Learning models for glucose prediction in diabetes therapy

Matthew Liston

Submitted for the Degree of Master of Science in

Machine Learning

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

August 29, 2020

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 9156

Student Name: Matthew Liston

Date of Submission: 29/8/2020

Signature: Matthew R Liston

I owe many thanks to Dr. Nicola Paoletti for his valuable comments and suggestions, and for providing the datasets necessary to this project.

Abstract

Model predictive control (MPC) algorithms have demonstrated the ability to achieve near optimal insulin control policies during in-silico artificial pancreas (AP) trials. However, almost all of these algorithms require an internal state estimation step in order to predict future blood glucose levels. This complicates their use in clinical trials, as state of the art continuous glucose monitor (CGM) sensors are imperfect, and faulty readings may leave MPC algorithms in an error prone state. Building on recent work to develop stateless representations of optimal MPC derived insulin control policies via deep neural networks (DNN), we perform a set of computational experiments to study their viability for stateless blood glucose prediction. Rather than seeking to replace MPC with a DNN approximation of its insulin policy, we instead hope to integrate DNNs within the MPC framework to specifically overcome its state estimation problem. In order to do this, we employ a form of supervised learning, such that DNNs are trained to predict future blood glucose levels from synthetically generated examples. We interpret our results by analysing the accuracy, computational efficiency, and uncertainty of the trained DNN models. We find that our models are highly accurate, can be significantly compressed, and are confident for short-term predictions.

Contents

1 Introduction

Type 1 diabetes (T1D) is a chronic disease which either severely limits or completely eliminates the production of insulin by pancreatic beta cells. This can be life threatening, as insulin is a critical hormone released by the pancreas to regulate blood glucose (BG) levels. Without it, the body fails to maintain a normal BG level, and may be subject to the known complications caused by hyperglycaemia or hypoglycaemia. The artificial pancreas (AP) is a proposed solution which would continuously inject T1D patients with insulin via an automated pump, based on continuous glucose monitor (CGM) sensor readings and other disturbances. The AP is comprised of three components; these are a CGM sensor, an insulin pump, and a control algorithm device (CAD). At a high level, the AP system functions as follows. First, the CGM sensor is placed in the patient subcutaneously and enabled to transmit its current BG reading to the CAD. The CAD then receives the BG reading and infers an appropriate insulin response using a selected algorithm. Finally, the CAD instructs the insulin pump to execute its desired response [4]. By continuously evaluating this algorithm, the AP hopes to functionally mimic the insulin response of the pancreas to changes in blood glucose levels.

Currently, the primary algorithm used by the AP for inference is model predictive control (MPC). MPC strategies will be extensively discussed in section 2, so here we will just focus on a brief overview of the approach. The objective of MPC is to find the best-fitting input to a given plant process in order to maintain some desired setpoint value. It aims to achieve this by way of a feedback loop, where projections of the plant output and an internal model of the plant are used to solve an optimization problem. The solution to this problem yields the input to the plant process for the next iteration of the loop [16]. It is straightforward to see how this framework can be adapted for the purpose of maintaining a safe BG level. The input process would be the insulin response of the AP, the setpoint would be a safe BG level, and the plant would be an approximation of patient BG levels using a CGM sensor.

However, there are two main problems with using MPC as the inference model for the AP. The first problem is that many MPC strategies formulate a non-convex optimization problem, which cannot be efficiently solved in real time within the constraints of modern medical hardware. As a result, most state-of-the-art AP systems which use MPC rely on linear plant models to simplify computational complexity. The second problem with MPC is that it generally requires state estimation for its plant model based on only CGM sensor readings, which are noisy approximations of BG levels [5]. It is easy to see why this is problematic in the context of the MPC feedback loop, as an errant CGM measurement will persistently affect future outputs of the plant model.

In this paper, we hope to provide insight which could help overcome both of these problems using variations of the deep neural network (DNN) architecture. DNNs are interesting because they are capable of estimating arbitrary composite functions, and are widely used across many fields such as computer vision and natural language processing [18]. Specific implementations necessary to this paper will be carefully explained in subsection 2.3, so here we will discuss at a high level how DNNs could potentially mitigate the state estimation problem of MPC.

The primary advantage of DNNs, in the context of the AP, is that they learn a direct mapping between the input variables and target value(s). Therefore, they do not require state estimation in order to make predictions. Furthermore, since they are capable of estimating arbitrary composite functions, it is intuitive that they may be able to estimate BG measurements produced by non-linear, non-convex MPC strategies. Combining these two points, it is plausible that a DNN could learn to imitate the plant of a complex MPC strategy given enough training examples [5]. This is important as the DNN learns a functional approximation of the plant which can predict future BG measurements without requiring state estimation. Thus, using the DNN model as a replacement for the plant model within MPC could potentially remedy its state estimation problem. One objective of this paper is to perform a set of computational experiments in order to evaluate the accuracy of different DNN approximations.

An additional objective is to study the extent to which trained DNN models can be miniaturized for this problem without sacrificing accuracy. Intuitively, it makes sense that the smallest accurate network is the most preferable from an efficiency perspective, since reducing the size of the network directly reduces its hardware cost. The primary method we will use to study this is iterative magnitude-based weight pruning, which will be detailed in subsection 2.5.

A final objective of this paper is to quantify model confidence in its predictions in order to better understand difficult examples. We construct our models such that they can be analysed using the tools of approximate Bayesian inference, which allows us to construct credible intervals around each model prediction. Furthermore, we exploit uncertainty in model parameters to create ensembles, which outperform their respective individual models in terms of accuracy.

The remainder of this paper will be structured as follows. In section 2, we detail all background material necessary to understand the computational experiments. In section 3, we specify the datasets used in our computational experiments. In section 4, we overview the implementation details of our computational experiments. In section 5, we present the results of our computational experiments in accordance with each objective and dataset. Finally, in section 6, we conclude by summarizing our findings, as well as speculating about future extensions of our research.

2 Background Research

2.1 Glucose-Insulin System

2.1.1 Overview

Modelling the complex relationship between BG levels and insulin secretion by pancreatic beta cells has long been a primary objective of AP researchers. Even prior to the development of Bergman's model in 1981, research efforts were already concentrated on developing sets of differential equations which could explain dynamic relationships within the glucose insulin system [2]. Unfortunately, these differential models varied widely in terms of complexity, and broadly struggled to capture the non-linear dynamics of the pancreatic insulin response. Observing these attempts, Bergman sought after a minimal model which could be analysed within the stimulus response framework. Rather than attempting to model the dynamics of the entire system, he instead derived a simple mathematical model which could explain the glucose and insulin responses in plasma to intravenous glucose injections. He did this by collecting plasma glucose and insulin measurements before and following injection via modified intravenous glucose tolerance tests (IVGTT) [2]. In the following sub-section, we will detail Bergman's model, as well as discuss models which extend his work. These patient models serve as a basis for understanding many of the internal models currently used by MPC strategies for the AP.

2.1.2 Bergman Model

Bergman's minimal model details the relationship between the concentration of glucose and insulin in plasma. It can be fully described by three equations. These are given as:

$$\frac{\mathrm{d}Gluc_{iv}(t)}{\mathrm{d}t} = -p_1 Gluc_{iv}(t) - I_{is}(t)(Gluc_{iv} + Gluc_{iv,b}) + d(t) \tag{1}$$

$$\frac{\mathrm{d}I_{is}(t)}{\mathrm{d}t} = -p_2 I_{is} + p_3 I_{iv}(t) \tag{2}$$

$$\frac{\mathrm{d}I_{iv}(t)}{\mathrm{d}t} = -n(I_{iv}(t) + I_{iv,b} + \frac{IIR_{iv}(t)}{I} \tag{3}$$

Before explaining these equations, let us first define each variable. $Gluc_{iv}(t)$ is the glucose concentration in plasma; $I_{iv}(t)$ is the insulin concentration in plasma; IIR_{iv} is the intravenous insulin infusion rate; $d(t)$ is the intravenous glucose injection, and $p1, p2, p3, n$ are model parameters.

Equation (1) and Equation (3) model changes in the respective concentration of glucose and insulin in plasma with respect to time. Intuitively Equation (1) is the result of two different effects. These are the effect of glucose naturally normalizing its own concentration, and the effect of insulin catalysing glucose to normalize its own concentration from a remote compartment [2]. Equation (3) demonstrates that the change in insulin concentration is driven by the current insulin concentration, basal insulin, and the intravenous insulin infusion rate. Equation (2) represents the insulin-effect remote compartment, which mathematically links glucose and insulin concentrations. It is necessary to include this in the model because insulin must first travel to a remote compartment before it can function as a catalyst, which affects its ability to normalize glucose concentrations.

Bergman's model inspired a wave of research into other patient models, many of which are currently used for in-silico trials. For example, in this paper we use data generated from a T1D patient model proposed by [11] to learn a glucose predictor. Other commonly used patient models are those proposed by Sorensen in 1985 and Dalla Man in 2007. Sorensen's model consists of 19 differential equations and links the insulin and glucagon systems (Sorensen 1985). Dalla Man's model consists of 12 differential equations, and was approved by the FDA to be used as an artificial patient for closed loop in silico trials in 2008 [16].

2.2 Model Predictive Control

In this section we define the general structure of MPC methods. We approach this by first detailing the high-level components necessary to any MPC algorithm, before discussing the advantages and disadvantages of linear and non-linear variations. There are three key components to any MPC algorithm; these are a plant process, an internal model of the plant process, and a cost function which minimizes the error between the projected plant output and a desired setpoint value, with respect to the plant input. Other components may be necessary depending on the problem; in the case of the AP, a target projector is typically used to forecast a path of convergence between the CGM sensor readings and BG setpoint, and full state estimation is typically required for the plant model to make predictions [16]. Figure 1 depicts these components as part of an MPC control algorithm.

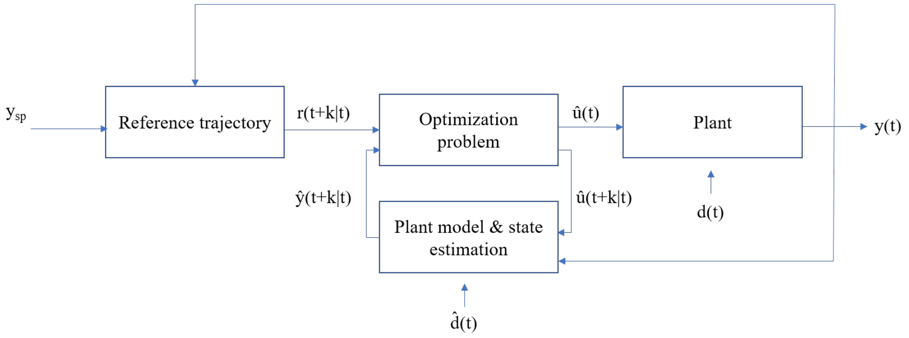

Figure 1: Model Predictive Control

In Figure 1, the target projector first computes a reference trajectory $r(t+k|t)$ (where $t+k$ is the predictive horizon and t are past observations), which ensures convergence between $y(t)$ and y_{sp}. A cost function involving $r(t+k|t)$ and $y(t+k|t)$ is then minimized with respect to the plant input $u(k)$, solving the optimization problem. The estimated solution, $\hat{u}(t+k|t)$, is next used by the plant and plant model to generate and predict the next output respectively. State estimation occurs before the plant model makes its prediction. Common methods for state estimation include the moving horizon estimation (MHE) method and Kalman filters [5]. The final step is to increment the predictive horizon by one timestep in order to complete one iteration of the loop.

The exact details of each component are to be decided by the modeller. However, MPC algorithms can be generally categorized based on whether the plant model follows a linear or non-linear specification. Linear model predictive control (LMPC) is currently one of the most popular control strategies for the commercial AP due to its simplicity and computational efficiency. Linear model specifications are easy to interpret, which is beneficial from a safety perspective as it is easy to predict the model's behaviour. They also guarantee a convex optimization problem, which can be solved in polynomial time. Many popular LMPC specifications are based on simplifications of the patient models mentioned in 2.1.2.

Non-linear model predictive control (NMPC) strategies are also based on the patient models discussed in 2.1.2. The difference is that they use the full non-linear specifications of these models, rather simplified linear versions. The trade-off is that the full specifications typically produce better control results at a higher computational cost, due to the fact that they generally do not formulate convex optimization problems. However, in some cases the higher computational cost may be worth it. For example, Magni et. al. 2008 [17] proposes a non-linear MPC strategy based on Dalla Man et. al. 2007 [6] which reduces time spent in hyperglycemia relative to its linear counterpart.

2.3 Deep Neural Networks

2.3.1 CNN

Since the inception of LeNet-5 in 1998, convolutional neural networks (CNN) have achieved state of the art results when applied to prediction problems across a variety of fields [15]. Most notably, they have been shown to excel at image classification tasks due to their localized method of learning. Widespread interest in the capabilities of the convolutional architecture soared in 2012 when a large convolutional model, colloquially named AlexNet, outperformed all other models on the ImageNet dataset by a wide margin. It achieved a top-5 classification error which was more than 10.8% lower than its nearest competitor [14]. Since then, CNN models have been both scaled to utilize the full extent of current computing power, and adapted to incorporate other advances in deep learning (eg. residual neural networks, recurrent neural networks). The remainder of this sub-section will detail the vanilla CNN architecture.

CNNs differ from traditional feed-forward neural networks in that they learn sets of weights which are then convolved across the input to each layer, rather than one global set of weights. These sets of weights are referred to as filters, and enable the model to learn different local features of the data which might be lost in a fully connected setup. Figure 2 depicts a simple 2D convolutional layer.

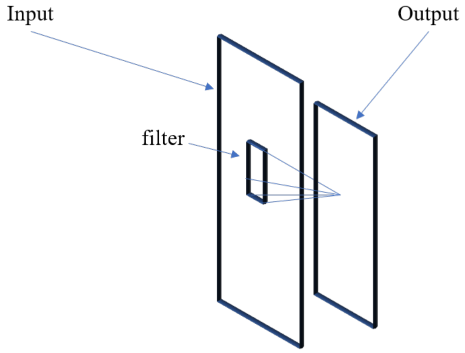

Figure 2: A simple convolutional layer

An important feature of localized weights is that they make the size of CNN models independent of the dimensions of the input data. This is not true for fully connected networks, as it is relatively straightforward to see increasing the dimensions of the input data will exponentially increase the number of parameters in a fully connected model. As in all DNN architectures, backpropagation is used to learn each filter in a CNN model.

2.3.2 RNN

In recent years, recurrent neural networks (RNN) have been demonstrated to excel at sequence prediction problems. RNNs were first conceived in the 1980s, but were notoriously difficult to train due to the vanishing gradient problem [9]. The vanishing gradient problem, with respect to RNNs, can be intuitively understood by first imagining the RNN unrolled in time such that it can be represented as a feedforward network. Figure 3 depicts a simple RNN architecture both in its compact form and as a feedforward network unrolled in time.

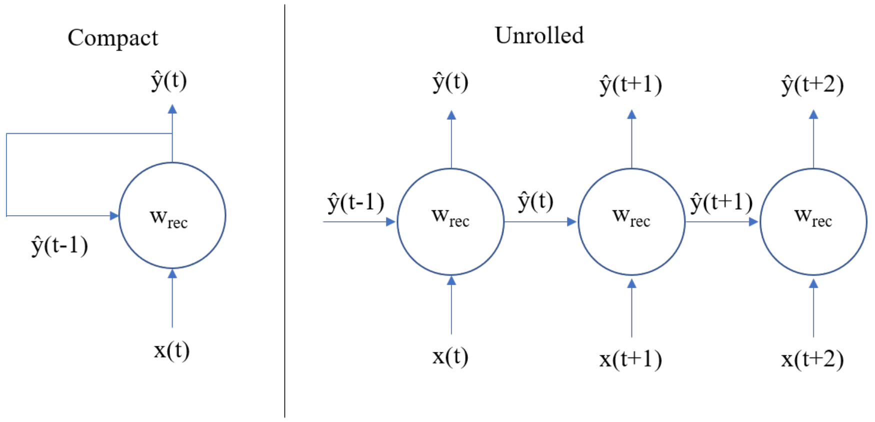

Figure 3: A simple RNN

Now the issue becomes clearer. When the weights are learned via backpropagation, the error with respect to the earlier layers becomes the product of w_{rec} many times. Thus, if w_{rec} becomes very large or very small, the gradient with respect to the earlier layers will also become very large or very small. This prevents these layers from learning any meaningful representation of the data. In subsubsection 2.3.3 we discuss how the long short-term memory variant of RNNs avoids this problem.

RNNs are particularly well suited to the problem of sequence prediction. This is due to the recurrence in the architecture, which allows RNNs to theoretically retain previously inputted information for an arbitrary amount of time. Thus, RNNs have the property of being able to make predictions using long range dependencies. To highlight why this property is useful, consider the natural language processing (NLP) problem of predicting the next word in a sequence. An examination of this problem reveals that a good solution may require context given much earlier in the text, hence making RNNs a natural modelling choice. In fact, up until the development of the Transformer architecture and attention mechanism in 2017, RNN models were a preferred solution to this problem [24].

2.3.3 LSTM

Long short-term memory (LSTM) is a recurrent layer developed to mitigate the vanishing gradient problem for recurrent neural networks (RNN) [10]. Since its first publication in 1997, it has been widely adopted by machine learning (ML) researchers when faced with the problem of time series prediction. The LSTM layer modifies the traditional RNN layer by adding two differentiable gates. These gates have intuitive interpretations; the first controls how much new information from the current timestep is discarded (a forgetting gate), while the second controls how much of the cell's memory is retained (a remembering gate). Summing the outputs of these two gates multiplied by current input and cell state respectively yields the cell's memory output. The importance of these gates is that they create two separate avenues for the gradient to propagate backwards to the cell state which are not related multiplicatively, and thus are less likely to suffer from a vanishing or exploding gradient.

Recently, DNNs comprised of LSTM layers have been used for the purpose of BG prediction. One interesting study by Mohebbi et. al. 2020 first investigates the ability of LSTM networks to generalize population-based CGM measurements to new patients, and then studies the extent to which fine tuning with patient specific CGM measurements can improve performance. They find that the population-based models out-perform both fine-tuned models and traditional autoregressive integrated moving average (ARIMA) models for 15, 30, 45, 60, and 90-minute prediction horizons, given a 20-minute historical window [19]. Another study by Sun et. al. 2018 examines the performance of DNNs with both LSTM and bidirectional LSTM layers, which will be described in the following sub-section, across 26 real and in-silico patient datasets. They also find that all LSTM networks out-perform all traditional ARIMA and support vector regression (SVR) models for 15, 30, 45, and 60-minute prediction horizons, given historical windows between 10 and 50 minutes [23]. Building on these studies, we also propose a LSTM model, which will be exactly detailed in section 4.

2.3.4 Bidirectional LSTM

The bidirectional LSTM layer is an extension of the LSTM layer. It combines two LSTM cells into a single layer, with one cell processing the input sequence in the positive time direction and the other cell processing it in the negative time direction [21]. The idea is that concatenating the output of these cells might improve predictive accuracy, as different, useful representations of the sequence might also be extracted by processing it backwards. Bidirectional LSTMs have been shown to empirically outperform other recurrent architectures in some problems such as text summarization, where it intuitively makes sense that providing the model with more context should improve results [1]. With regards to problem of BG prediction, a bidirectional LSTM layer has been used in conjunction with a vanilla LSTM layer by Sun et. al. 2018. In our paper, we propose separate LSTM and bidirectional LSTM models to isolate their effects.

2.3.5 CRNN

The convolutional recurrent neural network (CRNN) is a deep architecture comprised of both convolutional and recurrent layers. Proposed by Shi et. al. 2015 [22] for image-based sequence recognition, CRNNs first extract a sequence of local features using convolutional layers which are then processed by RNN layers. An example CRNN architecture is depicted in Figure 4.

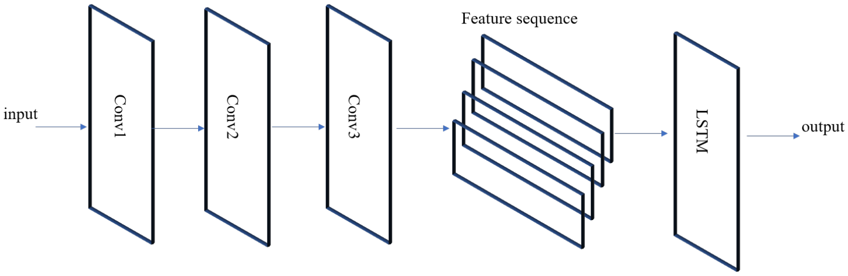

Figure 4: A simple CRNN

CRNNs are popularly applied to time series prediction problems. The idea is to use 1D convolutional layers to first extract local temporal patterns, and then to feed these patterns into a recurrent layer to predict the next timestep. The hope is that the features automatically extracted by the convolutional layers will be more informative to the recurrent layer than the raw data itself. Since time series data is ubiquitous, variations of CRNN models have been used across a range of domains from WiFi fingerprinting to speech enhancement [20], [12]. In this paper, we propose a CRNN model for BG prediction.

2.4 Approximate Bayesian Inference

2.4.1 Overview

2.4.2 Monte-Carlo Dropout

In 2016, Gal et. al. proposed a Monte-Carlo method for quantifying model uncertainty in deep learning. The method is rooted in the idea that training a DNN model with a dropout layer before each weight layer is a

Bayesian approximation [7]. The intuition behind this is that dropout creates uncertainty in the parameters of each layer, which can be exploited at test time to construct a sampling distribution of each model prediction. Therefore, it is possible to analyse the predictions of the model using approximate Bayesian inference. This is valuable for regression problems, as it allows credible intervals to be constructed around each model prediction. For sensitive tasks such as BG prediction, providing an estimate of the model's confidence in its predictions is essential to mitigate bad outcomes.

2.4.3 Credible Intervals

As mentioned in subsubsection 2.4.2, we compute credible intervals around each prediction to quantify model uncertainty. In this sub-section, we describe the procedure for computing these intervals. The procedure is fairly straightforward. We first sample model predictions 100 times for every validation example. Since dropout is applied during inference, this will create a distribution of predictions for each example. We next order the sampled predictions from smallest to largest for each example. Then to compute a $1 - \alpha\%$ credible interval for each example, we use the prediction at position $floor(100 * \alpha/2)$ in the ordered list as the interval lower bound, and the prediction at position $length(ordered\_list) - floor(100 * \alpha/2)$ in the ordered list as the interval upper bound.

2.5 Pruning

2.5.1 Overview

The development of pruning algorithms for neural networks has been a research focus in machine learning since the late 1980s [13]. The motive for developing these algorithms is straightforward. Since a network's size is directly linked to length of its run time, the smallest network which does not lose validation accuracy is the most preferred from a computational efficiency perspective. Furthermore, some domain problems may have explicit computational constraints which make large DNN models impractical, even if they can accurately model the problem. In the context of this paper, model size is an important consideration when learning a BG predictor for the AP. Smaller models can be evaluated on less computationally intensive CADs, which should help lower the economic cost of commercial AP systems.

2.5.2 Magnitude-based Pruning

The most common pruning algorithms for DNNs are based on the magnitude of the network weights. These algorithms can either be applied at the network level (i.e. globally) or to specific layers. The inspiration for pruning stems from the observation that it is difficult to know, from the outset, the optimal number of parameters to include in a deep model for a given problem. This observed problem is compounded by the fact that DNN architectures tend to be fixed at train time. The implication is that many deep models are overparameterized [8]. Magnitude-based pruning methods remedy this by learning network architecture whilst training. The exact details of the pruning procedure vary from model to model, but in general a magnitude-based pruning algorithm iteratively removes a percentage of the weights each epoch. The effect of this procedure is that it repeatedly decreases the number of trainable parameters in the model, thus providing a framework for learning the model size in addition to the weights learned via backpropagation [3]. We propose a layer-wise magnitude-based pruning scheme to miniaturize the models proposed in this paper, which is detailed in subsection 4.3.

3 Datasets

3.1 MPC Generated

The first dataset used for analysis in this paper is provided by Chen et. al. 2020. It is generated from an in silico MPC-based AP system, which uses the differential model proposed by Hovorka et. al. 2004 as its internal patient model. The dataset is synthetically generated such that full state information is always given to the plant model, and therefore the optimal insulin policy can be computed at each timestep. Chen et. al. 2020 proposed that this dataset could be used to train a DNN model which could directly map BG measurements to the appropriate insulin response. Thus, in effect, the DNN would learn to imitate the optimal insulin policy computed by MPC without having to perform its potentially error prone state estimation when used in practice [5]. They train an LSTM model to this effect and demonstrate that it outperforms MPC with state estimation on generalized patient cohorts.

We extend this work by considering a slightly different problem. Rather than using a DNN to map BG measurements to optimal insulin policies, we use a DNN to map BG measurements, optimal insulin policies, and estimated meal disturbances to future BG measurements. Thus, rather than learning a DNN representation of the insulin policy of the MPC system, we learn a DNN representation of the patient model. This is valuable as the DNN model could then be used in lieu of the plant model within the MPC system, bypassing state estimation.

The dataset is provided to us as a total of 1080 patient records, each consisting of a 1-day trajectory with a 1-minute resolution. These are broken into 756 train records and 324 test records for training and validation purposes. The dataset is publicly available at `https://github.com/nicopao/CGM_prediction_data`. In order to make the dataset more realistic to the capabilities of state-of-the-art CGM sensors, we downscaled its resolution to 10 minutes. To do this, we simply averaged every 10 observations for each provided record.

3.2 UVA Padova

The second dataset used for analysis in this paper is generated from the FDA approved UVA Padova T1D simulator. Fully detailed by Dalla Man et. al. 2014, the simulator uses sophisticated differential models of glucose kinetics to generate realistic T1D BG, insulin, and carbohydrate measurements. The simulator is well accepted amongst T1D researchers, and has been commonly used for in silico trials of closed loop AP systems. We study the ability of DNNs to imitate the behaviour of this simulator through the task of BG prediction. A close approximation of the BG curves of the simulator would indicate that the differential models are compressible into a DNN format.

The dataset is generated using 33 virtual patients. Ten 1-day trajectories with a 1-minute resolution are computed for each patient. We use the first 7 repetitions of each patient to train the DNN models, and the last 3 for validation purposes. For the same reasons given to justify downscaling the MPC generated dataset, we also downscale each trajectory in this dataset to a 10-minute resolution through simple averaging.

4 Computational Experiments

4.1 Objectives

The motivations for carrying out the computational experiments in this paper are three-fold. First, we seek to find the DNN architecture which most accurately approximates the patient model for each dataset for each predictive horizon. Naturally, this corresponds to the model with the lowest validation root mean squared error (RMSE) in the BG prediction task. The observed RMSE provides a tractable metric to analyse the loss between the given patient model and the DNN approximation.

Second, we seek to understand the compressibility of the learned DNN models. We initially train relatively large models (800K-1.32M trainable parameters) for this problem in the hopes that they will improve model accuracy. However, for the reasons given in subsection 2.5, this is not likely to result in the most computationally efficient models. Therefore, we apply a magnitude-based pruning procedure to each trained model in order to obtain an empirical estimate of the relationship between model sparsity and accuracy.

Finally, we seek to understand how confident models are in their predictions. We design our models such that evaluation is compatible with the Monte Carlo dropout method. Therefore, we are able to construct credible intervals around each model prediction via approximate Bayesian inference. Quantitative estimates of model uncertainty are important because they help us to benchmark the extent to which the DNN approximations are error prone.

4.2 Architectures

4.2.1 Overview

We train models based on three different DNN architectures in this paper; these correspond to LSTM, bidirectional LSTM, and CRNN networks. In total we train six models, one of each architecture for each dataset. All models are trained using a two-hour control horizon of past BG, insulin, and carbohydrate measurements, and output a one-hour predictive horizon of BG levels, with forecasts every 10 minutes. It is important to note that the predictive horizon is jointly generated such that predictions later in the horizon do not depend on those made earlier. Results over smaller prediction horizons can be inferred by applying evaluation metrics over only a subset of the network predictions. The remainder of subsection 4.2 will explicitly detail the parameters and hyperparameters used to construct each model.

4.2.2 LSTM

The LSTM models proposed in this paper are architecturally specified by Figure 5.

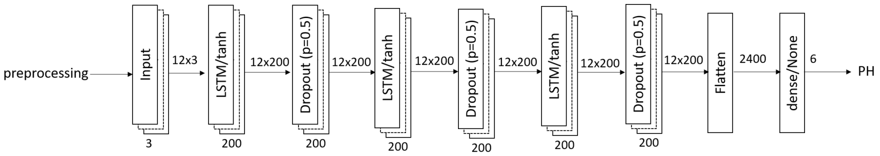

Figure 5: LSTM Architecture

As can be seen in Figure 5, we first stack three 200 unit LSTM layers, each of which is followed by a dropout layer, before flattening and jointly outputting the predictive horizon via a dense layer. The network input corresponds to the three channels (CGM, carbohydrate, and insulin measurements) over the two-hour control horizon (12 timesteps at 10-minute data resolution). The network output is the one-hour predictive horizon (6 timesteps at 10-minute data resolution). The models are trained using the Adam optimizer for weight updates and MSE loss function; they are trained for 100 epochs using a batch size of 32.

4.2.3 Bidirectional LSTM

The bidirectional LSTM models proposed in this paper are architecturally specified by Figure 6.

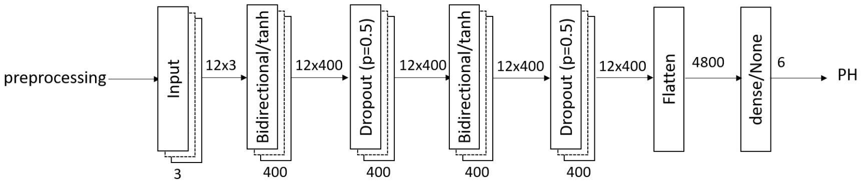

Figure 6: Bidirectional LSTM architecture

As can be seen in Figure 6, we first stack two bidirectional LSTM layers, consisting of 200 units per LSTM, each of which is followed by a dropout layer. We then flatten and jointly output the predictive horizon as we did with the LSTM architecture. Since each bidirectional LSTM layer consists of 2 LSTM cells, we use one fewer bidirectional LSTM to keep it of similar size to the LSTM models. Like the LSTM models, the bidirectional LSTM models are also trained using the Adam optimizer for weight updates and

MSE loss function. They are trained for 100 epochs using batch size equal to 32.

4.2.4 CRNN

The CRNN models proposed in this paper are architecturally specified by Figure 7.

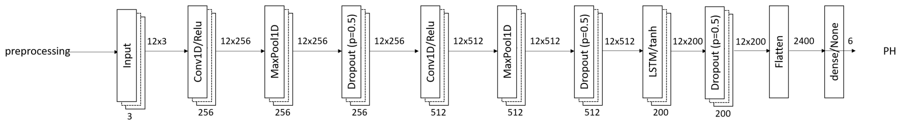

Figure 7: CRNN architecture

As seen in figure 7, the CNN consists of two consecutive 1D convolutional, max pool, dropout sequences. The first convolutional layer has 256 filters, while the second has 512 filters. Apart from that each 1D convolutional layer has the following parameters: filter size equal to 4, stride equal to 1, and same padding. Each max pooling layer has pool size equal to 2. The RNN consists of a single 200 unit LSTM layer, followed by a dropout layer. Like the other models in this paper, each CRNN is trained using the Adam optimizer for weight updates and MSE loss function. They are trained for 100 epochs using batch size equal to 32.

4.3 Pruning Procedure

We use the tensorflow model optimization (tfmot) library to prune the trained models in this paper. There are three steps to the pruning procedure. The first step is to train the unpruned version of the model for 100 epochs. The second step is to initialize the pruning scheduler from tfmot with the number of epochs to prune for and desired final model sparsity. The final step is to continue training the model with the scheduler for the specified number of pruning epochs. An important note about the scheduler is that it uses a polynomial decay function to smoothly decrease the number of trainable parameters to the final sparsity over the pruning epochs, rather than a fixed decrease each epoch. We initialize the scheduler to prune for 50 epochs in order to achieve a final model sparsity of 98% for all proposed models.

5 Results

5.1 Accuracy

5.1.1 MPC Generated

Our accuracy results for the MPC-guided dataset are presented in Figures 8 and 9.

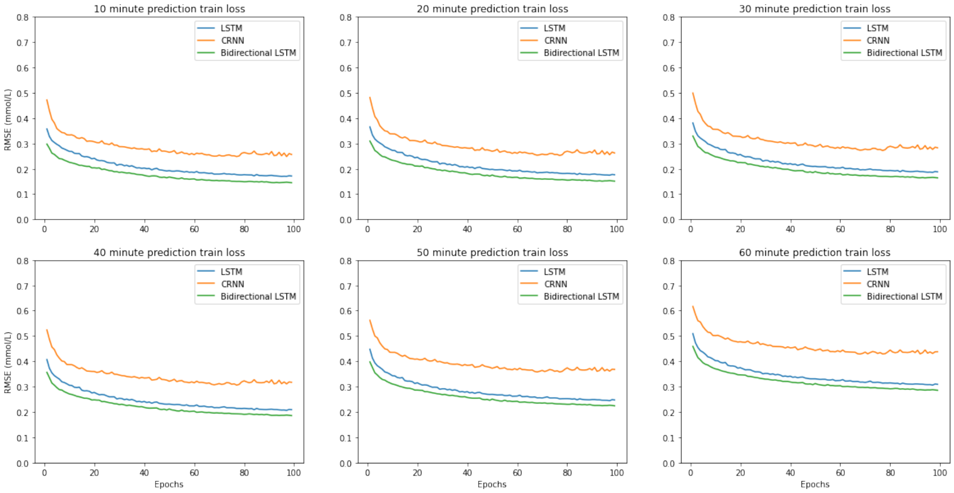

Figure 8: MPC generated train losses

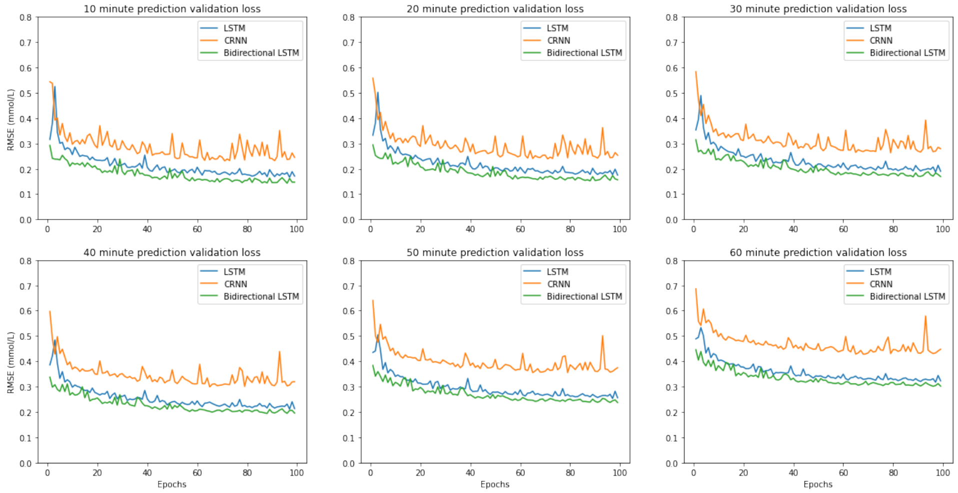

Figure 9: MPC generated validation losses

Figures 8 and 9 respectively depict the train and validation MSE between the MPC-generated examples and DNN predictions for each element of the prediction horizon. As can be visually observed, the bidirectional LSTM model achieves the best performance on this dataset for all elements of the prediction horizon, while the CRNN model performs the worst in all cases. All models become more inaccurate as they predict values further out in the horizon, with the 60 minute prediction RMSE being roughly 1.5-2x larger than the 10 minute prediction RMSE for all models. Notably the bidirectional LSTM outperforms the LSTM. The reason for this is not clear, however we can empirically infer that there might be some value to processing the control horizon in both temporal directions.

In the context of other studies which use DNNs for short term BG prediction, our results are promising. Direct comparisons are difficult as datasets and metrics vary between papers, but we can offer some analysis. Two studies similar to ours are those of Mohebbi et. al. 2020 and Sun et. al. 2018. Mohebbi et. al. 2020 uses a dataset of CGM measurements from 50 real diabetic patients gathered over a consecutive 14 day period, while Sun et. al. 2018 uses 26 real CGM datasets and synthetic data from the UVA Padova simulator for pre-training [19], [23]. Both studies use LSTM models for BG prediction, as was discussed in subsubsection 2.3.3. Since all three studies (Mohebbi et. al. 2020, Sun et. al. 2018, and ours) evaluate model performance for 30 and 60 minute prediction horizons, we use these metrics to relatively compare our results.

The best model proposed by Mohebbi et. al. 2020 is their population-based LSTM. It achieves a 1.1557 ± 0.0753 mmol/L validation RMSE using a 30 minute prediction horizon, and a 2.0317 ± 0.1397 mmol/L validation RMSE using a 60 minute prediction horizon. The best model proposed by Sun et. al. 2018 is also their LSTM. Adjusting their results from mg/dl to mmol/L, it achieves a validation RMSE of 1.207 mmol/L using a 30 minute prediction horizon and 2.0489 mmol/L using a 60 minute prediction horizon. Our best model is the bidirectional LSTM, it achieves a validation RMSE of 0.1691 mmol/L using a 30 minute prediction horizon and 0.302 mmol/L using a 60 minute prediction horizon.

From this statement of results, it seems that our model has the best out of sample performance. However, this may be misleading as the aim of our paper is to learn BG predictors which can imitate the measurements of state of the art CGM simulators, while both Mohebbi et. al. 2020 and Sun et. al. 2018 learn BG predictors to be evaluated on real CGM data. In order to make a true comparison, we would need to fine tune and evaluate our trained models on a real patient dataset. Nonetheless, our results indicate that DNNs can learn to closely predict simulated BG measurements for this dataset. Framing our results within the context of real BG prediction studies serves to highlight the predictive accuracy of these DNN models.

5.1.2 UVA Padova

Our accuracy results for the UVA Padova dataset are presented in Figures 10 and 11.

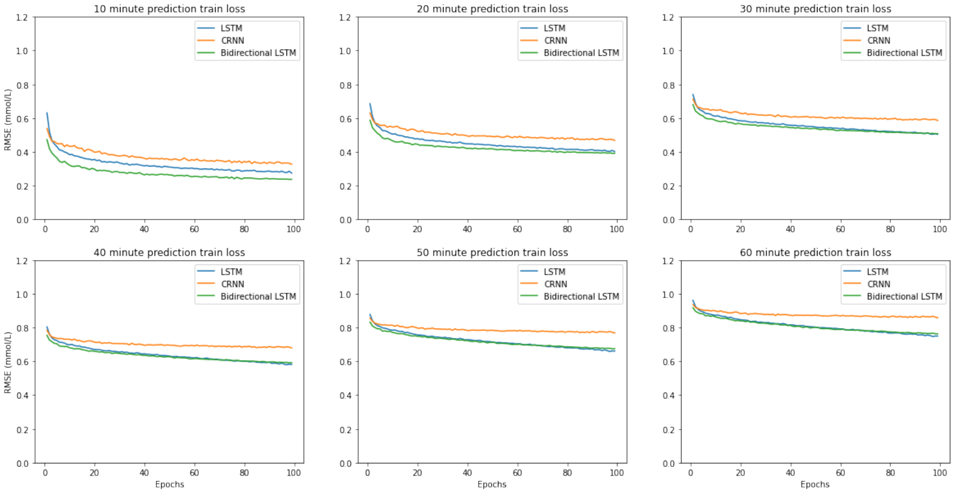

Figure 10: UVA Padova train losses

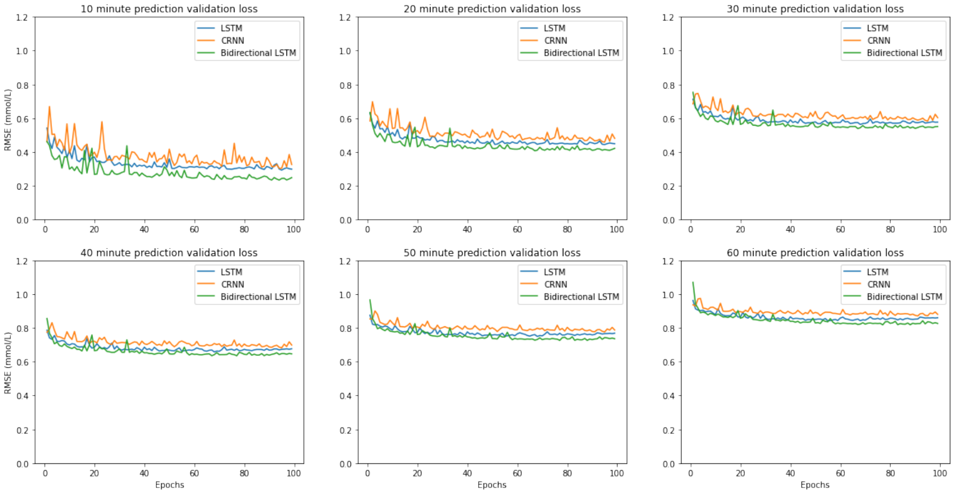

Figure 11: UVA Padova validation losses

Figures 10 and 11 respectively depict the train and validation RMSE between the UVA Padova generated examples and DNN predictions for each element of the prediction horizon. The first thing to notice is that the RMSE for every prediction in the horizon is much higher than the RMSE achieved by models trained for the MPC generated dataset. One reason for this might be that the UVA Padova dataset is generated using a basal bolus control strategy, whereas the MPC dataset is carefully generated to ensure optimal insulin control. Intuitively this could matter for a number of reasons. Most importantly, the basal bolus control strategy creates large one time bolus impulses, whereas the MPC control strategy creates a much smoother insulin curve, as it is only concerned with the optimal amount and does not distinguish between different types of insulin. Therefore, the MPC generated insulin data follows a more varied structure which might ease the BG prediction task for MPC trained deep models. Figures 12 and 13 respectively illustrate the insulin curves for the first train record for the MPC-generated and UVA Padova generated datasets.

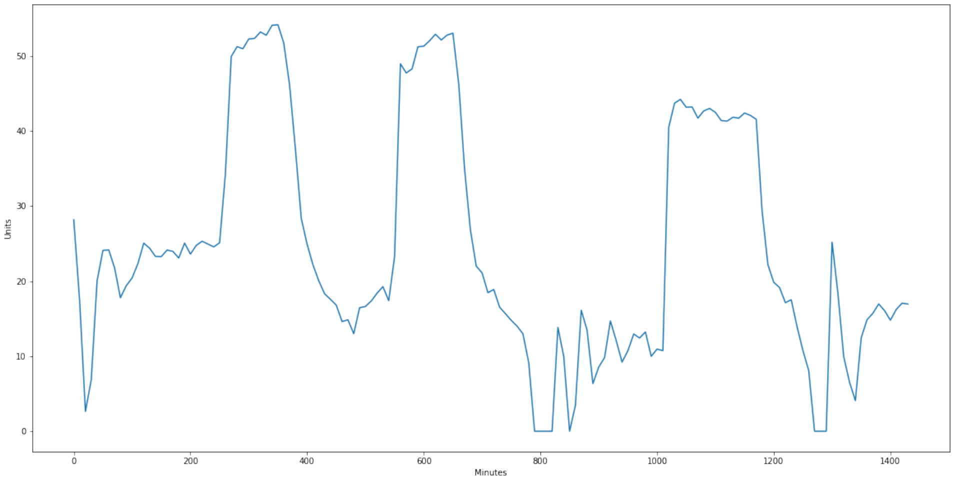

Figure 12: MPC generated total insulin

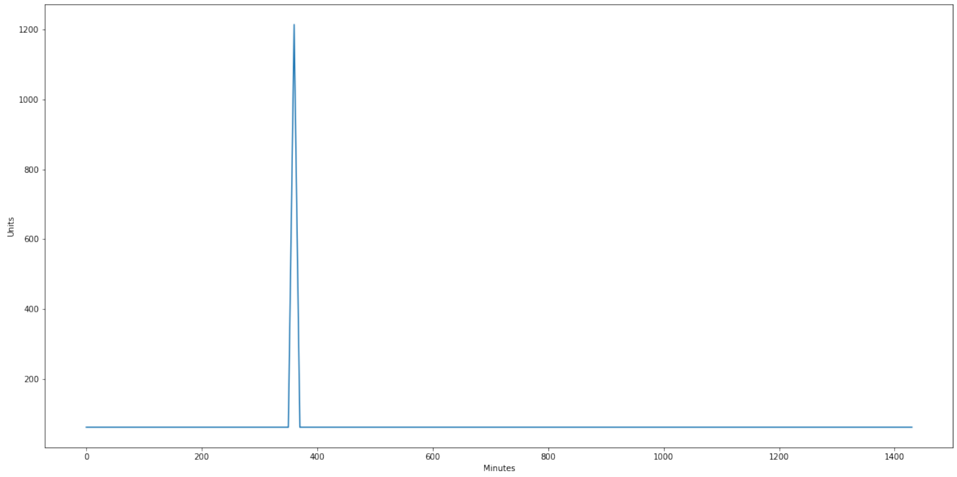

Figure 13: UVA Padova total insulin

Nonetheless, the UVA Padova trained deep models still achieve good results relative to the real BG prediction studies discussed in 5.1.1. The bidirectional LSTM is again the best performing model, achieving a final 10 minute prediction validation RMSE of 0.2484 mmol/L. Predictions made further out in the horizon are more inaccurate, as is also observed with the MPC trained models. The validation curves for each prediction in the horizon appear to plateau by 100 epochs, indicating that further training may cause the models to overfit.

5.2 Pruning

5.2.1 MPC Generated

Our pruning results for the MPC generated dataset are presented in Figure 14.

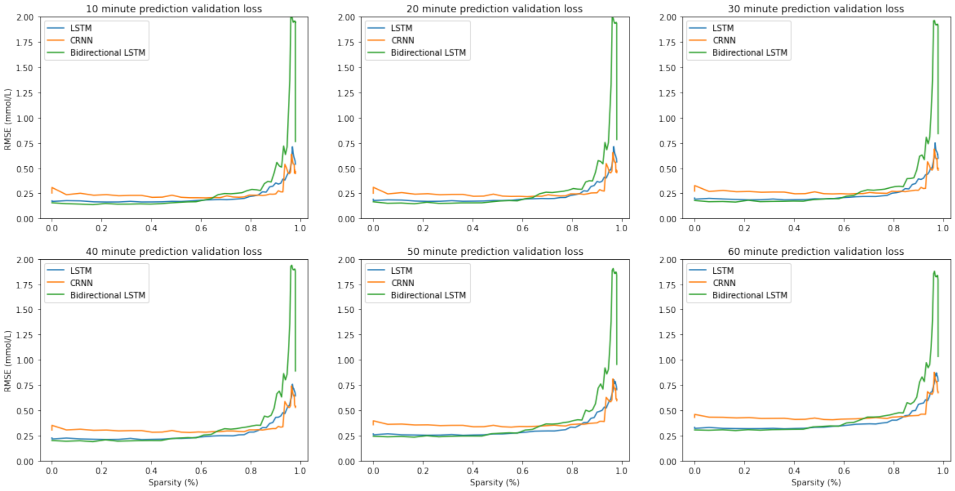

Figure 14: MPC pruning results

Figure 14 illustrates how out of sample (OOS) accuracy changes for each prediction for each model as each model becomes increasingly sparse. The first thing to notice about these results is that every model trained on this dataset can be made roughly 60% smaller without sacrificing significant OOS performance. This indicates that we initially overestimated the number of trainable parameters required for DNNs to learn to imitate the BG measurements of the patient model. For reference, the initial number of trainable parameters in the bidirectional LSTM, LSTM, and CRNN models are approximately 1.3M, 820K, and 1.1M respectively. Therefore, a reasonable initial dense model size based on these results could be between 300K and 500K trainable parameters. It is worth noting that this assessment does not factor in the inherent effect of sparsity in the pruned DNN architecture (i.e. the pruned models are not fully connected), so it is not entirely clear whether this smaller dense architecture would actually realize the same results. Testing the performance of fully connected models which contain roughly 300-500K trainable parameters could be a good extension of this work.

The next thing to notice about these results is that the bidirectional LSTM model is the most sensitive to pruning, while the CRNN and LSTM models are the least. Bidirectional LSTM OOS performance begins to worsen when the model becomes about 60% sparse, while CRNN OOS performance remains roughly the same until the model becomes about 90%

sparse and LSTM OOS performance gradually begins to worsen around 80% sparsity. These results are consistent across the prediction horizon. The bidirectional LSTM is the largest model in terms of trainable parameters and loses OOS accuracy first, so it follows that it is then the least compressible model using magnitude-based weight pruning. However, up until it begins to lose accuracy at about 60% sparsity, it is the best performing model. Therefore, the results imply that there is an architectural trade-off between accuracy and model size. In other words, the most efficient bidirectional LSTM is larger and more accurate than its LSTM and CRNN counterparts.

5.2.2 UVA Padova

Our pruning results for models trained on the UVA Padova dataset are presented in Figure 15.

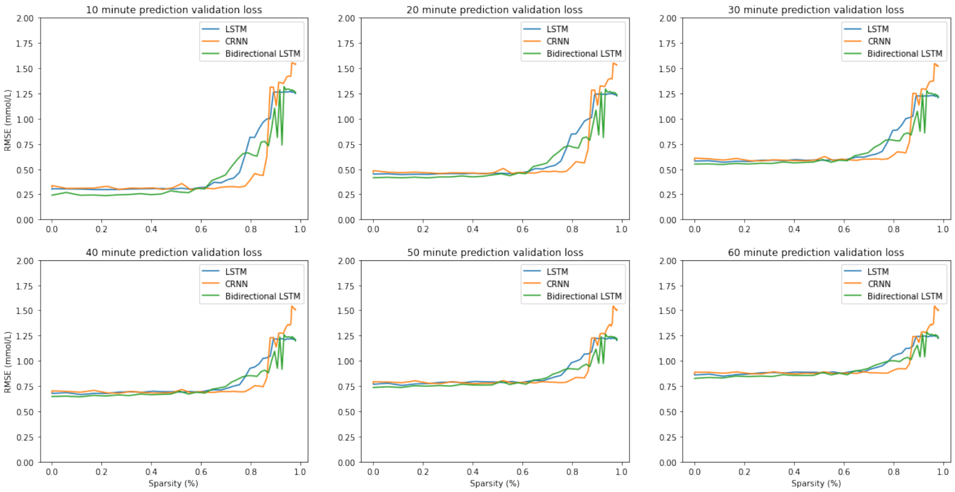

Figure 15: UVA Padova pruning results

Like Figure 14 in the MPC generated results, Figure 15 also illustrates how OOS performance changes as models become sparser. Similar to the MPC generated results, all models trained on the UVA Padova dataset can be significantly pruned without worsening OOS performance. The bidirectional LSTM and LSTM models begin to gradually lose OOS accuracy when they approach 60% sparsity, while the CRNN model worsens around 80%

sparsity. The model sizes are identical to those trained on the MPC generated dataset, so again we find that the bidirectional LSTM is the model most sensitive to pruning.

It is interesting to note that the predictions made earlier in the horizon are more sensitive to pruning than those made later. Comparing the 10 minute and 60 minute prediction RMSE plots, RMSE worsens by a much larger amount when the models begin to lose performance for the 10 minute prediction than for the 60 minute prediction. The 10 minute bidirectional LSTM and LSTM RMSE worsen by approximately 1 mmol/L at 98% sparsity compared to their unpruned counterparts, while the 60 minute bidirectional LSTM and LSTM RMSE worsen by approximately 0.5 mmol/L at 98% sparsity compared to their unpruned counterparts. This sensitivity is also observed for the MPC trained models, but it is less pronounced in absolute terms. Intuitively, the most likely explanation for this stems from the observation that the unpruned UVA Padova trained models are much more accurate for their 10 minute prediction than their 60 minute prediction compared to the unpruned MPC trained models. This can be visually observed by comparing the 10 and 60 minute validation RMSE plots Figures 9 and 11 in 5.1.

5.3 Model Confidence

5.3.1 MPC Generated

In order to better understand the precision of the DNN predictions, we compute 90% and 98% credible intervals around each validation DNN prediction. We sample the predictions of each model 100 times. These credible intervals are displayed for the first MPC validation record for LSTM, CRNN, and bidirectional LSTM models in figures 16, 17 and 18 respectively.

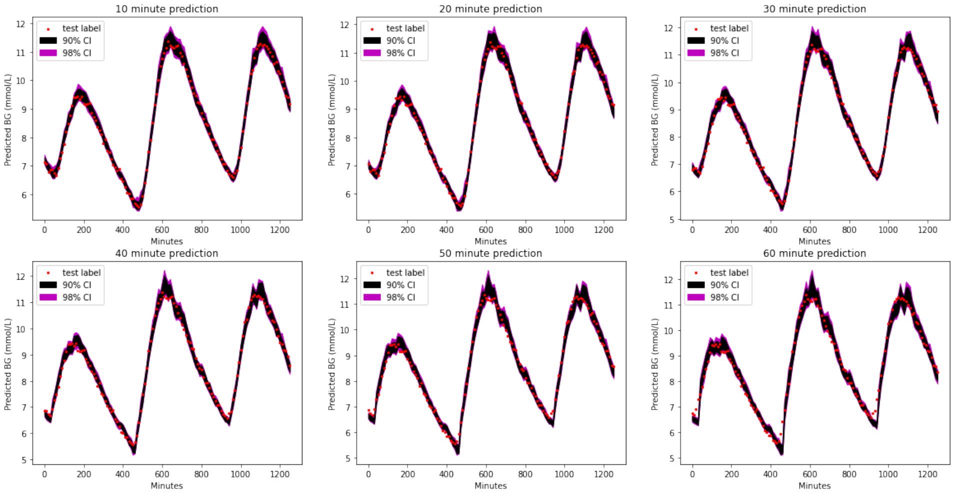

Figure 16: LSTM credible intervals

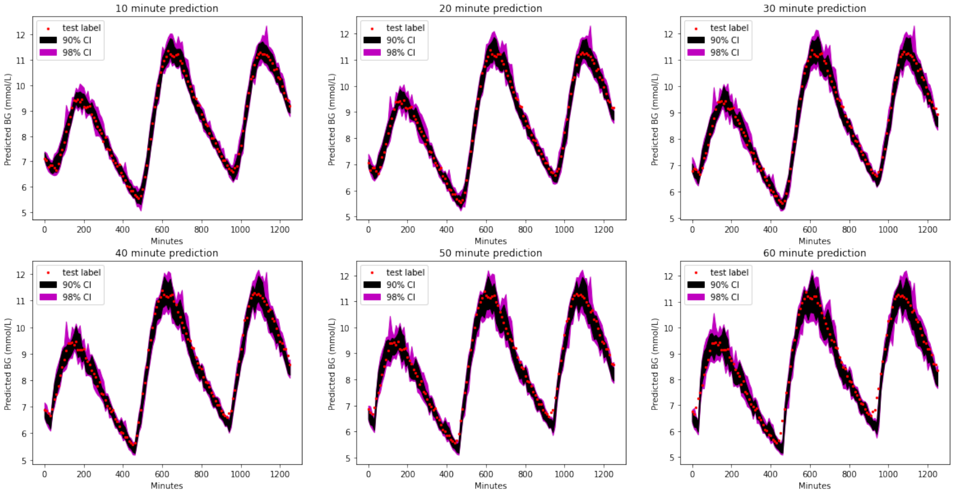

Figure 17: CRNN credible intervals

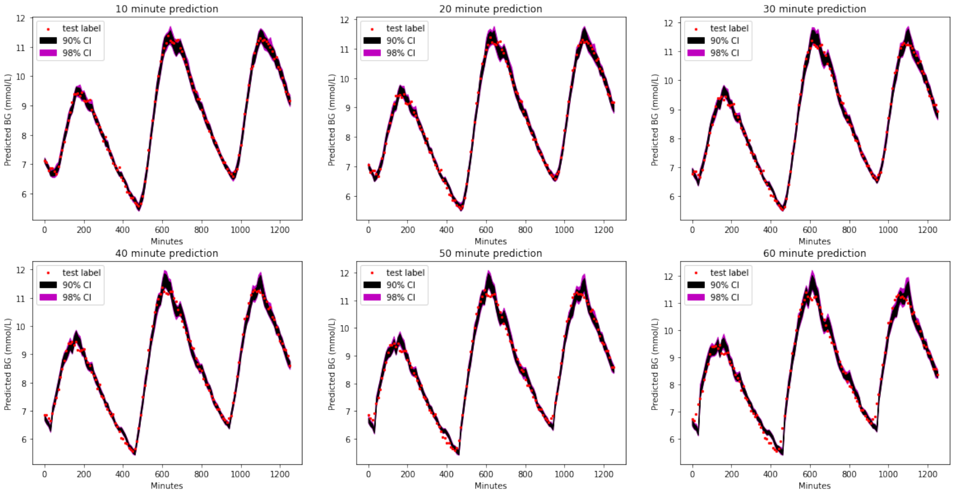

Figure 18: Bidirectional LSTM credible intervals

The first thing to note is that the bidirectional LSTM is the most confident in its predictions, whereas the CRNN is the least. This can be inferred as the bidirectional LSTM model produces the narrowest credible intervals, while the CRNN model produces the widest credible intervals. On average for the 10 minute prediction for the entire validation set, the width of the bidirectional LSTM intervals varies by 0.2504 mmol/L using a 90% credible interval and 0.3566 mmol/L using a 98% credible interval when sampling model parameters via dropout. Similarly, on average, the width of the CRNN intervals varies by 0.5676 mmol/L using a 90% credible interval and 0.9068 mmol/L using a 98% credible interval. The width of the LSTM intervals varies on average by 0.3581 mmol/L using a 90% credible interval and 0.5106 mmol/L using a 98% credible interval. These results make sense, as the bidirectional LSTM is the most accurate model while the CRNN is the least.

The second thing to note is that all models are more confident in predicting the BG response to meals than to insulin. This can be visually observed in figures 16, 17 and 18, as all credible intervals become narrower as BG levels rise, and wider as BG levels fall. Intuitively this makes sense, as it seems likely that the relationship between carbohydrates and BG levels is less complicated than the human glucose insulin system. Furthermore, all models appear to be least confident when predicting the point at which insulin begins to lower BG concentration. This is reflected by the fact that all credible intervals are widest at their peaks. This could potentially be mitigated by training the models further solely on examples where insulin begins to take effect (i.e. hard examples).

The final thing to note is that for all models the credible intervals widen for predictions made further out in the horizon. This makes sense, as subsubsection 5.1.1 demonstrated that all models become less accurate when forecasting BG levels further away in time, and hence they should be less certain. However, it is interesting for the bidirectional LSTM model in particular, that credible intervals do not widen by a large enough margin. Observing its 60 minute prediction credible interval plot, it is clear that the true test label is not contained within the credible intervals as much as it should be. Specifically around the 400 minute mark, it seems that the model should be much less certain than it is. Therefore, it can be speculated that the bidirectional LSTM is overconfident in its predictions made further out in the horizon.

5.3.2 UVA Padova

We also compute credible intervals for the models trained on the UVA Padova generated dataset. These credible intervals are displayed for the first UVA Padova validation record for LSTM, CRNN, and bidirectional LSTM models in figures 19, 20 and 21 respectively.

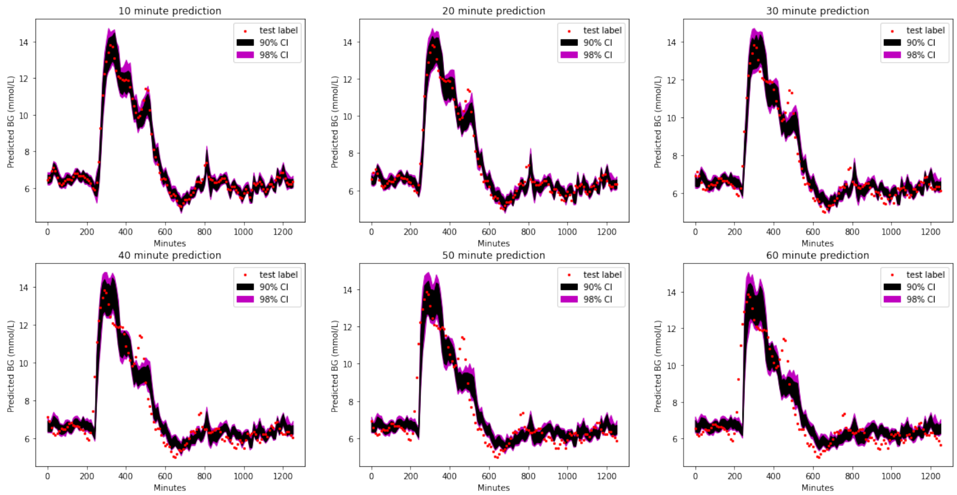

Figure 19: LSTM credible intervals

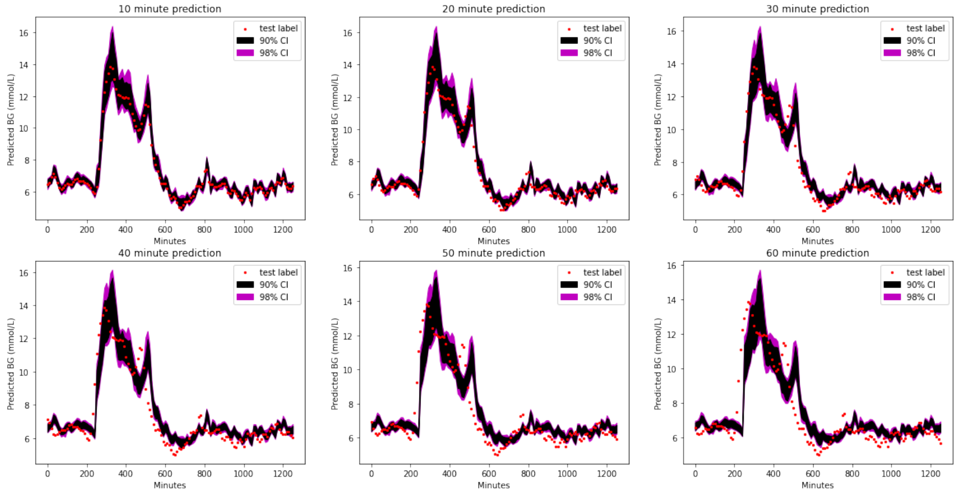

Figure 20: CRNN credible intervals

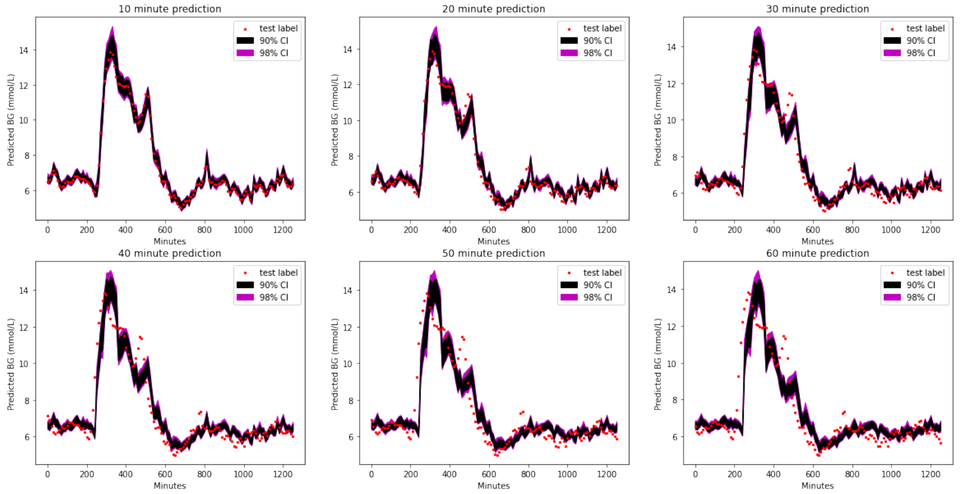

Figure 21: Bidirectional LSTM credible intervals

Similar to the MPC guided models, the bidirectional LSTM model is the most confident in its predictions while the CRNN is the least. On average for the 10 minute prediction for the entire UVA Padova validation set, the width of the bidirectional LSTM intervals varies by 0.4534 mmol/L using a 90% credible interval and 0.6451 mmol/L using a 98% credible interval; CRNN intervals vary by 0.6482 mmol/L using a 90% credible interval and 0.9331 mmol/L using a 98% credible interval; and LSTM intervals vary by 0.5859 mmol/L using a 90% credible interval and 0.8376 mmol/L using a 98% credible interval. It is also observed that the model is least confident at predicting the point at which insulin begins to take effect, and that all models are overconfident in their predictions made further out in the horizon.

5.3.3 Ensemble results

In addition to using uncertainty in the model parameters to compute credible prediction intervals, we also exploit it to create ensemble models. As with the credible intervals, we do this by first sampling model predictions 100 times. We then compute the RMSE between the test label and average prediction made by each model for every element in the validation set to compute the ensemble RMSE. MPC results are presented in Figure 22 and UVA Padova results are presented in Figure 23.

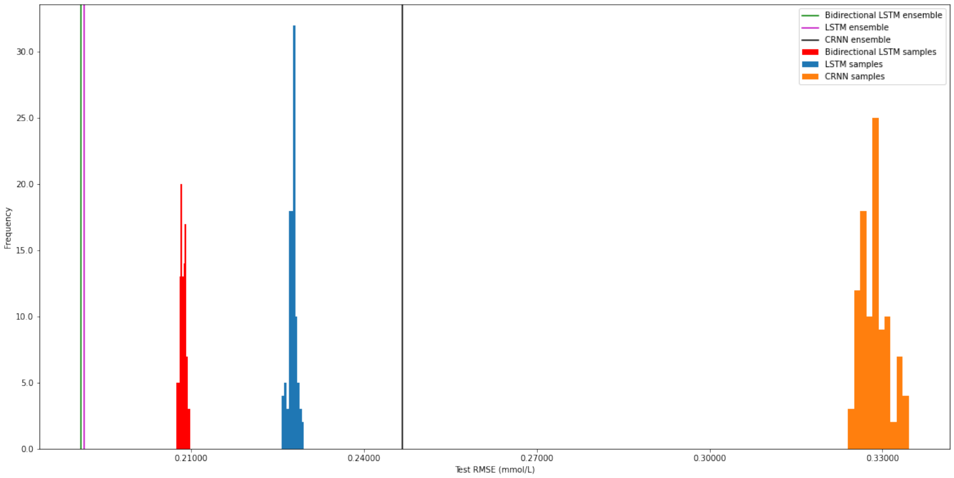

Figure 22: MPC ensemble results

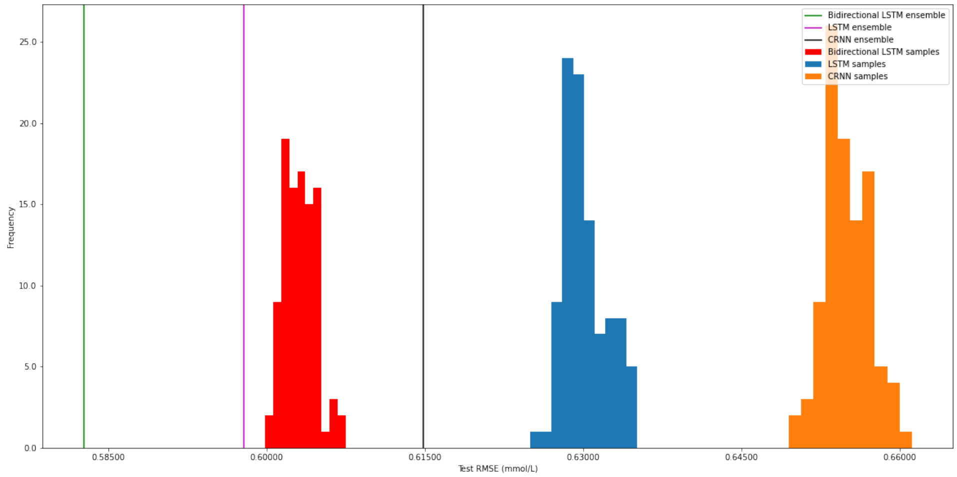

Figure 23: UVA Padova ensemble results

The histograms show the RMSE between the predictions and test label of each of the 100 inference passes, while the vertical lines show the RMSE between the average prediction and the test label (the ensemble RMSE). As can be seen in the figures, ensembling predictions improves the accuracy of all models for both datasets. This can be a significant improvement, for example the ensemble of CRNN predictions for the MPC generated dataset improves validation RMSE by approximately 0.08 mmol/L compared to a single set of CRNN predictions. It is clear that these ensemble models are less computationally efficient as they must perform more inference passes in order to compute their predictions. However, it may be possible to compress these models by training another DNN to predict the ensemble predictions. In this way, the ensemble model could be condensed into a single forward pass. This could be a good future extension of this paper, as it is clear that the ensemble models perform better than the individual models.

6 Conclusion

In this paper, we learned DNN approximations of the BG measurements of two commonly used AP patient models, using only previous BG, insulin, and carbohydrate measurements. We demonstrated that these DNN models are highly accurate relative to other deep BG models, can be significantly pruned without losing OOS accuracy, and are confident for predictions made early in the horizon. These models are valuable because they are stateless, and can be substituted into MPC based AP systems in place of the plant model to avoid potentially error prone state estimation. Learning DNN approximations is a novel method to compress well established differential models of the human glucose insulin system into simple input output representations. These input output representations are ideal for use within MPC based AP systems, since it is not necessary to recover the patient's internal state.

The work conducted in this paper can be extended to be used in real AP systems. DNN models could be first trained to predict BG measurements on MPC generated data which is derived from a number of different patient models with random initial states. Once close approximations are learned, transfer learning could be employed to fine tune the models using real patient datasets. In this way, a general BG predictor could be learned that is minimally dependent on patient state. This BG predictor could then be used within MPC to improve the AP.

References

[1] Kamal Al-Sabahi, Zhang Zuping, and Yang Kang. Bidirectional Attentional Encoder-Decoder Model and Bidirectional Beam Search for Abstractive Summarization. pages 1–9, 2018.

[2] Richard N. Bergman. Minimal model: Perspective from 2005. *Hormone Research*, 64(SUPPL. 3):8–15, 2005.

[3] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the State of Neural Network Pruning? 2020.

[4] Charlotte K. Boughton and Roman Hovorka. Advances in artificial pancreas systems. *Science Translational Medicine*, 11(484):3–6, 2019.

[5] Hongkai Chen, Nicola Paoletti, Scott A. Smolka, and Shan Lin. MPC-guided Imitation Learning of Neural Network Policies for the Artificial Pancreas. 2020.

[6] Chiara Dalla Man, Francesco Micheletto, Dayu Lv, Marc Breton, Boris Kovatchev, and Claudio Cobelli. The UVA/PADOVA type 1 diabetes simulator: New features. *Journal of Diabetes Science and Technology*, 8(1):26–34, 2014.

[7] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *33rd International Conference on Machine Learning, ICML 2016*, 2016.

[8] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, volume 2015-Janua, pages 1135–1143, 2015.

[9] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowlege-Based Systems*, 6(2):107–116, 1998.

[10] Sepp Hochreiter and Jürgen Schmidhuber. LSTM 1997. *Neural Computation*, 1997.

[11] Roman Hovorka, Valentina Canonico, Ludovic J. Chassin, Ulrich Haueter, Massimo Massi-Benedetti, Marco Orsini Federici, Thomas R. Pieber, Helga C. Schaller, Lukas Schaupp, Thomas Vering, and Malgorzata E. Wilinska. Nonlinear model predictive control of glucose concentration in subjects with type 1 diabetes. *Physiological Measurement*, 25(4):905–920, 2004.

[12] Tsun-An Hsieh, Hsin-Min Wang, Xugang Lu, and Yu Tsao. WaveCRN: An Efficient Convolutional Recurrent Neural Network for End-to-end Speech Enhancement. pages 1–5, 2020.

[13] Ehud D. Karnin. A Simple Procedure for Pruning Back-Propagation Trained Neural Networks, 1990.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 2017.

[15] Yoshua LeCun, Yann; Haffner, Patrick; Bottou, Leon; Bengio. Object Recognition with Gradient-Based Learning. (1):6–8, 1999.

[16] Katrin Lunze, Tarunraj Singh, Marian Walter, Mathias D. Brendel, and Steffen Leonhardt. Blood glucose control algorithms for type 1 diabetic patients: A methodological review. *Biomedical Signal Processing and Control*, 8(2):107–119, 2013.

[17] Lalo Magni, Davide M. Raimondo, Luca Bossi, Chiara Dalla Man, Giuseppe De Nicolao, Boris Kovatchev, and Claudio Cobelli. Model predictive control of type 1 diabetes: An in silico trial. *Journal of Diabetes Science and Technology*, 2007.

[18] Hrushikesh N. Mhaskar, Sergei V. Pereverzyev, and Maria D. van der Walt. A Deep Learning Approach to Diabetic Blood Glucose Prediction. *Frontiers in Applied Mathematics and Statistics*, 3:1–20, 2017.

[19] Ali Mohebbi, Alexander R. Johansen, Nicklas Hansen, Peter E. Christensen, Jens M. Tarp, Morten L. Jensen, Henrik Bengtsson, and Morten Mørup. Short Term Blood Glucose Prediction based on Continuous Glucose Monitoring Data. pages 1–6, 2020.

[20] Weizhu Qian, Fabrice Lauri, and Franck Gechter. Convolutional Mixture Density Recurrent Neural Network for Predicting User Location with WiFi Fingerprints. 2019.

[21] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[22] Baoguang Shi, Xiang Bai, and Cong Yao. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. *arXiv:1507.05717 [cs]*, 2015.

[23] Qingnan Sun, Marko V. Jankovic, Lia Bally, and Stavroula G. Mougiakakou. Predicting Blood Glucose with an LSTM and Bi-LSTM Based Deep Neural Network. *2018 14th Symposium on Neural Networks and Applications, NEUREL 2018*, 2018.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Transformer: Attention is all you need. *Advances in Neural Information Processing Systems 30*, (Nips):5998–6008, 2017.

A Appendix

A.1 Professional Issues

DNNs are powerful models with respect to predictive accuracy, as they have been shown to be able to approximately functionally imitate a number of processes given enough input output examples. However, fundamentally, they are black box models and as such it is impossible to discern exactly what a given DNN has learned for a given problem. Unlike other statistical fields, such as econometrics which places primary importance on interpreting model parameters to establish causal effect, deep learning is still establishing methods to better interpret and understand its trainable parameters. The current lack of understanding poses problems when applying DNNs to sensitive data, as it is not easy to predict model behaviour. One potential example of this is if DNN models are applied to household panel data (used in many empirical economic studies) to predict something like individual income. There is not a simple way to disentangle which features the network is basing its prediction off of, and it could very easily be making its predictions almost solely on things like age, race, or gender. Using a model similar to this in practice to guide real policy would be immoral.

A less theoretical example of a potentially unsafe model can be drawn directly from this paper. We learn a BG predictor to be used in an AP system based solely on previous BG, insulin, and carbohydrate readings. In this case, the safety concern does not stem from the features being used to make predictions. Rather, the issue occurs when a live sample is encountered that is very different from any training example. This could easily happen due to an errant CGM sensor reading. When this happens, the model will most likely make a random prediction, which could then cause the MPC algorithm to output a dangerous insulin dosage recommendation. We explicitly take steps to mitigate this scenario in this paper by computing credible intervals around each model prediction via approximate Bayesian inference. The hope is that if such a live sample is encountered, the model will output a very wide credible interval so that it is clear to us that it is guessing. We can then treat the MPC insulin recommendation for that timestep as unreliable, and instead substitute a safe dose.

It is especially important when designing critical automated systems, such as the AP, to seek to understand the worst case scenarios in terms of model performance. This includes adding safety measures to ensure that the model never makes dangerous predictions under any circumstances. It also means that significant work should be done to develop adversarial training examples in order to understand where the model might make bad predictions. Until enough empirical work has been conducted to demonstrate that the model has been amended to learn acceptable solutions for these examples, it should not be used for inference in cases where they are likely to occur.

In the case of the AP, any inference model used should be safeguarded to make sure that a harmful dose of insulin is never administered. This should probably involve creating a hard limit on the amount of insulin the AP is allowed to administer at one time, unrelated to the model. It should also have a method of notifying the patient (probably through the CAD) that the device is not functioning correctly so that it can be fixed or replaced. In this case the AP should cease to function and the patient should be made aware of and follow an appropriate insulin injection routine.

We specifically analyse credible intervals in this project to understand problematic examples for our models. We found that our inference models are much less certain when predicting the BG response to the effect of insulin than to changes in carbohydrates (meals). Since the BG predictions are relatively uncertain at the time when insulin begins to take effect, the MPC generated insulin control recommendations should be more carefully examined.

Further analysis that would be beneficial to gain an even deeper understanding of model uncertainty would be to assess model performance on noisy examples. Intuitively, this is a way to check that the credible intervals are meaningful. It should be observed that the credible intervals widen in accordance with the noise introduced. This corresponds to the model having less confidence in its predictions. Performing experiments to confirm that this is the case would be a good extension of this project. If it is not, then a more robust method for computing intervals should be tried.

A.2 How To Use My Project

```
myProject
├── report.pdf
├── notebooks
│   ├── confidence.ipynb
│   ├── accuracy.ipynb
│   └── pruning.ipynb
├── py
│   ├── train.py
│   ├── confidence.py
│   ├── accuracy.py
│   └── pruning.py
└── saved
    └── history
```

Figure 24: Directory structure

The directory structure of my project submission is detailed in Figure 24. The project paper can be found in report.pdf. Compiled jupyter notebooks which contain the code and all result plots can be found in the notebooks subdirectory. There are three notebooks which correspond to each objective of the computational experiments (accuracy, pruning, confidence). The code to generate the figures in each results section can be found in the corresponding notebook (accuracy results are in accuracy.ipynb, pruning results are in pruning.ipynb, and credible intervals and ensembling are in confidence.ipynb). It is recommended that you use the notebooks to inspect the code and results, as this is where the computational experiments were performed. The notebooks are not meant to be re-evaluated, as the datasets and saved models are not present (they would exceed the 100mb submission limit).

We also include executable files to train, prune, and generate the results. The following software must be installed for the files to run: Python 3.6, tensorflow 2.3.0, numpy 1.18.1, tensorflow-model-optimization, mat4py, matplotlib, tqdm, and scikit-learn. It is highly recommended that you do not retrain or re-prune the models, as this is time and computationally intensive. However, if you do wish to do so, you may retrain the models as follows. First you must download both datasets exactly as follows. Clone this repository: https://github.com/nicopao/CGM_prediction_data in the py subdirectory and download https://www.dropbox.com/s/mbnwlfe6yvcf597/uva-padova-data.zip?dl=0 zipfile in the py subdirectory. Now unzip the uva-padova-data.zip file in the py subdirectory. After you have downloaded the datasets, you may train the models using the following command: python3 train.py. By default this will train an LSTM model on the MPC generated dataset. You can choose which dataset and model to train by passing additional command line arguments. For example, to train a CRNN model on the UVA Padova dataset run the following command: python3 train.py --model crnn --dataset uva.

To prune or regenerate the results, you must first follow the steps given to download the datasets. Once this is done, you must also download the saved models. Clone this repository: https://github.com/mattliston/postgraduate_dissertation.git in the saved subdirectory. Now you may run any of the .py files. Some of these will take a long time to run (confidence.py will take around 30 minutes, pruning.py will take hours using a GTX 1060 GPU)

A.3 Self Assessment

I found that the project was very rewarding. I spent at least 600 hours working on it over the project duration. I gained significant knowledge about the human glucose insulin system, as well as MPC, pruning, DNNs, and Bayesian inference. I think that my biggest strength was that I consistently worked on the project every day, so that I never felt overwhelmed. I plan to continue to work on learning BG predictors this fall by fine tuning these models to a real dataset.