Dynamically-biased Fixed-point LSTM for Time Series Processing in AIoT Edge Device

Jinhai Hu[1,2], Wang Ling Goh[2] and Yuan Gao[1]

Email: jinhai001@e.ntu.edu.sg ewlgoh@ntu.edu.sg gaoy@ime.a-star.edu.sg

[1]IC Design Department, Institute of Microelectronics, Agency for Science, Technology and Research, Singapore

[2]School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore

***Abstract*— In this paper, a Dynamically-Biased Long Short-Term Memory (DB-LSTM) neural network architecture is proposed for artificial intelligence internet of things (AIoT) applications. Different from the conventional LSTM which uses static bias, DB-LSTM adjusts the cell bias dynamically based on the previous status. Hence, a DB-LSTM cell contains information of both the previous output and the current cell state. With more information, the DB-LSTM is able to achieve faster training convergence and better accuracy. Furthermore, weight quantization is performed to reduce the weights to either 1-bit or 2-bit, so that the algorithm can be implemented in portable edge device. With the same 100 epochs training setup, more than 70% loss reduction are achieved for floating 32-bit, 1-bit and 2-bit weights, respectively. The loss degradation due to weight quantization is also negligible. The performance of the proposed model is also validated with the classical air passenger forecasting problem. 0.075 loss and 94.96% accuracy are achieved with 2-bit weight when compared to the ground truth, which is comparable to full-length 32-bit weight.**

Keywords*— *LSTM, Recurrent Neural Network, fixed-point weight, time series forecasting

I. Introduction

Artificial intelligence internet of things (AIoT) is an emerging research area that combines sensor and artificial intelligence to enable real-time intelligent data analysis on edge devices. Many types of data in our daily life are in time series format. For example, the voice signal and the human vital signs such as electrocardiogram (ECG), electroencephalogram (EEG), etc. Time series data have a natural temporal property. Hence, the ability to extract distinct signal features from the time series becomes an interesting topic and it has attracted great research efforts recently.

The Long Short-Term Memory (LSTM) is a widely used approach in Recurrent Neural Network (RNN). It can solve the problems of gradient disappearance and explosion during long sequence training. [2] demonstrated backward pass training to counteract linearly growth in forward pass to allow one dimensional (1-D) long sequence data to fit in the LSTM. Thereafter, a hierarchical multi-dimensional RNN is innovated which extended from unsegmented sequence data labelling [3]. Recently, [4] utilized multi-dimensional LSTM and achieved good results in multivariate feature applications. However, more researchers are seeking to modify LSTM to enhance its ability in handling complicated sequence data. Thus, Gated Recurrent Unit (GRU) and empirical evaluation were established and reported in [5]. Both GRU and LSTM were proven to adapt in different multivariate time series problems based on their performance in [6, 7].

On the other hand, edge device has limited hardware resources including computation capacity, memory size and battery lifetime. It remains a great challenge to implement the artificial intelligence algorithm on edge devices. LSTM has been implemented in hardware platforms such as FPGA [8, 9] and memristor crossbar arrays [10]. Hence, it is vital and necessary to design a novel cell structure for LSTM with both floating and fixed-point weight design, that can enhance the neural network performance on software level and also be applied and operated on edge devices.

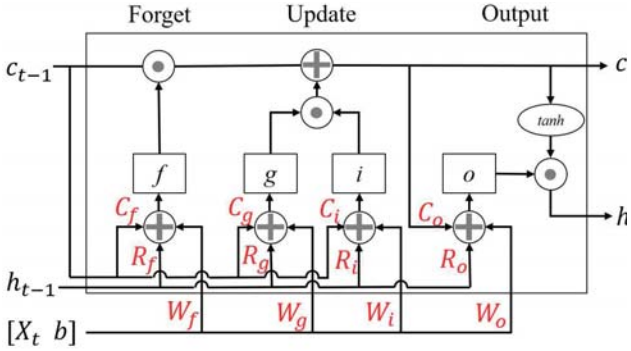

Fig. 1. The proposed DB-LSTM neural network structure.

TABLE I DB-LSTM Cell Structure Parameters

X_t	Current Input Vector
b	Bias
c_t	Current Cell State
h_t	Current Cell Output (Hidden State)
c_{t-1}	Previous Cell State
h_{t-1}	Previous Cell Output
f, g, i, o	Forget, Generate, Input, Output Statues
$W_{f,g,i,o}$	Input Weights for *f, g, i, o*
$R_{f,g,i,o}$	Recurrent Weights for *f, g, i, o*
$C_{f,g,i,o}$	Cell Weights for *f, g, i, o*

In this paper, a Dynamically-Biased LSTM (DB-LSTM) architecture is proposed to reduce training loss under fixed-point weight accuracy. An improved LSTM architecture with high accuracy, and fixed-point weight in 1-bit and 2-bit is proposed and validated at the algorithm level. It enhances the memory connection between cell state and four statuses in LSTM, and attained lower initial loss and higher loss reduction than the LSTM theoretical model reported in [1]. The DB-LSTM also showed good performance in sequence forecasting. This paper is organized as follows: Section II introduces the DB-LSTM cell structure. The fixed 1-bit, 2-bit weight quantization is discussed in Section III. The simulation results of both the theoretical and real-life data are presented in Section IV. Lastly, a conclusion is drawn in Section V.

II. DB-LSTM Model Architecture

A. Structure Illustration

The DB-LSTM cell structure is shown in Fig. 1, and the corresponding parameters are listed in Table I. The model consists of four statuses, that discards non useful message, update memory, and control the output, respectively. To

standardize the operation process at time step *t*, the current input X_t is a 1-D vector with $1 \times m$ size that contains *m* features to be trained, and the output dimension is $1 \times n$. Different from the conventional structure, the four bias in this model are not of fixed random numbers since they are simplified into single factor and to be combined with input vector. Initially, the bias of *f, g, i, o* statuses share the same random number. Following which they are split into four different bias due to the $W_{f,g,i,o}$ adjustment throughout the training. Thus, the bias can be fine-tuned, providing two-fold advantages to enable the gate output to be more accurate, as well as smooth the back-propagation loss gradient. As a result, the overall input vector is $[X_t \;\; b]$ with dimension $1 \times (m + 1)$, and its corresponding $W_{f,g,i,o}$ has the size $n \times (m + 1)$. If the sequence length is set to *k*, the total training steps within one epoch is *k*, and the input and output matrix of the DB-LSTM neural network are then $k \times (m + 1)$ and $k \times n$, respectively.

The cell state and $C_{f,g,i,o}$ are the showcases for advanced processing in this model. Different from the conventional LSTM structure, *f, g, i, o* statuses have one additional parameter, that is, the cell state. *f, g* and *i* gates consider both h_{t-1} and c_{t-1}, while *o* gate counts in the current cell state c_t instead of c_{t-1}. The four statuses are expressed in (1) – (4)

$$f_t = \sigma\big([X_t \quad b]W_f^T + h_{t-1}R_f^T + c_{t-1}C_f^T\big) \tag{1}$$

$$g_t = tanh\big([X_t \quad b]W_g^T + h_{t-1}R_g^T + c_{t-1}C_g^T\big) \tag{2}$$

$$i_t = \sigma([X_t \quad b]W_i^T + h_{t-1}R_i^T + c_{t-1}C_i^T) \tag{3}$$

$$o_t = \sigma([X_t \quad b]W_o^T + h_{t-1}R_o^T + c_tC_o^T) \tag{4}$$

where σ is the sigmoid function.

Since the hidden state h_t is a selected and scaled result from the cell state c_t, it may cause a huge difference upon final decision. As a result, *f* and *i* gates will analyse current input, previous cell state and the output to yield a moderated identification on the level of old messages and the latest information that the LSTM cell should discard or remember in . Similarly, the output gate *o* also monitors the current cell state to determine what needs to be sent to the hidden output. The difference between the selection of c_t and c_{t-1} is that *o* is the gating machine working on the process from the current cell state to the hidden state, while *f* and *i* are deciding on how the old data can affect the current *c*.

The Generation Status *g* prepares a group of preliminary information that is to be added into the cell state, thus, *tanh* is selected as the active function. According to the LSTM flow chart shown in Fig. 1, the cell state is updated firstly by the forget status followed by the input status.

Also, a single directional LSTM can only gain the learnable features from its previous information and cannot be influenced by the later key words. This is resolved by using bi-directional LSTM [11]. However, bi-directional LSTM doubles the size of all types of weight in its cell structure, which will require more memory space in hardware implementation. Therefore, in this model, to prevent the forget gate to accidentally abandon some useful data that possibly can have a strong connection with the later information, the add-on c_{t-1} in *g* will cause the update process to re-consider some data stores in the previous cell state, so that it can recover those data and decline initial information loss during the first epoch training. Moreover, the back-up data sent to *g* is filtered by its weight C_g, and the recovered message is determined by the input data and gating signal *i*.

The update of cell state is given by (5)

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{5}$$

where $\odot$ is the Hadamard product that states every entry in the old cell state and needs to be updated to attain the new cell state. (6) is the last step in feed forward pass to compute the hidden output.

$$h_t = o_t \odot \tanh(c_t) \tag{6}$$

In summary, the input parameters decrease due to the injection dynamic bias, which provides a better adjustment to the statuses' operations. The cell state enables the gating selection to be more accurate and also resilient to random events. And the cell state parameter in g_t allows LSTM to hold a larger memory and reduce the effect of single direction LSTM's deficiency. In forward pass, (1) - (3) are parallel operations, while (4), (5) and (6) are series operations. Thus, the current process gains the memory with more than one step as compared to the previous since c_{t-1} is computed in advance of h_{t-1}.

B. Back Propagation and Weight Update

The back propagation (BP) algorithm is used to adjust the three groups of weights. The inputs for BP are the output error and the established cell structure that contains all the setup parameters. The error matrix *e* stores the difference between the real values *y* and training result, which is the hidden output h_t. An intermediate function to simplify the BP complexity is shown in (7).

$$\begin{cases} (f,i,o)(t) = \sigma[(f,i,o)_s(t)] \\ g(t) = \tanh[g_s(t)] \end{cases} \tag{7}$$

The following equations (8) - (10) show the propagation step results for the DB-LSTM after BP derivation process. To synthesize the notation, φ is defined as the set of *f, g, i, o*, and $\delta\varphi_s$ needs to be derived from time step 1 to *k* to update the weights *W*, *R*, and *C*. For time step *t* reverses back from *k* to 1,

$$\delta h(t) = e(t) + \sum_{\varphi=f,g,i,o} \delta\,\varphi_s(t+1) \times R_\varphi \tag{8}$$

$$\delta o(t) = \delta h(t) \cdot \tanh\big(c(t)\big) \tag{9}$$

$$\delta o_s(t) = \delta o(t) \cdot \sigma'\big(o(t)\big) \tag{10}$$

where the result of $\delta o_s(t)$ is found and the rest $\delta\varphi_s$ can be calculated by iteration. Hence, the updated weights $\delta(W,R,C)_\varphi$ are the time average value based on $\delta\varphi_s$ shown in (11) – (13)

$$\delta W_\varphi = \frac{\delta\varphi_s^T \times x}{k} \tag{11}$$

$$\delta R_\varphi = \frac{\delta\varphi_s(2:end)^T \times h(1:end-1)}{k-1} \tag{12}$$

$$\delta C_\varphi = \begin{cases} \left[\frac{\sum \delta\varphi_s(2:end)\cdot h(1:end-1)}{k-1}\right]^T, \varphi = f,g,i \\ \left[\frac{\sum \delta\varphi_s \cdot h}{k-1}\right]^T, \varphi = o \end{cases} \tag{13}$$

The " · " operation stands for element-wise product, while " × " gives the matrix multiplication. " ′ " is the derivative notation and " ^T " is the transpose of the corresponding vectors or matrices. The ultima weight-change $\Delta(W,R,C)_\varphi$ are

TABLE II DETAILED PARAMETER IN THREE TYPES OF WEIGHT PRECISION

	Initial 32 bit	Fix 2-Point	Fix 1-Point
Initial Loss	0.1537	0.1669	0.1751
Final Loss	0.0425	0.0446	0.0453
Loss Reduction	72.35%	73.23%	74.13%
Loss at 5th epoch	0.0443	0.0591	0.0857
Front 5 epoch Loss Reduction	72.35%	64.59%	51.06%

the multiplication of learning rate and $\delta(W,R,C)_\varphi$, which may also consider the effect of penalty if it is provided. Moreover, the updated weights can be further fine-tuned by the gradient condition weight update [12], which states that the final changeable weight is the gradient-based partial amount of $\Delta(W,R,C)_\varphi$ in (11) – (13).

Equation (14) is the final step of the BP process and is within one epoch, where the three groups of weights are successfully updated for the program to run to the next epoch.

$$\Delta(W,R,C)_\varphi = gradient \cdot (W,R,C)_\varphi - \Delta(W,R,C)_\varphi \quad (14)$$

III. FIXED-POINT WEIGHT QUANTIZATION

The weights in the AI algorithms are normally represented in 32-bit floating point format. However, as the goal of this work is to implement the algorithm in portable device, it is important to quantize the weights. So that they can fit into the limit hardware resources of edge device with acceptable accuracy degradations.

In the fixed-point weight implementation, all the weights are quantized to signed 2×2^n states including negative values, which shows that all the entries in each class of weights from $(W,R,C)_\varphi$ can only be selected from $\pm2^n$ types of numbers. The quantization procedure is to firstly find the maximum and minimum numbers in that particular class of weight, noted as w_{max} and w_{min}, respectively, which create the boundary of states, and w_{min} is the 1st state. The interval of two states is given by

$$\Delta w = \frac{w_{max} - w_{min}}{2^n - 1} \quad (15)$$

where the classify interval is half of Δw. For each entry w_j in the original weight matrix, if it falls within the satisfied range shown in (16), it will be classified as the $(\beta+1)^{th}$ state,

$$\left|w_j - (w_{min} + \beta \times \Delta w)\right| \le \frac{\Delta w}{2} \quad (16)$$

where β = 0, 1, 2, ⋯ 2^n - 1. Fig. 2 illustrates the loss comparation of 1-, 2- and 32-bit precision weight. The fix-weight transformation process works for every epoch from weight initialization to the end. The plot for loss shows that 32-bit weight achieved the best result with lowest initial and final losses, as well as having the highest loss drop-off. The 2-bit precision weight came second in performance and the single bit weight is the worst off. However, it is still able to obtain a fast loss reduction and only 0.05% higher loss than the 32-bit. The detailed characteristics of all three types of loss results can be found in Table II. Although the initial loss for fix point weight is higher than the default 32-bit, they are all able to achieve similar or even higher loss reduction than the default 32-bit. On the other hand, within a limited training time, the 32-bit weight can yield a much better outcome.

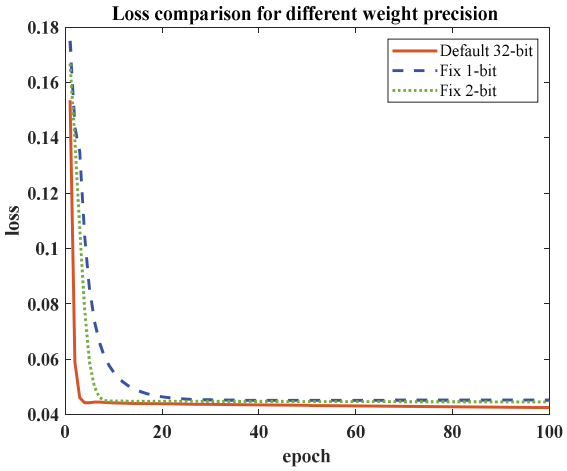

Fig. 2. Training loss comparison for default 32 bit, fixed 1-bit and 2-bit weight precisions.

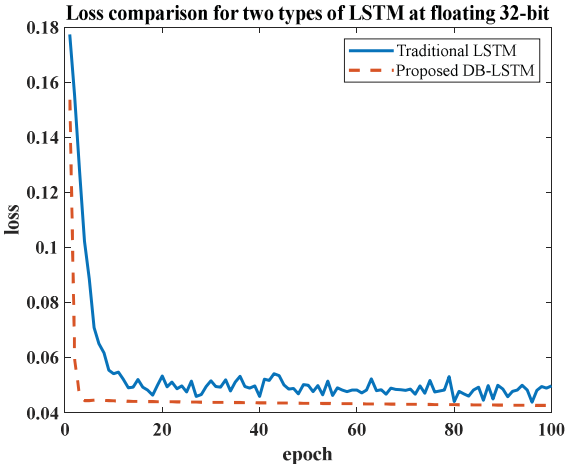

Fig. 3. Comparison of training loss at default 32-bit weight.

IV. RESULTS AND DISCUSSION

A. Theoretical Model Performance

The theoretical performance shown in Fig. 3 is a comparison of loss between traditional LSTM [1] and the proposed DB-LSTM model with 32-bit precision weight. The training data is a multi-sequence to sequence prediction. Here, the input is a 200×15 matrix and the output is a 200×1 vector, where 200 is the sequence length. It can be observed that the proposed method achieved better performance than the original LSTM in both final loss and training stability. Also, this model attained lower initial loss.

B. Application on Real Life Problem

To verify the performance of this DB-LSTM model solving practical problem, a time series forecasting for air passengers [13] is evaluated.

In this problem, a time series of monthly traveller numbers is fed into the model, while the label output is the same sequence but delay by one month. The aim is to have the model forecasts the next month's passenger volume by providing the current data, and then repeat this procedure to predict for as many months as it can. Fig. 4(a) shows the accuracy and loss diagram in three precision weights to train the model with prior 109 months' data, whereby the inputs are from 1 to 108 and the label output is from 2 to 109. The accuracy of this model reached 95.49%, 93.97%, 94.96% for

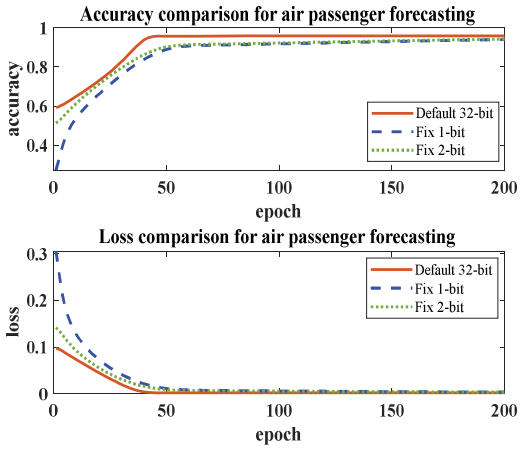

Fig. 4 (a). Comparison of training accuracies and losses for air passenger forecasting with different weight quantization accuracies.

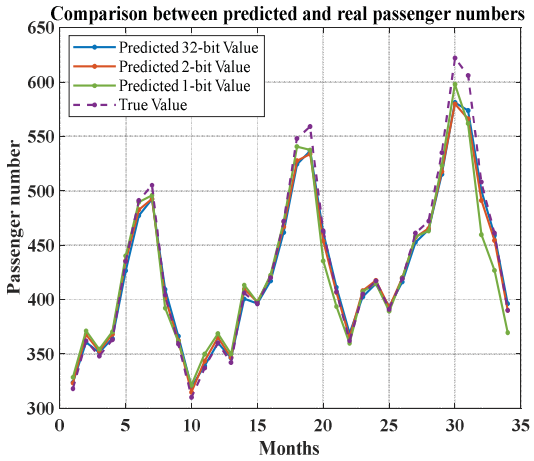

Fig. 4 (b). Comparison between air passenger number true value and forecast results with different weight quantization accuracies.

32bit, fix 1-, and 2-point weights, correspondingly. Fig. 4(b) shows the difference between real and predict passengers in the next 34 months with three types of weight precision, from which the forecasting results for the next two years indicated a small diversity. The mean square error for 32-bit, 2-bit and 1-bit weight validation are 156.34, 168.77 and 271.26, respectively using the same training parameters.

Additionally, this model can yield satisfactory results in a shorter training time for one-dimension time series forecasting. This implies a higher probability for online training on edge devices. In the 2-bit precision experiment, it takes 200 epochs to reach a 94.96% accuracy and around 96% accuracy for 1000 epochs. This is of a better result than [10] which required 800 epochs training using the same dataset. A similar ELSTM [14] was presented with less gates for time series forecasting and attained 90.89% accuracy after training for 20,000 epochs. It saves on hardware cost but is only suitable for pre-training due to its long training period. Compared with the 32-bit full-precision baseline, the 1-bit and 2-bit quantization achieved approximately 16.7% average memory saving and 9.8% inference acceleration on CPUs, with only a reasonable loss in the accuracy.

V. Conclusion

In this paper, a novel model of the LSTM cell structure for time series forecasting using both floating and fix-point weights on software platform are presented. The cell structure is designed with dynamic bias and advanced memory connection where all statues are linked to previous or current cell state. Moreover, fixed 1-bit and 2-bit weights are implemented onto the DB-LSTM model to fit into edge device for further purpose, which achieved satisfactory results contrasting with 32-bit weight. Compared with traditional LSTM model, the DB-LSTM yields a lower initial loss, had fast and stable training procedure, and provides higher accuracy. The results obtained from both theoretical and practical problem applications have proven that this innovative model can attain high quality performance, which is the most important factor when using dynamic bias and advanced memory connection onto LSTM cell structure.

Acknolodgement

This work was supported by Agency for Science, Technology and Research (A*STAR), Singapore under the Nanosystems at the Edge programme (Grant No. A18A1b0055)

References

[1] Hochreiter, S., and J. Schmidhuber. "Long short-term memory," Neural computation. Vol. 9, Number 8, 1997, pp.1735-1780.

[2] F. Gers, N. Schraudolph, and J. Schmidhuber, “Learning precise timing with LSTM recurrent networks,” Journal of Machine Learning Research, 3:115–143, 2002.

[3] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," In Proceedings of the International Conference on Machine Learning (ICML), Pittsburgh, USA, 2006.

[4] Gundram Leifert, Tobias Strauß, Tobias Grüning, Welf Wustlich, and Roger Labahn, "Cells in multidimensional recurrent neural networks," J. Mach. Learn. Res. 17, 1, 3313-3349, 2016.

[5] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, " Empirical evaluation of gated recurrent neural networks on sequence modeling," In NIPS 2014 Workshop on Deep Learning, December 2014.

[6] Peter T. Yamak, Li Yujian, and Pius K. Gadosey, "A Comparison between ARIMA, LSTM, and GRU for Time Series Forecasting," In Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence. Association for Computing Machinery, New York, NY, USA, 49-55, 2019.

[7] Z. Che, S. Purushotham, K. Cho, et al., "Recurrent Neural Networks for Multivariate Time Series with Missing Values," Sci Rep 8, 6085, 2018. https://doi.org/10.1038/s41598-018-24271-9

[8] Q. Liu et al., "A fully integrated analog ReRAM based 78.4TOPS/W compute-in-memory chip with fully parallel MAC computing," 2020 IEEE International Solid- State Circuits Conference - (ISSCC), San Francisco, CA, USA, 2020, pp. 500-502.

[9] Y. Zheng, et al, "A high energy-efficiency FPGA-based LSTM accelerator architecture design by structured pruning and normalized linear quantization," 2019 International Conference on Field-Programmable Technology (ICFPT), 2019, pp. 271-274.

[10] C. Li, Z. Wang, M. Rao, et al. "Long short-term memory networks in memristor crossbar arrays," Nat Mach Intell 1, 49-57, 2019. https://doi.org/10.1038/s42256-018-0001-4

[11] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," in IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673-2681, Nov. 1997, doi: 10.1109/78.650093.

[12] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," In Proceedings of the 30th International Conference on Machine Learning (ICML'13). ACM, 2013. URL http://icml.cc/2013/.

[13] Kaggle.com. 2020. Air Passengers. [online] Available at: https://www.kaggle.com/chirag19/air-passengers.

[14] K. Khalil, O. Eldash, A. Kumar and M. Bayoumi, "Economic LSTM Approach for Recurrent Neural Networks," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 66, no. 11, pp. 1885-1889, Nov. 2019, doi: 10.1109/TCSII.2019.2924663.