Proc. of the International Conference on Electrical, Computer and Energy Technologies (ICECET) 9-10
December 2021, Cape Town-South Africa

Neural Network based on Automatic Differentiation Transformation of Numeric Iterate-to-Fixedpoint

Mansura Habiba
Cloud Solution Lead IBM in Ireland Dublin, Ireland
0000-0001-9051-1370

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

2021 International Conference on Electrical, Computer and Energy Technologies (ICECET) | 978-1-6654-4231-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICECET52533.2021.9698707

***Abstract*—This work proposes a Neural Network model that can control its depth using an iterate-to-fixed-point operator. The architecture starts with a standard layered Network but with added connections from current later to earlier layers, along with a gate to make them inactive under most circumstances. These "temporal wormhole" connections create a shortcut that allows the Neural Network to use the information available at deeper layers and re-do earlier computations with modulated inputs. End-to-end training is accomplished by using appropriate calculations for a numeric iterate-to-fixed-point operator. In a typical case, where the "wormhole" connections are inactive, this is inexpensive; but when they are active, the network takes a longer time to settle down, and the gradient calculation is also more laborious, with an effect similar to making the network deeper. In contrast to the existing skip-connection concept, this proposed technique enables information to flow up and down in the network. Furthermore, the flow of information follows a fashion that seems analogous to the afferent and efferent flow of information through layers of processing in the brain. We evaluate models that use this novel mechanism on different long-term dependency tasks. The results are competitive with other studies, showing that the proposed model contributes significantly to overcoming traditional deep learning models' vanishing gradient descent problem. At the same time, the training time is significantly reduced, as the "easy" input cases are processed more quickly than "difficult" ones.**

***Index Terms*—Neural Network, Fixed Point Iteration**

I. Introduction

A well-known technique to preserve memory is Wormhole connection [1]. Wormhole connection in MANN [2] such as Neural Turing Machines [3] provides a shortcut connection to the previous hidden state through time by explicitly storing the previous hidden state in the memory. Temporal Automatic Relation Discovery in Sequences (TARDIS) [1] shows that the Wormhole connection created by the controller of the MANN can significantly reduce the effects of the vanishing gradients. Wormhole connection for MANN shortens the paths that the signal needs to travel between the dependencies. At every step of training, the controller ϕ in TARDIS as shown in Eq. (1), controls the hidden state $\mathbf{h}_{t-1}$ based on the content of weights $\mathbf{w}_t$ and memory $\mathbf{M}_t$ read from external memory.

$$\mathbf{h}_t = \phi(\mathbf{x}_t, \mathbf{h}_{t-1}, (\mathbf{M}_t^\top \mathbf{w}_t^r)) \tag{1}$$

In our proposed model, we leverage the strength of Wormhole connection. This work introduces a novel Neural Network model that uses Automatic Differentiation (AD) Transformation of Numeric Iterate-to-Fixed-point methodology for dataset training. In addition to dynamic training time for each layer, the proposed model can push backwards during training using a novel block in the architecture of a Deep Neural Network (DNN). The main contributions of this work are as follows,

- A novel DNN that leverage the Fixed-point iteration operator along with both the forward mode and the backward mode of AD. We call it Temporal Wormhole Neural Network (TWNN).
- New technique for training a DNN with the dynamic amount of time for different layers by creating push-backwards connection with the previous layer of the model.
- Finally, we show that the proposed DNN architectures demonstrate MNIST and Sine wave generation results with considerably improved precision and.

II. Motivation

The core concept of the TWNN model is to design an architecture for the Residual Neural Network

(RNN) or other DNN that allow efficient result with lower depth. TWNN model can have a low depth, i.e. 50, but each layer can have a dynamic training time instead of a fixed training time on each layer. Therefore, this new model can dynamically change the training time. Based on the performance of training on each layer ($\mathcal{L}_i$), the training method can decide to repeat the corresponding layer ($\mathcal{L}_i$) or move to either any previous layer ($\mathcal{L}_{i-k}$) or the next layer ($\mathcal{L}_i+1$). For the governance of this repetitive loop, we consider the Fixed-point-iteration approach. Mainly, this new model trains each layer until the output (z) settles down. To optimize the training time and minimize the cost, we enforce an additional codition for the loop to end. If the objective function (g) of the loop achieves an error less than or equal to a threshold value for error that we call the tolerance error (ε). TWNN can achieve accuracy by learning the data using the architecture of a 50-layer ResNet instead of using a deeper, e.g., 256-layer ResNet. The training time can vary, but the memory and computation consumption of the ResNet is significantly low with a shallow network of 50 layers. The motivation for the TWNN model is that a machine learning algorithm can reach the outcome sometimes right away, or sometimes it needs to think for a longer time. Therefore, it is not essential to always train a model for a fixed amount of time. For example, for each individual input x used in a DNN model f as shown in Eq. (2), the computation time of primal solution y varies. The computation time is short for some input x_1, and the corresponding solution can be achieved in time T_1. Similarly, for some input x_1, the computation time, T_2, for certain solution, where $T_2 \gg T_1$.

$$y = f(x) \tag{2}$$

As a result, a constant depth for the DNN models does not help to optimize the training. In addition, the depth of the model can vary for different batches of input during the training. Moreover, the memory requirement for different models constantly increases overtime. Therefore, most DNN adopts the push-forward technique where the training is unidirectional and skips connections are created to a future layer instead of a previous layer. However, in reality, learning a specific layer for a long time can achieve better accuracy, rather than learning each layer for a fixed time. Therefore, it can be helpful to create a connection with the previous layer by adopting a push-backwards technique. Therefore, our proposed model adopts both push-forward and push-backwards techniques to allow a bi-directional training mechanism.

In traditional ResNet, each layer is executed in a sequence, or the $IdentityBlock$ create a skip connection between the Basic ResNet $Block$ of the current layer and the last $ReLU$ layer. Thus, ResNet model always moves forward. $IdentityBlock$ help to optimize the training for ResNet, but at the same time, it also possesses some limitation. Creating skip connection using $IdentityBlock$ enable the gradient to avoid the main-stream flow of residual block weights, so theoretically, it can avoid learning useful pattern during training. However, this technique also provides a hypothesis that it is possible that only a few blocks can represent the hidden dynamics of the dataset while many other blocks contain very little information to learn useful representations of the data. Therefore, learning only a few blocks for a long time can optimize the performance of DNN models and provide a better result. In this work, we introduce Fixed-point iteration to control the training duration for each block of a DNN. Instead of learning each block, TWNN model focuses on blocks that provide useful representations for the hidden dynamics of the data for a longer time period.

In this work, a temporal wormhole connection between the current and previous layer of the model controls the flow of the hidden state. *Fixed-Point-Iterator* block controls the training time for each layer. The *Fixed-Point-Iterator* block in Fig. 1 takes the input (x) and run a fixed-point iteration loop until the error *err* $\leq \varepsilon$. The output for *Fixed-Point-Iterator* block is the output (x) and the next selected layer $\mathcal{L}$. If ($\mathcal{L}$) refers to a previous layer, the training restarted from that previous layer by creating an efferent connection between the selected layer ($\mathcal{L}$) and the main-stream for training. As shown in Fig 1, *Fixed-Point-Iterator* block can send the training to a previous layer or can forward it to the next layer of a Residual Neural Network.

At every layer, a local objective function finds the *fixed point*. Eq. (3) shows the condition for the *fixed point*. This loop iterate until difference between consecutive values x_{t-1} and x_t of variable x is less than the tolerance (ε) or the number of iteration is more than max-iter (the maximum number of iteration for each *Fixed-Point-Iterator* block).

$$\begin{aligned} &\text{for } t = 1, \ldots, \infty \ \{ \\ &\quad \mathrm{x}_t \leftarrow g\left(\mathrm{x}_t, \alpha, H_{t-1}, O_{t-1}\right); \\ &\quad \text{if } \|\mathrm{x}_t - \mathrm{x}_{t-1}\| \leq \varepsilon \text{ break; } \} \\ &\mathrm{z} = \mathrm{x}_t \end{aligned} \tag{3}$$

Here g computes the cost for each step locally. Fig. 2(c) shows that g takes the input of any current state(x_t), weight(α), hidden matrix(H_{t-1}) and output (O_{t-1}) of previous step ($t-1$) as input. For first iteration,t=1, $H_0 = null$ and $O_0 = null$, therefore, $H_1, O_1 = g(x_1, \alpha)$. Here, the hidden matrix $H1 =$

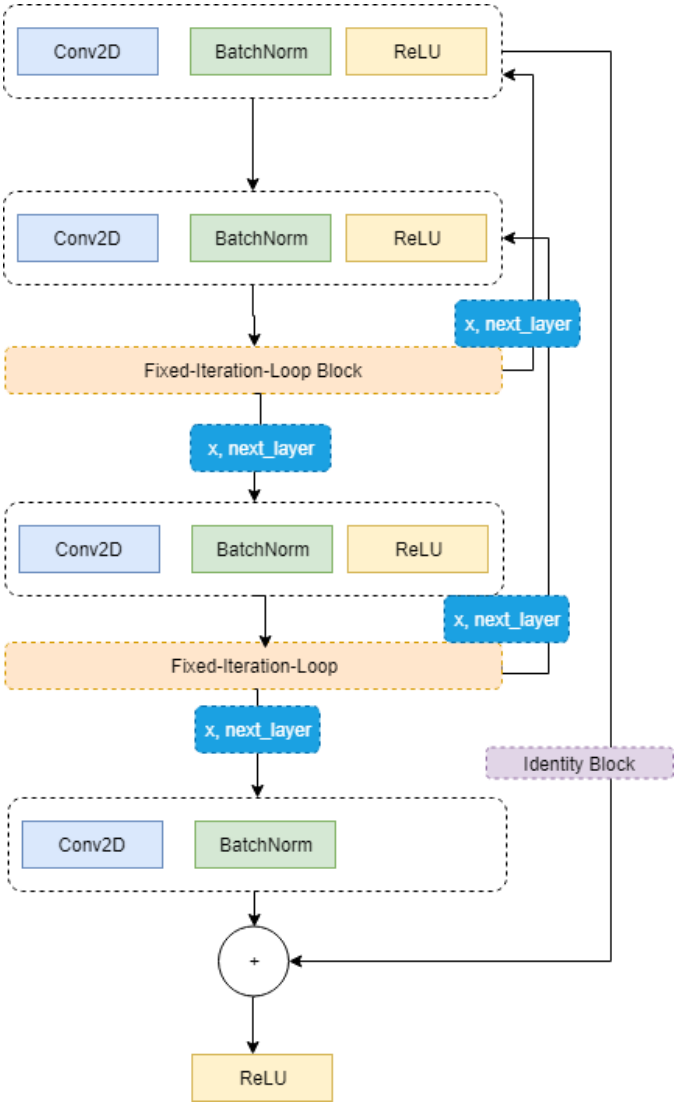


Fig. 1. Block Diagram of architecture of push-backward tech-nique for deep model

$(hidden1, hidden2, hidden3)$ as shown in Fig. 2(a). The hidden matrix of previous step $(t-1)$ is used as input for g in current step (t). Fig. 2(b) shows g computes the output for previous step $(t-1)$, which is used as input for current step t as shown as shown in Fig. 2(c). The loop $t = 1, \ldots, \infty$ iterates until z settles down. However, with the "wormhole" reverse connections as shown in "green" color in Fig. 2, the execution of g may iterate a bunch of times for z to settle down.

III. Model Design

A traditional ResNet usually has three different blocks

1) basic — two consecutive 3×3 convolutions followed by batch normalization and ReLU unit

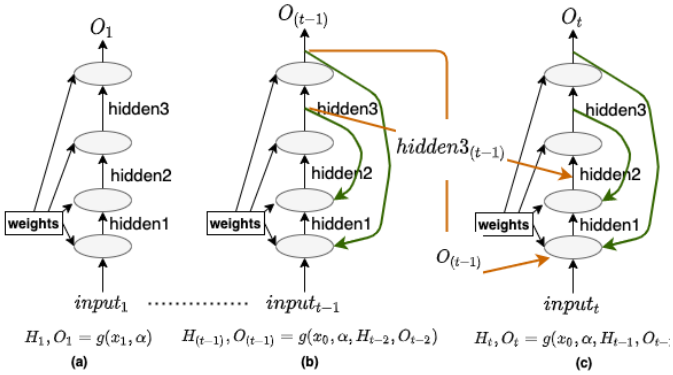


Fig. 2. The workflow for objective Neural Network g in Eq. (3)

2) bottleneck — one 3×3 convolution surrounded by dimensionality reducing and expanding 1×1 convolution layers
3) identityBlock — one skip connection with input and the ReLU unit

The sequence of three blocks repeats for a ResNet model based on the number of layers. This repetitive network can be described as Eq. (4).

$$x_D = \sum_{i=d}^{D-1} \mathcal{F}(x_i, W_i) \tag{4}$$

Here, $\mathcal{F}$ is a residual function. If a ResNet model is of depth D, Eq. (5) shows the distribution of layers of the ResNet model. Between the shallow layer (d) and deep layer (D), some layer (Dw) contributes to the final result more than other layers (Dp). So Dw layers learn useful representations of the hidden dynamics of the system and Dp layers provide very little information with a small contribution to the final goal for learning the system.

$$D = \sum_{j=0}^{P} Dw + \sum_{j=0}^{K} Dp \tag{5}$$

It is essential to identify layers (Dw) with useful representations of the hidden dynamics to optimize the model's performance. In this work, we introduce a new model that can identify the Dw layers and learn the system's hidden dynamics by learning these layers only. This new model has a *fixed-point-iterative* operator to control the training for the (Dw) layers. So for proposed model, Eq. (5) would look like Eq. (6).

$$D = \sum_{j=0}^{P} Dw -> (skip \sum_{j=0}^{K} Dp) \tag{6}$$

Fixed-Point-Iterator block trains current Dw_j layer. Once the training for current Dw_j layer is complete in the, *Fixed-Point-Iterator* block identified the next

layer $\mathcal{L}$ and create a connection between $\mathcal{L}$ layer and the main stream of training. *Fixed-Point-Iterator* block uses *fixed-point-iterative* operator to control the training for current Dw_j layer and enforce the above mentioned conditions to stop the loop. The *fixed-point-iterative* operator leverage Banach Fixed-point finder ***fix*** $:(\beta \to \beta) \to \beta$ that takes a contraction of a closed region which contains the initial point. The reverse AD transform has signature as shown in Eq. (7).

$$\begin{gathered}\overleftarrow{\mathcal{J}} : (\alpha_1 \to \cdots \to \alpha_n \to \beta) \to \\ \alpha_1 \to \cdots \to \alpha_n \to \\ (\beta \times (T^*\beta \to (T^*\alpha_1 \times \cdots \times T^*\alpha_n)))\end{gathered} \tag{7}$$

The forward and reverse transform of ***fix*** are shown as (8).

$$z = \text{fix}(fx) \quad f : \alpha \to \beta \to \beta \tag{8}$$

$$\begin{gathered}Tz = \text{fix}(\overrightarrow{\mathcal{J}} f(Tx)) \qquad \bar{f} : T^*\beta \to (T^*\alpha \times T^*\beta) \\ T^*x = \rho_1 \left(\text{fix}(\underbrace{\rho_2(\overleftarrow{\mathcal{J}} fxz)}_{\bar{f}} \circ (+(T^*z)) \circ \rho_2) \right)\end{gathered} \tag{9}$$

Here in Eq. (8), f, is a Neural Network. The weight to network (f) is defined by α. β represents the activity of all input. Therefore, $(\beta \to \beta) \to \beta$ can be used to compute the next moment activity. Table I describes different parameters in Equations 7, 8 and 9 .

TABLE I
THE NOTATIONS USED IN PROPOSED TEMPORAL WORMHOLE NEURAL NETWORK

$\overleftarrow{\mathcal{J}}$	Forward mode Algorithmic differentiation(AD) of network f
$\overrightarrow{\mathcal{J}}$	backward mode Algorithmic differentiation(AD) of network f
α	Weight to the proposed network
β	Activity to the proposed network
f	Neural Network for afferent connection
$\bar{f}$	Neural Network for efferent connection
T^*z	Gradient of all the activity with respect to z
$T^*\alpha$	Gradient of all the activity with respect to α
$T^*\beta$	Gradient of all the activity with respect to β
T^*x	Gradient of all the activity with respect to x

As shown in Eq.(8), the proposed Temporal Wormhole Neural Network (TWNN) uses Fixed-point iterations. For each input x, the forward pass of TWNN continue iterate until the output z in Eq.(8) settles. Fig 3 shows the loop for each fixed-point iteration for each batch input x. At the end of each loop, the controller $cntrl$ checks either z settles by checking if the conditions in Eq (3) are met.

Algorithm 1 explains the training method for proposed TWNN, the $model$ in algorithm 1 can be any

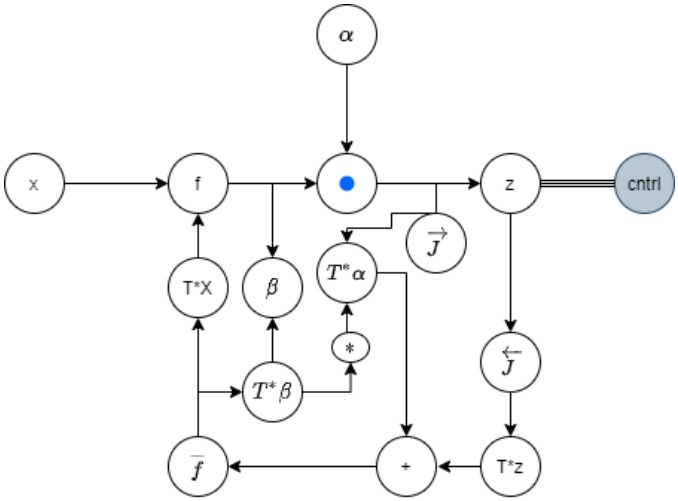


Fig. 3. Loop for Fixed-Point iteration of the model training method

Algorithm 1 Fixed point temporal wormhole based Neural Network

Input: Inputs are divided among batches, for each loop, a batch of inputs x_b^0, corresponding target y_b, learning rate l_r, tolerance error (ε), maximum iteration max_iter;

Output: predicted output z

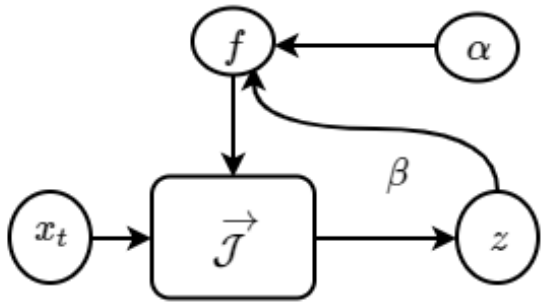
procedure $trainTWNNModel(x\_init)$
  $model, \alpha \leftarrow initializeNNBlock()$
  for each b in range $(total\_batch)$ **do**
    $model\_block \leftarrow next(model\_layers)$
    $loss = 1.0$
    while $model_b lock! = null$ **do**
      $z_b \leftarrow Fixed - Point - Iterator(model\_block, \alpha, x_b)$
      $loss = LossFun(z, y_b)$
      $model\_block = next(model\_layers)||identify\_prev\_layer(model)$
  return $trained_model$

Neural Network block such as GRU [4], ResNet [5] and others.

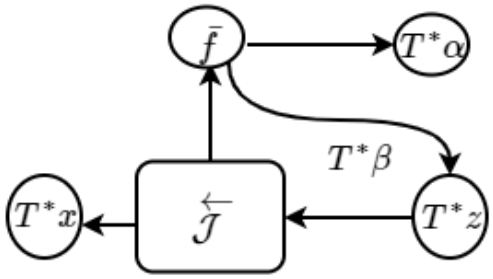
Proposed TWNN is initialized with the initial parameter $params$. *Fixed-Point-Iterator* can be computed using Forward mode AD and backward mode AD as shown in Eq. (10).

$$\textit{fixed-point-iterative} = (\overrightarrow{\mathcal{J}}\{f\}, \overleftarrow{\mathcal{J}}\{f\}) \tag{10}$$

Algorithm 2 describes the forward pass and backward pass for a single block of TWNN. Figure 4 explains the forward and backward computation of the *fixed-point-iterative* operator of proposed TWNN.



(a) Forward Mode for *fixedPoint*



(b) backward Mode for *fixedPoint*

Fig. 4. Forward and backward computation of the controller of TWNN

Algorithm 2 Forward and Backward Pass for fix function for a single block of TWNN

Input: The value of x at time t x_t, parameters α;
Output: Gradient of x

procedure $fix(f, \alpha, x_t)$
  Forward Pass:
  $z, \beta \leftarrow f(x_t, \alpha)$
  $T^*z, T^*\alpha = grad(f, z, \beta)$
  Backward Pass:
  $T^*x, T^*\beta \leftarrow \overline{f}(T^*z + T^*\beta)$
return T^*x

IV. Experimental Result

The performance of proposed model is evaluated against two common tasks:

1) MNIST [6] classification and
2) Sine wave generation.

The training parameters for the two tasks are described in Table II.

TABLE II
PARAMETERS FOR PROPOSED MODEL EVALUATION FOR MNIST CLASSIFICATION

Learning Rate (l_r)	0.001
Tolerance Error (ε)	0.001
Batch Length Size	64
Maximum Number of Iteration per batch (max_iter)	300
ε	0.01

A. Task-A : CIFR classification

For the training for this task, we have chosen a TWNN model described in Fig. 5. After the second block and fourth block, there is *Fixed-Point-iterator* Block. After finishing 2nd block of the ResNet model, *Fixed-Point-iterator* Block identify the next suitable block as B1 and the model continue training B1 and B2 blocks until the loss $>$ ε or iteration $<$ max-iter. After the condition is met, *Fixed-Point-iterator* forward the training to the B3 layer. Then, *Fixed-Point-iterator* Block after B3 identify the next layer as the final layer of the network and forward the training to layer B. Instead of training a deep ResNet-50 layer for this training, we trained a less deep RestNet-36 model. However, we trained specific layers of the model for a longer time. Therefore, the memory consumption is significantly less than DNN models. This model skipped block B4 during training. Therefore, the training time is not higher than ResNet.

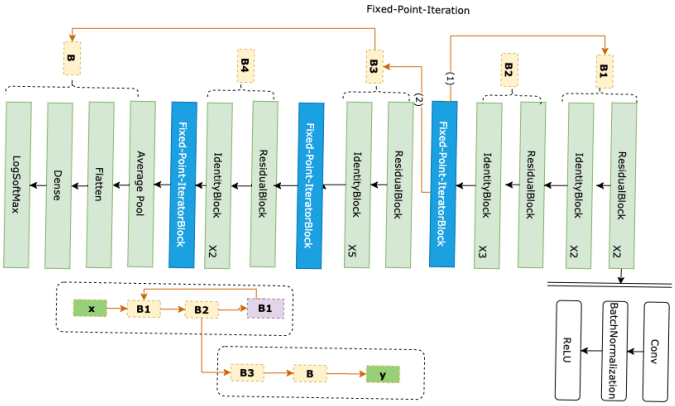


Fig. 5. TWNN model for CIFR training

Moreover, they show that the TWNN offers better results than the different DNN configurations. Table III shows the difference between the test accuracy of different similar configurations of ResNet and TWNN for CIFR.

B. Task-B: Sine Wave Generation

In this task, a corresponding sine wave is generated using the proposed TWNN model for a time

TABLE III
COMPARATIVE ACCURACY FOR PROPOSED MODEL AGAINST RESNET

Neural Network	Accuracy
RestNet-36	88.25%
RestNet-50	90.63%
TWRNN	96.54%

series T. Table IV shows the parameters used in this task.

TABLE IV
PARAMETERS FOR PROPOSED MODEL EVALUATION FOR CIFR CLASSIFICATION

parameter	value
Learning Rate (l_r)	0.001
Tolerance Error (ε)	0.001
Sequence Length Size	10000
Maximum Number of Iteration per batch (max_iter)	300

For this task, we take a simple Neural Network as the objective function shown in 3. $loss$ function as shown in 4 used this net procedure to compute gradient for the proposed TWNN.

Algorithm 3 Neural Network as the objective function for the proposed model

Input: The value of x at time t x_t, parameters α;
Output: the predicted value at time t x_t

procedure $net(f, \alpha, x_t)$
  $w1, b1, w2, b2 \leftarrow \alpha$
  $hidden \leftarrow tanh(dot(w1, x_t) + b1)$
  $y_t \leftarrow sigmoid(dot(w2, hidden) + b2)$
return y_t

Algorithm 4 Loss function for the proposed model

Input: The value of x at time t x_t, parameters α, target ;
Output: the predicted value at time t x_t

procedure $loss(net, \alpha, inputs, target)$
  $solver \leftarrow fix(net, \alpha, inputs)$
  $predictions \leftarrow solver(inputs, params)$
return $mean((targets - predictions) * *2)$

Fig. 6 shows the generated sine wave by training TWNN model for 300 iterations.

C. Discussion

The proposed TWNN model can be a suitable alternative against deep ResNet models with larger batch sizes; the TWNN model can achieve better accuracy with a shallow ResNet model. In the TWNN model, the same parameters are trained for a more extended period. Therefore, the corresponding memory requirement is lower than traditional DNN models.

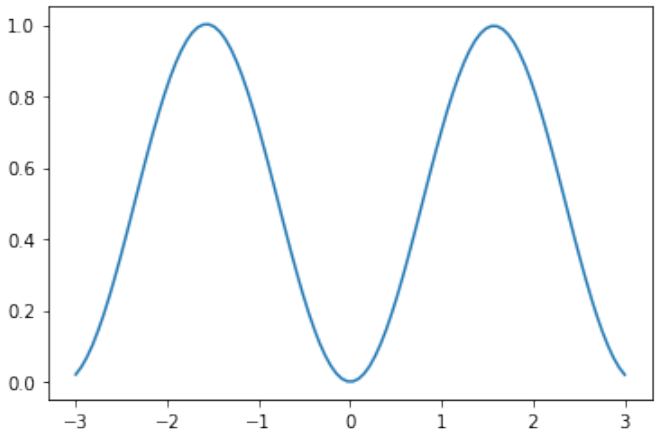


Fig. 6. Generated Sine Wave for time series

V. CONCLUSION

Proposed TWNN model leverages fixed-point iteration with automatic differentiation. In addition, TWNN model is explicitly more efficient as it computes gradients for input, hidden states and parameters. As only the influential layers are trained for a longer time, this new model can optimize the hidden dynamics of learning. The proposed model reaches accuracy faster than the related Neural Network as each of the batches is training under fixed point constraint through a fixed point iterated block. Therefore, the optimization of the TWNN model is more straightforward in comparison with other Deep Neural Networks. In future work, we will experiment proposed model with training data of higher dimension.

REFERENCES

[1] C. Gulcehre, S. Chandar, and Y. Bengio. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.
[2] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.
[3] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, Agnieszka Grabska-Barwi ´nska, Sergio G´omez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.
[4] R. Dey and F. M Salemt. Gate-variants of gated recurrent unit (GRU) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.
[5] J. D Keeler, Eric J Hartman, Kadir Liano, and Ralph B Ferguson. Residual activation neural network, October 4 1994. US Patent 5,353,207.
[6] Y. LeCun. The MNIST database of handwritten digits, 1998. Online resource.