Comparing Neural Network Methods to Predict Hypoglycemic Events in Type II Diabetic Patients

Tom Kirsh

November 21, 2023

Contents

Abstract

Abstract is written at the end.

1 Introduction

When food in consumed, the body breaks it down into sugar or glucose, which gets released into the bloodstream. If the glucose levels in the bloodstream get too high, the pancreas will release insulin to lower the blood sugar levels. Diabetes is when the pancreas does not produce enough insulin to lower the glucose levels or does so inefficiently. In these cases, glucose that remains in the bloodstream can cause problems in vision, kidneys, and the heart [CDC citation]. Taking medication, exercise and diet, and managing blood sugar levels are steps to prevent adverse outcomes, but there is not really a cure yet.

There are two types of diabetes. Type I diabetes is believed to be an autoimmune response where the immune system attacks [insert answer here] and the insulin production is halted. People with Type I diabetes need to take daily insulin. Type II diabetes is when the body doesn't use insulin well and blood glucose levels are not maintained.

Continuous Glucose Measurement Systems (CGMS) measure blood glucose levels by... These are prevalent in ... CGMS is typically used in people with Type I diabetes [source].

Deep learning models, or models using neural networks with many layers, have been used as state-of-the-art classification models for sequential data, including time series. There are a few neural network architectures, that is the design, that lead to better modeling of time series data.

1.1 Predictive Modeling

Multiple types of neural networks are compared to evaluate their performance, training time, and computational power.

1.1.1 Logistic Regression

Logistic regression is a binary classification approach that estimates the probability of belonging to either class. The logistic function is given by

$$\log \frac{P(C_1|\mathbf{x})}{1-P(C_1|\mathbf{x})} = \mathbf{w}^T\mathbf{x} + w_0 \tag{1}$$

where $\mathbf{x}$ is our design matrix, the features in our dataset. $\mathbf{w}$ is the weight vector or coefficients that are estimated in the model. These coefficients are typically exponentiated into the odds ratios that denote the . The logistic function is equivalent the sigmoid function expressed as

$$P(C_1|\mathbf{x}) = sigmoid(\mathbf{w}^T\mathbf{x} + w_0) = \frac{1}{1+\exp -(\mathbf{w}^T\mathbf{x} + w_0)} \tag{2}$$

The parameters can be estimated by either the maximum-likelihood estimation or by optimizing the parameters as discriminants to minimize the error using gradient descent. [cite book]

1.1.2 Random Forest

The Random Forest (RF) classifier is a collection of decision tree predictors that takes as input the data and independent identically distributed random vectors for each tree. These trees vote on the class of the input data and its prediction is assigned based on majority rule [cite RF paper]. Features are randomly selected at each node in the tree to determine the split. As the number of trees increase, the generalization error converges, reducing the overfitting that occurs. Random forests also have the ability to assess feature importance using the values in each feature after the tree has been constructed, randomly permuting the data and the out-of-bag (OOB) sampling is passed dow the tree. That classification is saved and the process is repeated for the remaining features. Afterwards, the misclassification error is calculated and used to get the overall importance by examining the percent increase in the misclassification rate [cite RF paper].

1.1.3 Artificial Neural Network

Artificial Neural Networks (ANNs) began by trying to model neurons in the brain. The most basic artificial neural network (ANN) consists of an input layer $\mathbf{X} \in \mathbb{R}^{n\times d}$, a hidden layer $\mathbf{H} \in \mathbb{R}^{n\times h}$, and an output layer $\mathbf{O}$. These can correspond

to information coming to the neuron cell via its dendrites, the synaptic strengths modeled by weights that are updated with each epoch in the forward and back propagation steps, and the output signals passed through an activation function σ.

$$\mathbf{O}_0 = \sigma(\mathbf{X}_0\mathbf{W}_0 + \mathbf{b}_0) \tag{3}$$

These can be used in either classification or regression problems. "Deep" ANNs consist of more than one hidden layer. For those, the outputs of the hidden layer are used as inputs to the subsequent layer.

$$\begin{aligned}\mathbf{H}_1 &= \sigma(\mathbf{O}_0\mathbf{W}_1 + \mathbf{b}_1)\\ &\vdots\\ \mathbf{O}_n &= \sigma(\mathbf{H}_{n-1}\mathbf{W}_n + \mathbf{b}_0)\end{aligned}$$

The most common activation function used it the ReLU function because of its good performance [cite].

$$ReLU(x) = max(x, 0) \tag{4}$$

To train the neural network, we want to find the optimal weights that minimize the error function between the predicted output and the expected output using gradient descent. The error function we minimize is the binary cross-entropy function, commonly used for binary classification [cite book].

$$E(\mathbf{w}, w_0|X) = -\sum_i y_i log(p_i) + (1 - y_i) log(1 - p_i) \tag{5}$$

The partial derivatives of the error function with respect to the weights are computed in two passes, a forward pass and a backward pass. The forward pass computes the weights as the inputs are passed through the model and the outputs p_i using the sigmoid function. The backward pass computes the partial derivatives of the error function with respect to the weights computed in the forward pass using the chain rule and is used to change the weights by. The change in weights is calculated by

$$\Delta w = -\eta \frac{\partial E}{\partial w} \tag{6}$$

A major drawback of this method is that it doesn't converge as quickly as methods that use higher order derivatives. To that end, I use the Adam optimizer when training my neural network models. Adam (adaptive moment estimation) is an

efficient stochastic optimization algorithm that only uses the first-order gradients. This method estimates first and second moments of the gradients and computes specific adaptive learning rates for the different parameters for these moments.

Additionally, there are certain layers that can be added to reduce overfitting called Dropout layers. The idea is that after the feedfoward layer, a certain fraction of nodes will be dropped to prevent the model from completely learning the training data [cite].

Feedforward networks have been shown to have good classification predictions [cite] and thus we used different variations to model on. Hyperparameters trained on were the number of layers (what makes it a "deep" network), the number of nodes in each layer, the learning rate, and the dropout rate. Dropout layers were included to reduce overfitting [cite] by randomly removing a fraction of the nodes in the hidden layer.

1.1.4 RNN

Recurrent Neural Networks (RNNs) are used for any data that can be modeled as a sequence (language, time series, etc.) where the previous data is important for the current state of the data. [cite] To have the previous information persist, the RNN cell contains a hidden state that contains the "memory". The hidden layer output $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ is saved after the previous time step ($\mathbf{H}_{t-1}$) and used in conjunction with the weight matrix for the hidden layer output $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ to describe how the information from the previous time step(s) is included with the current time step. The RNN cycles the information back unto itself and takes into account the previous inputs $\mathbf{X}_{t-1}$ as well as the current input $\mathbf{X}_t$. The output is then passed through an activation function ϕ_h to return the current hidden state, commonly using the hyperbolic tangent [cite].

$$\mathbf{H}_t = \phi_h(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h) \tag{7}$$

where W_r is the weight matrix for the previous state and W_x is the weight matrix for the input data, similar to feedforward networks, and ϕ_o is the output activation function.

$$\mathbf{O}_t = \phi_o(\mathbf{H}_t \mathbf{W}_{ho} + \mathbf{b}_o) \tag{8}$$

1.1.5 Backpropagation Through Time

Training a recurrent neural network is similar to training a feedforward network, with the added parameter of time to consider. The backpropagation algorithm can be adapted to take time into account [cite BPTT paper]. Starting wih

$$\mathscr{L}(\mathbf{O},\mathbf{Y}) = \sum_{t=1}^{T} \ell_t(\mathbf{O}_t,\mathbf{Y}_t) \tag{9}$$

where $\mathbf{O}$ is the predicted output of the RNN and $\mathbf{Y}$ are the actual values. Just like was done for a feedforward network, the weight matrices $\mathbf{W}_{xh}$, $\mathbf{W}_{hh}$, $\mathbf{W}_{ho}$ will be updated using the partial derivative of $\mathscr{L}$ with respect to each weight matrix. Again, the chain rule needs to be applied. For the output weight matrix $\mathbf{W}_{ho}$, the loss is a function of the output layer, which is a function of the activation function ϕ_o, which is a function of $\mathbf{H}_t$ and $\mathbf{W}_{ho}$. The partial derivative of the output activation function with respect to the output weight matrix is $\frac{\partial \phi_o}{\partial \mathbf{W}_{ho}} = \mathbf{H}_t$. Knowing that the partial derivative of the loss with respect to the output weight matrix is

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}_{ho}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \frac{\partial \phi_0}{\mathbf{W}_{ho}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{H}_t \tag{10}$$

The activation function ϕ_h is dependent on $\mathbf{H}_t$ and the hidden state weight matrix and the input weight matrix.

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}_{xh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \frac{\partial \phi_0}{\partial \mathbf{H_t}} \cdot \frac{\partial \mathbf{H}_t}{\partial \phi_h} \cdot \frac{\partial \phi_h}{\partial \mathbf{W}_{xh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{W}_{ho} \cdot \frac{\partial \mathbf{H}_t}{\partial \phi_h} \cdot \frac{\partial \phi_h}{\partial \mathbf{W}_{xh}} \tag{11}$$

Similarly, for the hidden state weight matrix

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \frac{\partial \phi_0}{\partial \mathbf{H_t}} \cdot \frac{\partial \mathbf{H}_t}{\partial \phi_h} \cdot \frac{\partial \phi_h}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{W}_{ho} \cdot \frac{\partial \mathbf{H}_t}{\partial \phi_h} \cdot \frac{\partial \phi_h}{\partial \mathbf{W}_{hh}} \tag{12}$$

Note that each $\mathbf{H}_t$ is dependent on the previous time steps. We can expand this using the summation

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}_{xh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{W}_{ho} \cdot \sum_{k=1}^{t} \frac{\partial \mathbf{H}_t}{\partial \mathbf{H}_k} \cdot \frac{\partial \mathbf{H}_k}{\partial \mathbf{W}_{xh}} \tag{13}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial \mathbf{O}_t} \cdot \frac{\partial \mathbf{O}_t}{\partial \phi_o} \cdot \mathbf{W}_{ho} \cdot \sum_{k=1}^{t} \frac{\partial \mathbf{H}_t}{\partial \mathbf{H}_k} \cdot \frac{\partial \mathbf{H}_k}{\partial \mathbf{W}_{hh}} \tag{14}$$

The term $\frac{\partial \mathbf{H}_t}{\partial \mathbf{H}_k}$ depends on the individual gradients which accumulate over time, have the potential to to grow exponentially, exploding. In a different situation where the original sequence grows larger, the response will eventually flatten, as this term approaches zero, vanishing. This means that the training of the RNN is puzzled by the exploding or vanishing gradients. The most effective solution to this problem is the Long Short-Term Memory network.

1.1.6 Long Short-Term Memory

The vanishing gradients problem led to the creation of Long Short-Term Memory (LSTM) neural networks. These networks are capable of long-term dependencies, meaning they can retain older information easier than vanilla RNNs [cite]. LSTMs can do that by utilizing a cell state, which is adjusted at different stages. Instead of one layer inside the RNN cell, in the LSTM cell there are four layers: the forget gate layer, the input gate layer, the update layer, and the output gate layer. The sigmoid function wrapping the forget, input, and output gates is used to limit the values in the range [0,1]. Values in this range tell the cell state to forget, get rid of, or not output if the value is 0 or remember, keep, or return is the value is 1 (with weights in between) [cite LSTM website].

$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f) \tag{15}$$

The forget gate (Eq. 15) decides which information to include or forget. After that, the input gate (Eq. 16) decides what to include. This is multiplied element-wise to the update layer of the new values (Eq. 17) to update the state. The hyperbolic tangent is used to limit the values between [-1, 1], helping limit the gradients [cite Funda LSTM].

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i) \tag{16}$$

$$\widetilde{\mathbf{C}} = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c) \tag{17}$$

Next, the cell state is updated by adding the element-wise multiplication ($\odot$) of the forget gate and previous cell state to the element-wise multiplication of the

input gate and new cell values as shown in Eq. 18.

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \widetilde{\mathbf{C}}_t \tag{18}$$

The output gate filters what comes out (Eq. 19) and is multiplied element-wise by the hyperbolic tangent of the cell state (Eq. 20), which limits the state to between [-1,1], to output the parts previously decided.

$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o) \tag{19}$$

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t) \tag{20}$$

The outputs of the LSTM can be the next predicted values in the sequence or the probabilities of the next value belonging to a class.

1.2 Anomaly Detection

Since the number of positive samples in each participant's data is $< 1\%$, another way to build a model is to look at those glucose measurements as anomalous to "normal" glucose measurements. The key principle behind anomaly detection is the distinction between "normal" and "anomalous" data. In anomaly detection, the model is trained on the normal data to learn it. When the model is applied to the test data, containing both the normal and anomalous data, the output is compared to the original in some manner, and depending on the error, classifying the samples based on the error. The model is optimized using the mean-squared error between the reconstructions and inputs. The error between the reconstructions and training data is calculated and from that distribution, a threshold is chosen [cite]. When applying the model to the test dataset, if the errors are greater than the threshold, the sample is reported as anomalous, normal otherwise.

1.2.1 Autoencoder

The autoencoder model is a neural network model consisting of an encoder-decoder setup, in which the input is "encoded" to a lower dimension by reducing the number of nodes in the hidden layer [cite conventional autoencoder]. The decoder takes the output of the hidden layer and reconstructs the input.

SHOW IMAGE HERE

2 Methods

The data was collected using two CGM sensors attached to each participants' arms [cite Lisa's study]. Over a period of 16 weeks, participants' glucose measurements were collected from both sensors. Participants were randomly given a double-blinded CGM system, meaning they couldn't see their glucose measurements, or a blinded/unblinded CGM system. Out of the 40 participants, 22 were classified as high risk for hypoglycemic events, and indeed had multiple hypoglycemic events. We excluded 4 participants that had $<$10 events and used 18 participants' glucose data. Hypoglycemic events are defined as glucose levels $<$54 mg/dL measured by both CGM sensors at the same time lasting $\geq$15 minutes.

2.1 Data Processing

Over the measurement time period, both sensors did not measure glucose simultaneously. There were periods where either the left or right sensor measured, neither sensor measured, or both sensors measured. Even when both sensors measured, there were small delays in time between the left and right sensors' measurements. To synchronize the left and right glucose measurements, only the times that both sensors were measuring were kept. To handle the small time delays ($<$15 minutes), linear interpolation was used to map the measurements to a single time array [cite for linear interpolation] [cite for justification in signal processing] [cite in other CGM use?].

After the left and right sensors were interpolated, missing values were imputed using last-operation-carried-forward (LOCF) [cite]. The time series was converted into a a windowed or "chunked" format where the previous 6 hours (24 measurements) of data were used to predict the outcome [cite]. During modeling, some sequences were padded with NaN values at the beginning of the signals and were not imputed with LOCF. During modeling, these values were dropped.

There were multiple combinations of models constructed. They were:

2.1.1 Model Type:

- Individual: Only uses glucose data collected from a single person.

- General: Uses glucose data collected from everyone except one person, whose data is used as holdout.

2.1.2 Event Horizon:

- 15 minutes before
- 1.25 hours before the event
- 2.25 hours before

2.1.3 Sensors Used:

- Left

2.1.4 Data Augmentation:

- None: Do not change the number of samples.
- Synthetic Minority Oversampling Technique (SMOTE): Uses k-nearest neighbors of the minority class to generate more samples of similar minority-set data. [cite]
- Random Undersampling: Randomly selects 4x the number of the minority class of the majority class for training. [cite]

All combinations of these data were modeled and compared for each subject.

2.2 Hyperparameter Optimization

Neural networks have many hyperparameters that can be adjusted in order to improve the modeling capabilities. To find the best combination of hyperparameters, one method is the grid search, testing all combinations and selecting the one with the highest performance. More recent methods that seem to identify good hyperparameter configurations more quickly are Bayesian optimization methods [cite]. Even more recent is the Hyperband algorithm that uses adaptive resource allocation and early-stopping to quickly identify the best configuration. This is the algorithm used to tune the hyperparameters in the models.

Hyperband starts by considering the number of possible configurations with the

allocated budget, basically performing a grid search over realistic values. This is used to determine the optimal combinations to evaluate the performance of, remove the worst half, then repeat until the last configuration, in the same method as the SuccessiveHalving algorithm. Relatedly, the minimum resource allocated to all the configurations are compared in the outer loop. The Hyperband algorithm was applied in a similar manner as the example given in Tensorflow [cite].

The table showing the range of hyperparameter values for each model tested are shown in Table REF.

3 Results

3.1 MLP

TEST RESULTS with Hyperband Optimization

4 Discussion

5 Conclusion