Balanced Sparsity for Efficient DNN Inference on GPU

Zhuliang Yao [1,4,*], Shijie Cao [2,4,*], Wencong Xiao [3,4], Chen Zhang [4], Lanshun Nie [2]
[1]Tsinghua University [2]Harbin Institute of Technology [3]Beihang University [4]Microsoft Research Asia
{v-zhuyao, v-shicao, v-wencxi, zhac}@microsoft.com, nls@hit.edu.cn

Abstract

In trained deep neural networks, unstructured pruning can reduce redundant weights to lower storage cost. However, it requires the customization of hardwares to speed up practical inference. Another trend accelerates sparse model inference on general-purpose hardwares by adopting coarse-grained sparsity to prune or regularize consecutive weights for efficient computation. But this method often sacrifices model accuracy. In this paper, we propose a novel fine-grained sparsity approach, *Balanced Sparsity*, to achieve high model accuracy with commercial hardwares efficiently. Our approach adapts to high parallelism property of GPU, showing incredible potential for sparsity in the widely deployment of deep learning services. Experiment results show that *Balanced Sparsity* achieves up to 3.1x practical speedup for model inference on GPU, while retains the same high model accuracy as fine-grained sparsity.

Introduction

In the past few years, deep neural network (DNN) has achieved remarkable state-of-the-art results with large-scale network models for many challenging tasks, including computer vision (CV), natural language processing (NLP), and speech recognition. However, recent researches show that the significant redundancy exists in trained model weights, reaching up to 98% for popular computer vision models (Han et al. 2015; Han, Mao, and Dally 2015). Driven by the great potentials to reduce the model sizes for accelerating DNNs, a series of work (Han et al. 2015; Guo, Yao, and Chen 2016; Molchanov et al. 2016; LeCun, Denker, and Solla 1990; Engelbrecht 2001) identify and zero out the unimportant weights at a high compression ratio. Redundant weight pruning methods keep model accuracy and often benefit DNN models in cost-effective service deployment with much fewer resources.

Despite a significant reduction in operative weights, the fine-grained sparsity can only save storage costs, but hardly speed up inference due to the fragmented unstructured weights in pruned models. The irregularity and random distribution in weight matrices poorly fit current general purpose accelerators (i.e. GPU), which often advocate highly parallel computing characteristic. The speedup could be negative when the sparsity ratio is quite low and only a sparsity ratio higher than 95% can lead to speedup.(Wen et al. 2016; Wang et al. 2018) Therefore, customized hardwares (Han et al. 2016; 2017; Parashar et al. 2017) are required for the widely deployment of model pruning.

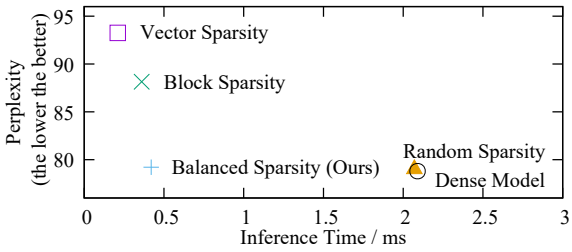


Figure 1: Perplexity and Inference Time trade-off of different sparsity patterns on the PTB dataset (Marcus et al. 1999). All the methods prune the same pre-trained LSTM model with single 1500-hidden-units cell to reach 90% sparsity.

Another research work chooses to maintain a dense structure during pruning. More specifically, pruning granularity often incorporates with neural network semantics in convolution neural network (CNN) structures, e.g., filter and channel (Li et al. 2016) and recurrent neural network (RNN) states, e.g., cell and gate (Wen et al. 2018). With coarse-grained DNN component pruned, the remaining parameters are still in a compact structure which is a quite hardware-friendly feature and make practical acceleration more possible. However, despite the notable speedup observed, the pruned models usually compromise accuracy.

Figure 1 shows the model accuracy and inference time trade-off for pruning a trained LSTM model with different sparsity patterns. The random sparsity (Han et al. 2015) approach is poor in inference speed while almost achieving the same accuracy as the original dense model. On the contrary, coarse-grained sparsity patterns, i.e both vector sparsity (Mao et al. 2017) and block sparsity (Narang, Undersander, and Diamos 2017) fit GPU architecture for matrix operation acceleration, however, losing in model accuracy.

To leverage sparsity for inference acceleration on GPUs while retaining model accuracy, we thereby propose a novel

*Equal contribution.

sparsity pattern, *Balanced Sparsity*. *Balanced Sparsity* aims at pruning model weights in a balanced structure. Instead of pruning a weight matrix in a monolithic way, we partition the weight matrix and perform independent pruning in sub-matrices. We conduct a set of experiments on typical neural networks to show the performance of our method, focusing on model accuracy and inference time. For accuracy, our experiments on three typical CV, NLP, and Speech tasks show that, we achieve less than 0.2% accuracy difference comparing with fine-grained random sparsity. For inference time, our benchmark result shows that, we achieve almost ideal performance speedup on GPU for matrix multiplication under the sparsity ratio ranging from 50% to 97%. On PTB dataset, our *Balanced Sparsity* obtains coarse-grained level speedup and keeps fine-grained level accuracy (Figure 1). Besides, a series of detailed measurements on typical networks, including VGG-16 net, LSTM model, and CTC model, show that *Balanced Sparsity* achieves 1.4x to 3.1x practical speedup in GPU inference.

Overall, we make three contributions in this paper:

- We propose a new sparsity pattern *Balanced Sparsity* and the corresponding pruning method that can both maintain model accuracy and achieve significant practical acceleration.
- We provide a matrix operation implementation based on the special architecture design inside GPU.
- Our *Balanced Sparsity* achieves the state-of-the-art practical speedup while keeps the same high model accuracy as both dense model and random sparsity approach.

Related Work

Fine-grained Sparsity

The redundancy of neural network is well recognized by LeCun et al. (LeCun, Denker, and Solla 1990) since 1990s. Recent years, fine-grained weight pruning approach removes over 90% of weight parameters in popular CV models, significantly reducing the model size for model deployment and inference services. Iterative pruning (Han et al. 2015) is firstly introduced, which prunes individual weights below a monotonically increasing threshold value and then retrains the remaining weights iteratively. Meanwhile, its capability to retain model accuracy is justified on a wide range of popular neural network models of CNN (Guo, Yao, and Chen 2016; Aghasi et al. 2017; Liu et al. 2018) and RNN (Giles and Omlin 1994; Lin et al. 2017). However, redundancy-orient pruning introduces irregularity in model. Custom hardwares (Han et al. 2016; 2017; Parashar et al. 2017) are essential to speedup the computing for fragmented random data accesses, which limit the deployment of sparse DNNs.

Coarse-grained Sparsity

Recent research observes the irregularity challenge in model sparsity and falls back to consider the support for general purposed processors. Not only weight importance but also neural network semantics are jointly considered in model pruning. The goal is to generate a sparse output while keeping dense sub-structures, therefore pruning is usually applied on coarse-grained model component. Filter and channel level sparsity for CNN (Li et al. 2016; Neklyudov et al. 2017; Wen et al. 2016), gate and cell state sparsity for RNN (Wen et al. 2018), low rank approximation (Jaderberg, Vedaldi, and Zisserman 2014; Liu et al. 2015), and block sparsity (Narang, Undersander, and Diamos 2017) are several sparsity patterns in which model structure is fully considered. As pointing out by (Mao et al. 2017; Zhu and Gupta 2017), the coarse-grained sparsity benefits computation-intensive accelerators (e.g. GPU), however, causes prominent accuracy penalty comparing with fine-grained approaches. These methods (Mao et al. 2017; Narang, Undersander, and Diamos 2017) modify the iterative pruning method to apply in consecutive weight blocks. They pick the maximum magnitude or the average magnitude of the weights within one block as a representative for the entire block. A monotonically increasing threshold is adopted also.

Methodology

Neural network pruning methods bring a restricted freedom to define the sparsity structure (e.g. hardware friendly sparsity) in weight matrices. More regular sparsity structure can increase hardware efficiency, but is also easier to destroy the original distribution of weight matrices which may hurt model accuracy significantly. Ideally, a good sparsity structure should balance model accuracy and hardware efficiency.

Our proposed sparsity pattern, *Balanced Sparsity*, achieves both high model accuracy and high hardware efficiency. In this section, we first introduce the *Balanced Sparsity* sparsity pattern and the balance-aware iterative pruning algorithm to induce *Balanced Sparsity*. Then, we use a mathematical way to prove that the influence on model accuracy is limited. Finally, we present an efficient GPU implementation for our *Balanced Sparsity*.

Balanced Sparsity

To maintain high model accuracy and achieve efficient GPU acceleration, we propose a novel fine-grained sparsity, called *Balanced Sparsity*. For weight matrices represented in *Balanced Sparsity*, each matrix row is split into multiple equal-sized blocks and each block has the same number of non-zero weights. Figure 2 shows an example of a block-balanced sparse matrix row pruned from a dense matrix row. In this example, the matrix row is split into 4 blocks, and each block has a sparsity of 50%. The balance range, i.e the length of each block, is 4. The same split method and sparsity apply to other rows in the weight matrix.

The intuitions of designing the *Balanced Sparsity* are: 1) the block partition with balanced computation work load for each block naturally fit GPUs with high practical parallelism. 2) the random distribution of non zero weights inside a block adds very few constraints on the sparsity structure and may not affect model accuracy.

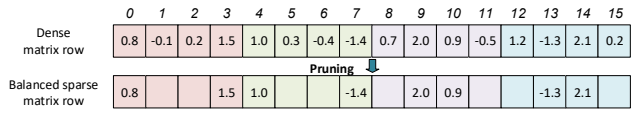


Figure 2: An example of pruning a dense matrix row to a *Balanced Sparsity* matrix row.

Balance-aware Iterative Pruning

We introduce a balance-aware iterative pruning method to induce *Balanced Sparsity* to weight matrices. For CNNs, the weights of all kernels in one convolution layer are considered as one weight matrix. Previous pruning methods usually adopt a monotonically increasing threshold value to zero out the weights less than this threshold. Those methods do not consider the distribution of non-zero values.

We use an expected sparsity instead of a threshold value to prune weights, which guarantees a balanced distribution of non-zero weights among block partitions during pruning iterations. Algorithm 1 illustrates our balance-aware iterative pruning method. In each pruning iteration, the pruning algorithm sorts the weights in each block by their absolute magnitude and then zeros out a fraction of weights with smallest absolute magnitudes under the threshold percentage. This threshold percentage is gradually increased from 0 to the target sparsity while the increase rate decreases with pruning iteration. Figure 2 illustrates a balance-aware pruning iteration with a threshold sparsity of 50%.

Algorithm 1: Balance-aware Iterative Pruning

Input: The matrix to be pruned, M;
The number of blocks per row, $BlockNum$;
The expected sparsity, $Sparsity$;
Output: The pruned matrix, M_p;

```
for M_i ∈ M.rows do
    Divide M_i into block_{i,j} (j = 1 to BlockNum);
end
tmp_sparsity = 0;
while tmp_sparsity < Sparsity do
    tmp_sparsity = GraduallyIncrease(tmp_sparsity);
    for block_{i,j} ∈ M do
        Sort elements and calculate the block internal
          threshold T_{i,j} based on tmp_sparsity;
        for each element ∈ block_{i,j} do
            prune element if |element| < T;
        end
    end
end
return the pruned matrix, M_p;
```

In our method, pruning followed by a retraining is one iteration, which is also defined in previous methods (Han et al. 2015; Mao et al. 2017; Narang, Undersander, and Diamos 2017). For multi-layer network like VGG-16 net, we adopt a straightforward strategy which separates the whole net into layers, then prune all those convolutional layers and FC layers one by one.

Asymptotic Analysis

We prove that the influence of our *Balanced Sparsity* on model accuracy is very slight, by theoretically showing that the differences between random sparsity (Han et al. 2015) and our method are negligible for practical situations. To compare the similarities and differences between these two methods, we perform a theoretical analysis on a fully-connected layer:

$$Y = W^{(0)} \cdot X + B, \tag{1}$$

where $W^{(0)}$ is an $M \times N$ matrix, X is an N-dimensional vector of input features, B is an M-dimensional vector of bias term, and Ydenotes the output of this fully-connected layer. For ease of elaboration, we assume that the bias vector B is a zero vector here.

Similar to many prior works (Hernández-Lobato and Adams 2015; Blundell et al. 2015; Salimans and Kingma 2016), we specify an independent Gaussian priors distribution $\mathcal{N}(0, \sigma_w^2)$ for each element w in $W^{(0)}$ and another $\mathcal{N}(0, \sigma_x^2)$ for each element x in input X. Then the output difference between sparse and dense FC-layer can be denoted as

$$Z^{(i)} = W^{(i)} \cdot X - W^{(0)} \cdot X = dW^{(i)} \cdot X, \quad \forall\, i \in \{1, 2\} \tag{2}$$

where $W^{(1)}$ is the matrix pruned with random sparsity and $W^{(2)}$ is the matrix pruned with *Balanced Sparsity*.

Firstly, we defined a function $H\left(k\right)$ as follows,

$$H\left(k\right) = \frac{k\left(MN - k\right)}{(MN)^3 \cdot \left[f\left(F^{-1}\left(\frac{k}{MN}\right)\right)\right]^2}, \tag{3}$$

where f and F are probability density function and cumulative distribution function of $W^{(0)}$'s Gaussian distribution, F^{-1} denotes the quantile function associated with F.

Lemma 1 *The characteristic functions of the variable z's distributions in $Z^{(i)}$,$\forall\, i \in \{1, 2\}$, are*

$$\Phi_{Z^{(1)}}\left(t\right) = \frac{\sigma_x}{r^{(1)}}\left(1 + t^2\right)^{-\frac{1}{2}} \sum_{i=1}^{r^{(1)}} H\left(i\right) \tag{4}$$

and

$$\Phi_{Z^{(2)}}\left(t\right) = \frac{\sigma_x}{r^{(1)}}\left(1 + t^2\right)^{-\frac{1}{2}} \sum_{i=1}^{r^{(1)}} H\left(\left\lceil \frac{i}{MK} \right\rceil \times MK\right), \tag{5}$$

where K is the number of balance range, $r^{(1)} = MK \cdot r^{(2)}$ is the total number of pruned elements.

With the help of Lemma 1, we get the following theorem:

Theorem 1 *The means of the variable z's distributions in $Z^{(i)}$,$\forall\, i \in \{1, 2\}$, are*

$$Mean_{Z^{(1)}}\left(z\right) = Mean_{Z^{(2)}}\left(z\right) = 0. \tag{6}$$

The variances the variable z's distributions in $Z^{(i)}$, $\forall\, i \in \{1, 2\}$, are

$$Var_{Z^{(1)}}\left(z\right) = \frac{\sigma_x}{r^{(1)}} \sum_{i=1}^{r^{(1)}} H\left(i\right) \tag{7}$$

and

$$Var_{Z^{(2)}}(z) = \frac{\sigma_x}{r^{(1)}} \sum_{i=1}^{r^{(1)}} H\left(\left\lceil \frac{i}{MK} \right\rceil \times MK\right). \quad (8)$$

As showed in equations (4) and (5), $\Phi_Z^{(1)}(t)$ and $\Phi_Z^{(2)}(t)$ have similar formulation. The mean values of random sparsity and our purposed *Balanced Sparsity* are both equal to zero. And the difference between their variances can be regarded as a limited quantization error (i.e., i v.s. $\lceil \frac{i}{MK} \rceil \times MK$). The analysis result is consistent to what we observe in real workloads as visualized in experiments. Please refer to `https://github.com/Howal/balanced-sparsity/blob/master/appendix-aaai19.pdf` for proof.

Efficient GPU Implementation

We now introduce our efficient GPU library of matrix multiplication for balanced sparse matrices.

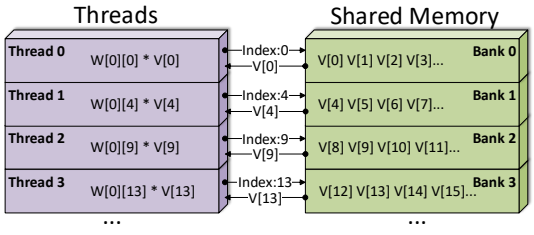


Figure 3: Parallelizing threads for efficient sparse matrix multiplication. Shared memory supplies V[0], V[4], V[9], V[13] simultaneously according to the indexes.

Our implementation first utilizes the block partition as a workload partition for GPUs to achieve high practical parallelism. Modern GPUs contain massive cores that can support thousands of threads running simultaneously. In our case, the multiplication and accumulation operations in one block partition are assigned to a single thread. The same number of non-zero values in each block partition can further increase the GPU efficiency because it makes the workloads between threads balance.

Sparse matrices after pruning lose the regular structure of dense matrices which results in irregular memory accesses in sparse matrix multiplication. Running massive threads in parallel causes concurrent random memory access problem. Improper handling of random memory accesses from various threads could stall the thread execution and decrease the performance significantly.

In order to overcome the challenge in random memory accesses, our implementation takes advantage of the shared memory in GPU architecture to support concurrent random accesses. In GPU architecture, a chunk of shared memory is visible to a fixed number of threads. To achieve high memory bandwidth for concurrent accesses, shared memory is divided into equally sized memory modules, which is called *banks* that can be accessed independently and simultaneously. Therefore, any memory load or store of n addresses that spans n distinct memory banks can be serviced simultaneously, yielding an effective bandwidth that is n times as high as the bandwidth of a single bank. In Figure 3, we use the balanced sparse matrix in Figure 2 as an example to shows how to parallelize the threads for sparse matrix multiplication. The dense vector to be multiplied is rearranged and stored in shared memory to avoid bank conflicts.

Experiments

In this section, we compare *Balanced Sparsity* against the dense model baseline, random sparsity(Han et al. 2015), block sparsity (Narang, Undersander, and Diamos 2017), and vector sparsity (Mao et al. 2017) for model accuracy. For the GPU inference performance test, we use different highly optimized libraries for different sparsity patterns. The baseline of dense matrices is tested with the cuBLAS library. For random sparse matrices, we use the cuSPARSE library. For block sparse matrices, we use an open sourced GPU library (Gray, Radford, and Kingma 2017), which is highly optimized for matrix multiplication of block sparse matrices on GPU. For balanced sparse matrices, we use our own GPU implementation as described above. Vector sparsity is not evaluated here, because there is no available GPU implementation as far as we know.

The experiments are divided into three parts. Firstly, we test our GPU implementation on a matrix multiplication benchmark. Then we apply our sparsity approach to multiple wide-used deep learning workloads, covering CV, NLP, and Speech. Finally, we investigate the feature of our sparsity pattern in further detail by visualizing the weight map and tuning the hyper-parameter, balance range. All the experiments in this section are done with a batch size of 1, the block number per row of our method is 32, and the block size of block sparsity is $8 * 8$, unless explicitly stated.

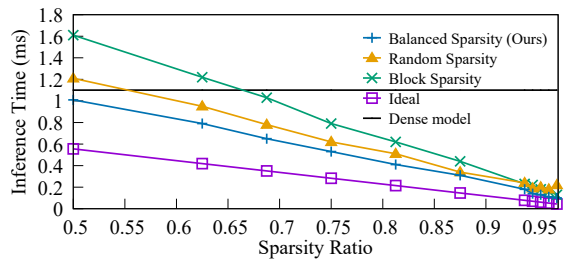
Benchmark

In order to show the hardware efficiency of our proposed *Balanced Sparsity*, we conduct a benchmark to compare the inference time of a matrix multiplication among all existing sparsity patterns. This benchmark uses a matrix size of 16384×8196.

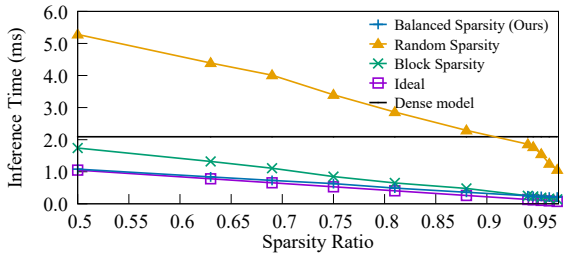
Figure 4 shows the speedup of *Balanced Sparsity* with our GPU implementation. In this benchmark of matrix multiplication, our method outperforms other sparsity patterns. When $batchsize = 1$, there is still a gap between our method and idea time, because the main benchmark bottleneck of this setting is the communication inside GPU. This disadvantage can be overcome by hiding the I/O time in more batches. For $batchsize = 8$ case, our method almost reaches the ideal inference time brought by skipping unnecessary computation. The ideal inference time (i_time) is calculated by the following equation:

$$i\_time = (d\_time - o\_time) * (1 - sparsity) + o\_time \quad (9)$$

where the d_time denotes the inference time of a dense matrix running on cuBLAS, the o_time denotes the time overhead of launching an execution kernel on GPU. Here we take 10us as o_time which is a widely used number (Chu et al. 2016).



(a) batchsize = 1



(b) batchsize = 8

Figure 4: Inference time benchmark comparisons of various sparsity patterns.

Notice that using cuSPARSE for sparse computation can achieve speedup only if the sparsity ratio is higher than around 91%, while our method is always faster than the dense baseline.

Real Workloads

In this subsection, we apply our balanced sparsity pattern to vision, language, and speech tasks. We compare the compression rate (i.e. achievable sparsity) of our balanced sparsity with other four alternatives, including dense model baseline, random sparsity, block sparsity, and vector sparsity. Random sparsity performs pruning in each independent weight matrix. Block sparsity treats a consecutive block of parameters as a pruning unit. If a pruning decision is made, the whole block weights will be removed. Vector sparsity means to consider a whole row or column in a weight matrix as a basic pruning unit.

In our pruning experiments, we apply the same hyper-parameters and fine-tune techniques to various sparsity patterns. During pruning, if the model accuracy drops significantly and cannot recover via retraining, we withdraw this pruning iteration and stop the pruning procedure. For practical speedup, we compare our GPU implementation with other available GPU implementations for dense model, random sparse model, and block sparse model.

VGG-16 on ImageNet For the vision task, we use VGG-16 network (Simonyan and Zisserman 2014) on ImageNet ILSVRC-2012 dataset (Krizhevsky, Sutskever, and Hinton 2012) to evaluate the compression rate and practical speedup. VGG-16 is a well-known network architecture which contains 13 convolutional layers and 3 FC layers,

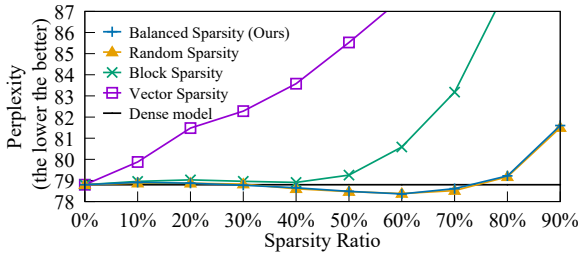


Figure 5: Sparsity-Perplexity curves of various sparsity patterns on PTB dataset.

Language Model / PTB		Inference Time / us	Sparsity
Sparsity Patterns	Dense Model	294.1	0%
	Random Sparsity	370.9	**80%**
	Block Sparsity	326.3	40%*
	Balanced Sparsity	**120.2**	**80%**

Table 1: Inference time comparisons of various sparsity patterns on PTB dataset. Our methods outperforms all the other methods. *Block Sparsity could only reach a sparsity ratio of 40% without hurting the performance.

while the dataset has 1.2M training examples and 50k validation examples.

We use random sparsity, block sparsity, and balanced sparsity to prune both convolutional and fully-connected layers of a pre-trained VGG-16 model, respectively. Then we evaluate the inference time of those pruned models with their customized GPU implementations. One popular implementation of convolution operation is using im2col that converts convolution operation to matrix-matrix multiplication (Chellapilla, Puri, and Simard 2006). The operation of a fully-connected layer is matrix-vector multiplication.

Table 2 reports the layer-wise results and the whole model result. All these three methods as well as the dense model baseline achieve similar top-5 accuracy of 90.3%, however, under different sparsity ratios. In terms of compression rate, both random sparsity and our balanced sparsity can compress the VGG-16 model with more than 12x, but block sparsity can only compress the model with less than 4x. Our GPU implementation for balanced sparsity also achieves the best practical speedup, which is 6x faster than random sparsity.

LSTM on PTB In the experiment of PTB dataset (Marcus et al. 1999), we adopts a 2-layer LSTM language model with LSTM hidden layer size of 1500. We compare *Balanced Sparsity* with other sparsity patterns by measuring the final pruned model perplexity, a metric to quantify language model quality (the lower the better).

Figure 5 shows perplexity results under different sparsity patterns. This figure shows that the perplexity curve of our balanced sparsity is very close to the perplexity curve of random sparsity. Both random sparsity and our balanced sparsity can preserve the perplexity until 80% of weights are pruned. These two patterns achieve even slightly bet-

	Dense Model		Random Sparsity		Block Sparsity		Balanced Sparsity	
	Inference Time \us	Sparsity	Inference Time \us	Sparsity	Inference Time \us	Sparsity	Inference Time \us	Sparsity
conv1_1	144.0	-	714.7	42%	78.3	31%	254.7	34%
conv1_2	612.5	-	2578.0	88%	949.4	56%	1018.4	68%
conv2_1	393.5	-	1842.5	70%	356.2	41%	474.4	65%
conv2_2	588.2	-	4640.0	71%	639.9	38%	557.0	71%
conv3_1	305.0	-	2668.6	57%	286.2	30%	371.4	45%
conv3_2	584.4	-	3768.9	84%	362.6	56%	396.5	79%
conv3_3	584.4	-	4257.4	71%	490.3	35%	355.7	88%
conv4_1	333.3	-	2005.3	79%	237.8	41%	295.4	86%
conv4_2	623.0	-	3196.0	86%	316.6	57%	366.2	91%
conv4_3	623.0	-	3205.9	85%	500.5	38%	396.5	88%
conv5_1	211.0	-	920.1	88%	170.7	41%	129.9	86%
conv5_2	211.0	-	926.3	91%	132.9	52%	126.4	90%
conv5_3	211.0	-	1053.6	89%	163.8	36%	110.2	95%
fc6	979.9	-	1084.6	93%	841.8	75%	231.1	93%
fc7	265.5	-	251.0	93%	238.6	75%	70.3	93%
fc8	144.5	-	294.5	75%	120.6	60%	58.9	75%
Total*	6814.141	-	33407.4	91.8%	5886.1	71.7%	**5213.0**	**92.0%**

Table 2: Inference time and sparsity comparisons of various sparsity patterns on VGG-16. Our balanced sparsity and customized GPU implementation achieve the best compression rate and practical speedup. *The time cost of other layers in VGG-16, such as pooling and batch normalization, is about 230us, which is less than 3% of entire inference time.

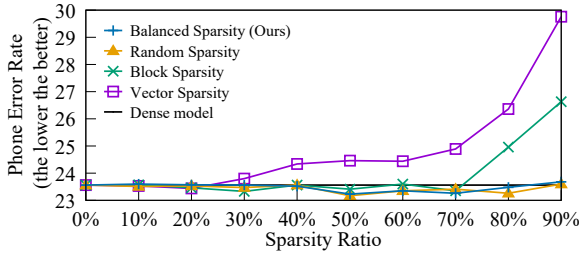


Figure 6: Sparsity - Phone Error Rate curves of various sparsity patterns on TIMIT dataset.

Speech Recognition / TIMIT		Inference Time / us	Sparsity
Sparsity Patterns	Dense Model	117.9	0%
	Random Sparsity	190.5	**87.5%**
	Block Sparsity	212.8	70%*
	Balanced Sparsity	**83.9**	**87.5%**

Table 3: Inference time comparisons of various sparsity patterns on TIMIT dataset. *Notice that the sparsity percentage is chosen based on the accuracy experiment in Figure 6. Block Sparsity could only reach a sparsity ratio of 70% without hurting the performance.

ter model quality, compared to the original one even around 60% sparsity. The perplexity of vector sparsity starts to increase significantly at a very low sparsity ratio. The perplexity of block sparsity starts to increase at a sparsity of 40%. In summary, our balanced sparsity has almost the same efficacy as random sparsity and outperforms both vector sparsity and block sparsity in terms of achievable accuracy and sparsity during pruning.

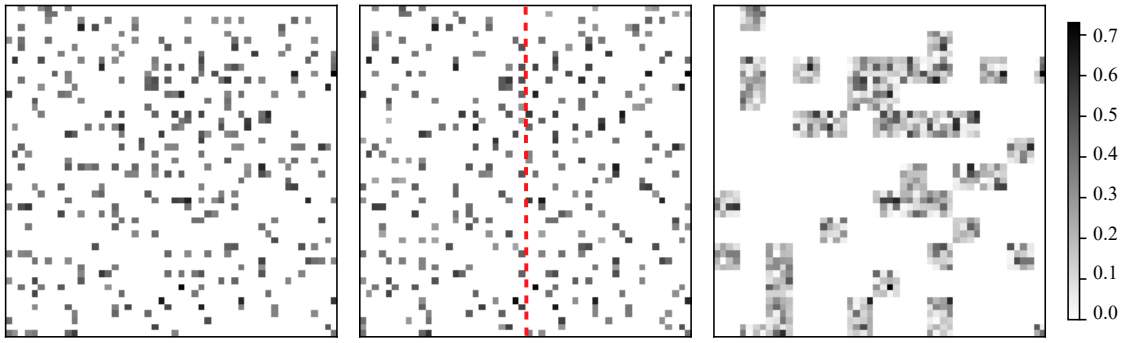
We also compare the inference time of our balanced sparsity with dense baseline, random sparsity, and block sparsity. Table 1 shows the speedup results. For the PTB LSTM model, our GPU implementation for balanced sparsity achieves 3.1x speedup compared to the random sparse model running on cuSPARSE, 2.7x speedup compared to the block sparse model running on block sparse library, 2.5x speedup compared to the baseline dense model running on cuBLAS.

CTC on TIMIT We further examine our *Balanced Sparsity* on the TIMIT dataset, which is a read speech benchmark and especially designed for acoustic-phonetic studies. A CTC (connectionist temporal classification) model (Graves et al. 2006) is used here, which mainly contains a Bi-LSTM (Bidirectional Long Short-Term Memory) cell with a hidden size of 1024. The settings of different sparsity patterns are the same as mentioned in previous subsection.

For the TIMIT Bi-LSTM model, Figure 6 shows the perplexity results under different sparsity patterns and Table 3 shows the inference time of different sparsity patterns. We get the same conclusions as the experiment of PTB LSTM model. In terms of pruning efficacy, our balanced sparsity is similar to random sparsity and outperforms vector sparsity and block sparsity. In terms of GPU acceleration, our implementation for balanced sparsity achieves around 1.4x-2.6x speedup compared to others.

Discussions

Visualization We use the visualization method to understand why we can achieve a high accuracy close to random sparsity. Figure 7 shows a random-selected 64 $\times$ 64



(a) Random Sparsity (b) Balanced Sparsity (c) Block Sparsity

Figure 7: Weight map visualizations after applying random sparsity, *Balanced Sparsity*, and block sparsity (sparsity = 90%). In (b), each row contains two block partitions (i.e., left side and right side of the dotted line).

Model		Perplexity on Sparsity		
		60%	70%	80%
Block Sparsity	block size: 4*4	80.6	83.2	88.1
	block size: 8*8	82.4	86.4	95.2
	block size: 16*16	83.7	88.3	99.5
Balanced Sparsity	balance range: 25	78.3	78.6	79.4
	balance range: 50	78.4	78.7	79.2
	balance range: 100	78.4	78.6	79.2

Table 4: Perplexity results on PTB dataset with different block size / balance range settings.

block from the same position of 1500 $\times$ 1500 weight matrix in our LSTM experiment, under the sparsity ratio of 90%. The colored regions of the figure indicate non-zero parameters. Figure 7c shows that, for block sparsity, the remaining blocks are randomly distributed, while intra-block, it is a dense weight matrix, suitable for parallel acceleration. After pruning, the weight map of *Balanced Sparsity* is very similar to random sparsity. Thus, *Balanced Sparsity* and random sparsity can maintain good accuracy. Besides, the visualization also indicates that *Balanced Sparsity* is in a balanced weight distribution, compared with random sparsity, which provides a valuable feature for GPU inference acceleration. In other words, each weight matrix row contains two blocks while each block contains three non-zero weights.

Sensitivity We also study the sensitivity of our *Balanced Sparsity* method by tuning the balance range. To show this more clearly, we take the block size of block sparsity as a comparison. Table 4 shows how the pruned model accuracy changes based on both different sparsity ratio and different balance ranges / block sizes. In this case, *Balanced Sparsity* keeps the same model accuracy regardless of the change of balance range value. Even a very small balance range value (i.e. 25) cannot hurt the model accuracy. On the contrary, for block sparsity, the light change of block size can lead to a significant perplexity increase.

Conclusion

In this work, we have proposed *Balanced Sparsity*, a new fine-grained sparsity pattern to represent weight matrices in deep neural networks. Experimental results on a set of neural networks show that *Balanced Sparsity* achieves almost the same model accuracy as random sparsity with various sparsity ratios. Our measurements in widely-used deep learning workloads show that our efficient GPU implementation for *Balanced Sparsity* can achieve significant speedup, up to 3.1x on GPU without accuracy loss. Our method shows not only the feasibility, but also the high potentials, for widely deployment of sparsity in neural network inference.

Acknowledgements

We would like to thank Dr. Ming Wu from Conflux and Dr. Yun Wang from Microsoft Research Asia for their valuable suggestions on improving this paper. We also thank the anonymous reviewers for their insightful feedbacks and comments. Shijie Cao was partly supported by National Nature Science Foundation of China (No.61772159).

References

Aghasi, A.; Abdi, A.; Nguyen, N.; and Romberg, J. 2017. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems*, 3180–3189.

Blundell, C.; Cornebise, J.; Kavukcuoglu, K.; and Wierstra, D. 2015. Weight uncertainty in neural network. In *International Conference on Machine Learning*, 1613–1622.

Chellapilla, K.; Puri, S.; and Simard, P. 2006. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft.

Chu, C.-H.; Hamidouche, K.; Venkatesh, A.; Awan, A. A.; and Panda, D. K. 2016. Cuda kernel based collective reduction operations on large-scale gpu clusters. In *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*, 726–735. IEEE.

Engelbrecht, A. P. 2001. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE transactions on Neural Networks* 12(6):1386–1399.

Giles, C. L., and Omlin, C. W. 1994. Pruning recurrent neural networks for improved generalization performance. *IEEE transactions on neural networks* 5(5):848–851.

Graves, A.; Fernández, S.; Gomez, F.; and Schmidhuber, J. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, 369–376. ACM.

Gray, S.; Radford, A.; and Kingma, D. P. 2017. Gpu kernels for block-sparse weights. Technical report, Technical report, OpenAI.

Guo, Y.; Yao, A.; and Chen, Y. 2016. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, 1379–1387.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, 1135–1143.

Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M. A.; and Dally, W. J. 2016. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, 243–254. IEEE.

Han, S.; Kang, J.; Mao, H.; Hu, Y.; Li, X.; Li, Y.; Xie, D.; Luo, H.; Yao, S.; Wang, Y.; et al. 2017. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 75–84. ACM.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Hernández-Lobato, J. M., and Adams, R. 2015. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, 1861–1869.

Jaderberg, M.; Vedaldi, A.; and Zisserman, A. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

LeCun, Y.; Denker, J. S.; and Solla, S. A. 1990. Optimal brain damage. In *Advances in neural information processing systems*, 598–605.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Lin, J.; Rao, Y.; Lu, J.; and Zhou, J. 2017. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, 2178–2188.

Liu, B.; Wang, M.; Foroosh, H.; Tappen, M.; and Pensky, M. 2015. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 806–814.

Liu, X.; Pool, J.; Han, S.; and Dally, W. J. 2018. Efficient sparse-winograd convolutional neural networks. *arXiv preprint arXiv:1802.06367*.

Mao, H.; Han, S.; Pool, J.; Li, W.; Liu, X.; Wang, Y.; and Dally, W. J. 2017. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*.

Marcus, M.; Santorini, B.; Marcinkiewicz, M. A.; and Taylor, A. 1999. Treebank-3 ldc99t42. *CD-ROM. Philadelphia, Penn.: Linguistic Data Consortium*.

Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; and Kautz, J. 2016. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*.

Narang, S.; Undersander, E.; and Diamos, G. 2017. Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*.

Neklyudov, K.; Molchanov, D.; Ashukha, A.; and Vetrov, D. P. 2017. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, 6778–6787.

Parashar, A.; Rhu, M.; Mukkara, A.; Puglielli, A.; Venkatesan, R.; Khailany, B.; Emer, J.; Keckler, S. W.; and Dally, W. J. 2017. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 27–40. ACM.

Salimans, T., and Kingma, D. P. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, 901–909.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Wang, H.; Zhang, Q.; Wang, Y.; and Hu, H. 2018. Structured probabilistic pruning for convolutional neural network acceleration. In *British Machine Vision Conference 2018, BMVC 2018, Northumbria University, Newcastle, UK, September 3-6, 2018*, 149.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, 2074–2082.

Wen, W.; He, Y.; Rajbhandari, S.; Zhang, M.; Wang, W.; Liu, F.; Hu, B.; Chen, Y.; and Li, H. 2018. Learning intrinsic sparse structures within long short-term memory. In *International Conference on Learning Representations*.

Zhu, M., and Gupta, S. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.