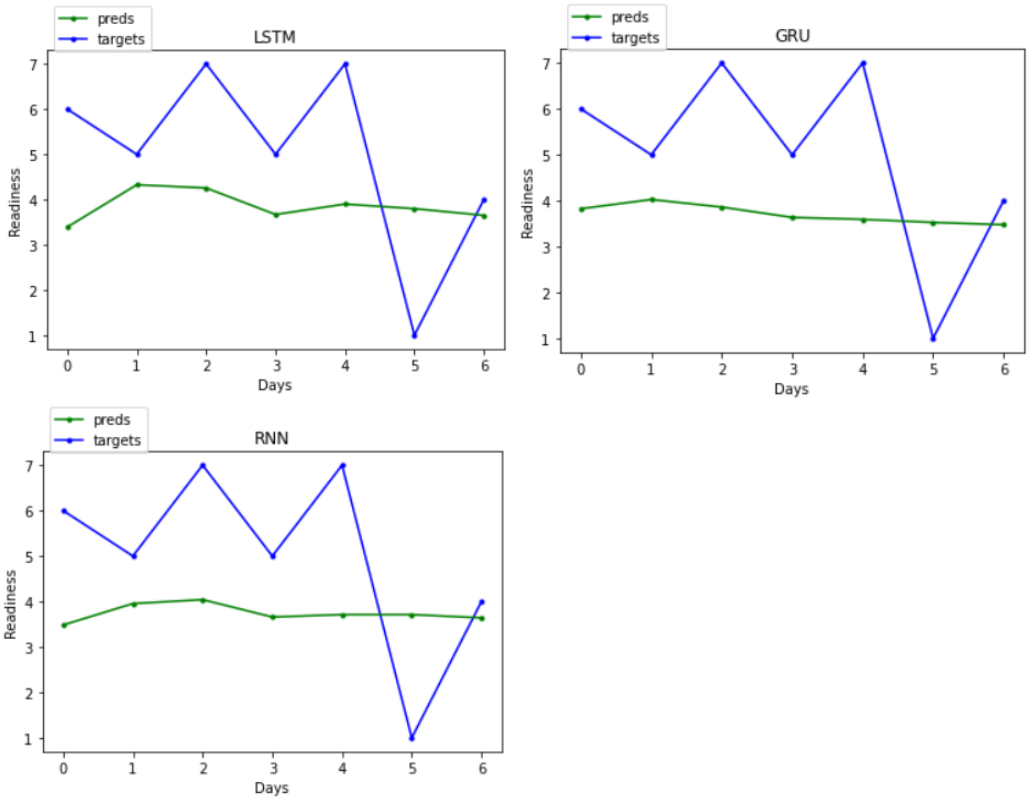

Figure 4.27: LSTM, GRU and RNN graphs from multi-step forecasting on zero value dataset with a window size of 3 days

be improved with time and energy going in to fitting the models better to the datasets and messing around with different configurations but from the time we had this results is respectable. The best model being LSTM saw some promising behaviour if we think about how the model could follow the peaks and dips seen in actual readiness values. When it came to the number of days multi-step forecasting could forecast into the future we saw a steady decrease in prediction volatility after around 4 or 5 days. All three models behaved the same when considering how the predictions tended towards one flattened prediction after 4 or 5 days. This might be caused by the volatility of the prediction decreasing over time which since the predictions are based on previous predictions will guide towards a constant value of prediction. To recap what we have learned from this experiment:

- Sliding window of 3 days generally saw better results than with 5 days
- Replacing NaNs with interpolated values showed the best results of 4.11 with LSTM, 3.79 with GRU and 4.03 with RNN.

- GRU saw the best MSPE score of 3.79 with interpolated NaN replacements and a window size of 3 days
- LSTM behaved the best considering how it more closely followed peaks and dips in readiness to play

4.9 Summary

In section 4.1 we gave an overview of what to expect from experiments. This included defining the types of NaN handling we are going to test out, the different training methods used and the different datasets made for all of these parts. We stated what libraries we will be experimenting on and how we will compare them. We also set the stakes for how we will be determining how well the different model interact with the different NaN handling methods and datasets. Everything information described in section 4.1 was used in section 4.2, the initial experimentation. In this section we stated the motivation of initial experimentation, how we want to use the information gotten in this section further down in the other sections. This section established the capacities and baseline of what we are using in the next experiments. The results gathered in section 4.2 was the different training methods using Zero values as NaN replacements. These results saw a change in the experiments done in section 4.4. Section 4.3 discussed the results gotten from initial experimentation, how the hyperparameters of the models saw changes for the better and how it the parameters were going to be used in the next section. We also discussed the results from the two different libraries used. Section 4.4 showed how the different NaN handling methods worked on the baseline of the models chosen from Tsai. Here we also saw how the results from initial experimentation had been improved by the changes suggested in the discussion section 4.3. Section 4.5 saw the results of experimenting with the different NaN handling methods on the models chosen from Keras. In section 4.6, we discussed the results from experimenting with the different imputations. We compared both how the different libraries performed and how the imputations affected the results gotten from these libraries. Section 4.7 was done to test out how well what we learned from Keras and experiments with imputations would work on multi-step forecasting. In section 4.8, we discussed the results gotten from multi-step forecasting. In the next chapter, we will draw a conclusion based on the results gotten from all of our research and we will provide some suggestions on what would make for some interesting future work.

Chapter 5

Conclusion and future work

In this thesis we aimed at comparing different approaches for imputing missing values in time series data and seeing how these approaches might affect performance in machine learning. When checking how the different imputing approaches affected machine learning performance we predicted athletes readiness to play based on the data provide by PMSys. By using two different machine learning libraries, Keras and Tsai we were able to compare how the imputation approaches affected chosen models from these libraries. When predicting readiness to play we had two different machine learning approaches, predicting the next day and multi-step forecasting along with three different training methods, training on all players, training on one player's self reported data and training on one players readiness to play data. We compared the results of both predicting the next day and multi-step forecasting on the different imputation approaches and models from the two machine learning libraries. The best results we got can be seen in figure 4.15 on the model ResCNN. This result was on imputing missing values with zero values and was done on univariate training on one player. Although this was the overall best results seen for predicting one day ahead, when multi-step forecasting we saw interpolation getting the best results with GRU as seen in table 4.15. In this thesis we only tested multi-step forecasting with Keras as the library of choice. Since Tsai got the best results from one day predictions, a proposal for future work could be to see how well Tsai would do when multi-step forecasting. In addition to this, some interesting thoughts about further research could be testing with larger window sizes when multi-step forecasting.

Bibliography

[1] Sarosij Bose and Avirup Dey. *ResCNN: An alternative implementation of Convolutional Neural Networks*. URL: https://ieeexplore-ieee-org.ezproxy.uio.no/document/9667654. 10.05.2022.

[2] *Decomposition Models*. URL: https://online.stat.psu.edu/stat510/lesson/5/5.1. 16.08.2021.

[3] IBM Cloud Education. 'Neural Networks'. In: (2020). URL: https://www.ibm.com/cloud/learn/neural-networks. 06.04.2021.

[4] IBM Cloud Education. 'Recurrent Neural Networks'. In: (2020). URL: https://www.ibm.com/cloud/learn/recurrent-neural-networks. 13.04.2021.

[5] Hassan Ismail Fawaz et al. *InceptionTime: Finding AlexNet for time series classification*. URL: https://link-springer-com.ezproxy.uio.no/article/10.1007/s10618-020-00710-y. 10.05.2022.

[6] J.R. Grove et al. 'Training distress and performance readiness: Laboratory and field validation of a brief self-report measure'. In: *Scandinavian journal of medicine and science in sports* 24 (2014), pp. 483–490. DOI: https://doi-org.ezproxy.uio.no/10.1111/sms.12214.

[7] Kaiming He, Xiangyu Zhang andShaoqing Ren and Jian Sun. *Deep Residual Learning for Image Recognition*. URL: https://ieeexplore.ieee.org/document/7780459. 10.05.2022.

[8] Bahrudin Hrnjica and Ognjen Bonacci. 'Lake Level Prediction using Feed Forward and Recurrent Neural Networks'. In: *Water Resources Management* (May 2019), pp. 1–14. DOI: 10.1007/s11269-019-02255-2.

[9] Rupesh K., Klaus Greff and Jürgen Schmidhuber. *Training Very Deep Networks*. URL: https://proceedings.neurips.cc/paper/2015/hash/215a71a12769b056c3c32e7299f1c5ed-Abstract.html. 10.05.2022.

[10] *Keras Homepage*. URL: https://keras.io/. 22.09.2021.

[11] *Linear interpolation and extrapolation with calculator*. URL: https://x-engineer.org/linear-interpolation-extrapolation-calculator/. 05.03.2022.

[12] *Multistep ahead forecast feature*. URL: https://github.com/sassoftware/python-dlpy/issues/101. 06.05.2022.

[13] Michael Phi. *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. URL: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. 10.05.2022.

[14] Elahe Rahimian et al. *XceptionTime: A NOVEL DEEP ARCHITECTURE BASED ON DEPTHWISE SEPARABLE CONVOLUTIONS FOR HAND GESTURE CLASSIFICATION*. URL: https://arxiv.org/abs/1911.03803. 10.05.2022.

[15] Evan Shelhamer, Jonathan Long and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. URL: https://arxiv.org/abs/1605.06211v1. 10.05.2022.

[16] Manik Soni. *Understanding architecture of LSTM cell from scratch with code*. URL: https://medium.com/hackernoon/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4. 15.04.2021.

[17] *Tsai documentation*. URL: https://timeseriesai.github.io/tsai/. 03.03.2022.

[18] Li-Pen Wang et al. 'An enhanced blend of SVM and Cascade methods for short-term rainfall forecasting'. In: (Sept. 2011).

[19] *What is linear interpolation - Definition and Meaning*. URL: https://www.easycalculation.com/maths-dictionary/linear_interpolation.html. 05.03.2022.

[20] Theodor Wiik et al. *Predicting Peek Readiness-to-Train of Soccer Players Using Long Short-Term Memory Recurrent Neural Network*. URL: http://home.simula.no/~paalh/publications/files/cbmi2019-PMSYS.pdf. accessed: 10.02.2021.