Dynamic DropConnect: Enhancing Neural Network Robustness through Adaptive Edge Dropping Strategies

Yuan-Chih Yang, Hung-Hsuan Chen[*]
National Central University
ericabd888@gmail.com, hhchen1105@acm.org

February 28, 2025

Abstract

Dropout and DropConnect are well-known techniques that apply a consistent drop rate to randomly deactivate neurons or edges in a neural network layer during training. This paper introduces a novel methodology that assigns dynamic drop rates to each edge within a layer, uniquely tailoring the dropping process without incorporating additional learning parameters. We perform experiments on synthetic and openly available datasets to validate the effectiveness of our approach. The results demonstrate that our method outperforms Dropout, DropConnect, and Standout, a classic mechanism known for its adaptive dropout capabilities. Furthermore, our approach improves the robustness and generalization of neural network training without increasing computational complexity. The complete implementation of our methodology is publicly accessible for research and replication purposes at `https://github.com/ericabd888/Adjusting-the-drop-probability-in-DropConnect-based-on-the-magnitude-of-the-gradient/`.

1 Introduction

Dropout [1, 2] and DropConnect [3] are prominent techniques designed to mitigate overfitting in deep neural networks. Dropout functions by independently zeroing each neuron within a layer with a predetermined fixed probability p. In contrast, DropConnect takes a slightly different approach by randomly eliminating an edge within the network with a fixed probability p. This makes DropConnect a generalization of Dropout; specifically, removing a single neuron as performed in Dropout equates to eliminating all incoming and outgoing edges associated with that neuron, which DropConnect facilitates.

Both Dropout and DropConnect uniformly apply a fixed dropping rate across all neurons or edges within a layer. However, this universal dropping rate might not always represent the optimal strategy; ideally, a model should utilize available data-driven insights to tailor the dropping rate for each individual edge or neuron based on their specific characteristics.

To address this, we introduce a novel methodology termed DynamicDropConnect (DDC). This approach dynamically assigns a drop probability to each edge based on the magnitude of the gradient associated with that edge. The underlying principle is that edges with larger gradients are crucial for learning and should be retained, whereas it might be acceptable to omit those with minimal impact on the model's output occasionally.

DDC offers several advantages over other methods such as Standout [4], which also employs a dynamic dropping rate. Firstly, DDC does not require any additional learning parameters, which simplifies the model architecture and reduces memory requirements during training. Secondly, DDC provides a more deterministic and transparent approach to deciding the dropping rate, as opposed to the potentially opaque and unpredictable learning processes used by methods like Standout. Moreover, empirical evidence from our experiments demonstrates that DDC achieves superior accuracy compared to Standout.
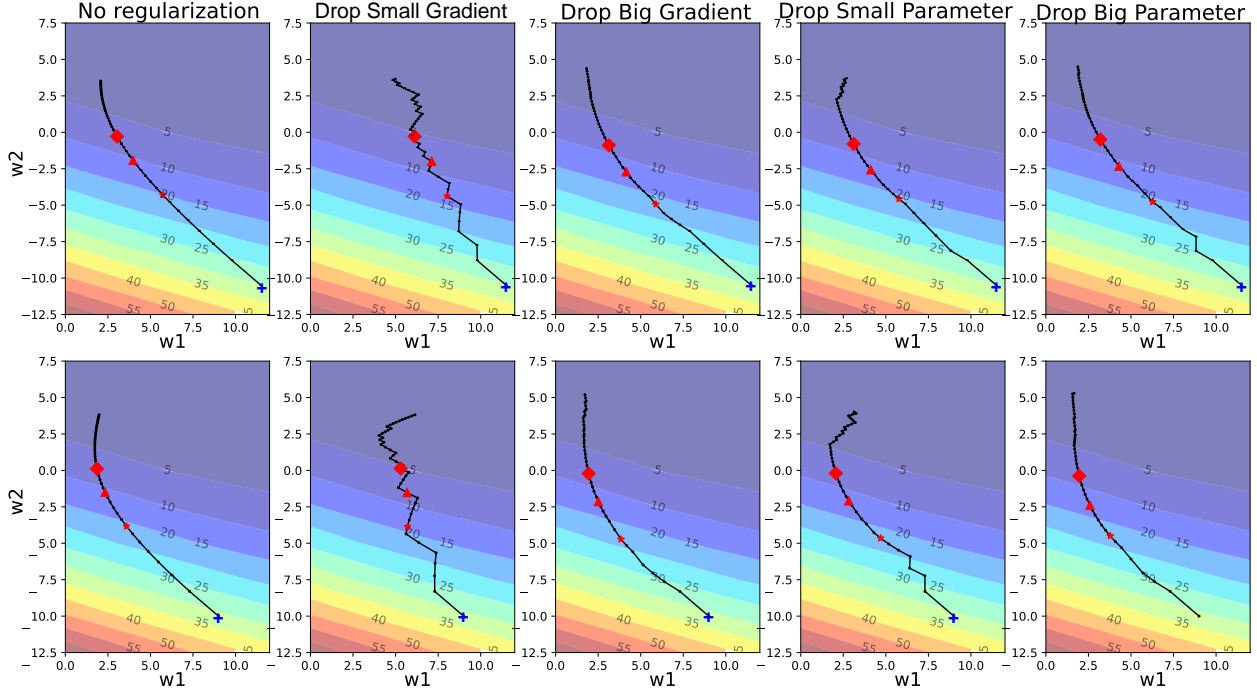
---

[*]Corresponding author

Figure 1: The contour plot of the loss and the parameter update process. The two rows represent two sets of initial values and their updating process. The blue pluses denote the initial values. The red stars, triangles, and diamonds represent the values of w_1 and w_2 after training for 7, 13, and 19 epochs. The method that tends to drop the edges with small gradients (the second column) reaches a small error faster.

Our experimental analysis extends beyond theoretical discussions. We conduct rigorous experiments on both synthetic and real open datasets to validate the efficacy of our methodology. Figure 1 illustrates the trajectory of parameter values and their corresponding losses using a synthetic dataset. The first column in the figure depicts a model without any regularization (labelled as "No regularization" in Figure 1, whereas the second column shows the impact of training the model by preferentially dropping edges associated with smaller gradients ("Drop Small Gradient"). For comparative purposes, we also trained the same model using different variants ("Drop Big Gradient", "Drop Small Parameter", and "Drop Big Parameter"). The plots clearly demonstrate that training a linear regressor using our methodology, which preferentially drops edges with smaller gradients, reaches minimal losses more quickly than other compared methods. This encouraging result has motivated us to further explore the performance of our method when applied to more complex networks on open datasets.

The rest of the paper is organized as follows: Section 2 reviews related studies. Section 3 details the proposed method. Section 4 evaluates various dropping strategies using different network architectures on diverse datasets. Finally, Section 5 concludes our work and outlines potential future research directions.

2 Related Work

Overfitting occurs when a model excels during the training phase but performs inadequately on unseen test data. A typical approach to counter overfitting involves incorporating regularization terms into the objective function, such as the L1 or L2 norms of the learnable parameters. These regularization terms are designed to encourage the development of simpler relationships between features and targets, thus mitigating the risk of overfitting. Early stopping is another frequently adopted strategy to prevent overfitting. This technique involves ceasing the parameter updates once the loss on the validation set begins to show signs of increase, thereby avoiding the overfitting of the training data. Additional well-established methods to combat overfitting include data augmentation [5], employing model ensembles [6], and modifying the architecture of neural networks by reducing their depth or width. These methods have been proven to be effective in enhancing the generalizability of models across unseen datasets.

Dropout is another widely recognized method specifically utilized in neural networks to curb overfitting [2]. During training, Dropout randomly deactivates a subset of neurons, ensuring that the neural network does not become overly

dependent on any particular set of neurons. This technique effectively simulates the use of multiple neural network models and aggregates their predictions, which enhances the robustness of the model. Notably, the random neuron deactivation is only applied during the training phase and not during inference. To maintain consistent expected values between training and inference, Inverted Dropout is commonly utilized.

DropConnect extends the idea of Dropout by randomly omitting connections, or edges, between neurons during the training phase. This approach is similar to Dropout in that masking a neuron in Dropout is analogous to masking both the incoming and outgoing connections of that neuron in DropConnect. Empirically, both Dropout and DropConnect often result in models with comparable performance levels.

From a Bayesian perspective, Dropout can also be interpreted as performing an approximated Bayesian inference in deep Gaussian processes [7]. Notable techniques within this framework include Monte Carlo Dropout [7], fast dropout [8], and variational dropout [9], each offering unique approaches to implementing Dropout within a Bayesian inference context.

Some recent studies have explored the idea of varying the dropping rate throughout the training process. A notable example is Standout [4], which dynamically adjusts the dropping rate based on the values of the model parameters. Unlike other variations of Standout, which often necessitate additional learning parameters, our approach simplifies the process by avoiding extra learning mechanisms. This not only reduces training time and memory requirements but also provides a more transparent method for determining dynamic dropping rates. Furthermore, our research has explored the efficacy of utilizing gradient values rather than parameter values for adjusting drop rates, which has shown promising results in our experimental evaluations.

For a comprehensive review of Dropout and its various adaptations, please refer to the survey by Labach et al. [10].

3 Methodology

This section introduces DropConnect as the background knowledge, followed by our methodology to assign dynamic dropping rates.

3.1 Preliminary: DropConnect

Let $l \in \{1, \ldots, L\}$ be the L hidden layers in a neural network and $\boldsymbol{y}^{(l)}$ be the output of layer l (and therefore the input of layer $l + 1$), each layer of a neural network transforms $\boldsymbol{y}^{(l-1)}$ to $\boldsymbol{y}^{(l)}$ by Equation 1.

$$\boldsymbol{y}^{(l)} = f_l\left(\boldsymbol{z}^{(l)}\right) = f_l\left(\boldsymbol{W}^{(l)}\boldsymbol{y}^{(l-1)}\right), \tag{1}$$

where $\boldsymbol{W}^{(l)}$ is the set of parameters in layer l, and $f_l(\cdot)$ is the activation function of the same layer.

DropConnect assigns a universal dropping rate p such that each edge has a probability p of being turned off during training. Therefore, a neural network, with DropConnect, transforms $\boldsymbol{y}^{(l-1)}$ to $\boldsymbol{y}^{(l)}$ by the equation below during training.

$$\boldsymbol{y}^{(l)} = f_l\left(\boldsymbol{z}^{(l)}\right) = f_l\left(\left(\left(1 - \boldsymbol{M}^{(l)}\right) \odot \boldsymbol{W}^{(l)}\right)\boldsymbol{y}^{(l-1)}\right), \tag{2}$$

where $\boldsymbol{M}^{(l)} = [m_{i,j}]$ is a $(0, 1)$-matrix whose shape is the same as $\boldsymbol{W}^{(l)}$; each entry $m_{i,j}$ is sampled from a Bernoulli distribution with a fixed hyper-parameter p, and $\odot$ performs the Hadamard product (i.e., element-wise product) of matrices. Thus, $\boldsymbol{M}^{(l)}$ is a mask matrix that decides which parameters in $\boldsymbol{W}^{(l)}$ (edges) to omit.

3.2 DDC – Mask Generation

The proposed DDC advances DropConnect by allowing variable dropping probabilities for distinct neural network edges.

We create a mask matrix, $\widetilde{\boldsymbol{M}}^{(l)}$ to drop edges. Each entry $\widetilde{m}_{i,j}^{(l)}$ in $\widetilde{\boldsymbol{M}}^{(l)}$ is sampled from a Bernoulli distribution with a unique parameter, $p_{i,j}^{(l)}$. An edge is omitted if $\widetilde{m}_{i,j}^{(l)} = 1$.

Algorithm 1 shows a pseudocode to generate $\widetilde{\boldsymbol{M}}^{(l)}$ during training. For every edge $e_{i,j}^{(l)}$ that bridges neuron i at layer $l - 1$ and neuron j at layer l, the algorithm decides the value of the corresponding mask cell $\widetilde{m}_{i,j}^{(l)}$ based on its current

---

Algorithm 1: Generating a mask for $\boldsymbol{W}^{(l)}$

Input : gradients $\boldsymbol{G}^{(l)} = \left[g_{i,j}^{(l)} \right] \in \mathcal{R}^{n_1 \times n_2}$,
hyperparameters p, p_g, and τ

Output: mask $\widetilde{\boldsymbol{M}}^{(l)} = \left[\widetilde{m}_{i,j}^{(l)} \right] \in \mathcal{R}^{n_1 \times n_2}$

1 **for** $i \leftarrow 1$ **to** n_1 **do**
2   **for** $j \leftarrow 1$ **to** n_2 **do**
3        $v_{i,j}^{(l)} \leftarrow \left| g_{i,j}^{(l)} \right|$;
4   **end**
5 **end**

6 $\mu^{(l)} \leftarrow \dfrac{\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} v_{i,j}^{(l)}}{n_1 \times n_2}$;

7 $\sigma^{(l)} \leftarrow \dfrac{\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} (v_{i,j}^{(l)} - \mu^{(l)})^2}{n_1 \times n_2}$;

8 **for** $i \leftarrow 1$ **to** n_1 **do**
9   **for** $j \leftarrow 1$ **to** n_2 **do**
10        $z_{i,j}^{(l)} \leftarrow \dfrac{v_{i,j}^{(l)} - \mu^{(l)}}{\sigma^{(l)}}$;
11   Update $q_{i,j}^{(l)}$ by Equation 3;
12   Update $p_{i,j}^{(l)}$ by Equation 4;
13        $\widetilde{m}_{i,j}^{(l)} = \text{Bernoulli}(p_{i,j}^{(l)})$;
14   **end**
15 **end**

---

gradient $g_{i,j}^{(l)}$ at every training iteration. The algorithm first computes the absolute gradient value of $g_{i,j}^{(l)}$, represented as $v_{i,j}^{(l)}$ (line 1 to line 5). DDC needs gradient normalization to harmonize the gradient magnitudes across distinct layers. Therefore, the mean and standard deviation of the set of $v_{i,j}^{(l)}$ values for layer l are calculated (line 6 and line 7). Subsequently, a z-score normalization is performed on $v_{i,j}^{(l)}$, yielding normalized gradient magnitudes $z_{i,j}^{(l)}$-s. DDC then calculates $q_{i,j}^{(l)}$ for each edge based on the normalized gradient magnitude $z_{i,j}^{(l)}$. This candidate dropping probability positively correlates with the edge's final dropping probability. Eventually, we define the candidate dropping probability based on Equation 3, so the model tends to keep the edge with a larger gradient magnitude.

$$q_{i,j}^{(l)} \leftarrow \begin{cases} 1 - \sigma\left(z_{i,j}^{(l)}\right) & \text{if } 1 - \sigma\left(z_{i,j}^{(l)}\right) \geq \tau \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $\sigma()$ is the sigmoid function. Since $\sigma()$ is monotonically increasing and the output is always between 0 and 1, a larger $z_{i,j}^{(l)}$ results in a smaller candidate dropping probability. We set hyperparameter τ to 0.5. So, half of the $q_{i,j}^{(l)}$-s are zero on average because the expected value of $\sigma\left(z_{i,j}^{(l)}\right)$ is 0.5.

The final dropping rate $p_{i,j}^{(l)}$ for each edge $e_{i,j}^{(l)}$ is a function of three variables: the candidate dropping probability $q_{i,j}^{(l)}$, a base dropping probability p (a hyper-parameter), and a gradient unit dropping rate p_g (another hyper-parameter). We limit the value of $p_{i,j}^{(l)}$ to be no greater than 1 to ensure $p_{i,j}^{(l)}$ as a proper probability value. The computation of $p_{i,j}^{(l)}$ is shown in Equation 4.

$$p_{i,j}^{(l)} \leftarrow \begin{cases} p + p_g \times q_{i,j}^{(l)} & \text{if } p + p_g \times q_{i,j}^{(l)} \leq 1 \\ 1 & \text{otherwise.} \end{cases} \tag{4}$$

Algorithm 2: One-layer forward re-calibration during training

---

Input : weights $\boldsymbol{W}^{(l)} = \left[w_{i,j}^{(l)} \right] \in \mathcal{R}^{n_1 \times n_2}$,

gradients $\boldsymbol{G}^{(l)} = \left[g_{i,j}^{(l)} \right] \in \mathcal{R}^{n_1 \times n_2}$,

outputs from previous layer $\boldsymbol{y}^{(l-1)} \in \mathcal{R}^{n_2 \times 1}$

Output: $\boldsymbol{y}^{(l)} \in \mathcal{R}^{n_1 \times 1}$

1 Compute $\widetilde{\boldsymbol{M}}^{(l)}$ by Algorithm 1 using $\boldsymbol{W}^{(l)}$ and $\boldsymbol{G}^{(l)}$;

2 $\widetilde{\boldsymbol{W}}^{(l)} \leftarrow \left(1 - \widetilde{\boldsymbol{M}}^{(l)} \right) \odot \boldsymbol{W}^{(l)}$;

3 Compute $r^{(l)}$ by Equation 5;

4 Compute $\boldsymbol{y}^{(l)}$ by Equation 6;

Finally, the DDC generates each $\widetilde{m}_{i,j}^{(l)}$ in $\widetilde{\boldsymbol{M}}^{(l)}$ by sampling from a Bernoulli distribution with the parameter $p_{i,j}^{(l)}$ (line 13).

3.3 DDC – Training Re-calibration and Testing

Since $\widetilde{\boldsymbol{M}}^{(l)}$ is the mask for the parameters in layer l, the effective parameters in this layer go from $\boldsymbol{W}^{(l)}$ to $(1 - \widetilde{\boldsymbol{M}}^{(l)}) \odot \boldsymbol{W}^{(l)}$. The dropping mechanism is only applied in training but not in prediction. However, if we set the value of $\boldsymbol{y}^{(l)}$ as $f_l \left((1 - \widetilde{\boldsymbol{M}}^{(l)}) \odot \boldsymbol{W}^{(l)} \boldsymbol{y}^{(l-1)} \right)$ in training but predict $\boldsymbol{y}^{(l)}$ as $f_l \left(\boldsymbol{W}^{(l)} \boldsymbol{y}^{(l-1)} \right)$ in inference, the learned $\boldsymbol{W}^{(l)}$ from training cannot be used directly in the tests. To fix the inconsistency, we need to recalibrate $(1 - \widetilde{\boldsymbol{M}}^{(l)}) \odot \boldsymbol{W}^{(l)}$ during training, so that the learned $\boldsymbol{W}^{(l)}$ in training can be used directly during inference by $f_l \left(\boldsymbol{W}^{(l)} \boldsymbol{y}^{(l-1)} \right)$.

Since $\widetilde{\boldsymbol{M}}^{(l)}$ is a $(0,1)$-matrix, $\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \widetilde{m}_{i,j}^{(l)}$ is the number of 1-s in $\widetilde{\boldsymbol{M}}^{(l)}$. So, we can compute the expected value of the dropping rate of layer l by Equation 5.

$$r^{(l)} = \frac{\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \widetilde{m}_{i,j}^{(l)}}{n_1 \times n_2}, \tag{5}$$

where n_1 and n_2 are the number of rows and the number of columns of the masking matrix $\widetilde{\boldsymbol{M}}^{(l)}$.

We re-calibrate the output of layer l during training such that the learned weights $\boldsymbol{W}$ can be used during inference. The re-calibration is achieved by dividing the output by the keep rate:

$$\boldsymbol{y}^{(l)} \leftarrow f_l \left(\widetilde{\boldsymbol{W}}^{(l)} \boldsymbol{y}^{(l-1)} \times \frac{1}{1 - r^{(l)}} \right). \tag{6}$$

The mask matrix $\widetilde{\boldsymbol{M}}^{(l)}$ and the masked parameter matrix $\widetilde{\boldsymbol{W}}^{(l)}$ are only used in training. At inference, we use only the unmasked parameter matrix $\boldsymbol{W}^{(l)}$ for forwarding: $\boldsymbol{y}^{(l)} \leftarrow f_l \left(\boldsymbol{W}^{(l)} \boldsymbol{y}^{(l-1)} \right)$.

Algorithm 2 shows one-layer forward re-calibration in training.

4 Experiments

This section introduces the experiment settings and results. All the models are implemented by PyTorch and trained on the NVIDIA GTX 3090. We conducted experiments based on one synthetic dataset and four open datasets: MNIST, CIFAR-10, CIFAR-100, and NORB. We split the labeled instances for each open dataset into the training set, the validation set, and the test set. Detailed settings, such as parameter initialization, learning rate, and batch size, are included in the experimental code.

4.1 Experiments on the Synthetic Dataset

We generate a synthetic dataset to analyze the learning process of various edge-dropping strategies. For each instance i, we sample two independent features $x_1^{(i)}$ and $x_2^{(i)}$, each $x_j^{(i)} \sim N(0,1)$. We fix the values of w_1 and w_2 and let the

(a) $w_1 = 11.5, w_2 = -10$
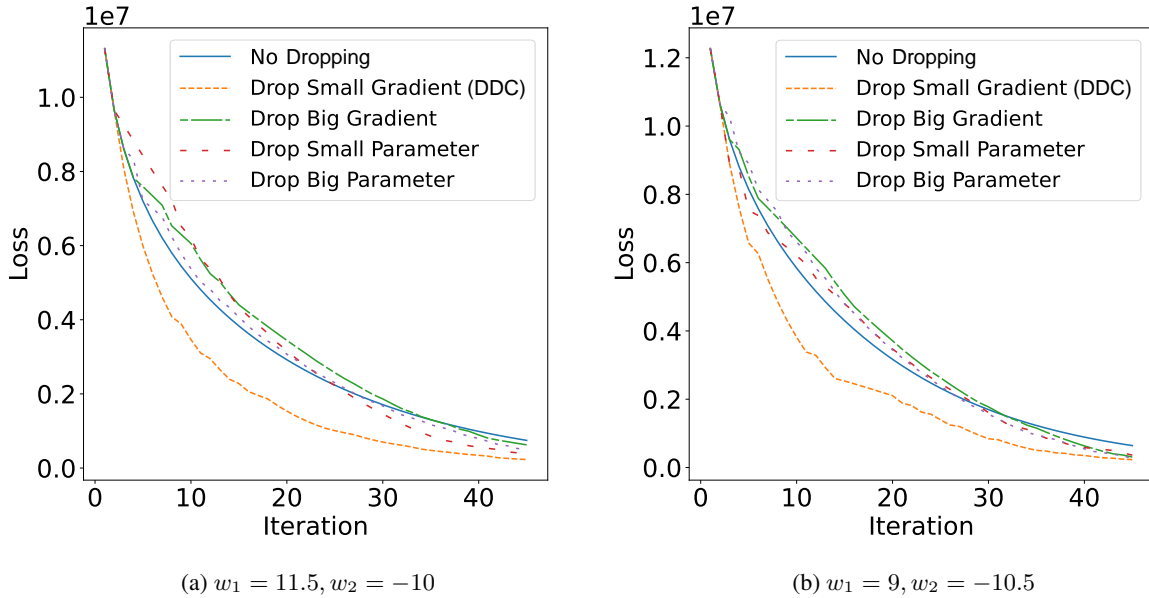
(b) $w_1 = 9, w_2 = -10.5$

Figure 2: Loss vs. epochs of different dropping strategies.

target $y^{(i)}$ be $w_1 x_1^{(i)} + w_2 x_2^{(i)} + \xi$, where $\xi \sim N(0, 1)$. We only use two parameters, w_1 and w_2, because it is easier to visualize the parameter updating/learning process in a 2-dimensional contour plot.

Figure 1 shows the parameter learning process of various edge-dropping strategies: the first column (labeled "No regularization") represents the scenario of the no-dropping mechanism. The second column (labeled "Drop Small Gradient") corresponds to the DDC method. As for the other methods, "Drop Big Gradient" tends to drop the edges with more significant gradients; "Drop Small Parameter" favors dropping edges with small weights; "Drop Big Parameter" tends to drop edges with large weights. The horizontal and vertical axes denote the values of the estimated w_1 and w_2 at different epochs, and the contour lines represent the loss. We highlight the initial values of w_1 and w_2 and their values at the 7th, 13th, and 19th epochs by blue pluses, red stars, red triangles, and red diamonds, respectively. As shown, updating the parameters with DDC reaches small losses faster than the compared baselines. We randomly initialize the values of w_1 and w_2 and repeat the process several times. Figure 1 shows two cases with different initializations.

We show the relationship between iteration and the losses of the compared methods in Figure 2. The results show that DDC gets the lowest loss when giving a fixed iteration count. The methodologies "Drop Small Parameter", "Drop Big Parameter", and "No Dropping" are in the middle, and "Drop Big Gradient" yields the largest loss given a fixed iteration count. We demonstrate the results using two sets of initial values for w_1 and w_2. Other sets of initial values give similar patterns. It appears that the parameter values per se are unlikely to be influential factors in deciding the dropping rate; the gradients are more influential.

4.2 Experiments on Open Datasets

The previous section shows that DDC helps linear models learn faster. This section explores DDC's capacity for more complicated deep learning models using four open datasets: MNIST [11], CIFAR-10, CIFAR-100 [12], and NORB [13].

In addition to the baselines introduced in Section 4.1, we include Dropout, DropConnect, and Standout for comparison. We adjust the dropping rate $r^{(l)}$ in Algorithm 2 to be close to the dropping probability of the compared baselines whenever possible.

We test the convolutional neural network used in the DropConnect paper [3] (called SimpleCNN below). In addition, we add two more complicated networks –AlexNet [5] and VGG [14] – for comparison.

Table 1: The test accuracies (%) of applying SimpleCNN on MNIST. We repeat each experiment 5 times and report the mean $\pm$ standard deviation. Let the test accuracies of "No Dropping" and a dropping method m be $a \pm b$ and $c \pm d$, respectively, if $a + b < c - d$, we label the method m with symbol $\Uparrow$ (much better). If $a < c$ but $a + b \not< c - d$, we denote m with $\uparrow$ (better). If $a - b > c + d$, we denote $\Downarrow$ (much worse). Finally, if $a > c$ but $a - b \not> c + d$, we label $\downarrow$ (worse). We also highlight the method with the largest average accuracy with a boldface.

Method	SimpleCNN
No Dropping	99.08 ± 0.04
DDC	$\mathbf{99.25 \pm 0.01}(\Uparrow)$
Dropout	$99.03 \pm 0.08(\downarrow)$
DropConnect	$99.22 \pm 0.03(\Uparrow)$
Standout	$99.01 \pm 0.02(\Downarrow)$
Drop Small Parameter	$99.24 \pm 0.01(\Uparrow)$
Drop Big Parameter	$99.10 \pm 0.01(\uparrow)$
Drop Big Gradient	$99.12 \pm 0.01(\uparrow)$

Table 2: The test accuracies (%) of applying AlexNet and VGG on the CIFAR-10 dataset, the symbols $(\Uparrow)$, $(\uparrow)$, $(\downarrow)$, and $(\Downarrow)$ are the same as in Table 1.

Method	Network Structure	
	AlexNet	VGG
No Dropping	81.04 ± 0.04	90.46 ± 0.12
DDC	$\mathbf{84.25 \pm 0.09}(\Uparrow)$	$\mathbf{90.94 \pm 0.11}(\Uparrow)$
Dropout	$83.86 \pm 0.09(\Uparrow)$	$90.66 \pm 0.07(\Uparrow)$
DropConnect	$83.52 \pm 0.17(\Uparrow)$	$90.68 \pm 0.08(\Uparrow)$
Standout	$83.75 \pm 0.04(\Uparrow)$	$90.64 \pm 0.08(\uparrow)$
Drop Small Parameter	$83.00 \pm 0.10(\Uparrow)$	$89.88 \pm 0.01(\Downarrow)$
Drop Big Parameter	$83.81 \pm 0.06(\uparrow)$	$90.45 \pm 0.02(\downarrow)$
Drop Big Gradient	$83.19 \pm 0.16(\uparrow)$	$90.88 \pm 0.03(\Uparrow)$

MNIST contains grayscale images; the size of each image is 28×28. The dataset is simple, so we tested only different dropping strategies on SimpleCNN, which includes three convolutional layers and two fully connected layers. The detailed structure is given in [3]; the hyperparameters and detailed settings are included in the experimental code.

The results are presented in Table 1. After repeating each experiment five times, we report the average accuracy $\pm$ standard deviation. As demonstrated, most test accuracies improve when dropping strategies are applied, and the proposed DDC method achieves the best accuracy among all compared methods. Although all methods exhibit accuracies above 99%, DDC is significantly better because it consistently shows very small standard deviations across all repeated trials.

CIFAR-10 is a more challenging dataset: it contains color images; each image's size is 32×32. We use AlexNet [5] and VGG [14] network structures for the experiments because CIFAR-10 is more complex than MNIST.

Table 2 shows the results of applying AlexNet and VGG using the CIFAR-10 dataset. For both the AlexNet and VGG networks, the dropping methods mostly increase test accuracies, and our DDC performs the best among all methods in both networks.

We also test VGG and AlexNet on CIFAR-100 by changing the output softmax layer to 100 categories; all other settings are identical to the previous setup.

Table 3 shows the results of applying the AlexNet and VGG on the CIFAR-100 dataset. The results are similar to those reported earlier: methods with dropping rates usually perform better than without dropping. Our DDC performs best on the VGG network and second on AlexNet.

Finally, we apply the AlexNet and the VGG network to the NORB dataset. The outcomes of these experiments are displayed in Table 4. Our DDC method still consistently outperforms the other techniques on both datasets. It is noteworthy that, in some cases when using the AlexNet, methodologies that involve dropping rates occasionally yield lower performance compared to methods that do not employ any dropping strategy on the NORB dataset. However, the

Table 3: The test accuracies (%) of applying AlexNet and VGG on the CIFAR-100 dataset, the symbols $(\Uparrow)$, $(\uparrow)$, $(\downarrow)$, and $(\Downarrow)$ are the same as in Table 1.

	Network Structure	
Method	AlexNet	VGG
No Dropping	62.53 ± 0.50	71.38 ± 0.03
DDC	$64.06 \pm 0.37(\Uparrow)$	$\mathbf{72.09 \pm 0.04}(\Uparrow)$
Dropout	$\mathbf{66.53 \pm 0.27}(\Uparrow)$	$71.69 \pm 0.06(\Uparrow)$
DropConnect	$63.72 \pm 0.11(\Uparrow)$	$71.60 \pm 0.21(\uparrow)$
Standout	$63.97 \pm 0.38(\Uparrow)$	$71.53 \pm 0.31(\uparrow)$
Drop Small Parameter	$63.10 \pm 0.13(\uparrow)$	$71.71 \pm 0.10(\Uparrow)$
Drop Big Parameter	$63.02 \pm 0.47(\uparrow)$	$71.82 \pm 0.03(\Uparrow)$
Drop Big Gradient	$62.92 \pm 0.22(\uparrow)$	$71.13 \pm 0.07(\Downarrow)$

Table 4: The test accuracies (%) of applying AlexNet and VGG on the NORB dataset, the symbols $(\Uparrow)$, $(\uparrow)$, $(\downarrow)$, and $(\Downarrow)$ are the same as in Table 1.

	Network Structure	
Method	AlexNet	VGG
No Dropping	92.08 ± 0.11	93.36 ± 0.03
DDC	$\mathbf{92.93 \pm 0.09}(\Uparrow)$	$\mathbf{94.20 \pm 0.16}(\Uparrow)$
Dropout	$92.04 \pm 0.07(\downarrow)$	$93.79 \pm 0.25(\Uparrow)$
DropConnect	$92.09 \pm 0.14(\uparrow)$	$93.94 \pm 0.19(\Uparrow)$
Standout	92.08 ± 0.07	$93.81 \pm 0.08(\Uparrow)$
Drop Small Parameter	$92.42 \pm 0.10(\Uparrow)$	$94.02 \pm 0.05(\Uparrow)$
Drop Big Parameter	$92.02 \pm 0.04(\downarrow)$	$93.82 \pm 0.15(\Uparrow)$
Drop Big Gradient	$91.85 \pm 0.13(\downarrow)$	$94.17 \pm 0.07(\Uparrow)$

difference is not statistically significant. By examining the mean accuracies plus or minus one standard deviation, we observe that the performance of the poorer performing methods with dropping rates overlaps with the "No Dropping" method.

5 Conclusion

This paper presents a straightforward yet effective methodology for dynamically adjusting the dropping rate of the edges in a neural network. This approach avoids the complexity of introducing additional learning parameters and simplifies the implementation process. We demonstrate the parameter updating process through a series of experiments using a synthetic dataset and multiple open datasets and compare our proposed methodology against Dropout and its various adaptations. The results of these experiments indicate that our method surpasses traditional approaches in nearly all scenarios.

The efficacy of gradient magnitude as a reliable indicator for setting the dropping rate has been validated through our results, suggesting a promising direction for future enhancements. Building on this foundation, we propose to develop a model that algorithmically uses gradients as features to determine optimal dropping rates. We hypothesize that this advanced strategy could potentially elevate the model's predictive accuracy even further. However, it's important to note that this approach would introduce additional parameters that require learning, which could extend the training duration and increase computational demands.

Furthermore, we are interested in delving into a more rigorous theoretical analysis of our methodology. Previous studies, such as those referenced in [7], have explored the connection between Dropout techniques and Bayesian learning. Inspired by this, we are interested in investigating potential theoretical links between our dynamic edge dropping approach and Bayesian inference principles. This exploration could offer deeper insights into the probabilistic foundations of our method and possibly reveal new theoretical reasoning that could explain why our Dynamic DropConnect method enhances model robustness and generalizability more effectively than traditional Dropout techniques.

In summary, our current methodology provides a significant improvement over traditional techniques. Our future efforts will focus on refining this approach by incorporating more sophisticated learning strategies and gaining deeper theoretical insight. Our goal is to further boost the robustness and accuracy of neural network training.

Acknowledgement

References

[1] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[2] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[3] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

[4] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. *Advances in neural information processing systems*, 26, 2013.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[6] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[7] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[8] Sida Wang and Christopher Manning. Fast dropout training. In *international conference on machine learning*, pages 118–126. PMLR, 2013.

[9] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

[10] Alex Labach, Hojjat Salehinejad, and Shahrokh Valaee. Survey of dropout methods for deep neural networks. *arXiv preprint arXiv:1904.13310*, 2019.

[11] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[12] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, ON, Canada, 2009.

[13] Yann LeCun, Fu Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–104. IEEE, 2004.

[14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.