Exploration of time-series models on time series data

A case study on athlete monitoring

Siarhei Kulakou

Thesis submitted for the degree of
Master in Programming and Networks
30 credits

Department of Informatics
The Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Autumn 2021

Exploration of time-series models on time series data

A case study on athlete monitoring

Siarhei Kulakou

Exploration of time-series models on time series data

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

Printed: Reprosentralen, University of Oslo

Abstract

In this thesis, we tried to use machine learning capabilities to process time-series data. We tried to find more modern implemented algorithms applicable to solve regression problems for the univariate and multivariate time series. Having compared and selected the found algorithms, we carried out experiments and compared our results. The experiments were carried out in several stages since their primary purpose was to analyze the results and adjust the parameters to obtain better results. In parallel, we conducted tests to measure the performance of our models to compare this performance with the prediction accuracy and select the best models for further training. After our experiments, we discussed the results in terms of precision and resources consumed. In the end, we tried to draw conclusions from work done and provide ideas for further research. We have tried to structure this thesis so that it is easy for the reader to go through each chapter and analyze the information. In the initial chapters, we provide theoretical knowledge regarding the research, and in the subsequent chapters, we give an account of the experiments done.

Contents

Abstract **i**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Problem statement . . . 2
1.3 Outline . . . 3

2 Background and Related work **4**
2.1 PMSys . . . 4
2.1.1 Introduction . . . 4
2.1.2 Architecture . . . 5
2.2 Data collection in PMSys . . . 7
2.3 Short introduction to time series . . . 9
2.4 Machine learning . . . 11
2.4.1 Introduction . . . 11
2.4.2 Models appropriate for time series processing . . . 12
2.5 Previous Machine Learning Research related to PMSys . . . 13
2.6 Brief description of 'Tsai' library . . . 16
2.7 Summary . . . 17

3 Methodology **18**
3.1 System specification . . . 18
3.2 Machine learning models chosen for experiments . . . 18
3.3 Data set . . . 24
3.3.1 Overview of use cases . . . 24
3.3.2 Data set for prediction of 'Readiness' to play using univariate time series . . . 26
3.3.3 Data set for prediction of 'Readiness' to play using multivariate time series . . . 27
3.3.4 Description of splitting the data set for one player into 'training', 'validation' and 'test' sets . . . 27
3.3.5 Detailed explanation of considered 'test' data sets for different cases . . . 28
3.4 Metrics . . . 29
3.4.1 Machine learning metrics . . . 29
3.4.2 Metrics for performance measurement . . . 30
3.5 Summary . . . 31

4 Experiments and results **33**
4.1 Experiments overview 33
4.2 Initial experiments 34
4.2.1 Overview 34
4.2.2 Results for the case with univariate time series 35
4.2.3 Results for the case with multivariate time series . . . 36
4.3 Experiments with corrected number of epochs 39
4.3.1 Overview 39
4.3.2 Results for the case with univariate time series 40
4.3.3 Results for the case with multivariate time series . . . 41
4.4 Discussion of the intermediate results 42
4.5 Experiments with the best models and one player's data without gaps filled with '0' 43
4.5.1 Overview 43
4.5.2 Description of the results for the cases with uni- and multivariate time series 44
4.6 Experiments with the best models and data for all players with / without '0' 46
4.6.1 Overview 46
4.6.2 Description of the results 46
4.6.3 Description of the results 49
4.7 Results of performance measurement 53
4.8 Discussions 55
4.9 Summary 56

5 Conclusion and future work **58**

A Tables **61**
A.1 Tables from the experiments 61

B Figures **64**
B.1 Figures from the experiments 64

List of Figures

2.1 Example of entering wellness data into the PMSys smartphone application [4]. 6
2.2 Overview of the architecture of the PMSys data storage system [4]. 7
2.3 Example of time series that shows values of the parameter 'Readiness' to play for one player. The plot was made during the experiments to this thesis. 9
2.4 Example of univariate time series. Values of parameter 'Readiness' from PMSys data. 10
2.5 Example of multivariate time series. Values of parameter 'Readiness', 'Mood', 'Stress', 'Soreness' and 'Fatigue' from PMSys data. 10
2.6 Example of dependency between Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL). 11
2.7 Actual data and predictions for a random player from team 1 based on the model trained on the specific player in the team 1 [4]. 14
2.8 Actual data and predictions for a random player from team 1 based on the model trained on all players in the team 1 [4]. 15
2.9 Actual data and predictions for a random player from team 2 based on the model trained on the specific player in the team 2 [4]. 15
2.10 Actual data and predictions for a random player from team 2 based on the model trained on all players in the team 2 [4]. 15

3.1 Inside the Inception module for time series classification. For simplicity a bottleneck layer of size $m = 1$ is illustrated. This picture was taken from this paper [15]. 20
3.2 RNN example. Source https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359. 21
3.3 LSTM example. Figure and description to it was taken from https://colah.github.io/posts/2015-08-Understanding-LSTMs/. 22
3.4 Illustration of gated recurrent units. r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation. 23
3.5 RNN-FCN, LSTM-FCN and GRU-FCN architecture. 24
3.6 MRNN-FCN, MLSTM-FCN and MGRU-FCN architecture. . 25

3.7 Diagram describes predicting 'Readiness' to play using univariate and multivariate time series. 25
3.8 Parameter 'Readiness' for one player. Values ranged between 0 and 10. There are 914 data units in total. 27
3.9 Parameters 'Readiness', 'Mood', 'Stress', 'Soreness', 'Fatigue' for one player used as multivariate time series. Values of 'Mood', 'Stress', 'Soreness', 'Fatigue' are ranged between 0 and 5, of 'Readiness' between 0 and 10. 28
3.10 General example of splitting the data for one player into training and validation sets. 28

4.1 Scheme for initial experiments. 34
4.2 Results of the initial experiments. Best models. Window length 3. Univariate time series for one player. Test data without '0'. 37
4.3 Results of the initial experiments. Best models. Window length 3. Multivariate time series for one player. Test data without '0'. 38
4.4 Scheme for experiments with corrected number of epochs. . 39
4.5 Results for GRU. Window length 3. Univariate time series for one player after correction of number of epochs. Test data without '0'. New MSE = 1.350 40
4.6 Results after corrected number of epochs. Best models. Window length 3. Multivariate time series for one player. Test data without '0'. 42
4.7 Scheme for experiments with the best models and one player's data without gaps filled with '0'. 43
4.8 Scheme for experiments with the best models and data for all players with / without gaps filled with '0'. 46
4.9 Results for the 'all but one player' case. Best models. Window length 3. Univariate time series. Data with gaps filled with '0'. 48
4.10 Results for the 'all but one player case'. Best models. Window length 3. Multivariate time series. Data with gaps filled with '0'. 49
4.11 Results for the 'all but one player' case. Best models. Window length 3. Univariate time series. Data without gaps filled with '0'. 51
4.12 Results for the 'all but one player' case. Best models. Window length 3. Multivariate time series. Data without gaps filled with '0'. 52

B.1 CPU, memory and disk usage. Univariate time series. Second 5 best models. All but one player case. Data without gaps/'0'. 65
B.2 CPU, memory and disk usage. Univariate time series. Second 5 best models. All but one player case. Data without gaps/'0'. 66

B.3 CPU, memory and disk usage. Best models. All but one player case. Multivariate time series. Data without gaps/'0'. 67

List of Tables

2.1 Example of the gradation of the parameters 'mood', 'stress', 'soreness', 'fatigue', 'sleep quality'. 8

3.1 System specifications. 19
3.2 Default configuration of the InceptionTime model. 21
3.3 Default configuration of the RNN, LSTM and GRU models. 23
3.4 Default configuration of the RNN-FCNPlus, LSTM-FCNPlus, GRU-FCNPlus, MRNN-FCNPlus, MLSTM-FCNPlus and MGRU-FCNPlus models. 26

4.1 Configurations for initial experiments. Univariate time series. 35
4.2 Models with their MSE values after initial experiments. Sliding window size 3. Univariate time series. 36
4.3 Best models after initial experiments. Multivariate time series. 38
4.4 Best models after corrected number of epochs. Univariate time series. 40
4.5 Best models after corrected number of epochs. Multivariate time series. 41
4.6 Updated configurations, were used for the experiments with the best models from tables 4.4 and 4.5. 44
4.7 Best models with MSE values. Data for one player without gaps. Uniivariate time series. 45
4.8 Best models with MSE values. Data for one player without gaps. Multivariate time series. 45
4.9 Best models with MSE values. Training on all but one and testing on the one. Data with gaps. Univariate time series. . 47
4.10 Best models with MSE values. Training on all but one and testing on the one. Data with gaps. Multivariate time series. 47
4.11 Best models with MSE values. Training on all but one and testing on the one. Data without gaps. Univariate time series. 50
4.12 Best models with MSE values. Training on all but one and testing on the one. Data without gaps. Multivariate time series. 52
4.13 Configurations used for performance measurement. 53
4.14 Performance measurement. Best models. Multivariate time series. 'Training on all but one and testing on the one' case. . 53
4.15 Performance measurement. Best models. Univariate time series. 'Training on all but one and testing on the one' case. . 54

A.1 Results of the 'MSE' metric from the experiments for all window sizes for univariate data set. Test data without/with '0' (upper/lower values in the table cells correspondingly). . 62
A.2 Results of the 'MSE' metric from the experiments for all window sizes for multivariate data set. Test data without/with '0' (upper/lower values in the table cells correspondingly). . 63

Chapter 1

Introduction

Machine learning and artificial intelligence have become technological breakthroughs. Face substitution and aging applications, voice assistants can reserve a table or book a ticket, solutions that recognize atrial fibrillation and heart attack, and that's just what happened recently in the past few decades.

The popularity of AI technologies is growing, which means that their demand is also increasing. This popularity leads to an increase in the entire developer community and the appearance of modern AI libraries that make learning and work easier in various fields of science.

1.1 Motivation

Nowadays, professional sports is not just sports, but a form of art, where the achieved results combine not only the individual physical abilities of athletes but also the most modern technologies in such areas as medicine, equipment production, nutrition, and physical and mental health monitoring. Proper diet, rest, and training regimens, selected by specialists for each athlete, monitoring and making the appropriate conclusion of his condition at a certain point in time can lead him or his team to a victory depending on whether it is an individual competition or a team sport.

It is not a secret that a professional athlete must constantly adhere to a strict nutrition plan, training process, and rest regime so that the athlete's body should be in the required state at some particular moment. The athlete's condition is influenced not only by the amount of consumed and burned calories, the duration and intensity of the training process but also by parameters such as the duration and quality of sleep and the general mental state or some others.

It turns out that if it would be possible to try to compile a set of parameters describing the general state of the athlete's body at a certain point of time and collects data belonging to these parameters, then it would be possible to try to predict the state/behavior of the athlete's body in the near future.

This idea was taken into consideration by Simula Research Laboratory,

University of Tromsø, and ForzaSys. The first aim was to create a player monitoring system (PMSys) for collecting and storing information about the athlete's condition based on a list of particular parameters.

Also, there is the fact that computer technology is constantly being improved. This improvement leads to increased computing power and presents the ability to process more significant amounts of data, and pushes for more sophisticated but more efficient machine learning models to be developed. So, the further intent was to try to expand this application to use modern deep learning algorithms to predict the athlete's well-being, readiness for the upcoming competition, and the occurrence of possible injuries.

1.2 Problem statement

As discussed earlier in this chapter, Simula Research Laboratory has worked with other partners to develop an athlete monitoring system. This system was put into operation and is currently being actively used to collect and store data on the status of each player individually of football teams from the Norwegian championship and the Norwegian national football team. The data registered in PMSys application at different points in time includes the measurement time, so one can easily refer it to a definition of a time series. In addition to collecting and storing information, no modern data processing method, which is based on machine learning, has yet been introduced into this system. Also, not much research has been done to find any suitable algorithms. Based on these facts, it was decided to set the following overall aim for research in this work:

> ***'Research and compare various machine learning models for processing time-series data. Predict readiness to play using the compared models and the collected data from PMSys. Evaluate the results, and propose the best model for further research/implementation in PMSys.'***

For a deeper understanding of the central task, it can be broken down into smaller ones:

1. explore the various time-series models on the data from PMSys,
2. compare in terms of accuracy and resources consumed the models' terms of time used and accuracy,
3. give an interpretation of the results, both in terms of machine learning and PMSys (sports) data analysis.

It is hoped that the work done in this thesis will bring good, objective, and structured results that can be applied in the further development of PMSys, or that these results can become a good starting point for other studies of this problem.

1.3 Outline

This thesis is divided into five main chapters. In the first chapter, we describe the motivation for this job and state the problem for research. In Chapter 2, we will give a short overview of the player monitoring system, its purpose, architecture, how data is collected, and what that data is. Also, in this chapter, we will make a short introduction to the concept of time series and machine learning. At the end of this chapter, we'll take a quick look at the 'Tsai' machine learning library. In Chapter 3, we will provide data about the specification of our system, discuss and select the machine learning models of interest to us for research in this thesis. We will also detail the dataset that will be used for our experiments. At the end of this chapter, we list the main metrics for assessing the prediction accuracy of our models, as well as metrics for measuring the performance of our models. In Chapter 4, we will provide a sequential description of all experiments, providing prediction results and system performance. At the end of the chapter, we will discuss the results obtained. In Chapter 5, we will summarize the work done in this thesis and suggest ideas for future work.

Chapter 2

Background and Related work

This chapter will give a brief presentation of PMSys, its main aim, users, and architecture. Also, it will describe the data collection process with detailed descriptions of the meaning of gradation for each registered parameter. Since all registrations include timestamps, the collected data can be classified as time series; therefore, we will briefly present what time series is. At the end of the chapter, we will briefly describe the Deep Learning research related to PMSys, which was done previously.

2.1 PMSys

2.1.1 Introduction

PMSys is a digital monitoring system that was developed for collection, storage, analysis, and visualization of athletes' health data [1]. The intention for creating this system was to replace the manual method of collecting information, and its storage on paper, with a digital one. According to the idea of the creators of this system, its primary users should be sportsmen (football players), coaches, and medical personnel [1].

One assumed that the primary tool for information registration should be a questionnaire that the players in a mobile application could easily and quickly fill out [2]. This approach allows athletes to manage their own time when filling out the questionnaire without the coach's insistent control of this process.

Another intention was to ensure that the data from the questionnaire should be stored in digital form and be accessible by the coach and medical staff [1]. This accessibility would help to improve communication between all users. Since the collected information should be analyzed, another purpose in developing this application was to replace the old-fashioned methods of analyzing data stored on paper with more modern digital ones for data processing and analysis [3].

It was one more aim in further developing the system in addition to the implemented opportunity of assessing the current state of an athlete using actual data. Creators considered the possibility of expanding functionality to introduce modern deep learning algorithms. These algorithms would

help predict such parameters as readiness to play, possible injuries, and the health status of athletes in the future. These predictions could play the primary role in adjusting the training plan or deciding on participation in competitions.

2.1.2 Architecture

In this subsection, we will present a brief description of PMSys architecture. Since PMSys supplied the data for processing in this thesis, it would be a good idea to familiarize readers with the system's architecture. The main units it consists of are [4]:

- PMSys mobile app,
- the Open mHealth compliant Data Storage Unit (DSU),
- Policy Server Unit (PSU),
- PMSys trainer.

The PMSys mobile app [5] is a modern application for smartphones. It contains a real-time questionnaire [2], consisting of a sequence of various input forms. Each athlete sequentially fills these forms, and the collected data is sent to the DSU for further storage. This mobile application is currently available for iOS and Android systems to make data collection for players more OS independent. An example of the questionnaire one can find in figure 2.1.

DSU is settled on the Amazon AWS cloud service and is designed to store and share this data from athletes for further analysis. DSU is a cluster of servers, and player's reports can be distributed between different servers depending on the level of isolation or replication required [4]. Each athlete is assigned a unique id, which allows data to be stored on servers anonymously.

PSU is a component taking care of policies that can include coach/trainer access, aggregation functions, time limited access. The PSU is managed centrally by the data owner. Access to the data on the DSU is limited, and obtaining it requires profile information from the PSU to obtain an identity, allowing policies for sharing pseudonymous player data in real-time to aggregate functions and deep learning [4].

Also, based on the fact that when the PMSys was in development, it was provided such kind of functionality that it should be responsible for processing and analyzing the collected data. And for this purpose was developed the PMSys web application called PMSys trainer. This application made it possible to get away from processing and analyzing data using the old-fashioned 'pen and paper' technology to make this process more efficient and less time-consuming. Currently, PMSys trainer can receive data from the server, visualize it graphically using various diagrams, which makes the analysis much faster and more convenient [3]. Also, during the development of this service, it was provided the possibility of further expanding the functionality in data analysis. One of

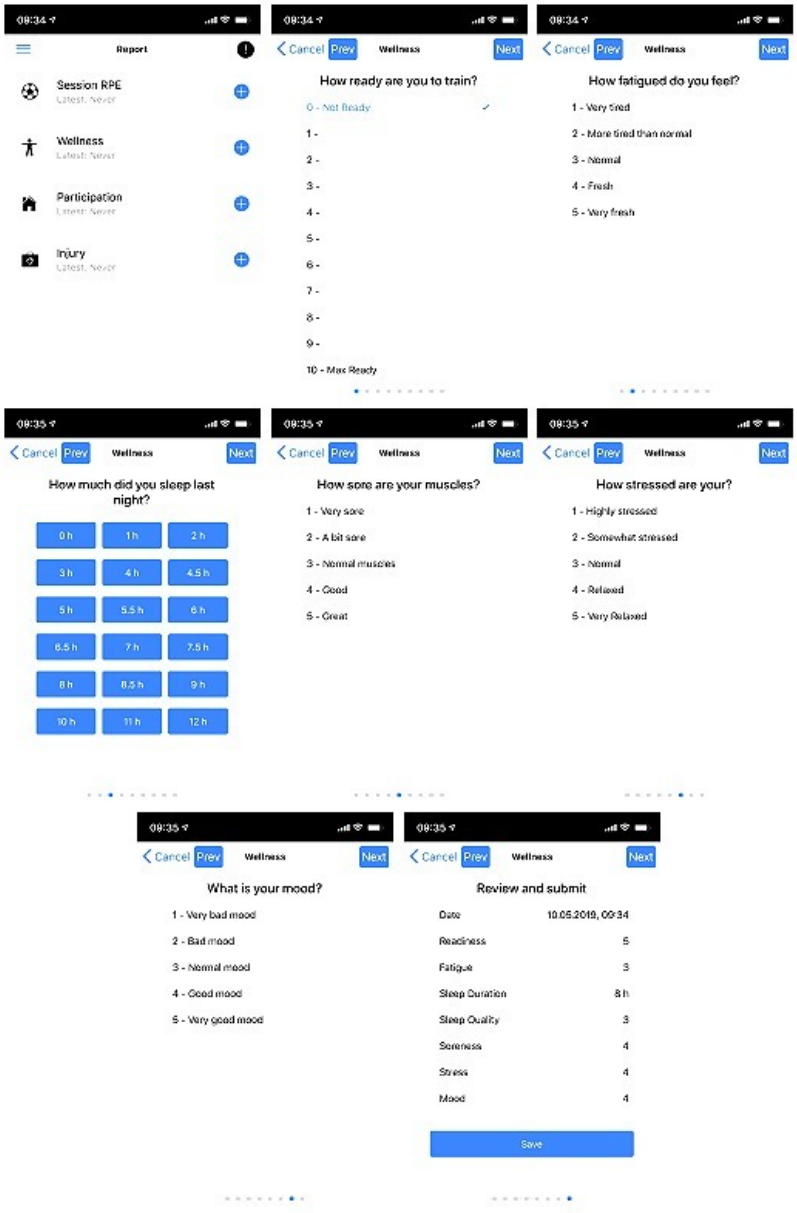


Figure 2.1: Example of entering wellness data into the PMSys smartphone application [4].

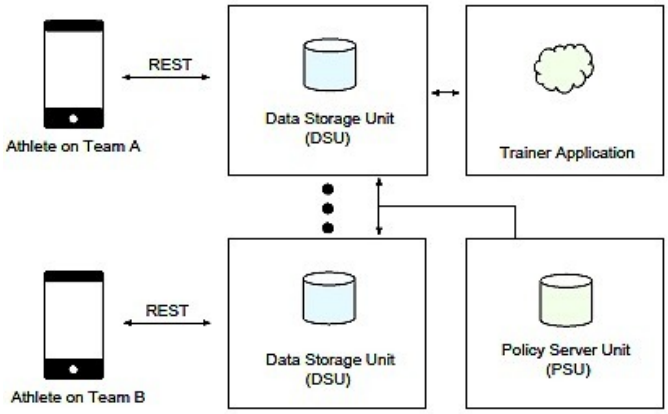


Figure 2.2: Overview of the architecture of the PMSys data storage system [4].

the proposals was the possible usage of modern deep learning algorithms, which could become helpful for the prediction of the state of athletes in the nearest future, based on the collected and processed data from the past.

2.2 Data collection in PMSys

Players used questionnaires for data registration. The data was collected and saved in real-time by completing the questionnaire in the PMSys mobile app by each player individually. The following list of parameters was presented in the questionnaires:

- Readiness to play (graded 0-10)
- Mood (graded 1-5)
- Stress levels (graded 1-5)
- General Muscle Soreness (graded 1-5)
- Fatigue (graded 1-5)
- Sleep quality (graded 1-5)
- Sleep duration (number of hours)
- Perceived Exertion (graded 1-10)

Parameters such as 'Mood', 'Stress', 'Soreness', 'Fatigue' and 'Sleep Quality' could take numerical values in the range between 1 and 5, where 1 is the worst case, and 5, respectively, the best. Explanation of the gradation can be found in the table 2.1.

Another parameter that will be of our interest is 'Readiness' to play. The gradation of the registered numerical values of this parameter differs from the parameters in the table 2.1. The values of the 'Readiness' could

	5	4	3	2	1
Fatigue	Very fresh	Fresh	Normal	More tired than normal	Always tired
Sleep Quality	Very restful	Good	Difficulty falling asleep	Restless sleep	Insomnia
Soreness	Feeling great	Feeling good	Normal	Increase in soreness/ tightness	Very sore
Stress	Very relaxed	Relaxed	Normal	Feeling stressed	Highly stressed
Mood	Very positiv mood	Generally good mood	Less interested in others and/or activities than usual	Snappiness at teammates, family and co-workers	Highly annoyed/ irritable/ down

Table 2.1: Example of the gradation of the parameters 'mood', 'stress', 'soreness', 'fatigue', 'sleep quality'.

be chosen between 0 and 10, where 0 is the worst and 10 is the best case, respectively.

Summing up the description of the collected data, we can assume that this data becomes of high value for all users of PMSys. With the proper machine learning model, one can try to predict the players' conditions and readiness either for training or for upcoming matches, or, for example, predict possible injuries.

It is also worth remembering that this data is personal for each player who registers it; therefore, this data's collection, storage, and processing should be considered from the technical point of view, ethics, and confidentiality. The collection, storage, and processing of data must, at a minimum, comply with a set of generally accepted standards of professional and ethical conduct that participants, acting in good faith and reasonably, must acknowledge and comply with. There needs to be a democratic approach to this issue that does not threaten the players' privacy. It is also necessary to have some compromise that would ensure that there is no excessive control over the players' conditions by those who will analyze and use their data.

2.3 Short introduction to time series

The data registered in the PMSys application at different points in time includes the measurement time, so one can easily refer it to a definition of a time series. Therefore, it is possible to use suitable models and algorithms to set and try to solve different kinds of regression problems with this time series.

A time series is an ordered collection of data points where each data point is an observation in time [6], [7]. The specified measurement time is the main characteristic distinguishing a time series from a simple data sample. Time series are used for analytics and forecasting when it is essential to determine what will happen to features in the future. For example, it is possible to try to predict how many users will download a mobile application per day. Features for compiling time series can be technical and economic, social, and even natural.

Figure 2.3 shows a simple example of time series. This figure is a plot taken from experiments predicting readiness to play. We used values of only one parameter called "Readiness" for this plot. The X-axis is used for timestamps indicating each day when submitting of questionnaires took place. Y-axis is used for values of the parameter "Readiness" itself. The plot shows changing of the 'Readiness' in time, which corresponds to the definition of time series. To remind that the definition says that 'time series' is a sequence taken at successive equally spaced points in time.

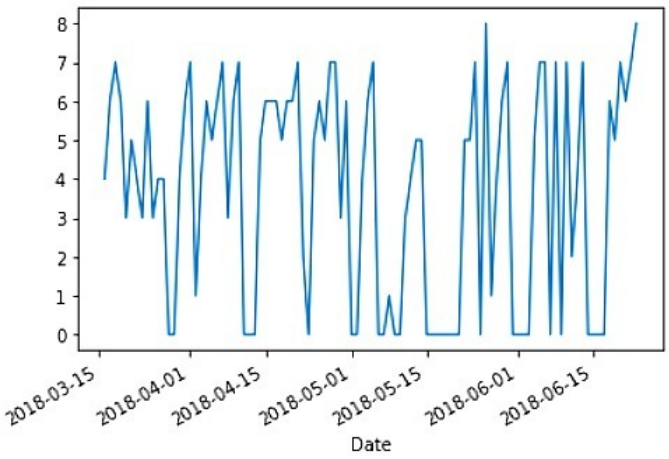


Figure 2.3: Example of time series that shows values of the parameter 'Readiness' to play for one player. The plot was made during the experiments to this thesis.

Based on the number of time-dependent variables, one can classify time series as:

- a univariate time series is a series with a single time-dependent variable. Measuring only readiness to play in PMSys at equally spaced points in time can be an excellent example of a univariate time series which we can see in figure 2.4,

- a multivariate time series, on the other hand, has more than one time-dependent variable. If we measure 'Mood', 'Stress', 'Soreness' and 'Fatigue' along with the 'Readiness', then we get a multivariate time series which we can see in figure 2.5.

```
0         0.0
1         0.0
2         0.0
3         0.0
4         0.0
         ...
31139     7.0
31140     0.0
31141     0.0
31142     0.0
31143     0.0
Name: Readiness
```

Figure 2.4: Example of univariate time series. Values of parameter 'Readiness' from PMSys data.

Time series analysis is a set of statistical methods for the current time series and its forecasting. Time series analysis methods differ significantly from simple sample data analysis methods. When analyzing a time series, the researcher is interested in the statistical characteristics of the time series and considers the relationship of measurements with time.

	Readiness	Mood	Stress	Soreness	Fatigue
0	3.0	4.0	3.0	3.0	3.0
1	3.0	4.0	3.0	3.0	3.0
2	3.0	4.0	3.0	3.0	3.0
3	3.0	4.0	3.0	3.0	3.0
4	3.0	4.0	3.0	3.0	3.0
...	...	...	...	...	...
7158	6.0	3.0	3.0	3.0	3.0
7159	6.0	3.0	4.0	3.0	3.0
7160	6.0	3.0	3.0	3.0	3.0
7161	7.0	3.0	4.0	3.0	3.0
7162	7.0	3.0	3.0	3.0	3.0

Figure 2.5: Example of multivariate time series. Values of parameter 'Readiness', 'Mood', 'Stress', 'Soreness' and 'Fatigue' from PMSys data.

The primary purpose of time series analysis is to forecast its values for future periods. The main tasks of time series analysis are:

- understand under the influence of which parameters the value of the time series is formed,
- build a mathematical model for each parameter or combination.

Any time series can be decomposed into a trend, seasonal, cyclical, and random components. The first three components form the non-random component of the time series. The random component is present in any time series, but components of a non-random constituent in the time series structure are not necessary.

This section gave a brief definition of time series. Also, we introduced the PMSys data in terms of time series and showed how this data could be used in uni- and multivariate cases during experiments. One of the main areas for time series research is machine learning, so we will briefly describe machine learning in the next section.

2.4 Machine learning

2.4.1 Introduction

One of the subtasks in this thesis is to research and compare various machine learning models for time series processing. In the previous section, we outlined what time series data were and gave some examples of what time series are used for. This section will describe one of the main areas of research in which time series is actively used. This area is machine learning (ML).

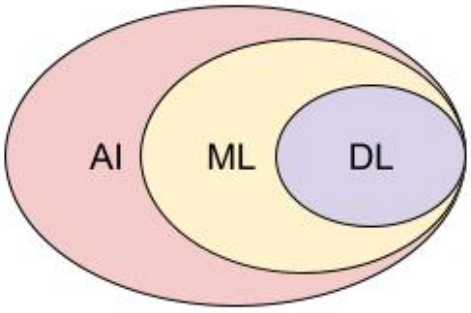


Figure 2.6: Example of dependency between Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL).

ML is a part of a massive area of research called artificial intelligence (AI). One can think of AI as a robot/machine capable of imitating human behavior by learning from previously obtained information/experience. The creation of this machine is a complicated task. Therefore, ML and DL concepts were introduced. These concepts were classified as artificial intelligence subsystems, which you can see in figure 2.6.

For solving machine learning problems, one usually tries to find appropriate models. The main tools for these models are mathematical statistics and analysis, probability and graph theory, and methods for processing large amounts of data. The main types of problems solved using machine learning models are:

- classification - most often used to determine the object type, for example, what or who is in the picture,
- regression - most often a prediction of the value of the parameter of interest to us.

These tasks can often be independent but sometimes can be used in combination. To give an example of the combination of these tasks, let say we can first try to predict some parameter and then classify it by referring to one or another group.

The main types of machine learning are:

- supervised learning,
- unsupervised learning.

The primary difference between these two types of training is that we have the opportunity to test our models on ground truth data in the first case, but for unsupervised training, we have no valid data for testing. In this thesis, we will predict readiness to play using PMSys data, which, as shown in the previous section, are time series. Therefore, we need to use machine learning models suitable for processing this data type which is discussed in the next subsection.

2.4.2 Models appropriate for time series processing

It is known that the prototype of an artificial neural network is the structure of the human brain, which consists of a vast number of neurons. The simplest type of neural network is the 'perceptron'. This is a mathematical model of information perception by the human brain, consisting of one layer of neurons. Each neuron has a certain weight, which indicates the importance of processing input information. The input data is processed individually by the neurons at the first step. Then the processed pieces of the data are added together and form some final output.

The described neural network is straightforward, called 'feed-forward neural networks', and its capabilities are minimal. Such a network would not be able to solve serious problems. To achieve a more significant effect, one can increase the number of layers in the network. After that, the new one can be classified as deep learning neural networks. It's also possible to add an operation of 'convolution', after which we get a 'convolutional' neural network. This type of network was successful with video processing and image classification. But, to summarize, from a mathematical point of view, this type of network is a function that has a fixed number of input parameters and a fixed type of the result obtained.

A completely different type of tasks if we need to process data with the following conditions:

- the inputs are long contiguous sequences,
- in these sequences, the order in which the data is received matters.

When solving this kind of tasks, it becomes necessary to store somehow the previous state of the network. Unlike feed-forward neural networks, where information is transmitted through layers only forward, in our case, it is necessary to refer to the state of the network to the previous layers. For this purpose, recurrent neural networks (RNN) have been developed. In these networks, for the first time, such a concept as 'memory' was used. This concept means that neurons receive basic information as input data and information about the state of the network in the past, which is responsible for the order of the data in the sequence. The neural network designers agreed that if convolutional neural networks can be compared to a mathematical function, as described above, then recurrent neural networks can be called a program.

The development of RNN made a significant contribution to the study and solving regression problems associated with time series. Therefore, recurrent neural networks are applicable in tasks where something holistic is broken into parts. One of the most popular types of recurrent neural networks is the Long Short-Term Memory (LSTM) network. Some of the most outstanding achievements of using LSTM are:

- revolutionization of speech recognition, outperforming traditional models in specific speech applications [8],
- improving large-vocabulary speech recognition [9], [10],
- breaking records for improved machine translation [11].

Also, in 2014, it was introduced a gate mechanism called (Gated Recurrent Units, GRU) for recurrent neural networks. One found that its effectiveness in solving problems of modeling music and speech signals is comparable to the use of long short term memory [12]. Compared to LSTM, this mechanism has fewer parameters because there is no output gates [13].

2.5 Previous Machine Learning Research related to PMSys

The work called 'Predicting Peek Readiness-to-Train of Soccer Players Using Long Short-Term Memory Recurrent Neural Networks' [4] was independent research of player performance using LSTM model. This work was carried out at the Simula Research Laboratory.

The paper [4] presented an extended version of PMSys, which included a machine learning algorithm based on the LSTM model. The data used in the experiments were taken from PMSys. This study aimed to investigate the possibility of using the LSTM model to predict readiness to play.

Two data sets from different football teams from the Norwegian championship were used for all experiments with information about the players' well-being. As indicated in the papers, every player recorded the data daily, and this was the standard list of parameters described in section 2.2. Table 2.1 provides a complete data set specified in the papers.

Both data sets contained gaps, which means that not all players logged data daily for the specified period. One data set included data from 19 players and another from 22 players. The gaps were not replaced or removed for the initial experiments to create a more realistic picture. Two different options were used to train and validate the model:

1. training on all other players on the team and then predicting readiness for the selected player,
2. training and predicting on the data of the same player.

As the tests showed, the best results were achieved when training on all other players and predicting on one, see figures 2.8 and 2.10. The prediction graph followed the actual data graph, but the peaks often did not coincide.

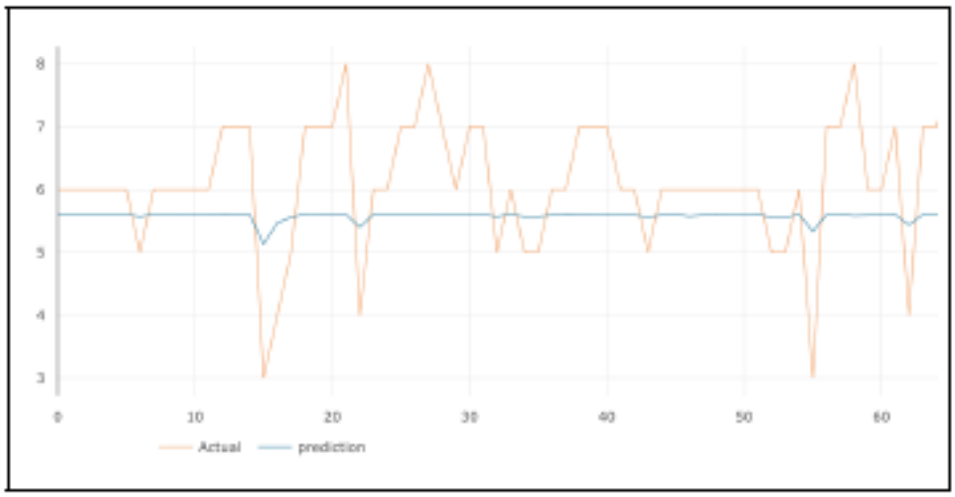


Figure 2.7: Actual data and predictions for a random player from team 1 based on the model trained on the specific player in the team 1 [4].

Further, using the LSTM model and the same training methods ended up with two models for each team. As stated in the document, it is likely that due to the lack of sufficient data for training, accurate prediction of "Readiness" is not possible, so the experiments were extended to identify positive and negative peaks. The following gradation was performed to determine the peaks for "Readiness":

- positive peak - values from 8 to 10,
- negative peak - value from 0 to 3.

The positive peaks were reasonably well predicted, as stated by the author [4], but it was not easy for the model to distinguish these high values from each other. Negative peaks were also well predicted, and the predictions varied well between the lower extremes.

As a result, it was concluded that the models trained on the data of both teams gave better predictions than the models trained on individual

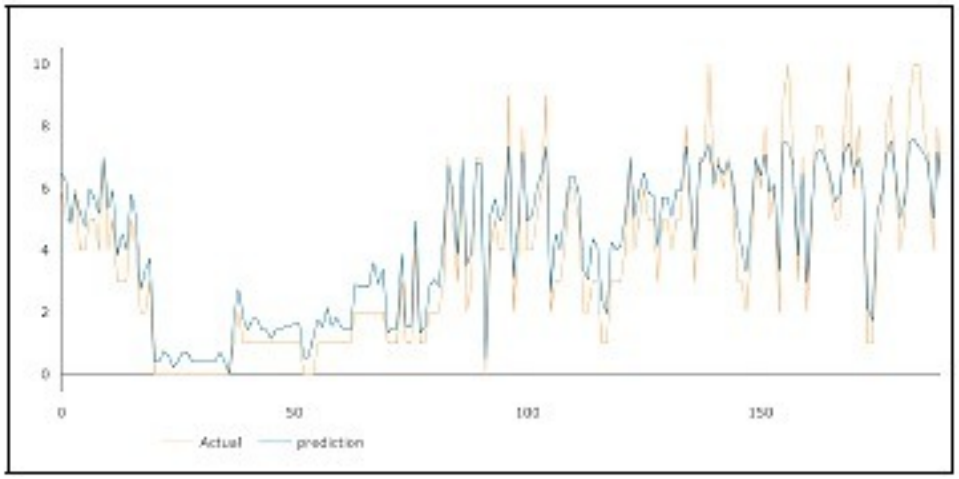


Figure 2.8: Actual data and predictions for a random player from team 1 based on the model trained on all players in the team 1 [4].

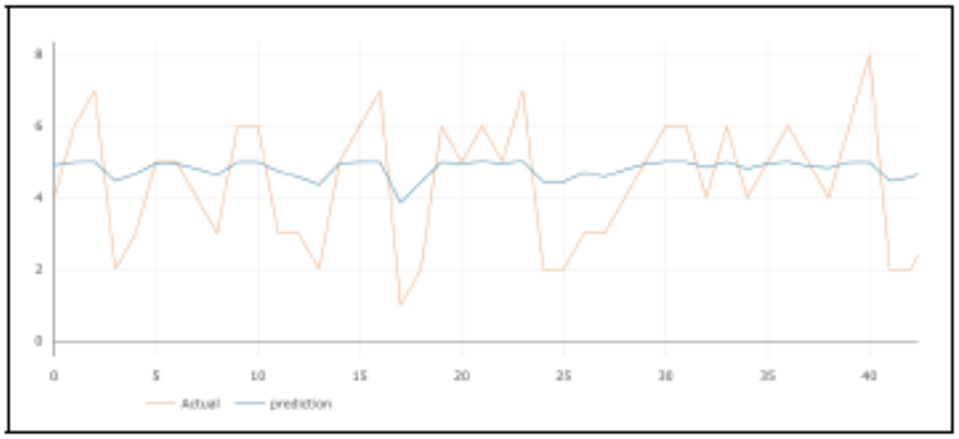


Figure 2.9: Actual data and predictions for a random player from team 2 based on the model trained on the specific player in the team 2 [4].

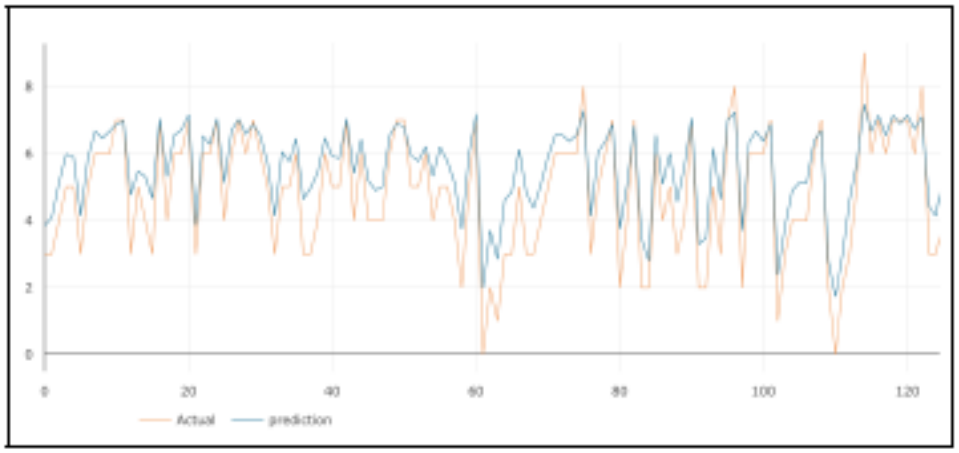


Figure 2.10: Actual data and predictions for a random player from team 2 based on the model trained on all players in the team 2 [4].

players [4]. It was assumed that this was possible because, in the first case, there was more data for training. Figures 2.8 and 2.10 show, that the results obtained with machine learning in the related research in predicting 'Readiness' to play based on the data registered in PMSys are promising; therefore it was decided to continue the same research in this thesis, but using a broader range of machine learning models for processing time-series data.

2.6 Brief description of 'Tsai' library

Due to the development of computing power, the number of problems related to time series and wanting to be solved is constantly increasing. As a result, it becomes possible either to develop more complex architectures of neural networks or to use an increasing number of layers or various combinations of already existing neural networks.

After some time searching for modern deep learning models suitable for further research in predicting readiness to play, a library called 'Tsai' was found. Here is a link [1] to 'Tsai' documentation. This library is positioned as 'State-of-the-art Deep Learning library for Time Series and Sequences', which is currently presented as an open-source project, but at the same time contains some ready-made deep learning models that is possible to apply to solving the problems posed in this thesis.

'Tsai' is an open-source deep-learning package built on top of Pytorch and fastai [14] focused on state-of-the-art techniques for time series tasks like classification, regression, forecasting, imputation. This information was taken from the main page [5]. The 'Tsai' documentation has the four main machine learning sections (information taken from 'Tsai' documentation[5]):

- 'Data' - this section mainly contains functionality for working with data, namely for loading existing data sets, data preparation to experiments, splitting, preprocessing, and some other operations,
- 'Training' is the next section that contains fastai [14] Learner extensions, a new set of time series learners, some optimizers, some metrics that are not in tsai, and other functionality referring to the training step in a machine learning pipeline,
- 'Inference' is the section that contains implemented functionality for the inference step,
- 'Models' - this section contains some helper functionality used to build PyTorch time-series models and a list of 'ready-to-use' models.

After doing some initial research and getting more details about this library and the examples given in it for solving various classification and regression problems, we concluded that this library may be well suited for

[1] https://timeseriesai.github.io/tsai/

further research in predicting readiness to play; therefore, the functionality from this library will be used for experiments in this thesis. In the next chapter, we will describe the models and other functionality chosen from 'Tsai' for experimenting with predicting readiness to play.

2.7 Summary

In section 2.1, we presented a brief description of the player monitoring system. It was described the main purpose of creating this system, who the users are, how the users register the data. Further, it was indicated which intention exists regarding developing this system, considering the introduction of machine learning algorithms for processing the collected data and the possible prediction of various parameters. At the end of this section, we briefly described the architecture and purpose of all the central units.

In section 2.2, the data collection process was presented with a detailed description of all the main recorded parameters, their possible values, and interpretation. Since the collected data can be defined as time series, section 2.3 was used to introduce time series. We described univariate and multivariate time series with pictures as examples and introduced briefly some types of modern machine learning models used to process time-series data. Also, some achievements that were achieved in the recent decades by the LSTM model were mentioned here.

Since the problem considered in this thesis refers to machine learning problems, in section 2.4, we presented a short introduction to this technology, briefly described the types of problems that machine learning can solve. We also explained the difference between supervised and unsupervised machine learning. After that, we gave examples of some modern models from the RNN family for processing time series and listed some of the problems solved by these models.

In section 2.5, we summarized some research that was done previously regarding making predictions based on the data from PMSys. Also, we referred to the section 2.6, in which we explained why the 'Tsai' library is of our interest for this thesis and briefly described the content of this library.

In the next chapter, we are going to discuss our research methodology in more detail. We will discuss the specification of our system with an indication of the hardware characteristics and a brief description of the software used. Then we will select a list of models to experiment with, indicating their configurations. Next, we will describe the data set used in the experiment; we will demonstrate how this data will be used to train and test our models. We will also list the metrics that will be used to assess the accuracy of the models and their performance.

Chapter 3

Methodology

In this chapter, we will briefly describe the specification of our system, the programs used to build the pipeline. We will also look at the 'Tsai' library and select a list of models for our experiments. After that, we will describe the data set that will be used in the experiments; we will show how this data will be divided into sets for training and testing our models. At the end of the chapter, we will list and describe the metrics that will be used to test the accuracy and performance of our models.

3.1 System specification

Table 4.1 shows the central hardware units with an indication of their type and version and the stack of technologies used to process data and build a pipeline for training and testing our models. Python is a suitable language for working with large data sets since it is easy to use and has many already implemented libraries related to machine learning. To extract data from a file and prepare it for experiments, we used the functionality of the pandas library. Pandas' data functionality is built on the NumPy library, a lower-level tool. Includes special techniques for working with numeric tables and time series. 'Tsai' was used as the main library providing functionality for preparing data for training models, as well as some ready-to-use machine learning models for various types of tasks, such as classification and regression. Torch is the core machine learning library on top of which the fastai and 'Tsai' libraries are built. To test the performance of our models, we will use the process and system utilities library (psutil). Now we move closer to describing those models and their configurations used in our experiments. This description will be presented in the next section.

3.2 Machine learning models chosen for experiments

One of the tasks set in this thesis was to research and compare various machine learning models for processing time series. After some searching, we found the 'Tsai' library, represented as the 'State-of-the-art Learning

Type	Name	Version	Description
Software	Ubuntu	18.04.6 LTS	OS
	Python	3.6.9	Implementation language
	Tsai	0.2.22	Machine Learning library
	Fastai	2.5.2	Machine learning library
	Torch	1.9.0+cu102	Machine Learning library
	Pandas	1.1.5	Data analysis library
	Psutil	5.8.0	Performance measurement library
Hardware	Memory	DDR4, 46.9 GiB	-
	Hard disk	SSD, 491.2 GiB	-
	CPU	Intel Core i7-9700, 8 cores	-

Table 3.1: System specifications.

library for Time Series and Sequences', and containing a set of already implemented models for classification and regression tasks using time series. There are two possibilities for using these models: either use their default configurations or create the same models using our desirable configurations. Since this library is entirely new for us, we decided to use models with default configurations in this thesis first and then make changes to these configurations if we have enough time. Otherwise, we can recommend testing these models with other configurations for further work with the 'Tsai' library if we get satisfactory initial results.

The set of models presented in 'Tsai' is quite diverse. There are purely recurrent models such as RNN, LSTM, GRU. There are also models of mixed recursive-convolutional types. One can also select convolutional models only. We decided to choose models of different types and architectures for our experiments for a more objective assessment. These models are:

- InceptionTime,
- RNN, LSTM, GRU,
- RNNPlus, LSTMPlus, GRUPlus,
- RNN-FCNPlus, LSTM-FCNPlus, GRU-FCNPlus,
- MRNN-FCNPlus, MLSTM-FCNPlus, MGRU-FCNPlus.

Further in this section, we will present short descriptions and configurations for all these models.

InceptionTime
InceptionTime is an ensemble of deep Convolutional Neural Network (CNN) models inspired by the Inception-v4 architecture [15].

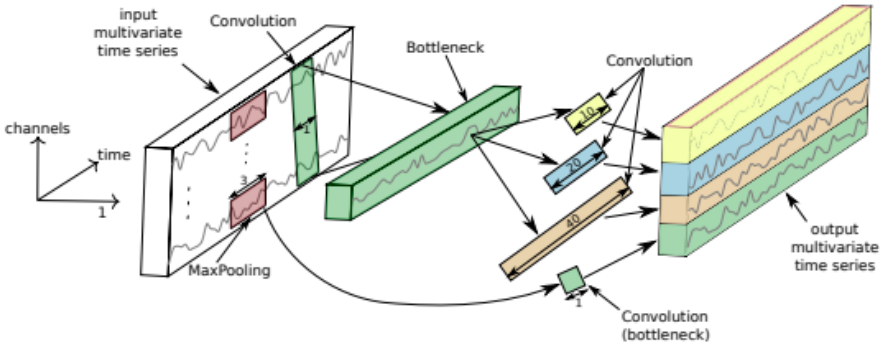


Figure 3.1: Inside the Inception module for time series classification. For simplicity a bottleneck layer of size $m = 1$ is illustrated. This picture was taken from this paper [15].

This model [15] consists of the following layers:

- A bottleneck layer to reduce the dimensionality (i.e. the depth) of the inputs. This cuts the computational cost and the number of parameters, speeding up training and improving generalization.
- The output of the bottleneck is fed to three one-dimensional convolutional layers of kernel size 10, 20 and 40.
- The input of the inception module is also passed through a Max Pooling layer of size 3 and in turn, through a bottleneck layer.
- The last layer is a depth concatenation layer where the outputs of the four convolutional layers at step 2 are concatenated along the depth dimension.

More detailed information about the 'InceptionTime' model can be found in the paper 'InceptionTime: Finding AlexNet for Time Series Classification' [15]. Table 3.2 shows the configuration of this model.

RNN - Recurrent Neural Network [16]
Some tasks in machine learning have been defined as tasks that are associated with the processing of sequential data. For these purposes, regular neural networks were not appropriately adapted. It became a reason for developing a new kind of neural network called a recurrent neural network. This class of neural networks is very well suited for working with tasks that are based on the processing of sequences. Recurrent neural networks have proven themselves very well in many

Parameter	Value
Number of filters (nf)	32
Residual	True
Depth	6
Kernel size	40
Bottleneck	True

Table 3.2: Default configuration of the InceptionTime model.

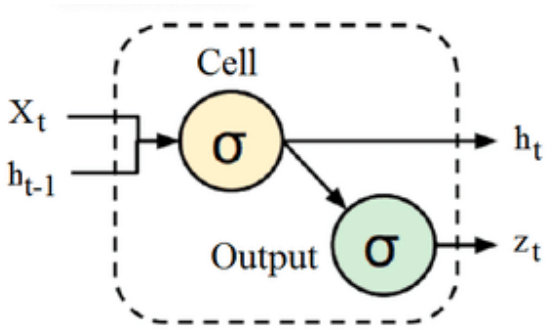


Figure 3.2: RNN example. Source https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359.

neuro-linguistic programming problems.

RNN training is similar to training a regular neural network. It also uses the backpropagation algorithm but with a slight change.The same parameters are used during the training at all time steps in the network. The gradient at each output depends not only on the calculations of the current step but also on the previous time steps. This algorithm is called the Backpropagation Through Time (BPTT) algorithm. Recurrent neural networks trained with BPTT have difficulty learning long-term dependencies. Two main factors affect the magnitude of the gradients:

- weights,
- derivatives of the activation function.

If one of these factors is more than 1, then exploding of gradients can occur, and if less than 1, then gradients can vanish over time. The picture 3.2 shows that input X_t and the previous hidden state h_{t-1} are used as input parameters.

LSTM - Long Short Term Memory [17]
'The next event builds on the previous one'. This approach was taken as

a basis when developing RNN. When taking the following action, it is necessary to focus on the experience from the past. Conventional neural networks do not possess this property, and the developed RNN turned out to be limited in the long-term storage of memory about events from the past. Therefore, the next step in the development of RNN was the development of networks that could store data about previous occasions for a more extended period. The LSTM network became such a network. LSTM

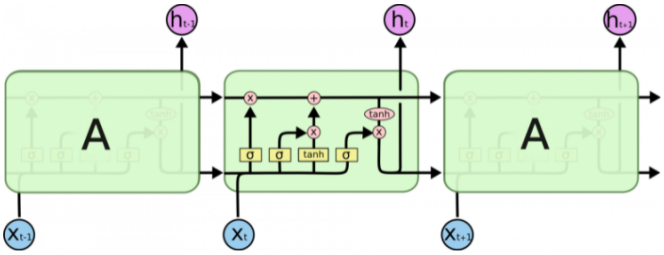


Figure 3.3: LSTM example. Figure and description to it was taken from https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

do not have a fundamentally different architecture from RNN, but it uses a different principle for computing a hidden state, see figure 3.3. The main component of LSTM is called the 'cell state'. The state of the cell resembles a pipeline. Information can easily flow through it without being subject to change. However, LSTM can modify information from the cell state. Such units as gates manage this process. They consist of a sigmoid neural network layer and a pointwise multiplication operation. Numbers between '0' and '1' is a result of this sigmoid layer. '0' and '1' are the peak values, where '0' says 'let nothing through', and '1' means 'everything goes through'. More detailed information about LSTM can be found in the paper 'LONG SHORT-TERM MEMORY' [17] or using this link [1].

GRU - Gated Recurrent Unit [12]
Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al [18]. Compared to LSTM, this mechanism has fewer parameters because there is no output gate [13]. Table 3.3 shows the configuration of the RNN, LSTM, and GRU models from the 'Tsai' library. This configuration was found here [2]. By default, these models have the same base RNN configuration. They differ only in the architecture of the cells in the network layers.

RNNPlus, LSTMPlus, GRUPlus [16]
We didn't find much information on how RNN, LSTM, and GRU differ from RNNPlus, LSTMPlus and GRUPlus. The only difference we found was that a feature extractor was included in the RNN, LSTM, and GRU. One can read here [3] that the idea of including a

[1] https://colah.github.io/posts/2015-08-Understanding-LSTMs/
[2] https://github.com/timeseriesAI/tsai/blob/main/tsai/models/RNN.py
[3] https://timeseriesai.github.io/tsai/models.RNNPlus.html

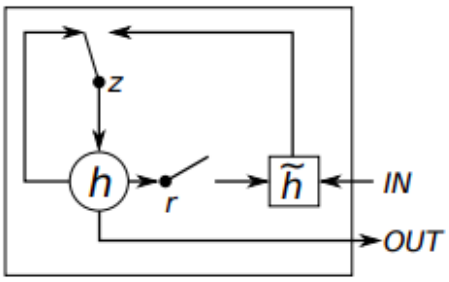


Figure 3.4: Illustration of gated recurrent units. r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation.

Parameter	Value
Number of layers	1
Hidden size	100
Bias	True
RNN dropout	0
Bidirectional	False
Type of cells	RNN for RNN model LSTM for LSTM model GRU for GRU model

Table 3.3: Default configuration of the RNN, LSTM and GRU models.

feature extractor in the RNN network comes from the solution developed by the UPSTAGE team (https://www.kaggle.com/songwonho, https://www.kaggle.com/limerobot, and https://www.kaggle.com/jungikhyo). They finished in 3rd position in Kaggle's Google Brain - Ventilator Pressure Prediction competition. They used a Conv1d + Stacked LSTM architecture.

RNN-FCNPlus, LSTM-FCNPlus, GRU-FCNPlus [16]
RNN-FCNPlus, LSTM-FCNPlus and GRU-FCNPlus are extensions of the regular RNN, LSTM and GRU models. Fully convolutional neural networks (FCN) have successfully solved sequential time series classification problems. Therefore, it was decided to change the architecture of conventional recurrent networks by combining them with FCN. As indicated in the papers [19][20], these model architectures significantly improve the performance of fully convolutional networks and require minimal data preprocessing. In the proposed models, a fully convolutional block is supplemented with an RNN / LSTM / GRU block followed by a dropout, as shown in figure 3.5. In these works [19][20], one can be familiarized in more detail with the architecture of these networks and get complete information about their use and capabilities. Table 3.4 shows the configuration of RNN-

FCNPlus, LSTM-FCNPlus, and GRU-FCNPlus used in experiments for this thesis.

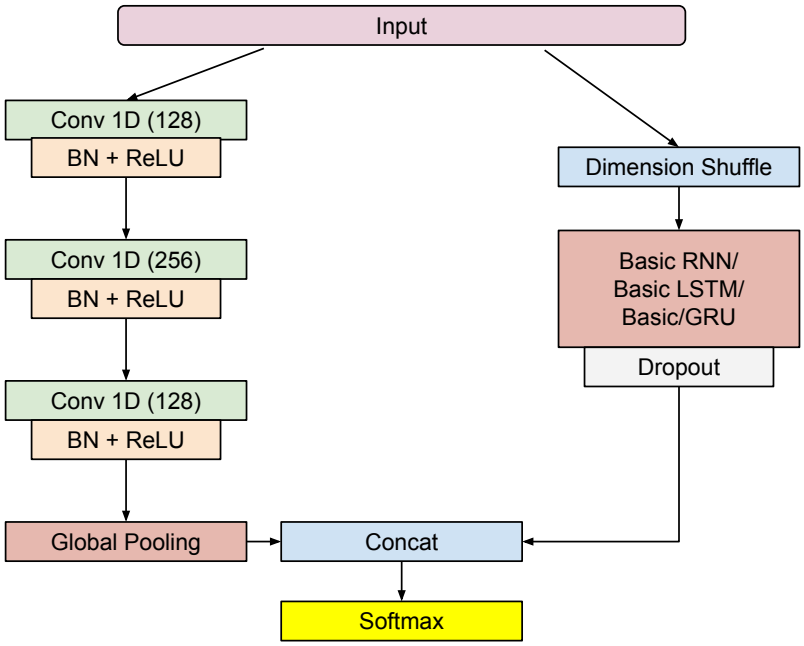


Figure 3.5: RNN-FCN, LSTM-FCN and GRU-FCN architecture.

MRNN-FCNPlus, MLSTM-FCNPlus, MGRU-FCNPlus [16]
Multivariate time-series models such as MRNN-FCNPlus, MLSTM-FCNPlus, and MGRU-FCNPlus were extended from the univariate time-series models RNN-FCNPlus, LSTM-FCNPlus, and GRU-FCNPlus. This modification is made by augmenting the fully convolutional block with a squeeze-and-excitation block to improve accuracy, as shown in figure 3.6. Our experiments will use uni- and multivariate cases to predict readiness to play; therefore, it will be interesting to test these models. In the work [21], one can be familiarized in more detail with the architecture of these networks and get complete information about their use and capabilities. Table 3.4 shows the configuration of MRNN-FCNPlus, MLSTM-FCNPlus, and MGRU-FCNPlus used in experiments for this thesis.

3.3 Data set

3.3.1 Overview of use cases

Based on the fact that the data set to be used for experiments (see section 2.2) contains several parameters, it was decided to predict readiness to play using uni- and multivariate time series, as shown in figure 3.7. In predicting with univariate time series for training our models, we

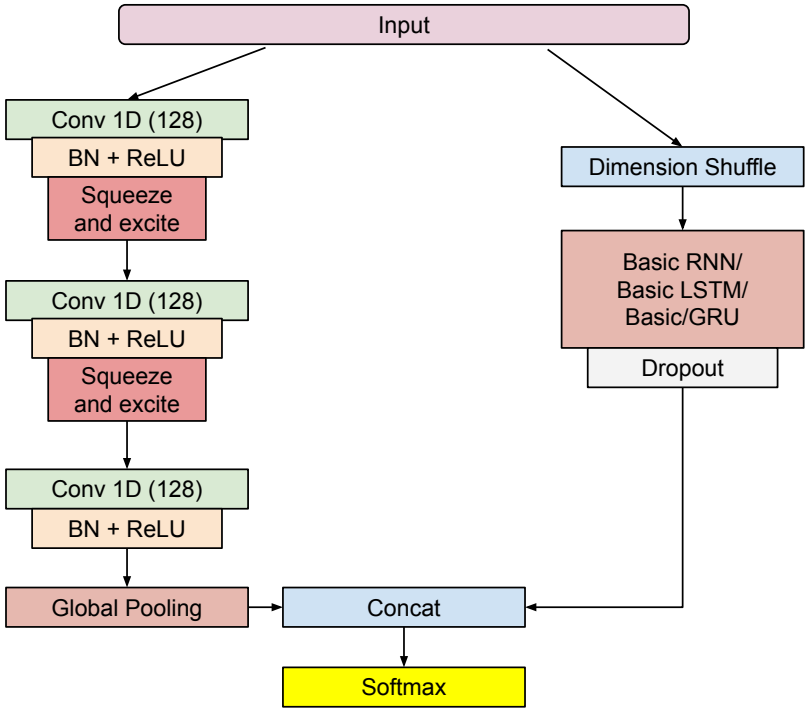


Figure 3.6: MRNN-FCN, MLSTM-FCN and MGRU-FCN architecture.

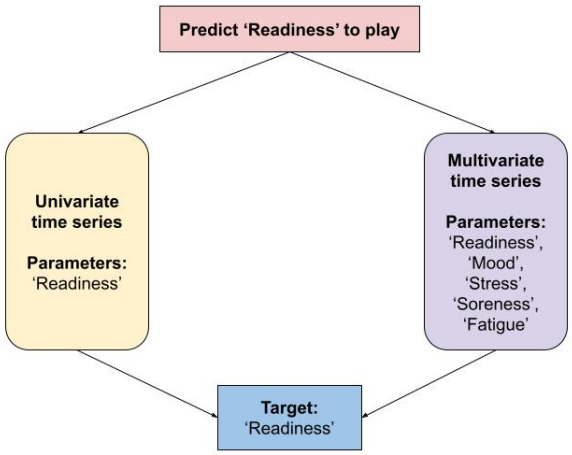


Figure 3.7: Diagram describes predicting 'Readiness' to play using univariate and multivariate time series.

Parameter	Value
Number of layers	1
Hidden size	100
Bias	True
RNN dropout	0.8
Bidirectional	False
Shuffle	True
Conv layers	128, 256, 128
Kernel sizes	7, 5, 3
Squeeze-and-excite block for MRNN-FCNPlus, MLSTM-FCNPlus, and MGRU-FCNPlus	16
Type of cells	RNN for RNN-FCNPlus and MRNN-FCNPlus models LSTM for LSTM-FCNPlus and MLSTM-FCNPlus models GRU for GRU-FCNPlus and MGRU-FCNPlus models

Table 3.4: Default configuration of the RNN-FCNPlus, LSTM-FCNPlus, GRU-FCNPlus, MRNN-FCNPlus, MLSTM-FCNPlus and MGRU-FCNPlus models.

will use only one parameter, 'Readiness', as training data. In contrast, with multivariate time series, we will use five parameters 'Readiness', 'Mood', 'Stress', 'Soreness' and 'Fatigue'. The target parameter for uni- and multivariate predictions will be 'Readiness'.

3.3.2 Data set for prediction of 'Readiness' to play using univariate time series

As discussed earlier in this chapter, only one parameter called 'Readiness' from the entire data set is used for the 'univariate' case of predicting readiness to play.

Unfortunately, for some reason, the data was not recorded every day, and as a result, the data array turned out with gaps. Several solutions were considered to fill these gaps:

- replace all gaps with '0' assuming that this is a natural process, that a person forgets to register the data,
- fill in the gaps with averaged values,

- remove all gaps and concatenate the array.

At the initial stage was decided to start with the data of only one player because we will have a lot of experiments and we are restricted in time. Also, it was decided to use the first option and fill in all the missing data with '0', considering that it is a natural process to forget to register the data. This option turned out to be attractive regarding the regression models' results with the natural presence of gaps. We got the following data array, a plot of which is shown in figure 3.8. This figure shows how the parameter 'Readiness' values are spread in time.

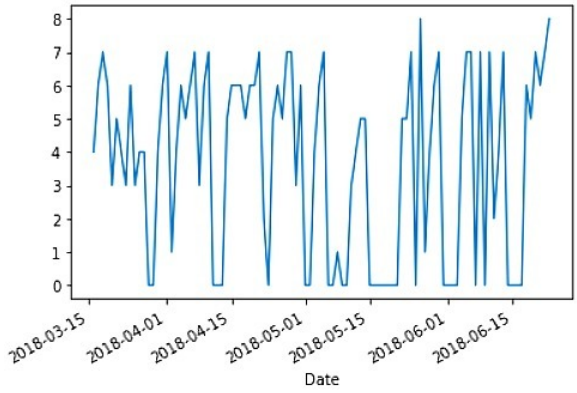


Figure 3.8: Parameter 'Readiness' for one player. Values ranged between 0 and 10. There are 914 data units in total.

3.3.3 Data set for prediction of 'Readiness' to play using multivariate time series

As it was shown in figure 3.7 we decided to use five parameters for the case with multivariate time series since all the parameters can influence readiness to play. Plot of the data for these five parameters is shown in figure 3.9. This figure shows how the values of the 'Readiness', 'Mood', 'Stress', 'Soreness', 'Fatigue' parameters are placed corresponding to equally spread timestamps.

3.3.4 Description of splitting the data set for one player into 'training', 'validation' and 'test' sets

As mentioned earlier, due to time constraints, it was decided to use the data of only one player at the initial stage. As a result, we've got a data array with a length of 914 units. One can see in the picture 3.10 the ratio of the data divided into testing and validation sets. The balance was 80 to 20 percent of the 'training + validation' set correspondingly. We used the functionality provided by the 'Tsai' library to separate the data.

It was also decided to test predictions of 'Readiness' for the last ten days. For more objective results, the test data must not be a part of training

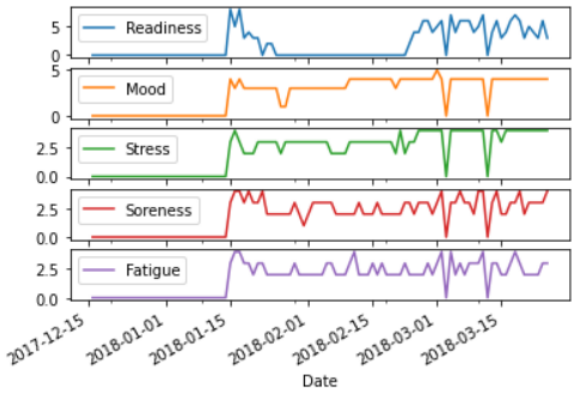


Figure 3.9: Parameters 'Readiness', 'Mood', 'Stress', 'Soreness', 'Fatigue' for one player used as multivariate time series. Values of 'Mood', 'Stress', 'Soreness', 'Fatigue' are ranged between 0 and 5, of 'Readiness' between 0 and 10.

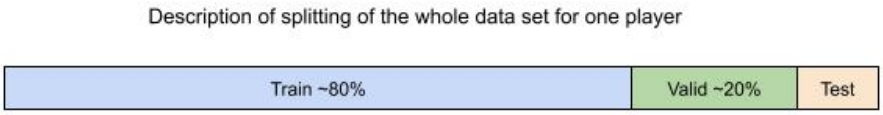


Figure 3.10: General example of splitting the data for one player into training and validation sets.

and validation sets or overlap them in any way. The following subsection will give a detailed explanation of test data sets for different experiments.

3.3.5 Detailed explanation of considered 'test' data sets for different cases

Here we consider that for training and validation, we use data from one player only. Therefore we can have two different cases we want to check, namely:

1. if the training data contains gaps filled with '0', then we want to have two test data sets:
   - one test data set containing '0' to check how well models can predict gaps,
   - one test data set without '0',
2. if gaps filled with '0' are removed from the training data set, then we want to have two data sets:
   - but one data set must have some peak values to check how well our models can predict them

It was also discussed that we could train and test our models with more data if we get enough time. Then we can try training on all data for all players but one, and testing on this one. In this case, we can also consider both cases with first having gaps with '0' and then removing them.

3.4 Metrics

To understand how effective a model is, we need to evaluate it regarding the accuracy of the prediction and the resources expended. This section describes metrics that we will use to evaluate our models.

3.4.1 Machine learning metrics

Regression models are a family of machine learning and statistical models used to predict continuous target values. They have a wide range of applications, from house price forecasting, e-commerce pricing systems, weather forecasting, stock market forecasting to super-resolution images, feature exploration with autoencoders, and image compression.

Our research is related to predicting readiness to play, and this problem belongs to the regression family, so we will use the metrics that are used for this type of models.

The metrics used to evaluate models related to prediction problems must operate on a set of continuous values (with an infinite number of items) and therefore differ slightly from the classification metrics. The most popular metrics for regression models are mean square error (MSE), root mean square error (RMSE), and mean absolute error (MAE).

MAE.
MAE is a metric that defines the average absolute distance between predicted and target values. MAE is defined as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

MAE is known to be more resilient to emissions than MSE. The main reason is that in MSE, by squaring the errors, outliers (which usually have higher errors than other samples) receive more attention and dominate the final error and affect the model's parameters.

MSE.
MSE measures the average sum of the square of the difference between the actual value and the predicted value for all data points. Exponentiation is performed, so negative values are not compensated with positive ones. Due to the properties of this metric, the influence of errors increases by quadrature from the original value. This increase means that if in the original measurements we were mistaken by 1, then the metric will show

1, 2-4, 3-9 and so on. The lower the MSE, the more accurate our prediction is.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Compared to the mean absolute error, MSE has several advantages: it emphasizes big mistakes over more minor mistakes. It is differentiable, which allows it to be more efficiently used to find the minimum or maximum values using mathematical methods.

RMSE.
Sometimes people use RMSE to have a metric with a scale as target values, which is essentially the square root of the MSE. The RMSE shows the average deviation in our model of predicted readiness to play from target values (the actual values of 'Readiness' to play).

$$RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

RMSE is easy to interpret since it has the same units as the original values (opposed to MSE). It also operates with smaller values in absolute value, which can be helpful for computing on a computer.

3.4.2 Metrics for performance measurement

The primary metrics for measuring performance are usually time spent on training and testing models, processor, memory, disk, and GPU usage. The computer on which the experiments will be carried out has no GPU; therefore, we will measure all other parameters except the GPU usage. Below we describe how the measurement will be done.

Time usage
Time will be measured in seconds. For measuring the time to train and test each model, we will use the 'time' module [4] from the python standard library, which provides various time-related functions. The form of usage of this 'time' module is shown in the listing 3.1.

```
import time as t

start = t.time()
<some code>
end = t.time()
print('Time used for learning of the <model_name>
```

[4] https://docs.python.org/3/library/time.html

```
model: ', end - start, ' sec')
```

Listing 3.1: Example of 'Time' module usage from Python library.

CPU, memory and disk usage
To measure CPU, memory, and disk usage, we will use a cross-platform library called psutil [5]. This library has a set of utilities for obtaining information about system performance. Since our processor consists of 8 cores, we will use the utility 'cpu_percent()' to get information about a load of each core in percentage. We will also use psutil to measure memory and disk usage. The 'virtual_memory()' utility function provides information about various memory parameters. We are interested in the parameter 'used', which shows the amount of memory used at a given time. The 'disk_usage()' function also has the parameter 'used' that provides information about the used disk space at a given time. We will measure the used memory and disk in GiB. We will measure CPU, memory, and disk usage with an interval of one second, store this data in a list, and after the measurement is performed, we will make a plot of this list. The form of usage of psutil module for measuring of CPU, memory and disk usage is shown in the listing 3.2.

```
import psutil
import matplotlib.pyplot as plt

<some code>

while count < secs:
    cpu.append(psutil.cpu_percent(interval=1, percpu=True))
    memory_used_GB.append(psutil.virtual_memory().used/10**9)
    disk_used_GB.append(psutil.disk_usage('/').used/10**9)
    count += 1
plot(cpu, memory_used_GB, disk_used_GB)

<some code>
```

Listing 3.2: Example of 'psutil' usage from Python library.

3.5 Summary

In section 3.1 we have listed the main hardware parameters of the computer on which the experiments will be carried out. We also briefly described the technology stack used in the experiments. In section 3.2,

[5] https://pypi.org/project/psutil/

we selected a set of ready-made models for experiments on predicting readiness to play, described these models, and tried to justify why we chose these particular models. We also described the configurations of the models and indicated additional sources in which the reader can get acquainted with them in more detail.

Section 3.3 detailed the use cases (usage of uni- and multivariate time series), based on which we prepared our data sets for experiments. We also described in detail the data set for each use case, gave examples of uni- and multivariate time series, described how the data will be split for training and testing of our models, and indicated with a description which cases we will use consider when testing our models.

In section 3.4, we described the metrics with which we will test our models for prediction accuracy and performance. For prediction accuracy, we have listed and described the main metrics that are usually used in regression problems. As a result, it was decided to use MSE as the primary metric for assessing the accuracy of predictions. We have also listed metrics for evaluating the performance of our models. We described the cross-platform library psutil, which contains utilities for measuring CPU, memory, and disk usage, and gave examples of their use in our experiments.

In the next chapter, we will make a detailed review of our experiments and present the obtained prediction results for all the cases we consider. We will also measure the performance of our models. At the end of the chapter, we will discuss the results obtained.

Chapter 4

Experiments and results

This chapter will give a detailed overview of our experiments, describe in detail how they were performed, and describe the results. We will explain how we changed our experiments at each step to determine the dependence of the results obtained on various parameters and determine which models were the best in predicting readiness to play. We will also present the results of measurements of the performance of our models and draw conclusions. At the end of the chapter, we will discuss the results obtained.

4.1 Experiments overview

Exploring the new 'Tsai' library, we decided to check which results we can achieve using ready-made models with default configurations. We selected 13 models from this library for our research. These models belong to the family of recurrent neural networks and should be suitable for time series processing. The list of models can be found in table A.1. Before the experiments, it was decided to highlight a list of parameters for research that can significantly affect the outcome. During the experiments, we will compare the influence of using the following parameters on the results:

- type of input data (univariate vs. multivariate time series),
- sliding window size,
- data for testing with and without gaps filled with '0', saying that it is missing in registrations,
- number of epochs,
- amount of input data.

We will execute our research in two stages. We have developed a series of experiments detailed in the following sections based on the above parameters. We will run these experiments at the first stage. The idea behind these initial experiments is to determine the most promising models based on the best results of predictions of readiness to play obtained. At

the second stage, the best models will be tested for performance on the following parameters:

- CPU usage,
- memory usage,
- time usage.

After that, we will evaluate the models in terms of prediction results vs. the results of performance measurements and propose the best models for further study.

4.2 Initial experiments

4.2.1 Overview

The initial experiments will give us the first results, based on which we will draw some conclusions and perform further research. We are interested in obtaining the broadest possible range of outcomes for getting an objective picture of the behavior of the models we have chosen. Based on these goals, we have drawn up a plan for the initial experiments, shown in figure 4.1.

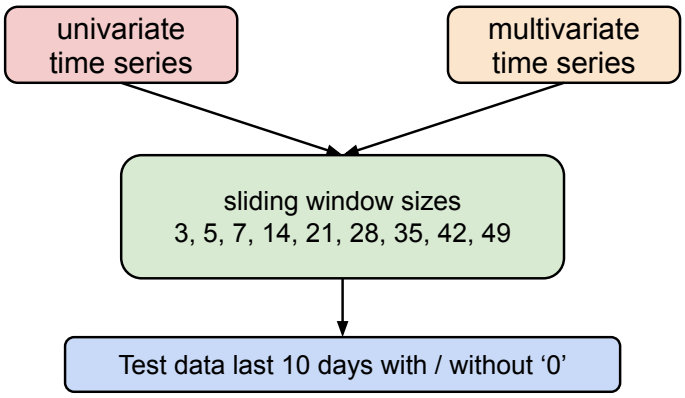


Figure 4.1: Scheme for initial experiments.

Figure 4.1 briefly describes executing the initial experiments to predict readiness to play. First, we choose the input data type. Next, we prepare the data using the sliding window algorithm. After that, we train our models on the prepared data and test predictions for the last ten days in the end. The top boxes in the figure indicate that experiments will separately be performed on two types of input data, univariate- containing only one parameter, 'Readiness' and multivariate time series with five parameters 'Readiness', 'Mood', 'Stress', 'Soreness', 'Fatigue'. The middlebox in the figure means that we will prepare our input data using a different sliding window size. And the lowest box in the figure indicates that testing of

predictions of readiness to play will be performed on two types of test data. One test data set will contain gaps filled with zeros, indicating missing registrations. The second test data set will be without gaps. At all steps of our research, except for testing models with the best prediction results, we will use data from PMSys for only one player. It will allow us to execute more experiments and get more analyses at the initial and intermediate stages.

4.2.2 Results for the case with univariate time series

In the initial experiments, we tested our 13 models (see table A.1). For these experiments we had to set some configurations shown in table 4.1.

Type	Parameter	Value
Data	Number of epochs	200
	Batch size	128
	Sliding window size	3, 5, 7, 14, 21, 28, 35, 42, 49
Other	Univariate time series	'Readiness'
	Number of input data	914
	Gaps filled with '0'	yes
	Test data with gaps filled with '0'	yes
	Test data without gaps filled with '0'	yes

Table 4.1: Configurations for initial experiments. Univariate time series.

Initial values of the parameters batch size and number of epochs were chosen based on examples of experiments found in the documentation [1] to the 'Tsai' library. It was planned to correct the number of epochs for subsequent experiments. As for the sliding window size, it was initially decided to consider as many values as possible to determine the impact of this parameter on the final result.

Also, it was decided to use the data of only one player for the initial and intermediate experiments to have time to execute as many experiments as possible. This can be a significant advantage on one side. This data belongs to the same person and is more homogeneous. We also used two kinds of

[1] https://github.com/timeseriesAI/tsai/tree/main/tutorial_nbs

test data, with and without gaps, to see how our models could handle these cases.

Model	MSE	Model	MSE
InceptionTime	1.191	**MGRU-FCNPlus**	1.280
MLSTM-FCNPlus	1.299	**RNN-FCNPlus**	1.321
LSTM	1.331	**MRNN-FCNPlus**	1.350
GRU-FCNPlus	1.369	**LSTM-FCNPlus**	1.409
LSTMPlus	1.480	**GRUPlus**	1.763

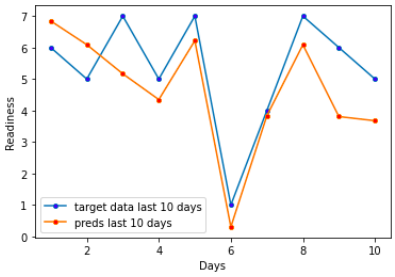
Table 4.2: Models with their MSE values after initial experiments. Sliding window size 3. Univariate time series.

After executing the initial experiments we got a lot of results which one can find in the table A.1. We analyzed the results and selected the best 10 models for further experiments (table 4.2), based on the value of the MSE metric. MSE measures the average sum of the square of the difference between the actual value and the predicted value for all data points. This metric has the advantage of emphasizing big mistakes over more minor ones.

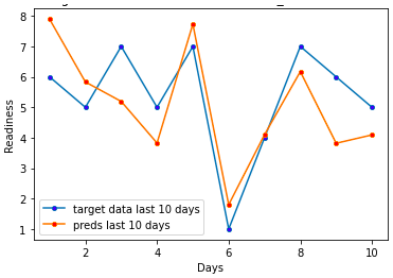
Now we can make some conclusions based on the results of these initial experiments. First, as it turned out, all the best results were obtained using the following values of these parameters: for a sliding window of size three and test data without gaps/'0'. The results were much worse for the other sliding window sizes and test data with gaps/'0'. Also, the chosen value of the number of epochs was suitable for almost all models, except for the GRU. In figure 4.9, one can see the graphical approximation of the obtained predictions compared to the ground truth for the best ten results. The results look promising since we didn't use much data to train the models. As we can see from the figure, almost all models (except for GRUPlus), with a slight deviation, could reproduce the ground truth contour quite well. Some models also were able to predict positive and negative peaks successfully. Especially the InceptionTime, MGRU-FCNPlus, MLSTM-FCNPlus, RNN-FCNPlus, MRNN-FCNPlus, GRU-FCNPlus, and LSTM-FCNPlus models managed this pretty well.

4.2.3 Results for the case with multivariate time series

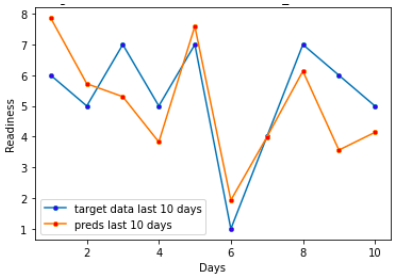
We used almost the same parameters' values indicated in table 4.1 to carry out the initial experiments for multivariate case. The difference was only in the input data type. In this case, we used not univariate but



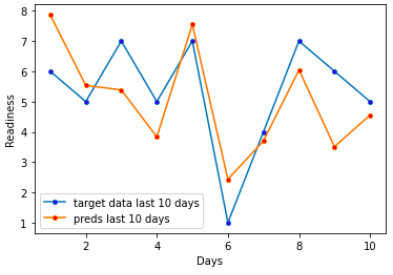
(a) InceptionTime, MSE = 1.191



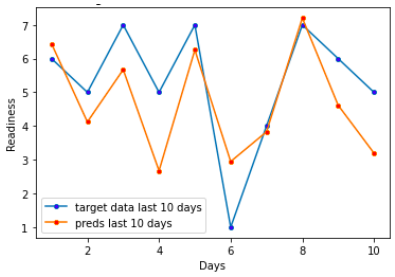
(b) MGRU-FCNPlus, MSE = 1.280



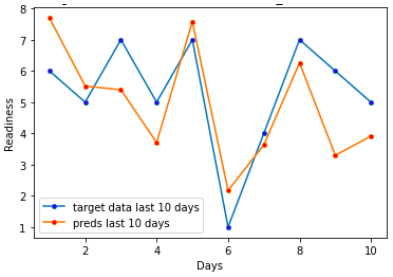
(c) MLSTM-FCNPlus, MSE = 1.299



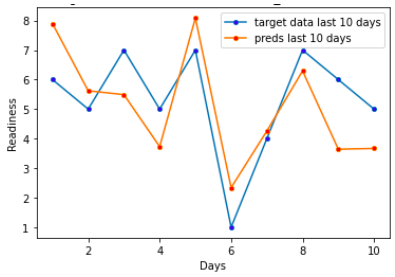
(d) RNN-FCNPlus, MSE = 1.321



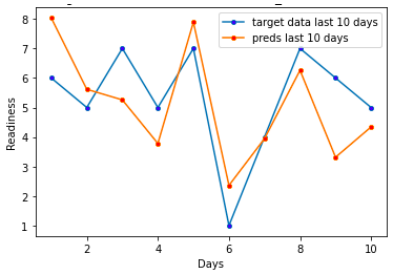
(e) LSTM, MSE = 1.331



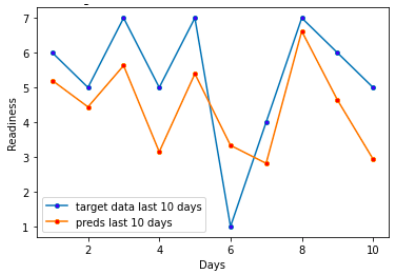
(f) MRNN-FCNPlus, MSE = 1.350



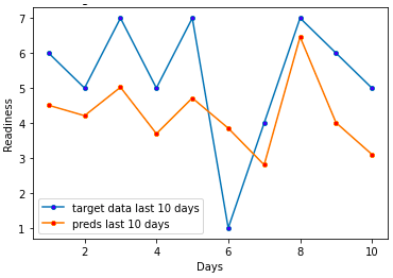
(g) GRU-FCNPlus, MSE = 1.369



(h) LSTM-FCNPlus, MSE = 1.409

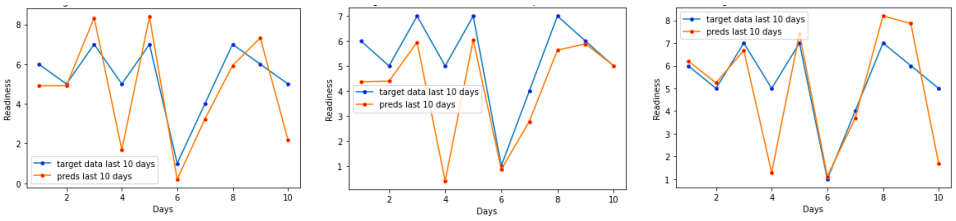


(i) LSTMPlus, MSE = 1.480



(j) GRUPlus, MSE = 1.763

Figure 4.2: Results of the initial experiments. Best models. Window length 3. Univariate time series for one player. Test data without '0'.



(a) LSTMPlus, MSE = 1.668 (b) InceptionTime, MSE = 1.721 (c) LSTM, MSE = 1.735

Figure 4.3: Results of the initial experiments. Best models. Window length 3. Multivariate time series for one player. Test data without '0'.

multivariate times series consisting of these five parameters: 'Readiness', 'Soreness', 'Mood', 'Stress', 'Fatigue'. The 'target' parameter we wanted to predict was 'Readiness'.

Model	**MSE**
LSTMPlus	1.668
InceptionTime	1.721
LSTM	1.735

Table 4.3: Best models after initial experiments. Multivariate time series.

Results of MSE for all initial experiments using multivariate time series are stored in table A.2. Generally, the results were much worse compared to the univariate case, and we chose only three neural networks shown in table 4.3 with MSE value below 2. It turned out that the best results for multivariate and univariate time series were obtained for a sliding window of size three and test data without gaps / '0'. However, unlike the initial experiment with univariate time series, the value of the number of epochs played a significant role in the initial research with multivariate one. Initially, we used 200 epochs, and many of the models were overfitted already after 20 epochs. Based on this, we performed further experiments for uni- and multivariate cases, in which we corrected the value of the number of epochs for some models and presented the results in the next section.

In figure 4.3, one can see the graphical approximation of the obtained predictions compared to the ground truth for the best three results. Again, as in the previous case with the univariate time series, InceptionTime was among the best models. Also, if we look at the plots in figure 4.3, we can see that, overall, the predictions turned out to be very good. The LSTMPlus and InceptionTime predictions were close to ground truth, except for value number 4, which is very different from ground truth and significantly affected the MSE value.

4.3 Experiments with corrected number of epochs

4.3.1 Overview

The cycle when all input data passes through a neural network in the forward direction and an error propagates in the opposite direction and changes all the neural network's weights on all data is called one epoch. Usually, at least 30 - 50 epochs are recommended for simple regression and classification problems. The number of epochs can reach several tens of thousands for image classification.

The data is usually divided into training and validation sets to train the neural network. An error is calculated for both data sets during training. Usually, at the beginning of the training process, the error value for training and validation sets decreases, but after a certain number of epochs, the error calculated using the validation set may start to move up. This is a clear sign that the system is being overfitted. Otherwise, if the training process stops when errors for training and validation sets still decrease, this is an indicator that the system is underfitted.

Therefore, choosing the optimal number of epochs for training is essential for better results when testing the model. This hyperparameter is often challenging to select right away in the initial experiment, and it needs to be adjusted later. Choosing the more appropriate number of epochs is a common practice, and we will also figure out the optimal value of this hyperparameter for different types of input data in our experiments. This value can differentiate a lot while using uni- and multivariate time series.

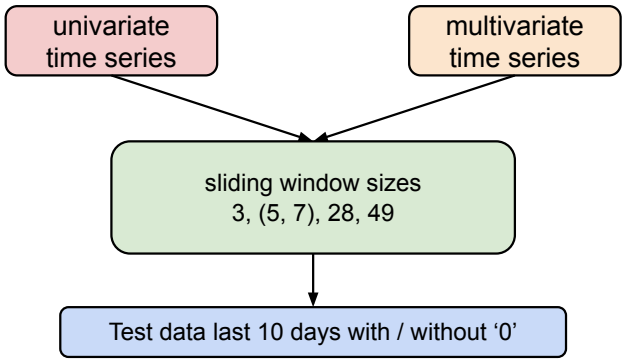


Figure 4.4: Scheme for experiments with corrected number of epochs.

In figure 4.4, one can see how this experiment will be executed. After the initial experiments, we will select underfitted or overfitted models for both univariate and multivariate time series and use the upper, lower, and average sliding window sizes to rerun the experiments. Then compare the results of prediction of readiness with ones from the initial experiments. If, for example, the new results for models with a sliding window size of 3 are improved, we will run more experiments for the adjacent window sizes; in this case, these are sizes 5 and 7. Testing of readiness to play

predictions will be performed in the same way as it was performed for the initial experiments.

4.3.2 Results for the case with univariate time series

As described in the previous section, we selected 200 epochs for our initial experiments. In the case of the univariate time series, this value turned out to be suitable for most models, only for the GRU model it was necessary to correct this value. According to the results of the loss function graph, it was seen that after 100 epochs, the error began to increase. Therefore, we repeated the initial experiment for the GRU model with all the parameters from table 4.1, only reducing the number of epochs from 200 to 100. The prediction results for this model improved significantly,

Model	**MSE**	**Model**	**MSE**
InceptionTime	1.191	**MGRU-FCNPlus**	1.280
MLSTM-FCNPlus	1.299	**RNN-FCNPlus**	1.321
LSTM	1.331	**MRNN-FCNPlus**	1.350
GRU	1.375	**GRU-FCNPlus**	1.369
LSTM-FCNPlus	1.409	**LSTMPlus**	1.480

Table 4.4: Best models after corrected number of epochs. Univariate time series.

and the list of the ten best neural networks after the initial tests is presented in table 4.4).

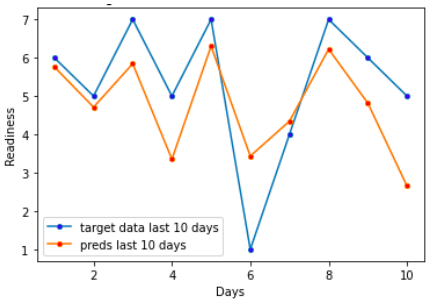


Figure 4.5: Results for GRU. Window length 3. Univariate time series for one player after correction of number of epochs. Test data without '0'. New MSE = 1.350

In figure 4.5 is shown the graphical approximation of the improved

predictions for the GRU model. As we can see from this figure, the prediction line, with a slight deviation, but rather well repeats the ground truth contour, especially evident in the positive peaks. The only rather serious deviation occurred in the prediction of the sixth value. Here it is the largest. As we can see from the ground truth values, there was a relatively sharp drop from seven to one, and the next day the readiness to play returned again to seven. It is not entirely clear how this difference can be explained. If this happened for health reasons, then the player quickly recovered to the previous level of readiness. If it was for some other non-health reason, it most likely should be classified in some other way. We see that this model did not cope with this drop in our case.

4.3.3 Results for the case with multivariate time series

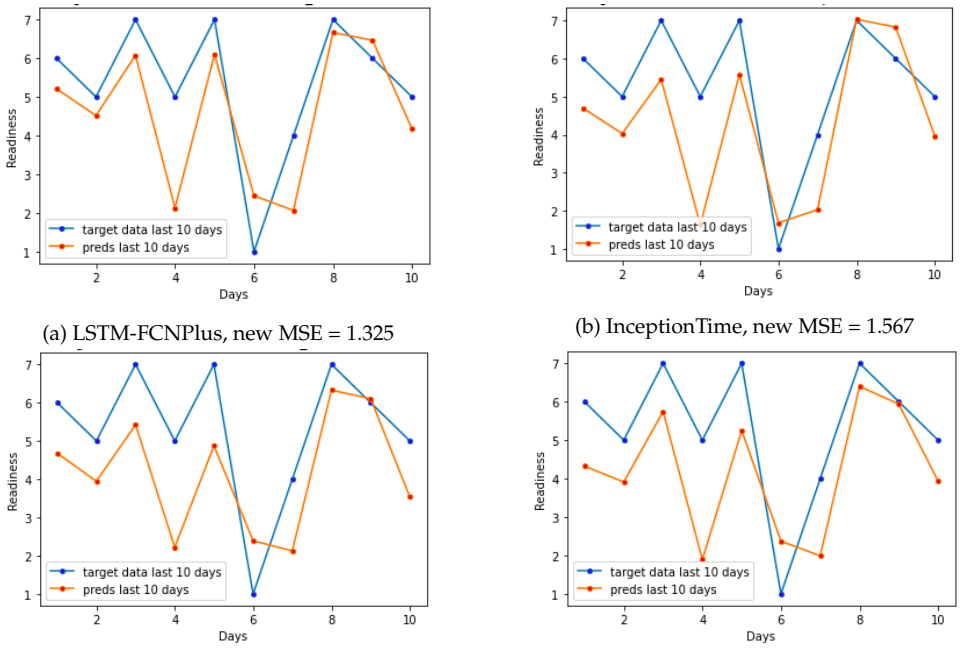
After doing the initial experiments for the multivariate time series, 200 epochs turned out to be very many and influenced the results of predictions. As the plots of the loss function for various models showed, many of them were already trained after 20 epochs. Therefore, we decided to correct the number of epochs for all models and rerun the initial experiments with sliding window sizes of 3, 5, 7, 28, and 49.

Model	**MSE**	**Model**	**MSE**
LSTM-FCNPlus	1.325	**InceptionTime**	1.567
GRU-FCNPlus	1.600	**MLSTM-FCNPlus**	1.603

Table 4.5: Best models after corrected number of epochs. Multivariate time series.

The results for some models have improved significantly after adjusting the number of epochs. Table 4.5 shows the updated data for the models with the best MSE. Comparing the data in tables 4.3 and 4.5, we can conclude that only InceptionTime was among the best models from the old list after adjusting the number of epochs. Also, the best were LSTM-FCNPlus, GRU-FCNPlus, and MLSTM-FCNPlus, which after the initial experiments, showed more modest results.

Figure 4.10 shows the improved approximation after the correction of the number of epochs. From this figure, one can see that the position of the prediction peaks is similar to the peaks of ground truth values but with relatively large deviations. The LSTM-FCNPlus model showed the best MSE. Also, this model obtained the best match among all other models from the table 4.5 in almost all peak values between predictions and ground truth.



(a) LSTM-FCNPlus, new MSE = 1.325

(b) InceptionTime, new MSE = 1.567

(c) GRU-FCNPlus, new MSE = 1.600

(d) MLSTM-FCNPlus, new MSE = 1.603

Figure 4.6: Results after corrected number of epochs. Best models. Window length 3. Multivariate time series for one player. Test data without '0'.

4.4 Discussion of the intermediate results

To this point, we have carried out initial experiments; as a result, we can already draw some intermediate conclusions. During the execution of the initial experiments, we were able to evaluate the influence of such parameters as the number of epochs and the size of the sliding window on the results obtained. We were also able to tentatively assess the impact of the input data type such as uni- and multivariate time series. In addition, we looked at how the type of test data affects predictions depending on whether it has gaps filled with '0' or not. Now we will describe which of the parameters ultimately led to the best result. Recall that the only data of one player was used for both training and testing up to this point. Also, this data was with gaps, which we decided to fill in with '0'. So, we have achieved the best result:

- with a sliding window size of 3 for all models that showed the best outcome.
- MSE for the top ten models with univariate time series was within 1.191 -:- 1.480, while MSE for only four best models with multivariate time series is within 1.325 -:- 1.603, which means using the univariate time series was more efficient.
- A higher value of the number of epochs (for most it was about 200), turned out to be more effective for training models using

the univariate time series, while for the multivariate time series, a lower value of epochs, about 20, turned out to be more effective; at higher values than 20, the models for the multivariate case became overfitted.

- All the best results were obtained using the test set without gaps filled with '0'.

In the next section, we will continue our research using the training and testing data for one player with removed gaps.

4.5 Experiments with the best models and one player's data without gaps filled with '0'

4.5.1 Overview

After experimenting with some corrected hyperparameters, we selected the models that showed the best predictions. We used data from one player in previous experiments, but this data had gaps in registrations that we filled with '0'. These gaps were more chaotic since it is inexplicable for what reasons the player skipped data registration. The length of the non-recording period was also challenging to explain. In previous experiments, we tested our models with different test data sets. One set was with gaps filled with '0', and another one was without them. All the best results have been obtained using the test data without gaps. Therefore, it will now be interesting to see how different the results will be if we run experiments on input data with removed '0' removed.

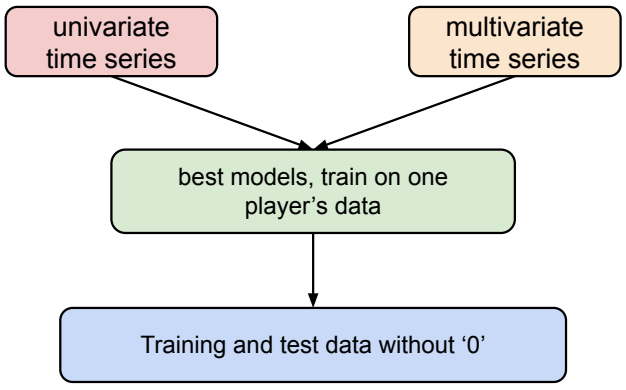


Figure 4.7: Scheme for experiments with the best models and one player's data without gaps filled with '0'.

The process of carrying out these experiments will look like this. First, we need to prepare the input data. To do this, we take the data of the same player as in the previous experiments and remove all gaps from them. Then we will use the best models we got using the uni- and multivariate time series. We train the selected models on the prepared data and predict the readiness. Figure 4.7 shows the described flow of these experiments.

4.5.2 Description of the results for the cases with uni- and multivariate time series

As it was said in the overview of the experiments described in this section, we decided to continue the research using the data of the same player as before, but with removed gaps. We continued using the models from tables 4.4 and 4.5 that performed best for the uni- and multivariate time series. Since we changed some parameters after the initial experiments, it would be right to reflect these updates in configurations in table 4.6.

Type	Parameter	Value
Data	Number of epochs, univariate	100 (for GRU) 200 (for other models)
	Number of epochs, multivariate	20 (for all models)
	Batch size	128 (for all models)
	Sliding window size	3 (for all models)
Other	Univariate time series	'Readiness'
	Multivariate time series	'Readiness', 'Soreness', 'Mood', 'Stress', 'Fatigue'
	Number of input data	468
	Gaps filled with '0'	no

Table 4.6: Updated configurations, were used for the experiments with the best models from tables 4.4 and 4.5.

After removing all the gaps before the experiments, the input data size was reduced from 914 to 468 units. By the standards of machine learning, this amount of data is too small, and it could be affecting the results obtained in experiments.

The results of these experiments for uni- and multivariate time series are shown in tables 4.7 and 4.8. As we can see from both tables, the MSE score got worse for both univariate and multivariate cases. Nevertheless, it remained much lower than the MSE indicator obtained by some neural networks with other sliding window sizes in the initial experiments (tables A.1 and A.2).

Model	MSE	Model	MSE
LSTM	1.381	**LSTMPlus**	1.482
GRU	1.573	**InceptionTime**	1.709
MGRU-FCNPlus	1.745	**MRNN-FCNPlus**	1.756
RNN-FCNPlus	1.764	**MLSTM-FCNPlus**	1.773
LSTM-FCNPlus	1.820	**GRU-FCNPlus**	1.916

Table 4.7: Best models with MSE values. Data for one player without gaps. Uniivariate time series.

It is rather challenging to say with certainty what could have led to this result, but with a very high degree of probability, it can be assumed that the reason for this was precisely the lack of data. We can check this assumption

Model	MSE	Model	MSE
LSTM-FCNPlus	1.622	**MLSTM-FCNPlus**	1.645
GRU-FCNPlus	1.730	**InceptionTime**	1.798

Table 4.8: Best models with MSE values. Data for one player without gaps. Multivariate time series.

by executing experiments with the increased amount of data for training and testing of the best models. The results of these experiments will be presented in the next section.

4.6 Experiments with the best models and data for all players with / without '0'

4.6.1 Overview

Up to this point, we did our research with the data of only one player, which consisted of 914 points. We selected two lists of models that showed the best MSE values for uni- and multivariate cases. Now, we will carry out final experiments in this thesis to test the best models on more inputs. General scheme of these experiments is shown in figure 4.8.

The previous section described the results obtained using data without gaps for one player. These results turned out to be worse than the initial ones. We assumed that the reason for that was most likely the small amount of input data after removing gaps filled with '0', only 468 data points.

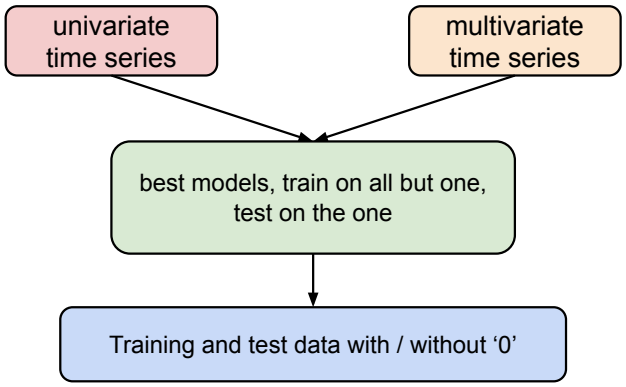


Figure 4.8: Scheme for experiments with the best models and data for all players with / without gaps filled with '0'.

For further experiments, we decided to use the data of all players. To increase the amount of points in training and test data sets, we applied the strategy from this paper [4]. We concatenated data of all but one player with gaps filled with '0' to train our models. To test the accuracy of the predictions of the readiness to play, we used the data from the remaining player. These experiments are described later in this section. For the experiments described in the next section, we removed the gaps from the data. As before, we considered both uni- and multivariate cases.

4.6.2 Description of the results

In this subsection we will describe the results of the experiments with the best models and data with gaps for 'training on all, but one and predicting on the one' case. The overall outcome we got when executing experiments on the principle of 'training on all, but one and predicting on the one' is very similar to the one we described in the previous section. 4.5.2. The results of the MSE value that we can see in tables 4.9 and 4.10 are very close to each other but far below our expectations. It looks like the

Model	MSE	Model	MSE
GRU-FCNPlus	3.351	**LSTMPlus**	3.352
MGRU-FCNPlus	3.354	**LSTM-FCNPlus**	3.357
RNN-FCNPlus	3.360	**MLSTM-FCNPlus**	3.360
MRNN-FCNPlus	3.376	**GRU**	3.377
LSTM	3.388	**InceptionTime**	4.700

Table 4.9: Best models with MSE values. Training on all but one and testing on the one. Data with gaps. Univariate time series.

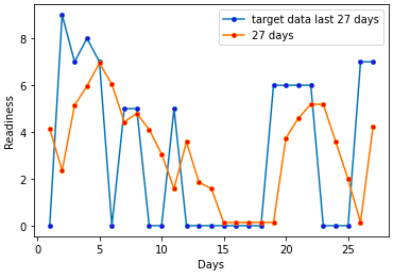
reason is the same for both uni- and multivariate time series. Therefore, we decided to combine the description of the results into one subsection. All plots showed a very bad approximation, which we can see in figures 4.9 and 4.10.

As we can see from tables 4.9 and 4.10, MSE values of most of the models are in the range of 3.235 -:- 3.377, which indicates that these models handle the data with gaps that we used in these experiments with almost the same accuracy. In figure 4.9, all the plots showed very similar outlines of predicted values. We can say the same about the plots in figure 4.10, where all the plots look very similar except for the InceptionTime model, which generally showed better results.

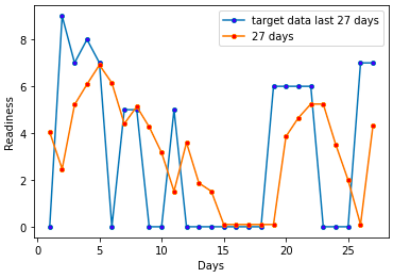
Model	MSE	Model	MSE
LSTM-FCNPlus	3.235	**MLSTM-FCNPlus**	3.238
GRU-FCNPlus	3.250	**InceptionTime**	3.252

Table 4.10: Best models with MSE values. Training on all but one and testing on the one. Data with gaps. Multivariate time series.

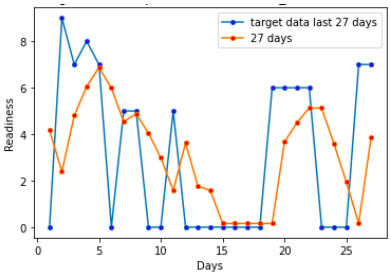
In total, after concatenation, we obtained about 31,000 data instances.



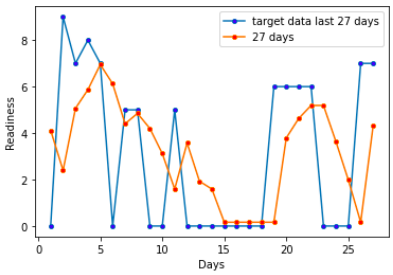
(a) GRU-FCNPlus, MSE = 3.351



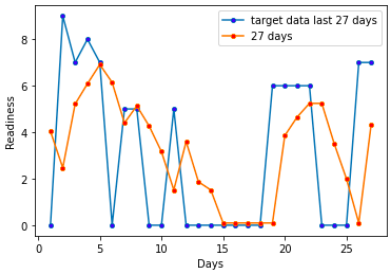
(b) LSTMPlus, MSE = 3.352



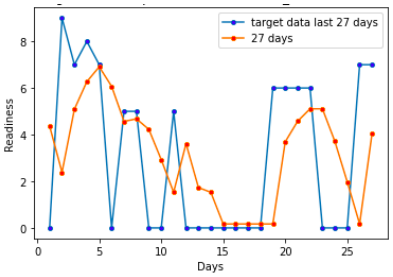
(c) MGRU-FCNPlus, MSE = 3.354



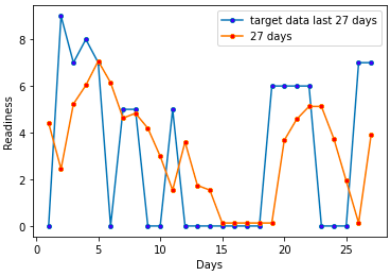
(d) LSTM-FCNPlus, MSE = 3.357



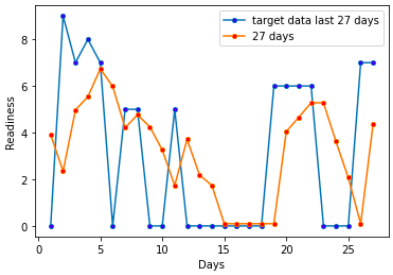
(e) RNN-FCNPlus, MSE = 3.360



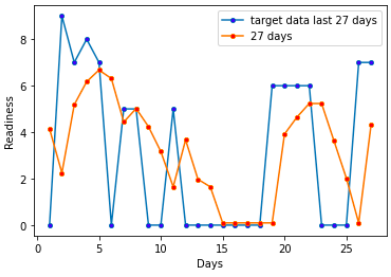
(f) MLSTM-FCNPlus, MSE = 3.360



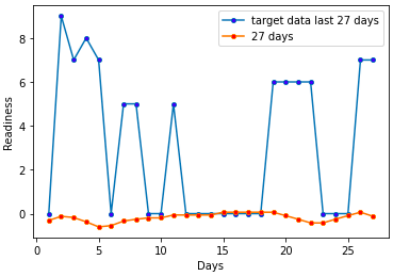
(g) MRNN-FCNPlus, MSE = 3.376



(h) GRU, MSE = 3.377

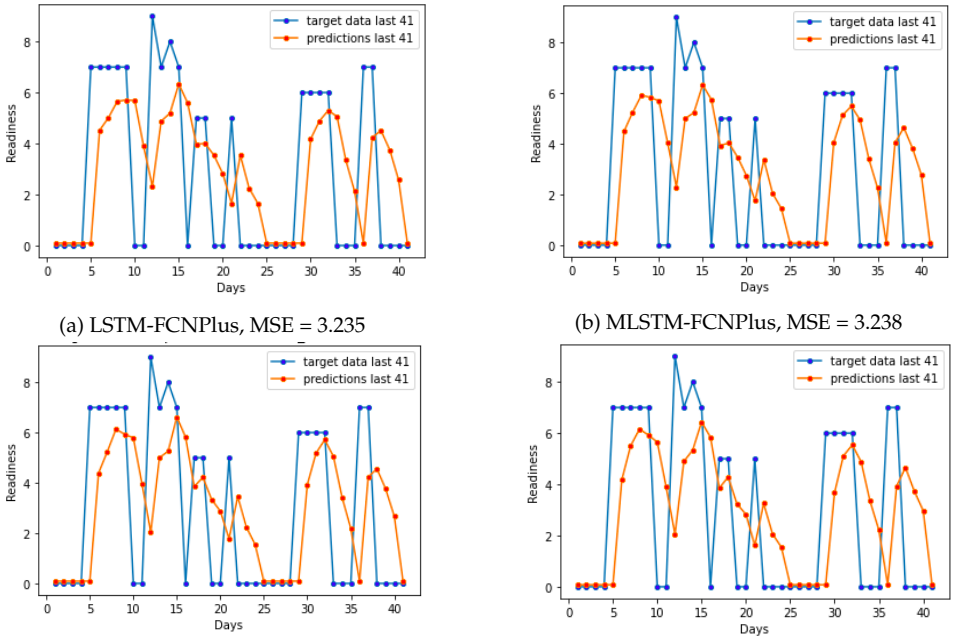


(i) LSTM, MSE = 3.388



(j) InceptionTime, MSE = 4.700

Figure 4.9: Results for the 'all but one player' case. Best models. Window length 3. Univariate time series. Data with gaps filled with '0'.



(a) LSTM-FCNPlus, MSE = 3.235

(b) MLSTM-FCNPlus, MSE = 3.238

(c) GRU-FCNPlus, MSE = 3.250

(d) InceptionTime, MSE = 3.252

Figure 4.10: Results for the 'all but one player case'. Best models. Window length 3. Multivariate time series. Data with gaps filled with '0'.

Only about 7000 of these data were registrations. The rest of the data turned out to be gaps. Moreover, these gaps were located in different places without any particular order. Also, the length of the gaps was chaotic. In this case, it can be assumed that our models did not cope with their prediction due to the not entirely clear reason for the appearance of these gaps and due to their chaotic duration. Although during the training process, the training loss and validation loss values were minimal and promising.

Even though we did not get good predictions for the readiness to play from the experiments described in this section, the results allowed us to gain a deeper insight into the ability of the models to handle such chaotic gaps. The appearance of these gaps in data registration in PMSys should be understandable and more controllable. At the moment, there is no clear understanding of how these gaps must be properly handled. First of all, we need to understand how to classify them. This analysis could be one of the topics for further research on the processing of data from PMSys and the key to creating more accurate models for predicting the state of athletes. Now we turn to the description of the last type of experiments in this thesis, namely experiments with the best models and data without gaps for 'training on all, but one and predicting one' case.

4.6.3 Description of the results

In this subsection we will describe the results of the experiments with the best models and data without gaps for 'training on all, but one and

predicting on the one' case. As we saw in the previous section, our best models failed to handle the gaps in data registration in PMSys. We have suggested that these gaps are chaotic and should be more predictable to handle them more successfully. This section presents the results of recent experiments in this thesis. These experiments are a continuation of the experiments from the previous section. The only difference here is that we will use concatenated data with gaps removed in advance. The size of the data without gaps is about 7000 units. We also tested the predictions of our models on the last 60 values from the initial data set. As we can see from tables 4.11 and 4.12, the MSE values for both uni- and multivariate time series are very close, so we can try to combine the presentation of the results of these experiments.

As noted above, the results of the MSE values for all the best models used in the latest experiments can be found in tables 4.11 and 4.12. The approximation of the prediction results of these models can be seen in figures 4.11 and B.3. The first thing to notice is that the MSE values from the tables are very close to each other within the range 1.203 -:- 1.486.

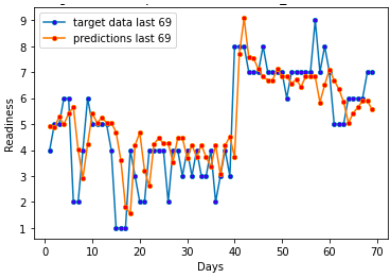
Model	**MSE**	**Model**	**MSE**
MRNN-FCNPlus	1.346	**InceptionTime**	1.349
LSTM-FCNPlus	1.351	**MGRU-FCNPlus**	1.352
GRU-FCNPlus	1.354	**MLSTM-FCNPlus**	1.362
RNN-FCNPlus	1.365	**GRU**	1.414
LSTM	1.424	**LSTMPlus**	1.486

Table 4.11: Best models with MSE values. Training on all but one and testing on the one. Data without gaps. Univariate time series.

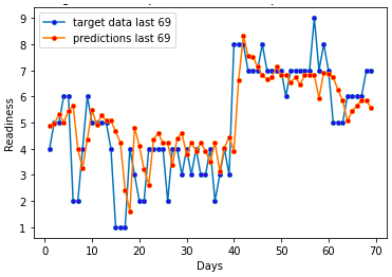
This means that the models processed the input data and made predictions with approximately the same accuracy. One can also observe in the figures that the prediction lines for readiness to play are pretty close to the ground truth lines.

Another point worth noting is that this was the first experiment in which the case with multivariate time series, based on the values of the MSE metric, performed better than univariate. Further, comparing the results obtained when processing concatenated input data, we can conclude that with the absence of gaps, the final result is several times better than in their presence.

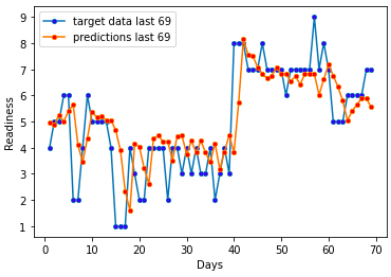
Now we can summarize the entire sequence of experiments that we went through. The main task in these experiments was to apply our chosen models to processing data from PMSys. The models that we have chosen



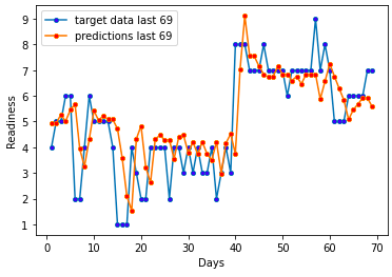
(a) MRNN-FCNPlus, MSE = 1.346



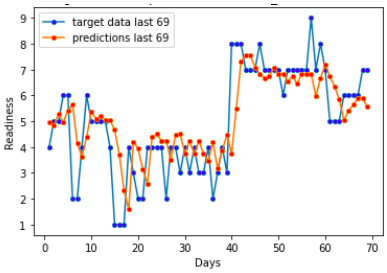
(b) InceptionTime, MSE = 1.349



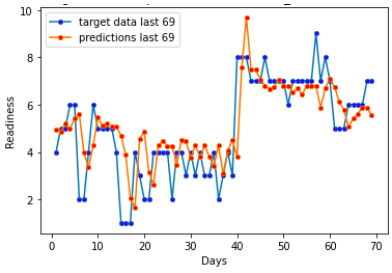
(c) LSTM-FCNPlus, MSE = 1.351



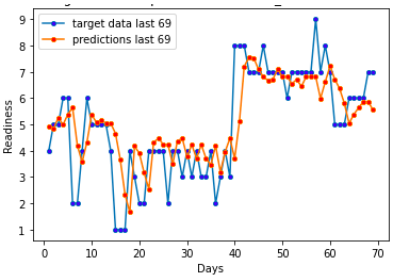
(d) MGRU-FCNPlus, MSE = 1.352



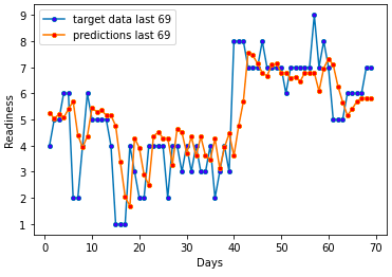
(e) GRU-FCNPlus, MSE = 1.354



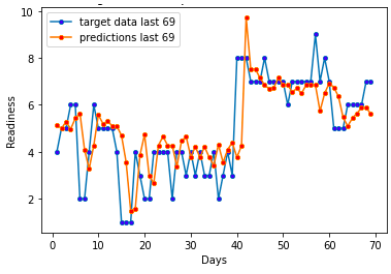
(f) MLSTM-FCNPlus, MSE = 1.362



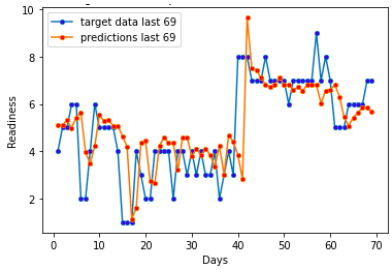
(g) RNN-FCNPlus, MSE = 1.365



(h) GRU, MSE = 1.414



(i) LSTM, MSE = 1.424

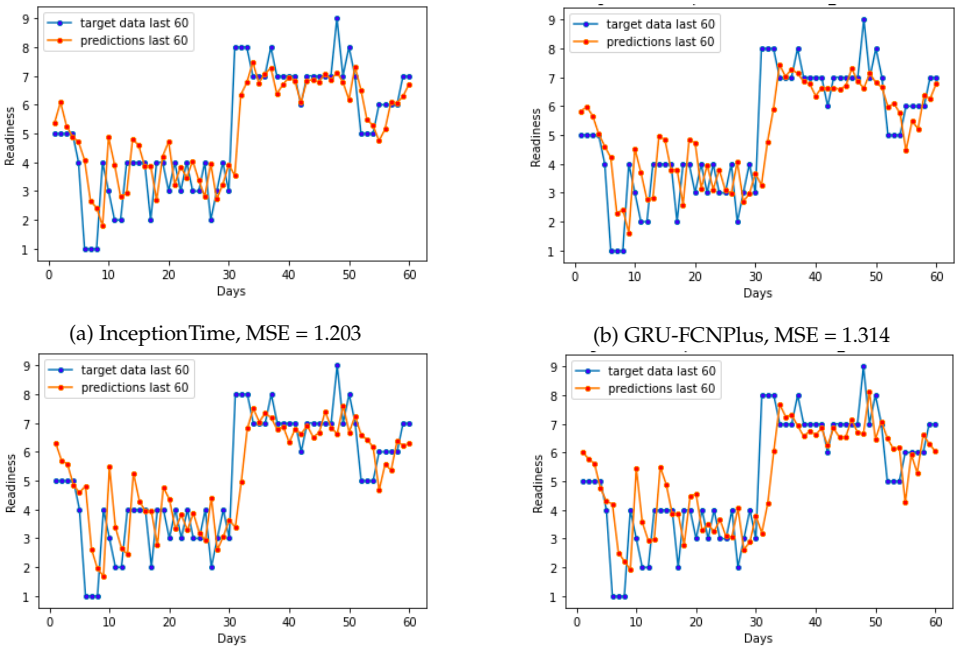


(j) LSTMPlus, MSE = 1.486

Figure 4.11: Results for the 'all but one player' case. Best models. Window length 3. Univariate time series. Data without gaps filled with '0'.

Model	MSE	Model	MSE
InceptionTime	1.203	**GRU-FCNPlus**	1.314
LSTM-FCNPlus	1.350	**MLSTM-FCNPlus**	1.376

Table 4.12: Best models with MSE values. Training on all but one and testing on the one. Data without gaps. Multivariate time series.



(a) InceptionTime, MSE = 1.203

(b) GRU-FCNPlus, MSE = 1.314

(c) LSTM-FCNPlus, MSE = 1.350

(d) MLSTM-FCNPlus, MSE = 1.376

Figure 4.12: Results for the 'all but one player' case. Best models. Window length 3. Multivariate time series. Data without gaps filled with '0'.

were taken from a library unknown to us before. At the initial stage, we set the task of obtaining the first results, analyzing which and changing the parameters of the experiments to arrive at the best outcomes. The next step in this thesis will be to evaluate the performance of the best models and their final comparison based on the performance results and readiness to play predictions.

4.7 Results of performance measurement

Parameter	Univariate case	Multivariate case
Number of data points/ training	7100	7100
Number of data points/ testing	72	63
Sliding window size	3	3
Number of epochs	100 for GRU/ 200 for other models	20
Batch size	128	128

Table 4.13: Configurations used for performance measurement.

Table 4.13 shows the values of the main parameters at which the performance was measured. As one can see from the table, the measurement was executed for uni- and multivariate cases. We did measurements for the best models with a sliding window size of 3 and a batch size of 128. For all models for the multivariate case, the number of epochs was 20, and for the univariate case, for all models except GRU, the number of epochs was 200, for GRU 100. Training data set for both uni- and multivariate cases consisted of 7100 data points, but the test data set for the univariate case was 72 and for multivariate case 63. The following two tables show the results of performance measurements.

Model	MSE	Training time (sec)	Testing time for 10 repetitions (sec)	CPU used min/max (%)	Memory used min/max (GB)	Disk used min/max (GB)
InceptionTime	1.203	46	0.295	80/100	17.875/ 17.895	34.048/ 34.050
GRU-FCNPlus	1.314	111	0.204	85/100	17.772/ 17.787	34.0534/ 34.0546
LSTM-FCNPlus	1.350	141	0.203	75/100	17.762/ 17.784	34.0502/ 34.0514
MLSTM-FCNPlus	1.376	147	0.240	75/100	17.690/ 17.800	34.0550/ 34.0575

Table 4.14: Performance measurement. Best models. Multivariate time series. 'Training on all but one and testing on the one' case.

In tables 4.14 and 4.15, along with the main performance metrics, we

have indicated the MSE value for each model to easily compare these models in terms of prediction and measurement at the same time. Before analyzing these results, it is worth recalling that the computer had the following hardware characteristics, which are indicated in the table:

- CPU Intel Core i7-9700, 8 cores,
- memory DDR4, 46.9 GiB,
- disk SSD 491.2 GiB.

Table 4.14 shows the performance results for the best models for the multivariate case. Plots of these measurements for multivariate case can be seen in figure B.3. As we can see from table 4.14, the InceptionTime model

Model	MSE	Training time (sec)	Testing time for 10 repetitions (sec)	CPU used min/max (%)	Memory used min/max (GB)	Disk used min/max (GB)
MRNN-FCNPlus	1.346	381	0.370	45/100	18.450/ 18.540	34.0400/ 34.0480
InceptionTime	1.349	554	0.539	60/100	18.130/ 18.200	34.0600/ 34.0675
LSTM-FCNPlus	1.351	341	0.324	50/100	18.330/ 18.400	34.0440/ 34.0540
MGRU-FCNPlus	1.352	387	0.306	45/100	18.500/ 18.570	34.0460/ 34.0560
GRU-FCNPlus	1.354	341	0.308	50/100	18.390/ 18.470	34.0575/ 34.0450
MLSTM-FCNPlus	1.362	452	0.391	50/100	18.450/ 18.720	34.0620/ 34.0450
RNN-FCNPlus	1.365	312	0.326	50/100	18.310/ 18.370	34.0535/ 34.0415
GRU	1.414	81 (after 100 epochs)	0.235	60/100	18.185/ 18.210	34.0420/ 34.0420
LSTM	1.424	209	0.227	50/100	18.190/ 18.400	34.0350/ 34.0380
LSTMPlus	1.486	216	0.261	60/100	18.210/ 18.270	34.0440/ 34.0510

Table 4.15: Performance measurement. Best models. Univariate time series. 'Training on all but one and testing on the one' case.

showed the best training time, which turned out to be several times less than the time for other models. A testing time for all models turned out to be approximately the same, and we see that testing proceeded very quickly. To calculate the CPU usage, we used a utility that measured the utilization for each core separately. The CPU utilization was measured only in the process of training the models. As a result, it turned out that the usage

increased several times at the start of the training process and remained at the level of 75-:-100 percent for all models. After the end of the training, the CPU utilization also immediately decreased. All this is very clearly visible in figure B.3. The values of memory and disk usage in the training process turned out to be insignificant. The minimum and maximum amount of memory and disk usage differed by about a few hundred MB. Graphs of these metrics can also be viewed in figure B.3.

Table 4.15 shows the performance results for the best models for the univariate case. Plots of these measurements for the univariate case can be seen in figures B.1 and B.2. As one can see from the table, the LSTM model spent the least time on training. The training time of the GRU model turned out to be the lowest since we used 100 epochs when training this model. It can also be noted that if for the multivariate case, in terms of the time used for training, InceptionTime turned out to be the leader, and its time was noticeably lower than for other models, then for the univariate case for training InceptionTime, on the contrary, the most time was spent.

The CPU usage for the univariate case is very similar to that for the multivariate case. CPU usage increased several times at the start of the training process and remained at 65-:-100 percent for all models. After the end of the training, the CPU utilization also immediately decreased. All this is very clearly visible in figures B.1 and B.2. The values of memory and disk usage in the training process turned out to be insignificant, like for the multivariate case. The minimum and maximum amount of memory and disk usage differed by about a few hundred MB. Graphs of these metrics can also be viewed in figures B.1 and B.2.

4.8 Discussions

We selected 13 models from the new machine learning library 'Tsai'. We used these models to execute experiments to predict readiness to play using data from PMSys. This data contained registration gaps, which we decided to fill in with '0' and see what kind of results we can get if we train and test our models on data sets with and without gaps. During initial experiments, many models performed poorly. All the best results were obtained using a sliding window size of 3 for both uni- and multivariate cases. We also noticed that many multivariate case models were overfitted already after 20 epochs. The InceptionTime model performed best and did a perfect job of predicting peaks. Models such as MGRU-FCNPlus, MLSTM-FCNPlus, and RNN-FCNPlus also showed similar results. In total, after initial experiments for the univariate case, there were ten models whose MSE indicator was in the range of 1.191-:-1.763. For the multivariate case, the results of the initial experiments were much worse. Only three models showed results comparable to those from the univariate case. However, these models predicted some peaks well. Since many models, especially in multivariate cases, quickly turned out to be overfitted, we decided to carry out the following experiments, adjusting the number of epochs. These experiments did not significantly improve the results of the initial

experiments for the univariate case, but did significantly improve the MSE for some of the multivariate case models. After these experiments, we came to an intermediate conclusion that the best results were obtained for uni- and multivariate cases with a sliding window size of 3, as well as 200 epochs for most models for a univariate case and 20 for a multivariate case. This number of epochs for the multivariate case turned out to be entirely unexpected since it is usually recommended to set the size of this parameter to at least 30-50.

Up to this point, we have used data with gaps filled with '0'. We had an assumption that these gaps could significantly affect the results because these gaps could not be explained in any way. They were chaotic. Therefore, it was further decided to try to remove these gaps from the input data set and experiment with this new data set for the models that showed the best results. Another question arose here. After removing the gaps, the number of data points decreased to 468, which is quite a bit by the standards of machine time. After experimenting with a new data set, our prediction results became much worse than the initial ones for both uni- and multivariate cases. Most likely the reason for this was the small amount of data in the new set. Therefore, we came up with the following thought about increasing the amount of data for training models. Since there was not enough data for one player, we decided to increase the number of data points by concatenating all players into one array. Then we decided to train our models on the data of all players except one and test on the data of this one player. As before, we filled in the '0' gaps and decided to conduct experiments both on data with and without gaps, using the models that showed the best results in previous experiments. In the end, we did not improve the prediction results on data with gaps, but on data without gaps, we repeated the best results obtained after the initial experiments. Approximations of predictions on plots were also promising after these last experiments. We also measured the performance parameters of our models. We measured CPU, memory, disk usage, and time for training our models. As the results of our measurements showed, the CPU load in most cases reached 100 percent during the training of the models. What turned out to be quite interesting is that memory and disk resources were used in small amounts. The most indicative metric was the time spent on training.

4.9 Summary

In section 4.1, we presented a list of parameters that were of our interest in terms of changing the results during the experiments due to these parameters. In section 4.2, we described how the initial experiments went, specifying all cases and parameters. Next, we presented the results and described them in detail for uni- and multivariate cases. We made the first intermediate conclusions based on the results obtained and proceeded to the following experiments.

Section 4.3 described how the experiments proceeded after adjusting

the number of epochs. After that, we presented the results and described them in detail for both uni- and multivariate cases. In section 4.4 we discussed the intermediate results and summarized how the various parameters affected them. The results of many models when using some combinations of parameters turned out to be unacceptable, so we chose those models and varieties of parameters for which the best results were shown and continued to experiment with them.

In sections 4.5 and 4.6 we described how we executed experiments with the models that performed best in previous ones. Next, we have detailed the results for cases with and without gaps filled with '0'. In section 4.7, we summarized and briefly discussed the performance measurements of our models. In section 4.8, we discussed the results obtained throughout the study in more detail. In the next chapter, we will draw conclusions based on the results obtained during the research and provide our suggestions for future work.

Chapter 5

Conclusion and future work

The main goal of this work was to compare different models for processing time series, predict readiness to play using model data and data from PMSys, evaluate the results, and suggest the best models for further research. We were able to find a new machine learning library called 'Tsai' that offers a wide range of machine learning functionality. Also, this library contains a relatively extensive list of ready-made models. We compared ordinary models belonging to the recurrent family with models that are a combination of recurrent-convolutional ones. Then we used these models in our experiments and got pretty good results. We got the best results for the models shown in tables 4.14 and 4.15. In terms of performance, the best model for the multivariate time series turned out to be InceptionTime, and for the univariate time series case, the LSTM model turned out to be. Despite this, many of the results turned out to be quite close to each other, so it will likely be correct to continue the research using all the models from tables 4.14 and 4.15. In this thesis, we have used the default configurations for our models. However, the 'Tsai' library provides the ability to change the configuration of the models depending on the user's needs. Therefore, it will be like one of the proposals for future work. Also, for further research, it would be correct to suggest paying more attention to how gaps in data registrations by PMSys users should be handled or classified.

Bibliography

[1] T. T. Hoang, 'Pmsys: Implementation of a digital player monitoring system,' M.S. thesis, 2015.

[2] C. N. Nguyen, 'Implementation of a digital player monitoring system: Pmsys,' M.S. thesis, 2015.

[3] K. K. Vuong, 'Pmsys: A monitoring system for sports athlete load, wellness & injury monitoring,' M.S. thesis, 2015.

[4] T. Wiik, H. D. Johansen, S.-A. Pettersen, I. Baptista, T. Kupka, D. Johansen, M. Riegler and P. Halvorsen, 'Predicting peek readiness-to-train of soccer players using long short-term memory recurrent neural networks,' in *2019 International Conference on Content-Based Multimedia Indexing (CBMI)*, IEEE, 2019, pp. 1–6.

[5] H. D. Johansen, D. Johansen, T. Kupka, M. A. Riegler and P. Halvorsen, 'Scalable infrastructure for efficient real-time sports analytics,' in *Companion Publication of the 2020 International Conference on Multimodal Interaction*, 2020, pp. 230–234.

[6] R. Adhikari and R. K. Agrawal, 'An introductory study on time series modeling and forecasting,' *arXiv preprint arXiv:1302.6613*, 2013.

[7] R. H. Shumway, D. S. Stoffer and D. S. Stoffer, *Time series analysis and its applications*. Springer, 2000, vol. 3.

[8] S. Fernández, A. Graves and J. Schmidhuber, 'An application of recurrent neural networks to discriminative keyword spotting,' in *International Conference on Artificial Neural Networks*, Springer, 2007, pp. 220–229.

[9] H. Sak, A. W. Senior and F. Beaufays, 'Long short-term memory recurrent neural network architectures for large scale acoustic modeling,' 2014.

[10] X. Li and X. Wu, 'Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition,' in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2015, pp. 4520–4524.

[11] I. Sutskever, O. Vinyals and Q. V. Le, 'Sequence to sequence learning with neural networks,' in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[12] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, 'Empirical evaluation of gated recurrent neural networks on sequence modeling,' *arXiv preprint arXiv:1412.3555*, 2014.

[13] D. Britz, 'Recurrent neural network tutorial, part 4 implementing a gru/lstm rnn with python and theano,' *URL http://www. wildml. com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano*, 2015.

[14] J. Howard and S. Gugger, 'Fastai: A layered api for deep learning,' *Information*, vol. 11, no. 2, p. 108, 2020.

[15] H. I. Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller and F. Petitjean, 'Inceptiontime: Finding alexnet for time series classification,' *Data Mining and Knowledge Discovery*, vol. 34, no. 6, pp. 1936–1962, 2020.

[16] I. Oguiza, *Tsai - a state-of-the-art deep learning library for time series and sequential data*, Github, 2020. [Online]. Available: https://github.com/timeseriesAI/tsai.

[17] S. Hochreiter and J. Schmidhuber, 'Lstm can solve hard long time lag problems,' *Advances in neural information processing systems*, pp. 473–479, 1997.

[18] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, 'Learning phrase representations using rnn encoder-decoder for statistical machine translation,' *arXiv preprint arXiv:1406.1078*, 2014.

[19] F. Karim, S. Majumdar, H. Darabi and S. Chen, 'Lstm fully convolutional networks for time series classification,' *IEEE access*, pp. 1662–1669, 2017.

[20] N. Elsayed, A. S. Maida and M. Bayoumi, 'Deep gated recurrent and convolutional network hybrid model for univariate time series classification,' *arXiv preprint arXiv:1812.07683*, 2018.

[21] F. Karim, S. Majumdar, H. Darabi and S. Harford, 'Multivariate lstm-fcns for time series classification,' *Neural Networks*, vol. 116, pp. 237–245, 2019.

Appendix A

Tables

A.1 Tables from the experiments

	3	5	7	14	21	28	35	42	49
Incept Time	1.191 2.660	3.307 3.428	2.856 2.668	3.753 2.856	3.306 2.988	2.256 2.524	2.840 2.843	2.393 2.780	2.766 2.720
RNN	2.300 2.550	2.209 2.635	2.314 2.796	2.927 3.232	2.164 2.396	1.997 2.444	3.814 3.065	2.067 2.839	3.724 3.561
LSTM	1.331 2.711	2.407 4.187	3.152 3.686	3.866 3.088	2.455 4.003	2.729 3.952	2.908 3.249	3.459 4.635	2.589 2.579
GRU	1.993 2.571	1.708 3.106	2.335 3.299	4.011 3.735	1.938 4.546	4.179 3.455	4.199 3.367	3.371 3.330	3.074 3.884
RNN Plus	2.251 2.571	2.200 2.831	2.172 2.403	3.037 3.246	2.231 2.437	1.994 2.513	3.199 3.793	2.158 2.491	2.580 2.511
LSTM Plus	1.480 2.641	2.779 4.208	2.540 3.580	3.167 3.129	2.432 3.571	2.903 3.234	2.429 2.727	2.304 4.760	3.474 2.659
GRU Plus	1.763 2.636	2.626 3.449	2.462 3.090	3.594 2.111	4.619 2.514	3.279 3.351	3.558 4.321	2.983 2.842	3.246 2.502
RNN FCN Plus	1.321 2.714	3.018 3.878	2.964 3.852	2.248 2.309	3.247 3.272	2.289 2.472	5.320 5.809	5.259 6.692	3.421 4.154
LSTM FCN Plus	1.409 2.744	3.189 4.012	3.258 3.292	2.868 2.350	2.595 3.850	3.160 2.833	3.918 3.959	5.043 5.975	2.585 3.086
GRU FCN Plus	1.369 2.683	3.220 3.813	3.079 3.059	2.628 2.423	2.995 4.362	3.215 3.062	4.607 4.788	4.831 5.606	2.237 2.981
MRNN FCN Plus	1.350 2.755	3.061 3.562	2.949 3.070	2.495 2.317	2.286 3.380	2.633 2.891	6.159 5.553	4.012 5.764	1.727 2.782
MLSTM FCN Plus	1.299 2.591	3.060 3.425	2.766 3.043	3.360 2.365	2.250 3.191	2.083 2.619	6.485 6.005	6.379 6.933	2.367 3.110
MGRU FCN Plus	1.280 2.514	3.390 3.623	2.931 3.135	2.615 2.330	2.579 3.927	2.835 2.402	5.448 6.419	5.612 5.599	3.919 4.158

Table A.1: Results of the 'MSE' metric from the experiments for all window sizes for univariate data set. Test data without/with '0' (upper/lower values in the table cells correspondingly).

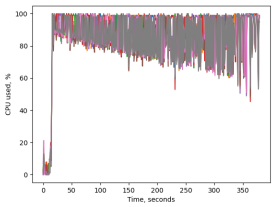
	3	5	7	14	21	28	35	42	49
Incept Time	1.721 2.378	2.865 2.203	2.465 3.168	3.794 2.421	4.083 2.875	3.292 2.332	3.762 2.853	3.329 2.683	2.269 2.214
RNN	2.134 2.569	2.468 2.552	2.860 2.764	4.305 3.226	2.816 2.870	4.227 3.736	3.228 3.740	2.559 3.201	3.487 3.963
LSTM	1.735 3.010	2.766 3.248	2.576 2.660	3.053 2.933	3.286 3.460	2.483 4.306	3.448 3.495	3.322 3.491	3.422 3.081
GRU	2.115 2.673	2.922 3.352	3.976 4.350	4.373 3.300	4.640 3.392	4.636 3.675	3.338 2.642	4.567 2.813	3.860 2.741
RNN Plus	2.047 2.541	2.292 2.441	2.769 2.654	3.046 2.105	3.147 2.637	2.973 2.395	3.115 3.744	6.255 2.829	3.118 3.999
LSTM Plus	1.668 2.519	2.795 3.572	2.789 3.839	2.849 2.706	4.010 3.400	3.974 3.378	2.191 2.072	2.203 3.071	1.942 3.939
GRU Plus	2.205 2.939	3.369 3.271	2.466 3.113	4.554 2.924	4.247 2.928	3.893 4.001	3.989 2.748	3.781 3.522	4.491 3.792
RNN FCN Plus	2.079 2.703	3.094 2.624	3.092 3.238	1.989 2.479	2.539 3.858	2.397 2.863	3.425 3.795	3.088 3.816	4.725 5.815
LSTM FCN Plus	2.577 2.525	3.091 2.311	2.767 3.088	2.041 2.129	2.581 2.856	3.946 3.338	2.738 2.621	2.852 2.853	3.291 2.880
GRU FCN Plus	2.088 2.668	3.164 2.639	3.091 2.834	2.369 2.905	2.812 3.866	2.874 2.647	3.113 2.945	3.045 2.613	3.416 3.998
MRNN FCN Plus	2.436 2.979	3.365 2.871	2.882 3.004	1.928 2.558	2.530 4.233	3.231 3.377	3.265 3.309	4.878 4.464	4.320 5.629
MLSTM FCN Plus	2.066 2.660	3.423 2.637	3.037 2.946	2.690 2.223	3.133 3.746	3.464 3.312	3.876 3.987	2.820 2.457	3.185 2.446
MGRU FCN Plus	2.459 2.790	2.909 2.376	2.984 2.940	2.833 2.547	2.418 3.650	2.914 2.682	2.710 2.859	2.737 2.492	3.123 2.653

Table A.2: Results of the 'MSE' metric from the experiments for all window sizes for multivariate data set. Test data without/with '0' (upper/lower values in the table cells correspondingly).

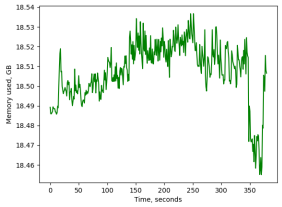
Appendix B

Figures

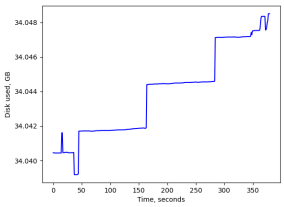
B.1 Figures from the experiments



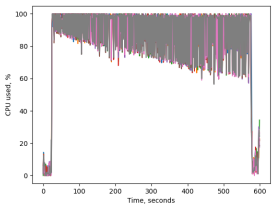
(a) MRNN-FCNPlus. CPU usage.



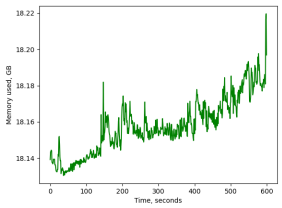
(b) MRNN-FCNPlus. Memory usage.



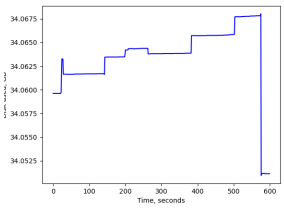
(c) MRNN-FCNPlus. Disk usage.



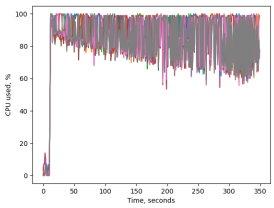
(d) InceptionTime. CPU usage.



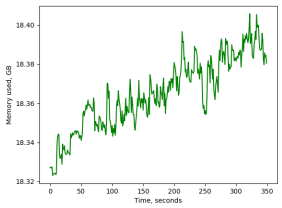
(e) InceptionTime. Memory usage.



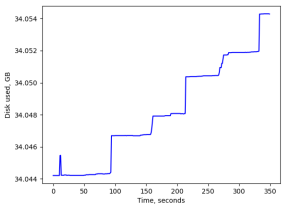
(f) InceptionTime. Disk usage.



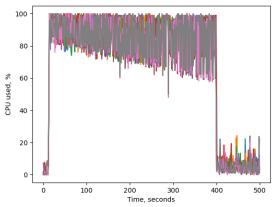
(g) LSTM-FCNPlus. CPU usage.



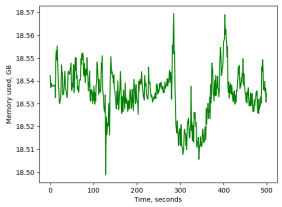
(h) LSTM-FCNPlus. Memory usage.



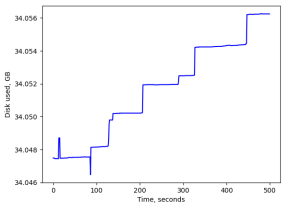
(i) LSTM-FCNPlus. Disk usage.



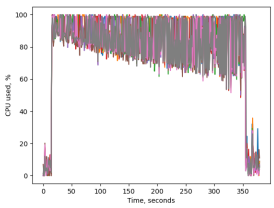
(j) MGRU-FCNPlus. CPU usage.



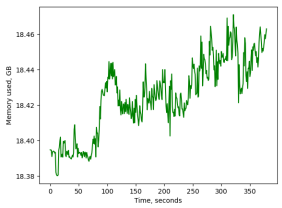
(k) MGRU-FCNPlus. Memory usage.



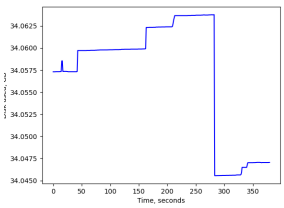
(l) MGRU-FCNPlus. Disk usage.



(m) GRU-FCNPlus. CPU usage.

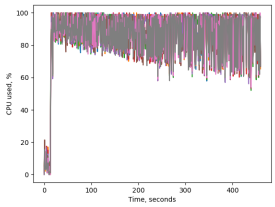


(n) GRU-FCNPlus. Memory usage.

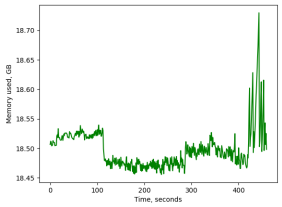


(o) GRU-FCNPlus. Disk usage.

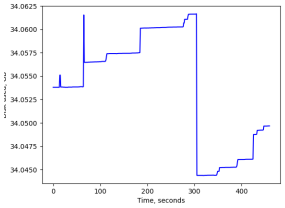
Figure B.1: CPU, memory and disk usage. Univariate time series. Second 5 best models. All but one player case. Data without gaps/'0'.



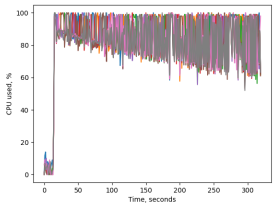
(a) MLSTM-FCNPlus. CPU usage.



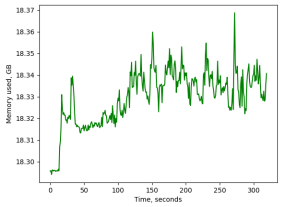
(b) MLSTM-FCNPlus. Memory usage.



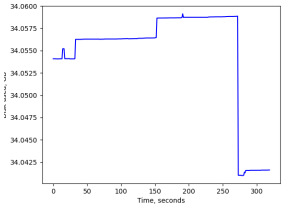
(c) MLSTM-FCNPlus. Disk usage.



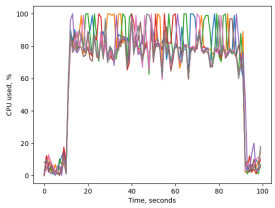
(d) RNN-FCNPlus. CPU usage.



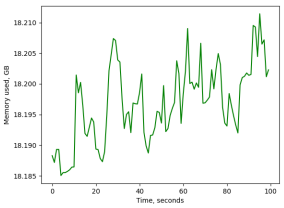
(e) RNN-FCNPlus. Memory usage.



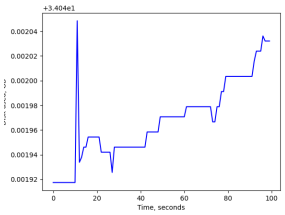
(f) RNN-FCNPlus. Disk usage.



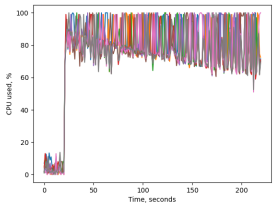
(g) GRU. CPU usage.



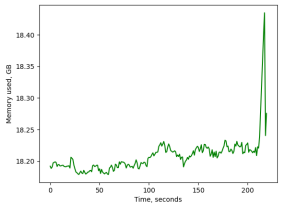
(h) GRU. Memory usage.



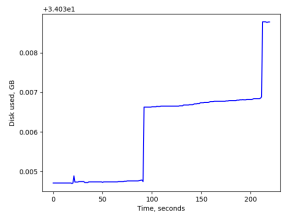
(i) GRU. Disk usage.



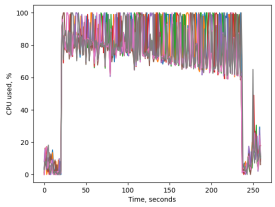
(j) LSTM. CPU usage.



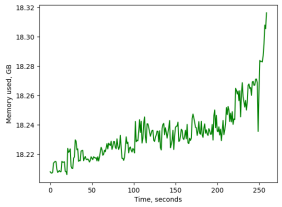
(k) LSTM. Memory usage.



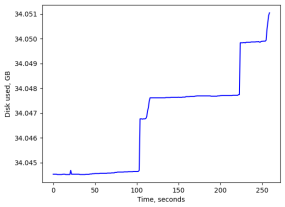
(l) LSTM. Disk usage.



(m) LSTMPlus. CPU usage.

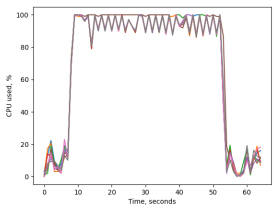


(n) LSTMPlus. Memory usage.

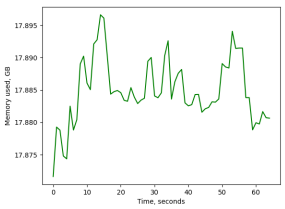


(o) LSTMPlus. Disk usage.

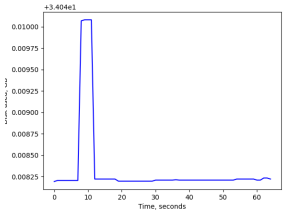
Figure B.2: CPU, memory and disk usage. Univariate time series. Second 5 best models. All but one player case. Data without gaps/'0'.



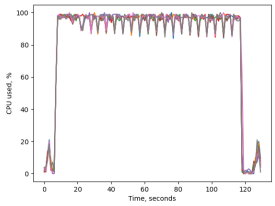
(a) InceptionTime. CPU usage.



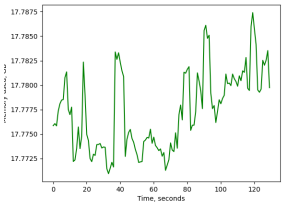
(b) InceptionTime. Memory usage.



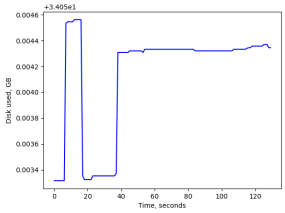
(c) InceptionTime. Disk usage.



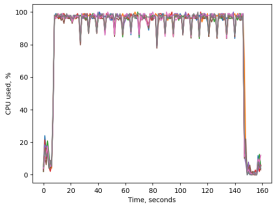
(d) GRU-FCNPlus. CPU usage.



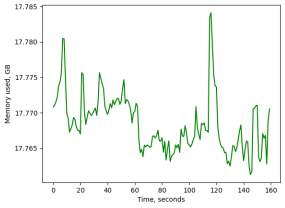
(e) GRU-FCNPlus. Memory usage.



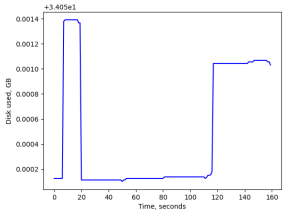
(f) GRU-FCNPlus. Disk usage.



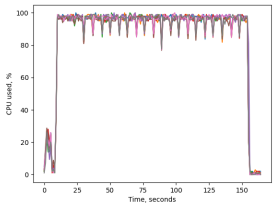
(g) LSTM-FCNPlus. CPU usage.



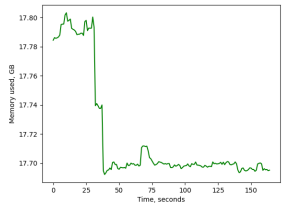
(h) LSTM-FCNPlus. Memory usage.



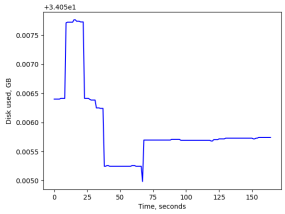
(i) LSTM-FCNPlus. Disk usage.



(j) MLSTM-FCNPlus. CPU usage.



(k) MLSTM-FCNPlus. Memory usage.



(l) MLSTM-FCNPlus. Disk usage.

Figure B.3: CPU, memory and disk usage. Best models. All but one player case. Multivariate time series. Data without gaps/'0'.