Predicting blood glucose levels using machine learning techniques with metaheuristic optimisers.

Matteo Rapa

Supervisor: Dr Michel Camilleri

L-Università ta' Malta

Department of Computer Information Systems

Faculty of ICT

University of Malta

October 2020

Submitted in partial fulfilment of the requirements for the degree of BSc. in Information Technology (Software Development)

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition	Pages
DM	Diabetes mellitus	1
T1DM	Type 1 diabetes	1
T2DM	Type 2 diabetes	1
BG	Blood glucose	-
IDF	International Diabetes Federation	2
RNN	Recurrent neural network	5, 13-19
CNN	Convolutional neural network	5-6, 16-17, 19
ANN	Artificial neural network	7
PSO	Particle swarm optimisation	9, 10
GA	Genetic algorithm	10
SGD	Stochastic gradient descent	8
AWS	Amazon web services	12
BGP	Blood glucose prediction	19
PH	Prediction horizon	13-17, 19
Seq-2-Seq	Sequence to sequence	17
GSR	Galvin skin response	18
LSTM	Long short-term memory	5, 13-19

SVR	Support vector regression	17
AR	Auto regressive	4
VAR	Vector auto regressive	5
GARCH	Generalized autoregressive conditional heteroskedasticity	5
ANN	Artificial neural network	7
ARIMA	Auto regressive Integrated Moving Average	4, 14, 20
DCNN	Dilated convolutional neural network	15-16, 19
ML	Machine learning	2-3, 6, 11-17, 20
MA	Moving Average	4
RMSE	Root mean square error	14-17, 19
PH	Prediction horizon	13-17, 19
DRNN	Dilated recurrent neural network	5, 19

Abstract

Introduction: Persons with Type-1 diabetes need to continuously monitor their blood glucose level to remain within a healthy range. Using machine learning techniques researchers can predict blood glucose values with the benefit of providing the patient with future blood glucose values with the aim of primitively taking action. The focus of this study was to investigate the use of metaheuristic optimisers to strategically tune the hyperparameter configuration of these machine learners in the context of blood glucose prediction using the OhioT1DM dataset with the aim of improving the predictive performance of the machine learners.

Research questions: *i)* What is the degree of improvement when using a metaheuristic approach over a completely random search given the same search space? *ii)* How can the computation be carried out in a shorter time, what are possible ways of distributing the workload among several machines?

Methodology: A few machine learners namely the MLP, RNN and XGBoost were implemented for the prediction of blood glucose level. Moreover, two metaheuristic optimisers, the genetic algorithm and particle swarm optimisation, and random search were used to perform hyperparameter optimisation. The experimentation was run three times to obtain an average of the performance. Due to the increased computation load in running multiple runs a Spark cluster running on EC2 instances was considered to reduce the computation time.

Results & evaluation: The results obtained from the experimentation give an indication that for the context of the ohioT1DM dataset and configurations set, the metaheuristic optimisers consistently provide a slightly better predictive performance when given enough iterations.

Conclusion: This study demonstrated that the use of metaheuristic optimisers in the context of blood glucose prediction when using the OhioT1DM dataset can provide improved results over random search. It is noted that using such techniques significantly increased computational load.

Acknowledgments

I would like to thank my supervisor Dr Michel Camilleri for his deep insights, constant motivation, and patience during the development of this study.

Chapter 1 - Introduction

The human body naturally keeps blood glucose levels within safe limits, but this is not the case for persons with diabetes. This raises the need for blood glucose level control, which involves regularly measuring blood glucose levels, and depending on the identified level, the appropriate treatment is administered. Currently, glucose measurement devices used by diabetic patients are considered invasive, hence the prediction of blood glucose levels by employing machine learning techniques is of interest to researchers. Various predictive algorithms are used to predict future blood glucose levels using current blood glucose values, by making use of physiological time series data obtained from wearable sensors. This study explores the machine learning techniques used for such an algorithm-based approaches for blood glucose level prediction, and investigates the application of hyperparameter optimisation algorithms, which may further improve the accuracy of the predictive algorithms.

1.1 Background and Focus

1.1.1 Diabetes Mellitus

Diabetes Mellitus is described as "a syndrome of impaired carbohyrdate, fat, and protein metabolism caused by either lack of insulin secretion or decreased sensitivity of insulin" [1]. Diabetes has two main types, Type 1 diabetes (T1DM) which is caused by a lack of insulin production by the Beta cells within the pancreas, and Type 2 diabetes (T2DM); caused by a buildup of resistance to the metabolic effects of insulin [1]. The blood glucose (BG) concentration is generally between 80 and 90 mg/dL for a fasting person each morning, and increases to 120 to 140 mg/dL during the first hour or so after a meal for a normal person [1]. The homeostasis system that controls the blood glucose returns the concentration back to normal ranges within 2-hours after the last absorption of carbohydrates [1]. Blood glucose levels are controlled by two primary

hormones, insulin, and glucagon, produced by the pancreas to keep blood glucose concentration within the normal limits, each having opposite effects on the BG level.

1.1.2 Blood Glucose Level Control

Insulin; triggered by high blood glucose levels, lowers blood glucose concentration back within normal ranges, by enhancing the transport of glucose from the blood to body cells, converting excess glucose into fats, and stopping the breakdown of glycogen in glucose [1]. Glucagon raises blood glucose concentration by targeting the liver; the primary store of glycogen, to release glucose in the blood when triggered by low blood glucose levels. The homeostasis system manages to keep the BG levels within the normal range by using insulin and glucagon accordingly. Unfortunately people suffering with diabetes cannot rely of this body function to keep a safe level of BG. This leads either to high blood glucose levels, referred as hyperglycaemia or low blood sugar levels referred as hypoglycaemia, both of which can have negative consequences on the health of the body. Diabetics need a reliable and non-invasive way of monitoring their blood glucose levels to avoid dangerous glycaemic events. Diabetic patients are recommended to measure their BG level four times a day [2]. However, glucose meters; an accepted method of measuring BG levels for diabetic patients, yields accurate results but is considered invasive as the patient needs to puncture their finger for a blood sample multiple times a day.

1.1.3 High Incidence

Diabetes is a global health problem, according to the International Diabetes Federation (IDF), their 2019 Atlas report [3] shows that around 463 million people are currently living with diabetes, amounting to 1 in 11 adults worldwide. The IDF projects that by 2030 more than 578 million people will be living with diabetes. Diabetic patients are at higher risk of critical glycaemic events. These events can be mitigated through timely intervention; however these preventive actions take time to act. Due to the pervasiveness of diabetes, the use of machine learning (ML) techniques for predicting blood glucose levels is an area of interest to researchers that attempt to provide the patient a non-invasive way to forecast future blood glucose levels with confidence, to

take action e.g., taking an insulin dose in time for it to act. Despite the great advancements in recent years, the typical accuracy of such predictions does not yet fall within clinically accepted parameters. Hence, further study is needed to improve the accuracy of such predictions.

1.1.4 Aim of this study

A proposed method for improving the accuracy of predictive algorithms when predicting blood glucose levels, is by using metaheuristic optimisers to find better hyperparameters. The aim of this study is to investigate the effectiveness of metaheuristic optimisers when using machine learning approaches to forecast the blood glucose levels, when using physiological data obtained from sensors. Additionally, to explore whether distributed computing techniques could help in reducing the envisaged training time required when using metaheuristic optimisers. The focus of this study includes the experimentation of different machine learning approaches with metaheuristic techniques for hyperparameter optimisation. In chapter 2 the background and concepts relating to the study of blood glucose prediction is discussed. In chapter 3 current literature pertaining to BGP using machine learning approaches is reviewed. In chapter 4 the research questions and objectives of this study are posed. In chapter 5 the implementation of the machine learners and metaheuristic optimisers is discussed. In chapter 6 a detailed outline of the experiments performed and metrics for comparison is given. In chapter 7 the results of the experimentation are observed. Finally in chapter 8 the conclusion of the study, along limitations and future works is discussed.

Chapter 2 - Background

This chapter will explore the main concepts relating to the study of data-driven blood glucose level prediction, using ML approaches to extract hidden dependencies from available data. This chapter discusses the forecasting of blood glucose levels using classical time series methods, and novel ML models, with further discussion on hyperparameter optimisers, with a focus on metaheuristics methods, and training of such models using distributed systems.

2.1 Time Series Forecasting using classical methods.

A time series data is a collection of observations measured sequentially through time. By using historical data of the past time-series, it is possible to forecast values into the future [4]. Time series forecasting has many applications such as forecasting sales, weather prediction, and load forecasting. Additional capabilities of time series analysis include anomaly detection; analysing historical data with current values, and data imputation; predicting the past to fill missing data within a time series. When evaluating a time series using a time plot, one can observe patterns such as trends, seasonality, and noise. The following sections will investigate statistical methods of forecasting for univariate and multivariate time series [5].

2.1.1 Univariate Time Series

A univariate time series represents a variable and it changes from one time step to the next. The autoregressive model (AR) forecasts future values based on the past behaviour to find the correlation between one time interval and the next (see equation 1). The moving average (MA) model does not rely on past values for the forecasting of future values but rather on past forecast errors (as shown in equation 2). A better way of modelling a univariate time series is using the ARIMA model by Box and Jenkins, where it combines the AR model and the MA model (as shown in equation 3) [6].

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t \quad (1)$$

$$y_t = c + \varepsilon t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} \quad (2)$$

where ε_t is white noise

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t, \quad (3)$$

where y'_t is the differenced series.

Equation 2.1.1 Time series model equations. Reproduced from [7].

2.1.2 Multivariate Time Series

A multivariate time series can represent multiple variables at a time point over the change in time. Multiple methods for forecasting such time series have been proposed such as the vector AR model (VAR), and the generalized autoregressive conditional heteroskedasticity (GARCH) [8].

2.2 Machine Learning approaches for Time Series Forecasting

This section introduces machine learning concepts that may be applied for the prediction of blood glucose levels.

2.2.1 Recurrent Neural Networks

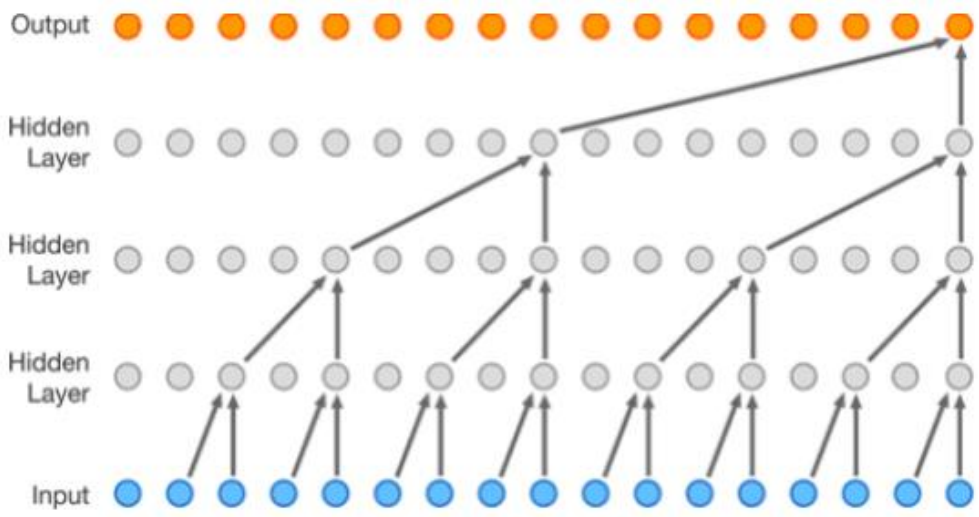
Recurrent neural networks (RNN) take a series of input, and recurrently use the output of the previous data input, to give the output of the current input. An RNN is capable of remembering what it learnt from the previous inputs by using an internal state. [9]. During training, the weights are shared and are updated for every time step in the sequence using the state. A shortcoming of RNN is that it could only remember time steps from previous iterations, as the distance grows it loses long term context. By using a long short term memory (LSTM) cell, the RNN is able to remember long term dependencies by using gates to change the internal cell state [10]. The gates help to decide which parts of the state should be remembered or forgotten.. A dilated RNN

(DRNN) structure is another method that may capture long term dependencies within time series data [11].

2.2.2 Convolutional Neural Network

A convolutional neural networks (CNN) uses convolutional layers for extracting high level features on which classification may be performed using a connected layer [9]. A CNN commonly implemented for image recognition tasks because of the ability to extract features from an image, while still preserving spatial information, and also reducing dimensionality by using pooling layers; to reduce the size without losing key information [9]. Additionally, the set of parameter weights are reusable for every convolution. A CNN can be adapted for time series data by using a 1-D layer instead, and stride the convolutions by time steps. A downside of such approach is that we do not have access to time steps well into the past. A dilated CNN proposed by DeepMind [12], (see figure 2.1) may be used to make better use of temporal dependencies found.

Figure 2.2.1 Taken from Deepmind's WaveNet paper for a dilated CNN [12].



2.2.3 XGBoost

Gradient boosting is an ensemble method technique used to predict by combining weak learners together, and iteratively improving the prediction. [9]. Extreme gradient boosting, or XGBoost is a portable library that implements a gradient boosted tree using a variety of different programming languages used for machine learning [13].

2.3 Optimisers

Optimisation has many applications in different fields such as engineering, architecture and economics. Optimisers in ML tasks can assist to tune the hyperparameters that are responsible for the configuration of the model before the training process starts. The right set of hyperparameters greatly influence the performance and accuracy of the predictive algorithm, amongst other factors. The specific configuration, i.e. set of hyperparameters chosen, e.g. number of layers, number of units within each layer, the learning rate, etc. can only be changed before every training run. Fine tuning these hyperparameters in a feasible way by making use of metaheuristics is of interest in this study.

2.3.1 Hyperparameter Optimisation

Hyperparameter optimisation is a distinct form of optimisation in which its objective is to select and tune the hyperparameters of a model. During the experimentation different combinations of hyperparameters could be used, where the combination which improves the predictive algorithm the most is selected. Due to the vast search space, and great number of combinations, it is not feasible to try out every combination to determine the optimal combination. The use of hyperparameter optimisers e.g. [14], [15] can allow for a near optimal combination to be found at a much lesser computational cost. Sampling techniques, such as the grid search algorithm are used to perform an extensive search over the search space, using fixed step sizes to reduce the computational time required, however such a sampling approach may entirely skip important regions of the search space.

2.3.2 Parameter Control

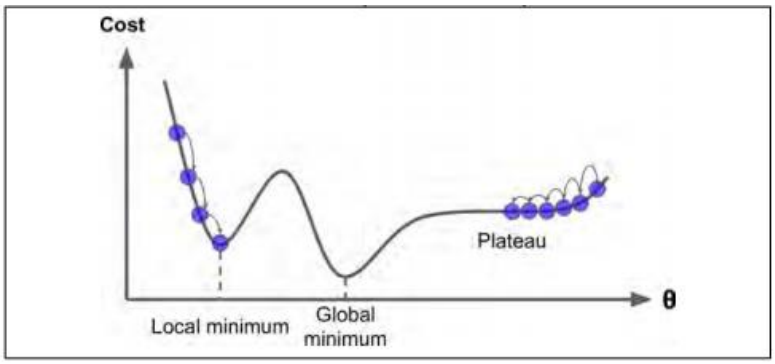
Machine learning models using artificial neural networks (ANN) use a pre-configured set of hyperparameters, e.g. dictating the number of units in each layer, which impact the performance. During the training process, the optimiser algorithm attempts to find

the weight vector that minimizes the error function for a particular model, after many iterations over a particular dataset. Certain optimisation algorithms use a number of hyperparameters that affect the manner in which each weight is altered, to potentially achieve a better result in the next iteration using the newer weights, however this is not always the case. Using an error function, it is possible to convert the optimisation problem; for the optimal combination of weights, into a minimum or maximum problem. When a change in weights provides better results, the loss in the error function is reduced and vice versa [9]. The selection of the optimiser and cost function, needs to be also carefully evaluated.

2.3.3 Gradient Descent

The selection of the optimising method may also vary results, hence finding the suitable method for the specific use case remains of interest to researchers [16]. The Gradient descent method attempts to find the better weights iteratively in order to find the minimum of the error function. The selection of the learning rate hyperparameter requires good consideration as it controls the size of each step to reach the minimum. When the learning rate is large we could skip the minimum completely. Gradient descent fails to address the problem to find the global optimum among many local minimums (refer to figure 2.1) [9]. Sebastian Ruder highlights the improvements made to gradient descent technique for tuning the parameters of a neural network [17]. In the first version of gradient descent; Batch gradient descent a change is only made every epoch, while in stochastic gradient descent (SGD) the algorithm updates the internal paramters every training example. Mini-batch gradient descent managed to combine the above approaches together to update the parameters every small batch of inputs [17].

Figure 2.3.1 Graph showing gradient descent problems. Taken from Hands-on Machine Learning [9]**.**



Momentum optimisation improves SGD by accelerating the descent in a manner to mitigate it from getting trapped within the local minimum. Momentum optimisers e.g., RMSprop, ADAM, etc. requires an additional hyperparameter, the momentum which is set from zero to one. This additional hyperparameter serves as the friction of the descend, where a value of zero has no friction and one has maximum friction. The Adam optimizer is most used as a black box optimiser, since it achieved good results in many different applications [9].

2.3.4 Metaheuristic Optimisers

A type of hyperparameter optimisation known as metaheuristic optimisers will be the focus of this study. Metaheuristic optimisation [18] uses intelligent ways of exploring the sample space to find near optimal solutions. Such optimization techniques have been of great interest to researchers [19]. Metaheuristic optimisers are governed by two fundamental principles, exploration and exploitation [20]. Exploration refers to intelligently looking into the sample space for possible candidate areas of search. Once good areas of search are found, exploitation performs a local search for the optimal solution within that space. Another characteristic of metaheuristics is history; ability to remember leading candidate solutions. The best local solutions in different areas of the search space are compared with other candidate solutions, to find the best available solution. Even though only a subset of search space is tested, metaheuristic optimisers are capable of finding near optimal solutions. Iteratively searching a large search space may be impossible due to the large computational effort. In contrast searching using a

metaheuristic optimiser can be done in feasible time. The following sections describe in detail particular metaheuristic optimisers, and their principle algorithm to conduct search.

2.3.5 Particle Swarm Optimisation

The particle swarm optimisation (PSO) algorithm, introduced by Kennedy et al. [21], was inspired by the behavior of flocking birds [22]. This metaheuristic optimiser algorithm is capable of logically finding near optimal hyperparameters by starting with a population of candidate solutions; referred to as particles. The algorithm conducts search by moving these particles around in the solution space. The movement; composed of both speed and direction (velocity), is influenced by observing the best local candidate solution, from what had already been searched. As every particle moves through the solution space, the better local solutions found alter the direction of the other particles. Using this method, the swarm, i.e the set of particles, navigate in the direction of the best available solution, possibly finding a better solution in the process. The paths taken by each candidate particle allows the PSO algorithm to search large spaces of candidate solutions with the aim of providing a near optimal solution without the need of exhaustive search [23]. Recently further improvements made to the PSO algorithm as shown in [24], that address short-comings of the base algorithm.

2.3.6 Evolutionary Algorithms

The genetic algorithm (GA); a population based search heuristic inspired by the biological evolution process. of natural evolution. The algorithm functions by selecting, based on fitness, the better solutions from a population, to be used to identify new candidates to add to the population in order to improve the next generation. This paradigm can be applied to find the near optimal hyperparameters based on previous hyperparameter values that previously gave promising results. The best candidates are chosen as the hyperparameters when no tangible improvements are made between generations [25]. When parents from the current generation have better fitness, their children will have a better chance at selection. The genetic algorithm (see figure 2.1) process, made up of key steps; the starting population of candidate solutions possible

chosen at random, the fitness function, selection, crossover and mutation. The fitness function compares a solution with other solutions in the population, determining the likelihood whether or not the solution is selected for the next generation. The concept behind the selection phase is to select the fittest solutions and let them pass their characteristics to the next generation. Using crossover, the algorithm exploits the leading characteristics, to generate better candidates. The mutation part of the algorithm introduces a random element where a subset of the new generation has parts of their characteristics altered. Mutation provides diversity to the new generation and is used to prevent early convergence. The algorithm ends when there is no improvement from one generation to the next. Differential evolution is another form of an evolutionary algorithm.

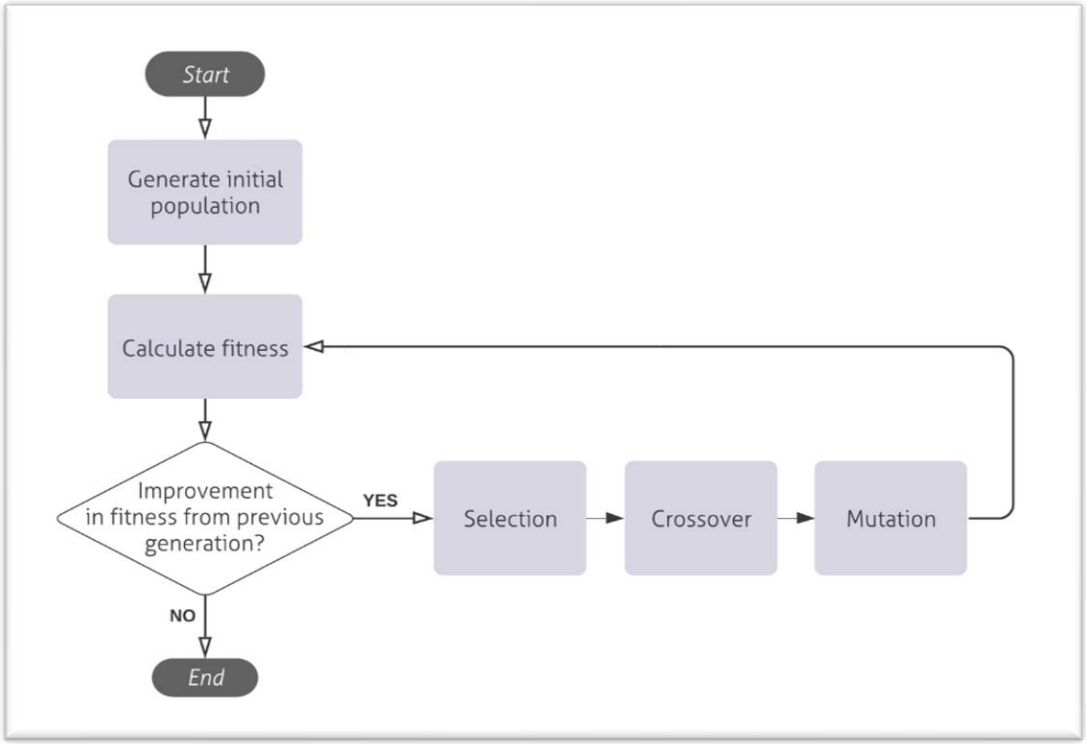


Figure 2.3 Flowchart of an evolutionary algorithm (Genetic Algorithm). Adapted from [25].

2.4 Distributed Systems

Modern ML applications require a substantial amount of computer resources to achieve their goal, e.g. computation, that challenge traditional computer systems to complete in a timely manner. When fine-tuning hyperparameters, the training computational cost needs to be repeated potentially thousands of times. Traditional systems use vertical scaling; adding more resources to a single machine to increase its capabilities, e.g. processors or storage devices. However, this strategy is unable to handle large datasets, as it is limited by a single point of failure, and I/O bandwidth limitations [26]. A distributed computer system can apportion the workload among many machines and is capable of horizontal scaling; increase the number of independent machines in the system. Rather than having a single extremely capable computer, the workload is split evenly between several high-performance machines, making scalability simple by just adding more machines. Distributed architectures are widely used by cloud computing providers e.g. Amazon Web Services (AWS). Available networks of standard PCs; commonly left unused in computer labs of organisations, research centres, and universities, could be used for intensive ML tasks, e.g., hyperparameter fine-tuning, using a distributed system. Using Beowulf network clusters [27], it is possible to compute ML tasks on standard PCs, which individually are incapable of performing. A number of distributed systems [26], [28] can be used to coordinate a large task among a large number of computers.

2.4.1 The MapReduce Model

The MapReduce model provides a way to distribute computing over several machines, using communication solely to coordinate jobs amongst the machines. The map stage splits the processes among the machines within the network, where each machine computes the allocated tasks individually. Upon completion of all the tasks, the reduce stage combines the results from all the machines into the single result. The primary advantage of the MapReduce model is that to scale the computation, more machines can be added. Apache Hadoop, an implementation of the MapReduce model, facilitates

the distribution of computation of a network. Apache Spark adds to this by improving on the MapReduce model [29].

2.4.2 Distributed Machine Learning

The training of ML models can be performed using a distributed fashion. There are two principal ways to distribute ML, either distributing the dataset, or by distributing the learning model. With data parallelism the dataset is split into even chunks that are trained simultaneously by identical model instances. Data parallelism is capable of training extremely large datasets among a sizable number of model instances. Model parallelism rather than distributing the dataset, the dataset is duplicated multiple times depending on the number of machines and parts of the model is split into separate and different instances [26].

Chapter 3 - Literature Review

This section will give discuss previous and current works about the application of machine learning models for the prediction of blood glucose levels. The prediction horizons (PH); how far into the future the prediction is made, frequently used in such studies are of 30, and 60 min. Current works propose the use of novel ML techniques and classical time series approaches. Apart from the selection of the predictive learning algorithms, an investigation will be made about the hyperparameter optimisation strategies used for fine tuning model hyperparameters, with a focus on metaheuristic optimisers. Related works make use physiological data collected from a number of sensors e.g. continuous blood glucose measurements, to train their predictive models. In this literature review the ohioT1DM dataset will be evaluated in detail further on (section 3.2), since it is made available to researchers.

3.1 Machine learning implementations of blood glucose level prediction

Using current blood glucose values, current studies show that it is possible to forecast values that help identify upcoming glyceamic events in the near futures. A number of studies experiment on the use of an RNN as the predictive model for such a prediction.

Martinsson et al. [30] propose the use of an RNN with an LSTM cell to forecast the blood glucose level but rather than using many features of the dataset their hypothesis is that the blood glucose level is solely feasible to carry out a prediction. To improve their result, it was decided to utilise a glucose-specific loss function [31] which is able to map a penalty term for predictions that can lead to clinically dangerous results. The only preprocessing technique performed was value scaling blood glucose levels by 0.01. Rather than using interpolation to fill the missing data gaps due to the possible bias, they used similar historical data pairs found within the dataset. The hyperparameter selection was carried out by training using two patients from the

dataset by the performance on the last 20% of the data. The hyperparameters relating to the RNN, such as the LSTM state size, have been chosen using trial and error in this study. The results of the experimentation are RMSE of 20.1 ±2.5 mg/dL for a PH of 30min and RMSE of 33.2±3.1 mg/dL for a PH of 60mins. In a similar study, Khadem et al. [32] use an RNN with an LSTM cell that achieves comparable results to [30].

Mohebbi et al. [33] compare a ARIMA; a classical time series forecasting technique, with an RNN for short term blood glucose prediction using continuous glucose monitoring (CGM) for a PH of up to 90 min. They note that BG levels are sufficient for such a prediction despite the accuracy of the prediction may benefit from other physiological data sources such carbohydrates and exercise. The reason for omitting such data fields is because they require manual input of data by the patient, which may result in poor quality of data. An advantage of their proposed approach, is its ability to function using limited historic blood glucose data. The approach involves using an RNN with a LSTM cell to preserve glyceamic events in memory, and also a classical approach by using an ARIMA model. The dataset used for the experimentation is 14 days of continuous BG data obtained from CGM devices, for 50 patients with diabetes, with a granularity of 5 min. Interestingly the best performing model, the LSTM used bayesian optimisation to find the better hyperparameters for the historical window size and the dropout level. The findings of this study include that even though the RNN methods give overall best results, the performance of the ARIMA model in certain situations is as good or even better than the RNN approach.

Chen et al. [34] propose a dilated recurrent neural network model to forecast BG levels for a PH of 30 minutes. They noted that due to the nature of the problem which involves time series data, a recurrent neural network is capable of providing acceptable level predictive performance. Their choice for a dilated RNN is inspired by a previous study by Chang et al [11]. Chen et al. note that use of a dilated RNN allows the neural network to better learn temporal dependencies associated with events in the ohioT1DM dataset [35]. Chen et al. decided to batch the inputs every hour (12 data points of 5 mins each). During their experimentation it was discovered that certain of the fields within the ohioT1DM dataset such as exercise, heart rate, and skin temperature fail to impact the accuracy of the model. The structure of their model consisted of 3 layered dilated RNN,

using 32 units in each layer with an exponential dilation. Vanilla RNN cells were found to have the better result. The hyperparameters of this model include the number of units at each layer, and the type of cell used within the RNN. It was found during their experimentations that results degraded when increasing the unit count, due to the size of the dataset (roughly 10,000 data points per patient). During their testing they noted that the batch size of the input can also have an effect on the performance of the model. The results achieved using the proposed DRNN model is RMSE of 19.04 mg/dL for a PH of 30min.

Midroni et al. [25] compare gradient boosted trees implemented using XGBoost with RNN models using LSTM cells. They conducted feature ablation experiments, to study whether an accurate prediction could be made without using certain data fields from invasive measurement devices e.g. BG measurements from glucometers. During their experimentation, it was discovered that the prediction is negatively affected when blood glucose is ablated. When analysing XGBoost's feature importance rank, Midroni et al. observed that the main influences on the result of the prediction is primarily based on current BG levels, secondly on BG level of one hour ago and other BG values within the past hour. Their main finding from the ablation experiments was that the XGBoost approach gives the best results when insulin or band features are ablated, giving an RMSE of 19.32 mg/dL for a PH of 30 min. A comparable study [36], uses XGBoost to predict blood glucose levels, while omitting data features from the predictor, with the aim of providing an accurate and non-invasive prediction, without using a continuous glucose monitor (CGM) device. The experimentation involved training the dataset with 1 to 3-hour time lags using XGBoost. The feature ablation results found in this study, are consistent to [25], when removing the BG feature from the prediction.

Another ML approach favoured amongst researchers for a blood glucose prediction is the CNN, due to the benefits that convolutional filtering and pooling layers give when dealing with time series data. Zhu et al. [37] argued that by converting the problem into a classification problem, a casual dilated convolutional neural network (DCNN) layers model based upon WaveNet [12] could be used to categorise the blood glucose level into 256 categories. The input for their experimentation was made up of only four fields from the ohioT1DM dataset [35], as it was discovered that the inclusion of

additional fields degraded the performance of the classification. Zhu et al. note the need for pre-processing, they firstly used first-order interpolation to fill in missing values in the dataset, then to increase the size of the usable dataset, they combined subsets of the datasets between patients for large sections of missing data in addition to filtering. The results of the model proposed in this study gives a RMSE of 21.73 ±2.52 mg/dL for a PH of 30 minutes.

Li et al. (2020) propose GluNet [38], a framework for glucose forecasting with a similar DCNN model proposed by Zhu et al. [37] The architecture of GluNet contains four components which are preprocessing, label transformations, multi-layer dilated convolutions and post-processing. This study uses multiple datasets for comparisons, but for the purpose of this study only the OhioT1DM dataset will be reviewed. The preprocessing techniques employed involve removing the outliers from CGM measurements and other user reported fields within the dataset, interpolation and extrapolation for filling missing data. The label transformation step aims to improve the predictive accuracy. The hyperparameters were tuned on the validation however no details are given on the specific technique used. The results of GluNet are slightly better than other models of this type, with a RMSE= 19.28 ±2.76 for PH of 30min and RMSE= 31.83 ±3.49 for a PH of 60mins.

Current studies also combine the use of a CNN with an RNN with the aim of improving the accuracy of the BG prediction. Li et al. [30] explore the use of a multi-layer convolutional recurrent neural network (CRNN) architecture for the prediction of blood glucose level. The CRNN architecture proposed is composed of a multi-layer convolutional neural network that is responsible for the extraction of data features using convolution and pooling techniques, followed by an RNN layer with LSTM cells and fully-connected layers. The dataset used for the training of the aforementioned consists of 6-months clinical data of 10 T1DM patients. Similar to other studies the dataset had missing data values, hence the preprocessing involved the use of interpolation and extrapolation. The results of the proposed model, RMSE of 21.07±2.35 mg/dL for a PH of 30min ,outperforms the other prediction models commonly used for BG prediction such as support vector regression (SVR). It was noted

that a limitation of the proposed CRNN model is that the performance degrades rapidly when the time horizon for the prediction is increased.

Bhimireddy et al. [39] evaluate the effectiveness of sequence-to-sequence (Seq-2-Seq) for time series modelling. The Seq-2-Seq model was experimented using LSTM, A biLSTM and a CNN-LSTM model. It was found that Seq-2-Seq BiLSTM outperforms the others. The result of the best model achieved an RMSE of 21.8±4 mg/dL for a PH of 30min and RMSE of 35 ±5.4 mg/dL for a PH of 60min.

Gu et al. [40] propose a neural physiological encoder in addition to an RNN for accurately predicting the blood glucose level. The neural physiological encoder uses decomposed convolutional filters that are capable of generating features that help when producing the prediction. Their technique produced leading results, when using the ohioT1DM dataset, with an RMSE of 17.80 mg/dL for a PH of 30 min.

Güemes et al. [41] contend that current research is focused on the prediction of BG for a PH of up to 2 hours; which is not suitable to prevent overnight hypoglycaemia. Hence, they propose the use of a binary classifier to predict BG overnight, using the OhioT1DM dataset to carry out their experimentation. The data fields used in this study are CGM blood glucose level, insulin doses, bolus, and basal, meal times, time and amount of rescue carbs, and time of self-reported hypoglycaemic events. After preprocessing of the dataset, the 8-weeks of data was partitioned by day, and further a night-time and day-time period. After extracting key features from each period, three binary classifiers are used to identify whether there will be either a hypoglycaemic night, hyperglycaemic night or on-target/off-target night. They found that using this technique it is possible to forecast overnight glycaemic events, however a larger dataset is needed to further validate their technique.

3.2 The OhioT1DM Dataset

Data-driven approaches using ML require a considerable amount of data, upon which the model may be trained. Unfortunately the appropriate clinical data relating to blood glucose on real patients is difficult to obtain and gather, and may require joint effort

with a healthcare institution. This lack of available patient data made it difficult for researchers to perform studies about blood glucose prediction by using contrived simulated data. However, recently the release of the OhioT1DM dataset [35] to interested researchers, and the Blood Glucose Level Prediction Challenge fuelled interest is shown in this area of study. The ohioT1DM dataset is made up of eight weeks of time series data for each of the 12 people with type 1 diabetes. The dataset includes data fields such as blood glucose level every 5 minutes using a Medtronic Enlite CGM sensor, galvanic skin response (GSR), step count etc. The dataset also includes self-reported data such as hypoglycaemic episodes, exercise, meals taken, and periodic blood glucose values obtained from the patient. Due to the manual input of parts of this dataset, the quality of the data is not consistent throughout, for example missing values and certain data fields are dependent on which type of sensors were used.

3.3 Metaheuristic optimisation approaches for time series forecasting

Hamdi et al. [42] propose the use of the differential evolution optimiser; a evolutionary metaheuristic optimiser, to improve the accuracy of their prediction when using support vector regression. Wang et al. [43] also particle swarm optimisation; a swarm intelligence metaheuristic, to enhance the prediction of their RNN-LSTM model. Despite there being a number of studies using hyperparameter optimisation, algorithmic hyperparameter optimisers e.g. metaheuristic optimisers, are not commonly used, where instead the majority of previous and current works use an ad hoc way to select hyperparameters for their predictive algorithms.

3.4 Comparison of related works

This section compares related works in this literature review using machine learning techniques, for blood glucose prediction using the ohioT1DM dataset. The table 1.1 shows the results of their experimentation, using the root square mean error (RMSE) standard metric, for prediction horizons of 30, and 60 minutes. In order for a fair comparison only studies using the ohioT1DM dataset are compared. The proposed

approach by Gu et al. achieves the best performance (RMSE= 17.80) for a PH of 30 minutes, and interestingly the RNN approach proposed by Khadem et al. [32] gives leading results (RMSE= 19.60±0.47) for a PH of 60 minutes. Xie et al. [44] provide a benchmark for many machine learning approaches for BGP using the ohioT1DM dataset.

Table 3.4.1 Comparison of prediction results of current literature using the ohioT1DM dataset.

Year	Paper	Model	RMSE (mg/dL)	
			PH=30min	PH=60min
2018	Zhu et al. [37]	DCNN	21.73 ±2.52	-
	Chen et al. [34]	DRNN	19.04	-
	Midroni et al. [25]	XGBoost	19.32	-
	Martinsson et al. [45]	RNN	20.1 ±2.5	33.2 ±3.1
2020	Bhimireddy et al. [39]	biLSTM	21.8 ±4	35 ±5.4
	Li et al. [38]	DCNN	19.28 ±2.76	31.83 ±3.49
	Gu et al. [40]	CNN-RNN	17.80	-
	Khadem et al. [32]	RNN-LSTM	19.4 ± 0.34	19.6 ±0.47

3.5 Summary of related works

It was commonly observed that related works using the ohioT1DM dataset, used a number of pre-processing techniques, such as interpolation, extrapolation, and using data from other patients to fill missing data series, in order to improve the quality of the available data. In [30], an RNN using only the blood glucose data field was used for training, and to perform the prediction. Mohebbi et al. [33] compare the classical forecasting technique ARIMA with a novel ML technique using an RNN. Chen et al. [34] introduce the use of a dilated RNN structure for the prediction of blood glucose. The studies [25], and [36] investigate the predictive performance of XGBoost when ablating certain data fields from the dataset with the aim of removing the need of blood glucose measuring devices. Rather than using regression to predict future BG values, [37] convert the problem into a classification problem. The RNN using a neural

physiological encoder, that uses convolutional layers, proposed in [40] gave the leading results for blood glucose level prediction for a prediction horizon of 30-min, when compared to the related works reviewed in this chapter. The use of metaheuristic optimisers have been also proposed by [40], and [41] for the domain of blood glucose prediction, making use of the differential evolution algorithm, and particle swarm optimisation to find near optimal hyperparameters. However, based on the studies reviewed, a common finding is the lack of focus for hyperparameter optimisation using algorithm based methods, such as metaheuristics. A number of hyperparameters e.g. the number of units in a layer, are chosen ad hoc or by using trial and improvement. This paper will investigate further the use of metaheuristic optimisers with predictive algorithms used in previous studies when predicting blood glucose, with the aim of increasing the accuracy of such ML approaches. In the next chapter the motivation and aims for conducting this particular study will be discussed, and defining the research questions for this study.

Chapter 4 - Motivation and Aims.

In this chapter the specific choice of focus regarding the prediction of blood glucose levels will discussed. The research questions posed in this study are defined, alongside the motivation for this study.

4.1 Research Questions

The prediction of blood glucose levels has several proposed machine learning techniques (refer to section 3.1). Each of these machine learners require a number of hyperparameter to be configured. The selection of these hyperparameters can affect the predictive accuracy of such models. It was found that several studies choose these hyperparameters in an ad hoc manner. This gave motivation to apply hyperparameter optimisation in the context of BGP on the ohioT1DM dataset, specifically by using metaheuristic optimisers. The following research questions describe what this study has addressed.

What is the degree of improvement when using a metaheuristic approach over a completely random search given the same search space?

What is the increased computational cost to achieve the improvement in predictive performance?

How can the computation be carried out in a shorter time, what are possible ways of distributing the workload among several machines?

4.2 Aims and Objectives

The aim of this study is to investigate the use of metaheuristic optimisers with machine learning model for blood glucose levels prediction when using the ohioT1DM dataset.

The primary objectives are to compare the predictive performance of different machine learners after performing hyperparameter optimisation using metaheuristic optimisers to perform model selection.

Additionally, to investigate whether such metaheuristic approaches can achieve better results when compared with random search optimisation given the same number of training runs.

The objective is to provide blood glucose using these techniques with a prediction horizon of 30 minutes and 60 minutes, as performed in literature reviewed in section 3.4.

Tuning the hyperparameters which affect the predictive performance is of interest. As the search space for hyperparameter is very large, defining the bounds of the hyperparameter search space was required for the algorithms to converge in a timely manner.

Given the significant increase in expected computation required, a secondary objective is to the investigate accelerated computing by making use of graphic processing units and using a distributed processing infrastructure. Additionally, to investigate the potential benefit of using a distributed processing architecture.

Chapter 5 - Methodology

In this chapter, the approach to the applied research is discussed. The technologies used in this paper will be addressed, alongside the particular use of the ohioT1DM dataset, the implementation of the machine learners, and metaheuristic algorithms used to perform the hyperparameter optimisation. Furthermore, an overview is given of process used for setting up the experimentation, and the comparisons made with random search optimisation to investigate the performance of the metaheuristic optimisers. To carry out the objectives three machine learners were implemented, an RNN with an LSTM cell, an MLP and a XGBoost. Hyperparameter optimisation is performed using two metaheuristic optimisers, the genetic algorithm and particle swarm optimisation, on the machine learners to tune the hyperparameters. The model selection was run on a distributed processing architecture using Apache Spark, to be capable of distributing the training among several EC2 instances on AWS.

5.1 Software and Tools

Python is a high-level general-purpose programming language. It is used to create scripts for data pre-processing, and to build the machine learners using python packages.

Pandas is a data analysis and data manipulation tool. The dataset XML files were converted into a Pandas data frames so that the data could be processed according to the specific format required by the machine learners.

Numpy is arithmetic library for multi-dimensional arrays. It is used throughout to reshape arrays.

PyTorch is an open-source machine learning library. Using this python package, the MLP and RNN models were implemented.

Apache Spark is an open-source general purpose distributed processing framework.

Virtualenv is a tool for creating python environments.

Jupyter notebook is an open-source interactive computational notebook. Using Jupyter notebook it was possible to obtain interactive results when performing the initial experimentation.

Amazon EMR is a tool for creating and configuring MapReduce clusters like Hadoop or Spark.

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud.

PySwarms is a research toolkit for particle swarm optimisation in Python [46].

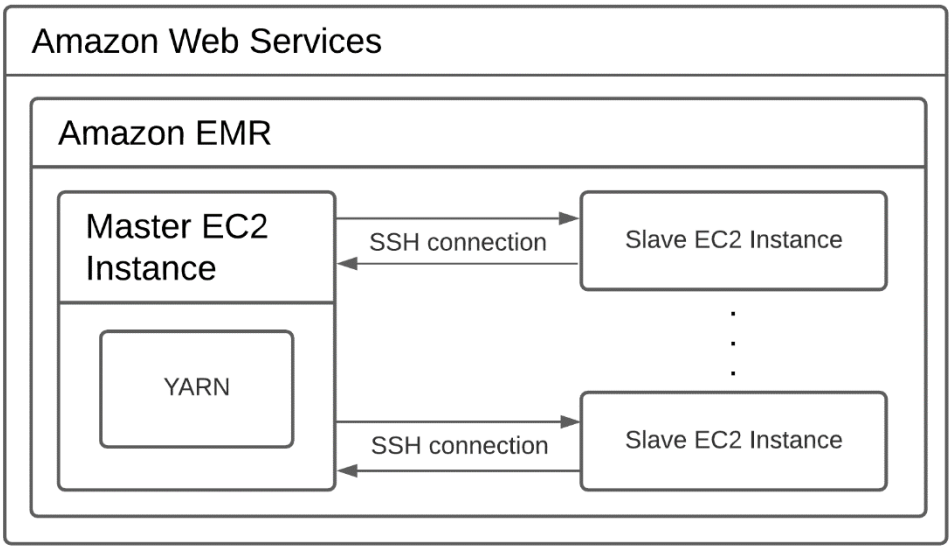
NVIDIA CUDA toolkit provides a development environment for creating high performance GPU-accelerated applications.

YARN is a resource manager for Hadoop applications.

5.2 Hardware and Distributed Infrastructure Using Apache Spark

Initial experimentation was performed on Google Colaboratory, a web IDE for python notebooks, using a single NVIDIA Tesla K80 Accelerator. However, running the experimentation multiple runs to obtain average performance significantly increases the overall training time. Hence, to further reduce the training time required to perform the experimentation, a distributed system was considered. By using Amazon EMR to manage and configure an Apache Spark cluster, composed of two AWS EC2 g4dn.xlarge instances – each equipped with an NVIDIA T4 GPU designed for accelerating machine learning workloads, and have NVIDIA CUDA installed to allow for parallel computing. The Spark cluster is made up of a master-slave architecture using the YARN resource manager (refer to figure 5.1), a single instance for the master node and a single slave/work node.

Figure 5.2.1 Architecture diagram of Apache Spark running on Amazon Web Services, where Amazon EMR is used to create, configure, and manage the cluster made up of accelerated EC2 instances.

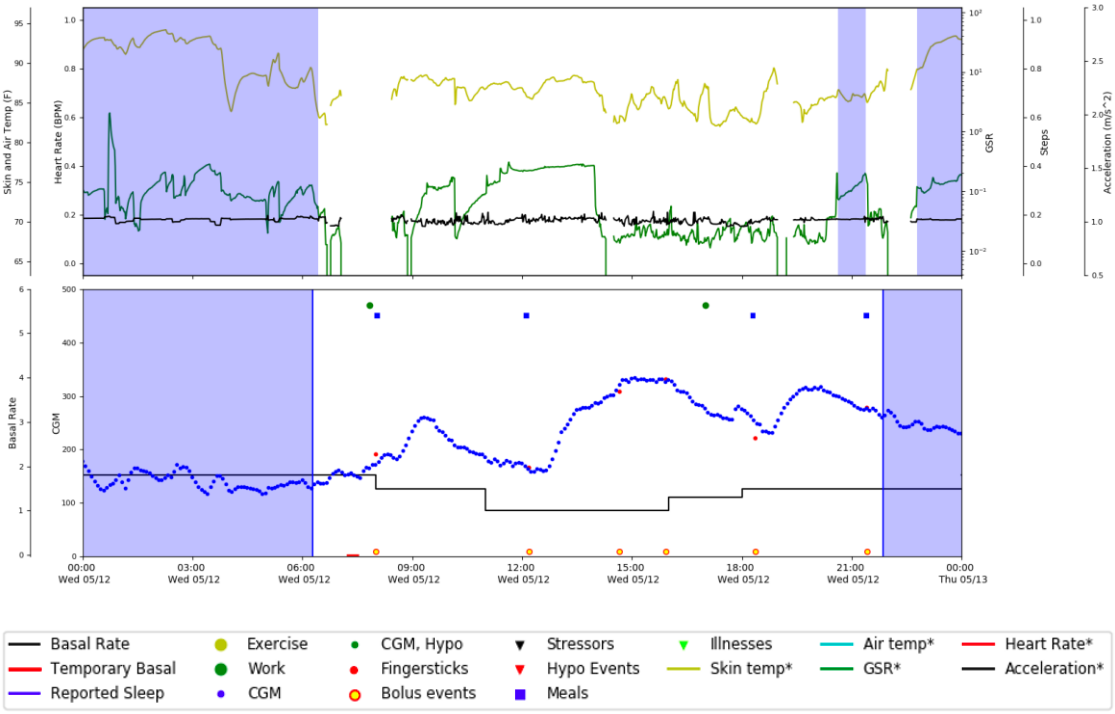


5.3 The dataset

The OhioT1DM data consists of 8 weeks of continuous glucose monitoring (CGM) data physiological data, and self-reported life-event data for each of 12 people with type 1 diabetes (refer to figure 5.3). The dataset split into the training and testing set, having 6 weeks and 2 weeks of data respectively. The testing set is further broken down into the testing and validation set, each having a week worth of data. For the use with the machine learning models, the data was converted to comma separated values from XML format using a python script, and then to tensors for faster computation on accelerated hardware.

To gain access to the ohioT1DM dataset, a Data Use Agreement (DUA) form was filled and signed by representatives from both the University of Ohio and the Department of Computer Information Systems at the University of Malta, which made this data available for research purposes. The data files included a visualisation tool called OhioT1DM Viewer that provides a daily view of the data for each of the patients.

Figure 5.3 Time series data of patient 544 using OhioT1DM Viewer. It was noted that multiple fields have missing data, including sparse parts with little data.



5.3.1 OhioT1DM Dataset Columns

For this study a few columns were considered as input to the machine learners. In section 6.1.1 a detailed explanation of the combination of these columns used in the experimentation is outlined. The Table 5.3.1 shows the considered columns for this study. The most important data column of the ohioT1DM dataset is the glucose level as it provides accurate measurements of the patients' blood glucose levels every five minutes.

Table 5.3.1 Description of data columns used from the OhioT1DM dataset. Contains both sensor time series and self-reported data. Reproduced from Marling et al. [35].

Field	Description
Patient	The patient ID number and insulin type.
Glucose Level	Continuous glucose monitoring (CGM) data, recorded every 5 minutes.

Finger Stick	Blood glucose values obtained through self- monitoring by the patient.
Temp Basal	A temporary basal insulin rate that supersedes the patient's normal basal rate. When the value is 0, this indicates that the basal insulin flow has been suspended. At the end of a temp basal, the basal rate goes back to the normal basal rate
Bolus	Insulin delivered to the patient, typically before a meal or when the patient is hyperglycaemic. The most common type of bolus, normal, delivers all insulin at once. Other bolus types can stretch out the insulin dose over the period between a begin and end timestep.
Basis GSR	Galvanic skin response, also known as skin conductance or electrodermal activity. For those who wore the Ba- sis Peak, the data was aggregated every 5 minutes, for those who wore the Empatica Embrace the data is aggregated every 1 minute.
Basis Heart Rate	Heart rate aggregated every 5 minutes. This data is only available for people who wore the Basis Peak sensor band.

5.3.2 Using CGM only

A crucial data field from the ohioT1DM dataset is the glucose level values obtained from the CGM. This data provides an accurate blood glucose level of the patient every five minutes. By using a time window of these blood glucose values with n time steps, where n >= 1, it is possible to predict a future value. The length of the time window can significantly affect the performance of the machine learner; hence this is set up as a hyperparameter for each of the ML models.

5.3.3 Interpolation for missing data in time series

The data contains missing values, including missing ranges of data. To fill the missing data, Newton's polynomial interpolation was used. After using this step the number of rows for each patient increase, refer to table 5.1 for more details. Newtons polynomial interpolation can be expressed as follows:

$$N(x) = [y_0] + [y_0, y_1](x - x_0) + \cdots + [y_0, \ldots, y_k](x - x_0)(x - x_1)\cdots(x - x_{k-1}).$$

5.3.4 Resampling the Time Series

The time index for the different fields found within the dataset had inconsistent time indexes. To better match the data, all fields where resampled every 5 minutes except for user reported data, which is sparser, e.g., the finger stick reading was resampled every hour.

5.3.5 Batching the data

Certain ML models require a rolling window of previous time indices to give a better prediction, this is shown in a number of literature, e.g. when using XGBoost Midroni et al. [36] show that the previous hour of CGM time series value has the biggest impact of the predictive performance

5.4 Hyperparameter Search Space

The primary motivation for the application of metaheuristic optimiser is to strategically traverse the hyperparameter search space with the aim of finding better hyperparameters. Different machine learners implemented in this study have varying hyperparameters which are to be tuned. As many of the hyperparameters are continuous values, a minimum and maximum bound needs to be defined in order to restrict the search space with the potential benefit that the metaheuristic optimisers use less iterations to converge. However, there are hyperparameters relating to the configuration of the dataset, which are common across all the implemented machine learners (refer to table 5.2). The time window reflects how many previous time steps of data points are used as the input to the machine learners, where the minimum is set to

12-time steps -equivalent to the previous hour when each step is equal to 5-mins, as it is shown in literature that the previous hour of data has significant effect on the predictive performance [33]. The sample timestep hyperparameter controls how much time each timestep represents, where it can be 5mins per step as the lowest, up to 30mins per timestep. The maximum for the sample timestep was set to the same magnitude as the shortest prediction horizon as it was thought that increasing the data reduction beyond that, the machine learners would not be capable of capturing the finer movements of blood glucose levels. These hyperparameter changes are made possible using the pre-processing techniques defined in section 5.3 using batching to create the time windows and the resampling technique to convert the time series. The prediction horizon hyperparameter controls how far in the future the prediction is made. This parameter is set to either PH=30mins or PH=60mins in the future as these prediction horizons are used for comparison in current literature (refer to section 3.1), is not tuneable by the optimisers. In the following sections, the focus was made on the machine learner dependent hyperparameters, along with their defined bounds and justifications for such bounds.

Table 5.4.1 Hyperparameters common across all machine learners.

Hyperparameter	Min. Bound	Max. Bound	Tuneable
Time window	12 timesteps	144 timesteps	Yes
Sample timestep	5mins	30mins	Yes
Prediction horizon	30mins	60mins	No

5.5 Machine Learning Models

Different ML models have different parameters in their configuration, each can potentially affect the performance of the model. In the section, the unique hyperparameters for each of the models will be explored.

5.5.1 Multi-Layer Perceptron

The multi-layer perceptron was implemented using the PyTorch library. It consists of an input layer, which its size changes depending on the specific time window size, and

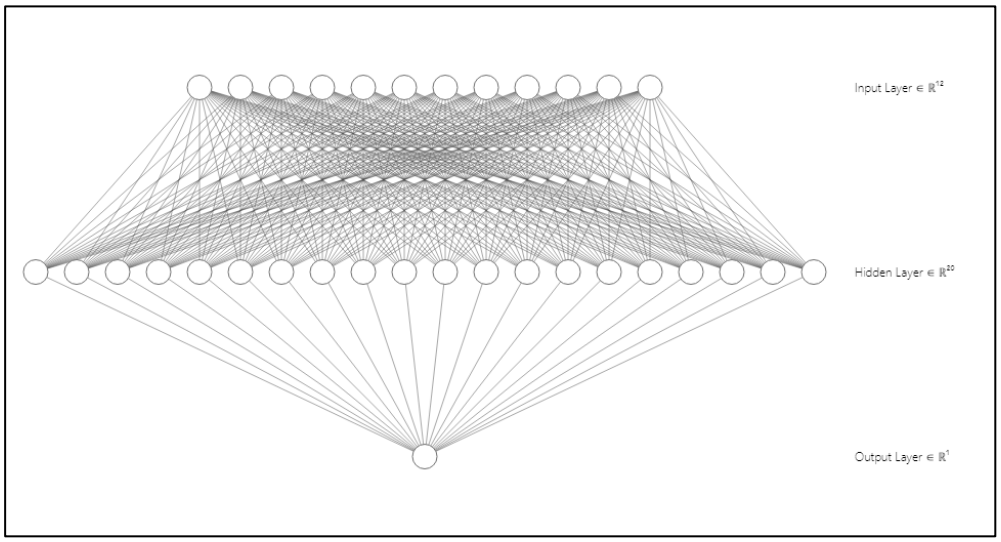
the subset of data fields from the ohioT1DM dataset (see figure 5.1). It can take n (integer) layers, where n >= 1, where each of the layers have the same number of hidden units. The activation function used is either the rectified linear activation function (ReLu) or Sigmoid, depending on the choice set by the hyperparameter optimiser. The loss function was set to the ADAM optimiser, with the learning rate as a tuneable hyperparameter. The activation functions considered can be expressed as follows:

Equation 5.5.1 Activation Functions for MLP. (1) Sigmoid function, (2) ReLu

$$\mathbf{x} = \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{-\mathbf{x}}} \quad (1)$$

$$\boldsymbol{x} = \mathbf{max}\,(\mathbf{0}, \boldsymbol{x}) \quad (2)$$

Figure 5.5.1.1 Multi-layer perceptron with 12 glucose inputs mapping into a single glucose value. The hyperparameter such as the inputs, number of hidden units (neurons), and the number of hidden layers may be tuned to provide better performance.



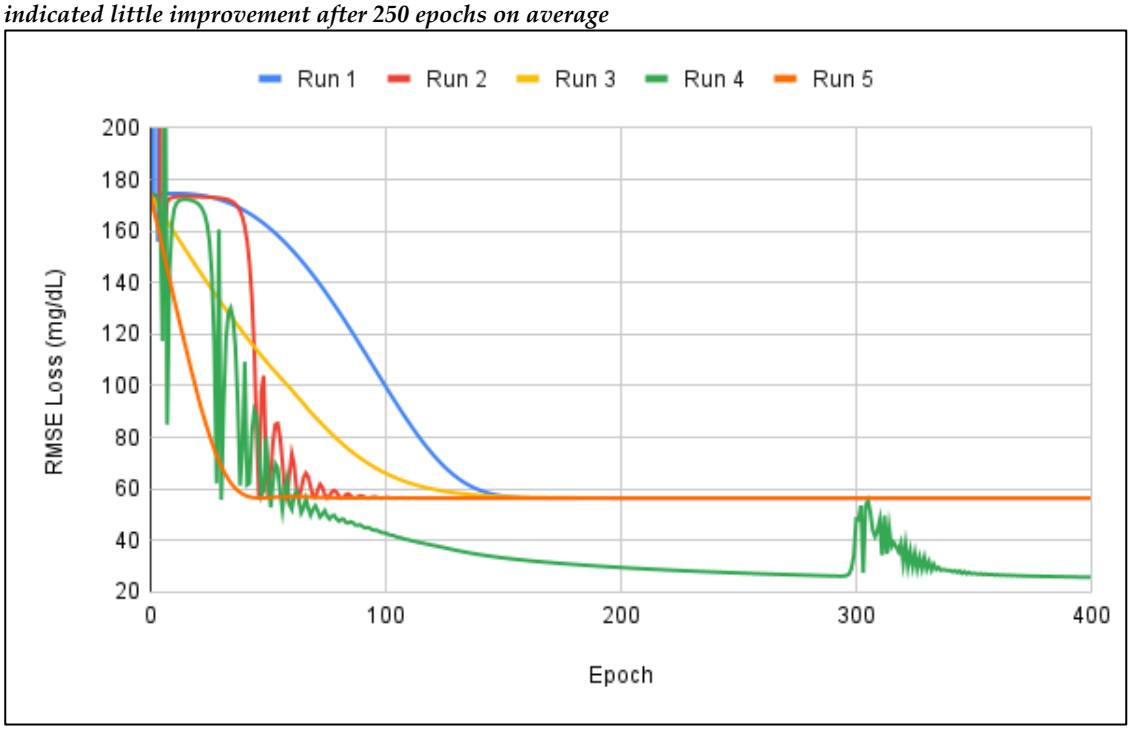
The hyperparameter search space considered in this study for the MLP neural network has the following hyperparameters:

- Number of input units which is determined by the columns used from the OhioT1DM dataset and the length of the moving window used.
- Number of hidden units found at each of the hidden layers
- Number of layers between the input and output layers.
- Number of epochs to train the MLP. The maximum number is set to 250 epochs as it was indicated in a short experiment where 5 random configurations of MLP

models were run for 1000 epochs to obtain an indication of how many epochs are required to capture most of the improvement (see figure 6.1).

- Learning rate for the Adam parameter optimiser (refer to section **Error! Reference source not found.**)

Figure 5.5.2 Line chart with five random combinations of hyperparameters for the MLP model. It indicated little improvement after 250 epochs on average



**Run 1 learning rate= 0.051, Run 2 learning rate= 0.071, Run 3 learning rate= 0.083, Run 4 learning rate= 0.069, Run 5 learning rate= 0.02*

The bounds that restrict the hyperparameter search space of the MLP were set (see table 5.6) to allow for a faster convergence by the metaheuristic optimisers.

Table 5.6 Hyperparameter search space bounds for MLP.

Hyperparameter	Min. Bound	Max. Bound
Input Units	12	144 * (number of data fields)
Hidden Units	12	64
Learning Rate	0.001	0.1
Number of Layers	1	24
Window Size	12 time steps	144 time steps
Sample Step	5min	30min
Activation Function	ReLu or Sigmoid	

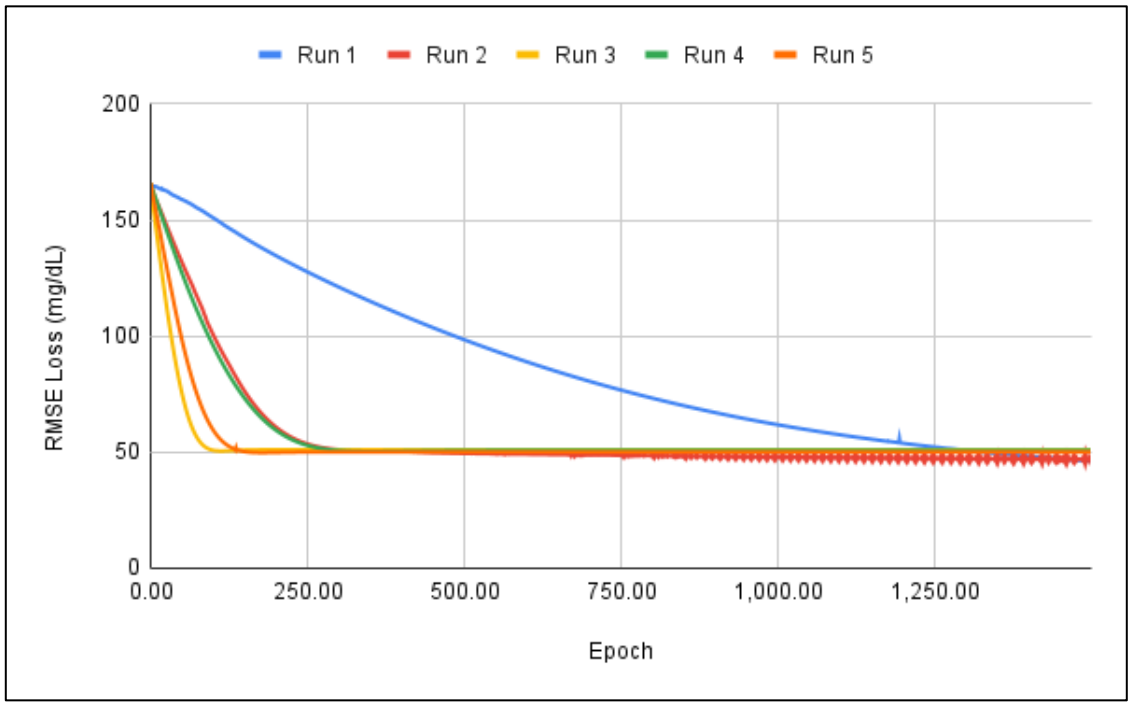
Max Epochs	250

5.5.2 Recurrent Neural Network with LSTM

The recurrent neural is implemented using the PyTorch library using a single LSTM layer. The parameter optimiser used is the Adam Optimiser, with the learning rate as a tuneable hyperparameter. This neural network has the following hyperparameters:

- Hidden state size. It is noted that in initial experimentation there was an indication that a larger hidden state size provides a better configuration for the RNN, however at an increased computational time. Due to this, it was decided to restrict the size to 24.
- Number of input units (sequence length)
- Learning rate
- The number of epochs trained for this RNN model is 400 epochs, refer to figure 4.3. Due to the nature of the machine learner, more epochs are needed than the MLP model.

Figure 5.5.3 Loss vs. Epoch chart for 5 RNN runs. Each of the runs had a random hyperparameter combination using the bounds defined. Run 1 required more epochs to achieve the same loss as the other four runs, this may be due to it having the smallest learning rate hyperparameter. This chart gives an indication that 400 epochs is sufficient to provide lesser loss on the RNN model.



**Run 1 learning rate = 0.005, Run 2 learning rate = 0.089, Run 3 learning rate = 0.093, Run 4 learning rate = 0.047, Run 5 learning rate = 0.036*

Table 5.6 Hyperparameter search space bounds for MLP.

Hyperparameter	Min. Bound	Max. Bound
Input units	12	144 * (number of data fields)
Hidden Size	1	24
Learning Rate	0.001	0.1

5.5.3 Extreme Gradient Boosting Trees

The extreme gradient boosting trees (XGBoost) was implemented using the python package xgboost by using the XGBRegression module to adapt the learner for a regression task. The XGBoost algorithm has multiple tuneable hyperparameters which effect different parts of the training process. The following hyperparameters were considered for tuning in this study:

- Step size (eta)
- Minimum cost reduction (gamma)
- Number of estimators
- Subsample size
- Maximum depth
- Minimum child weight

Table 5.6.3 Hyperparameter search space bounds for XGBoost learner.

Hyperparameter	Min. Bound	Max. Bound
Input Size	12	144
Eta	0.1	5
Number of Estimators	1	200
Gamma	0	50
Min. Child Weight	0.01	10
Maximum Depth	1	24
Subsample size	0.1	1

5.6 The Genetic Algorithm Configuration

5.6.1 Initial Population Generation

The first generation of the population is made up of randomly selected hyperparameters, similar to the random search approach. Beyond this step the optimisation loop starts until there is convergence, or the algorithm reaches the maximum allowed generation. In this study the maximum allowed generations was set to 8, as increases the number of generations increases the number of models that are required to be trained.

5.6.2 Crossover Method

The genetic algorithm was implemented using single crossover where an index in randomly selected and at the crossover point the genes are set to the other parents.

5.6.3 Mutation

Random mutation on the children is performed using a mutation rate variable to increase/decrease the number of changes to the genes for the children in the generation.

5.7 Particle Swarm Optimiser Algorithm and Configuration

The particle swarm optimiser defines randomly the position, i.e., the hyperparameter combination, for each of the particles, and computes the cost for each of the particles. Depending on the cost of the surrounding particle, each of the particles change velocity, and position to that of the local best (see figure 5.7.1). After several iterations, the particles converge to the position with the lesser cost.

Using the PySwarms python package, such a particle swarm optimiser was implemented for each of the machine learners. The configuration used for the PSO was a cognitive parameter c1 set to 0.5, a social parameter set to 0.8 and the inertia weight parameter set to 0.8.

Figure 5.7.1 Pseudo code of the PSO algorithm. Reproduced from [21].

```
for each particle i = 1, ..., S do
   Initialize the particle's position with a uniformly distributed random vector: x_i ~ U(b_lo, b_up)
   Initialize the particle's best known position to its initial position: p_i ← x_i
   if f(p_i) < f(g) then
       update the swarm's best known  position: g ← p_i
   Initialize the particle's velocity: v_i ~ U(-|b_up-b_lo|, |b_up-b_lo|)
while a termination criterion is not met do:
   for each particle i = 1, ..., S do
      for each dimension d = 1, ..., n do
         Pick random numbers: r_p, r_g ~ U(0,1)
         Update the particle's velocity: v_i,d ← ω v_i,d + φ_p r_p (p_i,d-x_i,d) + φ_g r_g (g_d-x_i,d)
         Update the particle's position: x_i ← x_i + v_i
         if f(x_i) < f(p_i) then
            Update the particle's best known position: p_i ← x_i
            if f(p_i) < f(g) then
               Update the swarm's best known position: g ← p_i
```

Chapter 6 - Experimentation

In this chapter the experimentation carried out in this paper is outlined. The RMSE metric for regressions error is presented, along with Parkes's error grid analysis to check for clinical usability of the results. The planned experiments and setups/variations of experiments are detailed to fulfil the objectives of this paper.

6.1 Planned Experiments

In this section the planned experiments to obtain the required results for the evaluation are discussed. Each of the experiment sets are explained. The experiments can vary by one or multiple of the following reasons:

- The subset of the ohioT1DM dataset
- The patient
- The machine learner
- The metaheuristic optimiser
- The prediction horizon

6.1.1 Subsets of the OhioT1DM dataset

The dataset contains many columns of data as expanded in section 5.3.1. The experiments were setup with two variations. The first variant using simply the continuous glucose machine time series as a time window. The second variation included the finger stick data column, temporary basal, bolus and the basis GSR.

6.1.2 Variations of the Patient

The ohioT1DM dataset contains two cohorts of patients (refer to table 6.1.1), a cohort of patients made available in the first edition of the BGLP Challenge in 2018 which make use of the Basis sensor band and 530G CGM. Additionally, in the second edition of the BGLP Challenge in 2020, another cohort of patient data was released that made use of the Empatica sensor band and 630G CGM. The 2020 cohort contain slightly different

physiological data fields than that of the 2018 cohort, hence different patients from both groups are selected for experimentation.

Table 6.1.1 OhioT1DM dataset patient differences. Reproduced from [35]

ID	Pump Model	Sensor Band	Cohort
540	630G	Empatica	2020
544	630G	Empatica	2020
552	630G	Empatica	2020
567	630G	Empatica	2020
584	630G	Empatica	2020
596	630G	Empatica	2020
559	530G	Basis	2018
563	530G	Basis	2018
570	530G	Basis	2018
575	530G	Basis	2018
588	530G	Basis	2018
591	530G	Basis	2018

From the 2018 cohort the following patients were used 559, 563, 570, and for the 2020 cohort the following patients were used 540, 544, 552. The experiments conducted were configured as follows:

Table 6.1.2 List of experiments that were conducted in this study.

Experiment	Patient	Machine Learner	Hyperparameter Optimiser	Dataset Features	Prediction Horizon
1	2018 cohort	RNN	Random Search, GA,	CGM Only	30
2	2020 cohort	RNN	Random Search, GA,	CGM Only	30
3	2018 cohort	XGBoost	Random Search, GA	CGM Only	30
4	2020 cohort	XGBoost	Random Search, PSO	CGM Only	30
6	2020 cohort	XGBoost	Random Search, PSO	Other Features	60

6.2 Root Square Mean Error

The root square mean error (RMSE) is a standard metric used in data science for understanding the regressive performance of machine learners. Such measure is based on the square root of the sum of squared differences between a given test set of actual and the predicted values from the machine learner. In this case, measured in mg/dL for blood glucose levels. RMSE can be formally expresses as:

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}} \tag{6.2}$$

where T is the number of data points observed, t is the current element, $\hat{y}_t$ is the predicted value and y is the actual value.

6.3 Parkes Error Grid Analysis

Using the Parkes error grid (EG) one can determine the clinal usability of predictions given by the ML models by making a comparison to the measured glucose values. The EG is made up of five risk categories as shown in table, where depending on the magnitude of the predictive error, a risk category can be assigned. After performing model selection, the best model with the found hyperparameters, the resulting error is plotted using Parkes EG. The ISO15197:2013 guideline specifies use of the Parkes error grid for assessing outlier data points that do not meet the analytical accuracy requirements. In the published version of this new guideline, 95% of the data points must fall within risk category A [47].

Table 6.3.1 Risk categories identified for the Parkes Error Grid. Adapted from [48].

Risk Category	Description
A	No effect on clinical action
B	Altered clinical action or little or no effect on clinical outcome
C	Altered clinical action—likely to effect clinical outcome
D	Altered clinical action—could have significant medical risk
E	Altered clinical action—could have dangerous consequences.

Evaluation strategy

As defined in section 4, the objective is to compare the performance of metaheuristic approaches with that of random search, to find out whether using a strategic selection of combinations, by using previously found candidate can provide better combinations than that of random search. The measure for the performance on regression, is the RMSE loss. Due to the random element in the optimisation processes, the optimisation runs were performed three times each to obtain the average improvement and thus a better estimate of the true performance. The comparisons were performed between different optimisation approaches, i.e., GA and PSO, but also the machine learner. A closer look was taken at the rate of improvement by looking at the average improvement and standard deviation from one generation to the next. Another measure used to compare the experiments was the best performance so far per run, and the average best so far per generation.

Chapter 7 - Results and Observations

In this section the results and predictive performance of the various ML models will be discussed, after model selection by optimising the hyperparameters using the Genetic Algorithm and particle swarm optimisation.

7.1 The Results

In the first experiment using the RNN model and performing hyperparameter optimisation using the genetic algorithm and random search optimiser using both 2018 cohort patients. The experiment included the use of patient 540, CGM only dataset, and consisted of a population size of 12 for the genetic algorithm. a trend is observed for the genetic algorithm were from the first generation as show in table 7.1.3, the best fitness is RMSE (mg/dL) =70.45 and the last generation the best fitness was found to be RMSE (mg/dL) =70.45. The average best over the generations for the GA shows that given the generations it could improve by using previously known hyperparameter combinations. The average best for the random search does not show any correlation with the number of generations. It is noted that for the random search the average best can vary in both directions as shown in figure 7.1.1 where in the first generation the average best is the lowest, whilst the highest in generation 4. The best fitness value achieved from this experiment is RMSE=66.58, from the GA Optimiser. This experiment was repeated for patients 544, 552.

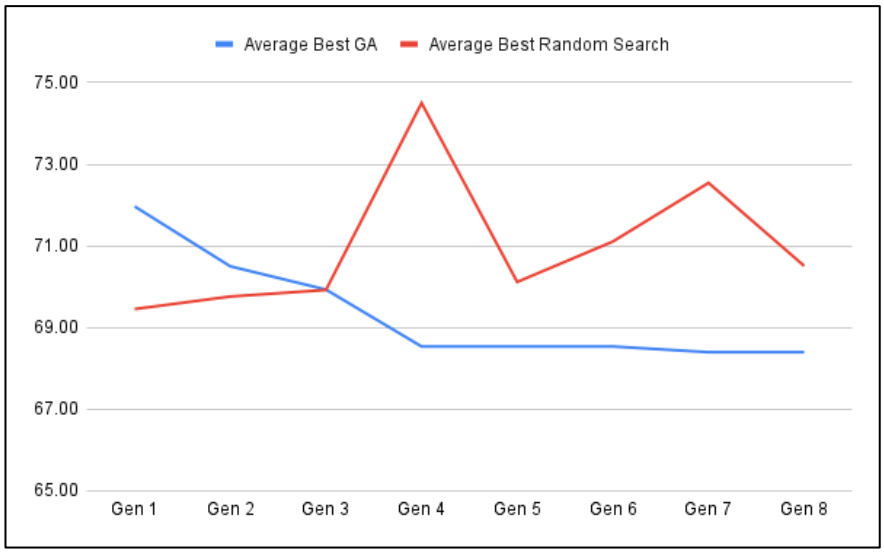


Figure 7.1.1 Line chart showing the average best fitness for both the genetic algorithm (blue), and the random search optimiser (red) for patient 540 using the RNN with PH=30min.

On average the Genetic algorithm takes longer to converge then the random search, as shown in figure 7.1.2, The genetic algorithm delivered the best result in the experimentation, however the random search gave satisfactory results given that it does not have a strategy for tuning the hyperparameters. The runs show that given enough generations the GA can provide better hypermeter, i.e., improve the predictive performance as shown in figure 7.1.3 where the average best so far is better for the GA when compared with the random search.

Figure 7.1.2 Chart depicting Best So Far Per Run over the generations for patient 540 using the RNN with PH=30min.

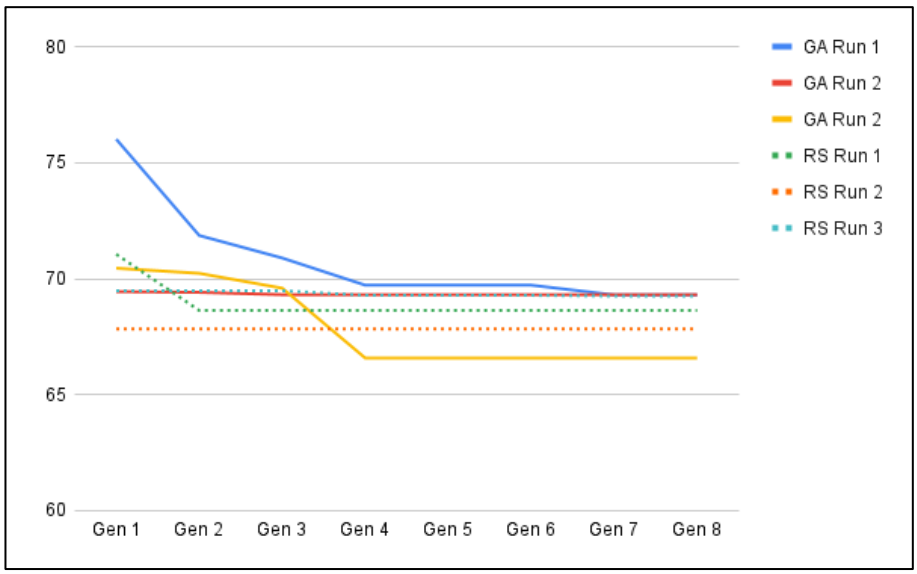
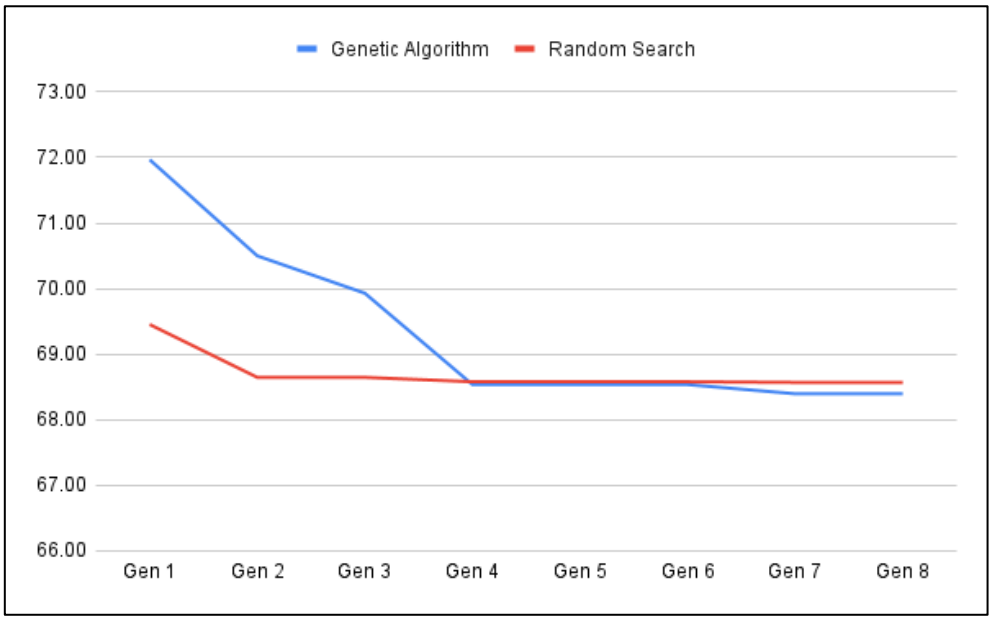


Figure 7.1.3 Line chart showing the average best so far for each of the hyperparameter optimisers over the generations.



	RNN using Genetic Algorithm												
Patient			Run 1			Run 2			Run 3		Overall Runs		
		BSF	BF	Average	BSF	BF	Average	BSF	BF	Average	Average best	SD of Average BF	Global BF	Average BSF
540	**Gen 1**	76.01	76.01	126.72	69.44	69.44	82.16	70.45	70.45	95.16	71.97	3.54	69.44	71.97
	Gen 2	71.86	71.86	104.2158333	69.41	69.41	77.23	70.23	70.23	82.58	70.50	1.25	69.41	70.50
	Gen 3	70.89	70.89	80.03	69.31	69.31	70.63	69.59	69.59	71.50	69.93	0.84	69.31	69.93
	Gen 4	69.72	69.72	84.31	69.31	69.31	73.79	66.58	66.58	70.19	68.54	1.71	66.58	68.54
	Gen 5	69.72	69.72	73.67	69.31	69.31	73.79	66.58	66.58	74.72	68.54	1.71	66.58	68.54
	Gen 6	69.72	69.72	76.29	69.31	69.31	69.99	66.58	66.58	73.05	68.54	1.71	66.58	68.54
	Gen 7	69.30	69.30	69.92	69.31	69.31	74.15	66.58	66.58	75.47	68.40	1.57	66.58	68.40
	Gen 8	69.30	69.30	78.03	69.31	69.31	76.28	66.58	66.58	70.70	68.40	1.57	66.58	68.40
	RNN using Random Search Optimiser													
	Gen 1	71.05	71.05	92.90	67.83	67.83	90.8025	69.48	69.48	111.51	69.45	1.61	67.83	69.45
	Gen 2	68.63	68.63	92.41	67.83	70.04	112.47	69.48	70.61	97.04	69.76	1.02	68.63	68.65
	Gen 3	68.63	69.39	100.04	67.83	70.83	101.80	69.48	69.54	116.90	69.92	0.79	69.39	68.65
	Gen 4	68.63	72.29	94.69	67.83	81.93	113.28	69.28	69.28	105.76	74.50	6.61	69.28	68.58
	Gen 5	68.63	70.39	109.43	67.83	70.03	106.99	69.28	69.93	110.39	70.12	0.24	69.93	68.58
	Gen 6	68.63	69.92	76.31	67.83	69.75	107.63	69.28	73.65	94.60	71.11	2.20	69.75	68.58
	Gen 7	68.63	78.20	112.01	67.83	70.18	79.55	69.25	69.25	102.94	72.54	4.92	69.25	68.57
	Gen 8	68.63	71.42	91.49	67.83	70.32	76.63	69.25	69.79	94.83	70.51	0.83	69.79	68.57

Table 7.1.1 Results of the first experiment, where the RNN was tuned using the genetic algorithm and random search optimiser for 3 runs on patient 540 for PH=30.

BSF=Best So Far, BF=Best Fitness, SD=Standard Deviation

The second experiment was run again on the patient 563, the results (refer to table 7.1.2) confirm those of experiment one where the average best is better for the genetic algorithm as shown in figure 7.1.4. In comparison to the previous experiment, the best so far per run of the GA is comparable with that of the random search (refer to figure 7.1.5). On the other hand, the average best so far over the generations is better for the GA after three generations.

Figure 7.1.4 Line chart showing the average best for each of the hyperparameter optimisers per generation.

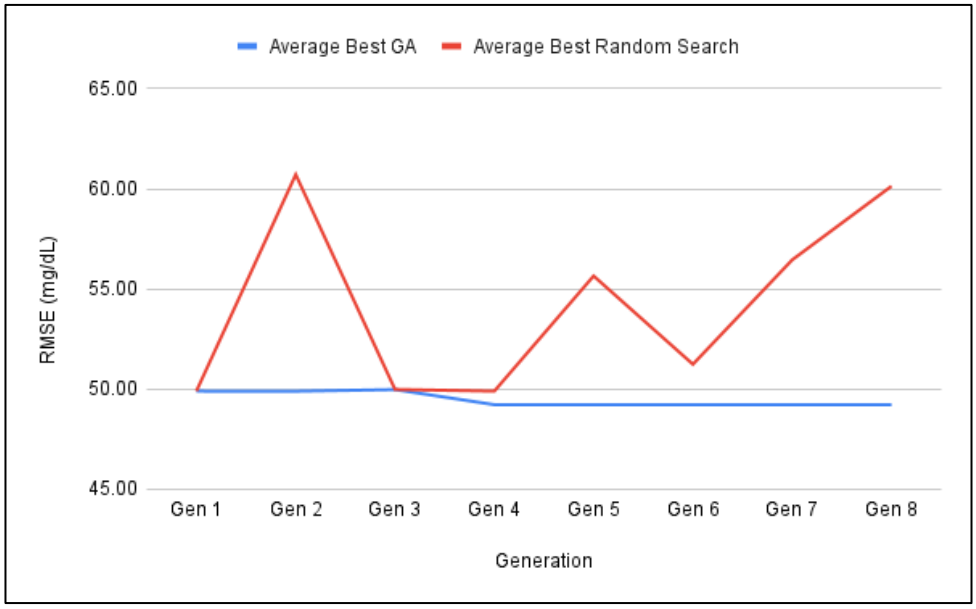


Figure 7.1.5 Line chart showing Best So Far per run for each of the hyperparameter optimisers. In this chart the random search performs similarly to the genetic algorithm, with the third run of the GA showing significant improvement from the rest of the rest.

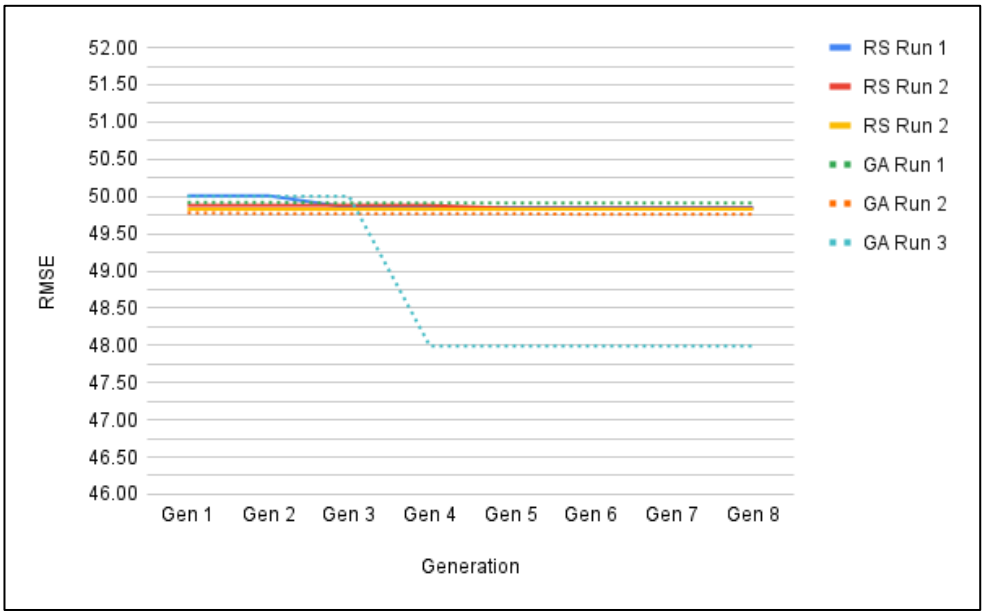


Figure 7.1.6 Line chart of average Best So Far over the generations, where the genetic algorithm shows improvement after several generations.

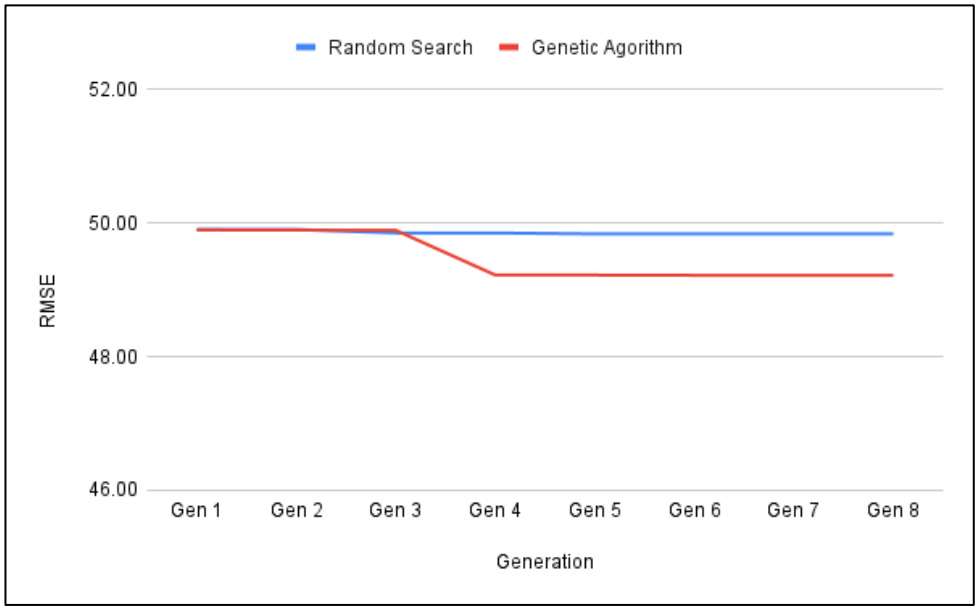


Table 7.1.2 Results of the second experiment, where the RNN was tuned using the genetic algorithm and random search optimiser for 3 runs on patient 563 for PH=30.

BSF=Best So Far, BF=Best Fitness, SD=Standard Deviation

	RNN using Genetic Algorithm												
Patient			**Run 1**			**Run 2**			**Run 3**		**Overall Runs**		
		BSF	**BF**	**Average**	**BSF**	**BF**	**Average**	**BSF**	**BF**	**Average**	**Average BF**	**SD of Average BF**	**Global best**	**Average BSF**
563	**Gen 1**	49.92	49.92	87.44	49.78	49.78	72.24	50.00	50.00	100.59	49.90	0.07	49.88	49.90
	Gen 2	49.92	49.92	59.99	49.77	49.77	61.86	50.00	50.00	65.50	49.90	0.27	50.00	49.90
	Gen 3	49.91	49.91	59.97	49.77	50.02	90.00	50.00	50.00	52.56	49.98	0.09	49.85	49.89
	Gen 4	49.91	**49.91**	**49.92**	49.77	49.77	55.35	47.99	47.99	52.55	49.22	1.10	47.99	49.22
	Gen 5	49.91	**49.91**	**53.53**	49.77	49.77	50.13	47.99	47.99	50.18	49.22	1.07	47.99	49.22
	Gen 6	49.91	**49.91**	**51.51**	49.76	49.76	52.71	47.99	47.99	53.97	49.22	1.73	47.99	49.22
	Gen 7	49.91	49.91	50.27	49.76	49.76	52.20	47.99	47.99	50.25	49.22	1.19	47.99	49.22
	Gen 8	49.91	49.91	55.37	49.76	49.76	59.37	47.99	47.99	54.45	49.22	14.93	47.99	49.22
	RNN using Random Search Optimiser													
	Gen 1	50.01	50.01	82.18	49.88	49.88	87.86	49.83	49.83	91.40	49.91	0.09	49.83	49.91
	Gen 2	50.01	50.54	82.49	49.88	50.27	65.72	49.83	81.32	118.37	60.71	17.85	50.27	49.91
	Gen 3	49.85	49.85	73.95	49.88	50.02	90.00	49.83	50.08	76.92	49.98	0.12	49.85	49.85
	Gen 4	49.85	**49.85**	**49.97**	49.88	49.93	77.94	49.83	49.92	84.74	49.90	0.04	49.85	49.85
	Gen 5	49.85	**49.85**	**49.96**	49.84	49.84	95.54	49.83	67.26	104.90	55.65	10.05	49.84	49.84
	Gen 6	49.85	**49.85**	**49.95**	49.84	51.45	70.21	49.83	52.42	78.62	51.24	1.30	49.85	49.84
	Gen 7	49.85	50.11	79.43	49.84	50.00	73.16	49.83	69.25	102.94	56.45	11.08	50.00	49.84
	Gen 8	49.85	53.13	85.79	49.84	76.03	111.01	49.83	51.24	92.90	60.13	13.80	51.24	49.84

In the third experiment using the XGBoost machine learner, a comparison is made between the random search optimiser and the genetic algorithm, on thepatient 559 from the 2018 cohort with PH=30. Using the genetic algorithm for hyperparameter optimisation, the experiment was run for three runs, each with an initial population of 12 for eight generations. The results show (refer to table 7.1.3) that even though random search finds hyperparameters that give satisfactory results, there is no noticeable improvement/trend as shown in figure 7.1.7 when using the average best RMSE (mg/dL). For the best so far RMSE metric for the final generation was achieved by the third run of the GA with the second best being the first run of the ransom search. The average best so far (Average BSF) for the random search was comparable with that of the GA (as shown in figure 7.1.9), however the GA provided slightly better results, similar to the previous experiments.

*Table 7.1.3 Results of the third experiment showing a comparison of predictive performance of the XGBoost after hyperparameter optimisation using the Genetic algorithm and Random search on the ohioT1DM dataset (CGM only) using patient 559 for PH=30min. *BSF=Best So Far, BF=Best Fitness, SD=Standard Deviation*

		XGBoost using Genetic Algorithm												
Patient			**Run 1**			**Run 2**			**Run 3**		**Overall Runs**			
		BSF	**BF**	**Average**	**BSF**	**BF**	**Average**	**BSF**	**BF**	**Average**	**Average best**	**SD of Average BF**	**Global best**	**Average BSF**
559	Gen 0	20.96	20.96	22.59	21.75	21.75	22.95	20.79	20.79	22.88	21.17	0.51	20.79	21.17
	Gen 1	20.90	20.90	21.86	21.39	21.39	22.32	20.79	20.79	22.02	21.03	0.32	20.79	21.03
	Gen 2	20.90	20.90	21.54	21.39	21.39	22.05	20.79	20.56	21.22	20.95	0.42	20.56	21.03
	Gen 3	20.90	20.88	21.27	21.39	21.39	21.85	20.79	20.56	21.28	20.94	0.42	20.56	21.03
	Gen 4	20.90	20.88	21.16	21.39	21.39	21.61	20.79	20.46	23.14	20.91	0.47	20.46	21.03
	Gen 5	20.90	20.78	20.96	21.39	21.22	21.56	20.79	20.46	21.23	20.82	0.38	20.46	21.03
	Gen 6	20.73	20.73	20.94	21.22	21.22	21.60	20.38	20.38	20.71	20.78	0.42	20.38	20.78
	Gen 7	20.73	20.73	20.97	21.14	21.14	21.34	20.38	20.38	20.52	20.75	0.38	20.38	20.75
		XGBoost using Random Search												
	Gen 0	21.07	21.07	22.55	21.59	21.59	28.75	21.17	21.17	21.87	21.28	0.28	21.07	21.28
	Gen 1	21.07	21.26	22.86	21.59	22.35	23.46	21.17	22.49	22.63	22.03	0.67	21.26	21.28
	Gen 2	21.07	22.50	23.04	21.59	21.92	22.55	21.17	22.81	22.91	22.41	0.45	21.92	21.28
	Gen 3	21.07	21.24	22.19	21.38	21.38	27.99	21.17	23.01	23.09	21.88	0.98	21.24	21.21
	Gen 4	21.07	21.49	25.25	21.38	23.07	25.82	21.17	23.18	23.25	22.58	0.95	21.49	21.21
	Gen 5	20.54	20.54	22.44	21.38	21.58	22.50	21.17	23.38	23.46	21.83	1.44	20.54	21.03
	Gen 6	20.54	22.16	23.26	20.85	20.85	30.97	21.17	23.65	23.94	22.22	1.40	20.85	20.85
	Gen 7	20.54	21.38	22.69	20.85	22.03	23.69	21.17	24.45	42.81	22.62	1.62	21.38	20.85

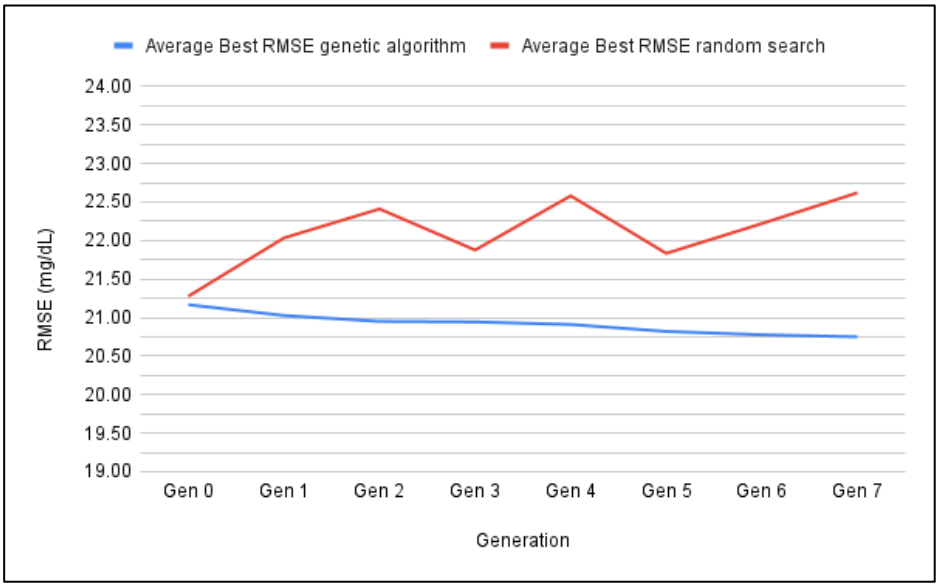


Figure 7.1.7 Line chart showing average best RMSE (mg/dL) for the genetic algorithm and random search for every generation. (Population size = 12)

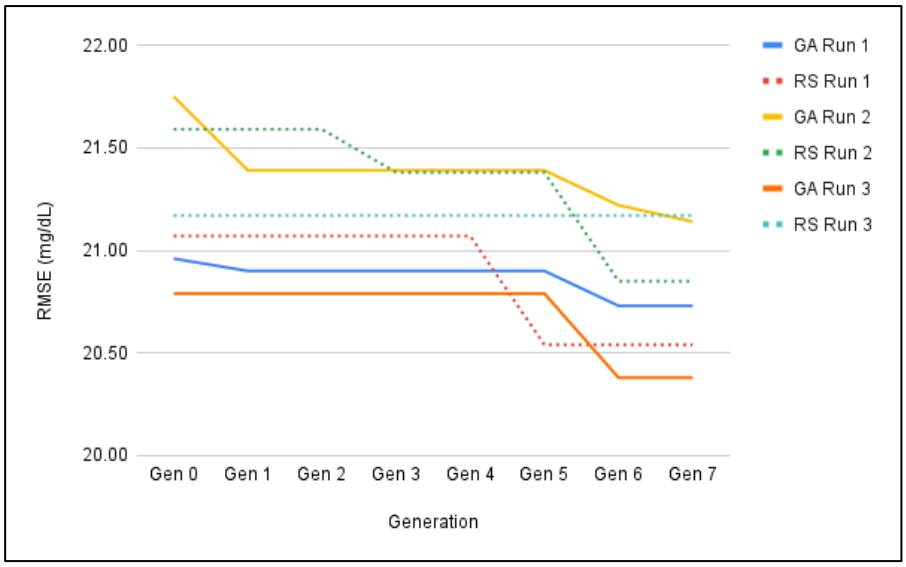


Figure 7.1.8 Line chart showing the best so far RMSE (mg/dL) for the genetic algorithm and random search for every generation. (Population size = 12)

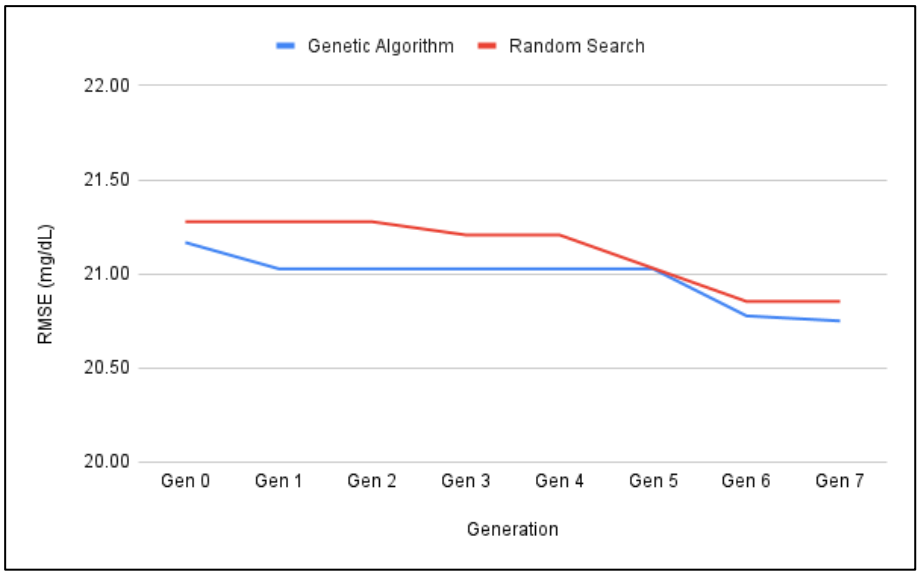


Figure 7.1.9 Line chart showing the average best so far RMSE (mg/dL) for the genetic algorithm and random search for every generation. (Population size = 12)

Further analysis could be carried out with additional variations of such experiments, e.g., using a different prediction horizon.

7.2 Evaluation of Results

The results give an indication that in the context of blood glucose prediction using the OhioT1DM dataset, the genetic algorithm gives slightly better results than the random search optimiser. It is possible that the degree of improvement achieved by using the genetic algorithm is affected by the specific configuration of the GA, i.e., the cross-over method, and the mutation process and amount. Furthermore, it is important to note that in order to perform the experimentation within a reasonable time, the search space for each of the machine learners was restricted, e.g. in the context of the RNN model, in the initial experimentation it was observed that a small hidden layer size can negatively affect the predictive performance, however with the benefit of reducing the training time. The initial population size set may also limit the predictive performance improvement when using the metaheuristic optimisers as the algorithm make use of previously found configurations with the objective being to exploit characteristic which can provide a better result. Additionally, restricting the number of epochs may also have a negative affect on the predictive performance for a particular hyperparameter configuration. The lack of significant improvement by the hyperparameter

optimisation process may be attributed to the limits in the relationship and/or quality of the data that the machine learner is evaluating. In section 5.3.3, a pre-processing step was made with the aim of improving the quality of the data by removing and/or filling the missing values to mitigate the aforementioned issue.

Chapter 8 - Limitations, Future Work and Conclusions

8.1 Limitations of the Study

In this paper two metaheuristic algorithms were studied, both of which are evolutionary methods. Due to the vast number of combinations of tests, given the available resources it was not feasible to use other type of metaheuristic approaches for the optimisation of hyperparameters.

8.2 Future Work

For the implementation of the metaheuristic optimiser in this paper, the parameters that govern the way the algorithm worked were set during the initial testing and were used throughout the experimentation. Hence, possible future work is exploring the use of adaptive metaheuristic optimisers and the tuning of parameters that impact the exploration and exploitation aspect of the metaheuristic algorithms.

Due to the prevalence of Type-2 diabetes in Malta, techniques implemented in this paper may be adapted for blood glucose levels prediction using physiological time series data obtained from Maltese Type-2 diabetes patients, especially for senior patients [49].

8.3 Conclusion

In this study the use of metaheuristic approaches to optimise machine learner performance on blood glucose levels prediction together with the use of other physiological data and alternative means to increase computational power were explored. The experiments indicate that a metaheuristic approach to hyperparameter optimisation in this context may produce better results than random search given

sufficient generations. It is noted that using such techniques significantly increases the computational cost, as the models must be retrained multiple times with different configurations.

Bibliography

[1] N. Stevanovic, *Guyton and Hall Textbook of Medical Physiology - 12th-Ed*. 2019.

[2] "IDF Type 1 Diabetes," [Online]. Available: https://www.idf.org/aboutdiabetes/type-1-diabetes.html.

[3] "IDF DIABETES ATLAS Ninth edition 2019." https://diabetesatlas.org/en/.

[4] O. D. Anderson and C. Chatfield, *The Analysis of Time Series: Theory and Practice.*, vol. 25, no. 4. 1976.

[5] I. A. Iwok and A. S. Okpe, "A Comparative Study between Univariate and Multivariate Linear Stationary Time Series Models," *Am. J. Math. Stat.*, vol. 6, no. 5, pp. 203–212, 2016, doi: 10.5923/j.ajms.20160605.02.

[6] M. A. Din, "ARIMA by Box Jenkins Methodology for Estimation and Forecasting Models in Higher Education," *ATINER's Conf. Pap. Ser.*, no. March, pp. 3–14, 2015.

[7] Robin John Hyndman and George Athanasopoulos, "ARIMA models," in *Forecasting: Principles and Practice*, 2018, pp. 223–243.

[8] Y. Xia and M. S. Kamel, "A generalized least absolute deviation method for parameter estimation of autoregressive signals," *IEEE Trans. Neural Networks*, vol. 19, no. 1, pp. 107–118, 2008, doi: 10.1109/TNN.2007.902962.

[9] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. 2017.

[10] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[11] S. Chang *et al.*, "Dilated recurrent neural networks," *arXiv*, no. Nips, pp. 1–11, 2017.

[12] A. van den Oord *et al.*, "WaveNet: A Generative Model for Raw Audio," pp. 1–15, 2016, [Online]. Available: http://arxiv.org/abs/1609.03499.

[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," *Proc. ACM*

SIGKDD Int. Conf. Knowl. Discov. Data Min., vol. 13-17-Augu, pp. 785–794, 2016, doi: 10.1145/2939672.2939785.

[14] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, "On Empirical Comparisons of Optimizers for Deep Learning." [Online]. Available: https://www.tensorflow.org/.

[15] C. Blum and A. Roli, "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, Sep. 2003, doi: 10.1145/937503.937505.

[16] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A Survey of Optimization Methods from a Machine Learning Perspective," *IEEE Trans. Cybern.*, vol. 50, no. 8, pp. 3668–3681, 2020, doi: 10.1109/TCYB.2019.2950779.

[17] S. Ruder, "An overview of gradient descent optimization algorithms," pp. 1–14, 2016, [Online]. Available: http://arxiv.org/abs/1609.04747.

[18] H. R. Maier, S. Razavi, Z. Kapelan, L. S. Matott, J. Kasprzyk, and B. A. Tolson, "Introductory overview: Optimization using evolutionary algorithms and other metaheuristics," *Environ. Model. Softw.*, vol. 114, no. November 2018, pp. 195–213, 2019, doi: 10.1016/j.envsoft.2018.11.018.

[19] K. Hussain, M. Najib, M. Salleh, · Shi Cheng, and · Yuhui Shi, "Metaheuristic research: a comprehensive survey," *Artif. Intell. Rev.*, vol. 52, 2018, doi: 10.1007/s10462-017-9605-z.

[20] D. Devikanniga, K. Vetrivel, and N. Badrinath, "Review of meta-heuristic optimization based artificial neural networks and its applications," *J. Phys. Conf. Ser.*, vol. 1362, no. 1, 2019, doi: 10.1088/1742-6596/1362/1/012074.

[21] R. E. James Kennedy, "Particle Swarm Optimisation," *Stud. Comput. Intell.*, vol. 927, pp. 5–13, 1995, doi: 10.1007/978-3-030-61111-8_2.

[22] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," *Proc. 1999 Congr. Evol. Comput. CEC 1999*, vol. 3, pp. 1945–1950, 1999, doi: 10.1109/CEC.1999.785511.

[23] S. Kiranyaz, "Particle swarm optimization," *Adapt. Learn. Optim.*, vol. 15, no. May

2011, pp. 45–82, 2014, doi: 10.1007/978-3-642-37846-1_3.

[24] I. Sousa-Ferreira and D. Sousa, "A review of velocity-type PSO variants," *J. Algorithm. Comput. Technol.*, vol. 11, no. 1, pp. 23–30, 2017, doi: 10.1177/1748301816665021.

[25] C. Midroni *et al.*, "Genetic algorithms: a survey," *CEUR Workshop Proc.*, vol. 2148, no. 6, pp. 17–26, Jun. 2018, doi: 10.1109/2.294849.

[26] S. Landset *et al.*, "A survey on distributed machine learning," *arXiv*, vol. 2, no. 1, pp. 1–36, 2019, doi: 10.1186/s40537-015-0032-1.

[27] T. Sterling, D. Savarese, D. J. Becker, J. Dorband, U. Ranawake, and C. V Packer, "BEOWULF: A Parallel Workstation for Scientific Computation," 1995.

[28] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *arXiv*, 2019.

[29] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop ecosystem," *J. Big Data*, vol. 2, no. 1, pp. 1–36, 2015, doi: 10.1186/s40537-015-0032-1.

[30] K. Li, J. Daniels, C. Liu, P. Herrero, and P. Georgiou, "Convolutional recurrent neural networks for glucose prediction," *arXiv*, vol. 24, no. 2, pp. 603–613, 2018.

[31] S. Mirshekarian, R. Bunescu, C. Marling, and F. Schwartz, "Using LSTMs to learn physiological models of blood glucose behavior," *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, pp. 2887–2891, 2017, doi: 10.1109/EMBC.2017.8037460.

[32] H. Khadem, H. Nemat, J. Elliott, and M. Benaissa, "Multi-Lag Stacking for Blood Glucose Level Prediction," *CEUR Workshop Proc.*, vol. 2675, pp. 146–150, 2020.

[33] A. Mohebbi *et al.*, "Short term blood glucose prediction based on continuous glucose monitoring data," *arXiv*, pp. 5140–5145, 2020.

[34] J. Chen, K. Li, P. Herrero, T. Zhu, and P. Georgiou, "Dilated recurrent neural network for short-time prediction of glucose concentration," *CEUR Workshop Proc.*, vol. 2148, pp. 69–73, 2018.

[35] C. Marling and R. Bunescu, "The ohioT1DM dataset for blood glucose level

prediction: Update 2020," *CEUR Workshop Proc.*, vol. 2675, pp. 71–74, 2020.

[36] D. A. Cilia, "A Data Analytic and Machine Learning Approach to Diabetes Monitoring," no. September, 2020.

[37] T. Zhu, K. Li, P. Herrero, J. Chen, and P. Georgiou, "A deep learning algorithm for personalized blood glucose prediction," *CEUR Workshop Proc.*, vol. 2148, pp. 64–78, 2018.

[38] P. B. Computer, I. Speller, S. Kundu, and S. Ari, "Glunet : A Deep Learning Framework for," vol. 2, no. 1, pp. 86–93, 2020.

[39] A. Bhimireddy, P. Sinha, B. Oluwalade, J. W. Gichoya, and S. Purkayastha, "Blood glucose level prediction as time-series modeling using sequence-to-sequence neural networks," *CEUR Workshop Proc.*, vol. 2675, pp. 125–130, 2020.

[40] K. Gu, R. Dang, and T. Prioleau, "Neural Physiological Model: A Simple Module for Blood Glucose Prediction," *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, vol. 2020-July, pp. 5476–5481, 2020, doi: 10.1109/EMBC44109.2020.9176004.

[41] A. Guemes *et al.*, "Predicting Quality of Overnight Glycaemic Control in Type 1 Diabetes Using Binary Classifiers," *IEEE J. Biomed. Heal. Informatics*, vol. 24, no. 5, pp. 1439–1446, 2020, doi: 10.1109/JBHI.2019.2938305.

[42] X. Li *et al.*, "Accurate prediction of continuous blood glucose based on support vector regression and differential evolution algorithm," *Swarm Evol. Comput.*, vol. 38, no. 2, pp. 135–140, 2018, doi: 10.1016/j.bbe.2018.02.005.

[43] W. Wang, M. Tong, and M. Yu, "Blood Glucose Prediction with VMD and LSTM Optimized by Improved Particle Swarm Optimization," *IEEE Access*, vol. 8, pp. 217908–217916, 2020, doi: 10.1109/ACCESS.2020.3041355.

[44] J. Xie and Q. Wang, "Benchmarking Machine Learning Algorithms on Blood Glucose Prediction for Type i Diabetes in Comparison with Classical Time-Series Models," *IEEE Trans. Biomed. Eng.*, vol. 67, no. 11, pp. 3101–3124, 2020, doi: 10.1109/TBME.2020.2975959.

[45] J. Martinsson, A. Schliep, B. Eliasson, C. Meijner, S. Persson, and O. Mogren,

"Automatic blood glucose prediction with confidence using recurrent neural networks," *CEUR Workshop Proc.*, vol. 2148, pp. 64–68, 2018.

[46] L. James V. Miranda, "PySwarms: a research toolkit for Particle Swarm Optimization in Python," *J. Open Source Softw.*, vol. 3, no. 21, p. 433, 2018, doi: 10.21105/joss.00433.

[47] J. L. Parkes and D. Ph, "Technical aspects of the Parkes error grid," vol. 7, no. 5, pp. 1275–1281, 2013.

[48] J. L. Parkes, S. L. Slatin, S. Pardo, and B. H. Ginsberg, "A new consensus error grid to evaluate the clinical significance of inaccuracies in the measurement of blood glucose," *Diabetes Care*, vol. 23, no. 8, pp. 1143–1148, 2000, doi: 10.2337/diacare.23.8.1143.

[49] S. Cuschieri, "The diabetes epidemic in Malta," *South East. Eur. J. Public Heal.*, vol. 13, no. February, pp. 1–10, 2020, doi: 10.4119/seejph-3322.