Deep Residual Time-Series Forecasting: Application to Blood Glucose Prediction

Harry Rubin-Falcone[1], **Ian Fox**[1] , **and Jenna Wiens**[1]

Abstract. Improved forecasting of blood glucose could aid in the management of diabetes. Recently proposed neural network architectures that use stacked fully connected layers with residual backcasting have achieved state-of-the-art performance on benchmark time-series forecasting tasks. Though promising, previous work ignores opportunities for additional supervision, and the use of fully connected layers fails to account for the temporal nature of the signal. Here, we propose a new architecture that builds on previous work by learning to forecast gradually in stages or blocks. Our updates include replacing the fully connected block structure with a recurrent neural network and adding additional losses to provide auxiliary supervision. In addition, we leverage important context in the form of additional input signals. Applied to the task of glucose forecasting, we find that each of these modifications offers an improvement in performance. On the task of predicting blood glucose values 30 minutes into the future, our proposed approach achieves a mean rMSE of 18.2 mg/dL, versus 21.2 mg/dL achieved by the baseline. These improvements get us closer to predictions that are reliable enough to be used in CGMs or insulin pumps to manage diabetes.

1 Introduction

Accurate blood glucose forecasting would improve diabetes treatment by enabling proactive treatment [11]. To this end, there has been significant interest in developing time-series forecasting methods for predicting blood glucose levels, using a large variety of statistical and machine learning methods [9]. This has inspired the OhioT1DM challenge [6], where participants are tasked with accurately forecasting blood glucose values in individuals with type 1 diabetes. As our entry to this competition, we build on recent work in time-series forecasting that focuses on iterative residual prediction, specifically Neural Basis Expansion for Interpretable Time-Series Forecasting, or N-BEATS [8]. This work uses a neural network architecture consisting of network *blocks* that output both a *forecast* and a *backcast* (*i.e.*, a reconstruction of the block's input). The backcast is subtracted from the block's input forming a *residual* which then serves as input to the following block. At the final layer, the forecasts from each block are combined to form the final prediction. The iterative and residual nature of this architecture aims to encourage gradual signal reconstruction and forecasting. We build upon this idea in several ways (**Figure 1**):

- We replace the fully connected architecture in each block with a recurrent neural network (RNN) [12]. We hypothesize that this will allow the model to more accurately capture important temporal relationships within the input/output.

[1] University of Michigan, USA, email: hrf@umich.edu

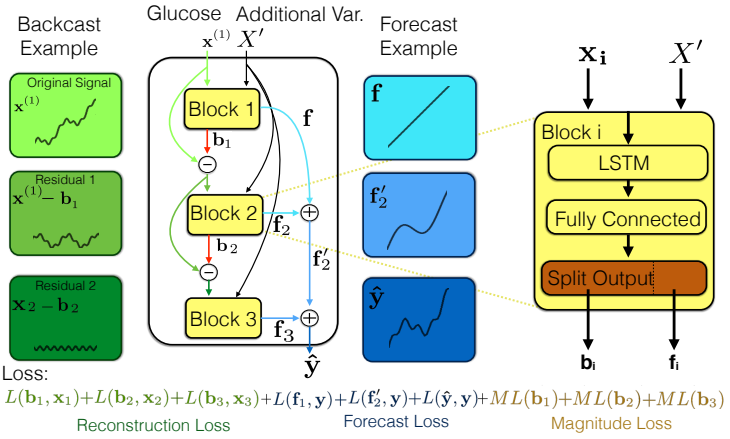

Figure 1. Our glucose forecasting architecture, shown with three blocks. Each residual block contains a bidirectional LSTM with a single output layer that produces the forecast and backcast simultaneously. Additional variables are added as input channels to each block, but residuals are calculated only for the glucose signal. Losses (described in section 2.2) are calculated after each block. Here, *L* is the primary loss function (MSE) and *ML* is magnitude loss, which is a penalization for small signals (the inverse of the norm).

- We include additional variables as input to the model (*e.g.*, bolus insulin data), which we hypothesize will provide valuable context. However, due to the sparse nature of some of these variables, we backcast only on our primary variable of interest: blood glucose.
- We include additional loss terms that act as auxiliary supervision. We hypothesize that these terms will further encourage the model's blocks to gradually learn accurate components of the signal (*i.e.*, backcasts and forecasts) that sum to the correct signal.

Additionally, we study the effects of pre-training with related datasets. Applied to the task of glucose forecasting, these modifications lead to notable improvements over baseline.

2 Methods

Given the strong performance of deep residual forecasting across a range of tasks [8], we build on recent work in this area, tailoring the approach to the task of forecasting blood glucose.

2.1 Problem Setting and Notation

We focus on the task of univariate time-series forecasting in which we aim to predict future values of a single variable, but assume

we have access to additional inputs. Let $X = [\mathbf{x}^{(1)}, ..., \mathbf{x}^{(d)}]$ represent a multivariate time series where d is the number of variables. For each variable i: $\mathbf{x}^{(i)} \in \mathbb{R}^T$ is a sequence of length T. For our problem, $\mathbf{x}^{(1)}$ corresponds to glucose measurements and $X' = [\mathbf{x}^{(2)}, ..., \mathbf{x}^{(d)}]$ represents other variables of interest (*e.g.*, bolus insulin). Given X, we aim to predict the next h glucose measurements $\mathbf{y} = x^{(1)}_{T+1}, x^{(1)}_{T+2}, ..., x^{(1)}_{T+h}$.

2.2 Proposed Approach

We build on a recent univariate time-series forecasting architecture, N-BEATS [8], which consists of a series of n blocks, each composed of a series of fully-connected layers. The i^{th} block takes as input some vector $\mathbf{x_i} \in \mathbb{R}^{\mathbf{T}}$, where for the first block, $\mathbf{x_1}$ is the original input. Each block's output $\Phi(\mathbf{x_i}) \in \mathbb{R}^{h'}$ produces a forecast $\mathbf{f_i} \in \mathbb{R}^h$ and a backcast $\mathbf{b_i} \in \mathbb{R}^T$. The backcast is subtracted from the current block's input before being input to the next block (*i.e.*, $\mathbf{x_{i+1}} = \mathbf{x_i} - \mathbf{b_i}$). The output of the network is the sum of forecasts across all blocks: $\hat{\mathbf{y}} = \sum_{i=1}^{n} \mathbf{f_i}$. The residual nature of the prediction means that each block learns only what the previous block could not. Thus, the model learns to gradually reconstruct the signal, while predicting components of the forecast. We build on this idea by identifying several opportunities for improvement. Our modification are as follows:

Accounting for Temporal Structure. We account for the temporal nature of the input sequence by using a bidirectional LSTM network [3] within each block, in lieu of the fully connected layers. Although using fully connected networks results in a faster training time, we hypothesized that an LSTM would be better able to capture block-level time-varying patterns by enforcing a sequential prior on the signal. The final LSTM hidden state is used as input to an output layer that produces a sequence of length $T + h$ that is then split into the backcast and forecast signal.

Including Additional Input Variables. We include auxiliary variables as input at each block. This provides important context when backcasting and forecasting the main signal of interest. However, rather than simply augmenting the dimensionality of the signal throughout, we backcast/forecast only the primary signal, *i.e.*, each block's input is $[\mathbf{x}^{(1)}_{i-1} - \mathbf{b}_{i-1}, X']$ (**Figure 1**). Backcasting/forecasting was not performed on additional variables due to their sparse nature. We hypothesized that learning to backcast/forecast abrupt carbohydrate and bolus inputs would increase the difficulty of the overall task and decrease performance on the main task.

Auxiliary Tasks and Supervision. In addition to training based on the loss of the final forecast prediction, $MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{h}\sum_{i=1}^{h}(y_i - \hat{y}_i)^2$, we include additional losses with the goal of improving the quality of intermediate representations. We add three auxiliary tasks to our model (shown in **Figure 1**):

- Per-block reconstruction loss ($\mathcal{L}_r$): We hypothesized that explicitly supervising the backcasts could improve their accuracy, and supervising at the block level could improve the intermediate reconstructions. To do this, we add an MSE loss, $MSE(\mathbf{x}^{(1)}_i, \mathbf{b}_i)$ after each block's output. Each block's loss is weighted proportional to its position in the network, encouraging the network to gradually construct a backcast. The final loss $\mathcal{L}_r$ is scaled by the sum of the weights of each block, and summed over n blocks:

$$\mathcal{L}_r = \frac{\sum_{i=1}^{n} i \cdot MSE(\mathbf{x}^{(1)}_i, \mathbf{b}_i)}{\sum_{i=1}^{n} i}$$

- Per-block forecast loss ($\mathcal{L}_f$): To encourage each block to contribute towards an accurate forecast, we calculate the loss on the running forecast after each block. We apply a similar per-block scaling term as in the backcast, but increase the weight using cubic values in order to emphasize the network's later predictions. Let $\mathbf{f'_i}$ denote the sum of all forecasts up to the i^{th} block, ie $\mathbf{f'_i} = \sum_{k=1}^{i} \mathbf{f_k}$. The total forecast loss L_f is:

$$\mathcal{L}_f = \frac{\sum_{i=1}^{n} i^3 \cdot MSE(\mathbf{f'_i}, \mathbf{y})}{\sum_{i=1}^{n} i^3}$$

- Per-block magnitude loss ($\mathcal{L}_m$): Intuitively, deep residual forecasting aims to incrementally learn parts of a signal. To ensure that each block contributes to the backcast, we penalize blocks by the inverse of their output size (measured using an L_1 norm) and scale this penalty by the inverse of their position in the stack, so earlier blocks are encouraged to output larger values. We apply this loss to only the backcast. We do not penalize low-magnitude forecasts because it is possible that a block could account for part of the input signal that is associated with a zero-value forecast. The complete magnitude loss is:

$$\mathcal{L}_m = \frac{\sum_{i=1}^{n} \frac{1}{i} \cdot \frac{1}{|\mathbf{b_i}|}}{\sum_{i=1}^{n} \frac{1}{i}}$$

Loss is calculated as a weighted sum of these three losses:

$$\mathcal{L} = \mathcal{L}_f + \beta\mathcal{L}_r + \gamma\mathcal{L}_m,$$

where $\beta, \gamma \in \mathbb{R}^+$are hyperparameters. Note that the final forecast is included as a part of $\mathcal{L}_f$.

2.3 Ensembling

We ensemble models that use different input lengths. As others have shown, this allows us to account for trends at different time scales [8]. We train a model for each of 6 different input lengths. We use T = 12, 18, 24, 30, 36, and 42 time steps. These values were selected as multiples of the shortest input length, $h = 6$. Shorter backcast windows (*i.e.*, $T < 12$) were not found to be beneficial. When ensembling the predictions, we output the median.

3 Experimental Set Up

Datasets. We evaluate the proposed approach using the 2020 OhioT1DM dataset [6]. This dataset consists of CGM, insulin pump, and other variables (*i.e.*, sleep data, activity levels) for six individuals. We explored using all variables in the dataset, but found only glucose (CGM and finger stick), bolus dose, carbohydrate input, and time of day to be helpful. Each individual has approximately 10,000 samples for training and 2,500 for testing, recorded at 5-minute intervals, where the test data temporally follow the training data. We use the first 80% of each individual's specified training set for training and the remaining 20% for early-stopping validation, holding out the test data. Beyond the 2020 OhioT1DM dataset, we had access to Tidepool data, a large repository of CGM and insulin pump data for approximately 100 participants with a total 15

million time steps [7]. We used these data and data from the 2018 OhioT1DM challenge [6] during pre-training and model selection.

Preprocessing. Bolus dose and carbohydrate input are less frequently recorded compared to CGM. Thus, to be used as input to the model, we align and resample these additional data to length T. In addition, we encode time using sine and cosine embeddings over 24-hour periods. Approximately 15% of all time-steps were missing for CGM. We replace missing values with a value of zero and include an additional variable that indicates missingness. In this way, the model can learn to handle missing values differently [5]. This results in a model input of $X \in \mathbb{R}^{T \times 7}$ (CGM, finger stick glucose, bolus values, carbohydrate inputs, sine and cosine of time, and missingness indicators for CGM values). Time steps with missing glucose values are ignored during all loss calculations and forecast evaluations.

Baseline Architecture Implementation. Our baseline is based on N-BEATS and uses hyperparameters similar to those reported by Oreshkin *et al.*: 10 blocks with 4 layers each, with 512 hidden units output by each layer, with a ReLU activation function between each layer [8]. Though Oreshkin *et al.* originally used 30 blocks, we found 10 blocks worked better and thus report performance against this stronger baseline.

Proposed Architecture Implementation. To learn the model parameters of our proposed approach, we use Adam [4] with a learning rate of 0.0002 (selected to maximize learning speed while maintaining reasonable convergence behavior), and a batchsize of 512. We run the optimization algorithm for up to 300 epochs, or until validation performance does not improve for 20 iterations.

To reduce the chance of overfitting to the 2020 data, we tune all hyperparameters, excluding γ and β, on the six subjects from the previous competition. We performed a grid search to select the number of blocks (range: 5-10, best: 7) and number of hidden LSTM units (range: 50-300, best: 300), optimizing for rMSE of the final time step of the prediction on the held out data. Including more hidden units could perhaps have further improved performance, but we were limited by memory constraints.

During tuning, we measured the added value of including each auxiliary variable (*e.g.*, sleep data), including only those variables that improved performance for the final evaluation. We tuned the block-weighting terms for our auxiliary loss functions, considering inverse, constant, and linear relationships for $\mathcal{L}_r$, and powers of 2 and 3 for $\mathcal{L}_f$. We tuned the auxiliary loss function weights β, and γ using the validation data for the new subjects. We performed a grid search over $[.1, .15, .25, .3, .5, 1, 2, 4]$ for β and $10e5$ times those values for γ, selecting $\beta = .3$ and $\gamma = 10e4$. Grid search analyses were tuned with a subset of backcast lengths ($3f$ and $6f$) across all patients, for efficiency.

For both the 30 and 60 minute prediction horizons, we train using loss for the full window (*i.e.*, $h = 6$ for the 30 minute analyses and $h = 12$ for the hour prediction), but report results using the last time step only (as dictated by the competition).

Pre-training. As mentioned above, we pre-train on the 2018 OhioT1DM and Tidepool data. Pooling these datasets together, we train a single model. The weights of this model are used to initialize the weights of a 'global' model trained on the 2020 OhioT1DM training set. This 'global' model is then fine-tuned to each participant from the 2020 OhioT1DM dataset, resulting in six participant-specific models. Baseline analyses do not use the large pooled dataset for pre-training, but start from a single global model (trained from random initialization) and then fine-tune.

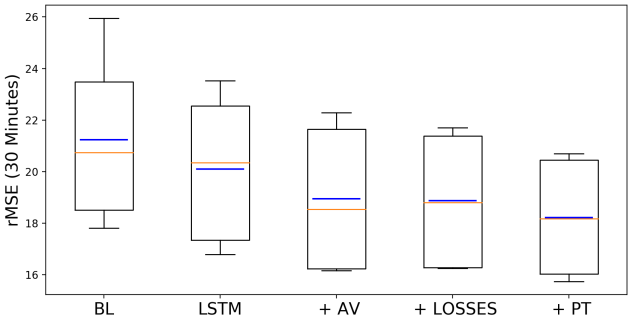

Figure 2. Cumulative effect of each proposed modification on 30-minute horizon forecast performance. Each modification includes the modifications to the left as well. BL = baseline, LSTM = updated block structure, +AV = added variables, +LOSSES = additional task losses added, +PT = used large pooled dataset for pre-training. Each box plot shows the distribution across the six subjects, with blue and orange lines indicating the mean and median across subjects, respectively.

Code. All of our experiments were implemented in Python/PyTorch [10]. The final initialization models and all code used in our analyses is publicly available [2].

3.1 Evaluation

We report results for models trained to predict 12 time steps (one hour horizon) and six time steps (30 minute horizon), reporting outcomes on only the final time step. We evaluate using the square root of the mean squared error (rMSE), and mean absolute error (MAE), defined as $MAE(\mathbf{y}, \mathbf{y'}) = \frac{|\mathbf{y}-\mathbf{y'}|}{n}$ for n samples. Metrics are applied directly to the raw CGM values with no preprocessing.

To examine the effect of each architecture modification, we perform an ablation study where we add each modification to the architecture incrementally (*i.e.*, first RNN blocks only, then RNN blocks with added variables, then RNN blocks with added variables and additional loss terms, then the full architecture with pre-training). We also experiment with each modification made in isolation.

Finally, beyond the evaluation dictated by the competition, we also calculate outcomes for the most crucial forecasting windows: those that represent the beginning of a hyper- or hypoglycemic event. These correspond to predictions for which the most recent time step is in the euglycemic range (blood glucose from 70-180 mg/dL), but the participant becomes hypoglycemic (< 70 mg/dL) or hyperglycemic (> 180 mg/dL) within the prediction horizon. We also examine the distribution of predictions vs. actual measurements in terms of the corresponding would-be treatment recommendations with a Clarke Error Grid Analysis [2].

4 Results and Discussion

Overall, the proposed approach leads to average rMSEs of 18.2 and 31.7 on the 30 and 60 minute horizons, respectively, and average

[2] https://gitlab.eecs.umich.edu/mld3/deep-residual-time-series-forecasting

MAEs of 12.8 and 23.6 for those horizons (**Table 1**). These values are comparable to the results achieved in the 2018 challenge (*i.e.*, 18.9 to 21.7 for 30 minute rMSE) [1, 13].

Based on our ablation study, each of our modifications improves performance over baseline, although to varying degrees **(Figure 2)**. For the 30 minute horizon, mean rMSE across participants is 21.2 for the baseline model (BL). Adding the RNN block structure (bidirectional LSTM) and additional variables (+AV) improves performance to 20.1 and then 18.94, respectively. Adding the additional loss functions (+LOSSES) results in a slightly improved mean performance of 18.87, and pre-training on the Tidepool and 2018 OhioT1DM datasets (+PT) further improves performance to 18.2. We observed similar trends for the 60 minute horizon, and for MAE.

Table 1. Participant-level results for 30 and 60 minute horizons on the held-out test data from the 2020 competition.

	Prediction Horizon			
	30 Minutes		60 Minutes	
Participant ID	rMSE	MAE	rMSE	MAE
567	20.70	13.78	35.99	25.84
544	15.97	11.20	26.01	19.01
552	15.74	11.51	29.17	22.78
596	16.24	11.26	27.11	19.73
540	20.10	14.77	38.43	29.51
584	20.57	14.47	33.26	24.72
Mean	18.22	12.83	31.66	23.60

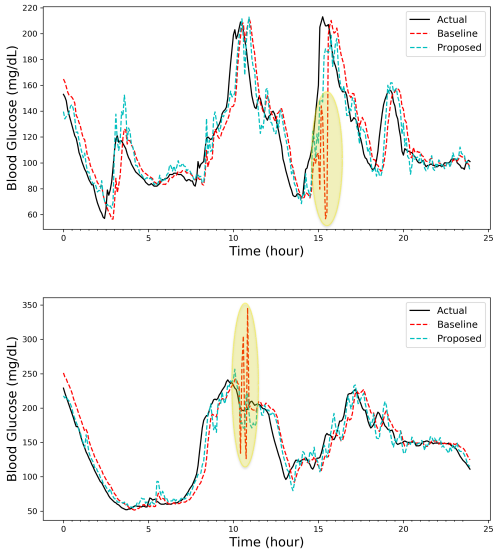

Figure 3. A full day of 30-minute predictions on two participants, comparing the baseline and our proposed approach. Qualitatively, our modifications result in fewer sudden extreme failures. In general, our predictions appear more accurate, compared to the baseline. (a) Best performance individual - 552 (rMSE = 15.7). (b) Worst performance individual - 567 (rMSE = 20.7).

While all of our modifications reduce rMSE, one of the largest improvements in performance comes from replacing the fully connected layers of N-BEATS with recurrent layers (improves performance from 21.2 to 20.1). These recurrent layers directly model each time step as a function of the previous, allowing for more accurate temporal representations. Compared to the baseline, the proposed approach leads to more faithful predictions and fewer extreme failures **(Figure 3**). Even without pre-training, the modified architecture eliminates the extreme failures observed in the baseline case (highlighted in **Figure 3**).

Additional variables are significantly more impactful when added to the RNN-based block structure (a 30 minute rMSE improvement from 20.1 to 18.9), when compared to adding them in isolation (an improvement from 21.2 to 21.0). Despite the poor performance when used in the original baseline architecture, we chose to include these variables in our updated RNN-based model. The RNN block structure has multiple input channels, which allows for the direct use of the variables. In contrast, the fully connected layers require a flat input, which removes much of the timing information. This illustrates the importance of the ordering of decision making in model architecture development. Although such decisions are easy when an exhaustive search of combinations is performed, this is not always feasible, so these decisions often must be made sequentially. If we had selected additional variables based on the performance of the baseline model alone, we may not have opted to use them, and would have lost the large benefit that they provide when used in tandem with the RNN block structure.

To test our hypothesis regarding the efficacy of backcasting on blood glucose values only (and not additional variables), we ran a post-hoc analysis with a model using residual backcasting for all input variables. As expected, we found that this decreased performance (30 minute rMSE = 19.2 vs. 18.9).

In contrast to the other modifications, the additional loss terms offer only slight improvements over the baseline. During model selection, this modification offered substantially more improvement on the OhioT1DM 2018 dataset (decreasing rMSE of the RNN + added variables model from 19.6 to 18.9). Yet even with additional tuning, this modification is only minimally helpful on the 2020 dataset, suggesting that the method may not be universally beneficial. In an ablation analysis, a model using only ($\mathcal{L}_f$) performs best (rMSE = 18.82) on the 2020 data. However, in contrast, the added variables seem to help more for the 2020 data than for the 2018 data. This suggests that perhaps the losses serve a regularizing effect when insufficient information is present in the additional variables, as we observed in the 2018 validation data.

Pre-training improves the performance of both the modified and baseline architectures, although it offers the most improvement for the baseline (improving performance from 21.2 to 19.8 when applied alone). This suggests that pre-training becomes less crucial once other improvements are made. Running an analysis using our proposed architecture and no pretraining over 4 random seeds results in similar performance across seeds (30-minute rMSEs were 18.87, 18.82, 18.95, and 18.89), indicating fairly robust performance over different random initializations. Further, pre-training improves performance significantly over all random initialization, demonstrating the benefit of pre-training beyond chance.

In our event-specific evaluation (includes only prediction windows where a hyperglycemic or hypoglycemic event occur), the mean rMSE is significantly higher compared to the holistic evaluation (for our proposed approach 30 minute horizon: 27.9 vs. 18.2). This is expected, since such events are difficult to anticipate. Specifically, the proposed approach achieves a higher rMSE for hyperglycemic events (30.5) than hypoglycemic (23.8). Interestingly, while our model performs better than the baseline for all events on average (our model: 27.9 vs baseline: 31.0) and for hyperglycemic events (30.5 vs 34.2), the two models perform comparably in the hypoglycemic range (23.8 vs 23.6). This could be due to the relative rarity of hypoglycemic events (for example, there were 322 in the test data, vs 868 hyper-

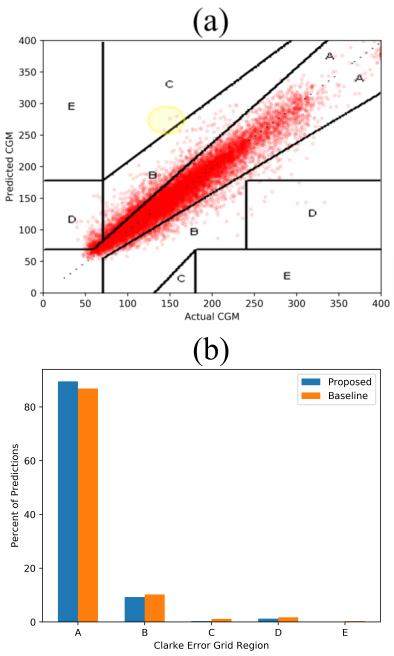

Figure 4. (a) A Clarke Error Grid depicting which treatment options would be recommended based on our proposed approach's predictions vs what treatments should be given based on the actual value. 99% of predictions fall within regions A and B, which indicate appropriate treatment recommendations. (b) Proportion of predictions in each region of the grid for our proposed approach and the baseline model.

glycemic events), or it could be a reflection of the MSE loss function over-emphasizing large values, which is unaccounted for in our approach.

In our Clarke Error Grid Analysis [2], we find that 99% of predictions fall in regions A and B (regions that would not lead to inappropriate treatment; *i.e.* an unnecessary bolus or rescue carbohydrates), with 90% in region A (predictions within 20% of the actual value) and 9% in region B (predictions that are more than 20% from the actual value but that would not lead to inappropriate treatment), indicating generally strong performance **(Figure 4)**. The proportion of points in regions A and B is slightly lower (97%) for our baseline model. Only 3 points (0.02%) fall into region C (points that would lead to unnecessary treatment, specifically predicting high CGM when it would actually be lower, which could lead to an inappropriate bolus and hypoglycemia), and 1% fall into region D (points that miss hyperglycemia or hypoglycemia). Reassuringly, no points fall into region E (regions that would treat hyperglycemia as hypoglycemia or vice versa).

5 Conclusion

We find deep residual forecasting to be effective when applied to the task of predicting blood glucose values. Augmenting a previously proposed architecture with RNNs in place of fully connected stacks, additional variables, and self-supervising loss functions all lead to improvements when applied to the task of blood glucose forecasting. Beyond blood glucose forecasting, we hypothesize that many of the proposed changes could be beneficial when applied to other forecasting tasks.

ACKNOWLEDGEMENTS

This work was supported by JDRF (award no. 1-SRA-2019-824-S-B). In particular, this award provided access to the Tidepool dataset that was used in pre-training the models. The views and conclusions in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of JDRF.

REFERENCES

[1] J. Chen, K. Li, P. Herrero, T. Zhu, and P. Georgiou, 'Dilated recurrent neural network for short-time prediction of glucose concentration.', *KDH*, (2018).

[2] W.L. Clarke, D. Cox, L.A. Gonder-Frederick, W. Carter, and S.L. Pohl, 'Evaluating clinical accuracy of systems for self-monitoring of blood glucose.', *Diabetes Care*, **10**, (1987).

[3] F.A. Gers, F. Schmidhuber, and F. Cummins, 'Learning to forget: continual prediction with LSTM.', *9th International Conference on Artificial Neural Networks*, 850–855, (1999).

[4] D.P. Kingma and J. Ba, 'Adam: A method for stochastic optimization.', *International Conference for Learning Representations*, (2014).

[5] Z.C. Lipton, D.C. Kale, and R. Wetzel, 'Modeling missing data in clinical time series with rnns.', *Proceedings of Machine Learning for Healthcare*, (2016).

[6] C. Marling and R. Bunescu, 'The OhioT1DM dataset for blood glucose level prediction: Update 2020.', (2020).

[7] A. Neinstein, J. Wong, H. Look, B. Arbiter, K. Quirk, S. McCanne, Y. Sun, M. Blum, and S. Adi, 'A case study in open source innovation: developing the tidepool platform for interoperability in type 1 diabetes management.', *Journal of the American Medical Informatics Association*, **23**, (2016).

[8] B.N Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, 'N-BEATS: Neural basis expansion analysis for interpretable time series forecasting', *ICLR*, (2020).

[9] S. Oviedo, J. Vehí, R. Calm, and J. Armengol, 'A review of personalized blood glucose prediction strategies for T1DM patients.', *Numerical Methods in Biomedical Engineering*, (2016).

[10] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, 'Automatic differentiation in PyTorch.', *NeurIPS*, (2017).

[11] J. Reifman, S. Rajaraman, A. Gribok, and W.K. Ward, 'Predictive monitoring for improved management of glucose levels.', *J Diabetes Sci Technol*, (2007).

[12] R.J. Williams and G.E. Hinton, 'Learning representations by back-propagating errors.', *Nature*, **323**, 533–536, (1986).

[13] T. Zhu, K. Li, P. Herrero, J. Chen, and P. Georgiou, 'Predictive monitoring for improved management of glucose levels.', *KDH*, (2018).