FIXED-POINT PERFORMANCE ANALYSIS OF RECURRENT NEURAL NETWORKS

Sungho Shin, Kyuyeon Hwang, and Wonyong Sung

Department of Electrical and Computer Engineering
Seoul National University
Seoul, 08826 Korea
Email : shshin@dsp.snu.ac.kr, kyuyeon.hwang@gmail.com, wysung@snu.ac.kr

ABSTRACT

Recurrent neural networks have shown excellent performance in many applications; however they require increased complexity in hardware or software based implementations. The hardware complexity can be much lowered by minimizing the word-length of weights and signals. This work analyzes the fixed-point performance of recurrent neural networks using a retrain based quantization method. The quantization sensitivity of each layer in RNNs is studied, and the overall fixed-point optimization results minimizing the capacity of weights while not sacrificing the performance are presented. A language model and a phoneme recognition examples are used.

Index Terms— recurrent neural network, quantization, word length optimization, fixed-point optimization, deep neural network

1. INTRODUCTION

Recurrent neural networks (RNNs) employ feedback paths inside, and they are suitable for processing input data whose dimension is not fixed. Important applications of RNNs include language models for automatic speech recognition, human action recognition and text generation [1, 2, 3].

Among many types of RNNs [4, 5, 6], the most powerful one is the long short term memory (LSTM) RNN [5]. A standard LSTM RNN is depicted in Figure 1. A layer of LSTM RNN contains an input gate, an output gate, and a forget gate. An LSTM layer with N units demands a total of approximately $4N^2 + 4NM + 5N$ weights where M is the unit size of the previous layer. Considering a character level language model that employs three 1024 size LSTM layers, the network demands about 22.3 million weights. Thus, reducing the size of weights in RNNs is very important for VLSI or embedded computer based implementations.

There have been many studies on efficient hardware implementation of artificial neural networks applying direct quantization [7, 8, 9, 10]. Retraining based quantization that readjust the fixed-point weights by training after direct quantization of floating-point parameters was adopted in [11, 12, 13]. Previous research works for retrain-based fixed-point optimization use 3-8 bits for implementing feed-forward neural networks (FFNNs) and convolutional neural networks (CNNs). However RNN (especially LSTM RNN) employs a more complex feed-back based structure and hence optimum quantization is challenging.

In this paper, we optimize the word-length of weights and signals for fixed-point LSTM RNNs using the retraining method. The proposed scheme consists of three parts which are floating point training, sensitivity analysis of fixed-point RNNs and retraining. To the best of our knowledge, this is the first work that tries fixed-point quantization of large size recurrent neural networks.

This paper is organized as follows. In Section 2, the quantization procedure for weights and signals is given. Section 3 describes layerwise fixed-point sensitivity analysis for weights and signals. Experimental results are provided in Section 4 and concluding remarks follow in Section 5.

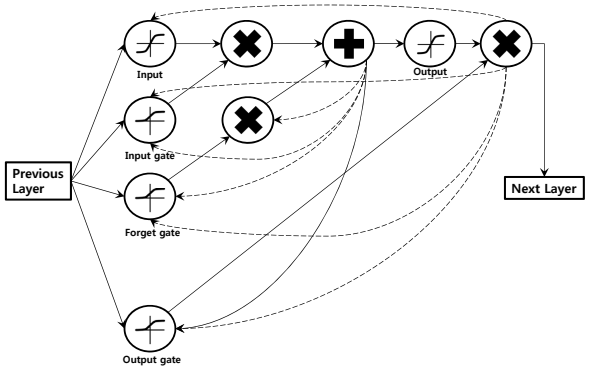
2. RETRAIN-BASED WEIGHT QUANTIZATION

The retrain based quantization method includes the fixed-point conversion process inside of the training procedure so that the network learns the quantization effects [12]. This method shows much better performance when the number of bits is small. In this method, the floating-point weights for RNNs are prepared with the backpropagation through time (BPTT) algorithm [14]. BPTT begins by unfolding a recurrent neural network through time, then training proceeds in a manner similar to adapting a feed-forward neural network with backpropagation.

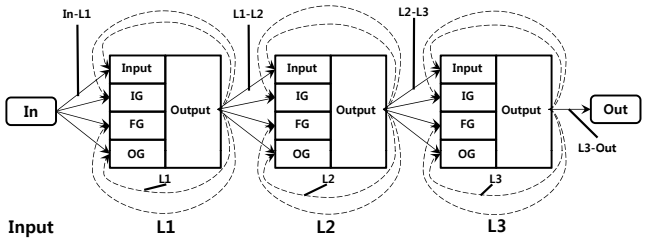
For direct quantization, a uniform quantization function is employed and the function $Q(\cdot)$ is defined as follows:

$$Q(w) = sgn(w) \cdot \Delta \cdot min\left(\left\lfloor \frac{|w|}{\Delta} + 0.5 \right\rfloor, \frac{M-1}{2} \right) = \Delta \cdot z, \quad (1)$$

This work was supported in part by the Brain Korea 21 Plus Project and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2015R1A2A1A10056051).



(a) Each layer of LSTM RNN



(b) Whole connection of three layer LSTM RNN

Fig. 1. The standard structure of LSTM RNN. Each circle represents one layer which is part of the LSTM. A dotted line means a backward path and a solid line is a forward path. The plus and multiplication signs show aggregation functions for summing and multiplication, respectively. The graph in the circles means an activation function for logistic sigmoid or tanh.

where $sgn(\cdot)$ is the sign function, Δ is a quantization step size, w is the set of the floating-point weights, and M represents the number of quantization levels. Note that M is normally an odd number since the weight values can be positive or negative. When M is 5, the weights are represented by -2Δ, $-\Delta$, 0, Δ, and 2Δ, which can be stored in 3 bits.

For selecting a proper step size Δ, the L2 error minimization criteria is applied as adopted in [12]. The quantization error E is represented as follows:

$$E = \frac{1}{2}\sum_{i=1}^{N}\big(Q(w_i) - w_i\big)^2 = \frac{1}{2}\sum_{i=1}^{N}\big(\Delta \cdot z_i - w_i\big)^2, \quad (2)$$

where N is the number of weights in each layer, w_i is the i-th weight value in floating-point, and z_i is the integer membership of w_i. The quantization error E is minimized by the following two step iterative computation.

$$\begin{aligned} \boldsymbol{z}^{(t)} &= \underset{\boldsymbol{z}}{\operatorname{argmin}} E(\boldsymbol{w}, \boldsymbol{z}, \Delta^{(t-1)}) \\ &= sgn(w_i) \cdot min\left(\left\lfloor \frac{|w_i|}{\Delta^{(t-1)}} + 0.5 \right\rfloor, \frac{M-1}{2}\right) \quad (3) \end{aligned}$$

$$\Delta^{(t)} = \underset{\Delta}{\operatorname{argmin}} E(\boldsymbol{w}, \boldsymbol{z}^{(t)}, \Delta) = \frac{\sum_{i=1}^{N} w_i \cdot z_i^{(t)}}{\sum_{i=1}^{N} \big(z_i^{(t)}\big)^2}, \quad (4)$$

where the superscript (t) indicates the iteration step. The first step equation (3) can be computed using (1). The second step equation (4) can be solved by using the derivative of the error with respect to $\Delta^{(t)}$ to be zero. The iteration stops when $\Delta^{(t)}$ is converged.

After obtaining the fixed-point weights, $z_i\Delta$, the retraining procedure follows. We maintain both floating-point and quantized weights, since applying BPTT algorithm directly with quantized weights usually does not work. The reason is the amount of weights to be changed on each training step is much smaller than the quantization step size Δ. Assuming that the RNN is unfolded, the algorithm can be described as follows:

$$\begin{aligned} net_i &= \sum_{j \in A_i} w_{ij}^{(q)} y_j^{(q)} \\ y_i^{(q)} &= R_i(\phi_i(net_i)) \quad (5) \end{aligned}$$

$$\delta_j = \phi_j^{'}(net_j) \sum_{i \in P_j} \delta_i w_{ij}^{(q)} \quad (6)$$

$$\frac{\partial E}{\partial w_{ij}} = -\delta_i y_j^{(q)} \quad (7)$$

$$\begin{aligned} w_{ij,new} &= w_{ij} - \alpha\left\langle \frac{\partial E}{\partial w_{ij}} \right\rangle \\ w_{ij,new}^{(q)} &= Q_{ij}(w_{ij,new}) \quad (8) \end{aligned}$$

where net_i is the summed input value of the unit i, δ_i is the error signal of the unit i, w_{ij} is the weight from the unit j to the unit i, y_j is the output signal of the unit j, α is the learning rate, A_i is the set of units anterior to the unit i, P_j is the set of units posterior to the unit j, $R(\cdot)$ is the signal quantizer, $Q(\cdot)$ is the weight quantizer, $\phi(\cdot)$ is the activation function, the superscript (q) indicates quantization, and $\langle\cdot\rangle$ is an average operation via the mini-batch. Equation (5), (6), (7), and (8) represent the forward, backward, gradient calculation, and weights update phases each.

3. QUANTIZATION SENSITIVITY ANALYSIS

LSTM RNNs usually contain millions of weights and thousands of signals. Therefore, it is necessary to group them

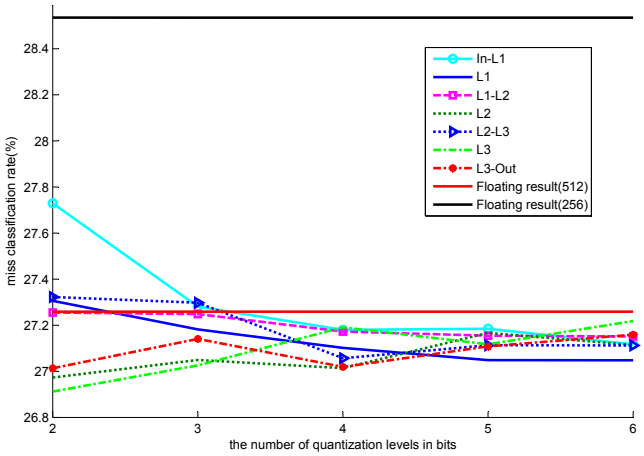


Fig. 2. Layerwise sensitivity analysis results of the weights in the phoneme recognition example. The red and black horizontal lines indicate the floating-point results for 512 LSTM size and 256 LSTM size each.

according to their range and the quantization sensitivity [15]. Fortunately, a neural network can easily be layerwisely grouped. Throughout this sensitivity check, we can identify which layer in the neural network needs more bits for quantization. The network for phoneme recognition contains three hidden RNN layers. Thus the weights can be grouped into 7 groups, which are In-L1, L1, L1-L2, L2, L2-L3, L3, and L3-Out groups, where In-L1 connects input and the first LSTM layer and L1 is the recurrent path in the first level LSTM. Figure1.(b) illustrates the grouping. In the sensitivity analysis, we only quantize the weights of the selected group while those in other groups are unquantized. Signal layerwise sensitivity analysis also follows the same scheme. In the standard LSTM RNN, the popular activation functions are logistic sigmoid or tanh.

$$sigmoid(x) = \frac{1}{1+e^{-x}} \tag{9}$$

$$tanh(x) = \frac{e^{x}+e^{-x}}{e^{x}-e^{-x}} \tag{10}$$

The output ranges of the sigmoid and the tanh are limited by 0 to 1 and -1 to 1, respectively. The quantization step size Δ is determined by the quantization level M. For example, if the signal word-length is two bits (M is four), the quantization points are 0/3, 1/3, 2/3, and 3/3 for the sigmoid and -1/1, 0/1, and 1/1 for the tanh. However signals of linear units are not bounded and their quantization range should be determined empirically. In our phoneme recognition example, each component of the input data is normalized to have zero mean and a unit variance over the training set. The input range is chosen to be from -3 to 3. One hot encoding is used for the input linear units in the language model example.

Table 1. Frame-level phoneme error rates (%) on the test set with the TIMIT phoneme recognition example using quantized weights and signals. Numbers in the parenthesis indicate the ratio of the weights capacity compared to the floating-point version.

Layerwise quantization bits		FER(%)	
Weights bits (In-L1, L1, L1-L2, L2, L2-L3, L3, L3-Out)	Signal bits (Input, L1, L2, L3)	Direct	Retrain
3-2-2-2-2-2-2 (6.39%)	4-4-3-5	48.00	28.74
4-3-3-3-3-3-3 (9.52%)	4-4-3-5	34.37	28.87
3-2-2-2-2-2-2 (6.39%)	5-5-4-6	31.65	27.79
4-3-3-3-3-3-3 (9.52%)	5-5-4-6	31.54	27.74

4. EXPERIMENTAL RESULTS

In this section, we will first show the result of the layerwise sensitivity analysis for signals and weights, and then fully quantized network performances. The proposed quantization strategy is evaluated using two RNNs, one for phoneme recognition and the other for character level language modeling. Advanced training techniques such as early stopping, adaptive learning rate, and Nesterov momentum are employed [16, 17, 18].

4.1. Phoneme Recognition

Phoneme recognition experiments were performed on the TIMIT corpus [19]. The detailed experimental conditions are the same with [20] except the mapping of output classes for scoring. Therefore, the input layer consists of 123 linear units (Fourier-transfomed-based filter-bank with 40 coefficients plus energy with their first and second temporal derivatives). Three hidden LSTM layers have the same size of 512. The output layer consists of 61 softmax units which correspond to 61 target phoneme labels. The network is trained using the Fractal with the training parameters of 32 forward steps and 64 backward steps with 64 streams [21]. Initial learning rate was 10^{-5} and it is decreased at proper timing until 10^{-7}. Momentum was 0.9 [22] and adadelta was adopted for weights update [23]. The network demands approximately 5.5 million weights. As a result it needs about 22MB with a 32bit floating-point format.

Figure 2 shows the result of the layerwise sensitivity analysis for the weights. The original phoneme frame error rate was 27.26% with the LSTM layer size of 512, and that with the LSTM size of 256 was 28.63%. The result indicates that all layers except the input-LSTM1 group shows the almost the same quantization sensitivity and requires only two bits for weight representation. Input-LSTM1 weights group demands at least three quantization bits. In the signal sensitivity analysis, all layers need only three to five quantization bits.

Using the sensitivity analysis results, we construct a fully

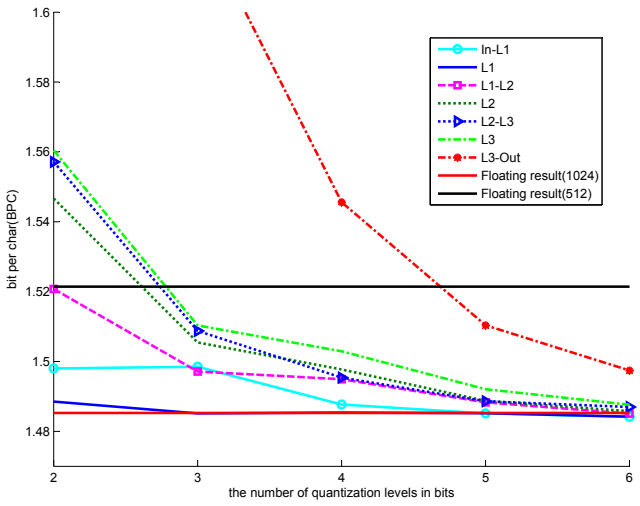


Fig. 3. Layerwise sensitivity analysis results of the weights in the language model example. The red and black horizontal lines indicate the floating-point results for 1024 LSTM size and 512 LSTM size each.

Table 2. Bit per character on the test set with the English Wikipedia language model example using quantized weights and signals. Numbers in the parenthesis indicate the ratio of the weights capacity compared to the floating-point version.

Layerwise quantization bits		BPC	
Weights bits (In-L1, L1, L1-L2, L2, L2-L3, L3, L3-Out)	Signal bits (L1, L2, L3)	Direct	Retrain
2-2-3-4-4-4-6 (10.52%)	6-6-7	3.623	1.546
3-3-4-5-5-5-7 (13.64%)	6-6-7	1.641	1.510
4-4-5-6-6-6-8 (16.75%)	6-6-7	1.517	1.499
2-2-3-4-4-4-6 (10.52%)	7-7-8	3.613	1.545
3-3-4-5-5-5-7 (13.64%)	7-7-8	1.639	1.508
4-4-5-6-6-6-8 (16.75%)	7-7-8	1.517	1.499

quantized LSTM RNN and the results are shown in Table1. The result shows that the phoneme error rate of 27.74% is achieved with only about 10% of the weight capacity needed for the floating-point version.

4.2. Language Model

A character level language model was trained using English Wikipedia dataset which was used in [24]. Each character was put into the neural network input using their own ASCII code values with one-hot encoding, which needs 256 units.

The input layer contains 256 linear units to accommodate real valued inputs. Three hidden LSTM layers have the same size of 1024. The output layer consists of 256 softmax units that correspond to 256 character level one-hot encoding. The network is trained using Fractal with the training parameters of 128 forward steps and 128 backward steps employing 64 streams [21]. The learning rate was decreased from 10^{-6} to 10^{-8} during training. Momentum was 0.9 and adadelta was adopted for weights update. This network needs approximately 22.3 million weights.

Figure 3 shows the layerwise sensitivity analysis results for the weights. The bit per character (BPC) of the floating-point language model with the layer size of 1024 was 1.485. The analysis shows that the most sensitive weights group of the network is the L3-Out layer. Note that the last RNN layer is connected to the softmax layer. The first weights group shows low sensitivity when compared to the phoneme recognition example. This is because one-hot encoding of the ASCII code is used as for the input in this language model. The network has two types of paths, the forward and the recurrent connections. For both paths, the layer that is close to the output shows higher quantization sensitivity. The sensitivity analysis of signals shows the minimum of four or five bits for quantization.

We next try fixed-point optimization of all signals and weights. While the sensitivity analysis quantizes only one group of weights or signals, the fixed-point optimization quantizes all the weights and signals simultaneously to find out the most optimum set of word-lengths. The results are summarized in Table 2. Unlike the phoneme recognition example, this application needs more quantization bits over the results obtained from the sensitivity analysis. A reasonable result was achieved with two more bits for both weights and signals than the sensitivity analysis result. The memory space needed is 16.75% when compared to the floating-point representation.

5. CONCLUDING REMARKS

This work investigates the fixed-point characteristics of RNNs for phoneme recognition and language modeling. The retrain-based fixed-point optimization greatly reduces the word-length of weights and signals. In the phoneme recognition example, most of the weights can be represented in 3 bits, while the language modeling needs 5 or 6 bits for obtaining near floating-point results. By this optimization, the weights capacity needed can be reduced to only 10% or 17% of that required for floating-point implementations. The reduced weights and signals can lead to efficient hardware implementations or higher cache memory hit ratio in software based systems.

6. REFERENCES

[1] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Honza Černockỳ, and Sanjeev Khudanpur, "Extensions of recurrent neural network language model," in *Acoustics, Speech and Signal Processing (ICASSP),*

2011 IEEE International Conference on. IEEE, 2011, pp. 5528–5531.

[2] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt, "Sequential deep learning for human action recognition," in *Human Behavior Understanding*, pp. 29–39. Springer, 2011.

[3] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan, "Show and tell: A neural image caption generator," *arXiv preprint arXiv:1411.4555*, 2014.

[4] Jeffrey L Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[5] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] Herbert Jaeger, "The echo state approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, pp. 34, 2001.

[7] Jordan L Holi and Jenq-Neng Hwang, "Finite precision error analysis of neural network hardware implementations," *Computers, IEEE Transactions on*, vol. 42, no. 3, pp. 281–290, 1993.

[8] Gunhan Dundar and Kenneth Rose, "The effects of quantization on multilayer neural networks.," *IEEE transactions on neural networks, a publication of the IEEE Neural Networks Council*, vol. 6, no. 6, pp. 1446–1451, 1994.

[9] Clément Farabet, Berin Martini, Polina Akselrod, Selçuk Talay, Yann LeCun, and Eugenio Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010, pp. 257–260.

[10] Chuan Zhang Tang and Hon Keung Kwan, "Multilayer feedforward neural networks with single powers-of-two weights," *Signal Processing, IEEE Transactions on*, vol. 41, no. 8, pp. 2724–2727, 1993.

[11] Jonghong Kim, Kyuyeon Hwang, and Wonyong Sung, "X1000 real-time phoneme recognition vlsi using feed-forward deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 7510–7514.

[12] Kyuyeon Hwang and Wonyong Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and -1," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE, 2014, pp. 1–6.

[13] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1131–1135.

[14] Paul J Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[15] Wonyong Sung and Ki-II Kum, "Simulation-based word-length optimization method for fixed-point digital signal processing systems," *Signal Processing, IEEE Transactions on*, vol. 43, no. 12, pp. 3087–3090, 1995.

[16] Rich Caruana Steve Lawrence Lee Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference*. MIT Press, 2001, vol. 13, p. 402.

[17] Michael K Weir, "A method for self-determination of adaptive learning rates in back propagation," *Neural Networks*, vol. 4, no. 3, pp. 371–379, 1991.

[18] Yurii Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o (1/k2)," in *Doklady an SSSR*, 1983, vol. 269, pp. 543–547.

[19] John S Garofolo, Lori F Lamel, William M Fisher, Jonathon G Fiscus, and David S Pallett, "Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1," *NASA STI/Recon Technical Report N*, vol. 93, pp. 27403, 1993.

[20] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6645–6649.

[21] Kyuyeon Hwang and Wonyong Sung, "Single stream parallelization of generalized LSTM-like RNNs on a GPU," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1047–1051.

[22] Robert A Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural networks*, vol. 1, no. 4, pp. 295–307, 1988.

[23] Matthew D Zeiler, "Adadelta: An adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[24] Ilya Sutskever, James Martens, and Geoffrey E Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.