An efficient hardware-oriented dropout algorithm

doi: https://doi.org/10.1016/j.neucom.2020.11.055

An Efficient Hardware-Oriented Dropout Algorithm

Y.J. YEOH*, T. MORIE and H. TAMUKOH

Kyushu Institute of Technology, Japan

ARTICLE INFO

Keywords:
Dropout
FPGA
DNN

ABSTRACT

This paper proposes a hardware-oriented dropout algorithm, which is efficient for field programmable gate array (FPGA) implementation. In deep neural networks (DNNs), overfitting occurs when networks are overtrained and adapt too well to training data. Consequently, they fail in predicting unseen data used as test data. Dropout is a common technique that is often applied in DNNs to overcome this problem. In general, implementing such training algorithms of DNNs in embedded systems is difficult due to power and memory constraints. Training DNNs is power-, time-, and memory- intensive; however, embedded systems require low power consumption and real-time processing. An FPGA is suitable for embedded systems for its parallel processing characteristic and low operating power; however, due to its limited memory and different architecture, it is difficult to apply general neural network algorithms. Therefore, we propose a hardware-oriented dropout algorithm that can effectively utilize the characteristics of an FPGA with less memory required. Software program verification demonstrates that the performance of the proposed method is identical to that of conventional dropout, and hardware synthesis demonstrates that it results in significant resource reduction.

1. Introduction

Deep neural networks (DNNs) have demonstrated promising performance in numerous applications, such as data mining, automation, and natural language processing [1, 2, 3, 4]. Examples include GoogLeNet-, which succeeded in image recognition in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [5, 6]; and, Alpha-Go which defeated the top human player in the game Go [7]. Advances in computer technology and networks have led to breakthrough in DNNs, which have become a popular topic among researchers. The number of studies on deep learning increases yearly. As neural networks become increasingly deeper, the computational costs, memory and power consumption increase. These factors make training more difficult, and it also become difficult to implement DNNs in an embedded system, which requires high mobility and real-time processing [8].

In the field of robotics and automation, mobility and power consumption have become quite important. Researchers generally focus on the implementation of DNNs in hardware devices, which consume less power than software devices such as a CPU or graphics processing unit (GPU) [8, 9]. A field Programmable Gate Array (FPGA) is a digital hardware device composed mainly of transistors and programmable wires that enable flexible implementation and parallel processing [10, 11]. Unlike software devices that process serially, an FPGA enables parallel processing, making real-time processing a promising goal [12]. However, a large drawback of the FPGA is its memory constraint. The memory resources in an FPGA are limited, and computations are executed digitally. Thus to process a floating point number calculation or analog input from sensors, implementation becomes difficult. Consequently, research has been conducted on modifying the algorithm of general DNNs, such as the binarized neural network (BNN) and XNOR net, to simplify the calculation, reduce the computational cost and memory resources, and make the algorithm suitable for hardware implementation [13, 14, 15, 8]. In addition to research on algorithms, there has also been research on architecture configuration design of FPGAs to achieve a parallel, resource-saving implementation [9]. This research includes designing, controlling, and manipulating the utilization of block RAM, digital signal processing (DSP), and look-up table (LUT) [9]. Although many studies have focused on accelerating neural networks with an FPGA, most implemented only the inference phase, as the learning and update process of a DNN involves complex computation and high resource cost [16, 9].

As illustrated in Fig. 1, this paper focuses on developing a hardware-oriented algorithm for a trainable DNN in an FPGA, and we propose a hardware-oriented dropout algorithm for efficient implementation. While training DNNs,

*Corresponding author
yeoh.yoeng-jye542@mail.kyutech.jp (Y.J. YEOH); morie@brain.kyutech.ac.jp (T. MORIE); tamukoh@brain.kyutech.ac.jp (H. TAMUKOH)

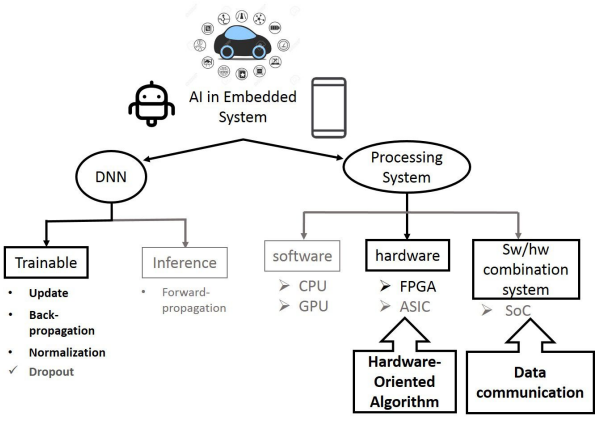


Figure 1: Overview of research target.

the hyperparameters of the networks and the size of the dataset are difficult to initialize. The larger and deeper the networks are, the more features must be trained and learned. This may cause the networks to become too specified to the training data and learn even the noise. As a result, overfitting occurs and resulting the networks fail during testing. However, networks that are too shallow and small are also unable to train well, also resulting in poor performance. As the size of a dataset decreases, the networks tend to adapt to all the data, thus overfitting to the data. In contrast, when the dataset size is too large, the learning process become difficult and is too generalized. The optimal hyperparameters and size of a dataset are difficult to determine, as they have no baseline and are case-dependent.

Dropout is a technique that is often applied to DNNs to solve the overfitting problem [17]. By randomly dropping neurons with a certain ratio, it prevent the neurons from collaborating with each other [17, 18]. In addition, forcing part of the neuron to acquire a zero value simplifies the calculation.

However, to randomly drop neurons, random number generators (RNGs) are required, which are costly in terms of hardware. Our proposed method eliminates the use of RNGs, substituting them with simple operators for performing dropout. This results in saving hardware resources for other purposes. The contributions of this paper are summarized as follows:

- An efficient hardware-oriented algorithm for the dropout technique is proposed.
- Evaluation of the proposed method is performed with feedforward neural networks and recurrent neural network.
- A comparison between general dropout and the proposed method is performed.
- Hardware synthesis is performed for resource analysis.

2. Related Works

2.1. Feedforward neural networks

A feedforward neural network is generally used in image classification tasks. This network is also the oldest and simplest neural network. As implied by its name, the input is directly fed forward to the output, passing through a hidden layer with the sum of products of the weights and activation functions. In this network, there is no looping or feedback. A multi-layer perceptron (MLP) and convolutional neural network (CNN) which are used in this study, are categorized as feedforward neural networks.

2.1.1. Multi-layer perceptron (MLP)

An MLP is a simple neural network that maps the inputs to outputs with multiple hidden layers in between [19]. All neurons are fully connected by weights. The sum of the products of the weights W and inputs x is computed and fed to the next layer after the activation function, f. Figure 2 illustrates the four-layers MLP used in this work, with 500 and 200 hidden units in the two hidden layers respectively.

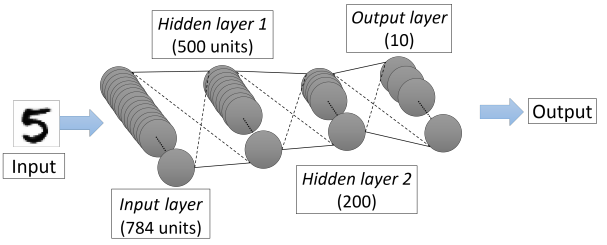


Figure 2: Four-layers multi-layet perceptron with 500 and 200 hidden units.

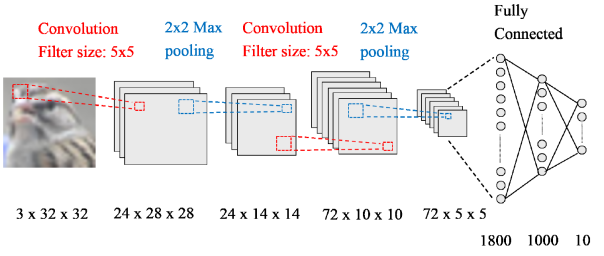


Figure 3: Example of convolutional neural network: LeNet.

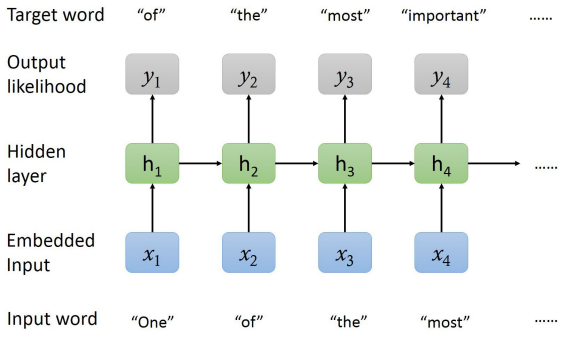


Figure 4: Illustration of recurrent neural network language model

2.1.2. Convolutional neural network (CNN)

The first CNN was introduced in 1998 and has become popular in recent years due in part to its excellent performance in image recognition [20] In 2012, CNNs significantly reduced error rates and successfully outperformed other models in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [5]. Deeper CNNs such as GoogLeNet, ResNet, and VGGNet were introduced in subsequent years. In a CNN, input is computed by convolving with multiple filters, also known as kernels. The filters convolve part of the input and slide to the ascending part until all the input has passed through. The weights are shared spatially. The pooling layer in a CNN reduces the resolution of the input to a smaller scale, allowing the extracted features to have scale invariant properties. CNNs usually consist of pairs of convolutional-pooling layers and fully-connected layers as a classifier, as illustrated in Fig. 3. Figure 3 presents LeNet which consists of two pairs of convolutional-pooling layers.

2.2. Recurrent neural network language model

Unlike feedforward neural networks, recurrent neural networks (RNNs) make use of sequential information in predicting output [21]. In another words, RNNs' inference does not solely depend on the current input, but also on information from the previous state. This is similar to human decision-making, in which a case is judged not only using

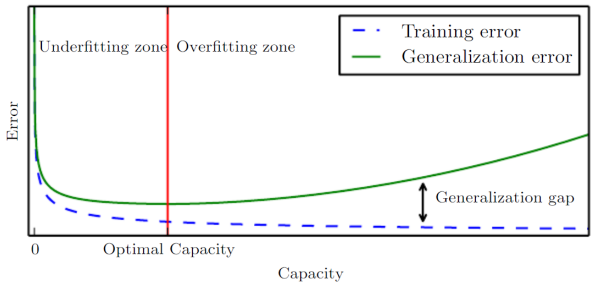


Figure 5: Overfitting problem in a neural network

current information, but also using past experience. The RNN language model (RNNLM) is a network model used for predicting a sentence in language and an example is presented in Fig. 4 [22]. By carrying previous information to the next layer, the model links all input and predicts the next word with the highest possibility of appearing after the current word.

In the RNNLM, perplexity is used instead of accuracy to measure performance as in Eq. 1. The perplexity of discrete probability distribution p is defined such that $H(p)$ is the entropy (in bits) of the distribution and x ranges over events [23]. The model is trained to minimize the cross entropy $H(p)$ to increase performance. In other words, the smaller the perplexity, the better the RNNLM.

$$2^{H(p)} = 2^{-\sum_x p(x)\log_2 p(x)} \tag{1}$$

2.3. Dropout

In neural networks, overfitting is a common problem during training. As illustrated in Fig. 5, the network is trained to learn and fit the training data. Consequently, although the training error is minimized, the generalization gap and error are increased over time. Dropout is a regularization technique used to overcome this problem [17]. This technique randomly drops neurons in the network with a certain ratio to avoid complex co-adaptions between neurons and to learn more robust features as shown in Fig. 6 [18, 24]. In addition, because training neural networks with dropout deactivates neurons with a certain ratio, this eventually decreases the size of networks, resulting in sparser networks. As a result, the computational cost is decreased [25]. To apply the dropout technique, a dropout mask is attached to each layers of the network. The dropout mask is a binary vector with a Bernoulli distribution, as in Eq. 2 and 3. The neurons are dropped when multiplied by 0, whereas neurons multiplied with 1 will remain as same and are fed forward to next layer. The dropout masks are changed for every input.

$$mask_i^{(l)} \sim Bernoulli(p) \tag{2}$$

$$y_j^{(l+1)} = f\{\boldsymbol{W}_j^{(l+1)} \cdot (\boldsymbol{mask}^{(l)} * \boldsymbol{x}^{(l)}) + b_j^{(l+1)}\} \tag{3}$$

2.4. The Extended Works of Dropout

The dropout method has demonstrated effectiveness in solving the overfitting problem for various networks [17, 24], and several studies have further widened its application [26, 27, 28]. One extended dropout method, Dropconnect, drops the connections (weights) between neurons rather than the neurons themselves [27]. Several studies have also implemented dropout in an FPGA [25, 29]. In a study of Su et. al., a restricted boltzmann machine (RBM) with dropout was implemented on an FPGA. Instead of comparing random numbers to dropout ratio, only HSd random numbers were generated and the dropout ratio was determined by HSd/HS, where HS was the size of neuron layer [25]. The random numbers were used as the indices of column in weight matrix, to address the selected weight which stored in external memory [25]. The serial process of RNG and the transfer cost of weight between RNG, external memory and

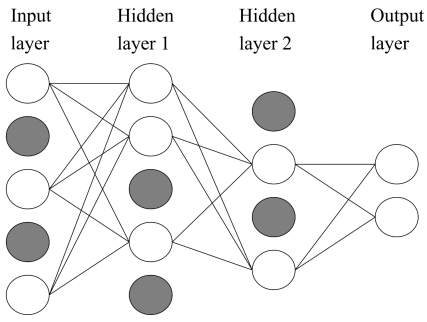


Figure 6: Illustration of dropout in neural network

on-chip RAM is concerned. In a study of Sawaguchi et. al., slightly-slacked dropout was proposed, to alleviate the transfer cost of the method of Su et. al. and accelerate the training [29]. A neural central controller was introduced to control the neuron group (NG) and combination (NC) information of slightly-slacked dropout [29]. Four subsequent neurons formed a group (NG) with a certain dropout rate, and the approximation of dropout rate was computed across all groups [29]. The dropout rate of each NG can only be set to 1, 0.5 or 0, where when the dropout rate was set to 0.5, two neurons will be chosen out of four patterns (fixed) which was controlled using 2-bits (NG Info) [29]. The 2-bits NC info were used to assign one out of four combination of the dropout rate of either one, two or three NGs [29]. However, these approaches have limitations, such as accuracy degradation, transfer costs, and problems as operating the dropout technique externally between software and hardware [25, 29]. In this paper, we propose an alternative approach that fully enables the application of dropout in hardware with parallel processing in order to address the problem of transfer costs. In addition, resources are reduced by eliminating RNG. The proposed method is compared to the conventional dropout method as a baseline.

3. Proposed Method

The current dropout approach is primarily a software-based implementation. To generate a dropout mask, which is a binary vector, a random number generated from an RNG is compared to the dropout ratio which is usually set to 0.5 by a user for a hidden layer [17]. If the random number is larger than the dropout ratio, the value of the dropout mask is set to 1, and if the random number is smaller than the dropout ratio, the value is set to 0 [17]. This process is looped until a binary vector (equal in size to the corresponding layer) is generated. The dropout mask is generated differently for each layer of the network, and regenerates with changed input. In software, a random number is easily generated from a program, and executed in a fast clock cycle, thus the looping process does not have a large impact. However, the looping process significantly increase the burden of execution in an FPGA due to different architecture. The serial looping process is not preferred in an FPGA and is ineffective for FPGA implementation, as RNGs and comparators are required, which consume significant resources. To process in parallel, the number of RNG and comparator will eventually increase, further consuming FPGA resources.

Figure 7 presents a block diagram of general dropout in hardware implementation with serial processing. The input neurons and weight parameters are summed and multiplied with multiply-accumulate units, while at the same time, the dropout mask is generated by comparing the random number and dropout ratio. A looping block is required, as only one bit is generated in the comparison. The dropout mask is connected to the enable of the D-latch, which controls the activation of the neurons. The looping block (green) can be eliminated by multiplying the RNG and comparator block and processing the comparison in parallel; however, resource consumption significantly increases, as illustrated in Fig. 8.

Thus, to efficiently apply the dropout method in hardware, we propose an alternative approach. The proposed method eliminates the use of an RNG and comparator; instead, a predefined mask is used with the addition of a control block and reconfiguration block. The proposed method not only saves resources by eliminating the use of an RNG and comparator, but also processes in parallel, allowing the regeneration of the dropout mask in a single clock cycle. In general, dropout drops a neuron with true randomness, whereas the proposed method drops neurons with pseudorandomness, which is considered sufficient for most applications even though the randomness has a pattern and

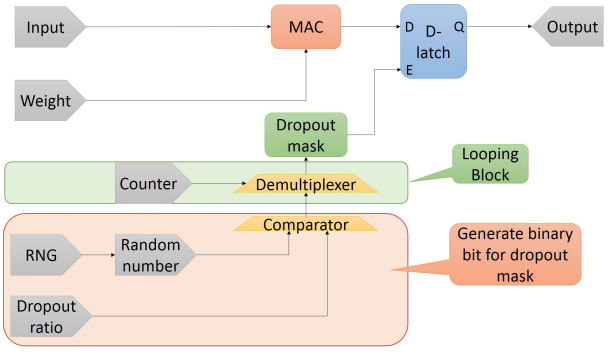


Figure 7: Block diagram of general dropout in hardware implementation. (MAC: multiply-accumulate)

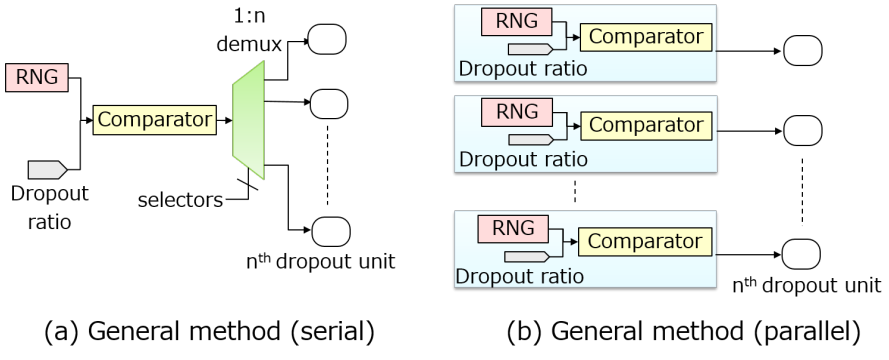


Figure 8: Block diagram of comparator block in serial and in parallel

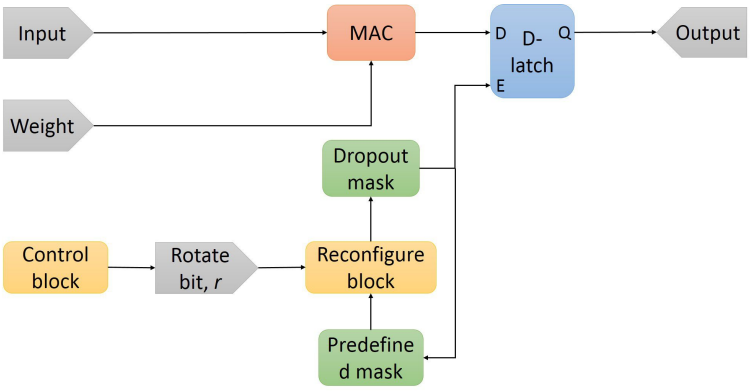


Figure 9: Block diagram of proposed dropout method

is predictable [30]. It has been hypothesized that true randomness in dropout is not essential, and that dropout can perform well even with pseudorandomness [31].

A predefined mask is generated and saved in memory for initialization purposes. For each generation of the dropout mask, a predefined mask is loaded to the reconfiguration block to reconstruct the mask, changing the sequence of the mask as a new dropout mask. In the reconfiguration block, a simple, parallel operation is executed. For simplicity, the experiments in this study applied rotation in the reconfiguration block, whereas in the control block, the parameter of

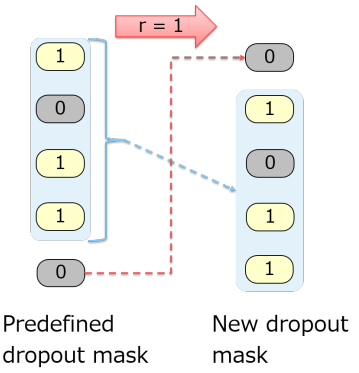


Figure 10: Configuration block of proposed dropout method

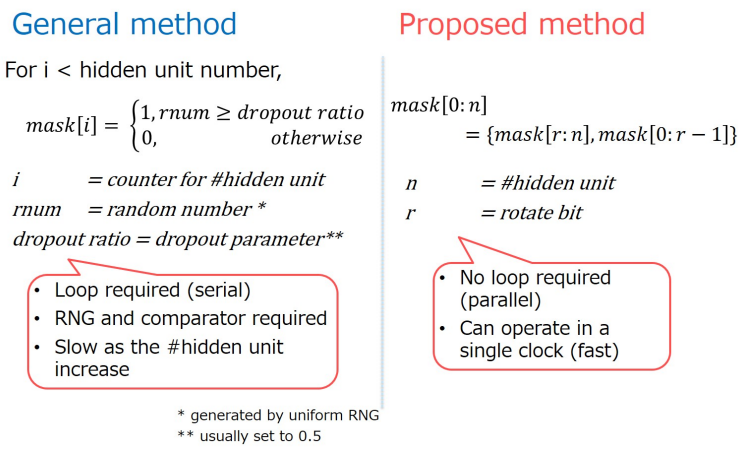


Figure 11: Comparison between general dropout algorithm and proposed algorithm

the rotate bit, r, was used to control the bit rotation in the configuration block. By only rotating bits of the mask, the distribution is maintained and the operation remains simple. In this paper, the effect of parameter r was studied by setting it to be constant, a sequence, and random. Instead of an RNG and comparator, the reconfiguration block and control block were used to consume fewer resources with high processing speed. Figure 9 presents a block diagram of the proposed method, while Fig. 10 presents a simple illustration of the rotation in the configuration block. A new dropout mask can be generated by rotating the bit of the dropout mask in parallel. The figure illustrates that no looping process is required in the proposed method. In Fig. 11, algorithms for general dropout and the proposed method are compared. As demonstrated in the figure, the serial looping for general dropout is effective in hardware, as the clock speed is slow, and the clock cycle required is proportional to the number of neurons. In contrast, the proposed method is simple and operates in parallel in a single clock cycle with fewer resources.

4. Experimental Results

We conducted several experiments to verify the effectiveness of the proposed method. The experiments had two main approaches: software program verification and hardware resources analysis. In software program verification, the algorithm of the proposed method was examined and compared with that of general dropout. Various types of neural networks with different datasets were used to test the effect and robustness of the proposed method. In hardware resources analysis, the resources consumed in generating the dropout masks were compared, to observe the amount of resources that could be preserved.

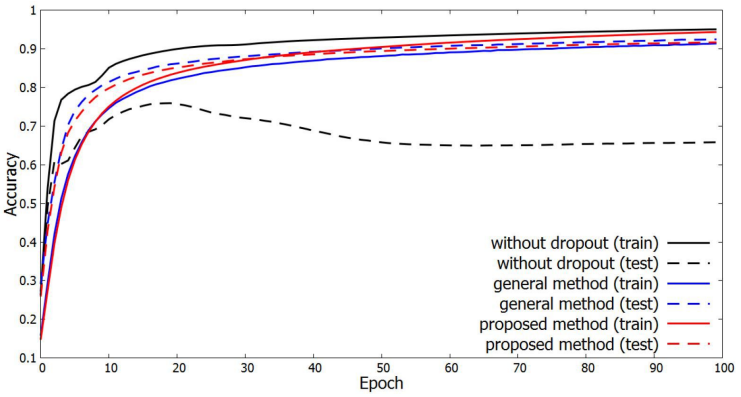


Figure 12: Results of multi-layer perceptron (MLP) trained with MNIST dataset.

4.1. Software program verification

Four different types of network architectures with four different datasets were tested, as follows:

1. MLP - MNIST
2. LeNet - CIFAR10
3. GoogLeNet - @home dataset
4. RNNLM - Penn Treebank (PTB)

The MLP network was trained using the C language, whereas the other networks were trained and tested using the Chainer platform [32]. Each network was trained for five trial to ensure its robustness, and the average results were obtained and plotted as a figure. The performance of the conventional method and proposed method was compared in terms of classification accuracy.

4.1.1. MLP - MNIST

The first network implemented was a four-layer MLP, which was 784-500-200-10 as illustrated in Fig. 2. The MLP was trained with the MNIST dataset, which is a dataset of handwritten number images [20]. Each image in MNIST was a grayscale image with a size of 28 x 28 pixels (784 pixels). It contained 10 classes (from 0 to 9), and a total of 60,000 training data images and 10,000 test data images. To train the MLP, mini-batch stochastic gradient descent (SGD) with a batch size of 100 was used as the optimizer. Softmax regression was used as the cost function.

The training and test accuracy of the three approaches is presented in Fig. 12. The overfitting phenomenon can be clearly observed for the MLP that did not apply dropout (black lines). The training accuracy (solid- line) was close to 95%, while the test accuracy (dotted- line) was saturated at approximately 65%, revealing a large gap between the two. The MLP failed to predict new test data due to overtraining and overfitting to the training data. In contrast, the general dropout method (blue lines) was able to avoid the overfitting problem and achieve accuracy of over 90% during the training and inference phase. The proposed method (red lines) performed well and achieved a similar effect to the general dropout method.

4.1.2. LeNet

Here, the proposed method was evaluated with LeNet, a CNN with the architecture illustrated in Fig. 3. Adam optimization was used, and dropout was only applied in the fully-connected layer to avoid unnecessary loss. Spatially shared- weights in a CNN have relatively few parameters and generally do not cause overfitting. The dataset used in this experiment was CIFAR10, a dataset containing 10 classes of objects, including an airplane, automobile, and bird [33]. CIFAR10 contained 50,000 images as training data and 10,000 images as test data. Each image size was 32 x 32 pixels and consisted of RGB, three color channels.

Figure 13 demonstrates that all of the approaches had similar results; the training accuracy was over 90%, while the test accuracy was saturated at approximately 70%. Although dropout was applied for LeNet, overfitting still occurred. This may be because LeNet is a shallow CNN and may not perform well with the CIFAR10 dataset, which comprises color image data that are relatively difficult to classify. However, the main objective of this paper is not to examine

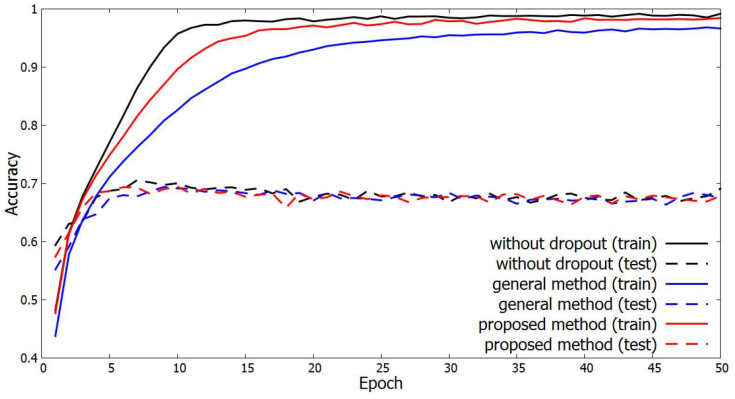


Figure 13: Results of LeNet trained with CIFAR10 dataset.

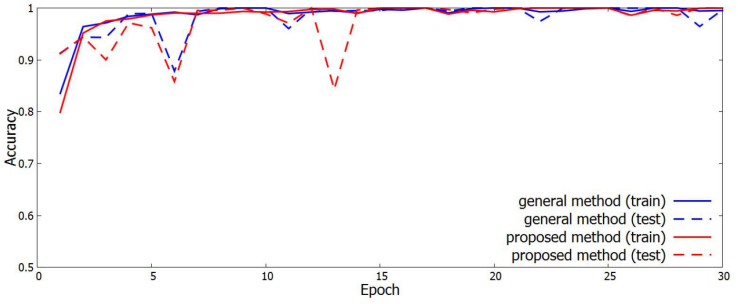


Figure 14: Results of GoogLeNet trained with @home dataset.

the effect of dropout, but rather to compare the proposed method with conventional approaches. Therefore, although dropout did not demonstrate favorable results in this experiment, the proposed method was observed to have identical results to general dropout.

4.1.3. GoogLeNet

This experiment involved a deeper CNN called GoogLeNet which consists of 22 network layers and took first place for object classification in the ImageNet Large- Scale Visual Recognition Challenge (ILVRC) in 2014 with excellent recognition performance [5, 6]. The @home dataset was created for the RoboCup Japan Open 2016 @Home competition [34, 35]. The CNN was trained with the dataset and implemented into a home service robot to allow the robot to recognize objects and perform service tasks. The @home dataset was a 15 object class dataset, in which each class contained of 2,000 training data images and 700 test data images, as illustrated in Table 1. Each image was resized to 224 x 224 pixels with three color channels. To match the dataset, the output layer of GoogLeNet was modified to 15 neurons instead of 1,000 neurons.

The main concern of this study is to compare and evaluate the performance between general dropout and the proposed method, thus the result of GoogLeNet without dropout is not demonstrated. The performance of general dropout and the proposed method is compared in Fig. 14. Both approaches achieved stable and recognition accuracy close to 100% after 15 epochs. The high accuracy in both the training and inference phases indicates that the network was trained well and no overfitting occurred. The deep CNN was highly effective for the @home dataset, and the proposed method achieved the same effect as the general method.

In the proposed method, the dropout mask length is equal to the number of neurons in each respective layer. Parameter r is adjustable and was set to a sequential constant number in the previous experiments. The effect of changing r is another concern; thus, with the same environment settings as GoogLeNet experiment, r was changed to a constant number, 1, 2, 4, 8, 16, and 32, to observe the produced effect. Figure 15 indicates that the accuracy was

Table 1
Object classes in @home dataset

Item Categories	Examples		
PET bottle	 Iced-tea	 Cafe-Au Lait	 Green tea
Snack	 Potato stick	 Potato chips	 Chocolate cookies
Fruit juice	 Orange juice	 Strawberry juice	
Fruit	 Orange	 Apple	
Instant soup	 Egg soup	 Potato soup	
Container	 Tray	 Bowl	 Cup

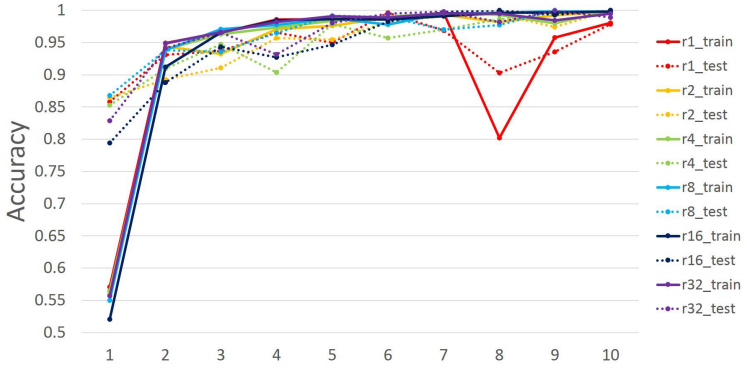


Figure 15: Effect of proposed method with varied parameter r

relatively unstable when r was set to 1; however, it did not significantly change when r was increased. Thus, r can be defined as a constant or varying function depending on the available resources.

4.1.4. RNN

In addition to examining feedforward neural networks, the experiments were extended to verify the effectiveness of an RNN. RNNLM was implemented and trained with the Penn Treebank (PTB) dataset, which contains 10,000 vocabulary words in sequence and is commonly used in natural language processing [36]. Words corresponding to their ID were input as a one-hot vector that was embedded into an embedded matrix before inputting to RNN. The RNNLM used in this experiment consisted of two hidden layers with long-short-term memory (LSTM), and the dropout layer

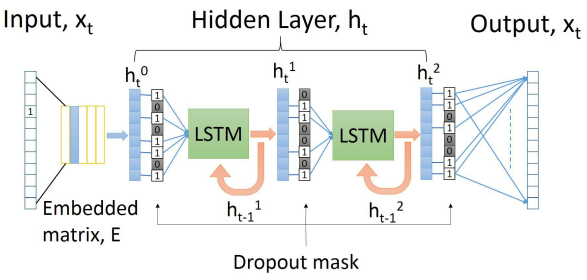


Figure 16: Structure of recurrent neural network language model (LSTM: long-short-term memory)

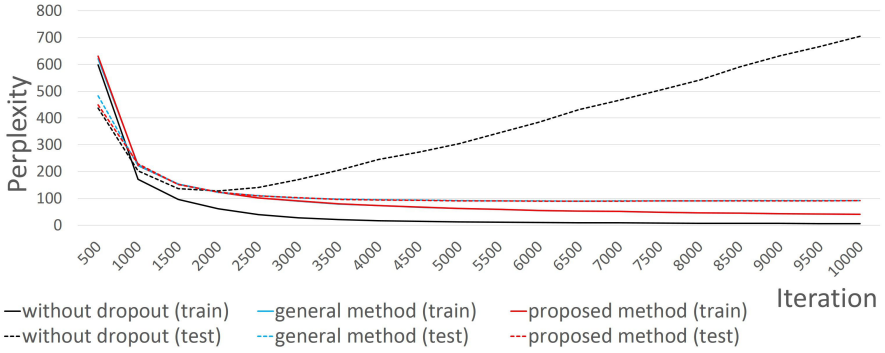


Figure 17: Comparison of recurrent neural network language model

Table 2
Comparison of test perplexity between different methods

Approaches	Test perplexity
Without dropout	757.67
General dropout	88.99
Proposed dropout	89.96

was applied before the LSTM layer. The structure of the RNNLM was composed by five layers (10,000-650-650-650-10,000), as illustrated in Fig. 16.

When dropout was not applied, overfitting occurred, as indicated in Fig 17. The test perplexity increased while the training perplexity remained low. However, when dropout was applied to the RNNLM, the gap between training perplexity and test perplexity was reduced. Table 2 also illustrates that the test perplexity reached over 750, which was in contrast to approaches using dropout, whose perplexity values were only approximately 89. The results of the proposed method were thus identical to those of the general dropout method.

4.2. Hardware resources analysis

4.2.1. Dropout Mask Generation using Verilog HDL

After verifying the effectiveness of the proposed method, we examined resources saving. Hardware resource analysis was performed to compare the amount of resources required using the general method and proposed method. An experiment was conducted by implementing the dropout mask only. We generated 8- and 64- bit dropout masks using the general method in series, general method in parallel, and the proposed method. In actual application, the size of the dropout mask is proportional to neurons in the layers, and multiple dropout masks are required for different layers of neural network. Thus, it is expected that more resources would be saved for actual application than are presented in

Table 3
Comparison of field programmable gate array resources consumed for 8-bit dropout masks

Logic utilization	General dropout (serial)	General dropout (parallel)	Proposed method
Number of slice registers	32	7	8
Number of slice LUTs	44	80	7
Number of fully used LUT-FF pairs	27	72	0
Clock cycle required to generate a mask	8	1	1
RNG required	1	8	N/A

Table 4
Comparison of field programmable gate array resources consumed for 64-bits dropout masks

Logic utilization	General dropout (serial)	General dropout (parallel)	Proposed method
Number of slice registers	149	588	70
Number of slice LUTs	190	640	64
Number of fully used LUT-FF pairs	141	576	64
Clock cycle required to generate a mask	64	1	1
RNG required	1	64	N/A

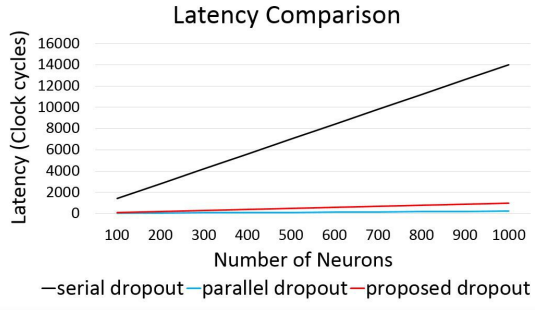
Table 3 and 4.

In Table 3, the resources consumed by an 8-bit dropout mask generated by different approaches are tabulated through hardware synthesis. The most resources were consumed with general method with parallel processing due to the duplication of RNGs and comparator blocks. For the general method with serial processing, fewer resources were required; however, a longer time and clock cycle were required. The proposed method had the advantages of the other two approaches, as it consumed the fewest resources and required only a single clock cycle to generate the mask. Similarly, a comparison was performed between the three approaches in generating a 64-bit dropout mask, as presented in Table 4. As indicated in the table, a large number of resources was required with the serial approach, which was further increased with the parallel approach. Fewer resources were consumed with the proposed method, and only a single clock cycle was required. Therefore, the proposed method is effective in hardware implementation and is characterized by low resource consumption and a fast processing speed.

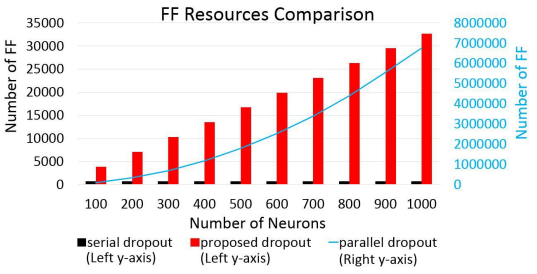
4.2.2. Dropout Mask Application using HLS

In this section, we observe the resources when applying different approaches of dropout in neural network and compare them with increasing number of neurons in a layer. Thus, the experiments were executed with initial 100 neurons, and increasing 100 neurons each time until 1000 neurons. The experiments were synthesized using High Level Synthesis (HLS) coding in Vivado HLS 2018, with targeted to ZYNQ UltraScale+ ZCU102 FPGA board as a reference. The comparison results in aspect of latency, flipflop (FF), look-up table (LUT), and digital signal processor (DSP) are shown in Fig 18.

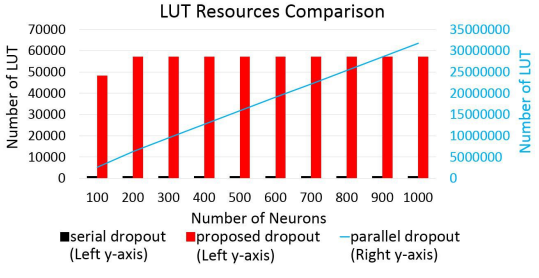
As in Fig 18, four approaches of dropout were run and analyzed, which are conventional dropout in serial connection (black), conventional dropout in parallel connection (blue), and the proposed dropout (red). For latency comparison (Fig 18a), as the expectation result based on the conclusion drawn from previous section, the serial dropout is showing highest latency, where the parallel dropout shows a very high speed processing in contrast. The proposed dropout is



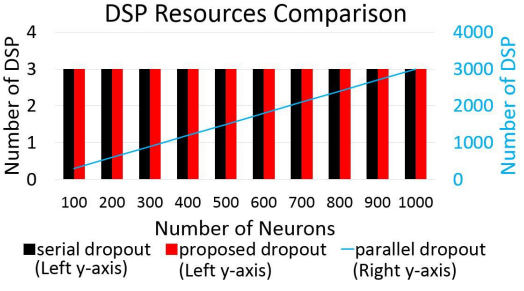
(a) Latency comparison



(b) Flip-flop (FF) resources comparison



(c) Look-up Table (LUT) resources comparison



(d) Digital signal processor (DSP) comparison

Figure 18: Latency and resources comparison with increasing in number of neurons

able to achieve high speed processing as well with a slightly increasing in latency compare to parallel dropout. In the aspect of consumption of FF (Fig. 18b), the serial dropout achieves to consume a very low amount of resources, which average is below 1000 in number. For the proposed method, the FF consumption were increased proportionally to the number of neurons, as the resources for configuration block. However, the parallel dropout is showing a huge leap of FF consumption (right y-axis), which is painful and not affordable for FPGA implementation. The similar result can be observed in Fig 18c, as the serial dropout consumes the least LUT resources, where the parallel dropout is at extremely high cost. Lastly, the number of DSP consumption are observed in Fig. 18d. For serial dropout and proposed dropout, only three DSP were used, where for the parallel dropout, over hundreds of DSP are used and shown an increasing corresponding to the number of neurons. Based on the result, a short conclusion can be drawn in this section, that the serial dropout consumes relatively low resources, but at higher latency. On the other hand, the parallel dropout achieves a very low latency, which can be insignificant in the overview of neural network, yet the cost is extremely painful, and exceeding the total available resources in FPGA, which is unrealistic. The proposed dropout can be consider as the compromise and balance between serial dropout and parallel dropout, allowing a low latency process at an acceptable range of resources consumption.

Note that the results showing in this section may have a slightly contradiction to the results in section 4.2.1. These may due to the situation when attaching the dropout mask to neuron layers, the resources for input and output neurons are computed as well, instead of only the mask generation. For the serial dropout, the input and output can also be connected in serial during processing, where for the proposed dropout, the input and output layers required to be in parallel, causing the increased in number. Also, the results for this section are generated from HLS coding, the auto-generation of HDL code may not fully optimized, which causing undesired additional resources. Further optimization may apply for improvement, such as the pipeline optimization.

By importing the HLS project to Vivado, implementation can be executed and the total on-chip power can be obtained. The total on-chip power is the power consumed internally within FPGA, which is the sum of the static power (device static power: power from transistor leakage; design static power: power for design configuration) and the dynamic power (switching power, the average power due to the design internal activity) [37]. We tabulate the total on-chip power and calculate the energy required for generating the dropout mask with 100 neurons as in Table 5. The latency is tabulated from the results in Fig. 18a. In the targeted ZCU102 board in this experiment, multiple clocks are

Table 5
The power and energy for dropout mask with 100 neurons at 100MHz

	Serial dropout	Parallel dropout	Proposed dropout
Total On-chip Power (W)	0.73	2.029	0.678
Latency (clock cycles)	1402	39	107
Energy (µJ)	10.2346	0.79131	0.72546

Table 6
Latency and resources comparison in MLP: 784-100-10.
Red numbers indicate that the amount is exceeding the available amount in FPGA.

	Available amount	Serial dropout	Parallel dropout	Proposed dropout
Latency	-	2502607	2389957	2397637
FF	548160	80291	24951168	115628
LUT	274080	190362	4393285	161449
BRAM	1824	787	788	789
DSP	2520	5	2483	8

available. The calculation of energy in Table 5 is computed with clock frequency set as 100MHz where the energy may varies with different frequencies. From the table, it can be observed that although the power required for the serial dropout is small enough, the energy required is larger as the clock cycles required is larger, resulting energy inefficient. For parallel dropout, the energy required is similar to the proposed method, but as the number of neurons increased, the higher resources required which will increased the power, thus higher energy required is expected. The proposed dropout achieved the most energy efficient among the three approaches. Moreover, based on Fig. 18, the resources required for proposed method is significantly less than the parallel dropout with low latency, indicating the proposed method would be more energy efficient as the number of neurons increased, compare to both the other approaches.

4.2.3. Application in MLP

A further experiment was carried out to observe the effect of dropout to the number of resources consumed in application of neural network. We had run the experiment with the structure of MLP 784-100-10 as for different approaches of dropout, and only feed forward propagation was run for this experiment. Floating point operation is used throughout the synthesis. Further optimization techniques, such as fixed point operation, pipeline processing, memory allocation and others, may be applied during the implementation to increase effectiveness. However, this paper would like to focus on the dropout and the proposed method's effectiveness, thus, we do not apply optimization for the feed forward MLP in this case. As the data can be observed in Table 6, the proposed method achieve to reduce the latency by around 100k clock cycles compare to serial dropout, which similar performance as in parallel dropout. In the aspect of resources, the amount of FF used by proposed method is slightly higher than serial dropout, but the amount of LUT is slightly less, which is balanced in overall. Whereas for the parallel dropout, even though the least latency is achieved, yet the resources consumed for both FF and LUT, are extremely high and exceeding the total available amount in FPGA board, which make it impossible for implementation. Also, the parallel dropout used up almost all of the amount of DSP, where the other approaches only consume less than 10 DSP.

4.3. Discussion

4.3.1. The Proposed Method

We made the comparison between the general dropout and proposed dropout in aspect of processing, time, randomness, generation of dropout mask, resources required and its suitability device, as tabulated in Table 7. The general dropout is a serial processing algorithm where the proposed dropout designed for parallel processing. Thus, the proposed dropout is faster than general dropout and independent to the number of neurons (the size of dropout mask). However, the proposed method has lower randomness as it generates dropout masks based on a predefined mask initially. In aspect of resource, as the general dropout requires RNGs and comparators, thus the resource consumed is high; Where the proposed dropout requires only very small amount of resource. For software application such as CPU,

Table 7
Summary of general dropout and proposed dropout

	General dropout	**Proposed dropout**
Processing	Serial looping	Parallel
Time	Slow (dependent on # of neurons)	Fast (independent of # of neurons)
Randomness	High	Normal
Dropout mask	Regenerated for each layer	Predefined
Resources Required	Very high (for parallel) high (for serial)	Low
Suitable application	Software	Hardware

the general dropout is more suitable, whereas for hardware devices such as FPGA, the proposed dropout has advantages to it and is efficient for the implementation.

4.3.2. Pseudo RNG

The proposed method is demonstrating the dropout technique for neural network in a simpler way to achieve pseudo random dropout mask with minimum resource. There are lots of research working on the RNG in hardware implementation, both pseudo random number generators (PRNGs) and true random number generators (TRNGs) as review in [38]. Each approaches has its advantages and limitations. Incontrovertibly, implementing dropout with these approaches will increase the efficiency. However, the problems for conventional dropout are still remained unsolved which are the looping process in serial and the duplication of RNG and comparator blocks in parallel (mentioned in Chapter 3). Applying the random number from RNG, directly to the neuron layers seems to be a possible alternative option, without generating the dropout mask. Nevertheless, the number of bit, the distribution will be the concern as the dropout ratio is difficult to fix and control in such situation. Instead, the proposed method is providing a simpler solution, without the implementation of RNG in regeneration of dropout mask. The dropout ratio is fix and set by the predefined mask, and is controllable by resetting the predefined mask.

4.3.3. Other Hardware-oriented Implementation

As mentioned in section 2.4, there are other researches working on hardware implementation as well, such in [25] and [29]. In the implementation in [25], the authors utilize the external memory to address the selected weight after dropout. Instead of optimizing dropout algorithm, the paper more focus on their proposed model with dropout [25]. Also, the paper also mentioned that the implementation of dropout in RBM has poor scalability of DSPs efficiency due to the transfer time and linearity. We are expecting our proposed method that can operate in parallel, and without using the external memory, can overcome this problem to increase the efficiency. Further in [29], the authors are targeting to minimize the transfer cost of dropout between hardware and software. As the result of the paper, 35026 LUTs, 49784 Registers and 112 DSPs were used for the implementation, with around 5% accuracy degradation [29]. On the other hand, we are proposing an alternative approach that fully implement dropout algorithm using hardware, which no transfer cost is required. Although the LUTs and registers consumed is higher compare to [29], but the DSP is significantly minimized. Therefore, it would be an alternative option for different compromisation in aspect of transfer cost, DSPs and etc.

5. Conclusions

We demonstrated that our proposed dropout method has the same performance as the general dropout method and it is characterized by parallel processing and greater resource savings. The proposed method is effective in hardware implementation with its parallel processing characteristic and Using the proposed method, the dropout technique can be applied with fewer resources without reducing the performance of the neural network. The number of resources can be further reduced as the number and size of the dropout masks increase. The saved resources can be utilized in other algorithms in neural networks, overcoming the memory constraint in FPGAs.

Acknowledgment

This research was supported by JSPS KAKENHI Grant Numbers 17H01798 and 17K20010.

References

[1] J. Schmidhuber, Deep learning in neural networks: An overview, Neural networks 61 (2015) 85–117.
[2] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[3] B. Jan, H. Farman, M. Khan, M. Imran, I. U. Islam, A. Ahmad, S. Ali, G. Jeon, Deep learning in big data analytics: A comparative study, Computers & Electrical Engineering.
[4] E. Fathi, B. M. Shoja, Deep neural networks for natural language processing, Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications 38 (2018) 229.
[5] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.
[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.
[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, nature 529 (7587) (2016) 484–489.
[8] S. Liang, S. Yin, L. Liu, W. Luk, S. Wei, Fp-bnn: Binarized neural network on fpga, Neurocomputing 275 (2018) 1072–1086.
[9] Z. Hajduk, Reconfigurable fpga implementation of neural networks, Neurocomputing 308 (2018) 227–234.
[10] E. Sanchez, Field programmable gate array (fpga) circuits, Towards Evolvable Hardware (1996) 1–18.
[11] S. D. Brown, R. J. Francis, J. Rose, Z. G. Vranesic, Field-programmable gate arrays, Vol. 180, Springer Science & Business Media, 2012.
[12] A. R. Omondi, J. C. Rajapakse, FPGA implementations of neural networks, Vol. 365, Springer, 2006.
[13] Z. Lin, M. Courbariaux, R. Memisevic, Y. Bengio, Neural networks with few multiplications, arXiv preprint arXiv:1510.03009.
[14] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1, arXiv preprint arXiv:1602.02830.
[15] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, Xnor-net: Imagenet classification using binary convolutional neural networks, in: European Conference on Computer Vision, Springer, 2016, pp. 525–542.
[16] T. Posewsky, D. Ziener, Throughput optimizations for fpga-based deep neural network inference, Microprocessors and Microsystems 60 (2018) 151–161.
[17] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting., Journal of Machine Learning Research 15 (1) (2014) 1929–1958.
[18] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, arXiv preprint arXiv:1207.0580.
[19] M. W. Gardner, S. Dorling, Artificial neural networks (the multilayer perceptron)-a review of applications in the atmospheric sciences, Atmospheric environment 32 (14) (1998) 2627–2636.
[20] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[21] L. Medsker, L. Jain, Recurrent neural networks, Design and Applications 5.
[22] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, S. Khudanpur, Recurrent neural network based language model, in: Eleventh Annual Conference of the International Speech Communication Association, 2010.
[23] F. Jelinek, R. L. Mercer, L. R. Bahl, J. K. Baker, Perplexity?a measure of the difficulty of speech recognition tasks, The Journal of the Acoustical Society of America 62 (S1) (1977) S63–S63.
[24] H. Wu, X. Gu, Towards dropout training for convolutional neural networks, Neural Networks 71 (2015) 1–10.
[25] J. Su, D. B. Thomas, P. Y. Cheung, Increasing network size and training throughput of fpga restricted boltzmann machines using dropout, in: 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2016, pp. 48–51.
[26] S. Wang, C. Manning, Fast dropout training, in: Proceedings of the 30th International Conference on Machine Learning (ICML-13), 2013, pp. 118–126.
[27] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, R. Fergus, Regularization of neural networks using dropconnect, in: Proceedings of the 30th international conference on machine learning (ICML-13), 2013, pp. 1058–1066.
[28] S. H. Khan, M. Hayat, F. Porikli, Regularization of deep neural networks with spectral dropout, Neural Networks 110 (2019) 82–90.
[29] S. Sawaguchi, H. Nishi, Slightly-slacked dropout for improving neural network learning on fpga, ICT Express 4 (2) (2018) 75–80.
[30] V. V. Bonde, A. Kale, Design and implementation of a random number generator on fpga, International Journal of Science and Research 4 (5) (2015) 203–208.
[31] Y. J. Yeoh, T. Morie, H. Tamukoh, A hardware-oriented dropout algorithm for efficient fpga implementation, in: International Conference on Neural Information Processing, Springer, 2017, pp. 821–829.
[32] S. Tokui, K. Oono, S. Hido, J. Clayton, Chainer: a next-generation open source framework for deep learning, in: Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS), Vol. 5, 2015.
[33] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images.
[34] Robocup@home, http://www.robocupathome.org/.
[35] @home dataset, https://github.com/hibikino-musashi-athome/rcj2016_object_image_dataset/.
[36] M. Marcus, B. Santorini, M. A. Marcinkiewicz, A. Taylor, Treebank-3 ldc99t42, CD-ROM. Philadelphia, Penn.: Linguistic Data Consortium.

[37] I. Xilinx, Vivado design suite user guide (2014).
[38] M. Bakiri, C. Guyeux, J.-F. Couchot, A. K. Oudjida, Survey on hardware implementation of random number generators on fpga: Theory and experimental analyses, Computer Science Review 27 (2018) 135–153.