Chapter

Quantized Neural Networks and Neuromorphic Computing for Embedded Systems

Shiya Liu and Yang Yi

Abstract

Deep learning techniques have made great success in areas such as computer vision, speech recognition and natural language processing. Those breakthroughs made by deep learning techniques are changing every aspect of our lives. However, deep learning techniques have not realized their full potential in embedded systems such as mobiles, vehicles etc. because the high performance of deep learning techniques comes at the cost of high computation resource and energy consumption. Therefore, it is very challenging to deploy deep learning models in embedded systems because such systems have very limited computation resources and power constraints. Extensive research on deploying deep learning techniques in embedded systems has been conducted and considerable progress has been made. In this book chapter, we are going to introduce two approaches. The first approach is model compression, which is one of the very popular approaches proposed in recent years. Another approach is neuromorphic computing, which is a novel computing system that mimicks the human brain.

Keywords: machine learning, deep learning, model compression, algorithms, pattern recognition, neuromorphic computing

1. Introduction

Deep learning is a branch of machine learning that is inspired by the biological processes of human brain and it is not a new concept. The reason it was not popular earlier is because there were not enough computational power and data available many years ago. With the development of the semiconductor industry and Internet, the stronger computational power and tremendous data generated by the Internet make the use of deep learning techniques possible [1–10].

Even though deep learning techniques have made great success in many fields, we still have not realized their full potential, especially in embedded systems because such systems do not have enough computation power. In the era of mobile computing, enabling deep learning techniques running on mobile devices is very important and a lot of researchers have been working on this area [4, 6]. Researches have been conducted in two directions. The first direction aims to reduce model size and computation of deep learning models. The second direction is to design new hardware architectures that have much stronger computation power. In this chapter, we are going to introduce two approaches. The first technique is quantization, which is used to reduce the computation and model size of deep learning models. The second technique is neuromorphic computing, which is a new hardware architecture to enhance the computation power.

2. Neural network

Artificial neural network is a computing system that is capable of mimicking the human brain. The purpose of an artificial neural network is to identify patterns in input data and learn an approximate function that maps inputs to outputs. The most basic building units of neural network are neurons, which have inputs, outputs and a processing unit. To better learn complicated patterns in input data, a neural network consists of a huge number of neurons, which are organized into layers of neurons [11, 12]. These layers of neurons are stacked on each other so that the output of a layer is the input of the following layer. A neuron in a layer is connected to multiple neurons in previous layer in order to receive data from those neurons. Data received from neurons in previous layer are multiplied by corresponding weights and the product results are accumulated together to generate an output.

2.1 Single neuron

The most basic building unit of a neural network is neuron. A neuron receives data from multiple neurons from previous layer and each of these data is multiplied by a weight. Then, these weighted data are accumulated together to generate an output. A non-linear function is applied to the output before the output is sent out to other neurons. More details about why we need a non-linear function are presented in Section 2.3. The working mechanism of a single neuron can be expressed as,

$$Out = \sum_i w_i x_i + b \tag{1}$$

where x_i is the i_{th} input, w_i is the weight corresponding to i_{th} input, b is the bias value and Out is the accumulated output.

Figure 1 illustrates a single neuron with three inputs where x_1, x_2 and x_3 are the three inputs and w_1, w_2 and w_3 are weights corresponding to inputs x_1, x_2 and x_3. The output of this neuron can be computed using Eq. (1).

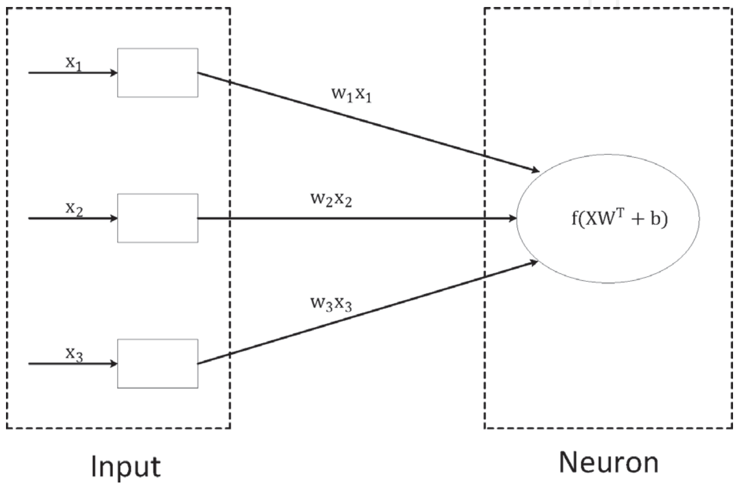

Figure 1.
Single neuron with three inputs.

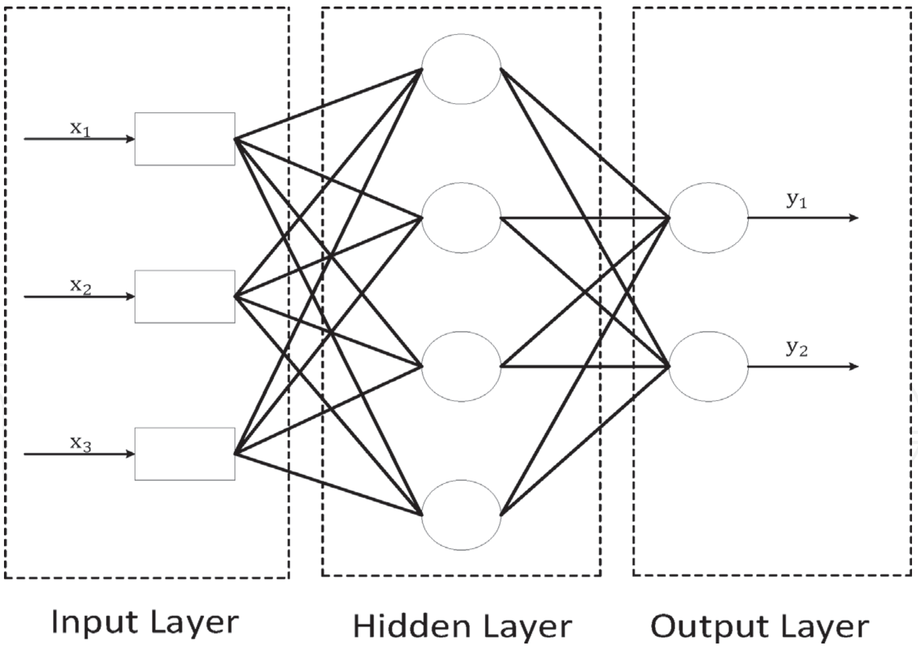

Figure 2.
Neural network with hidden layer.

2.2 Multilayer perceptron

Multilayer perceptron (MLP) is a kind of neural network that has at least one layer of neurons between the input and output layer. Hidden layers are layers between the input and output layer. The reason why multilayer perceptron is introduced is that it makes a neural network much more powerful and is able to learn very complicated patterns in input data. If there is no layer between the input and output layer, the input is transformed to output by a linear transformation function and the neural network can only work on linearly separable data. To enable neural network to handle data that are not linearly separable, we need to have at least one layer between the input and output layer. Meanwhile, non-linear functions are applied to the outputs of hidden layers. Let us take the MLP neural network in **Figure 2** as an example. This neural network has three inputs, two outputs and one hidden layer with four neurons. Lines represent connections between neurons. Each connection has an associated weight and we use weight matrices to represent the connections between the input, hidden and output layer.

The behaviour of the neural network in **Figure 2** can be expressed mathematically using Eq. (2).

$$\begin{aligned} h_1 &= \theta(W_1 \cdot x + b_1) \\ y &= W_2 \cdot h_1 + b_2 \end{aligned} \tag{2}$$

where θ is the non-linear activation function; *symbol* $\cdot$ is the dot product between two matrices; W_1 is the weight matrix between the input and hidden layer; W_2 is the weight matrix between the hidden and output layer; h_1 is the output of the hidden layer; x is the neural network input and y is the neural network output.

Input x is mapped to four neurons by a weight matrix W_1 first and then each neuron is applied a non-linear activation function. h_1 is the output of the hidden layer and this output is multiplied by another weight matrix W_2 to obtain final result y.

2.3 Non-linear activation function

Non-linear activation function [13, 14] is very important for MLP. Without a non-linear activation function, neural network does a linear transformation from input to output no matter how many hidden layers exist between the input and output layer. It is because the linear transformation of a linear transformation is still a linear transformation and thus any number of hidden layers can be deducted to a single linear transformation. Let us take the MLP neural network in **Figure 2** as an example. Without non-linear activation, the neural network can be expressed mathematically as,

$$\begin{gathered} h_1 = W_1 \cdot x + b_1 \\ y = W_2 \cdot h_1 + b_2 \end{gathered} \tag{3}$$

Substituting h_1 with $W_1 \cdot x + b_1$, we can have,

$$\begin{gathered} y = W_2 \cdot (W_1 \cdot x + b_1) + b_2 \\ => y = W_2 \cdot W_1 \cdot x + W_2 \cdot b_1 + b_2 \end{gathered} \tag{4}$$

Assume, $W = W_1 \cdot W_2$ and $b = W_2 \cdot b_1 + b_2$,

$$y = W \cdot x + b \tag{5}$$

Therefore, the MLP neural network in **Figure 2** can be expressed mathematically as a single linear transformation from input to output.

2.4 Types of hidden layers

In MLP neural networks [15], there are hidden layers between the input and output layer and these hidden layers play a very important role in performances of MLP neural networks. There are many different types of hidden layers such as convolutional layers, fully-connected layers, pooling layers and so on. In this section, we are going to present more details about convolutional layers and fully-connected layers.

2.4.1 Fully-connected layers

In fully-connected layers, each neuron is connected to all neurons in previous layers and each connection has an associated weight. Each output of a neuron from previous layers is multiplied by a weight associated with the connection. Then, the product result is accumulated together.

Let us take the hidden layer of MLP neural network in **Figure 2** as an example. The hidden layer in **Figure 2** is a fully-connected layer. Each neuron in the hidden layer connects all three inputs in the input layer and generates one output. The weight matrix is represented in Eq. (6). In the weight matrix, each row represents the weights of a neuron and thus the matrix size is 4 × 3 since there are four outputs and three inputs.

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \tag{6}$$

The input can be represented as matrix that has the size of 3 × 1.

$$X = [x_1 \;\; x_2 \;\; x_3] \tag{7}$$

Mathematically, fully-connected layer is computed as a matrix multiplication,

$$Out = W \cdot X \tag{8}$$

2.4.2 Convolutional layer

The convolutional layer is a layer used in many deep learning applications, especially in computer vision [1, 2, 16–18]. In computer vision, processing and understanding an image is a major task. An image has three dimensions, which are width, height and channel. Meanwhile, an image is highly structured and has strong spatial dependency [11].

The convolution layer has a group of kernels and each of these kernels has three dimensions, which are width, height and channel. The width and height of a kernel are hyper-parameters defined by designers. The size of a channel is equal to the channel size of previous layer. Unlike a fully-connected layer, each neuron in a convolutional layer is only connected to a small spatial region of neurons but all channels in the previous layer. The size of this spatial region depends on the width and height of each kernel. Each kernel slides over the whole image with a specific stride to extract features such as edge feature from the image. Therefore, each kernel extracts a specific feature we want to obtain from each local region.

Let us use **Figure 3** above as an example to demonstrate how convolution layer works. In **Figure 3**, the image only has one channel with size $6 \times 6 \times 1$ and there is one kernel with size $3 \times 3 \times 1$. Assume the weight of this kernel is

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \tag{9}$$

Then, the weight matrix W is multiplied by the pixels of a small region in the image element-wise and then these product results are accumulated together. Assume we are applying our kernel on the yellow region of the image in **Figure 3**. Then, we can get the output Out using Eq. (10),

$$\begin{aligned} Out &= W \odot X \\ Out &= 1 \times 0 + 0 \times 1 + 1 \times 0 + \\ &\quad 0 \times 0 + 1 \times 1 + 0 \times 1 + \\ &\quad 1 \times 1 + 0 \times 1 + 1 \times 1 = 3 \end{aligned} \tag{10}$$

where $\odot$ represents element-wise product between two matrixes.

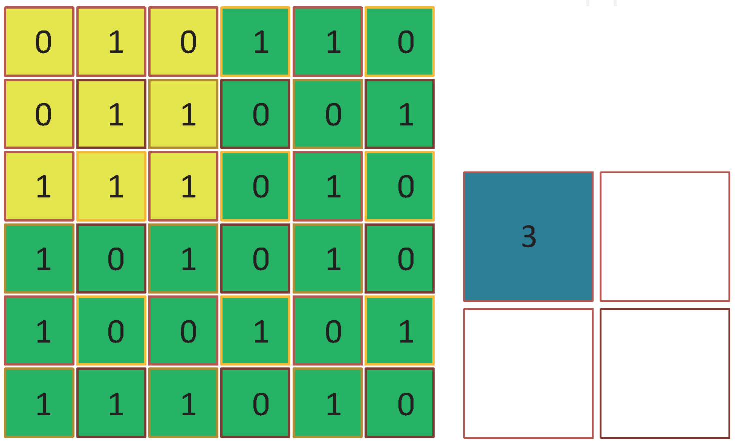

Figure 3.
Image and convoluted output.

Convolution layer has a couple of advantages compared to other layers when dealing with images. These features make the convolution layer very popular in the area of computer vision. First of all, convolutional layers need much less weights compared to fully-connected layers. In a fully-connected layer, a neuron is connected to all neurons in previous layers. If the dimension of previous layer is very large, the number of weights required by the fully-connected layer is very large since the total number of weights is equal to the number of neurons in previous layer times the number of neurons in fully-connected layer. Secondly, the convolution layer focuses on local spatial regions instead of the whole image and many applications benefit from this characteristic. For example, when dealing with object detection in an image, we only need to focus on regions where the object appears and other regions such as background are not needed when we are trying to detect the object in an image. Thirdly, the convolution layer is translation invariant. It means that the responses of a kernel to an object are the same regardless the location of the object in an image.

3. Model compression

In the era of mobile computing, enabling deep learning techniques running on mobile devices is very important. In general, large and complicated models tend to have high performance, but it increases the computation requirement dramatically. Embedded systems do not have sufficient computation and memory resource to support the model complexity of a high-performance deep learning model. Therefore, deploying deep learning models in embedded systems without sacrificing much performance has been a hot research topic [6, 19–22].

3.1 Model quantization

Deep learning models use floating-point arithmetic in both training and inference phases. Floating-point arithmetic needs many computation resources, which is one of the reasons why deploying deep learning models in embedded systems is difficult. To address this issue, researchers have proposed many approaches [4–6, 23–29] to replace the floating-point arithmetic during the inference phase. Low bit-precision arithmetic [30–32] is one of those approaches.

In the low-bit arithmetic approach, floating-point numbers and floating-point arithmetic are still used in the training phase. After training is done, model weights and activation layers are quantized using low-bit integer numbers such as 8-bit or 16-bit integers. During inference, integer arithmetic is used instead of floating-point arithmetic and thus the computation resource requirement is reduced dramatically.

3.1.1 Quantization scheme

In Ref. [4], the authors proposed a quantization scheme that is successfully adopted in TensorFlow [33]. During the inference phase, the proposed quantization scheme uses integer-only arithmetic while floating-point arithmetic is still used in the training phase. Since this approach uses different data type in training and inference phases, creating a one-to-one mapping between floating-point number and integer number is needed. The authors use Eq. (11) [4] to describe the mapping between a floating-point number and integer number.

$$r = S(q - Z) \tag{11}$$

In this equation, S and Z are quantization parameters, which are constant for each layer. There is only one set of quantization parameters associated for each activation layer and weights layer.

The constant S is a floating-point number, which is a scale constant to represent the size of each quantization level. For example, assume we are going to quantize a floating point r in a layer to an 8-bit integer number. To calculate the scale constant S, we obtain the maximum and minimum floating-point number of the layer first. We use r_{max} and r_{min} to represent the maximum and minimum floating-point number respectively. Since we are using an 8-bit integer number, there are $n = 2^8 = 256$ quantization levels. Then, the constant scale S can be computed as,

$$S = \frac{r_{max} - r_{min}}{n - 1} \tag{12}$$

In terms of Z, it represents real number 0 using quantized integer. The reason why we need an exact number to represent number 0 is that number 0 is wildly used in deep learning such as zero-padding in convolutional neural network. Representing number 0 exactly improves the performance of deep learning models. The number 0 can be calculated using Eq. (13) [4].

$$Z = q_{min} - rounding\left(\frac{r_{min}}{S}\right) \tag{13}$$

In Eq. (13), q_{min} is the minimum quantization level of our quantized integer. For example, if we use an unsigned 8-bit integer, q_{min} is equal to 0. r_{min} and S are two floating-point numbers representing the minimum values of a layer and the scale constant of this layer, respectively. Because Z is an integer, we need to round $\frac{r_{min}}{S}$ to the nearest integer. However, Eq. (13) only works for the case where r_{min} is smaller than 0 and r_{max} is larger than 0. To make it work for all cases, we use the following approaches to handle those cases. If r_{min} is larger than 0, we set r_{min} to 0 and calculate scale constant S using the new r_{min}. If r_{max} is smaller than 0, then we set r_{max} to 0 and calculate scale constant S using the new r_{max}. After we obtain the scale constant S, then we can calculate the zero-point.

3.1.2 Integer-only multiplication

After floating-point numbers are quantized into integers using the quantization scheme described in Eq. (11), the authors in [4] describe their approaches of how to compute multiplication between two floating-point numbers using quantized integers.

Assume we are going to compute the multiplication between two floating-point numbers r_1 and r_2. The multiplication result is stored in floating-point number r_3. We first quantize the floating-point numbers r_1, r_2 and r_3 into quantized integer numbers q_1, q_2 and q_3 respectively using Eq. (11). We have scale constants S_1, S_2 and S_3 corresponding to r_1, r_2 and r_3. Meanwhile, we have quantized zero-points Z_1, Z_2 and Z_3 corresponding to r_1, r_2 and r_3.

$$\begin{aligned} r_1 &= S_1(q_1 - Z_1) \\ r_2 &= S_2(q_2 - Z_2) \\ r_3 &= S_3(q_3 - Z_3) \end{aligned} \tag{14}$$

Then, we want to compute the product between r_1 and r_2. We have,

$$\begin{aligned} r_3 &= r_1 r_2 \\ &=> S_3(q_3 - Z_3) = S_1(q_1 - Z_1)S_2(q_2 - Z_2) \\ &=> q_3 = Z_3 + \frac{S_1 S_2}{S_3}(q_1 - Z_1)(q_2 - Z_2) \end{aligned} \tag{15}$$

In Eq. (15) [4], every arithmetic is between two integers except $\frac{S_1 S_2}{S_3}$, which is a floating-point number. To make the whole computation integer-only, the authors in [4] proposed an approach to quantize the floating-point number $\frac{S_1 S_2}{S_3}$.

Firstly, the authors found that $M = \frac{S_1 S_2}{S_3}$ is always in the interval $(0, 1)$ and used Eq. (16) [4] to describe the relationship between M and M_0,

$$M = 2^{-n} M_0 \tag{16}$$

In Eq. (16), the authors in [4] set M_0 to a number between 0.5 and 1. n is a positive integer number. Using Eq. (16), the authors in [4] make M to be a fixed-point multiplier. If a 16-bit integer is used in the multiplication, M_0 can be represented as a 16-bit integer, which is $2^{15}M_0$ and bit-shift operation is used to compute the multiplication of $2^{-\mathbf{n}}$. Then, the whole expression can be computed using integer-only arithmetic.

3.2 Quantization-aware training

There are two common approaches to train quantized neural networks. The first approach is to train neural networks using floating-point numbers and then quantize weights and activation layers after training. However, this approach might not work for some models. In [4], the authors found that this approach does not work for small models because small models tend to have significant accuracy drops. The authors in [4] listed two reasons for accuracy drops. The first one is that weight distribution is large for different output channels. The large weight distribution makes channels with small weights range have large quantization errors. The second reason is that outlier weight values cause the quantization of weights much less accurate.

Because of the reasons mentioned above, the authors in [4] proposed a training approach that includes the quantization effects in the forward pass of training. Backward pass of training works as traditional training method and floating-point numbers are still used for weight and activation layers. During forward pass of training, the authors in [4] use Eq. (17) to quantize each layer and these equations are applied to each floating-point number element-wise.

$$\begin{aligned} &Clamp(r; a, b) := min\,(max\,(x, a), b) \\ &s(a, b, n) := \frac{b - a}{n - 1} \\ &q(r; a, b, n) := round\left(\frac{clamp(r; a, b) - a}{s(a, b, n)}\right)s(a, b, n) + a \end{aligned} \tag{17}$$

where r is a floating-point number; a, b are the maximum and minimum values of a layer and n is quantization level. For example, n = $2^8 = 256$ quantization levels if a 8-bit integer is used.

The function *round* is to rounding the number to its nearest integer.

For weights, the authors' proposed to set a and b to the minimum and maximum floating-point number of a weight layer respectively. In terms of activation layer, the authors used exponential moving averages to track the minimum and maximum floating-point numbers of an activation layer. After we have the range parameter a and b, we can compute other parameters easily. This approach has been implemented in Tensorflow [33, 34].

3.3 Comparison between different quantization approaches

Binarized neural network: Binarized neural network is an aggressive quantization approach that quantizes each weight to a binary value. In binary neural networks, dot product between two matrices can be completed by bit count operation, which is an operation to count the number of 1 s in a vector. The binary neural network in [5] achieves 32× reduction in model size and 58× speed up without losing much accuracy compared to equivalent neural network using single-precision values.

DoReFa-Net: DoReFa-Net is one of the most popular quantization approaches. This quantization approach not only applies quantization on weight and activation layers, but also on gradients. Through applying quantization on gradients, the training speed could be increased significantly. In [25], the proposed DoReFa-Net achieved 46.1% top-1 accuracy on ImageNet using 2-bit activations, 1-bit weights and trained with 6-bit gradients.

Log-based quantization: In [35], the authors proposed a multiplication-free hardware accelerator for deep neural networks. The proposed approach quantizes each weight to the nearest powers of two using logarithmic and rounding functions. In terms of the activation layer, the authors quantize each output to an 8-bit integer. By quantizing each weight to the nearest powers of two, multiplication between two integers could be replaced by bit-shift operations, which could reduce the resource utilization significantly. In [35], the authors demonstrate that the proposed quantization approach achieves almost the same accuracy as floating-point version but reduces energy consumption significantly.

3.4 Progress in model compression

Besides quantization approaches, many other model compression approaches are proposed. Pruning approach is one of the most popular approaches for model compression [6]. Pruning approach reduces the size of weights by removing some weights if these weights meet certain criteria. Besides weights, pruning approach could be also applied to activations and biases.

Knowledge distillation is another very popular approach for model compression [36]. There are two models, namely teacher model and student model, in knowledge distillation approach. Teacher model is a trained model. In addition, it has much larger model size than the student model. The main idea of knowledge distillation is to transfer the knowledge of teacher model to student model so that student model could have comparable performance to that of teacher model.

4. Neuromorphic computing

Neuromorphic computing [10, 37–50] is an emerging computing system that mimics the architecture of human brain. Carver Mead proposed the concept of neuromorphic computing in the late 1980s [43, 51–53]. Neuromorphic computing systems exploit spiking neural network to process information. Compared to conventional neural networks, spiking neural networks are more analogous to human brains and consume much less power. Recently, neuromorphic computing has been successfully applied to many applications [54–59].

4.1 Spiking neurons

The basic building block of spiking neuron networks is spiking neurons. The working mechanism of spiking neuron is different from that of neurons introduced in Section 2.1. Spiking neurons exchange information through electrical pulses, which are also called spikes. Spikes are discrete, time-dependent data and are represented as binary signals. In [8], the authors introduced several properties of spiking neurons. In the first place, spiking neurons receive information from many inputs and generate one output. Secondly, generating a spike depends on the amount of excitatory and inhibitory inputs. Thirdly, a spiking neuron's received spikes from other spiking neurons are integrated over time and will fire spikes if the integrated result is over a certain threshold.

4.1.1 Neuron models

There are several commonly used spiking neuron models such as leaky integrate-and-fire, Hodgkin-Huxley [60] and Fitzhugh Nagumo neuron models.

Leaky integrate-and-fire: According to [61], leaky integrate-and-fire neuron model is the simplest model to implement and the operation of leaky integrate-and-fire neuron can be completed using few floating-point operations such as additions and multiplications. However, there is no phasic spiking in leaky integrate-and-fire model since the model only has one variable. Meanwhile, spiking latencies do not exist in spikes because the threshold is fixed. The behaviour of leaky integrate-and-fire neuron model can be expressed using Eq. (18) [61]. If voltage V reaches a certain threshold level V_{th}, then a spike is fired and voltage V is reset to c.

$$\frac{dV}{dt} = I + a - bV, if V \geq V_{th}, then V reset to c \tag{18}$$

In Eq. (18), a, b, c and V_{th} are the parameters.

FitzHugh-Nagumo: FitzHugh-Nagumo neuron model [61] is more complicated compared to the leaky integrate-and-fire model and needs slightly more floating-point operations. The model has multiple variables and thus it has phasic spiking. Meanwhile, spikes of FitzHugh-Nagumo neuron model have spiking latencies because the threshold is not fixed. The behaviour of the FitzHugh-Nagumo neuron model can be expressed using Eq. (19) [61].

$$\begin{aligned} \frac{\mathrm{dV}}{\mathrm{dt}} &= a + bV + cV^2 + dV^3 - u \\ \frac{\mathrm{du}}{\mathrm{dt}} &= \varepsilon(eV - u) \end{aligned} \tag{19}$$

Hodgkin-Huxley: Hodgkin-Huxley [60] is a much more complicated neuron model compared to leaky integrate-and-fire and Fitzhugh-Nagumo neuron models. It is described by multiple equations and many parameters. In [61], the authors state that the parameters of the Hodgkin-Huxley neuron model are biophysically meaningful. More importantly, the Hodgkin-Huxley neuron model is very helpful for researchers to investigate single-cell dynamics. However, this model is hard to implement since it requires over 100 floating-point operations. More details about this model can be found in [60].

4.1.2 Leaky integrate-and-fire spiking neuron model

In this section, we are going to present more details about leaky integrate-and-fire spiking neuron models. The behaviour of integrate-and-fire spiking neuron model can be described using Eq. (18). If voltage V is above a certain voltage threshold V_{th}, it will fire spikes and voltage V is reset to 0. The behaviour of leaky integrate-and-fire model can be described by the circuit shown in **Figure 4**.

4.2 Neuromorphic computing for embedded systems

As we stated above, embedded systems have very limited computation resources and power constraints. Compared to conventional neural networks, spiking neural networks are more analogous to human brains and consume much less power. Because of these features, neuromorphic computing is suitable for embedded systems. A lot of researchers [62–66] have implemented neuromorphic computing in embedded systems such as FPGA. In [63], the authors implement liquid state machine on FPGA for speech recognition. The overall architecture achieves 88× speed up compared to CPU implementation. Meanwhile, the proposed approach reduces 30% power consumption.

4.3 Hardware implementation of spiking neural networks

A lot of researchers have been working on the hardware implementation of spiking neural networks and many neuromorphic chips have been developed. For example, in Stanford University, Neurogrid [67] and TrueNorth [68] have been developed by IBM.

Neurogrid: Neurogrid [67] is a mixed-signal hardware system for simulating biological brains. This system exploits analog circuits to implement all circuits except axonal arbors to improve energy efficiency and axonal arbors are implemented using digital circuits. The whole system consists of 16 Neurocores and each Neurocore has a 256 × 256 silicon-neuron array, a receiver, a transmitter and two RAMs. Neurogrid is able to simulate a million neurons by only consuming few watts.

TrueNorth: TrueNorth [68] is a brain-inspired neurosynaptic processor and uses non-von Neumann architecture. The whole system has 4096 cores, 1 million digital neurons and 256 million synapses. TruhNorth achieves 58 giga-synaptic operations per second (GSOPS) and 400 GSOPS per watt. More importantly, the authors have successfully implemented several applications such as object

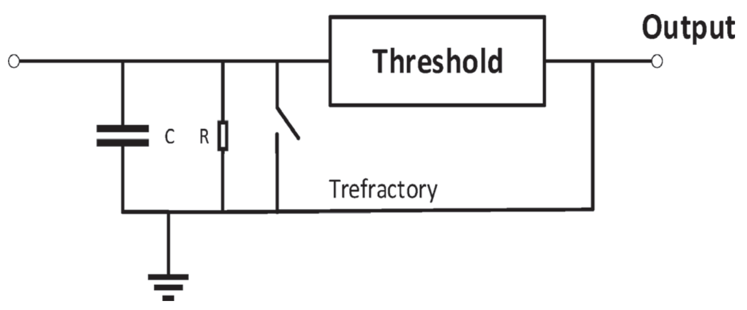

Figure 4.
Leaky integrate-and-fire model.

recognition on TrueNorth, and it has much lower power consumption compared to conventional processors.

4.4 Recent progress in neuromorphic computing

Spiking neural networks exploit spikes to represent information and thus effective and efficient approaches of representing information using spikes are very important. In spiking neural networks, input is encoded into spikes and each spike is represented as a single binary bit. There are two types of encoding approaches [7, 10, 45, 47, 69–72]. The first type of encoding approach is rate encoding. In rate encoding, input is encoded as the rate of spikes over an encoding window [7, 69]. Temporal encoding is also an encoding approach. Inter-spike interval encoding is a method of doing temporal encoding. In inter-spike encoding, the information is encoded by the time difference between two adjacent spikes [7, 69, 70].

Researchers have successfully implemented neural encoder using hardware [10, 62, 69]. In [62], the authors proposed a spike time-dependent encoder on FPGA. In [69], the authors implemented an inter-spike interval-based encoder for neuromorphic processors using analog integrated circuits. The proposed analog implementation of inter-spike interval encoder gets rid of ADCs and Op-amp and thus consumes less power.

In recent years, an increasing number of researchers have started to implement neuromorphic computing using analog integrated circuits [46, 47, 49, 50, 71, 73–79]. Compared to digital implementation, analog implementation of neuromorphic computing is more energy efficient. Meanwhile, analog implementation consumes less chip area.

Three-dimensional integrated circuits (3D IC) technique [80–82] is an emerging technique to improve the performance of integrated circuits. Compared to conventional fabrication techniques, three-dimensional integrated circuits technique consumes less power and uses small footprint. Recently, 3D IC technique has been applied to neuromorphic computing [79, 83–93]. Through the 3D IC technique, power consumption and chip area are reduced dramatically [88].

5. Conclusion

In the era of mobile computing and internet of things, embedded systems are everywhere. It can be found in consumer electronics, automobile, industrial and many other applications. Without embedded systems, our daily life would become extremely inconvenient. Deep learning is a technology, which is as important as embedded systems to our daily life. In recent years, deep learning is becoming a fundamental technology that impacts every aspect of our daily life. Therefore, deploying deep learning in embedded systems draws a lot of attention nowadays. Researchers have been conducting researches in many directions. For example, researchers are designing new layers and applying quantization techniques to reduce computation. Meanwhile, new architectures such neuromorphic computing are proposed. Through these techniques, many deep learning models are implemented in embedded systems successfully.

Author details

Shiya Liu* and Yang Yi
The Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University (Virginia Tech), Virginia, USA

*Address all correspondence to: shiyal@vt.edu

References

[1] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. pp. 770-778

[2] Ren S, He K, Girshick R, Sun J. Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems. 2015. pp. 91-99

[3] Graves A, Mohamed A-R, Hinton G. Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing; IEEE. 2013. pp. 6645-6649

[4] Jacob B et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018. pp. 2704-2713

[5] Rastegari M, Ordonez V, Redmon J, Farhadi A. Xnor-net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision; Springer. 2016. pp. 525-542

[6] Han S, Mao H, Dally WJ. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. Computer Vision and Pattern Recognition. 2015

[7] Bouvier M et al. Spiking neural networks hardware implementations and challenges: A survey. ACM Journal on Emerging Technologies in Computing Systems (JETC). 2019;**15**(2):22

[8] Ponulak F, Kasiński A. Introduction to spiking neural networks: Information processing, learning and applications. Acta Neurobiologiae Experimentalis. 2011;**71**(4):409-433

[9] Rathi N, Panda P, Roy K. STDP-based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition. Neural and Evolutionary Computing. 2018;**38**(4):668-677

[10] Zhao C et al. Energy efficient spiking temporal encoder design for neuromorphic computing systems. IEEE Transactions on Multi-Scale Computing Systems. 2016;**2**(4):265-276

[11] Zhang A, et al. Dive into deep learning; Unpublished draft. 2019. p. 319

[12] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016

[13] Maas AL, Hannun AY, Ng AY. Rectifier nonlinearities improve neural network acoustic models. Proceedings ICML. 2013;**30**(1):3

[14] Leshno M, Lin VY, Pinkus A, Schocken SJ. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. 1993;**6**(6):861-867

[15] Pinkus A. Approximation theory of the MLP model in neural networks. 1999;**8**:143-195

[16] Qi CR, Su H, Mo K, Guibas LJ. Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. pp. 652-660

[17] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. 2012. pp. 1097-1105

[18] Ji S, Xu W, Yang M, Yu K. 3D convolutional neural networks for human action recognition. IEEE

Transactions on Pattern Analysis and Machine Intelligence. 2012;**35**(1): 221-231

[19] Han S, Pool J, Tran J, Dally W. Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems. 2015. pp. 1135-1143

[20] Desoli G et al. 14.1 a 2.9 tops/w deep convolutional neural network soc in fd-soi 28 nm for intelligent embedded systems. In: 2017 IEEE International Solid-State Circuits Conference (ISSCC); IEEE. 2017. pp. 238-239

[21] Howard AG et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017. arXiv preprint arXiv:1704.04861

[22] Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018. pp. 4510-4520

[23] Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y. Quantized neural networks: Training neural networks with low precision weights and activations. 2017;**18**(1):6869-6898

[24] Wu J, Leng C, Wang Y, Hu Q, Cheng J. Quantized convolutional neural networks for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. pp. 4820-4828

[25] Zhou S, Wu Y, Ni Z, Zhou X, Wen H, Zou Y. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. 2016. arXiv preprint arXiv:1606.06160

[26] Zhu C, et al. Trained ternary quantization. 2016. arXiv preprint arXiv:1612.01064

[27] Lin X, Zhao C, Pan W. Towards accurate binary convolutional neural network. In: Advances in Neural Information Processing Systems. 2017. pp. 345-353

[28] Han S, Mao H, Dally WJ. A deep neural network compression pipeline: Pruning, quantization, huffman encoding. 2015;**10**

[29] Zhang X, Zhou X, Lin M, Sun J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018. pp. 6848-6856

[30] Lin D, Talathi S, Annapureddy S. Fixed point quantization of deep convolutional networks. In: International Conference on Machine Learning. 2016. pp. 2849-2858

[31] Anwar S, Hwang K, Sung W. Fixed point optimization of deep convolutional neural networks for object recognition. In: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP); IEEE. 2015. pp. 1131-1135

[32] Zhou A, Yao A, Guo Y, Xu L, Chen YJ. Incremental network quantization: Towards lossless cnns with low-precision weights. 2017. arXiv preprint arXiv:1702.03044

[33] Abadi M et al. Tensorflow: A system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). 2016. pp. 265-283

[34] Abadi M et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 2016. arXiv preprint arXiv:1603.04467

[35] Tann H, Hashemi S, Bahar RI, Reda S. Hardware-software codesign of accurate, multiplier-free deep neural

networks. In: 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC); IEEE. 2017. pp. 1-6

[36] Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network. Machine Learning. 2015

[37] Burr GW et al. Neuromorphic computing using non-volatile memory. Advances in Physics: X. 2017;**2**(1):89-124

[38] Furber S. Large-scale neuromorphic computing systems. Journal of Neural Engineering. 2016;**13**(5):051001

[39] Kim D, Kung J, Chai S, Yalamanchili S, Mukhopadhyay S. Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory. In: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA); IEEE. 2016. pp. 380-392

[40] Liu X et al. RENO: A high-efficient reconfigurable neuromorphic computing accelerator design. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC); IEEE. 2015. pp. 1-6

[41] Monroe D. Neuromorphic computing gets ready for the (really) big time. Communications of the ACM. 2014;**57**(6):13-15. DOI: 10.1145/2601069

[42] Schuman CD et al. A survey of neuromorphic computing and neural networks in hardware. 2017. arXiv preprint arXiv:1705.06963

[43] Mead C, Ismail M. Analog VLSI Implementation of Neural Systems. Vol. 80. Springer Science & Business Media; 2012

[44] Calimera A, Macii E, Poncino M. The human brain project and neuromorphic computing. Functional Neurology. 2013;**28**(3):191

[45] Zhao C, et al. Spike-time-dependent encoding for neuromorphic processors. ACM Journal on Emerging Technologies in Computing Systems (JETC). 2015;**12**(3):23

[46] Zhao C, Danesh W, Wysocki BT, Yi Y. Neuromorphic encoding system design with chaos based CMOS analog neuron. In: 2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA); IEEE. 2015. pp. 1-6

[47] Zhao C, Li J, Liu L, Koutha LS, Liu J, Yi Y. Novel spike based reservoir node design with high performance spike delay loop. In: Proceedings of the 3rd ACM International Conference on Nanoscale Computing and Communication; ACM. 2016. p. 14

[48] Mosleh S, Sahin C, Liu L, Zheng R, Yi Y. An energy efficient decoding scheme for nonlinear MIMO-OFDM network using reservoir computing. In: 2016 International Joint Conference on Neural Networks (IJCNN); IEEE. 2016. pp. 1166-1173

[49] Bai K, Yi Y. DFR: An energy-efficient analog delay feedback reservoir computing system for brain-inspired computing. ACM Journal on Emerging Technologies in Computing Systems (JETC). 2018;**14**(4):1-22

[50] Bai K, Li J, Hamedani K, Yi Y. Enabling an new era of brain-inspired computing: Energy-efficient spiking neural network with ring topology. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC); IEEE. 2018. pp. 1-6

[51] Mead C. Neuromorphic electronic systems. Proceedings of the IEEE. 1990;**78**(10):1629-1636

[52] Douglas R, Mahowald M, Mead C. Neuromorphic analogue VLSI. Review of Neuroscience. 1995;**18**(1):255-281

[53] Mead C. Analog VLSI and neutral systems. NASA STI/Recon Technical Report A. 1989;**90**

[54] Mosleh S, Liu L, Sahin C, Zheng YR, Yi Y. Brain-inspired wireless communications: Where reservoir computing meets MIMO-OFDM. IEEE Transactions on Neural Networks and Learning Systems. 2017;**99**:1-15

[55] Hamedani K, Liu L, Atat R, Wu J, Yi Y. Reservoir computing meets smart grids: Attack detection using delayed feedback networks. IEEE Transactions on Industrial Informatics. 2017;**14**(2): 734-743

[56] Danesh W, Zhao C, Wysocki BT, Medley MJ, Thawdar NN, Yi Y. Channel estimation in wireless OFDM systems using reservoir computing. In: 2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA); IEEE. 2015. pp. 1-5

[57] Li J, Liu L, Zhao C, Hamedani K, Atat R, Yi Y. Enabling sustainable cyber physical security systems through neuromorphic computing. IEEE Transactions on Sustainable Computing. 2017;**3**(2):112-125

[58] Shafin R et al. Realizing green symbol detection via reservoir computing: An energy-efficiency perspective. In: 2018 IEEE International Conference on Communications (ICC); IEEE. 2018. pp. 1-6

[59] Yi Y. Neuron Design in Neuromorphic Computing Systems and Its Application in Wireless Communications. Lawrence: The University of Kansas Center for Research, Inc.; 2017

[60] Hodgkin AL, Huxley AF. A quantitative description of membrane current and its application to conduction and excitation in nerve. The Journal of Physiology. 1952;**117**(4):500-544

[61] Izhikevich EM. Which model to use for cortical spiking neurons? 2004; **15**(5):1063-1070

[62] Yi Y et al. FPGA based spike-time dependent encoder and reservoir design in neuromorphic computing processors. Microprocessors and Microsystems. 2016;**46**:175-183

[63] Wang Q, Li Y, Li P. Liquid state machine based pattern recognition on FPGA with firing-activity dependent power gating and approximate computing. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS); IEEE. 2016. pp. 361-364

[64] Gomar S, Ahmadi A. Digital multiplierless implementation of biological adaptive-exponential neuron model. IEEE Transactions on Circuits and Systems I: Regular Papers. 2013; **61**(4):1206-1219

[65] Rostro-Gonzalez H, Cessac B, Girau B, Torres-Huitzil C. The role of the asymptotic dynamics in the design of FPGA-based hardware implementations of gIF-type neural networks. Journal of Physiology-Paris. 2011;**105**(1–3):91-97

[66] Neil D, Liu S-C. Minitaur, an event-driven FPGA-based spiking network accelerator. 2014;**22**(12):2621-2628

[67] Benjamin BV et al. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. 2014; **102**(5):699-716

[68] Akopyan F et al. Truenorth: Design and tool flow of a 65 MW 1 million neuron programmable neurosynaptic chip. 2015;**34**(10):1537-1557

[69] Zhao C et al. Interspike-interval-based analog spike-time-dependent encoder for neuromorphic processors. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2017;**25**(8): 2193-2205

[70] Zhao C, Li J, Yi Y. Making neural encoding robust and energy efficient:

an advanced analog temporal encoder for brain-inspired computing systems. In: Proceedings of the 35th International Conference on Computer-Aided Design; ACM. 2016. p. 115

[71] Li J, Zhao C, Hamedani K, Yi Y. Analog hardware implementation of spike-based delayed feedback reservoir computing system. In: 2017 International Joint Conference on Neural Networks (IJCNN); IEEE. 2017. pp. 3439-3446

[72] Zhao C, Li J, An H, Yi Y. Energy efficient analog spiking temporal encoder with verification and recovery scheme for neuromorphic computing systems. In: 2017 18th International Symposium on Quality Electronic Design (ISQED); IEEE. 2017. pp. 138-143

[73] Jiang H et al. Cyclical sensing integrate-and-fire circuit for memristor array based neuromorphic computing. In: 2016 IEEE International Symposium on Circuits and Systems (ISCAS); IEEE. 2016. pp. 930-933

[74] Li J, Bai K, Liu L, Yi Y. A deep learning based approach for analog hardware implementation of delayed feedback reservoir computing system. In: 2018 19th International Symposium on Quality Electronic Design (ISQED); IEEE. 2018. pp. 308-313

[75] Bai K, An Q, Yi Y. Deep-DFR: A memristive deep delayed feedback reservoir computing system with hybrid neural network topology. In: Proceedings of the 56th Annual Design Automation Conference; 2019. ACM. 2019. p. 54

[76] Bai K, Bradley YY. A path to energy-efficient spiking delayed feedback reservoir computing system for brain-inspired neuromorphic processors. In: 2018 19th International Symposium on Quality Electronic Design (ISQED); IEEE. 2018. pp. 322-328

[77] Yi Y. Analog Integrated Circuit Design for Spike Time Dependent Encoder and Reservoir in Reservoir Computing Processors. Lawrence, United States: University of Kansas Center for Research, Inc.; 2018

[78] Zhao C, Hamedani K, Li J, Yi Y. Analog spike-timing-dependent resistive crossbar design for brain inspired computing. IEEE Journal on Emerging and Selected Topics in Circuits, and Systems. 2017;**8**(1):38-50

[79] Ehsan MA, An H, Zhou Z, Yi Y. Design challenges and methodologies in 3D integration for neuromorphic computing systems. In: 2016 17th International Symposium on Quality Electronic Design (ISQED); IEEE. 2016. pp. 24-28

[80] Garrou P, Bower C, Ramm P. Handbook of 3D Integration, Volume 1: Technology and Applications of 3D Integrated Circuits. John Wiley & Sons; 2011

[81] Knickerbocker JU et al. 3D silicon integration. In: 2008 58th Electronic Components and Technology Conference; IEEE. 2008. pp. 538-543

[82] Topol AW et al. Three-dimensional integrated circuits. IBM Journal of Research and Development. 2006;**50** (4.5):491-506

[83] An H, Zhou Z, Yi Y. 3D memristor-based adjustable deep recurrent neural network with programmable attention mechanism. In: Proceedings of the Neuromorphic Computing Symposium; ACM. 2017. p. 11

[84] Ehsan MA, An H, Zhou Z, Yi Y. Adaptation of enhanced TSV capacitance as membrane property in 3D brain-inspired computing system. In: 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC); IEEE. 2017. pp. 1-6

[85] An H, Ehsan MA, Zhou Z, Yi Y. Electrical modeling and analysis of 3D neuromorphic IC with monolithic inter-tier vias. In: 2016 IEEE 25th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS); IEEE. 2016. pp. 87-90

[86] An H, Ehsan MA, Zhou Z, Yi Y. Electrical modeling and analysis of 3D synaptic array using vertical RRAM structure. In: 2017 18th International Symposium on Quality Electronic Design (ISQED); IEEE. 2017. pp. 1-6

[87] Ehsan MA, Zhou Z, Yi Y. Modeling and analysis of neuronal membrane electrical activities in 3d neuromorphic computing system. In: 2017 IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity (EMCSI); IEEE. 2017. pp. 745-750

[88] An H, Ehsan MA, Zhou Z, Shen F, Yi YJI. Monolithic 3D neuromorphic computing system with hybrid CMOS and memristor-based synapses and neurons. Integration. 2019;**65**:273-281

[89] Ehsan MA, Zhou Z, Yi Y. Neuromorphic 3D integrated circuit: A hybrid, reliable and energy efficient approach for next generation computing. In: Proceedings of the on Great Lakes Symposium on VLSI 2017; ACM. 2017. pp. 221-226

[90] An H, Zhou Z, Yi Y. Opportunities and challenges on nanoscale 3D neuromorphic computing system. In: 2017 IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity (EMCSI); IEEE. 2017. pp. 416-421

[91] Ehsan MA, Zhou Z, Yi Y. Hybrid three-dimensional integrated circuits: A viable solution for high efficiency neuromorphic computing. In: 2017 International Symposium on VLSI Design, Automation and Test (VLSI-DAT); IEEE. 2017. pp. 1-2

[92] Ehsan MA, An H, Zhou Z, Yi Y. A novel approach for using TSVs as membrane capacitance in neuromorphic 3-D IC. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2017;**37**(8): 1640-1653

[93] Ehsan MA, Zhou Z, Yi Y. Three dimensional integration technology applied to neuromorphic hardware implementation. In: 2015 IEEE International Symposium on Nanoelectronic and Information Systems; IEEE. 2015. pp. 203-206