Controlled Dropout: a Different Approach to Using Dropout on Deep Neural Network

ByungSoo Ko, Han-Gyu Kim, Kyo-Joong Oh, Ho-Jin Choi
Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST)
291 Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea
kobiso, kimhangyu, aomaru, hojinc@kaist.ac.kr

***Abstract*—Deep neural networks (DNNs), which show outstanding performance in various areas, consume considerable amounts of memory and time during training. Our research led us to propose a controlled dropout technique with the potential of reducing the memory space and training time of DNNs. Dropout is a popular algorithm that solves the overfitting problem of DNNs by randomly dropping units in the training process. The proposed controlled dropout intentionally chooses which units to drop compared to conventional dropout, thereby possibly facilitating a reduction in training time and memory usage. In this paper, we focus on validating whether controlled dropout can replace the traditional dropout technique to enable us to further our research aimed at improving the training speed and memory efficiency. A performance comparison between controlled dropout and traditional dropout is carried out by implementing an image classification experiment on data comprising handwritten digits from the MNIST dataset (Mixed National Institute of Standards and Technology dataset). The experimental results show that the proposed controlled dropout is as effective as traditional dropout. Furthermore, the experimental result implies that controlled dropout is more efficient when an appropriate dropout rate and number of hidden layers are used.**

***Keywords*—dropout; neural network; deep learning**

I. Introduction

Deep learning has become increasingly popular even for ordinary people after the Google DeepMind Challenge Match[1] between AlphaGo and Lee Sedol. The event showed people that deep learning technology is no longer simply a theory but a new way to change our lives. There has been a considerable amount of research in this field to adjust the technology to enable it to be used in practical applications. Examples include the analysis of medical data for more accurate and faster diagnoses [1][2], speech recognition for AI assistants [3], and image classification for self-driving cars [4].

Even though a deep neural network is a very powerful machine-learning technique, it has always been adversely affected by the overfitting problem [5][6], which continues to remain a challenge in this field. To address this problem, Srivastava *et. al.* [7] suggested a simple regularization technique named "Dropout". The main idea of this technique is to randomly drop units from the neural network during training. However, since the removal of units by way of dropout necessitates the replacement of many elements in a matrix with zero elements, it incurs redundant matrix computations.

[1]https://deepmind.com/research/alphago/

We attempted to eliminate these redundant computations after dropout by considering a new approach to using dropout, namely to drop units out non-randomly. A decision as to which units to drop out would enable us to arrange matrices in the network into an organized form to increase the possibility of reducing redundant matrix computations. Reducing these redundant computations is expected to increase both the computational and memory efficiency, which is necessary for training deep neural networks that are generally memory and time consuming. In this paper, we do not suggest how to reduce redundant matrix computations; instead, we propose a method to implement controlled dropout and validate whether it can replace traditional dropout.

The remainder of this paper is structured as follows. In the next section, we introduce the basic architecture of artificial neural networks and the traditional dropout regularization method. Section 3 describes two approaches to the controlled dropout method in detail. In Section 4, we present the implementation of controlled dropout and the experimental results of its evaluation. Finally, we conclude this paper in Section 5 and discuss future work.

II. Background

A. Artificial Neural Network

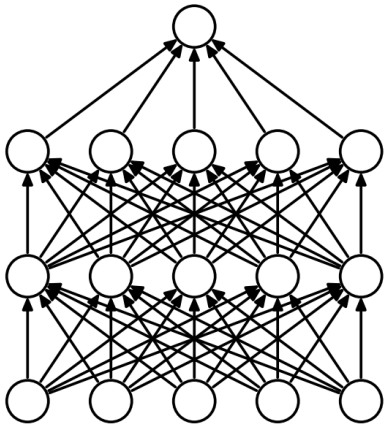

Fig. 1. Standard neural network with two hidden layers [7]

A simple model of an artificial neural network consists of an input layer, one or several hidden layers, and an output layer

when every layer has multiple neurons [6] as shown in Fig. 1. Each connection between neurons is associated with weight and bias. In the forward propagation step, with activations $a^{(l)}$ of the given layer l, we can compute activations $a^{(l+1)}$ of the layer $l+1$ as below.

$$z^{(l+1)} = a^{(l)}W^{(l)} + b^{(l)} \tag{1}$$

$$a^{(l+1)} = f(z^{(l+1)}), \tag{2}$$

where $W^{(l)}$ is a weight matrix, $b^{(l)}$ is a bias vector in the layer l, and the function f is an activation such as a sigmoid function, rectified linear unit, or softmax function.

In the back propagation step, we can train the neural network using gradient descent. The goal of this step is to minimize the cost function $J(W, b)$, such as cross entropy for the classification problem [8] and mean-squared error for the regression problem [9]. Every iteration of gradient descent updates the parameters W, b as below.

$$W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) \tag{3}$$

$$b_i^{(l)} \leftarrow b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b), \tag{4}$$

where $W_{ij}^{(l)}$ denotes an element of the weight matrix associated with the connection between unit j in layer l and unit i in layer $l+1$. Further, $b_i^{(l)}$ is an element of the bias vector associated with unit i in layer $l+1$ and α is the learning rate.

The construction of a deep neural network usually results in memory shortage and speed problems since the network has to save huge matrices for subsequent computation. As the amount of GPU memory or RAM memory is limited, the size of the network usually needs to be bounded [10]. In addition, because of the iterative process associated with forward and backward propagation, the network model has to be optimized for efficiency. In order to address this problem, some researchers used model parallelism and data parallelism with multiple GPUs [11] [12]. This approach is effective if sufficient hardware is available, but not everyone can afford a high-performance GPU.

B. Dropout

Overfitting involves the use of models or procedures that breach parsimony. This occurs when the model includes more terms than necessary or uses more complicated approaches than necessary [13]. Especially with limited data, many of these complicated relationships between training data are likely to be the result of sampling noise, and they would exist in the training set rather than in the real test data. This is a serious problem in the neural network model, and leads to a highly accurate training set but a low-accuracy test set.

Srivastava *et al.* [7] proposed a simple regularization technique named "Dropout" to solve the overfitting problem. This regularization technique prevents overfitting and also improves the performance by approximately combining exponentially many different neural network models efficiently. It randomly chooses units to drop out and removes them from the layer temporarily as Fig. 2 shows an example of a thinned net after applying dropout on the network from Fig. 1. Each unit has a fixed probability p termed the dropout rate, where p can be determined using a validation set or can simply be set at 0.5 for the hidden layers and assigned a number close to 1 for the input layer. Applying the dropout technique is only efficient for the training step, whereas for the test step, it is necessary to use a single neural net without dropout.

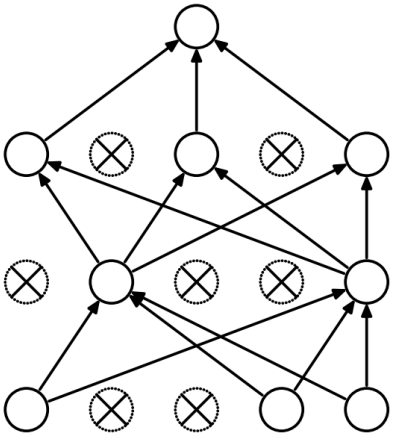

Fig. 2. Example of a thinned net after applying dropout [7]

The use of dropout generates many zero elements in each hidden layer. The expectation of the number of zero elements after dropout is $(1/2)mn$ when n hidden layers exist, in which case each layer has m units and the dropout rate is 0.5, which means almost half of the elements exists. For example, if there are 10 hidden layers, 500 units for each layer and, subsequently, 2,500 zero elements are generated. These zeros incur redundant computation on matrix multiplication.

III. Proposed method

Training a neural network consists of storing large-sized matrices and performing computations among them, which requires a considerable amount of memory space and computational power. After applying dropout, it causes many zeros in a matrix and this can lead to redundant computation since any numerical product with zero would be zero. This motivated us to consider dropping units intentionally with the aim of obtaining an organized form of matrix and reduce matrix computation by dumping the redundant part of the matrix. Before we suggest a way to reduce the size of matrices and their computation, we need to validate whether dropping units intentionally would be as effective as traditional dropout.

Fig. 3 shows an example of matrix multiplication while training a network model without dropout using (1) excluding the addition of a bias vector. All $W^{(l)}$, $a^{(l)}$, and $z^{(l+1)}$ matrices have a dimension of (6×6) and each small black square represents an regular element of each matrix. We assume that this model does not use the dropout technique; thus, $a^{(l)}$ is filled with black squares.

Fig. 4 describes an example of matrix multiplication after applying traditional dropout with a dropout rate of 0.5. In $a^{(l)}$,

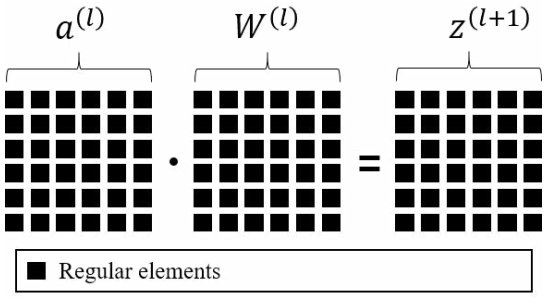

Fig. 3. Example of matrix multiplication without dropout

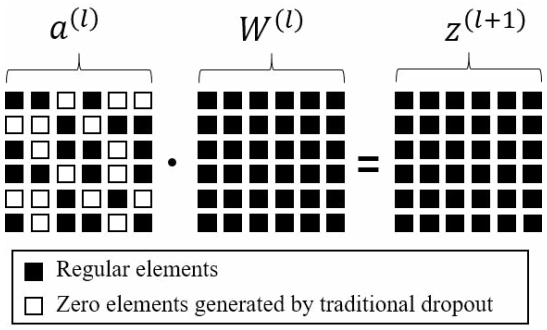

Fig. 4. Example of matrix multiplication after applying dropout

unlike with $a^{(l)}$ in Fig. 3, it contains white squares representing zero elements, which occurs as a result of dropout. The elements in $W^{(l)}$ can be multiplied with the zero elements in the white squares and with regular elements. Since the dropout technique drops units randomly, we cannot predict which unit will be dropped and what kind of shape the matrix will have.

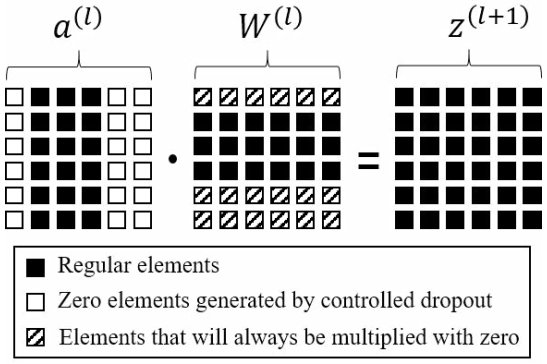

Fig. 5. Example of matrix multiplication after applying controlled dropout

An example of matrix multiplication after applying our proposed controlled dropout technique, with a dropout rate of 0.5, is shown in Fig. 5. We intentionally drop units occurring from index (1, 1) to (6, 1) and from (1, 5) to (6, 6) on the matrix $a^{(l)}$ to obtain a well-formed rectangle consisting of only black squares from index (1, 2) to (6, 4). This rectangle is placed randomly for each computation and layer in order to obtain the randomness exhibited by traditional dropout. Moreover, $W^{(l)}$ would have a well-organized hatched pattern of squares that would be multiplied only with the white squares (zero elements), which constitutes redundant computation. As all the hatched patterned squares and white squares form an organized shape, they have an increased possibility to be erased to minimize redundant computation.

IV. Experiment and Analysis

A. Experimental Environment and Data

Since dropout is a general technique that needs to be adjusted for every kind of neural network, we designed three experiments with a simple fully connected neural network architecture. We experimented with the difference in accuracy between controlled dropout and traditional dropout and investigated the test error by differentiating the dropout rate and the number of hidden layers. We implemented this model by using the Tensorflow [14] library from Google. The system we used for experimentation has an Intel core 4th generation i7-5930k CPU and 64 GB DDR4 RAM memory. The model was trained on the CPU.

We used the MNIST dataset, which is widely used for digit recognition [15]. This dataset consists of images of handwritten digits from 0 to 9. The size of the training data is 60,000 images, and the test data comprises a set of 10,000 images, each of which represents a 28×28 digit image.

B. Recognition Accuracy

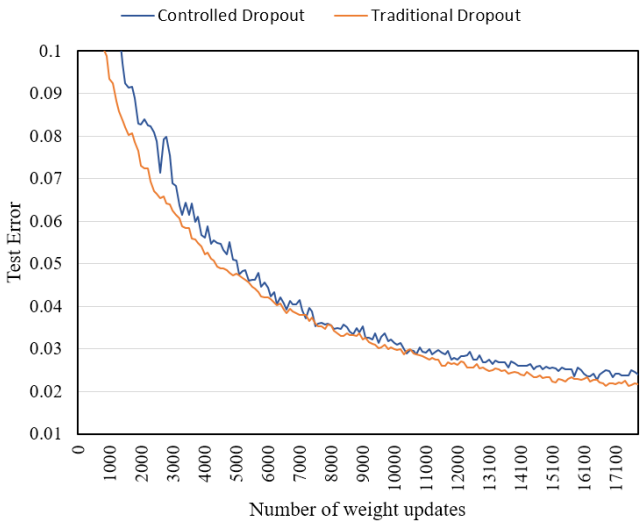

Fig. 6. Test error of traditional dropout and controlled dropout

We investigated whether controlled dropout adequately performs regularization by comparing its test accuracy with that of traditional dropout. We built two models for each case with both models having exactly the same architecture except for the dropout approach. Both are fully connected neural networks with a 784-2000-2000-2000-10 architecture. We used rectified linear units (ReLUs) [16] for each hidden layer and a softmax function for the output layer. The dropout

Fig. 7. Accuracy difference between controlled dropout and traditional dropout

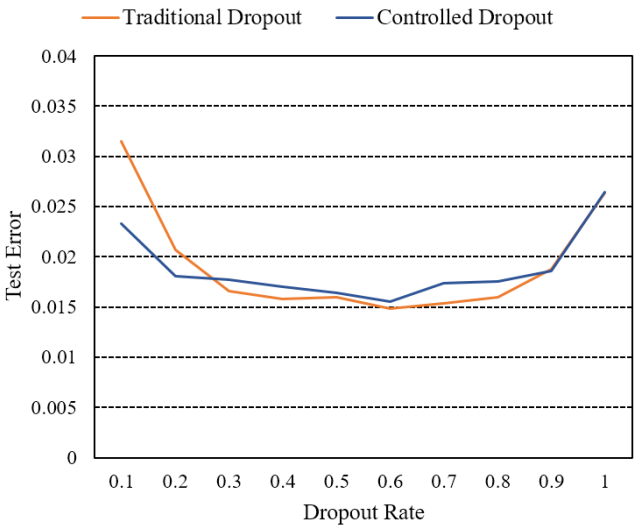

Fig. 8. Effect of changing dropout rates

rate was 0.5, no input dropout was used, and the learning rate was 0.00005.

Fig. 6 shows the result of the test error of traditional dropout and controlled dropout. We included test errors below 0.1 on the graph to enable us to visualize the difference between the two dropout approaches more clearly. Fig. 7 shows the difference in accuracy between the two different dropout models obtained as a function of the number of weight updates. The difference in accuracy was computed as the accuracy of traditional dropout minus the accuracy of controlled dropout. Initially, traditional dropout has higher accuracy since controlled dropout trained the model with chunked matrices that are biased against the data properties. However, after approximately 5,000 steps, controlled dropout started approaching traditional dropout because it had various data properties after sufficient training. After that, the accuracy difference between the two dropout approaches reached zero point and started to fluctuate around it. The accuracy difference is approximately around $1 \times 10^{(-3)}$ which is not significant.

It shows that, with enough number of weight updates, the controlled dropout works as well as the traditional dropout does. This result could have occurred because MNIST has insufficient diversity. Thus, we need to conduct further research with other types of data that have more features than the data in MNIST. However, we succeeded in showing that controlled dropout can obtain adequate performance as traditional dropout.

C. Effect of Dropout Rate

The dropout technique has a tunable hyperparameter (the dropout rate) p, which is the probability of retaining a unit in the network. The paper in which traditional dropout was originally proposed suggested to use 0.5 for the hidden layer and 0.2 for the input layer. However, since the dropout rate is a "hyperparameter", the most optimal rate could differ depending on the network models, data, the number of layers, and so on. In this section, we examine the effect of varying this hyperparameter. We used 784-2000-2000-10 network architecture and varied the dropout rate from 0.1 to 1.0.

Fig. 8 shows the test error of both traditional dropout and controlled dropout obtained as a function of the dropout rate. When the dropout rate is low, i.e., ranging from 0.1 to 0.3, their test errors are higher than other dropout rates since employing very few units for tuning leads to underfitting because of the low dropout rate. For a dropout rate in the range from 0.4 to 0.8, the graph tends to be flat and then the error increases as the dropout rate approaches 1. This tendency occurs for both traditional dropout and controlled dropout similarly. Thus, by choosing values of p that are close to 0.6, controlled dropout shows reasonable performance as traditional dropout.

D. Effect of the Number of Layer

We built an experiment by differentiating the number of layers and compared the tendency between traditional dropout and controlled dropout. The model we used is the same as in the first experiment with a 784-2000-2000-2000-10 architecture except for the number of hidden layers. Further, we kept the dropout rate constant at 0.5 and the learning rate at 0.00005 for every model.

Fig. 9 shows the test error of both traditional dropout and controlled dropout obtained as a function of the number of hidden layers. The test errors of both dropout techniques increase as the number of hidden layers increases. The reason why the test errors increase is that an increase in the number of layers leads to a dramatic increase in the number of parameters. An excessively large number of parameters for non-complicated data such as MNIST can incur lower accuracy [17]. When the number of hidden layers exceeds three, neither controlled nor traditional dropout could train the model and failed with much reduced accuracy. However, the experiment shows that

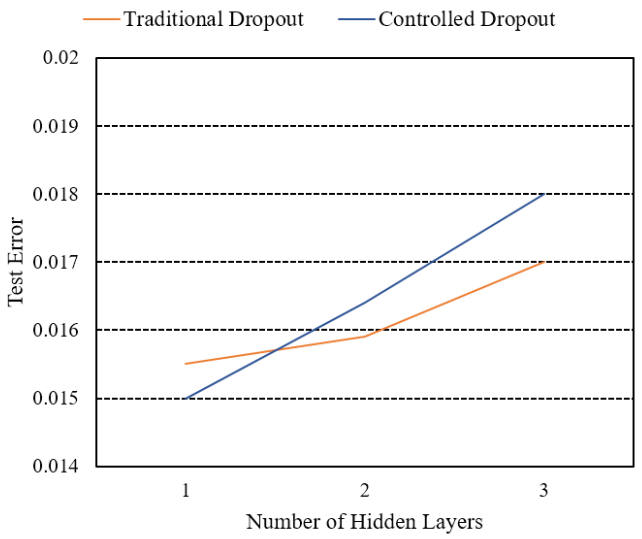

Fig. 9. Effect of the number of hidden layers

both controlled and traditional dropout produces the similar tendency as the number of hidden layers increases.

V. Conclusion

This paper proposed a different approach to using dropout, termed controlled dropout, which intentionally chooses the units to be dropped to form a matrix with an organized shape. We showed that, with the same network model, the difference in accuracy between controlled dropout and traditional dropout decreases as the number of weight updates increases. Furthermore, the accuracy of these two methods approaches each other with a sufficient number of weight updates. Our investigation showed that controlled dropout is most effective when the dropout rate is between 0.4 and 0.8. Moreover, the model is more accurate when it has less than three hidden layers with an MNIST dataset, which is similar to traditional dropout.

In future, we plan to specify how to reduce memory consumption and matrix computation by using controlled dropout to enhance the performance to train neural networks. We aim to implement this approach and compare it with traditional dropout with respect to memory, speed, and matrix computation. Furthermore, we will focus on investigating whether this method could be used on various neural network models, such as a recurrent neural network, convolutional neural network, and deep belief network. Also, we plan to use various dataset other than MNIST dataset as it is not complicated enough to validate regularization performance of dropout.

Acknowledgment

This research was carried out as a collaborative research project (Supercomputer development for leveraging the leadership of national supercomputing) and supported by the Korea Advanced Institute of Science and Technology RED/B.

References

[1] M. A. Mazurowski, P. A. Habas, J. M. Zurada, J. Y. Lo, J. A. Baker, and G. D. Tourassi, "Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance," *Neural Networks*, vol. 21, pp. 427–436, 2008.

[2] 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 *International Conference on Emerging Database (EDB)*, 2016, pp. 28–36.

[3] G. E. Hinton, L. Deng, D. Yu, G. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," 2012.

[4] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3642–3649.

[5] C. M. Bishop, "Pattern recognition," *Machine Learning*, vol. 128, 2006.

[6] Y. Bengio, I. J. Goodfellow, and A. Courville, "Deep learning," *An MIT Press book in preparation. Draft chapters available at http://www. iro. umontreal. ca/ bengioy/dlbook*, 2015.

[7] 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.

[8] S. Mannor, D. Peleg, and R. Y. Rubinstein, "The cross entropy method for classification," in *ICML*, 2005.

[9] F. Akdeniz and H. Erol, "Mean squared error matrix comparisons of some biased estimators in linear regression," *Communications in Statistics-Theory and Methods*, vol. 32, no. 12, pp. 2389–2413, 2003.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[12] O. Yadan, K. Adams, Y. Taigman, and M. Ranzato, "Multi-gpu training of convnets," *arXiv preprint arXiv:1312.5853*, vol. 9, 2013.

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

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

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

[16] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 807–814.

[17] Y. Bengio, Y. LeCun *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, 2007.