Balanced Sparsity for Efficient DNN Inference on GPU

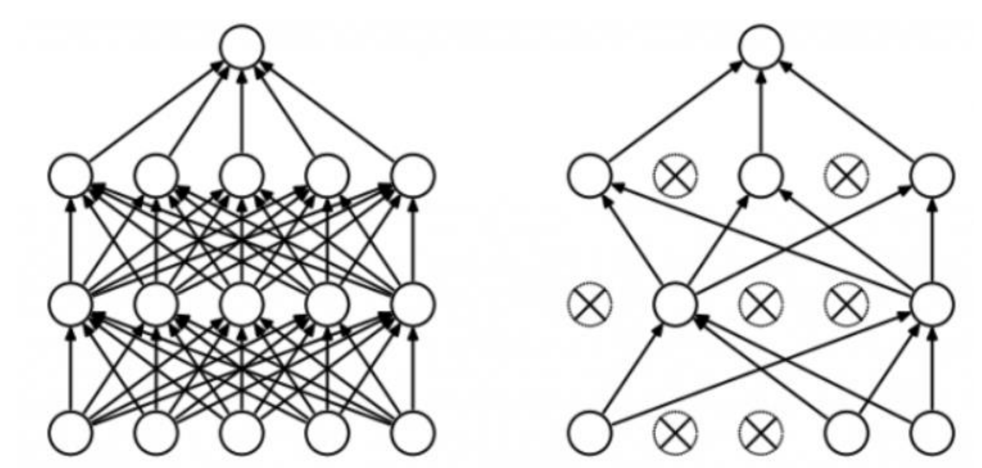
Zhuliang Yao♠†*, Shijie Cao ♣†*, Wencong Xiao ♦†, Chen Zhang†, Lanshun Nie ♣

♠ Tsinghua University, † Microsoft Research, * Equal Contribution
♣ Harbin Institute of Technology, ♦ Beihang University

Microsoft Research

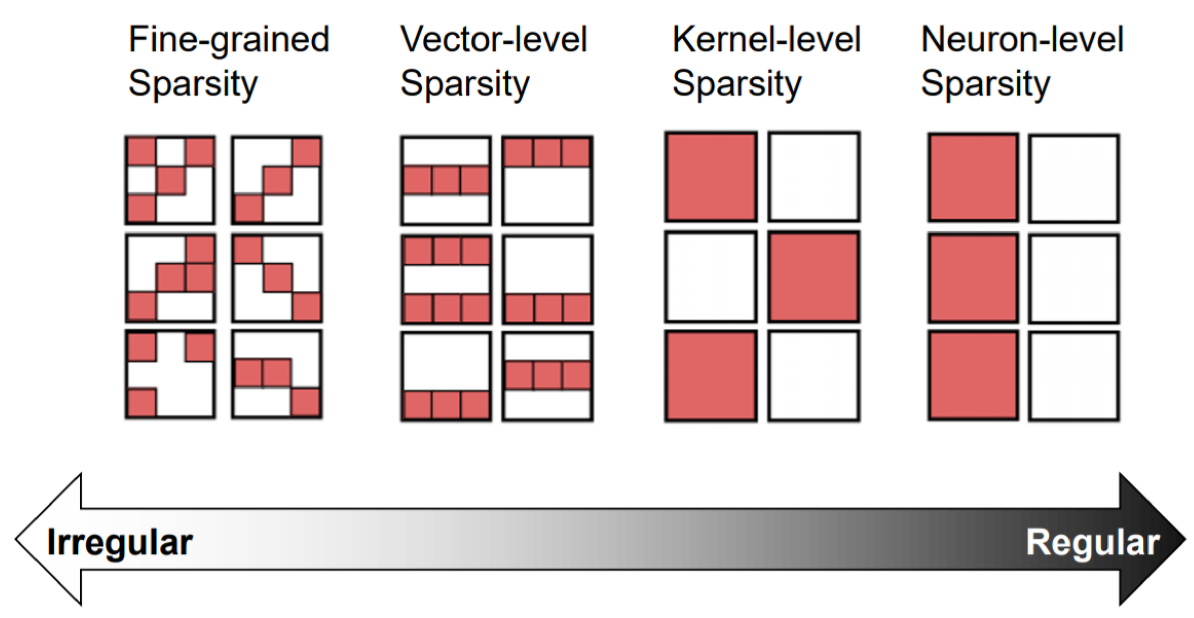
Sparsity in Deep Learning

Redundancy in DNNs



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

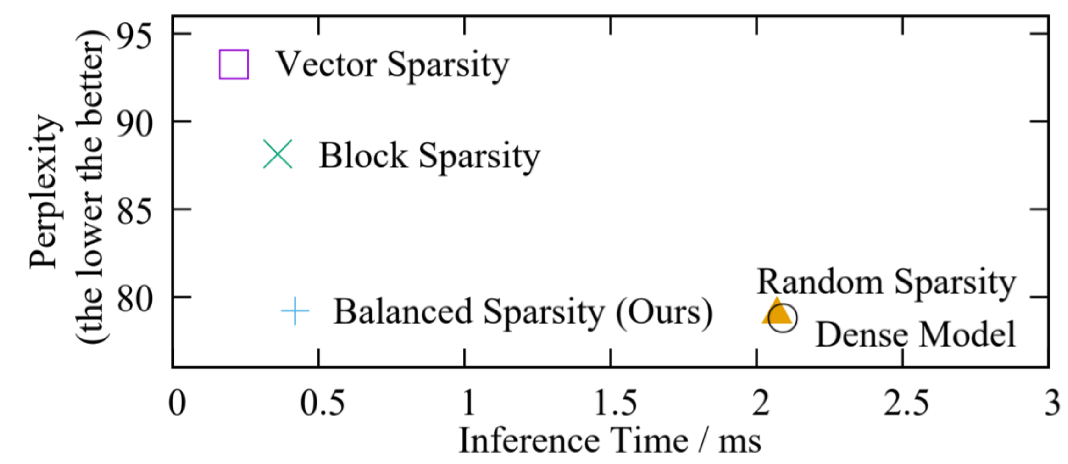
Speedup and Accuracy Tradeoff



- High model accuracy
- High compression rate
- Irregular pattern
- Difficult to accelerate

- Low model accuracy
- Low compression rate
- Regular pattern
- Easy to accelerate

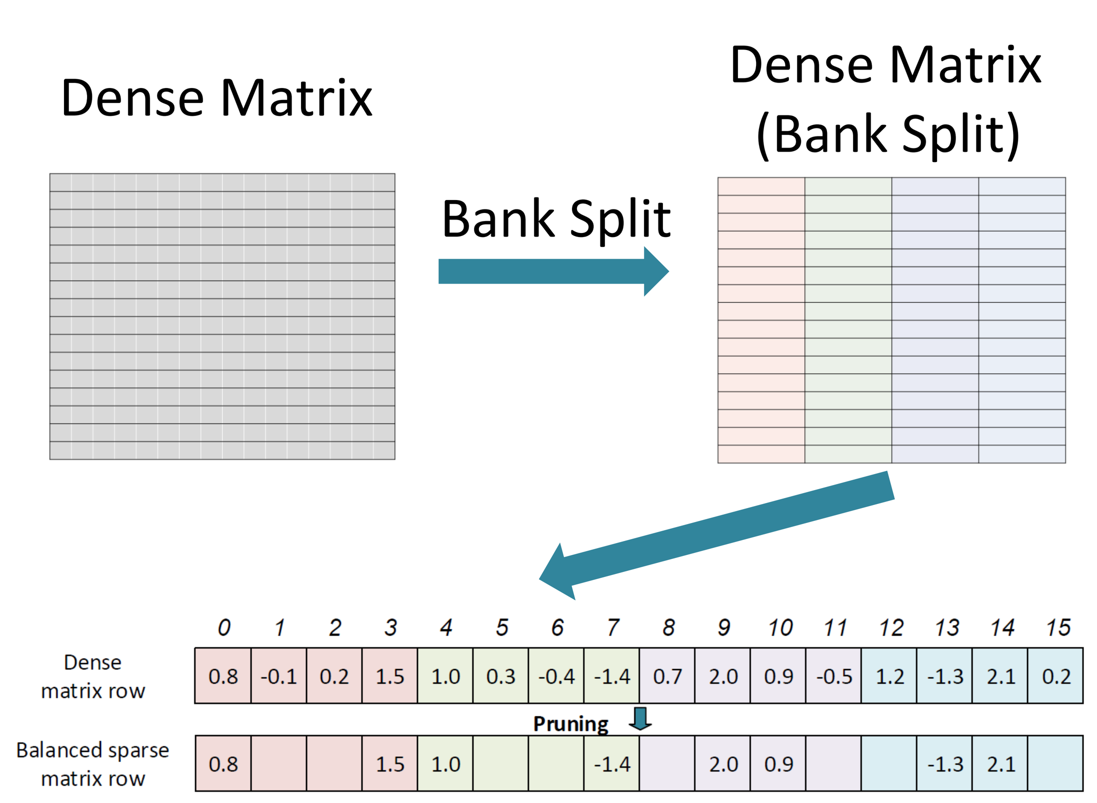
Our Method



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

Methodology

Balanced Sparsity



Traverse each row in Dense Matrix

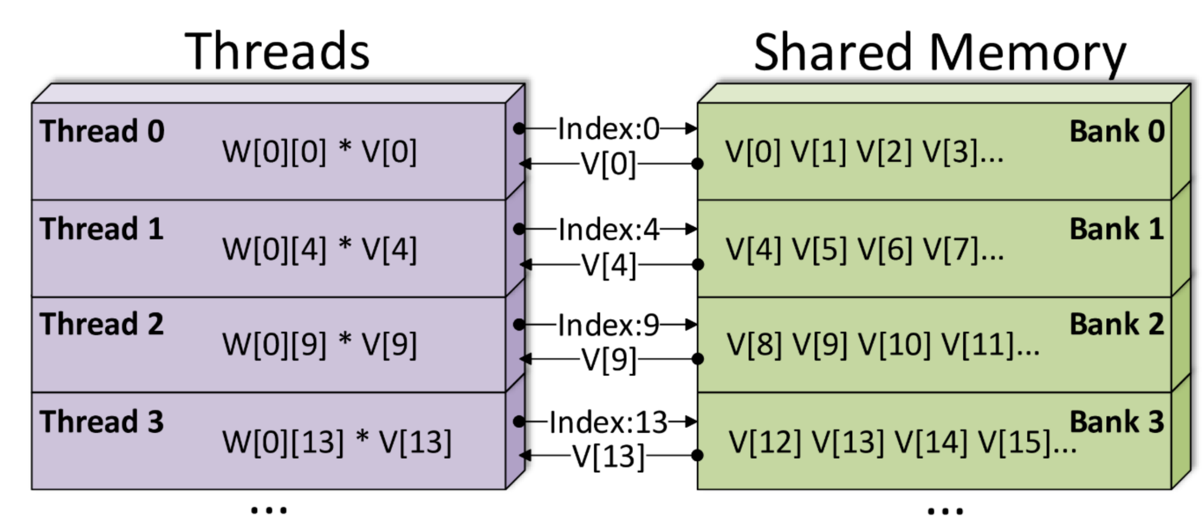
Iterative Pruning

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 $M_i \in M.rows$ **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;

Efficient GPU Implementation

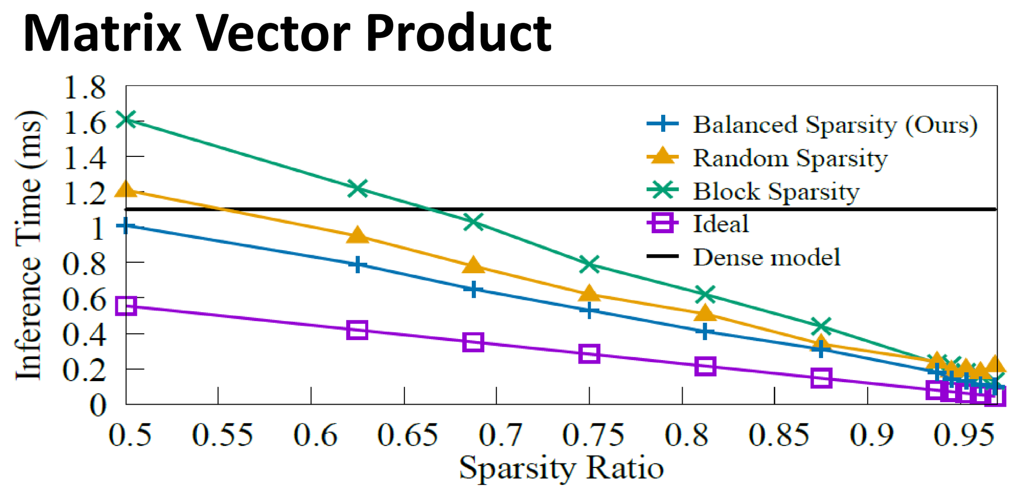


- Load balancing between threads
- Single Instruction Multiple Data (SIMD)
- Conflict-free shared memory access

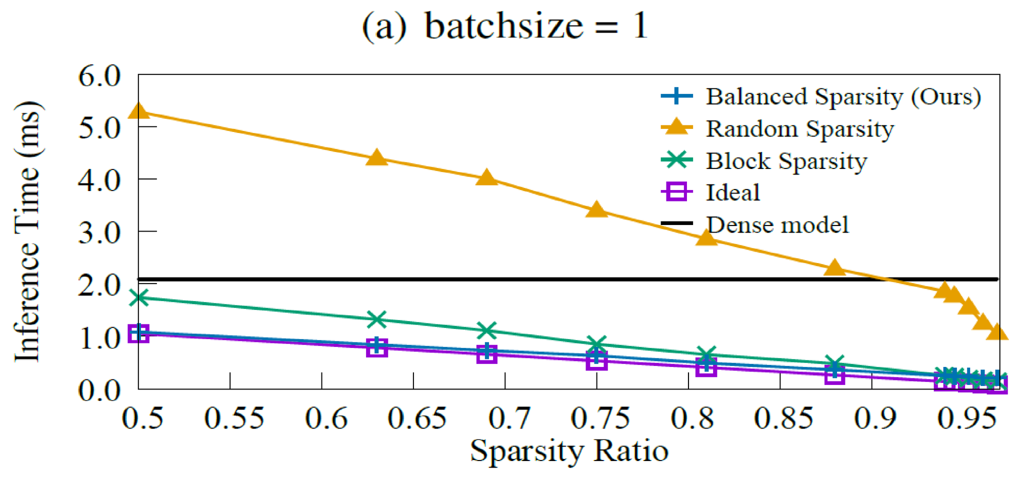
Experimental Results

Benchmark

Matrix Vector Product



(a) batchsize = 1



(b) batchsize = 8

Always faster than RS and BS
Almost reach ideal bound when batchsize = 8

Real Workloads

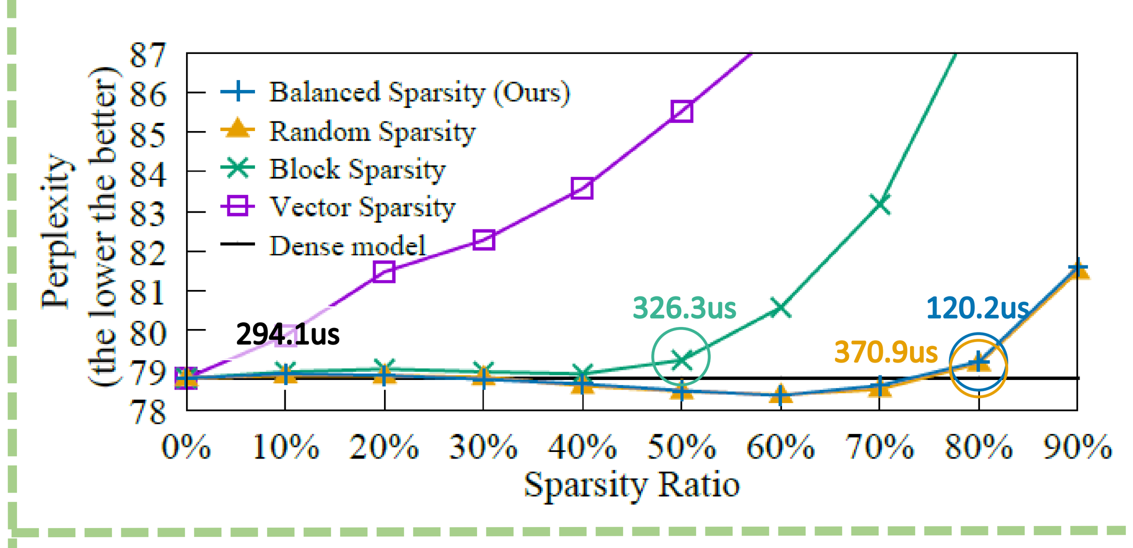
VGG on ImageNet

	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%**

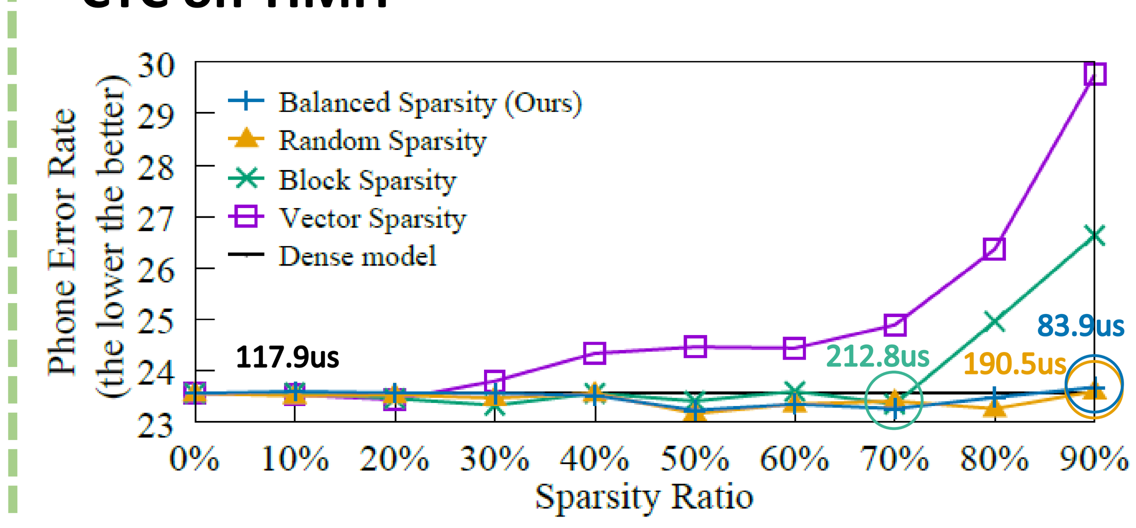
- All methods achieve top-5 accuracy of 90.3%, but under different sparsity ratio.
- Time cost of other layers (such as Pooling, Batch Normalization) is less than 230us.

12x model compression rate
6x faster inference time

LSTM on PTB

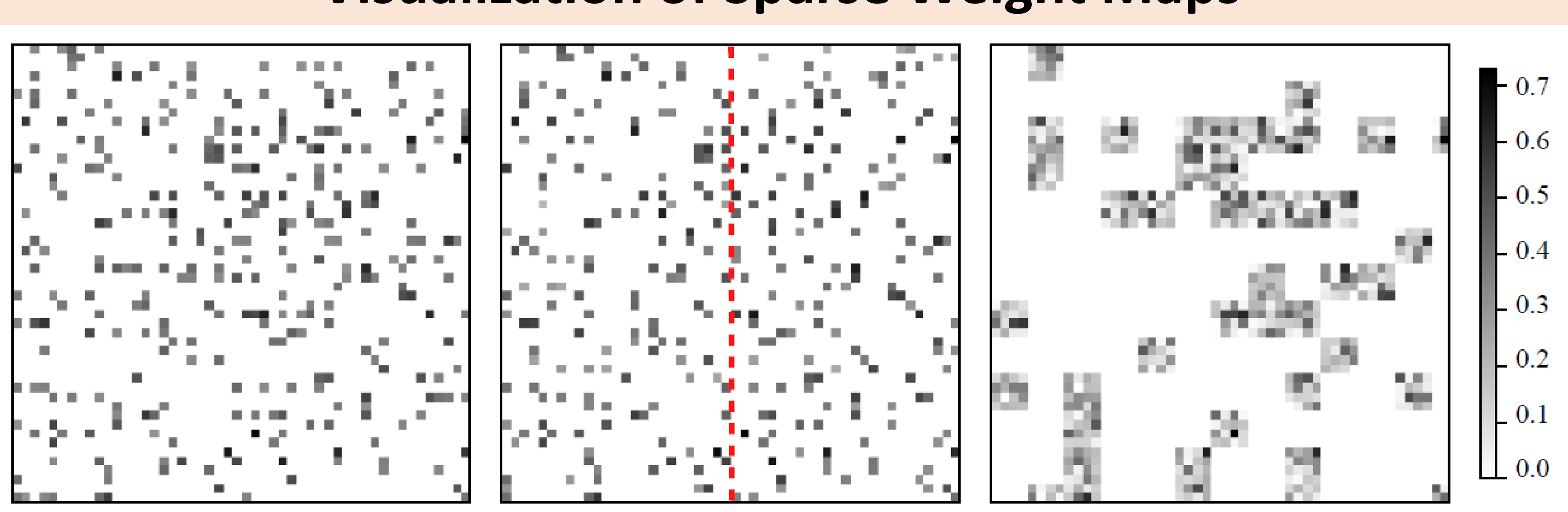


CTC on TIMIT



Further Explorations

Visualization of Sparse Weight Maps



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

Hyper Parameter 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