Balanced Sparsity for Efficient DNN Inference on GPU

Neural Network Acceleration Study Season #2
20200721 박준상

Contents

- Pruning & Motivation
- Methodology
  - Algorithm
  - GPU implement
- Experiments & Results
  - Benchmarks
  - VGG-16
  - LSTM
    - PTB dataset
    - TIMIT dataset
- Discussions
  - Weight visualization
  - Sensitivity check
- Summary

Pruning & Motivation

- 딥러닝 고속화
  - 모델 파라메터수 줄이기
  - FLOPs수 줄이기
  - 모델 파일 사이즈 줄이기
  - 추론 시간 단축
  - 훈련 시간 단축

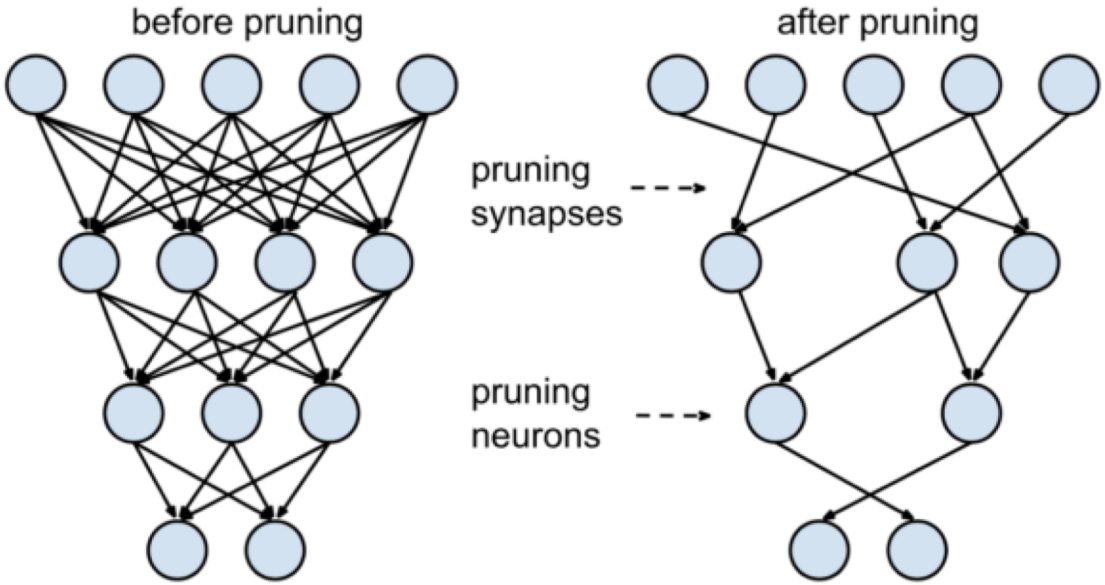
Pruning & Motivation

- 딥러닝 모델 고속화
  - Pruning
  - Distillation
  - Quantization
  - Factorization
  - Neural Architecture Search(NAS)
  - Early termination, Dynamic Computation Graph

Pruning & Motivation

- Pruning

Pruning시 파라메터들을 sparse하게 했을 때,
모델을 경량화 시킬수 있으나 accuracy를 낮출 수 있음:
얼마나 경량화 시키며 accuracy는 유지하는가가 중요

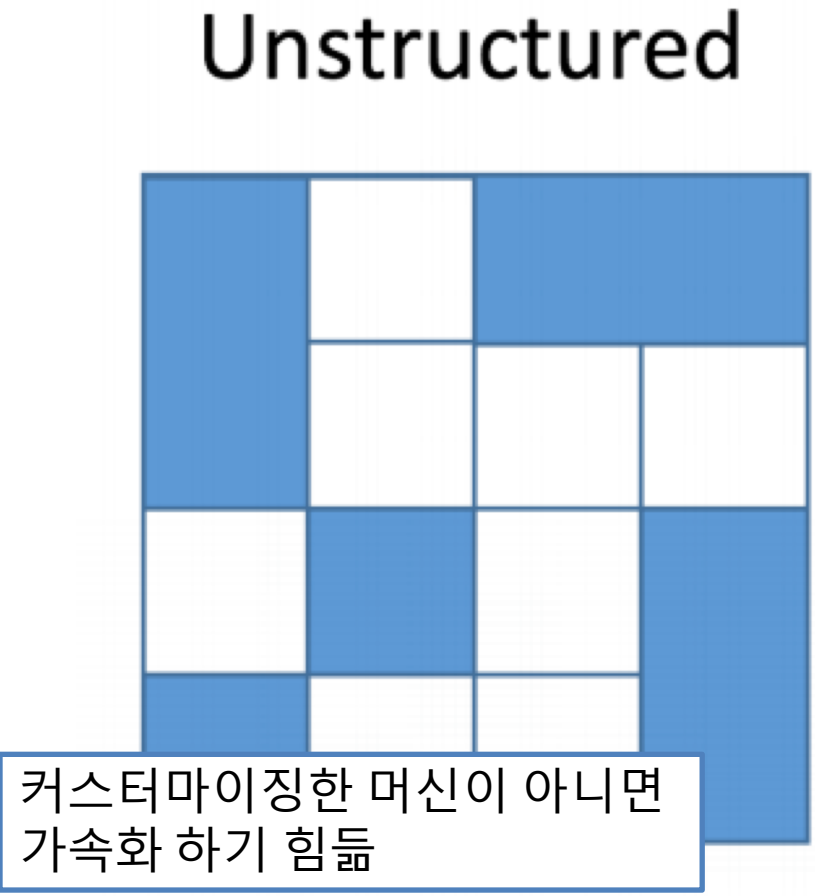


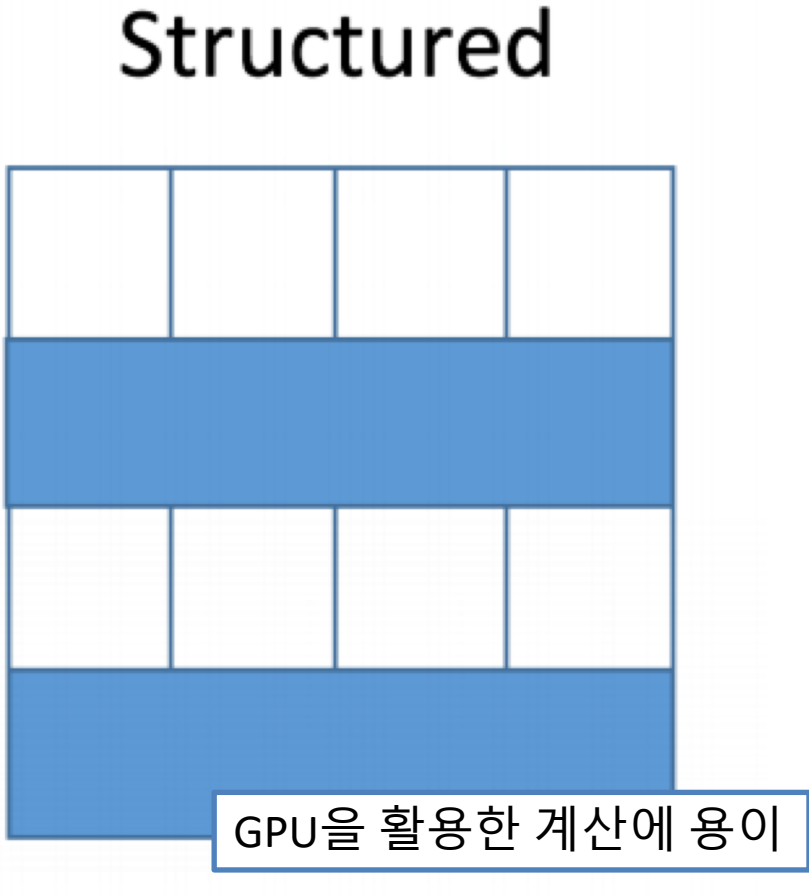
[Lecun et al. NIPS'89]

[Han et al. NIPS'15]

Pruning & Motivation

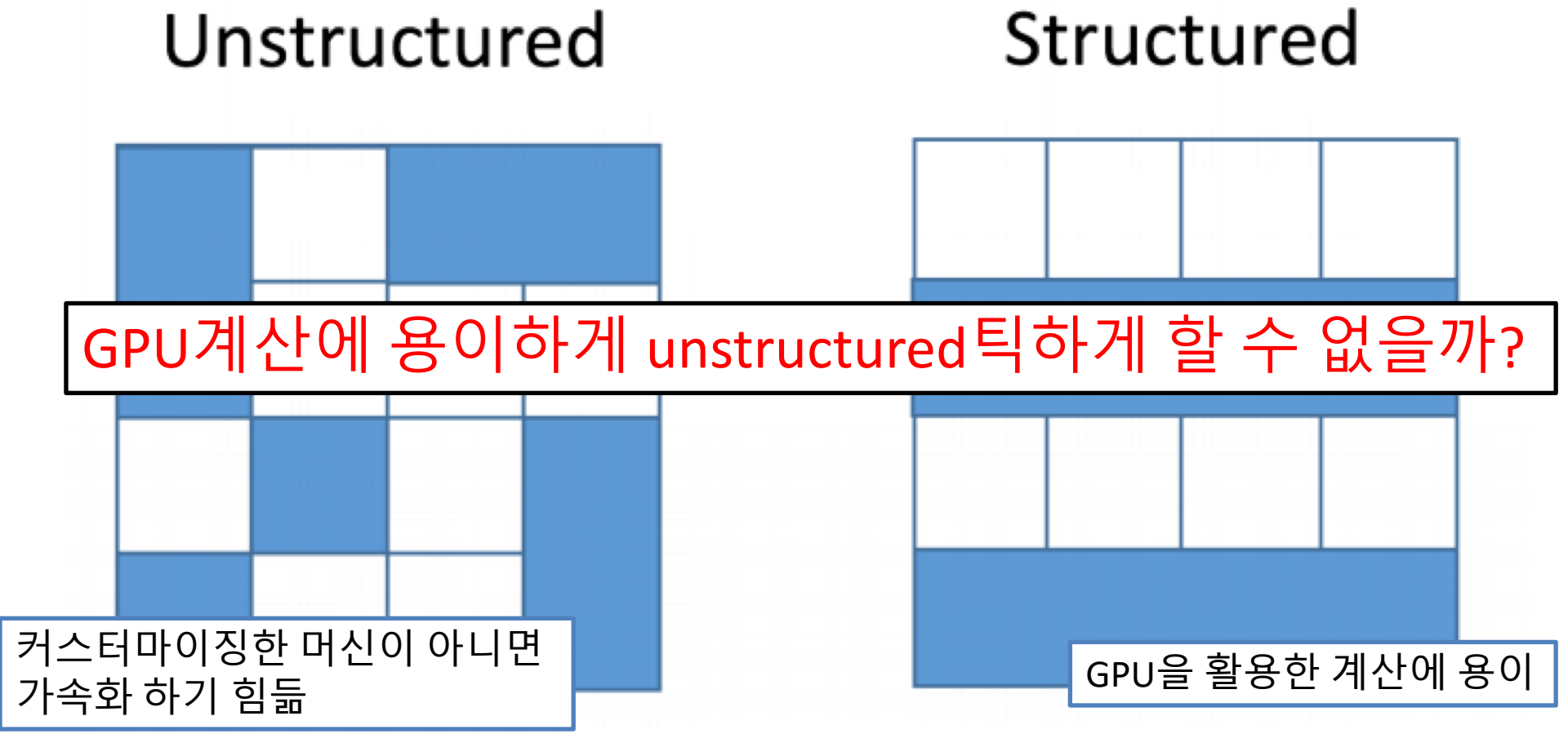
- Unstructured vs Structured





Pruning & Motivation

- Unstructured vs Structured



Methodology

Algorithm:Balance-aware Iterative Pruning

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;
```

Methodology

Algorithm:Balance-aware Iterative Pruning

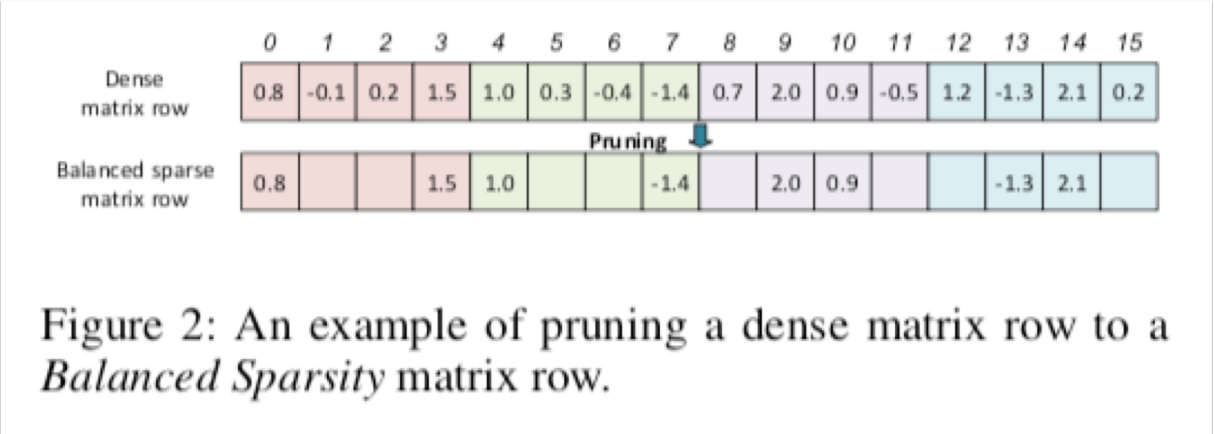
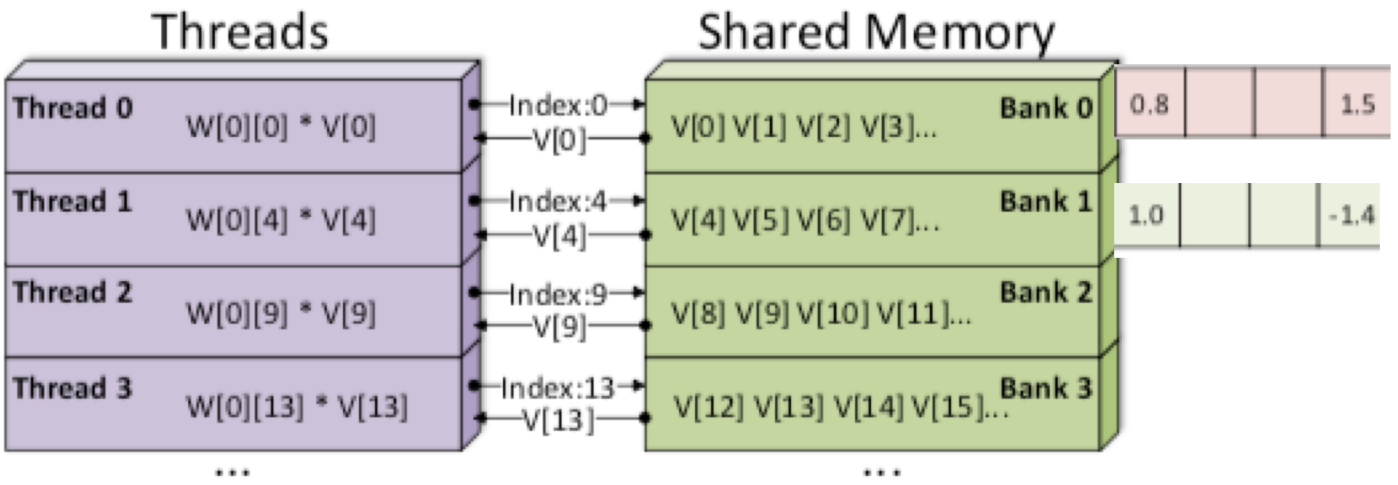


Figure 2: An example of pruning a dense matrix row to a *Balanced Sparsity* matrix row.

- 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.
- 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 percent- age. This threshold percentage is gradually increased from 0 to the target sparsity while the increase rate decreases with pruning iteration.

Methodology

Efficient GPU Implementation



- utilizes the block partition as a workload partition for GPUs to achieve high practical parallelism.
- one block partition are assigned to a single thread.
- shared memory is divided into equally sized memory modules, which is called banks that can be accessed independently and simultane- ously.

Experiments & Results

- VS 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.
- batch size of 1, the block number per row of our method is 32, and the block size of block sparsity is $8 * 8$
- 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)
- Vector sparsity is not evaluated here, because there is no available GPU implementation as far as we know.

Experiments & Results

Benchmark

- uses a matrix size of 16384 × 8196.

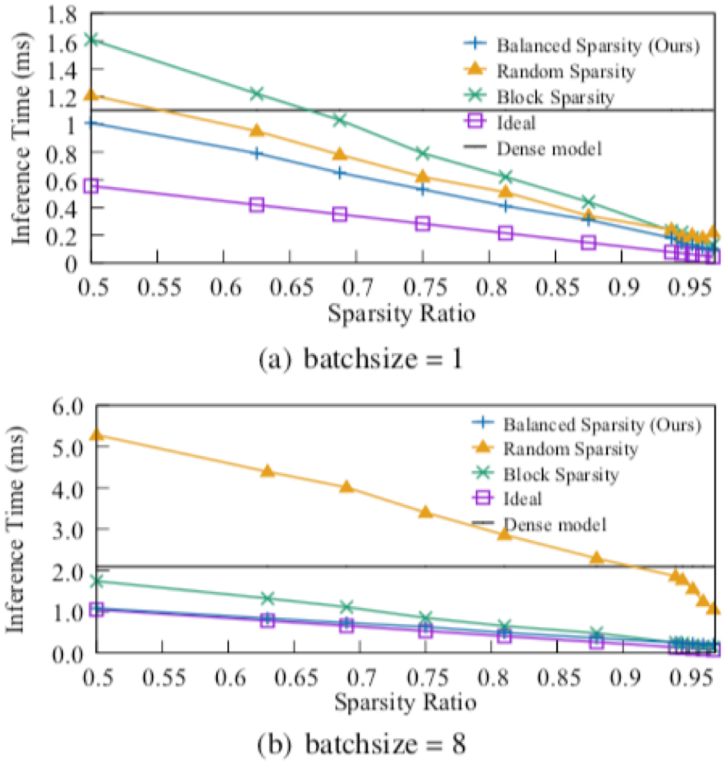


Figure 4: Inference time benchmark comparisons of various sparsity patterns.

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

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

Experiments & Results

VGG-16

- 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.
- dataset has 1.2M training examples and 50k validation examples.

Experiments & Results

VGG-16

	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.

- Compression: more than 12x, but block sparsity can only compress the model with less than 4x
- Speed: 6x faster than random sparsity.

Experiments & Results

LSTM(PTB dataset)

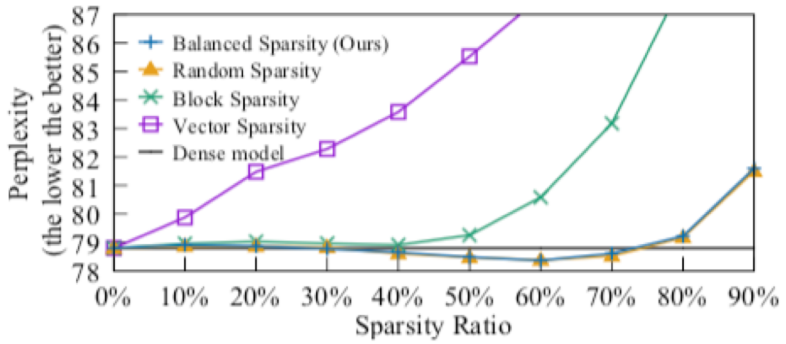


Figure 5: Sparsity-Perplexity curves of various sparsity patterns on PTB dataset.

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%**

- 2-layer LSTM language model with LSTM hidden layer size of 1500
- measuring the final pruned model perplexity(the lower the better)
- random sparsity and balanced sparsity can preserve the perplexity until 80% of weights are pruned. These two patterns achieve even slightly better model quality, compared to the original one even around 60% sparsity

Experiments & Results

LSTM(TIMIT dataset)

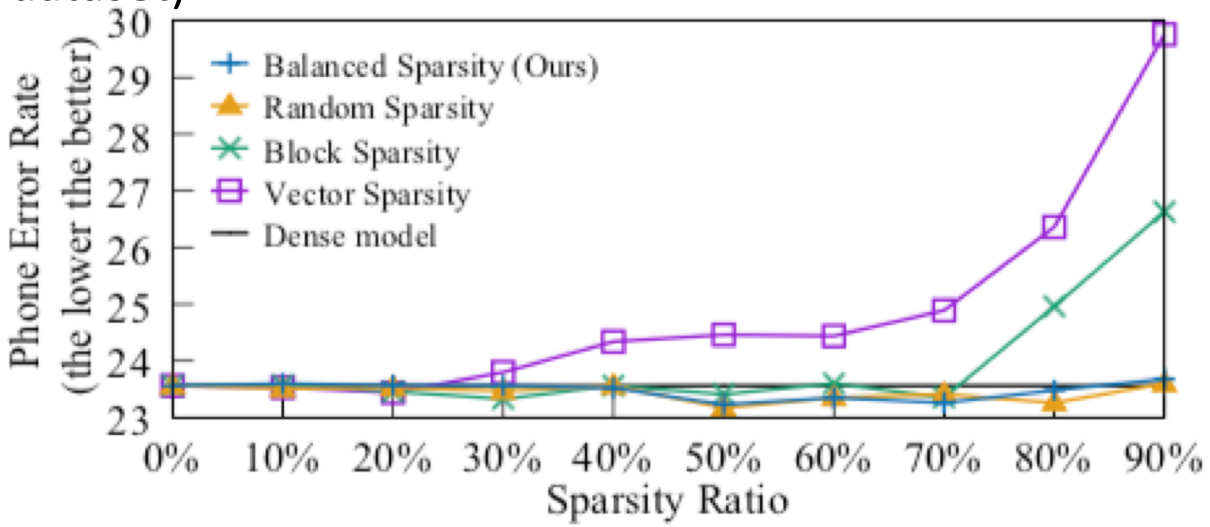


Figure 6: Sparsity - Phone Error Rate curves of various sparsity patterns on TIMIT dataset.

- CTC (connectionist temporal classification) model (Graves et al. 2006)
  - mainly contains a Bi-LSTM (Bidirectional Long Short-Term Memory) cell with a hidden size of 1024
- achieves around 1.4x-2.6x speedup compared to others.

Discussions

Weight visualization

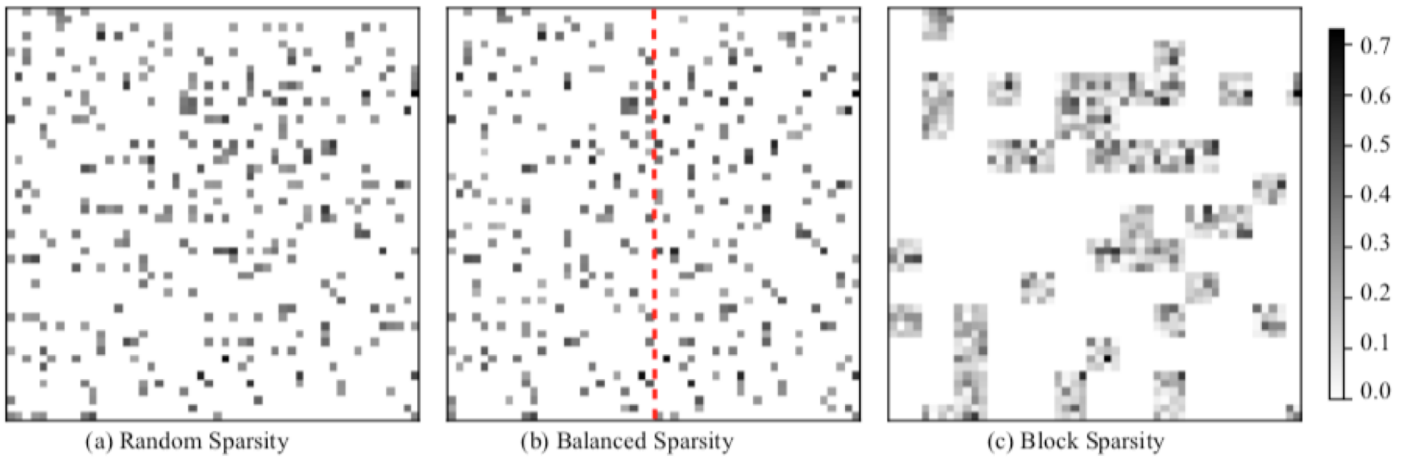


Figure 7: Weight map visualizations after applying random sparsity, *Balanced Sparsity*, and block sparsity (sparsity = 90%). In (b), each row contains two block partitions (i.e., left side and right side of the dotted line).

- random-selected 64 × 64 block from the same position of 1500 × 1500 weight matrix in our LSTM experiment, under the sparsity ratio of 90%
- colored regions of the figure indicate non-zero parameters
- Balanced Sparsity is in a balanced weight distribution, compared with random sparsity, which provides a valuable feature for GPU inference acceleration.

Discussions

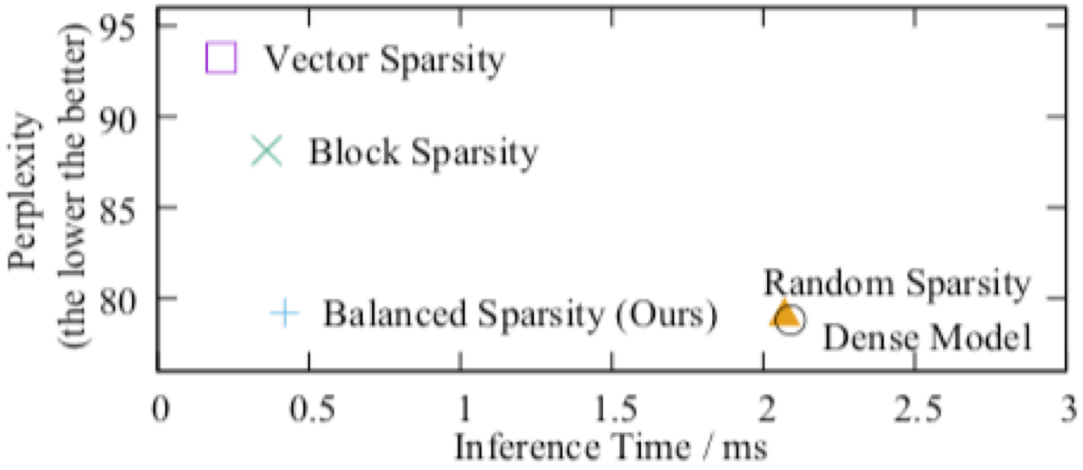
Sensitivity

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.

- 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
- On the contrary, for block sparsity, the light change of block size can lead to a significant perplexity increase.

Summary



- Pruning은 1990년도부터 연구되어 왔고 지금도 활발히 연구가 진행 중이다
- Unstructured sparse는 GPU를 활용한 딥러닝 가속에 적합하지 않은 한계가 있으나 본 논문에서는 한 블럭 내에서 균일한 수의 weight을 sparse하게 함으로써 GPU를 활용한 딥러닝 가속이 가능하게 했다
- Compression-accuracy trade-off는 dense model, block-sparse, vector-sparse, random-sparse에 비해서 가장 작은(좋은) 결과를 보여주었다.
- 3.1x practical speedup for model inference on GPU, while retains the same high model accuracy as fine grained sparsity.