GYRO DROPOUT: MAXIMIZING ENSEMBLE EFFECT IN NEURAL NETWORK TRAINING

Junyeol Lee [1] Hyeongju Kim [1] Hyungjun Oh [1] Jaemin Kim [1] Hongseok Jeung [1] Yung-Kyun Noh [1] Jiwon Seo [1]

ABSTRACT

This paper proposes *gyro dropout*, a variant of dropout that improves the efficiency of training neural networks. Instead of randomly dropping out neurons in every training iteration, gyro dropout pre-selects and trains a fixed number of subnetworks. Because each subnetwork is more stably trained, they are more diversified and thus their ensemble achieves good generalization. We further propose *block-wise* gyro dropout, or simply block-wise dropout, which is a GPU-friendly variant of gyro dropout. Block-wise dropout partitions hidden neurons into a number of groups that should be dropped out together throughout learning; this makes it efficient to prune the corresponding warp executions on GPUs. We evaluate the two dropout methods with seven neural networks and ten public datasets. In our evaluation, gyro dropout improves the accuracy of trained models by up to 1.93%; gyro dropout consistently achieves higher accuracy than conventional dropout in all experiments. Moreover, block-wise dropout speeds up the training of neural networks by up to 29.8% with little to no accuracy loss. Our implementation of gyro dropout is publicly available at https://github.com/mlsys-seo/gyro-dropout.

1 INTRODUCTION

Dropout is a regularization technique for training deep neural network models (Srivastava et al., 2014). It reduces the overfitting of neural networks by preventing the co-adaptation of hidden neurons. As the technique is simple and effective, dropout is widely used for training many different neural network structures including multi-layer perceptrons, convolutional neural networks, and transformer-based neural networks.

Training with dropout can be understood as an approximation of training with an ensemble of many smaller neural networks (Baldi & Sadowski, 2013; Veit et al., 2016; Olson et al., 2018). In each training iteration, dropout randomly selects subnetworks of the whole network and train them collectively. Therefore, a large number of subnetworks are trained and their ensemble achieves good generalization.

In ensemble learning, the performance and diversity of the base models are important factors for the ensemble performance (Kuncheva & Whitaker, 2003; Rokach, 2010). If the base models are similar to each other or they have poor performance, their ensemble does not generalize well. The ensemble size, i.e. the number of base models, also affects the performance. Although large ensemble size generally gives better performance, it is shown that beyond a certain point adding more base models does not improve the ensemble performance (Margineantu & Dietterich, 1997).

Inspired by these studies in ensemble learning, this paper aims to improve the dropout technique for more efficient neural network training. Specifically, we propose *gyro dropout* that pre-selects a fixed number of subnetworks and train with them throughout learning. In contrast to conventional dropout, the pre-selected subnetworks in gyro dropout are trained for multiple iterations; in our analysis the trained subnetworks are more diverse and thus their ensemble results in better generalization and performance. Moreover, we propose *block-wise gyro dropout*, or simply *block-wise dropout*, which is a GPU-friendly variant of gyro dropout. Block-wise dropout groups adjacent neurons and drops them altogether, making it cheap to prune the corresponding computations in GPU.

Gyro dropout is simple to understand and implement. It does not have any trainable parameters, nor does it require extra computation or monitoring. Still, it is highly effective, consistently outperforming conventional dropout in all our experiments. The contribution of this paper is gyro dropout and block-wise dropout. The specific contribution is summarized as follows.

Concept of Gyro Dropout. We design and propose gyro dropout as a fundamental technique for training neural networks. It is as simple as conventional dropout yet consistently achieves higher accuracy. Moreover, block-wise

[1]Hanyang University, Korea. Correspondence to: Jiwon Seo <seojiwon@hanyang.ac.kr>.

Proceedings of the 5^{th} MLSys Conference, Santa Clara, CA, USA, 2022. Copyright 2022 by the author(s).

dropout is a variant of gyro dropout, which can effectively prune the dropped computations on GPU. Without any accuracy loss compared to conventional dropout, block-wise dropout largely improves the training throughput.

Implementation in TensorFlow and CUTLASS. We implemented the two dropout methods in TensorFlow, a well-known deep learning system and CUTLASS, the state-of-the-art GEMM kernel. Because of its simplicity, gyro dropout is compactly implemented in TensorFlow. For block-wise dropout, however, we need to consider the tiling optimization in CUTLASS to determine the dropout block shape. We analyze the effect of the block shapes on the pruning efficiency and model accuracy to find the optimal shape, with which we largely speed up the training throughput without any accuracy loss over conventional dropout.

Extensive Evaluation and Analysis. We evaluate gyro dropout and block-wise dropout with seven neural network models and ten public datasets in computer vision, network packet detection, and natural language processing. In our evaluation, gyro dropout consistently outperforms conventional dropout for all the neural networks and datasets by up to 1.93% of accuracy. We studied the accuracy gain of gyro dropout by investigating the diversity and accuracy of the subnetworks; also, we examined the amount of overfitting by conventional dropout and gyro dropout. Moreover, block-wise dropout substantially improves the training throughput (by up to 29.8%) without any accuracy loss compared to conventional dropout. Our analysis shows that block-wise dropout effectively reduces the memory loads and floating-point computations. We made our implementation of gyro dropout publicly available at https://github.com/mlsys-seo/gyro-dropout.

The rest of the paper is organized as follows. Section 2 describes the preliminary experiments that motivate our study. Section 3 presents gyro dropout and Section 4 describes its variant block-wise dropout. Section 5 evaluates the two dropout methods. Section 6 discusses the related work and Section 7 concludes.

2 Effect of Subnetwork Pre-Selection

Dropout is known to reduce the overfitting of neural networks by preventing the co-adaptation of hidden neurons. In another view, dropout induces the training of a large number of subnetworks that consist of the neurons that are not dropped out in each training iteration. The ensemble of these subnetworks, i.e. the whole neural network, collectively achieves high performance with good generalization.

In the ensemble view, a subnetwork, which is the base learner of the ensemble model, is trained with a random subset of training data. Because of the random nature of dropout, each subnetwork is trained for only once with a single data point. If we pre-select a fixed number of subnetworks and train with them for longer iterations instead, would it improve their ensemble performance? In this section we observe the effect of the subnetwork pre-selection on the performance of the neural network. Note that this is only preliminary experiments to motivate our study; we present more extensive evaluation in Section 5.

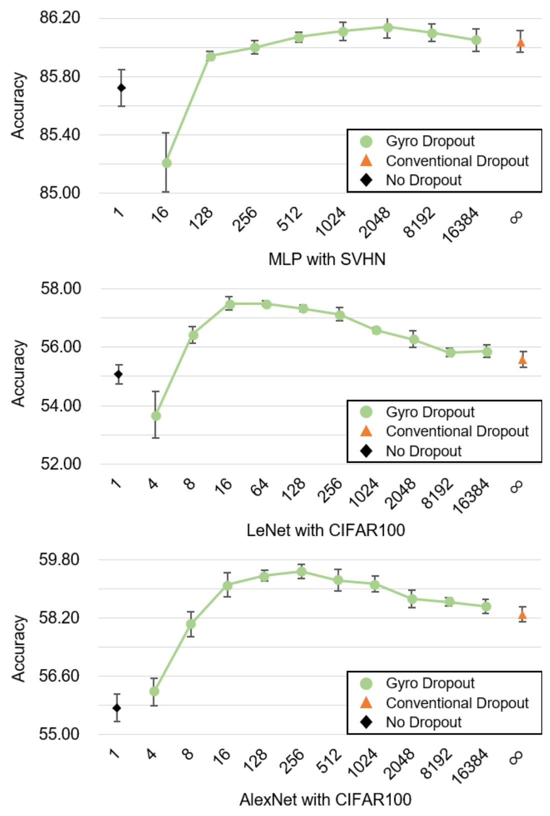

Figure 1. Accuracy of MLP, LeNet, and AlexNet that are trained with 1–16384 pre-selected subnetworks. MLP is trained with SVHN; LeNet and AlexNet are trained with CIFAR100.

Scheduling of Pre-Selected Subnetworks. We considered the following two scheduling methods: 1) randomly selecting the subnetworks in each training iteration, or 2) applying the (randomly) selected subnetworks for a number of consecutive iterations and discarding those subnetworks (consecutive scheduling in short). The first scheduling method requires the materialization of the pre-selected subnetworks in main memory at the beginning or some point of the training.This may incur non-trivial memory overhead for large neural networks such as BERT. Moreover, when we compare the two scheduling methods in Section 5.3, the consecutive scheduling gives higher accuracy for all the evaluated models. Thus we experiment only with the consecutive scheduling in this section.

Number of Subnetworks. To understand the effect of the number of subnetworks on their ensemble performance, we trained two neural network models with an increasing number of pre-selected subnetworks. We trained a three-layer multi-layer perceptron (MLP) with SVHN,LeNet with CIFAR100, and AlexNet with CIFAR100. For the three models, we pre-select 1–16384 subnetworks and train with those selected subnetworks; each subnetwork is trained for $\frac{B \cdot N}{|S|}$ consecutive iterations, where N is the number of total training iterations for the given batch size B and S is the set of the selected subnetworks. The batch size is 256, and we simultaneously train 256 subnetworks in each iteration, if the number of all the selected subnetworks is larger than 256; otherwise, we train all the subnetworks in each iteration. We performed the training for a hundred epochs, which is sufficient for the training to converge for all three models. We repeated the training ten times for each configuration and report the average accuracy of the ten runs.

Figure 1 shows the experimental results. We can see that as we increase the number of subnetworks, the accuracy of the trained models initially increases but then it decreases beyond a certain point. For the SVHN dataset, the training with 2048 subnetworks reaches 86.14% accuracy, which is 0.1% higher than conventional dropout or 0.42% higher than training without dropout (no-dropout). For CIFAR100, the training with 64 subnetworks achieves 57.49% accuracy for LeNet; for the AlexNet model, the training with 256 subnetworks achieves 59.47% accuracy, while no-dropout is 55.7% and conventional dropout is 58.3%. We can see that increasing the number of subnetworks does not improve the performance beyond a certain point and may even negatively affect the performance.

Number of Co-Scheduled Subnetworks. In the previous experiments, we trained 256 subnetworks in each iteration, which is equivalent to the mini batch size. Because these subnetworks share a large portion of their structures, their training may interfere with each other. Training with a smaller number of subnetworks in each iteration may reduce the interference and improve the performance. With this intuition, we run another set of experiments and trained the models with varying number of concurrently scheduled subnetworks. That is, we trained with 1–16384 subnetworks as before, but in each iteration we co-scheduled the training of 1–128 subnetworks instead of 256.

Table 1 shows the results of MLP and AlexNet that are trained with SVHN and CIFAR100 respectively; we do not present the results of LeNet as they show similar trends to the AlexNet case. From the table we first notice once again that as the number of total subnetworks increases, the accuracy of the neural network initially increases but then it decreases beyond a certain point; this is valid for any number of co-scheduled subnetworks, that is, for any column

Table 1. Accuracy of MLP and AlexNet models that are trained with varying numbers of total subnetworks and co-scheduled subnetworks. The first and second highest values in each row are in bold and underlined font respectively.

Model Dataset	Total Subnets (Σ)	Co-scheduled Subnetworks (τ)						
		4	8	16	32	64	128	256
MLP SVHN	16	84.89	$\underline{85.09}$	**85.21**	N/A	N/A	N/A	N/A
	256	**86.08**	$\underline{86.06}$	86.04	86.05	86.03	85.92	86.00
	1024	86.16	**86.20**	$\underline{86.17}$	86.15	86.09	86.13	86.11
	2048	86.12	86.14	**86.19**	86.17	$\underline{86.16}$	86.15	86.14
	8192	86.03	86.13	$\underline{86.15}$	86.11	**86.16**	86.12	86.10
AlexNet CIFAR100	16	58.53	$\underline{58.80}$	**59.19**	N/A	N/A	N/A	N/A
	256	59.42	59.69	**59.98**	$\underline{59.90}$	59.75	59.36	59.56
	1024	59.46	$\underline{59.49}$	**59.52**	59.31	59.41	59.30	58.95
	2048	58.79	$\underline{59.19}$	58.99	**59.23**	59.13	58.92	59.01
	8192	58.31	58.42	$\underline{58.53}$	58.30	**58.66**	58.52	58.42

in the table, except for one case (AlexNet with 256 co-scheduled subnetworks). We also observed that the number of co-scheduled subnetworks (τ) affects the accuracy of the neural network. As we increase τ, the accuracy initially increases but then drops off after 8–16 co-scheduled subnetworks. The highest accuracy is achieved for MLP when training with 1024 pre-selected subnetworks and co-scheduling 8 subnetworks in each iteration. The accuracy is 86.2%, which is 0.16% higher than conventional dropout. For the AlexNet model, it achieved the highest accuracy of 59.98% with 256 subnetworks and co-scheduling 16 subnetworks in each iteration; this is 1.68% higher than conventional dropout. Again this is a limited study to motivate our work and we present more extensive analysis of gyro dropout in our evaluation.

3 GYRO DROPOUT

Based on our preliminary study in the previous section, we propose *gyro dropout*, a variant of dropout that improves the accuracy of deep neural networks. While conventional dropout randomly selects different subnetworks in each training iteration, gyro dropout pre-selects a fixed number of subnetworks and train with them throughout learning. Because the selected subnetworks are trained more robustly, their diversity increases and thus their ensemble achieves higher accuracy.

Gyro dropout has two hyperparameters, that is, the number of pre-selected subnetworks (denoted by Σ) and the number of concurrently scheduled subnetworks in an iteration (denoted by τ). As our study shows, the training with gyro dropout is not too sensitive to these two hyperparameters. When the number of pre-selected subnetworks is 128–2048 and the number of co-scheduled subnetworks is 4–16, the trained models in the study generally achieve good per-

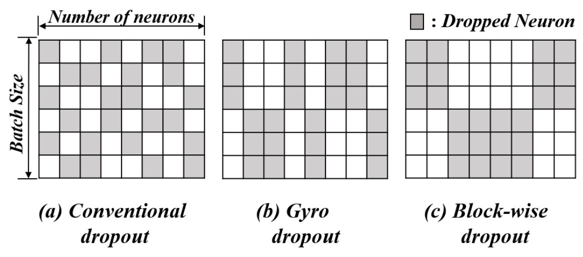

Figure 2. Example layer output with (a) conventional dropout, (b) gyro dropout, and (c) block-wise dropout. The gray filled boxes denote dropped out neurons and the white filled boxes in each row denote a selected subnetwork; (a) uses 6 different subnetworks, and (b) and (c) uses 2 subnetworks.

formance. We set their default values be 1024 (Σ) and 8 (τ), that is, pre-selecting 1024 subnetworks with co-scheduling 8 subnetworks.

Figure 2 illustrates three dropout methods applied to an output of a layer with eight neurons and the batch size of six; (a) is conventional dropout, (b) is gyro dropout, and (c) is block-wise dropout (described later in Section 4). We can see that for the six data in a mini batch, conventional dropout trains six different subnetworks; that is, each row has a dropout pattern that is different from those of all other rows. Gyro dropout in the figure, however, trains with only two subnetworks because the top three rows and bottom three rows have the same dropout patterns. Hence the number of co-scheduled subnetworks (τ) is two in this case. Figure 3 (a) shows an example timeline of training with gyro dropout. In each iteration, four subnetworks are trained; these subnetworks are trained for ten consecutive iterations and then switched to a next set of subnetworks. During the entire training iteration, sixteen subnetworks are trained in total.

The strength of gyro dropout is its simplicity. Unlike other dropout variants (Ba & Frey, 2013; Kingma et al., 2015; Keshari et al., 2019; Pham & Le, 2021; Liang et al., 2021), gyro dropout does not add any trainable parameters; it does not require extra computation or monitoring. It can be applied in the same way as conventional dropout without any extra complexity. Despite its simplicity, gyro dropout performs consistently better than conventional dropout in all our experiments in Section 5.

4 Block-wise Gyro Dropout

Gyro dropout pre-selects a fixed number of subnetworks and train with them. For the subnetwork pre-selection, if it is possible to select those that are cheap to prune the computations for those dropped out neurons, we can speed up the training by reducing the amount of computations. To that end we propose *block-wise* gyro dropout, or simply block-wise dropout, that pre-selects the subnetworks in the way that makes the pruning of the dropped out computations more GPU-friendly. Instead of dropping individual neurons, block-wise dropout selects a block of neurons and sets all their outputs to be zero. Because GPUs are inefficient with randomly sparse matrices, we make dropouts to have block structures and make their pruning more efficient on GPUs. Figure 2 (b) and (c) illustrate layer outputs with gyro dropout and block-wise dropout; compared to the gyro dropout example in (b), block-wise dropout groups adjacent neurons (two neurons in this example) and drops them altogether in (c). To understand the efficacy of block-wise dropout, we first describe the GPU execution model and how matrix multiplication is computed on GPUs.

4.1 Matrix Multiplication on GPU

GPU execution model. A GPU has a number of streaming multiprocessors, or SMs, each of which simultaneously executes multiple hardware threads (Lindholm et al., 2008). GPU adopts *SIMT* (single instruction multiple thread) model, where a group of threads concurrently execute a same instruction sequence. The execution model is supported by its programming abstraction, which provides the concept of *thread blocks* (Sanders & Kandrot, 2010). A thread block is a group of threads that are assigned to a same SM, running in parallel. A thread block consists of hundreds to thousands of threads and is organized as 1D, 2D, or 3D structure to represent multi-dimensional data in application domains. GPU internally divides each thread block into a scheduled unit, called *warp*, which consists of 32 threads executing a same instruction sequence.

SIMT is inefficient for randomly sparse matrix operations, because all threads in a warp must execute a same instruction sequence (Saule et al., 2013). For example, consider the multiplication of randomly sparse matrices on GPU; when a warp executes the multiply-and-add computation it cannot skip the multiplication with zeros and spare GPU cycles because all threads in a warp execute same instructions. In the SIMT execution model, we may skip the multiplication and speedup the execution only if all threads in a warp multiply exclusively with zeroes.

GEMM GPU Kernel. To hide DRAM latency, GEMM (General Matrix Multiplication) kernels commonly apply *tiling* and *pipelining*. Instead of computing each element in an output matrix, the tiling optimization computes the multiplication of a group of elements, called a *tile*, at the same time. The pipelining optimization, then overlaps the computation of the tiles with their loading from DRAM.

CUTLASS (Kerr et al., 2017), the state-of-the-art open-source GEMM kernel, also applies the two optimizations. In CUTLASS, a tile in an output matrix is computed by a

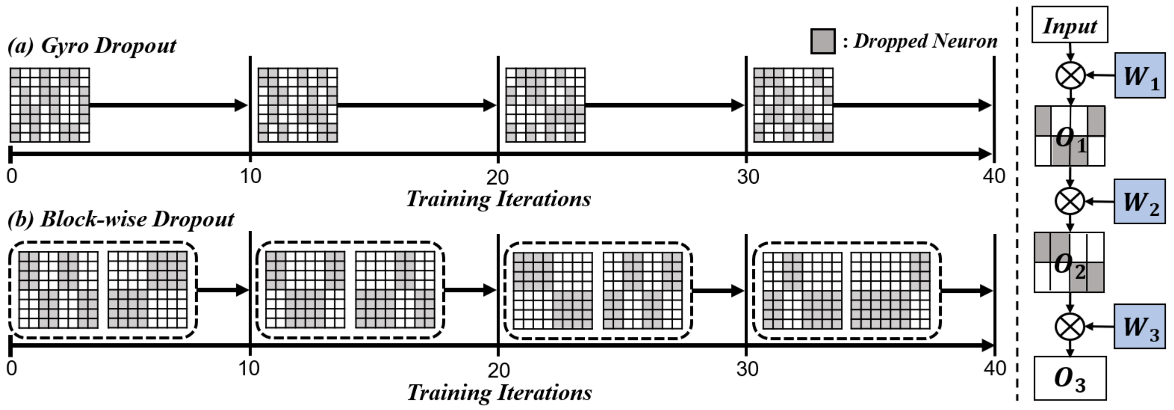

Figure 3. Timeline of training with (a) gyro dropout and (b) block-wise dropout. The output matrix of a first layer is shown. The height of the matrix denotes the batch size; the width is the number of neurons. Both (a) and (b) co-schedule four subnetworks at a time; in (a) the four subnetworks are within a single batch and in (b) the four subnetworks span across two batches. The figure on right illustrates the forward computation of a three-layer neural network with block-wise dropout.

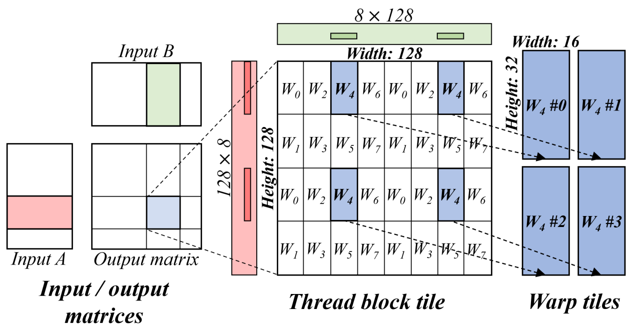

Figure 4. Thread block tile in CUTLASS. Each thread block is assigned a 128× 128 tile by default; each of eight warps in a thread block is assigned four 32×16 tiles. W_i denotes warp with id i.

single thread block, and thus the tile is called *thread block tile*. Figure 4 shows a thread block tile in CUTLASS; The default size of a thread block tile is 128×128. The shapes of thread block tile and other tiles (later described) are determined to achieve coalesced memory access and maximize the memory bandwidth efficiency. To compute the multiplication for a tile, input matrices are loaded in an *input tile* of 128×8 (or 8×128) shape to level-1 cache (i.e., shared memory), as shown at the top and left of the thread block tile (the green and red boxes). Thread block tile is processed by a single thread block consisting of 256 threads, or eight warps (denoted by W_0–W_7), each of which computes four warp tiles of 32×16 shape shown on the right of Figure 4. While input tiles are being loaded, the computation with previously loaded tiles are executed, thus pipelining and overlapping the memory load and the computation.

4.2 Tiling and Dropout Block Shape

Block-wise dropout makes it cheap to prune the dropped out computations and thus improves the training throughput. The shape of dropout blocks, however, affects the model accuracy and also the training throughput. Generally, large block shapes make the pruning cheaper but they incur accuracy loss because they limit the number of total subnetworks and co-scheduled subnetworks; conversely, small block shapes make it more expensive to prune the computations on GPU. To decide the optimal dropout block shapes, we need to consider following two factors: 1) coalesced memory access in GEMM kernels, and 2) effect of subnetwork pre-selection.

GEMM kernels commonly exploit the architecture of GPU's memory subsystem and fetch data from DRAM to shared memory at a cache line granularity, which is 128 bytes. Pruning at a smaller granularity cannot fully utilize the memory subsystem and wastes the memory bandwidth. Hence the dropout block size needs to be larger than or equal to 32 elements, i.e., 128 bytes, for efficient memory transactions. At the same time, we need to prune the computation of an entire row or column of a thread block tile so that we can skip the loading of the corresponding input data (from shared memory to registers). This requires that one dimension of the dropout block be 128 (elements). Thus we consider 128×32 and 32×128 dropout block shapes as the candidates. The former assigns the larger value to the dimension for batch size (i.e., height of thread block tile) and the latter sets the dimension for output neurons to be larger. Now recall the results of our preliminary experiments that the number of co-scheduled subnetworks to maximize the accuracy is fairly small; also, using the larger value for the output neuron dimension and dropout with 32×128 block significantly reduces the possible number of total subnet-

Table 2. Accuracy of an MLP model that is trained with the SVHN dataset and with block-wise dropout. Block shapes of 128×{128,64,32,16} are used with co-scheduling 2, 4, and 8 subnetworks (τ). The first and second highest values in each column are in bold and underlined font respectively.

Block size		Total Subnetworks (Σ)			
		256	1024	2048	8192
128	$\tau = 2$	85.89	85.71	85.62	84.70
	$\tau = 4$	85.81	85.78	85.75	85.49
	$\tau = 8$	85.76	85.71	85.64	85.56
64	$\tau = 2$	**86.03**	85.89	85.73	85.01
	$\tau = 4$	85.90	85.92	85.90	85.67
	$\tau = 8$	85.86	85.87	85.80	85.73
32	$\tau = 2$	85.95	85.95	85.87	85.28
	$\tau = 4$	85.93	85.99	85.98	85.86
	$\tau = 8$	85.94	85.95	85.93	85.81
16	$\tau = 2$	85.95	**86.10**	85.91	85.51
	$\tau = 4$	85.98	86.00	86.00	**85.99**
	$\tau = 8$	85.90	85.96	**86.06**	85.95

works. Taking these factors into account, we decide our candidate dropout block shape be 128×32.

We now test if the dropout block of 128×32 incurs accuracy loss. We trained three MLP models with SVHN and CIFAR{10,100}, applying block-wise dropout with the block shape of 128×{128,64,32,16}; i.e., each group of 128, 64, 32, and 16 adjacent neurons are dropped together respectively. We set the total subnetworks be 256–8192 and the co-scheduled subnetworks (τ) be 2–8. Because the batch size is 256 and the dropout blocks are 128×*, two subnetworks may be trained in a single iteration; if τ is 4 or 8, the co-scheduled subnetworks may span across 2–4 iterations; for example, Figure 3 (b) shows 4 subnetworks across 2 batches.

Table 2 shows the experimental results for the SVHN dataset. We do not report the results for the CIFAR{10,100} dataset as they have similar trends; we also report more extensive evaluation of block-wise dropout in Section 5. We can see from the table that smaller dropout blocks give better accuracy in general. The trained model achieved as high as 86.10% accuracy with 128×16 blocks, which is higher than that of conventional dropout (86.04%). With 128×32 dropout blocks, it achieves 85.99% accuracy that is only 0.05% lower than that of conventional dropout. For CIFAR{10,100}, block-wise dropout achieves 0.02% and 0.17% higher accuracy than that of conventional dropout (see CIFAR{10,100}/MLP row of Table 4 in Section 5). In general, applying block-wise dropout with 128×32 blocks achieves higher accuracy than conventional dropout as shown in our evaluation. Without any accuracy loss, block-wise dropout largely improves the training throughput, which is shown in Section 5.

Table 3. Evaluated models, datasets, and subnetwork settings.

Model	**Dataset**	**Structure**	**Total Subnets(Σ)**	**Co-schedule Subnets(τ)**
MLP_1	SVHN	3×FC	1024	8
MLP_2	CIFAR10 CIFAR100	4×FC	256	16
LeNet	CIFAR10 CIFAR100	2×Conv 3×FC		
AlexNet	CIFAR10 CIFAR100	3×Conv 3×FC		
MLP_3	KDDCup99	4×FC	1024	8
	NSL-KDD			
	UNSW-NB15			
ResNet-18	SVHN	17×Conv 2×FC		
	CIFAR10 CIFAR100			
	TinyImageNet			
	ImageNet			
BERT	MRPC	BERT-Base		
	SQuAD v1.1			

4.3 Input- and Output-Based Pruning

With block-wise dropout, we support two types of pruning, namely, input-based pruning and output-based pruning. As the name suggests, the former prunes with the dropped out neurons in the input matrix and the latter with those in the output matrix. Between the two types of pruning, input-based pruning is slightly more efficient because it makes the pruning of loading input data more efficient than output-based pruning. Thus we prioritize input pruning over output pruning. In Figure 3 right, we show the forward computation of a three-layer neural network with block-wise dropout. For the first layer we apply output pruning and for the last two layers we apply input pruning.

5 Evaluation

We implemented gyro dropout and block-wise dropout in TensorFlow version 1.12 (Abadi et al., 2016) and CUTLASS version 1.3. For the block-wise dropout evaluation, we modified TensorFlow's *matmul* operator to use CUTLASS GEMM instead of the default cuBLAS GEMM. In CUTLASS, we implemented the bypassing of loading and computation for the dropped blocks; we pass the bitmap of dropped out neurons as an extra argument of GEMM and skip the loading and computation for those neurons.

We evaluate the two dropout techniques with seven neural network models and ten public datasets. Out of seven neural networks, three are convolutional neural networks (CNNs), three are multi-layer perceptrons (MLPs), and one is a transformer-based network (BERT) (Devlin et al., 2018). The ten datasets are from three different domains; i.e., computer vision (SVHN, CIFAR10, CIFAR100, and Tiny ImageNet, ImageNet), network packet detection (KD-

Table 4. Accuracy of the neural network models trained with no dropout, conventional dropout, gyro dropout, and block-wise dropout. The average and SD of ten runs are shown. The first and second highest accuracy scores in each row are in bold and underlined font respectively. For BERT, in addition to the accuracy scores, F1 scores are shown in parenthesis. We do not evaluate block-wise dropout for a subset of the models for which the performance gain by block-wise dropout is negligible (denoted by N/A).

Dataset	Model	No Dropout	Conventional Dropout	Gyro Dropout	Block-wise Dropout
SVHN	MLP$_1$	85.72±0.12	86.04±0.07	**86.20±0.05**	85.99±0.07
	ResNet-18	94.40±0.11	94.40±0.09	**94.42±0.05**	N/A
CIFAR10	MLP$_2$	56.97±0.25	57.55±0.23	**57.72±0.23**	57.57±0.28
	LeNet	84.25±0.18	84.27±0.22	84.43±0.10	**84.50±0.11**
	AlexNet	85.75±0.25	86.34±0.23	**86.58±0.13**	86.32±0.17
	ResNet-18	90.03±0.12	92.09±0.12	**92.12±0.09**	N/A
CIFAR100	MLP$_2$	28.34±0.32	29.04±0.29	**29.24±0.22**	29.21±0.26
	LeNet	55.07±0.33	55.57±0.27	**57.50±0.24**	57.28±0.33
	AlexNet	55.73±0.38	58.30±0.21	**59.98±0.20**	59.47±0.28
	ResNet-18	70.65±0.42	70.84±0.30	**71.07±0.32**	N/A
Tiny ImageNet	ResNet-18	56.98±0.21	57.16±0.15	**57.19±0.20**	N/A
ImageNet	ResNet-18	68.70±0.0012	69.40±0.0015	**69.54±0.0012**	N/A
KDDCup99	MLP$_3$	92.57±0.02	92.64±0.02	92.76±0.002	**92.77±0.03**
NSL-KDD	MLP$_3$	80.35±0.12	81.50±0.28	**81.81±0.18**	81.79±0.13
UNSW-NB15	MLP$_3$	84.36±0.43	85.27±0.38	**86.40±0.30**	85.90±0.36
MRPC	BERT	**87.99±1.13**	86.91±0.60	87.67±0.48	N/A
SQuAD v1.1	BERT	79.1±0.43(87.53±0.27)	80.74±0.41(88.18±0.29)	**80.97±0.34(88.41±0.22)**	N/A

DCup99, NSL-KDD, and UNSW-NB15) (Bay et al., 2000; Tavallaee et al., 2009; Moustafa & Slay, 2015), and natural language processing (MRPC and SQuAD) (Wang et al., 2018; Rajpurkar et al., 2016). The models and datasets are summarized in Table 3. Unless otherwise stated, we run all experiments ten times and report the average of the ten runs. We used RMSProp optimizer (Tieleman & Hinton, 2012) for the computer vision and network packet detection models. For BERT we use Adam optimizer (Loshchilov & Hutter, 2017). Dropout rate is set to 50% for all the experiments except for the fine-tuning of BERT, for which we applied 10% drop rate.

All the experiments are performed on NVIDIA Titan XP and V100 GPU. Titan XP has 12GB DRAM and 30 SMs. The host machine has Intel Xeon E5-2620 running at 2.1 GHz with 64GB DRAM. V100 is used for the BERT experiments; it has 32GB DRAM and 80 SMs. Its host machine has Intel Xeon E5-2698 running at 2.2 GHz with 256GB DRAM. Linux with kernel 4.4.0 is used for all experiments.

5.1 Evaluation of Model Accuracy

We evaluate the accuracy of the trained models with four different dropout settings: no dropout, conventional dropout, gyro, and block-wise dropout. For the evaluation, we trained the models in Table 3 for ten times and report the average accuracy and the standard deviation.

Table 4 shows the results. Multiple observations can be made from these results. First, for all the models and datasets, gyro dropout yields higher accuracy than conventional dropout. The maximum accuracy gain is 1.93% (CIFAR100/LeNet) followed by 1.68% (CIFAR100/AlexNet); the minimum gain is 0.02% (SVHN/ResNet). Second, block-wise dropout achieves higher accuracy than conventional dropout for most cases (eight out of ten); the accuracy of block-wise dropout is close to that of gyro dropout (only 0.24% lower on average). Third, for BERT and the natural language processing datasets, the accuracy gain of gyro dropout is relatively small (0.76% and 0.23%). This is because the dropout rate of BERT is 10%, which is much smaller than 50% of other models.

Discussion. We compare the accuracy gain of gyro dropout with that of guided dropout, the state-of-the-art dropout technique proposed by Keshari et al. (2019). Guided dropout finds relatively *less-trained* neurons to train the subnetworks mainly consisting of those neurons. To identify such neurons, they introduce a trainable parameter representing the *strength* of each neuron. Guided dropout changes its dropout rate multiple times throughout learning. However, the authors do not clearly describe the policy for when and how the dropout rate changes; hence we are not able to run experiments with their technique. Instead, we compare with their reported experimental results. Guided dropout has 0.03–2.08% accuracy gain over conventional dropout for the computer vision datasets and models we used. In comparison, gyro dropout has 0.02–1.93% accuracy gain for the same datasets and models. More or less, the two methods achieve similar accuracy gain. We conjecture that the accuracy gain of guided dropout partly comes from reducing the number of subnetworks for the training; applying dropout to only high strength neurons largely limits the possible subnetworks.

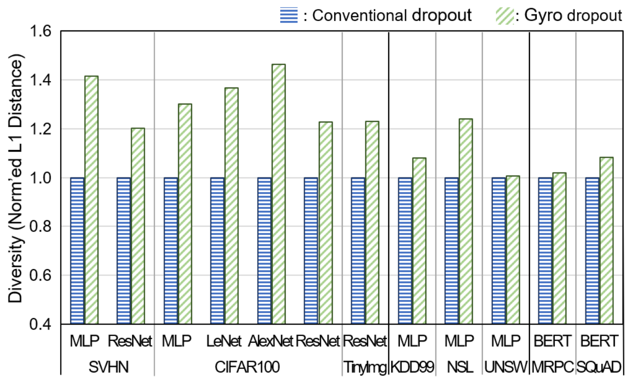

Figure 5. Comparing subnetwork diversity of the models trained with conventional dropout and gyro dropout. Subnetwork diversity is measured as normalized L1 distances of last hidden features among the subnetworks.

Also, gyro dropout may adopt the strategy of guided dropout and train *weak* subnetworks for longer iterations. We leave this as a future work.

5.2 Analysis of Accuracy Improvements

We studied the accuracy improvements from gyro dropout by evaluating: 1) diversity of trained subnetworks, 2) interaction of dropout and subnetwork training, and 3) overfitting of the neural networks. We excluded ImageNet/ResNet from the experiments in this section because its analysis is too expensive. We report our analysis results in the following starting with the diversity analysis.

Subnetwork Diversity. We evaluated the diversity of the subnetworks that are selected for the training with gyro dropout and conventional dropout. For the two dropout methods we randomly selected ten subnetworks that are trained in the last epoch and examined the diversity of their inference computations. We measured the diversity of the subnetworks with the last hidden layer's feature values; that is, we feed the subnetworks with the test data and measured the pairwise L1 distance of the features between the subnetworks. We presume that if the subnetworks are more diverse then the pairwise feature distance is larger.

Figure 5 is the average pairwise feature distances of the subnetworks for the evaluated models and datasets. The blue bars are the distances of conventional dropout and the green bars are those of gyro dropout; the distances are normalized by those of conventional dropout. We can see that for all the models, gyro dropout yields higher feature diversity; on average the feature distances for gyro dropout is 21.9% higher than those for conventional dropout. In other words, the subnetworks of gyro dropout are more diverse than those of conventional dropout, which explains why it achieves better generalization.

Table 5. The generalization gap of the evaluated models. The best (i.e., lowest) values for each row are in bold font. For SQuAD we show two generalization gaps for the accuracy and F1 scores.

Dataset	Model	No Dropout	Conventional	Gyro
SVHN	MLP_1	0.132	0.053	**0.052**
	ResNet-18	0.055	0.056	**0.055**
CIFAR10	MLP_2	0.433	**0.175**	0.195
	LeNet	0.107	0.009	**0.006**
	AlexNet	0.102	0.067	**0.065**
	ResNet-18	0.073	0.069	**0.068**
CIFAR100	MLP_2	0.715	**0.097**	0.108
	LeNet	0.372	-0.010	**-0.013**
	AlexNet	0.396	**0.055**	0.066
	ResNet-18	0.253	0.191	**0.184**
Tiny ImageNet	ResNet-18	0.252	**0.169**	0.175
KDDCup99	MLP_3	0.074	0.073	**0.072**
NSL-KDD	MLP_3	0.196	0.187	**0.182**
UNSW-NB15	MLP_3	0.115	0.098	**0.082**
MRPC	BERT	0.127	0.125	**0.123**
SQuAD v1.1	BERT	0.101	0.011	**0.006**
	BERT (F1)	0.081	0.037	**0.026**

Dropout and Subnetwork Training. We examined the interaction of dropout and subnetwork training in a similar manner to that done by Frankle & Carbin (2018), who applied dropout and their iterative pruning altogether to examine the performance of the *winning tickets*, i.e., the subnetworks derived from the pruning. We run similar experiments and applied their iterative pruning with gyro and conventional dropout to investigate the accuracy of the derived subnetworks when they are completely trained.

Figure 6 shows the accuracy of the pruned subnetworks of MLP, LeNet, AlexNet, and ResNet-18 that are trained with CIFAR100. We can see that the winning tickets with gyro dropout generally achieve higher accuracy. For all the evaluated networks with the vision and network packet datasets, when we examine their subnetworks that are 20, 10, and 5% of the original size, the subnetworks derived with gyro dropout achieve higher accuracy in 26 cases out of 42, or 62% of total. Similar to the experiments conducted by Frankle & Carbin, who reported that dropout improves the accuracy of winning subnetworks, we observed that gyro dropout further improves their accuracy.

Dropout and Overfitting. To find out if gyro dropout reduces overfitting better than conventional dropout, we measured the generalization gap, i.e., test error subtracted by training error, of the trained neural networks (Bousquet & Elisseeff, 2001). Although not absolute, a larger value of the generalization gap may indicate larger amount of overfitting. Table 5 compares the generalization gap of the models that are trained with no dropout, conventional dropout, and gyro dropout. We first observed that in all cases except one (ResNet for SVHN) training with the two dropout

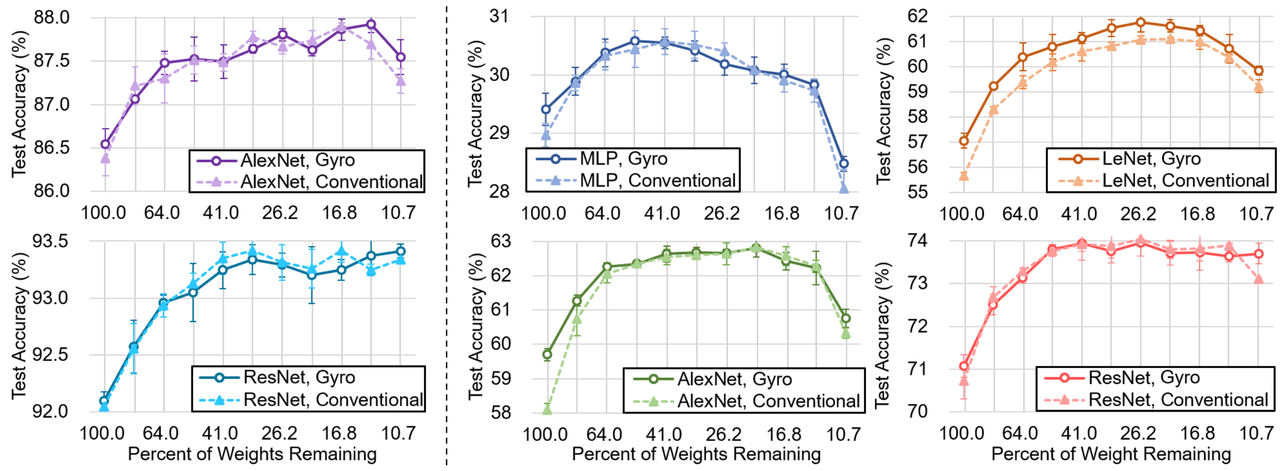

Figure 6. Dropout and subnetwork accuracy for AlexNet and ResNet trained with CIFAR10 (left) and for MLP, LeNet, AlexNet, and ResNet trained with CIFAR100 (right).

Table 6. The accuracies of the models that are trained with two subnetwork scheduling methods.

Model Dataset	Random Mask Scheduling					Consecutive Mask Scheduling (Ours)
	Total Subnets (Σ)	Co-scheduled Subnets (τ)				
		4	8	16	32	
MLP$_1$ SVHN	256	85.87	85.96	85.99	86.04	**86.20**
	512	85.91	86.01	85.97	**86.07**	
	1024	85.82	85.99	85.98	86.01	
LeNet CIFAR100	64	56.81	56.97	**57.15**	56.97	**57.50**
	128	56.59	56.50	56.75	56.66	
	256	56.35	56.53	56.17	56.23	
AlexNet CIFAR100	64	59.27	59.35	**59.45**	59.39	**59.98**
	128	58.96	58.77	58.86	58.97	
	256	58.71	58.67	58.62	58.64	
ResNet-18 CIFAR100	256	70.48	70.73	70.60	70.67	**71.07**
	512	70.41	70.58	70.41	70.67	
	1024	70.73	**70.97**	70.85	70.60	

methods achieves better (i.e., lower) generalization gap than training without dropout. Comparing the two dropout methods, training with gyro dropout achieves better generalization gap in 12 out of 16 cases, or 75% of total. This shows that gyro dropout reduces overfitting and achieves better generalization.

5.3 Comparing Subnetwork Scheduling Methods

In gyro dropout, we schedule subnetworks for consecutive training iterations, discard them, and switch to different subnetworks. An alternative scheduling method is to schedule random subnetworks in each iteration among the pre-selected ones. We compared the two scheduling methods with four neural network models (MLP, LeNet, AlexNet, and ResNet-18) and two datasets (SVHN and CIFAR100). That is, we trained the models with the datasets using the two subnetwork scheduling methods and compared the accuracy of the trained models. Table 6 shows the results. Random mask scheduling denotes scheduling different subnetworks in each iteration and consecutive mask scheduling denotes our scheduling method used in gyro dropout. For the random mask scheduling, we evaluated with multiple configurations of total subnetworks and co-scheduled subnetworks; the highest accuracy for each model/dataset is in bold font. The table shows that the consecutive mask scheduling in gyro dropout consistently achieves higher accuracy for all the evaluated models.

5.4 Evaluation of Training Throughput

We now evaluate the training throughput of block-wise dropout and compare to that of conventional dropout. Because the throughput of gyro dropout is identical to that of conventional dropout, we do not evaluate gyro dropout here. We trained the models in Table 3 and measure their training throughput. BERT is excluded from this evaluation, as the model is typically trained with low dropout rate of 10%; the performance gain of pruning does not exceed its overhead. We also excluded ResNet as the model mainly consists of convolution layers where we do not apply dropout.

Figure 7 shows the evaluation results. Let us first examine the convolutional neural networks. The throughput gain for LeNet and AlexNet is 2–7.9%. For these models, the majority of the execution time is spent on the convolution computations. Because block-wise dropout is applied to fully-connected layers, the performance gain is limited for these CNN models. When we examine the execution times of their FC layers, the performance gain is 14–19% for the two models as shown in the figure. We next ex-

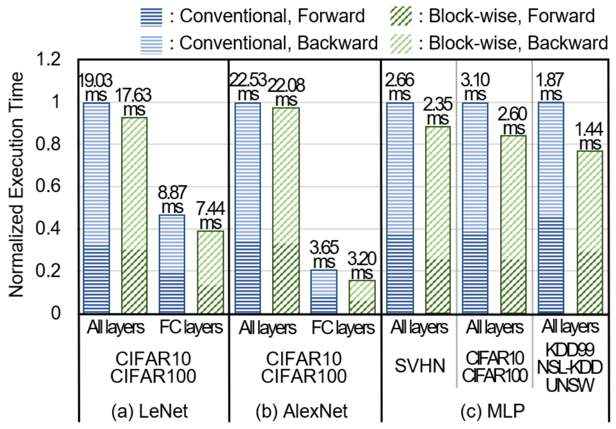

Figure 7. Execution times of single training iterations with conventional and block-wise dropout. For the convolutional neural networks, we also show the execution times of fully-connected layers excluding those of convolutional layers. The throughput of gyro dropout is not shown because it is identical to that of conventional dropout.

amine the speedups of the MLP models for SVHN and CIFAR{10,100}. For these models their training throughput is significantly improved by 13–19.3% with block-wise dropout. Moreover, the MLP models for the network packet detection achieve even better performance gain of 29.8%. Because the MLP models consist of only fully-connected layers where most of the execution time is spent on matrix multiplication, the performance gain is large for these models. When we examine the speedup of forward and backward computation separately, forward computation is improved by 40.0–54.7% and backpropagation is improved by 7.2–14.4%. For the two gradient computations in backpropagation, i.e., neuron and weight gradient computations, we apply pruning only to neuron gradient computation; weight gradient computation requires pruning with 32×32 granularity, which cannot be efficiently executed on GPUs. Also, the amount of weight gradient computation is larger than that of neuron gradient computation in the MLP models. Thus the performance gain of block-wise dropout is smaller in backpropagation.

Analysis of Speedup. We now look closer into the throughput gain of block-wise dropout. For the analysis, we used a 1024×1024 fully-connected layer with 256 batch size. We evaluated its forward computation with varying dropout rate and examined the reduction of the computation and memory load (from DRAM and shared memory). Table 7 shows the experimental results for input-based pruning; the results for output-based pruning are similar but with smaller speedup and reduction in memory loads (not shown due to space). As we increase the drop ratio, the amount of memory load and floating-point computation decreases accordingly. The speedup, however, is lower than the reduction of memory load or computation. The reason is because the reduction is not uniform across the pipeline stages. That is, even if the memory load is reduced by half at one execution point, the overlapping computation (at that point) may not be reduced as much; thus the reduction of the execution time in that case is bounded by a less reduced operation.

Table 7. Profiling of CUTLASS with block-wise dropout and varying dropout ratio for computing a single 1024×1024 fully-connected layer with 256 batch size.

Drop Ratio	Speedup	Global Load	Shared Load	FLOPs
0%	1.00×	100%	100%	100%
10%	1.09×	87.50%	88.25%	87.51%
25%	1.24×	75.39%	76.50%	75.02%
50%	1.58×	50.79%	53.00%	50.05%
75%	2.80×	26.18%	28.79%	25.07%

6 Related Work

Dropout Variants. A number of dropout variants have been proposed to improve the performance and generalization of neural networks (Srivastava et al., 2014; Kingma et al., 2015; Ba & Frey, 2013; Keshari et al., 2019; Gal et al., 2017; Tompson et al., 2015; Huang et al., 2016; Larsson et al., 2016; Ghiasi et al., 2018; Zoph et al., 2018; Pham & Le, 2021). Particularly, the technique of learning and adjusting dropout rate has been extensively studied (Kingma et al., 2015; Gal et al., 2017; Ba & Frey, 2013). Guided dropout, for example, uses trainable parameters representing the *strength* of each neuron to primarily train the neurons that have low strength values (Keshari et al., 2019). More recently, AutoDropout proposed to train a transformer-based model that generates the dropout masks for a given neural network (Pham & Le, 2021). However, the overhead of training the model is large and the accuracy gain brought by the technique is relatively modest.

The technique of dropping channels and residual connections rather than individual neurons has been also widely studied for CNNs (Tompson et al., 2015; Huang et al., 2016; Larsson et al., 2016; Ghiasi et al., 2018; Zoph et al., 2018). DropBlock supports dropping a sub-area (i.e., a block) of a channel to improve the robustness of the features in convolution layers (Ghiasi et al., 2018). Although block-wise dropout also applies dropout at a block level, it improves the training performance by efficiently pruning the corresponding computations on GPUs.

Imposing and Exploiting Structural Sparsity. Because training and running deep neural networks is computationally expensive, pruning the sub-structures of neural networks has been studied (Wen et al., 2016; He et al., 2017; Liu et al., 2017). Wen et al. (2016) proposed Structure Sparsity Learning to impose user-defined sparsity structures to speedup the inference computations. Pruning the channels of convolutional neural networks (He et al., 2017; Liu et al., 2017) has been also studied to accelerate the inference computation. More recently, Oh et al. (2020) proposed to improve the training throughput by taking advantage of converged parameters during the training and pruning the corresponding computations. In block-wise dropout, we take advantage of dropout, i.e., a regularization technique, to improve the performance of training neural networks.

Ensemble Learning and Neural Networks. Training a number of base models and constructing the ensemble model has been widely studied especially in the context of random forest and tree boosting algorithm (Chen & Guestrin, 2016; Ke et al., 2017). In ensemble learning, the number of base models is an important factor of the ensemble performance. However, it is reported that beyond a certain point, adding base models does not improve the performance of the ensemble model (Oshiro et al., 2012; Bonab & Can, 2019), which is one of the motivations of our study. In deep learning, training a neural network is viewed as training a number of subnetworks that are embedded in the whole network (Baldi & Sadowski, 2013; Veit et al., 2016; Olson et al., 2018) . This view is exploited in recent work that studied efficient ensemble structures (Wen et al., 2020; Havasi et al., 2020; Dusenberry et al., 2020; Wenzel et al., 2020). That is, in these methods, most of the model parameters are shared among the base models that make use of different subnetworks of the shared neural network. While we do not explicitly train ensemble models, our study is motivated by the principles of ensemble learning.

7 Conclusion

We proposed *gyro dropout*, a variant of dropout that preselects a fixed number of subnetworks and train with them throughout learning. Compared to conventional dropout, a smaller number of subnetworks are trained in gyro dropout. Our analysis shows that these subnetworks are more diversified and thus their ensemble performs better. Moreover, we proposed a variant of gyro dropout, namely *block-wise dropout*, for efficient pruning of dropped out computations on GPU. Instead of dropping individual neurons, block-wise dropout selects a block of neurons and drops them altogether, thereby making it cheap to prune the warp executions on GPU. We evaluate the two dropout techniques with seven neural networks and ten public datasets. Gyro dropout outperforms conventional dropout in all experiments with achieving up to 1.93% higher accuracy. Moreover, block-wise dropout improves the training throughput by up to 29.8% with little to no accuracy loss compared to gyro dropout.

Acknowledgements

This work is supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2013-0-00109, WiseKB: Big data based self-evolving knowledge base and reasoning platform, No.2018-0-00503, Researches on next generation memory-centric computing system architecture, No.2020-0-01373, Artificial Intelligence Graduate School Program (Hanyang University), and No.2021-0-01817, Development of Next-Generation Computing Techniques for Hyper-Composable Datacenters). The corresponding author is Jiwon Seo.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

Ba, J. and Frey, B. Adaptive dropout for training deep neural networks. *Advances in neural information processing systems*, 26:3084–3092, 2013.

Baldi, P. and Sadowski, P. J. Understanding dropout. *Advances in neural information processing systems*, 26: 2814–2822, 2013.

Bay, S. D., Kibler, D., Pazzani, M. J., and Smyth, P. The uci kdd archive of large data sets for data mining research and experimentation. *ACM SIGKDD explorations newsletter*, 2(2):81–85, 2000.

Bonab, H. and Can, F. Less is more: A comprehensive framework for the number of components of ensemble classifiers. *IEEE Transactions on neural networks and learning systems*, 30(9):2735–2745, 2019.

Bousquet, O. and Elisseeff, A. Algorithmic stability and generalization performance. *Advances in Neural Information Processing Systems*, pp. 196–202, 2001.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dusenberry, M., Jerfel, G., Wen, Y., Ma, Y., Snoek, J., Heller, K., Lakshminarayanan, B., and Tran, D. Efficient and scalable bayesian neural nets with rank-1 factors. In *International conference on machine learning*, pp. 2782–2792. PMLR, 2020.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Gal, Y., Hron, J., and Kendall, A. Concrete dropout. *Advances in Neural Information Processing Systems*, 30, 2017.

Ghiasi, G., Lin, T.-Y., and Le, Q. V. Dropblock: A regularization method for convolutional networks. *arXiv preprint arXiv:1810.12890*, 2018.

Havasi, M., Jenatton, R., Fort, S., Liu, J. Z., Snoek, J., Lakshminarayanan, B., Dai, A. M., and Tran, D. Training independent subnetworks for robust prediction. *arXiv preprint arXiv:2010.06610*, 2020.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 1389–1397, 2017.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Q. Deep networks with stochastic depth. In *European conference on computer vision*, pp. 646–661. Springer, 2016.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

Kerr, J. A., Merrill, D., and Tran, J. Cutlass: Fast linear algebra in cuda c++. *NVIDIA Developer Blog*, 2017.

Keshari, R., Singh, R., and Vatsa, M. Guided dropout. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4065–4072, 2019.

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28: 2575–2583, 2015.

Kuncheva, L. I. and Whitaker, C. J. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003.

Larsson, G., Maire, M., and Shakhnarovich, G. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

Liang, X., Wu, L., Li, J., Wang, Y., Meng, Q., Qin, T., Chen, W., Zhang, M., and Liu, T.-Y. R-drop: Regularized dropout for neural networks. *arXiv preprint arXiv:2106.14448*, 2021.

Lindholm, E., Nickolls, J., Oberman, S., and Montrym, J. Nvidia tesla: A unified graphics and computing architecture. *IEEE micro*, 28(2):39–55, 2008.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pp. 2736–2744, 2017.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Margineantu, D. D. and Dietterich, T. G. Pruning adaptive boosting. In *ICML*, volume 97, pp. 211–218. Citeseer, 1997.

Moustafa, N. and Slay, J. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pp. 1–6. IEEE, 2015.

Oh, H., Yu, Y., Ryu, G., Ahn, G., Jeong, Y., Park, Y., and Seo, J. Convergence-aware neural network training. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. IEEE, 2020.

Olson, M., Wyner, A., and Berk, R. Modern neural networks generalize on small data sets. In *Advances in Neural Information Processing Systems*, pp. 3619–3628, 2018.

Oshiro, T. M., Perez, P. S., and Baranauskas, J. A. How many trees in a random forest? In *International workshop on machine learning and data mining in pattern recognition*, pp. 154–168. Springer, 2012.

Pham, H. and Le, Q. V. Autodropout: Learning dropout patterns to regularize deep networks. *arXiv preprint arXiv:2101.01761*, 1(2):3, 2021.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Rokach, L. Ensemble-based classifiers. *Artificial intelligence review*, 33(1-2):1–39, 2010.

Sanders, J. and Kandrot, E. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

Saule, E., Kaya, K., and Çatalyürek, Ü. V. Performance evaluation of sparse matrix multiplication kernels on intel xeon phi. In *International Conference on Parallel Processing and Applied Mathematics*, pp. 559–570. Springer, 2013.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pp. 1–6. IEEE, 2009.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Tompson, J., Goroshin, R., Jain, A., LeCun, Y., and Bregler, C. Efficient object localization using convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 648–656, 2015.

Veit, A., Wilber, M. J., and Belongie, S. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29: 550–558, 2016.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29:2074–2082, 2016.

Wen, Y., Tran, D., and Ba, J. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*, 2020.

Wenzel, F., Snoek, J., Tran, D., and Jenatton, R. Hyperparameter ensembles for robustness and uncertainty quantification. *arXiv preprint arXiv:2006.13570*, 2020.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.