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

* Equal contribution

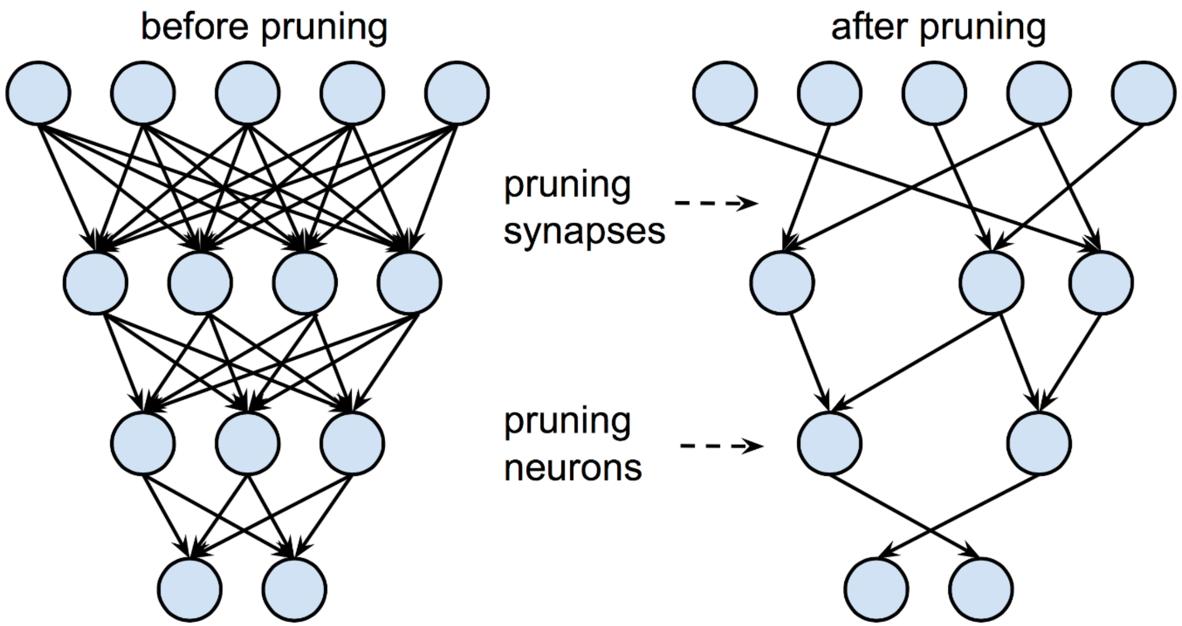
Ways to Compress / Speedup DL

- Weight Pruning / Sparsity
- Quantization
- Matrix Factorization (SVD)
- Distilling / Mimic
- Smaller Network / SqueezeNet / MobileNet

What is Pruning / Sparsity

- Why - Network Redundancy
  - "Dead" / little activation
  - Uncorrelated with output
  - Correlated with other neurons

- How - Pruning



Related Work

Fine-grained Sparsity

- Random Sparsity
  - Learning both weights and connections (Han et al. 2015)
  - Dynamic Network Surgery (Guo et al. 2016)
  - Net-Trim: Convex Pruning (Alireza et al. 2017)

Coarse-grained Sparsity

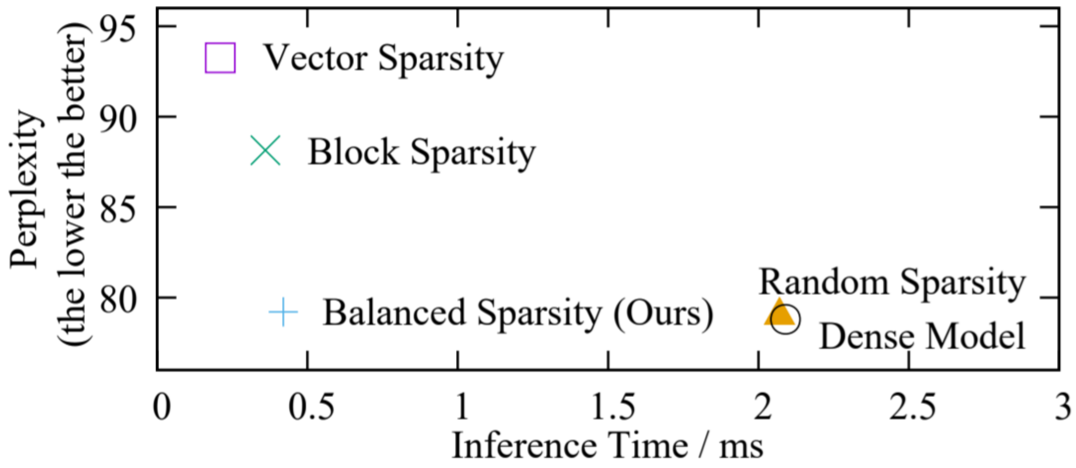
- Block/Vector Sparsity
  - Vector-Sparse (Mao et al. 2017)
  - Block-Sparse (Narang et al. 2017)
- Filter/Channel-Level Sparsity
  - Learning Structured Sparsity (Wen et al. 2016)
  - Structured Bayesian Pruning (Neklyudov et al. 2017)

High Accuracy V.S. Practical Speedup

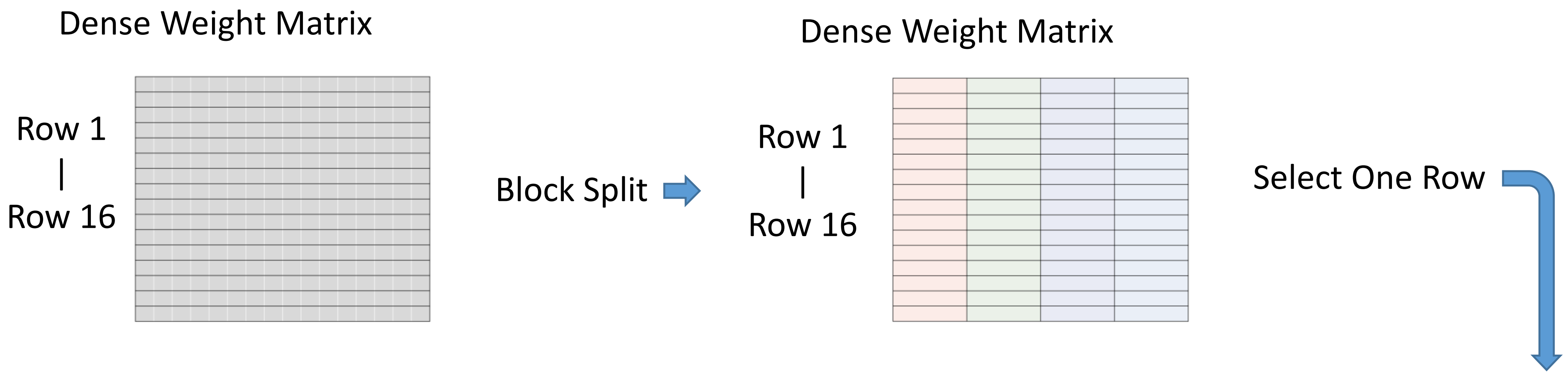
Balanced Sparsity on GPU

We propose a novel fine-grained sparsity pattern – Balanced Sparsity and the corresponding pruning method based on the special architecture design inside GPU.

- Maintain model accuracy
- Achieve significant practical speedup
- Flexible for any kind of networks



Balanced Sparsity



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Dense matrix row	0.8	-0.1	0.2	1.5	1.0	0.3	-0.4	-1.4	0.7	2.0	0.9	-0.5	1.2	-1.3	2.1	0.2

Pruning

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Balanced sparse matrix row	0.8			1.5	1.0			-1.4		2.0	0.9			-1.3	2.1	

For CNNs, the weights of all kernels in one convolution layer are considered as one weight matrix.

Iterative Pruning

$tmp_{sparsity}$

$GraduallyIncrease()$

$while\{\ldots\}do\{\ldots\}$

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 $M_i \in M.rows$ **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 $block_{i,j} \in M$ **do**
    Sort elements and calculate the block internal threshold $T_{i,j}$ based on $tmp_{sparsity}$;
    for $each\ element \in block_{i,j}$ **do**
      prune $element$ **if** $|element| < T$;
    end
  end
end
return the pruned matrix, M_p;

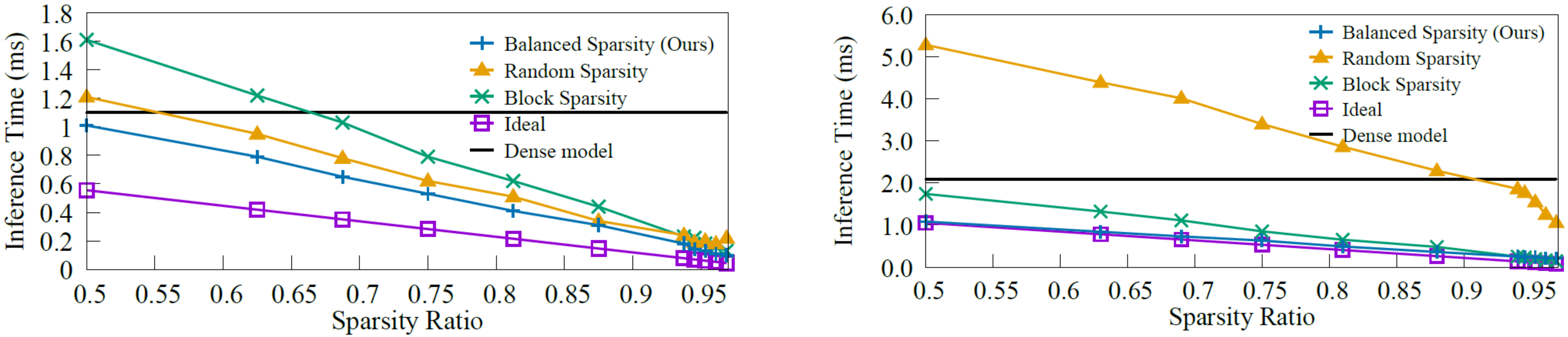
Experiments

- Dense Model (baseline) – cuBLAS
- Random Sparsity – cuSPARSE
- Block Sparsity – an open sourced GPU library from OpenAI
- Balanced Sparsity – our own GPU implementation
- Vector sparsity – only evaluated for accuracy
- Same hyper-parameters and fine-tune techniques

Experiments - Benchmark

- This benchmark uses a matrix size of 16384x8196.
- The ideal inference time:

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



(a) batchsize = 1

(b) batchsize = 8

Experiments – Real Workloads

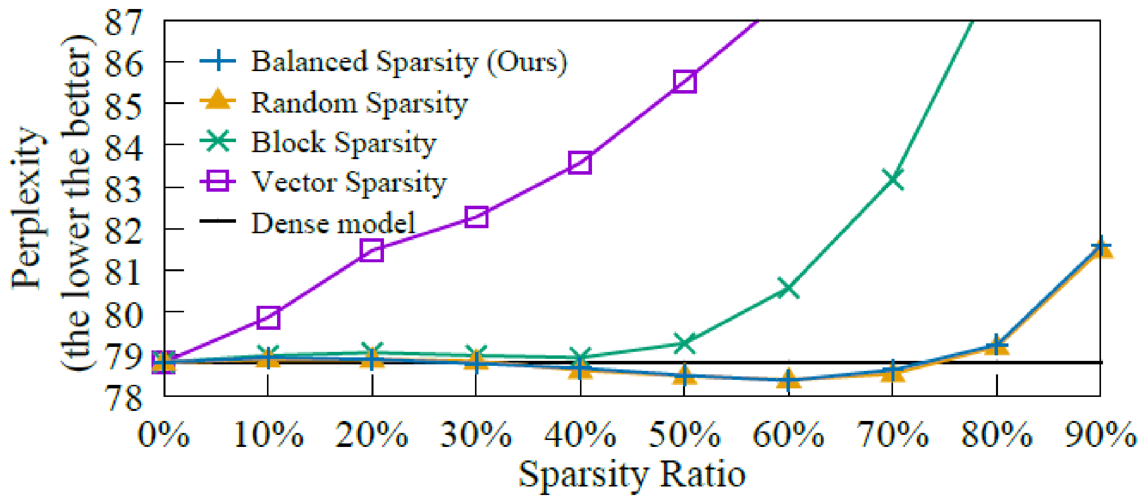
VGG16 on ImageNet

	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%**

Experiments – Real Workloads

LSTM on PTB

- A 2-layer LSTM language model with LSTM hidden layer size of 1500.
- Perplexity - a metric to quantify language model quality (the lower the better)

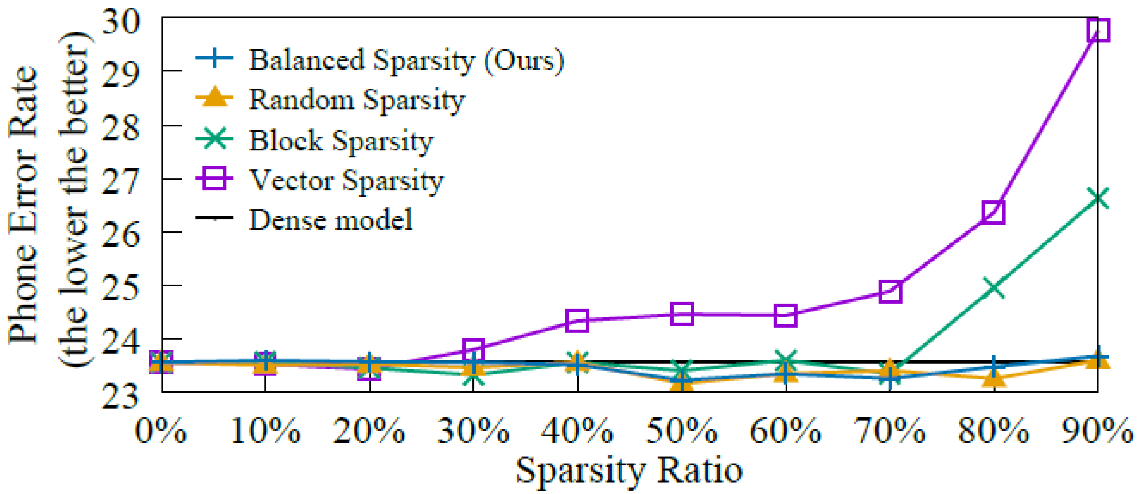


Language Model / PTB		Inference Time / us	Sparsity
Sparsity Patterns	Dense Model	294.1	0%
	Random Sparsity	370.9	**80%**
	Block Sparsity	326.3	40%*
	Balanced Sparsity	**120.2**	**80%**

Experiments – Real Workloads

CTC on TIMIT

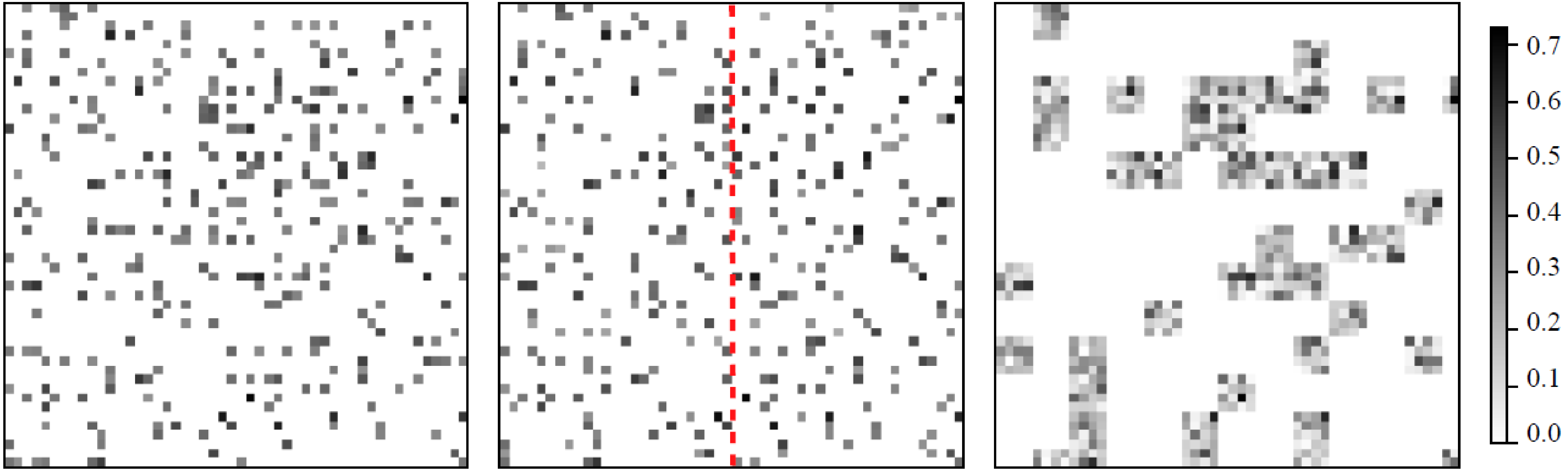
- TIMIT is a read speech benchmark
- CTC contains a Bi-LSTM cell with a hidden size of 1024



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%**

Discussions - Visualization

- Random-selected 64x64 block from the same position of 1500x1500 weight matrix in our LSTM experiment with 90% sparsity ratio.



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

Discussions - Sensitivity

- Perplexity results on PTB dataset with different block size / balance range settings.

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

Conclusion

- We proposed a new fine-grained sparsity pattern to represent weight matrices in deep neural networks.
- Experimental results on a set of neural networks show that our method achieves almost the same model accuracy as random sparsity with various sparsity ratios.
- Our method shows not only the feasibility, but also the high potentials, for widely deployment of sparsity in neural network inference.

Thanks!