Neural Ordinary Differential Equation based Recurrent Neural Network Model

Mansura Habiba
Cloud Solutions Architect
IBM in Ireland
Dublin, Ireland
0000-0001-9051-1370

Barak A. Pearlmutter
Dept of Computer Science & Hamilton Institute
Maynooth University
Maynooth, Ireland
0000-0003-0521-4553

***Abstract*—Neural differential equations are a promising new member in the neural network family. They show the potential of differential equations for time-series data analysis. In this paper, the strength of the ordinary differential equation (ODE) is explored with a new extension. The main goal of this work is to answer the following questions: (i) can ODE be used to redefine the existing neural network model? (ii) can Neural ODEs solve the irregular sampling rate challenge of existing neural network models for a continuous time series, i.e., length and dynamic nature, (iii) how to reduce the training and evaluation time of existing Neural ODE systems? This work leverages the mathematical foundation of ODEs to redesign traditional RNNs such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). The main contribution of this paper is to illustrate the design of two new ODE-based RNN models (GRU-ODE model and LSTM-ODE) which can compute the hidden state and cell state at any point of time using an ODE solver. These models reduce the computation overhead of hidden state and cell state by a vast amount. The performance evaluation of these two new models for learning continuous time series with irregular sampling rate is then demonstrated. Experiments show that these new ODE based RNN models require less training time than Latent ODEs and conventional Neural ODEs. They can achieve higher accuracy quickly, and the design of the neural network is more straightforward than the previous neural ODE systems.**

***Index Terms*—Recurrent Neural Network, Time series, Ordinary Differential equation**

I. Introduction

Continuous time series is a fundamental data structure for different research areas, i.e., sensor-based IoT, energy consumption, weather, music generation etc. It also has some dynamic characteristics such as high sampling frequency, inconsistent sampling rate, multi-variate, large in length, dynamic and uncertain. Neural network models need to support these characteristics to process continuous time series. Among all the available neural network models, RNN has become the pioneer in terms of time series modelling and processing due to its gating unit and memory storing capacity. The majority of existing RNNs processes continuous time series as a discrete-time sequence with a fixed sampling rate and fixed sampling frequency [8]. This characteristics of RNN makes real-time processing difficult. It also imposes high computation load and memory usage. In addition, due to the dependency on the computation of previous states, such RNN models are prone to some vulnerabilities such as if the time gap, between two consecutive observations, is too big, it can affect the efficiency of the model adversely. Therefore, such RNNs are only suitable for time series of moderate length with fixed sampling rate, few missing values, and short time intervals between observations.

Chen et al. [5], Rubanova et al. [10] have demonstrated the strength of Neural ODEs [2, 9] for processing time series using deep learning. Chen et al. [5] proposes to compute iterative updates of RNN as Euler discretization of continuous transformation [3]. Based on this idea, the continuous hidden dynamics of a neural network can be parametrized by ordinary differential equation (1). Using the initial value problem to compute $h(t)$ at any time t from the initial hidden units $h(0)$, this continuous-depth model leverages ODE solver (f) to define and evaluate models. Latent-ODE [5] uses an ODE as an encoder whereas ODE-RNN [10] uses an RNN as an encoder.

$$\frac{d}{dt}h(t) = f(h_t, t, \delta_t) \tag{1}$$

ODE-RNN [10] uses encoder and decoder to compute the hidden dynamics of the neural network. The input is fed to an encoder and a different equation solver is used to calculate the hidden state and later the decoder generates the output. This is an auto-regressive model, similar to Chen et al. [5], ODE-RNN can also compute the hidden dynamics at any time t. The dynamics between observations is learned rather than pre-defined, which makes ODE-RNN a suitable model for irregular and sparse data. As the computation is not dependent on the availability of data points, both ODE-RNN and Latent-ODE take more time for training and evaluation. The time depends on the length of the time series and the complexity of data. The focus of this paper is to consider the neural network as a function of continuous time; it, therefore, does not use any additional encoder or decoder.

In case of time series precessing, RNN based neural network model are already ahead of others. However, different RNN models use a discrete sequence of observations with fixed sampling rate for modelling continuous time series. Therefore, often it is difficult to compute a time series in real time. The main goal of this work is to leverage the continuousness of ODE-NET to design Neural Ordinary Differential Equation

978-1-7281-9418-9/20/$31.00 ©2020 IEEE

based RNN which can reduce the overhead of additional encoding and decoding for simple time series data processing. These proposed models can overcome the limitation of fixed time steps with marginal budget cost in shorter time duration and accurate real time computation.

This work proposed to design the architecture of different RNN, e.g., GRU [6] or LSTM [7] using an ordinary differential equation, which is referred as ODE-GRU and ODE-LSTM throughout this paper. These proposed models can train the gradients itself within marginal error using the ordinary differential equation solver. All parameters are learned during training. The cell state and dynamics of hidden state are computed using a black-box ordinary differential equation solver. The update and reset functions of existing RNN models are redefined to leverage ODE solver. These proposed models can process continuous time series in real-time with comparatively less computation and memory usage. These models provide a generative process over time series similar to Rubanova et al. [10]. We compare proposed models to several RNN variants and find that these new models can perform better when the data is sparse.

II. Methodology

Fig. 1 shows the architecture of the standard GRU model. In a standard RNN cell either GRU or LSTM, the input, output and hidden state (h_t) are controlled by gating units, i.e reset gate (r_t) and update gate (z_t) as shown by (3a) and (3b) respectively. Each cell in standard RNN uses the hidden state of precious cell (h_{t-1}) as input along with raw data input. In

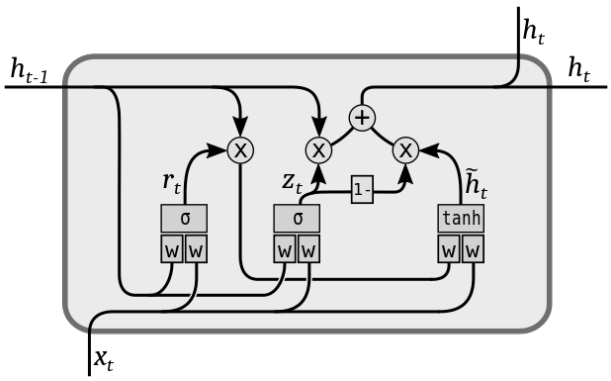

Fig. 1. Architecture of GRU model

this paper, the existing RNN cell is redesigned as a function of ordinary differential equations. Two different RNN models are proposed using the similar cell structure of GRU and LSTM. In this new RNN model, the hidden dynamics is computed using the derivative of the hidden state between observations. Fig. 2 shows that the proposed model progress over continuous time, where h_0 is the initial hidden state passed through an ordinary differential equation (ODE) which uses update function for solving ODE similar to standard GRU or LSTM cell (f). For two different gating units of RNN, two different ODE- based RNN models (i) ODE-GRU and (ii) ODE-LSTM are proposed in this paper.

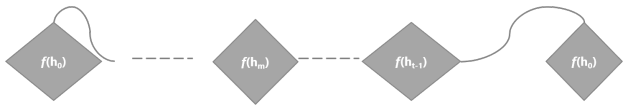

Fig. 2. The architecture of ODE-GRU model

These two proposed models leverage the capabilities of ordinary differential equation solvers. The final cell state y_t for the model can be computed using (2). *odeRNNCell* refers to an ODE based RNN cell, either GRU-ODE or LSTM-ODE.

$$y_t = \text{ODESOLVER}(\textit{odeRNNCell}, y_0, t) \tag{2a}$$

$$y_t = \text{ODESOLVER}(\textit{odeRNNCell}, \textit{tuple}(y_0, h_0), t) \tag{2b}$$

ODESOLVER can be any kind of ode solver, e.g., a Euler or adjoint or implicit method. Models proposed in this paper use a neural ordinary function *odeRNNCell* to compute change or derivative of the hidden dynamics at any time t. *odeRNNCell* is usually an initial value problem function as shown in (2a) which requires the initial observation y_0 and hidden state h_0 at time $t = t_0$. In the case of (2a), the cell state at time t is computed based on initial time state, and the hidden dynamics is considered as gradient dynamics and computed by the gradients update. *odeRNNCell*, can be either *odeGRU*, as shown in Algorithm 1 or *odeLSTM*, as shown in Algorithm 4. In other cases, as in (2b), the cell state or output at any time is computed based on the initial cell state (y_0) and hidden dynamics (h_0) at time t_0. In this implementation, the proposed neural network is initialized with an initial hidden state (h_0), and only the initial observation value is passed as the input for the model.

A. Proposed ODE-GRU model

The proposed ODE-GRU Model uses the traditional equations (3a), (3b) , (3d) to compute different interemidiate states. In addition, an Ordinary differential solver compute the chnages in state in order to learn the evolution of the state of any dynamical system.

Algorithm 1 ODE-GRU ode solver function

Input:: initial hidden and cell state,time series and weight as well as bias parameters

procedure ODEGRU($states, t, parameters$)
  $x \leftarrow states[0]$
  $h \leftarrow states[1]$
  $r_t, z_t, \tilde{h}_t \leftarrow updateGRUFunction(x, h, parameters)$
  $h_t \leftarrow \tilde{h}_t(1 - z_t)$
  $o_t \leftarrow \sigma(W_o h_t + b_o)$
  $\frac{dh_t}{dt} \leftarrow \frac{d\tilde{h}_t}{dt}(1 - z_t)$
  $\frac{do_t}{dt} \leftarrow \frac{dh_t}{dt}$
  return $\frac{do_t}{dt}, \frac{dh_t}{dt}$

Here *states* are the initial hidden state along with the observation value at start time, $t = t_0$, t is the time

series, *parameters* is a tuple of all weight parameters $(W_r, U_r, W_r, U_z, W_h, U_h)$ and bias parameters (b_r, b_z, b_h). *updateGRUFunction* takes the input and compute the traditional GRU update functions:

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right) \tag{3a}$$
$$z_t = \sigma\left(W_z x_t + U_z h_{t-1} + b_r\right) \tag{3b}$$
$$\tilde{h}_t = \tanh\left(W_r x_t + U_r r_\tau + b_r\right) \tag{3c}$$
$$o_t = \sigma\left(W_o h_t + b_o\right) \tag{3d}$$

ODE-GRU pass the changes to the ODE solver and uses only the change in hidden state as (4a). The proposed RNN models do not carry hidden state of precious cell (h_{t-1}) as like standard RNN models. Only the derivative of the hidden state over time$(fracd\left(h_t\right)dt)$ is progressed as a function of time .

$$\frac{dh_t}{dt} = \frac{d\tilde{h}_t}{dt}(1 - z_t) \tag{4a}$$

Algorithm 2 describes the complete flow for the training the model using the ODE-GRU model we propose. Algorith 3

Algorithm 2 Train the ODE-GRU model

Input:: Initial condition (y_0 and h_0), continuous time series(t) and Initial parameter

procedure TRAIN(y_0, h_0, target, t, initalParams)
  $input \leftarrow tuple(y_0, h_0)$
  $trainedParams \leftarrow$ optimizer (gradient(GRADF), $initalParams$, $iterations$, callback = GRADFUNCTION.backward())
  return $trainedParams$

describes the gradient function $GRADF$ which uses ***odeGRU*** from algorithm 1.

Algorithm 3 Gradient function for ODE-GRU model

Input:: ($input, t$ and $target$), continuous time series(t) and Initial parameter

procedure GRADF($input, t, initalParams, target$)
  $prediction \leftarrow$ ODESOLVE($odeGRU$, $input$, t, $initalParams$)
  $loss \leftarrow lossFunction(prediction, target)$
  return $loss$

Here any *optimizer* and *lossFunction* can be used based on the complexity of the dataset. *GRADF* is the gradient function uses by optimizer.

B. Proposed ODE-LSTM Model

Proposed ODE-LSTM model predicts the output (y_t) at any time t using *odeLSTM* ODE solver using the initial value of the observation (y_0) at time $t = t_0$. Therefore, instead of computing each state based on its previous state individually, the predicted output is generated as a function of time. (5) describes ODE Solvers, those have been used in this work:

$$y_t = \text{ODESOLVER}(\textit{odeLSTM}, \textit{tuple}(y_0, h_0, c_0), t) \tag{5}$$

Here, *odeLSTM* is the function that constructs the dynamics of hidden as well as cell state and call the ODE solver to compute all the gradients at once. The algorithm of *odeLSTM* is discussed in 4. This model uses the same input parameter as ODE-GRU, the initial observation value, (y_0) and the initial hidden state of the network h_0. Besides, this model also uses the initial cell state (c_0) as an input parameter. This proposed ODE solver function construct the network as an initial value problem function, therefore, providing the initial condition is important.

Algorithm 4 ODE-LSTM update function

Input:: initial hidden and cell state,time series and weight as well as bias parameters

procedure ODELSTM($states, t, param$)
  $x \leftarrow states[0]$
  $h \leftarrow states[1]$
  $c \leftarrow states[2]$
  $f_t, i_t, c_t, h_t, o_t \leftarrow updateLSTMGate(x, h, c, param)$
  $cell \leftarrow \frac{df_t}{dt} * c + \frac{di_t}{dt} * c_t$
  $hidden \leftarrow \frac{do_t}{dt} * \frac{dcell}{dt}$
  return $\frac{do_t}{dt}, hidden, cell$

Here *states* contains the initial hidden state, initial cell state and the observation value at start time, $t = t_0$, t is the time series, *param* is a tuple of all weight parameters $(W_r, U_r, W_r, U_z, W_h, U_h)$ and bias parameters (b_r, b_z, b_h). *updateLSTMGate* takes the input and compute the traditional LSTM update functions:

$$i_t = \sigma_i\left(x_t W_{xi} + h_{t-1} W_{hi} + w_{ci} \odot c_{t-1} + b_i\right) \tag{6a}$$
$$f_t = \sigma_f\left(x_t W_{xf} + h_{t-1} W_{hf} + w_{cf} \odot c_{t-1} + b_f\right) \tag{6b}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \sigma_c\left(x_t W_{xc} + h_{t-1} W_{hc} + b_c\right) \tag{6c}$$
$$o_t = \sigma_o\left(x_t W_{xo} + h_{t-1} W_{ho} + w_{co} \odot c_t + b_o\right) \tag{6d}$$
$$h_t = o_t \odot \sigma_h\left(c_t\right) \tag{6e}$$

Different update gate function such as input (i_t), forget(f_t) and output(o_t) use the typical sigmoidal nonlinearities $\sigma_i, \sigma_f, \sigma_o$ and cell (c_t) and hidden (h_t) activation functions uses tanh nonlinearities. σ_c and σ_h along with weight parameters(W_{hi}, W_{hf}, W_{ho}, W_{xi}, W_{xf}, W_{xo}), connect the different inputs and gates with the memory cells and outputs, as well as biases (b_i, b_f, and b_o). Both biases and connection weights (w_{ci}, w_{cf}, and w_{co}) are optional for further influence the update of input, forget, and output gates.

ODE-LSTM pass the changes in different gates such as input, forget and hidden gate to the ODESOLVER and uses only the change in smaller fraction to compute the final output, hidden and cell state using (7b), (7c) and (7a):

$$c_t = \frac{df_t}{dt} * c_0 + \frac{di_t}{dt} * c_t \tag{7a}$$

$$\tilde{o}_t = \frac{do_t}{dt} \tag{7b}$$

$$h_t = \tilde{o}_t * \frac{dc_t}{dt} \tag{7c}$$

Algorithm 5 describes the complete flow for the training the model using the proposed ODE-LSTM model. *odeLSTM* is computing all gradients at once for the model. The parameters are learned during training using the loss function. The change in hidden data dynamics is computed and learned during the training session of the proposed model. Similar to algorithm 2, the optimizer uses gradient function (*GRADF*)

Algorithm 5 Train the ODE-LSTM model

Input:: Initial conditions (y_0, h_0), input(t), $initialParams$

procedure TRAIN(y_0, h_0, c_0, $target$, t, $initalParams$)
    input $\leftarrow (y_0, h_0)$
    $trainedParam \leftarrow$ optimizer(gradient($GRADF$), $initalParams$, $iterations$, callback = GRADFUNCTION.backward())
    return $trainedParameters$

Similar to ODE-GRU model, ODE-LSTM also uses a gradient function $GRADF$ described in algorithm II-B, where the ordinary differential solver (ODESOLVER) uses *odeLSTM* from alforithm 4 to optimize the output.

procedure GRADF($input, t, initalParams, target$)
    $prediction \leftarrow ODESOLVE(odeLSTM, input, t, initalParams)$
    $loss \leftarrow lossFunction(prediction, target)$
    return $loss$

III. Results

A number of experiments are done to understand the dynamic characteristics of the dataset.These experiments also show how the complexity of the data influences the proposed models.

A. Curve datasets

In order to understand the learning mechanism of the proposed models, two different curve time series, e.g., (i) eight-curve with low complexity, (ii) tri-knot with medium complexity are chosen. Each of them possesses different complexity. This experiment helps to understand the impact of time series flow as well as the complexity and sparseness of time series dataset on the learning of the proposed model. The learning vector for tri-knot curve also demonstrates that the proposed model can learn itself comparatively faster than traditional LSTM and other ODE based RNN [5, 10]. Fig. 3 shows that the proposed ODE-GRU model needs only 20 iterations to identify learning vector and the curve dynamics.

Fig. 4 and shows the learning vector of 1st and 20th iteration for eight curves. This demonstrates that the learning growth is very fast, even for only 50 hidden dimensions. This proposed model can play a significant role to identify the types of dynamic of the corresponding time series data. These experiments show that with few parameters, these proposed models can learn time series data faster than traditional RNN cells.

For further comparative performance evaluation, two different datasets (a Toy dataset and a Human activity dataset) are used.

B. Toy dataset

A simple time series with variable sampling rate or frequency is used to learn the dynamics of the proposed model by mimicking a spiral ODE. The spiral ODE used in this test consists of 1000 spiral, and each spiral has 100 irregularly sampled time points. In [4], a simple Feed Forward Neural network (FFN) is used for ODE solver as shown in (2a).

Algorithm 6 Forward fucntion in [4]

Input: t, y

procedure FORWARD(t, y)
    return $net(y**3)$

$$pred_y = odeint(FFN, batch_{y_0}, batch_t) \tag{8}$$

Fig. 5 shows the result after the 25 iterations of the implementation of Chen et al. [5] (left) and ODE-GRU (right). This experiment demonstrate that the proposed ODE-GRU can reduce the training time significantly, which is one of the limitations of existing work [5].

C. Human Activity dataset

For this experiment and comparative performance evaluation of the proposed model and Rubanova et al. [10], the human activity recognition dataset [1] is used. This dataset consists of 6 different activities along with 12 feature. After normalization, this dataset can be transferred to a time series of 6554 sequence at 211 time points. The configuration of Latent-ODE [10] for this comparative analysis is shown at Fig. 6a.

There are three different neural networks are used for Latent-ODE, i.e., encoder, differential equation solver and decoder. On the other hand, the proposed neural network only requires a single neural network in order to compute the derivative of the hidden dynamics, as shown in Fig. 6b. Proposed models need one a single neural network which is used by a simple differential equation solver function as shown in Algorithm 1. The proposed models are very simple in structure in comparison to other neural ODE, therefore, the training time is less than Rubanova et al. [10] and Chen et al. [5]. We have also compare our proposed models against standard GRU and LSTM. One important characteristics of the proposed model is the difference in loss between consecutive iterations of the training process is comparatively lower than

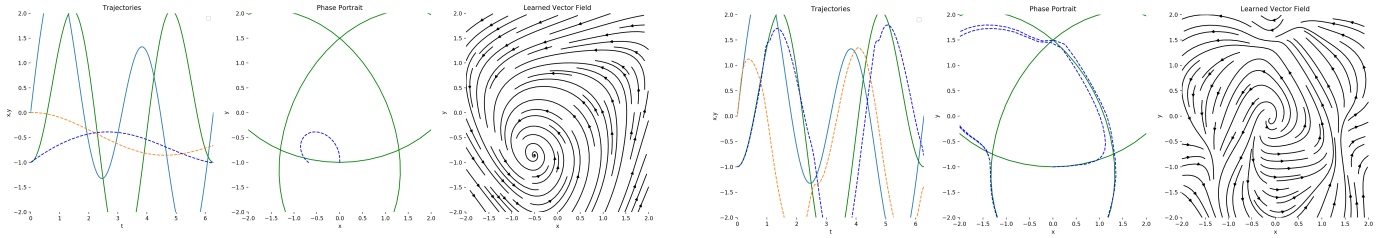

Fig. 3. Learning Tri-Knot curve function at iteration 1(Left) and 20(Right)

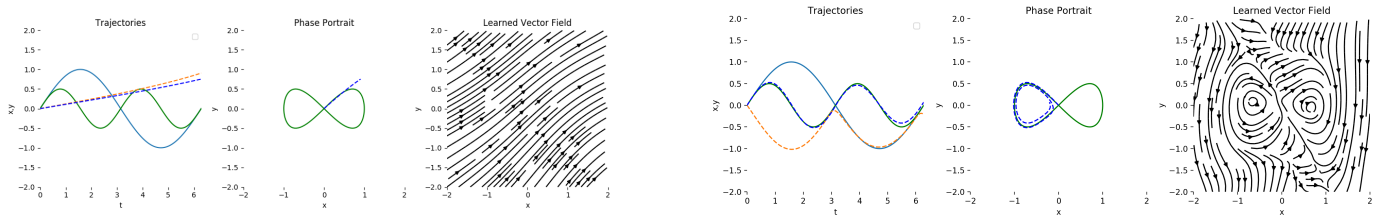

Fig. 4. Learning eight curve function at iteration 1 (Left) and 20(Right)

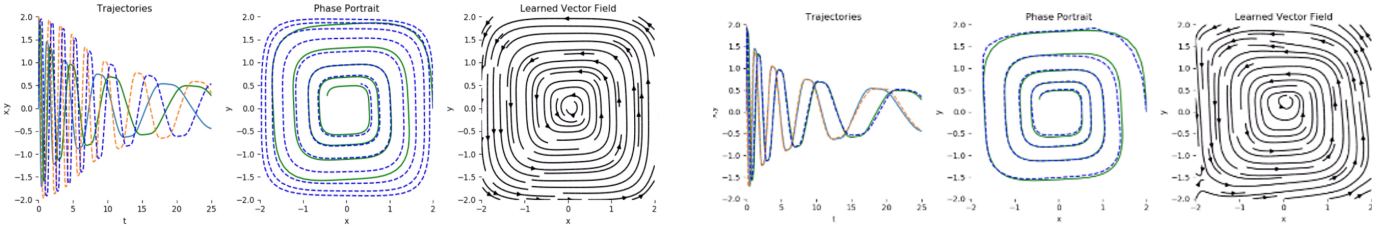

Fig. 5. Result of ODE neural network (left) and the proposed ODE-GRU (right) after 25 iterations of training.

```
model = LatentODE(
    input_dim = gen_data_dim, // the dimension of input vector
    latent_dim = latent_dim, // the dimension of hidden vector
    rnn_encoder_z0 = encoder_z0, // RNN as an ecoder
    rnn_decoder = decoder, // RNN as decoder
    diffeq_solver_nn = diffeq_solver, // Neural network to solve differential equation
    z0 = z0_prior, // Initial hidden state
    device = device,
    obsrv_std = obsrv_std,
    use_poisson_proc = True, // using poisson
    use_binary_classif = False,
    linear_classifier = True, // using linear classifier
    numer_of_labels = n_labels // number of labels
    ).to(device)
```

(a) Latent-ODE

```
model = RNNODE(
    input_dim = gen_data_dim, // the dimension of input vector
    latent_dim = latent_dim, // the dimension of hidden vector
    ode_net = ode_gru_func,
    z0_prior = z0_prior,
    device = device,
    obsrv_std = obsrv_std,
    use_poisson_proc = True, // using poisson
    use_binary_classif = False,
    linear_classifier = True, // using linear classifier
    numer_of_labels = n_labels // number of labels
    ).to(device)
```

(b) Proposed ODE-GRU/ODE-LSTM

Fig. 6. Configuration for neural network used in Human activity experiment

other neural network models. The reason is it only computes the derivative of the hidden dynamic $\frac{d}{dt}h_t$ at any time (t), therefore, the proposed models progress through iterations with small changes in loss value, as shown in Fig. 7.

The significant contributions of these proposed models are as following

1) *Simple architecture:* The architecture design of the proposed models is straightforward, it uses the update functions of standard RNN models and ODE differential equation solver to design RNN models as a function of the continuous time. As RNN models are already pioneering in terms of time series modelling, these proposed models leverage the strength of RNN and neural ODE in a single neural network.
2) *Data dynamics:* Due to continuous and straightforward characteristics, these models are very suitable for understanding the data dynamics. These models can compare the dynamics of different dynamical system to understand the data. This feature can significantly contribute to design data-driven system. These models can also participate in explainable machine learning.
3) *Faster Training:* As these proposed models learn the change in hidden dynamics of the data along with the gating unit of RNN, it takes comparatively less time for training and evaluation without compromising the accuracy of the result.
4) *Fewer parameters:* In comparison to standard RNN cells and different neural ODEs, even fewer weight, as well as bias parameters, can achieve the result even faster for proposed models.

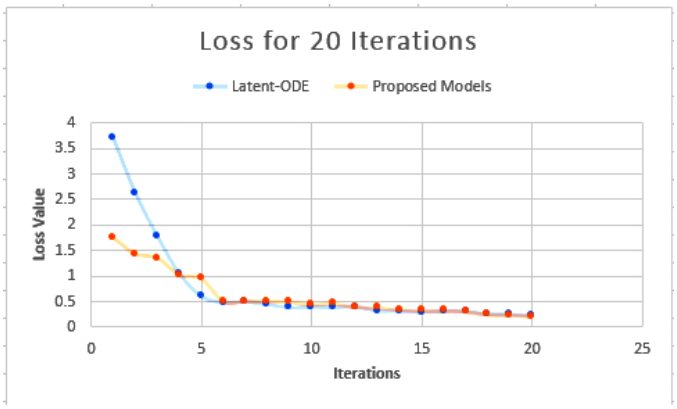

Fig. 7. Loss value for 20 iterations for Latent-ODE and proposed models

5) *Memory Usage:* Proposed models use the autograd to compute graphs for storing the previous state. However, not all previous states are relevant as it only focuses on the parameters of the model and the hidden dynamics. Based on the complexity of the dataset, the graph length can vary. Data with higher complexity uses more memory than a simple dataset. But a successful implementation with properly cleaning unnecessary state can control the memory usage for complex dataset.

IV. Conclusion

Unlike traditional RNN models, these proposed models do not need to discretize continuous time series into discrete-time sequences. In contrast to generic GRU model, this proposed ODE-GRU model can update the dynamics at irregularly sampled time points. Besides, the proposed ODE based two RNN models leverage the distinctive architecture of GRU and LSTM gating unit to model data-driven , event-driven asynchronously sampled input of continuous series. These ODE-GRU and ODE-LSTM models have significant advantages. Any large sequence can be learned within a short time and fewer parameters. ODE-GRU computes continuously, therefore, it is updated at every time slot where there is a change $\frac{d}{dt}h_t > 0$. If there is no change in consecutive time slots, ODE solver compute $\frac{d}{dt}h_t = 0$. As a result, no additional masking vector is required to distinguish missing values. The significant contribution of these proposed models is they are effortless to compute with faster training as well as evaluation time.

References

[1] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *Esann*, 2013.

[2] A. E. Bryson, Jr. A steepest ascent method for solving optimum programming problems. *Journal of Applied Mechanics*, 29(2):247, 1962.

[3] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[4] R. Chen. Differentiable ODE solvers with full GPU support and $O(1)$-memory backpropagation, Dec. 2018. URL https://github.com/rtqichen/torchdiffeq.

[5] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6572–6583, 2018.

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

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] D. Neil, M. Pfeiffer, and S.-C. Liu. Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890, 2016.

[9] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2): 263–9, 1989.

[10] Y. Rubanova, R. T. Chen, and D. K. Duvenaud. Latent ODEs for irregularly-sampled time series. *arXiv preprint arXiv:1907.03907*, 2019.