Controlled Dropout: a Different Dropout for Improving Training Speed on Deep Neural Network

ByungSoo Ko, Han-Gyu Kim, Ho-Jin Choi
School of Computing, Korea Advanced Institute of Science and Technology (KAIST)
291 Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea
{kobiso, kimhangyu, hojinc}@kaist.ac.kr

***Abstract*—Dropout is a technique widely used for preventing overfitting while training deep neural networks. However, applying dropout to a neural network typically increases the training time. This paper proposes a different dropout approach called controlled dropout that improves training speed by dropping units in a column-wise or row-wise manner on the matrices. In controlled dropout, a network is trained using compressed matrices of smaller size, which results in notable improvement of training speed. In the experiment on feed-forward neural networks for MNIST data set and convolutional neural networks for CIFAR-10 and SVHN data sets, our proposed method achieves faster training speed than conventional methods both on CPU and GPU, while exhibiting the same regularization performance as conventional dropout. Moreover, the improvement of training speed increases when the number of fully-connected layers increases. As the training process of neural network is an iterative process comprising forward propagation and backpropagation, speed improvement using controlled dropout would provide a significantly decreased training time.**

***Index Terms*—Dropout, deep neural network, training speed improvement.**

I. Introduction

Deep learning has been among the hottest topics in the field of science and technology these days because of its outstanding performance in various areas such as automatic machine translation [1], object classification and detection [2], analysis of medical data [3], and so on. However, training a deep neural network might suffer from overfitting problem that occurs when the model memorizes the training data rather than finds out implicit patterns. Moreover, training a deep neural network usually takes a vast amount of time because of the iterative process of forward propagation and backpropagation.

The *dropout* technique [4] was proposed to avoid overfitting by randomly dropping units from neural network during training time. Many network models adopt the dropout technique. However, one of the drawbacks of dropout is that a dropout network typically takes longer time to train in comparison with a standard neural network for the same architecture. Moreover, it results in many zero element multiplications in the matrix computation, which is a redundant computation, as some units are randomly dropped in each layer. From this observation, we consider that the training process will be faster if such redundant matrix computations are eliminated.

Therefore, we propose *controlled dropout*, a different approach to dropout that enhances the training speed. The main idea of controlled dropout is that elements of the matrix are dropped in a column-wise or row-wise manner in the training stage, instead of element-wise random selection. Such an approach enables reforming of matrices in the network into a smaller one by eliminating dropped columns and rows, which helps improve training speed. This paper is an extension of our previous work [5], where the performance improvement is achieved by replacing the block-wise manner with the column-wise or row-wise manner.

II. Related Work

One of the most common problems that arise during deep neural network training is that of overfitting. Overfitting happens when there are too many parameters in a network model in comparison with the size of the training data [6]. A model that has been overfit to the training data generally exhibits poor performance on the test data despite showing good performance on training data. Many regularization techniques have been introduced, to prevent overfitting such as L1 and L2 regularization [7], max-norm constraints [8], and dropout [4].

Among these, dropout is one of the simplest and the most powerful regularization techniques. It prevents units from complex co-adapting by randomly dropping units from the network. The *dropout rate* represents the probability that a unit will be retained. After deciding on the units that are to be dropped using the dropout rate, a thinned network is constructed. Such random unit dropping has the effect of sampling a large number of different thinned networks during the training stage. In the inference stage, a single unthinned network is used, which approximates the effect of averaging the predictions of all thinned networks from the training process. In this case, the weights of the single unthinned network has to be multiplied by the dropout rate, so that the scale of the output at test stage will be the same as that of the output at training stage. In [4], it is suggested that the optimal dropout rate is 0.5 for units in hidden layers and close to 1 for units in input layer.

One of the drawbacks of dropout is that it increases training time. This can be attributed to noisy parameter updates caused when each training step tries to train a different architecture. Moreover, dropout is accompanied by a considerable amount of redundant matrix computations, which are caused by multiplications of zero units. The motivation for our idea comes from this observation.

III. Controlled Dropout

This section describes the concept of the proposed method, controlled dropout, by comparing it with conventional dropout. Furthermore, we suggest an algorithm to construct and train a controlled dropout network and prove its validity. Finally, we analyze the increase in training speed obtained by applying controlled dropout.

A. Concept of Controlled Dropout

Generally, artificial neural networks are trained using mini-batch algorithms, an effective technique used to speed-up convex optimization problems. In order to take the concept of mini-batch into consideration in forward propagation, consider a standard neural network with L hidden layers and let $l \in \{1, ..., L\}$ denote the index of the hidden layers. Let $\mathbf{Z}^{(l)}$ represent the input matrix into layer l of a mini-batch, and $\mathbf{Y}^{(l)}$ be the matrix of a mini-batch obtained by applying activation function and dropout on layer l. $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and the bias vector of layer l, respectively. The forward propagation process can be described as follows.

$$\mathbf{Z}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{Y}^{(l)} + \mathbf{b}^{(l+1)}\mathbf{e}_{1\times n},$$
$$\mathbf{Y}^{(l+1)} = \mathbf{R}^{(l+1)} \circ f(\mathbf{Z}^{(l+1)}),$$

where $\mathbf{e}_{1\times n}$ is all-one vector of size $1 \times n$, n is batch size, $f(\cdot)$ denotes an activation function, such as sigmoid, hyperbolic tangent, and rectified linear unit, the operator $\circ$ denotes element-wise multiplication, and $\mathbf{R}^{(l+1)}$ is the indicator matrix for dropout whose (i, j) element equals zero if the i-th unit in j-th instance of mini-batch is dropped and equals one otherwise. $\mathbf{R}^{(l+1)}$ is decided either by the conventional dropout or the proposed controlled dropout.

Considering the mini-batch, there will be many zero elements generated by dropout in a disorderly manner in $\mathbf{Y}^{(l)}$. Compared to the conventional dropout, the controlled dropout drops units in a column-wise or row-wise manner that are randomly decided at each layer. Specifically, all instances in a mini-batch should drop the same unit in our proposed controlled dropout. The number of columns or rows that must be active is decided by the dropout rate.

In controlled dropout, parts of matrices that belong to redundant multiplication are omitted to construct a compressed network. Figure 1 is an example of matrix compression using controlled dropout at the forward propagation step. The example describes the matrix computation from layer l to $l+1$, where layer l has $m^{(l)}$ units, layer $l+1$ has $m^{(l+1)}$ units, n is the size of one mini-batch, and $\mathbf{R}^{(l)}$ denotes the indicator matrix for dropout on layer l with a dropout rate of 0.5. There are two points to be noted in this example. First, the white coloured elements in $\mathbf{Y}^{(l)}$ are dropped and elements with hatched pattern in $\mathbf{W}^{(l+1)}$ will always be multiplied by them. Second, if we know that white coloured elements in $\mathbf{Y}^{(l+1)}$ are going to be dropped in the next layer, the elements with grey coloured elements in $\mathbf{W}^{(l+1)}$ and $\mathbf{b}^{(l+1)}$ may be omitted in the forward propagation step as they will not affect the final forward propagation result. Considering these two points, the elements that are not necessary for the forward propagation, represented as squares with hatched patterns or those in grey or white color in Figure 1, may be eliminated from every matrix to get a compressed matrix computation.

B. Learning Controlled Dropout Networks

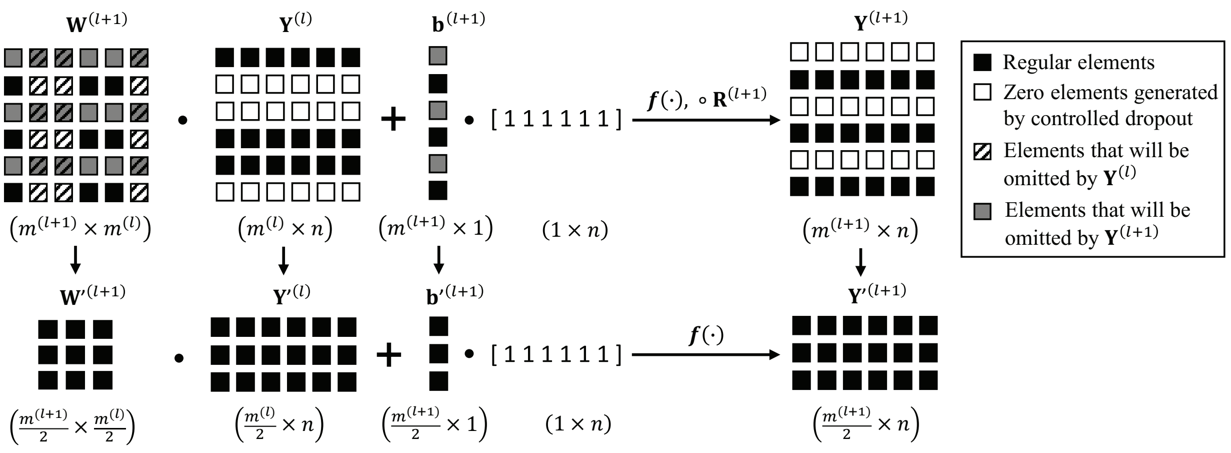
The training process for the network with controlled dropout is described in algorithm 1. The main idea of this algorithm is to train the network with compressed parameters $\{\mathbf{W}^{(c)}, \mathbf{b}^{(c)}\}$ and test the network with original parameters $\{\mathbf{W}^{(o)}, \mathbf{b}^{(o)}\}$. In the training step of controlled dropout, the decision on which columns or rows would be retained for each layer is made in the first step of the iterative training before the forward propagation as described in line 6. Then, $\{\mathbf{W}^{(c)}, \mathbf{b}^{(c)}\}$ are assigned with the selected rows and columns of $\{\mathbf{W}^{(o)}, \mathbf{b}^{(o)}\}$ for each layer as described in line 7 to 9. After that, the forward propagation, the backpropagation and the parameter update are processed on the compressed parameters. In the final step of the iterative training, the previously-selected rows and columns of $\{\mathbf{W}^{(o)}, \mathbf{b}^{(o)}\}$ are assigned with the updated $\{\mathbf{W}^{(c)}, \mathbf{b}^{(c)}\}$, as explained in line 14 to 16. By repeating these steps, the controlled dropout will dramatically reduce the training time of the neural network by training compressed network in each iteration. In the inference stage, the network with the original parameters $\{\mathbf{W}^{(o)}, \mathbf{b}^{(o)}\}$ is used. The weights of the network have to be multiplied by the dropout rate in the inference stage just like in conventional dropout, to guarantee that the scales of the activations remain the same in the training and test step.

Algorithm 1 Training process of controlled dropout networks

```
1: procedure TRAIN A NETWORK()
2:     Initialize $\mathbf{W}_l^{(o)}$ for each layer $l$
3:     Initialize $\mathbf{b}_l^{(o)}$ for each layer $l$
4:     for $step = 1 \ldots last\_step$ do
5:         Get mini-batch data for one training iteration
6:         $indices_l \leftarrow$ Randomly selected indices
               to retain during the dropout for layer $l$
7:         Assign $\mathbf{W}^{(c)}$ and $\mathbf{b}^{(c)}$ for each layer $l$:
8:             $\mathbf{W}_l^{(c)} \leftarrow \mathbf{W}_l^{(o)}[indices_l, indices_{l+1}]$
9:             $\mathbf{b}_l^{(c)} \leftarrow \mathbf{b}_l^{(o)}[indices_l]$
10:        Compute $\{\triangle\mathbf{W}_l^{(c)}, \triangle\mathbf{b}_l^{(c)}\}$ by forward propagation
               and backpropagation for each layer $l$
11:        Update $\mathbf{W}^{(c)}$ and $\mathbf{b}^{(c)}$ for each layer $l$:
12:            $\mathbf{W}_l^{(c)} \leftarrow \mathbf{W}_l^{(c)} - \triangle\mathbf{W}_l^{(c)}$
13:            $\mathbf{b}_l^{(c)} \leftarrow \mathbf{b}_l^{(c)} - \triangle\mathbf{b}_l^{(c)}$
14:        Assign $\mathbf{W}^{(o)}$ and $\mathbf{b}^{(o)}$ for each layer $l$:
15:            $\mathbf{W}_l^{(o)}[indices_l, indices_{l+1}] \leftarrow \mathbf{W}_l^{(c)}$
16:            $\mathbf{b}_l^{(o)}[indices_l] \leftarrow \mathbf{b}_l^{(c)}$
17:    end for
18: end procedure
```

C. Proof of Forward Propagation

In this section, a proof that forward propagation with compressed parameters would have the same result with original



$\mathbf{W}^{(l+1)}$
$\mathbf{Y}^{(l)}$
$\mathbf{b}^{(l+1)}$
$\mathbf{Y}^{(l+1)}$
$[1\,1\,1\,1\,1\,1]$
$f(\cdot), \circ\mathbf{R}^{(l+1)}$
$(m^{(l+1)} \times m^{(l)})$
$(m^{(l)} \times n)$
$(m^{(l+1)} \times 1)$
$(1 \times n)$
$(m^{(l+1)} \times n)$
■ Regular elements
□ Zero elements generated by controlled dropout
▨ Elements that will be omitted by $\mathbf{Y}^{(l)}$
■ Elements that will be omitted by $\mathbf{Y}^{(l+1)}$
$\mathbf{W}'^{(l+1)}$
$\mathbf{Y}'^{(l)}$
$\mathbf{b}'^{(l+1)}$
$\mathbf{Y}'^{(l+1)}$
$f(\cdot)$
$\left(\frac{m^{(l+1)}}{2} \times \frac{m^{(l)}}{2}\right)$
$\left(\frac{m^{(l)}}{2} \times n\right)$
$\left(\frac{m^{(l+1)}}{2} \times 1\right)$
$\left(\frac{m^{(l+1)}}{2} \times n\right)$

Fig. 1. An example of matrix compression by controlled dropout.

parameters is given. Let $(k,i),(i,j),(k,1),(k,j)$ denote row and column indices of $\mathbf{W}^{(l+1)},\mathbf{Y}^{(l)},\mathbf{b}^{(l+1)},\mathbf{Y}^{(l+1)}$, respectively. i' and k' denote row indices of compressed matrices corresponding to retained row i and k of original matrices, respectively. Compression rule we designed for $i \in A^{(l)}, k \in A^{(l+1)}$ is that

$$\begin{aligned} \mathbf{W}^{(l+1)}_{ki} &= \mathbf{W}'^{(l+1)}_{k'i'}, \\ \mathbf{b}^{(l+1)}_{k} &= \mathbf{b}'^{(l+1)}_{k'}, \\ \mathbf{R}^{(l+1)} &= \mathbf{r}^{(l+1)}\mathbf{e}_{1\times n}. \end{aligned} \tag{1}$$

Our proof is made via mathematical induction. Let us assume following equation to be true

$$\mathbf{Y}^{(l)}_{ij} = \mathbf{Y}'^{(l)}_{i'j} \quad \texttt{for} \quad i \in A^{(l)}. \tag{2}$$

We should show that the following equation is satisfied,

$$\mathbf{Y}^{(l+1)}_{kj} = \mathbf{Y}'^{(l+1)}_{k'j} \quad \texttt{for} \quad k \in A^{(l+1)},$$

and we should also prove that $\mathbf{Y}^{(l)}_{ij}\left(i \notin A^{(l)}\right)$ is not necessary to compute $\mathbf{Y}^{(l+1)}_{kj}$. In the basis step of the proof, Equation 2 is true when $l=1$ since both $\mathbf{Y}^{(1)}$ and $\mathbf{Y}'^{(1)}$ are exactly the same matrix. In the induction step, $\mathbf{Y}^{(l+1)}_{kj}$ can be derived by

$$\begin{aligned} \mathbf{Y}^{(l+1)}_{kj} &= [\mathbf{R}^{(l+1)} \circ f(\mathbf{W}^{(l+1)}\mathbf{Y}^{(l)} + \mathbf{b}^{(l+1)}\mathbf{e}_{1\times n})]_{kj} \\ &= \mathbf{r}^{(l+1)}_{k} f\left(\left[\sum_{i} \mathbf{W}^{(l+1)}_{ki}\mathbf{Y}^{(l)}_{ij} + \mathbf{b}^{(l+1)}_{k}\right]_{kj}\right) \\ &= f\left(\left[\sum_{i\in A^l} \mathbf{W}^{(l+1)}_{ki}\mathbf{Y}^{(l)}_{ij} + \mathbf{b}^{(l+1)}_{k}\right]_{kj}\right). \end{aligned} \tag{3}$$

$\mathbf{Y}'^{(l+1)}_{k'j}$ can be derived with the same derivation:

$$\mathbf{Y}'^{(l+1)}_{k'j} = f\left(\left[\sum_{i'} \mathbf{W}'^{(l+1)}_{k'i'}\mathbf{Y}'^{(l)}_{i'j} + \mathbf{b}'^{(l+1)}_{k'}\right]_{k'j}\right).$$

With the assumption in Equation 2 and the compression rule in Equation 1, it is obvious that

$$\begin{aligned} Z^{(l+1)}_{kj} &= \left[\sum_{i\in A^l} \mathbf{W}^{(l+1)}_{ki}\mathbf{Y}^{(l)}_{ij} + \mathbf{b}^{(l+1)}_{k}\right]_{kj} \\ &= \left[\sum_{i'} \mathbf{W}'^{(l+1)}_{k'i'}\mathbf{Y}'^{(l)}_{i'j} + \mathbf{b}'^{(l+1)}_{k'}\right]_{k'j} = Z'^{(l+1)}_{k'j}. \end{aligned} \tag{4}$$

Thus, $\mathbf{Y}^{(l+1)}_{kj} = \mathbf{Y}'^{(l+1)}_{k'j}$ for $k \in A^{(l+1)}$. And it is also shown in Equation 3 that, $\mathbf{Y}^{(l)}_{ij}\left(i \notin A^{(l)}\right)$ does not affect $\mathbf{Y}^{(l+1)}_{kj}$.

D. Proof of Backpropagation

In this section, a proof that backpropagation with compressed parameters would give the same result with original parameters is provided. To construct the proof via mathematical induction, let us assume following equation to be true

$$\frac{\partial E}{\partial \mathbf{Y}^{(l+1)}_{kj}} = \frac{\partial E}{\partial \mathbf{Y}'^{(l+1)}_{k'j}} \quad \texttt{for} \quad k \in A^{(l+1)}. \tag{5}$$

From the assumption, we should show for $i \in A^{(l)}$ that

$$\frac{\partial E}{\partial \mathbf{Y}^{(l)}_{ij}} = \frac{\partial E}{\partial \mathbf{Y}'^{(l)}_{i'j}}, \tag{6}$$

$$\frac{\partial E}{\partial \mathbf{W}^{(l+1)}_{ki}} = \frac{\partial E}{\partial \mathbf{W}'^{(l+1)}_{k'i'}}, \tag{7}$$

$$\frac{\partial E}{\partial \mathbf{b}^{(l+1)}_{k}} = \frac{\partial E}{\partial \mathbf{b}'^{(l+1)}_{k'}}. \tag{8}$$

In the basis step of the proof, Equation 5 is true when l is the last layer L of the network since both $\mathbf{Y}^{(L)}$ and $\mathbf{Y}'^{(L)}$

would be exactly the same. In the induction step of $\mathbf{Y}$, $\frac{\partial E}{\partial \mathbf{Y}_{ij}^{(l)}}$ can be derived as follows,

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{Y}_{ij}^{(l)}} &= \sum_k \frac{\partial E}{\partial \mathbf{Y}_{kj}^{(l+1)}} \frac{\partial \mathbf{Y}_{kj}^{(l+1)}}{\partial \mathbf{Y}_{ij}^{(l)}} \\
&= \sum_k \frac{\partial E}{\partial \mathbf{Y}_{kj}^{(l+1)}} \frac{\partial [\mathbf{R}^{(l+1)} \circ f(\mathbf{Z}^{(l+1)})]_{kj}}{\partial \mathbf{Y}_{ij}^{(l)}} \\
&= \sum_{k \in A^{(l+1)}} \frac{\partial E}{\partial \mathbf{Y}_{kj}^{(l+1)}} \frac{\partial f(\mathbf{Z}_{kj}^{(l+1)})}{\partial \mathbf{Z}_{kj}^{(l+1)}} \mathbf{W}_{ki}^{(l+1)}. \quad (9)
\end{aligned}$$

Besides, $\frac{\partial E}{\partial \mathbf{Y'}_{i'j}^{(l)}}$ can also be achieved with the same derivation:

$$\frac{\partial E}{\partial \mathbf{Y'}_{i'j}^{(l)}} = \sum_{k'} \frac{\partial E}{\partial \mathbf{Y'}_{k'j}^{(l+1)}} \frac{\partial f(\mathbf{Z'}_{k'j}^{(l+1)})}{\partial \mathbf{Z'}_{k'j}^{(l+1)}} \mathbf{W'}_{k'i'}^{(l+1)}.$$

With the assumption in Equation 5, the compression rule in Equation 1, and Equation 4, Equation 6 is satisfied obviously. It is also shown in Equation 9 that, $\frac{\partial E}{\partial \mathbf{Y}_{kj}^{(l+1)}} \left(k \notin A^{(l+1)}\right)$ is not necessary in computing $\frac{\partial E}{\partial \mathbf{Y}_{ij}^{(l)}}$.

In the induction step of $\mathbf{W}$, $\frac{\partial E}{\partial \mathbf{W}_{ki}^{(l+1)}}$ can be derived as

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{W}_{ki}^{(l+1)}} &= \sum_j \frac{\partial E}{\partial \mathbf{Y}_{kj}^{(l+1)}} \frac{\partial \mathbf{Y}_{kj}^{(l+1)}}{\partial \mathbf{W}_{ki}^{(l+1)}} \\
&= \sum_j \frac{\partial E}{\partial \mathbf{Y}_{kj}^{(l+1)}} \frac{\partial [\mathbf{R}^{(l+1)} \circ f(\mathbf{Z}^{(l+1)})]_{kj}}{\partial \mathbf{W}_{ki}^{(l+1)}} \\
&= \sum_j \frac{\partial E}{\partial \mathbf{Y}_{kj}^{(l+1)}} \frac{\partial f(\mathbf{Z}_{kj}^{(l+1)})}{\partial \mathbf{Z}_{kj}^{(l+1)}} \mathbf{Y}_{ij}^{(l)}. \quad (10)
\end{aligned}$$

$\frac{\partial E}{\partial \mathbf{W'}_{k'i'}^{(l+1)}}$ can also be derived similarly:

$$\frac{\partial E}{\partial \mathbf{W'}_{k'i'}^{(l+1)}} = \sum_j \frac{\partial E}{\partial \mathbf{Y'}_{k'j}^{(l+1)}} \frac{\partial f(\mathbf{Z'}_{k'j}^{(l+1)})}{\partial \mathbf{Z'}_{k'j}^{(l+1)}} \mathbf{Y'}_{i'j}^{(l)}.$$

With the assumption in Equation 5, the compression rule in Equation 1, and Equation 4, Equation 7 is true obviously. It is also shown in Equation 10 that, $\frac{\partial E}{\partial \mathbf{Y}_{kj}^{(l+1)}} \left(k \notin A^{(l+1)}\right)$ is not necessary for $\frac{\partial E}{\partial \mathbf{W}_{ki}^{(l+1)}}$, as $\frac{\partial E}{\partial \mathbf{W}_{ki}^{(l+1)}} = 0$ when $k \notin A^{(l+1)}$. Equation 8 can be proved in exactly the same way of proving Equation 7. In conclusion, computing gradient for dropped rows is not necessary as they do not affect the backpropagation process at all.

E. Training Speed Analysis

In order to analyze the training speed of controlled dropout networks, we approximate training time using major factors that affect training speed more than other factors. In the following equations, $T^{(o)}$ and $T^{(c)}$ represent the training time of standard neural network and the network with controlled dropout, respectively. T_m, and T_a are the time cost for one scalar multiplication, and one scalar addition. N_s, N_m, N_a, and N_l are the number of iterations, multiplication, addition, and layers, respectively. The function $\mathtt{num}(\cdot)$ returns the number of scalar computations of the matrix multiplication.

The training time of a network consists of computation time for multiplications and additions. Such computations repeat for N_s iterations, and repeat twice for each iteration, once each for forward propagation and backpropagation. Considering such iterative computations, the training time of a neural network without controlled dropout can be described by Equation 11. As the time complexity of addition is much smaller than that of multiplication [9], Equation 11 can be approximated by Equation 12. $N_m^{(o)}$ can be expressed using the function $\mathtt{num}(\cdot)$ and the dimensions of the matrices in Figure 1, which leads to Equation 13 by discarding multiplication of bias vector as it is much smaller than multiplication of matrices. The final training time without controlled dropout can be expressed as Equation 14.

$$\begin{aligned}
T^{(o)} =& 2N_s(N_m^{(o)}T_m + N_a^{(o)}T_a) && (11) \\
\approx& 2N_s\left(N_m^{(o)}T_m\right) && (12) \\
\approx& 2N_sT_m \sum_l^{N_l} \mathtt{num}\left((m^{(l+1)} \times m^{(l)})(m^{(l)} \times n)\right) && (13) \\
\approx& 2nN_sT_m \sum_l^{N_l} m^{(l+1)}m^{(l)}. && (14)
\end{aligned}$$

The training time of controlled dropout networks can be analyzed in a manner similar to that of standard neural networks. One difference is that controlled dropout networks take additional time for updating parameters T_u. Another is that the dimensions of the matrices are reduced by the dropout rate p. Equation 16 is approximated from Equation 15 by discarding the time required for addition and parameter update as memory access for parameter update takes insignificant time. Similar to the derivation of $T^{(o)}$, $T^{(c)}$ can be finally arranged as Equation 17. From this it is shown that controlled dropout networks takes much less training time than standard neural networks because the square of the dropout rate p is smaller than 1.

$$\begin{aligned}
T^{(c)} =& 2N_s(N_m^{(c)}T_m + N_a^{(c)}T_a + T_u) && (15) \\
\approx& 2N_s(N_m^{(c)}T_m) && (16) \\
\approx& 2N_sT_m \sum_l^{N_l} \mathtt{num}\left((pm^{(l+1)} \times pm^{(l)})(pm^{(l)} \times n)\right) && \\
\approx& p^2 2nN_sT_m \sum_l^{N_l} m^{(l+1)}m^{(l)} \quad = p^2T^{(o)}. && (17)
\end{aligned}$$

IV. Experimental Results

In this section, we discuss experimental results of the proposed controlled dropout technique by comparing it with the standard neural network and the network with conventional dropout for different data sets. In order to get reasonable comparison between conventional dropout and controlled dropout, all networks in our experiment were implemented based on the source code of *deepnet*[1] [4]. We conducted experiments by implementing feed-forward neural networks (FFNNs) and convolutional neural networks (CNNs) on both CPU and GPU.

In our experiments, we focus on two points. Specifically, the modification of the dropout in our proposed method must not affect the regularization performance compared to the conventional dropout and the proposed method works much faster than the conventional dropout does.

A. *Network Structures and Data Sets*

In order to test the regularization performance and training speed on FFNN models, we used three different models and the Mixed National Institute of Standards and Technology (MNIST) data set which is a standard data set of handwritten digits [10]. The first model is a standard FFNN without dropout, which has 3 hidden layers of 1,024 units followed by the rectified linear activation function in each layer; the second one is the same network with dropout; and the third one is the same network with controlled dropout. For the dropout and the controlled dropout networks, we used dropout for all layers with dropout rate $p = 0.5$ for hidden layers and $p = 0.8$ for the input layer.

Furthermore, we conducted experiments for CNNs with the Canadian Institute for Advanced Research (CIFAR-10) data set which comprises labeled subsets of 80 million tiny images dataset [11] and the Street View House Numbers (SVHN) data set which comprises images of house numbers collected by Google Street View [12]. We used three different CNNs as we did in the MNIST experiment. The standard architecture had three convolutional layers composed of 96, 128, and 256 filters respectively, and each convolutional layer was followed by a max-pooling layer. Each convolutional layer was applied to a 5×5 receptive field with a stride of one pixel and max-pooling layer pools 3×3 regions at strides of two pixels. We placed two fully connected hidden layers over the convolutional layers, each of which had 2,048 units. This was followed by the rectified linear activation function. For the dropout network, we applied dropout rates of $0.9, 0.75, 0.75, 0.5$, and 0.5 to the hidden layers respectively. The same dropout rates were used in the controlled dropout network where the controlled dropout was only applied to the fully connected hidden layers. The same CNN architectures were used for both CIFAR-10 and SVHN.

B. *Results*

1) Regularization Performance: In our experiment, we expect that controlled dropout will improve training speed while regularization performance remains as good as conventional dropout. Figure 2 shows the test error rates obtained for different network architectures as training progresses with $1,000,000$ iterations for FFNN and $300,000$ iterations for CNN. The details of test error rates for each data set are shown in Table I. In Figure 2a, the network with dropout and that with controlled dropout show similar performance while they have significantly lower test errors compared to the network without dropout. Moreover, Figures 2b and 2c also show that the classification errors of dropout and controlled dropout are similar and both of them are lower than the error of the network without dropout. Such results implies that the controlled dropout works as good as the conventional dropout for regularization purposes.

2) Training Speed: In order to test the amount of speed improvement that can be obtained, classification experiments were undertaken that measure the training time for each case. Table I shows the training time taken for each data set and each network architecture. In all cases, the controlled dropout network provided the fastest result and the dropout network was slightly slower than the network without dropout. Especially, the MNIST experiment on CPU with controlled dropout network showed 54.42% of speed improvement with $37,636.44$ sec different training time, which is more than 10 hours, when compared with the dropout network. The speed improvement of the MNIST experiment on GPU with controlled dropout was 19.96%, which was 622.4 sec faster than dropout network. Since the MNIST experiments were conducted on FFNN and the size of the controlled dropout network was almost half of the dropout network owing to the dropout rate of 0.5 on each hidden layers, it showed significant improvement of training speed, especially on the CPU. Experiments with Cifar10 and SVHN on GPU showed similar result on each of the networks, as the architecture of CNN was the same for both cases. They showed 6.27% and 7.72% of speed improvement which is 982.39 sec and $1,318.89$ sec training time difference between controlled dropout and dropout network. As both experiments used CNN and most of the training time depended on training convolutional layers, speed improvement of controlled dropout was not significant compared with that of MNIST experiments.

TABLE I
TEST ERROR AND TRAINING TIME

	Without Dropout	**Dropout**	**Controlled Dropout**
MNIST on CPU			
Error (%)	1.77	1.11	1.12
Time (sec)	68,619.29	69,159.77	31,523.33
MNIST on GPU			
Error (%)	1.75	1.12	1.10
Time (sec)	2,841.05	3,116.84	2,494.44
CIFAR-10 on GPU			
Error (%)	17.53	15.23	15.27
Time (sec)	15,659.11	15,661.19	14,678.80
SVHN on GPU			
Error (%)	5.31	3.35	3.37
Time (sec)	16,623.96	17,079.10	15,760.21

To analyze the improvement rate of the training speed, we conducted experiments by differentiating the number of hidden layers for each case and the experimental results are shown in Table II. Controlled dropout was used on every layer of FFNN and only used on fully connected layers for CNNs. The

[1]https://github.com/nitishsrivastava/deepnet

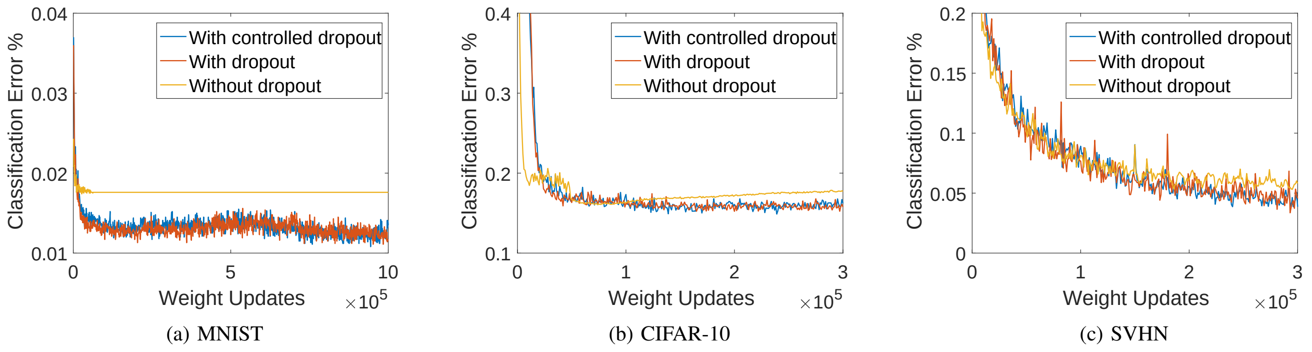


Fig. 2. Test error on each dataset for different network architectures.

architecture and parameters of each network were the same as that for previous experiments, except for the number of fully-connected hidden layers. Table II shows that the improvement rate of training speed on every case increased when the number of hidden layers increased. CNN on GPU showed less improvement than FFNN since controlled dropout was only used on fully connected layers while most of the training time could be accounted on computing convolutional layers. As the FFNN can be trained with compressed parameters on every layer, it showed significant speed improvement, especially on the CPU which is larger than 50% when there are more than two hidden layers. As the training process of neural network is an iterative process comprising forward propagation and backpropagation, which cost a lot of time, even a small percentage of speed improvement with controlled dropout would help save considerable time in training.

TABLE II
PERCENTAGE(%) OF SPEED IMPROVEMENT

Processor	Model	The Number of Hidden Layers				
		1	**2**	**3**	**4**	**5**
CPU	**FFNN**	23.48	50.52	54.91	56.50	57.83
GPU	**FFNN**	7.87	11.97	19.19	20.12	21.75
	CNN	1.88	3.79	6.43	8.89	9.00

V. Conclusion

This paper proposed a different dropout approach called controlled dropout that improves the training speed of deep neural network by dropping units in a column-wise or row-wise manner. In controlled dropout, a network was trained with compressed parameters achieved by eliminating dropped units to obtain an improved training speed. We showed that regularization performance of controlled dropout was as good as conventional dropout on both FFNN and CNN. The training speed improvement of the proposed technique increased when the number of fully-connected layers increased. Moreover, using controlled dropout on a CPU provided significant speed improvement than that obtained on a GPU, and an FFNN exhibited better speed improvement when compared to that obtained on a CNN. Overall, controlled dropout considerably improved training speed of both FFNN and CNN.

In the future, we plan to investigate the use of controlled dropout on various neural network models, such as convolutional layers on CNN, a recurrent neural network, a deep belief network, and so on. Furthermore, we will use various data sets to find out whether controlled dropout is applicable to a wide variety of domains.

Acknowledgment

This work was funded by the Korea Meteorological Administration Research and Development Program under Grant KMIPA2017-5030.

References

[1] J. Zhang, C. Zong *et al.*, "Deep neural networks in machine translation: An overview." *IEEE Intelligent Systems*, vol. 30, no. 5, pp. 16–25, 2015.

[2] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Advances in Neural Information Processing Systems*, 2013, pp. 2553–2561.

[3] H.-G. Kim, G.-J. Jang, H.-J. Choi, M. Kim, Y.-W. Kim, and J. Choi, "Medical examination data prediction using simple recurrent network and long short-term memory," in *Proceedings of the Sixth International Conference on Emerging Databases: Technologies, Applications, and Theory*. ACM, 2016, pp. 26–34.

[4] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[5] B. Ko, H.-G. Kim, K.-J. Oh, and H.-J. Choi, "Controlled dropout: A different approach to using dropout on deep neural network," in *Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*. IEEE, 2017, pp. 358–362.

[6] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[7] Z. Xu, X. Chang, F. Xu, and H. Zhang, "L1/2 regularization: A thresholding representation theory and a fast solver," *IEEE Transactions on neural networks and learning systems*, pp. 1013–1027, 2012.

[8] B. Neyshabur, R. Tomioka, and N. Srebro, "Norm-based capacity control in neural networks." in *COLT*, 2015, pp. 1376–1401.

[9] L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and real computation*. Springer Science & Business Media, 2012.

[10] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," 1998.

[11] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[12] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2, 2011, p. 5.