Deep Learning Challenges for the Internet of Things

Shuochao Yao

Outline

DeepSense: A unified deep learning framework for time-series mobile sensing data processing. (WWW 2017)

RDeepSense: Reliable Deep Mobile Computing Models with Uncertainty Estimations. (Ubicomp 2018)

DeepIoT: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework. (SenSys 2017)

Existing solutions

Weight Pruning

- Magnitude-based method
  - Iterative pruning + Retraining
  - Pruning with rehabilitation

Quantization method

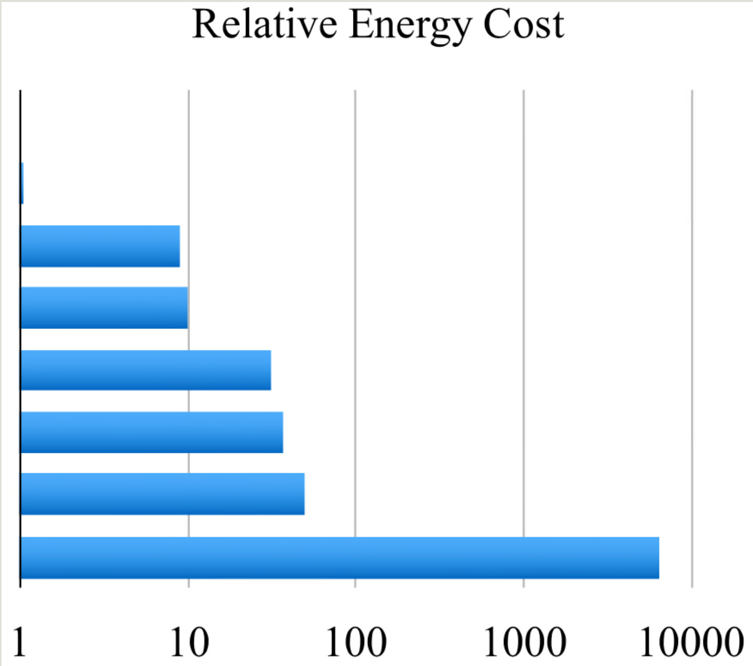
DeepIoT: Structure compression

Magnitude-based method: Iterative Pruning + Retraining

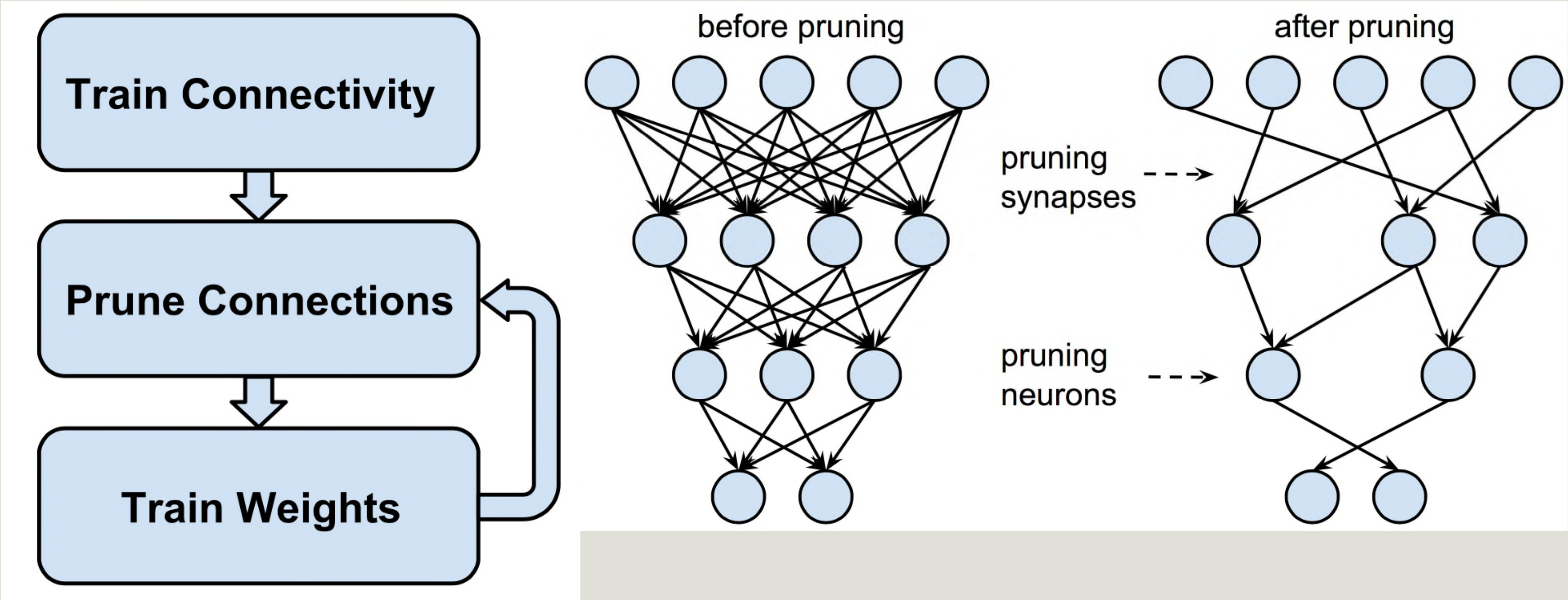
Pruning connection with small magnitude.

Iterative pruning an re-training.

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit SRAM Memory	640	6400



Magnitude-based method: Iterative Pruning + Retraining



Magnitude-based method: Iterative Pruning + Retraining (Experiment: Overall)

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	12X
LeNet-300-100 Pruned	1.59%	-	22K	
LeNet-5 Ref	0.80%	-	431K	12X
LeNet-5 Pruned	0.77%	-	36K	
AlexNet Ref	42.78%	19.73%	61M	9X
AlexNet Pruned	42.77%	19.67%	6.7M	
VGG-16 Ref	31.50%	11.32%	138M	13X
VGG-16 Pruned	31.34%	10.88%	10.3M	

Magnitude-based method: Iterative Pruning + Retraining (Experiment: Lenet)

Lenet-300-100

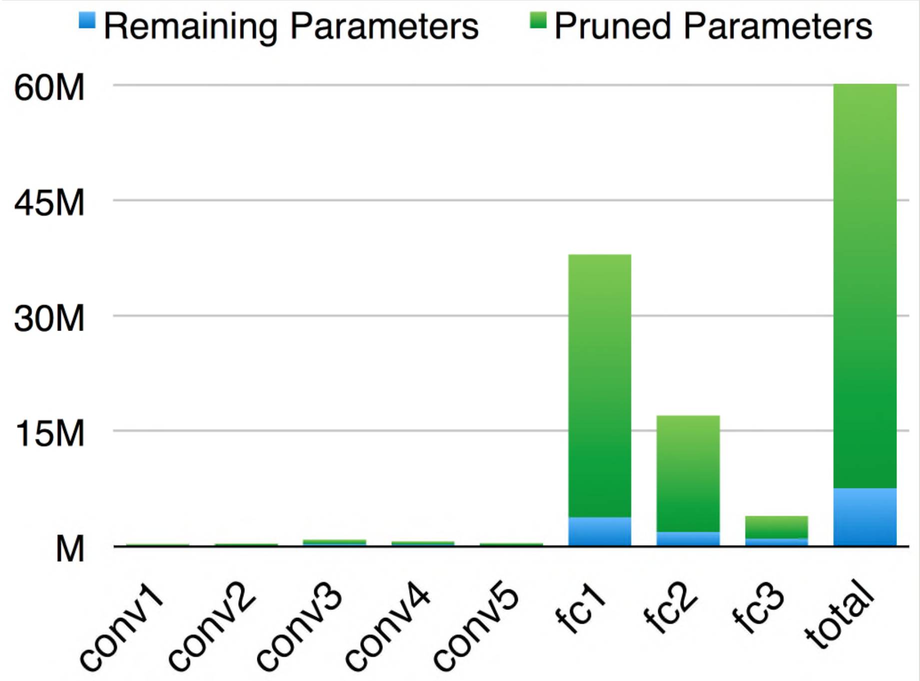
Layer	Weights	FLOP	Act%	Weights%	FLOP%
fc1	235K	470K	38%	8%	8%
fc2	30K	60K	65%	9%	4%
fc3	1K	2K	100%	26%	17%
Total	266K	532K	46%	8%	8%

Lenet-5

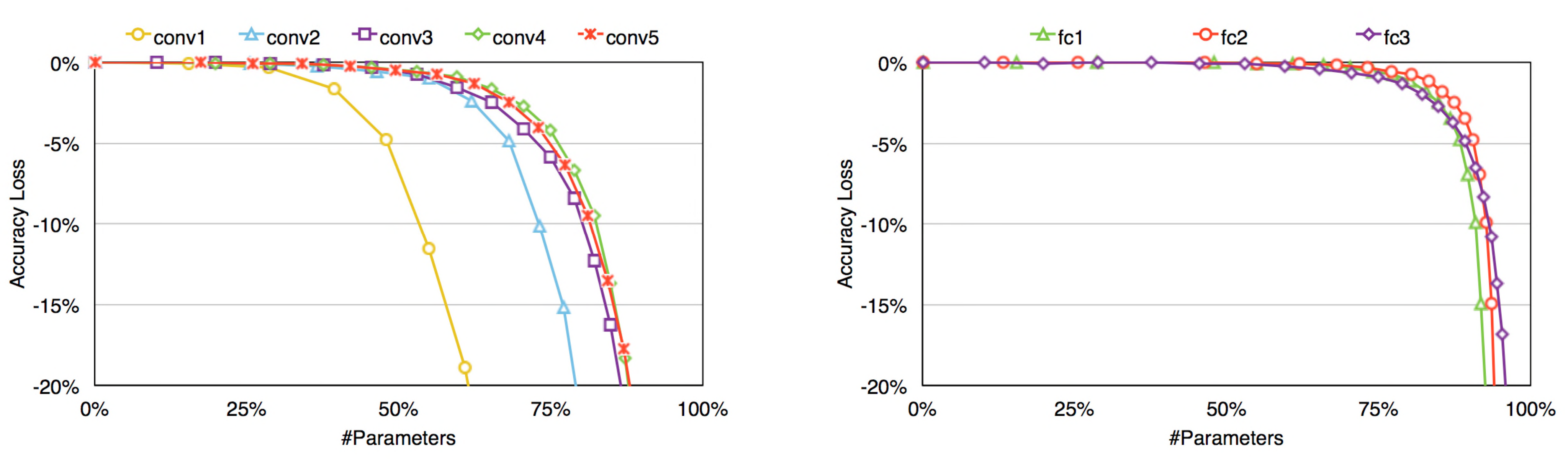
Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	0.5K	576K	82%	66%	66%
conv2	25K	3200K	72%	12%	10%
fc1	400K	800K	55%	8%	6%
fc2	5K	10K	100%	19%	10%
Total	431K	4586K	77%	8%	16%

Magnitude-based method: Iterative Pruning + Retraining (Experiment: AlexNet)

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10
Total	61M	1.5B	54%	11%	30%



Magnitude-based method: Iterative Pruning + Retraining (Experiment: Tradeoff)



Pruning with rehabilitation: Dynamic Network Surgery (Formulation)

$$\min_{W_k, T_k} L(W_k \odot T_k) \quad s.t.\ {T_k}^{(i,j)} = h_k\left({W_k}^{(i,j)}\right), \forall (i,j) \in \mathfrak{I}$$

- $\odot$ is the element-wise product. $L(\cdot)$ is the loss function.

DNS updates only W_k. T_k is updated based on $h_k(\cdot)$.

$$h_k\left({W_k}^{(i,j)}\right) = \begin{cases} 0 & a_k \geq \left|{W_k}^{(i,j)}\right| \\ {T_k}^{(i,j)} & a_k \leq \left|{W_k}^{(i,j)}\right| \leq b_k \\ 1 & b_k \leq \left|{W_k}^{(i,j)}\right| \end{cases}$$

- a_k is the pruning threshold. $b_k = a_k + t$, where t is a pre-defined small margin.

Pruning with rehabilitation: Dynamic Network Surgery (Algorithm)

1. Choose a neural network architecture.

2. Train the network until a reasonable solution is obtained.

3. Update T_k based on $h_k(\cdot)$.

4. Update W_k based on back-propagation.

5. Iterate to step 3.

Pruning with rehabilitation: Dynamic Network Surgery (Experiment on LeNet)

Model	Layer	Parameters	Parameters (DNS)
LeNet-5	conv1	0.5K	14.2%
	conv2	25K	3.1%
	fc1	400K	0.7%
	fc2	5K	4.3%
	Total	431K	0.9%
LeNet-300-100	fc1	236K	1.8%
	fc2	30K	1.8%
	fc3	1K	5.5%
	Total	267K	1.8%

Pruning with rehabilitation: Dynamic Network Surgery (Experiment on AlexNet)

Layer	Parameters	Parameters (DNS)
conv1	35K	53.8%
conv2	307K	40.6%
conv3	885K	29.0%
conv4	664K	32.3%
conv5	443K	32.5%
fc1	38M	3.7%
fc2	17M	6.6%
fc3	4M	4.6%
Total	61M	5.7%

Existing solutions

Weight Pruning

Quantization method

- Fully Quantization
  - Fixed-point format
  - Code book
- Quantization with full-precision copy

DeepIoT: Structure Compression

Fully Quantization: Fixed-point format

Limited Precision Arithmetic

- $[QI.QF]$, where QI and QF correspond to the integer and the fractional part of the number.
- The number of integer bits (IL) plus the number of fractional bits (FL) yields the total number of bits used to represent the number.
- WL = IL + FL.
- Can be represented as $\langle IL, FL \rangle$.
- $\langle IL, FL \rangle$ limits the precision to FL bits.
- $\langle IL, FL \rangle$ sets the range to $[-2^{IL-1}, 2^{IL-1} - 2^{-FL}]$.

Fully Quantization: Fixed-point format (Rounding Modes)

Define $\lfloor x \rfloor$ as the largest integer multiple of $\epsilon = 2^{-FL}$.

Round-to-nearest:

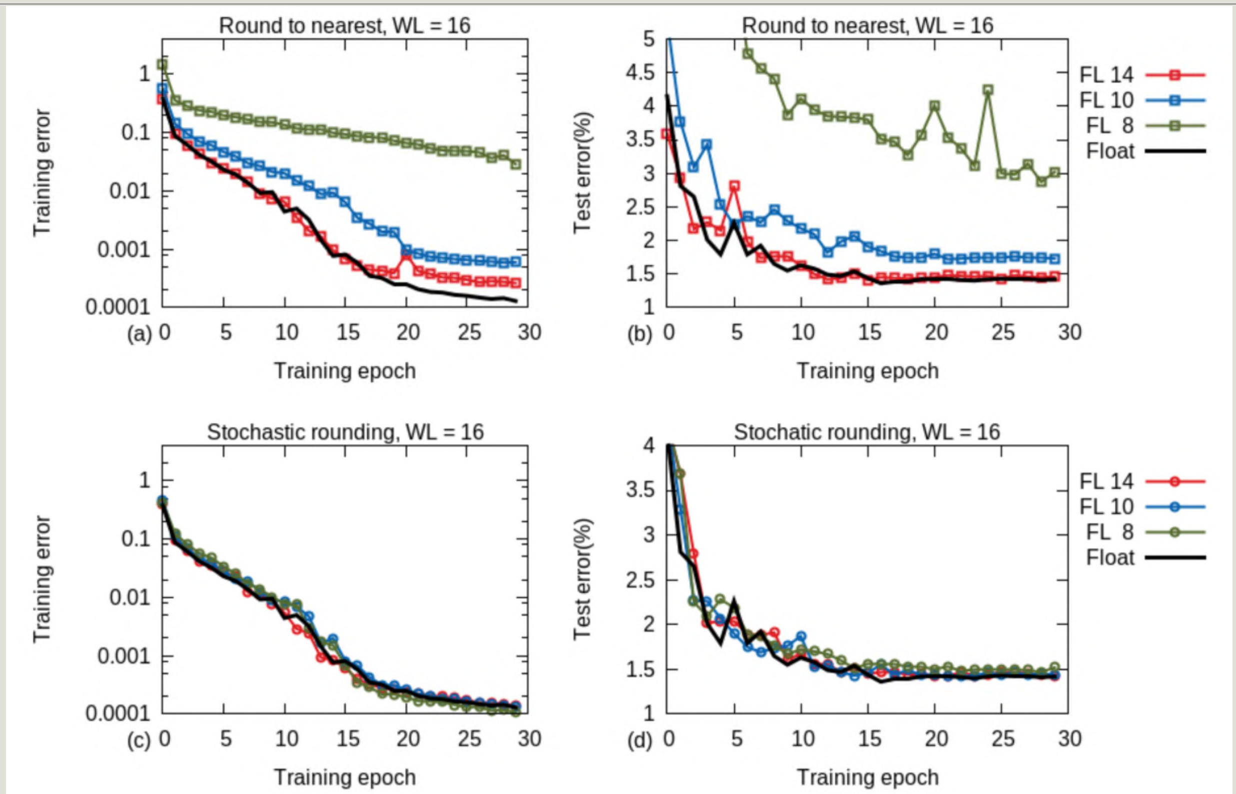
- $Round(x, \langle IL, FL \rangle) = \begin{cases} \lfloor x \rfloor & \lfloor x \rfloor \le x \le \lfloor x \rfloor + \frac{\epsilon}{2} \\ \lfloor x \rfloor + \epsilon & \lfloor x \rfloor + \frac{\epsilon}{2} \le x \le \lfloor x \rfloor + \epsilon \end{cases}$

Stochastic rounding (unbiased):

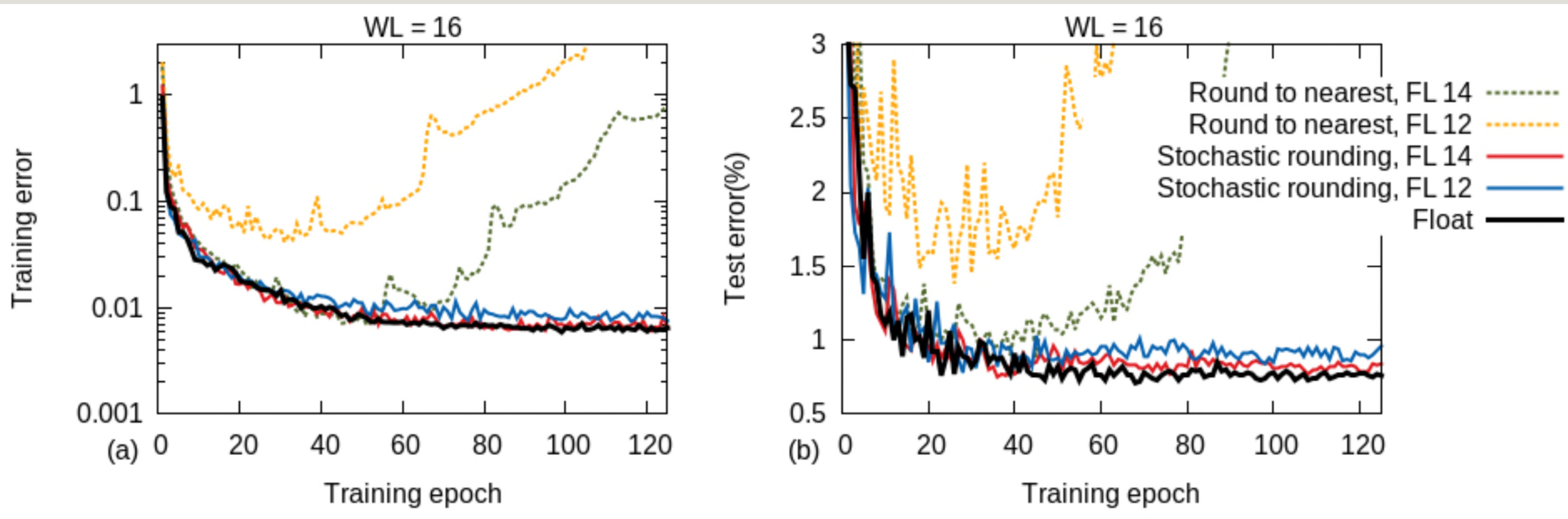
- $Round(x, \langle IL, FL \rangle) = \begin{cases} \lfloor x \rfloor & w.p. \quad 1 - \frac{x - \lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon & w.p. \quad \frac{x - \lfloor x \rfloor}{\epsilon} \end{cases}$

If x lies outside the range of $\langle IL, FL \rangle$, we saturate the result to either the lower or the upper limit of $\langle IL, FL \rangle$:

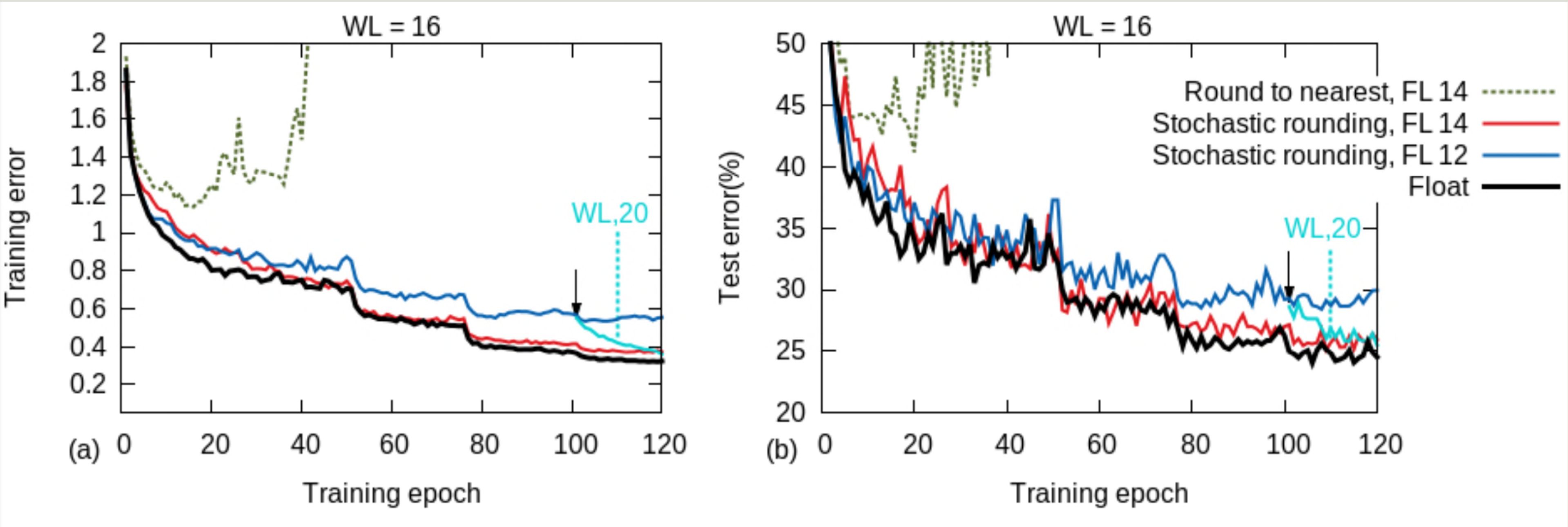
Fully Quantization: Fixed-point format (Experiment on MNIST with fully connected DNNs)



Fully Quantization: Fixed-point format (Experiment on MNIST with CNNs)



Fully Quantization: Fixed-point format (Experiment on CIFAR10 with fully connected DNNs)



Fully Quantization: Code book

Quantization using k-means

- Perform k-means to find k centers $\{c_z\}$ for weights W.
- $\widehat{W_{ij}} = c_z$ where $\min_z \left\|W_{ij} - c_z\right\|^2$.

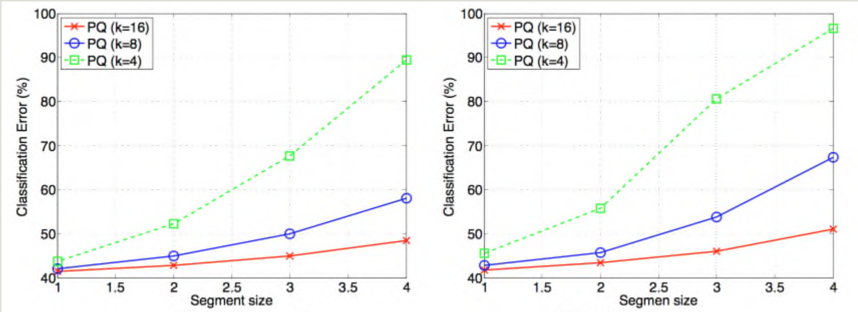
Product Quantization

- $W = [W^1, W^2, \cdots, W^s]$.
- Perform k-means for elements in W^i to find k centers $\{c_z^i\}$.
- $\widehat{W_j^i} = c_z^i$ where $\min_z \left\|W_j^i - c_z^i\right\|^2$.

Residual Quantization

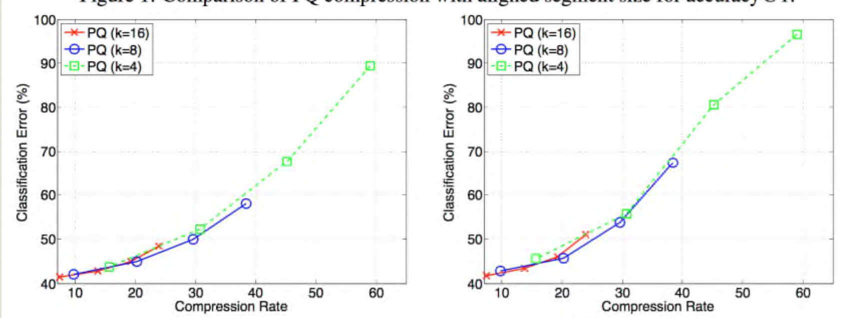
- Quantize the vectors into k centers.
- Then recursively quantize the residuals.

Fully Quantization: Code book (Experiment on PQ)



(a) PQ for X-axis (b) PQ for Y-axis

Figure 1: Comparison of PQ compression with aligned segment size for accuracy@1.



(a) PQ for X-axis (b) PQ for Y-axis

Figure 2: Comparison of PQ compression with aligned compression rate for accuracy@1. We can clearly find when taking codebook size into account, using more centers do not necessarily lead to better accuracy with same compression rate. See text for detailed discussion.

Fully Quantization: Code book

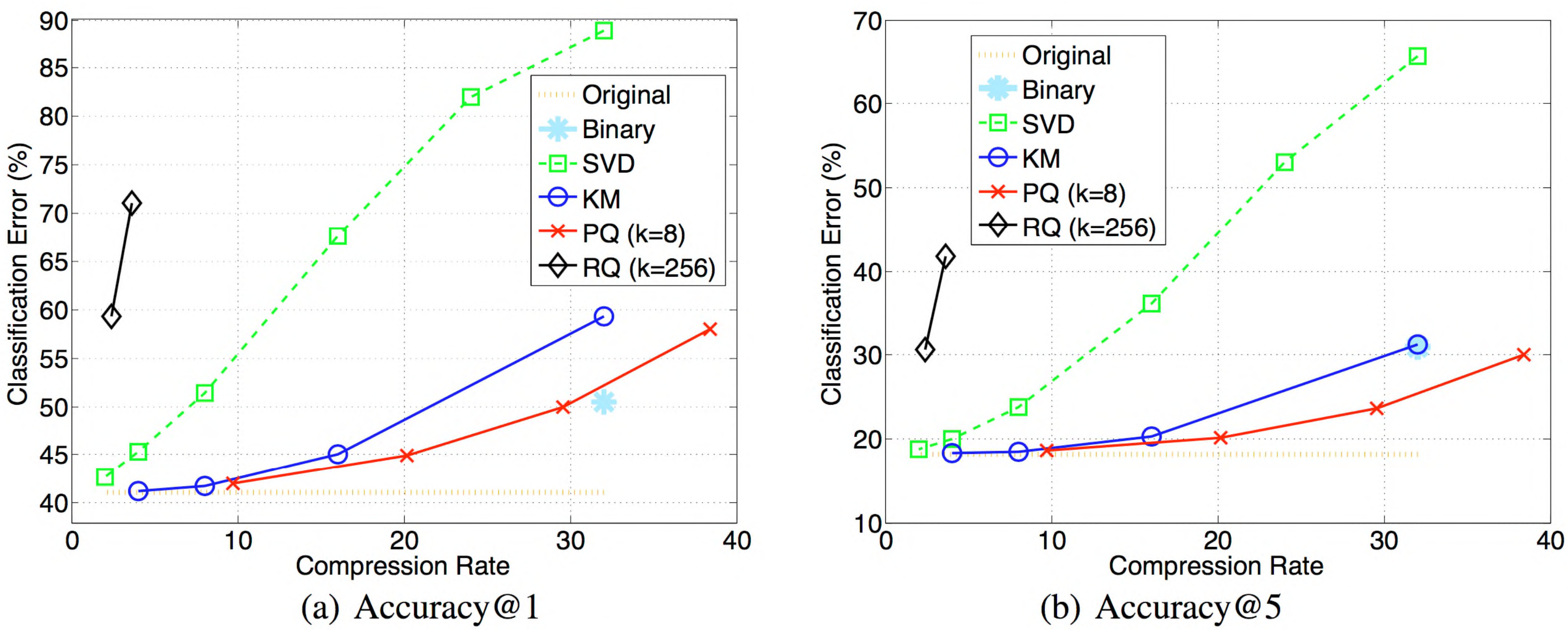


Figure 3: Comparison of different compression methods on ILSVRC dataset.

Existing solutions

Weight Pruning

Quantization method

- Fully Quantization
- Quantization with full-precision copy
  - Binnaryconnect
  - BNN

DeepIoT: Structure compression

Quantization with full-precision copy: Binaryconnect

Use only two possible value (e.g. +1 or -1) for weights.

Replace many multiply-accumulate operations by simple accumulations.

Fixed-point adders are much less expensive both in terms of area and energy than fixed-point multiply-accumulators.

Quantization with full-precision copy: Binaryconnect (Binarization)

Deterministic Binarization:

- $w_b = \begin{cases} +1 & if\ w \geq 0 \\ -1 & otherwise \end{cases}$

Stochastic Binarization:

- $w_b = \begin{cases} +1 & with\ probability\ p = \sigma(w_b) \\ -1 & with\ probability\ 1 - p \end{cases}$
- $\sigma(x) = clip\left(\frac{x+1}{2}, 0, 1\right) = max\left(0, min\left(1, \frac{x+1}{2}\right)\right)$

Stochastic binarization is more theoretically appealing than the deterministic one, but harder to implement as it requires the hardware to generate random bits when quantizing.

Quantization with full-precision copy: Binaryconnect

1. Given the DNN input, compute the unit activations layer by layer, leading to the top layer which is the output of the DNN, given its input. This step is referred as the **forward propagation**.

2. Given the DNN target, compute the training objective's gradient w.r.t. each layer's activations, starting from the top layer and going down layer by layer until the first hidden layer. This step is referred to as the **backward propagation or backward phase of back-propagation**.

3. Compute the gradient w.r.t. each layer's parameters and then update the parameters using their computed gradients and their previous values. This step is referred to as the **parameter update**.

Quantization with full-precision copy: Binaryconnect

BinaryConnect only binarize the weights during the for- ward and backward propagations (steps 1 and 2) but not during the parameter update (step 3).

Quantization with full-precision copy: Binaryconnect

Algorithm 1 SGD training with BinaryConnect. C is the cost function for minibatch and the functions binarize(w) and clip(w) specify how to binarize and clip weights. L is the number of layers.

Require: a minibatch of (inputs, targets), previous parameters w_{t-1} (weights) and b_{t-1} (biases), and learning rate η.

Ensure: updated parameters w_t and b_t.

1. Forward propagation:

$w_b \leftarrow \text{binarize}(w_{t-1})$

For $k = 1$ to L, compute a_k knowing a_{k-1}, w_b and b_{t-1}

2. Backward propagation:

Initialize output layer's activations gradient $\frac{\partial C}{\partial a_L}$

For $k = L$ to 2, compute $\frac{\partial C}{\partial a_{k-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and w_b

3. Parameter update:

Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{d b_{t-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and a_{k-1}

$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$

$b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$

Quantization with full-precision copy: Binaryconnect

Method	MNIST	CIFAR-10	SVHN
No regularizer	$1.30 \pm 0.04\%$	10.64%	2.44%
BinaryConnect (det.)	$1.29 \pm 0.08\%$	9.90%	2.30%
BinaryConnect (stoch.)	$1.18 \pm 0.04\%$	**8.27%**	2.15%
50% Dropout	$1.01 \pm 0.04\%$		
Maxout Networks [29]	0.94%	11.68%	2.47%
Deep L2-SVM [30]	**0.87%**		
Network in Network [31]		10.41%	2.35%
DropConnect [21]			1.94%
Deeply-Supervised Nets [32]		9.78%	**1.92%**

Quantization with full-precision copy: Binarized Neural Networks

Neural networks with **binary weights and activations** at run-time and when computing the parameters' gradient at train time.

Quantization with full-precision copy: Binarized Neural Networks

Propagating Gradients Through Discretization (“straight-through estimator ”)

- $q = Sign(r)$
- Estimator g_q of the gradient $\frac{\partial C}{\partial q}$
- Straight-through estimator of $\frac{\partial C}{\partial r}$:
  - $g_r = g_q 1_{|r| \leq 1}$
  - Can be viewed as propagating the gradient through *hard tanh*

Replace multiplications with bit-shift

- Replace batch normalization with shift-based batch normalization
- Replace ADAM with shift-based AdaMax

Quantization with full-precision copy: Binarized Neural Networks

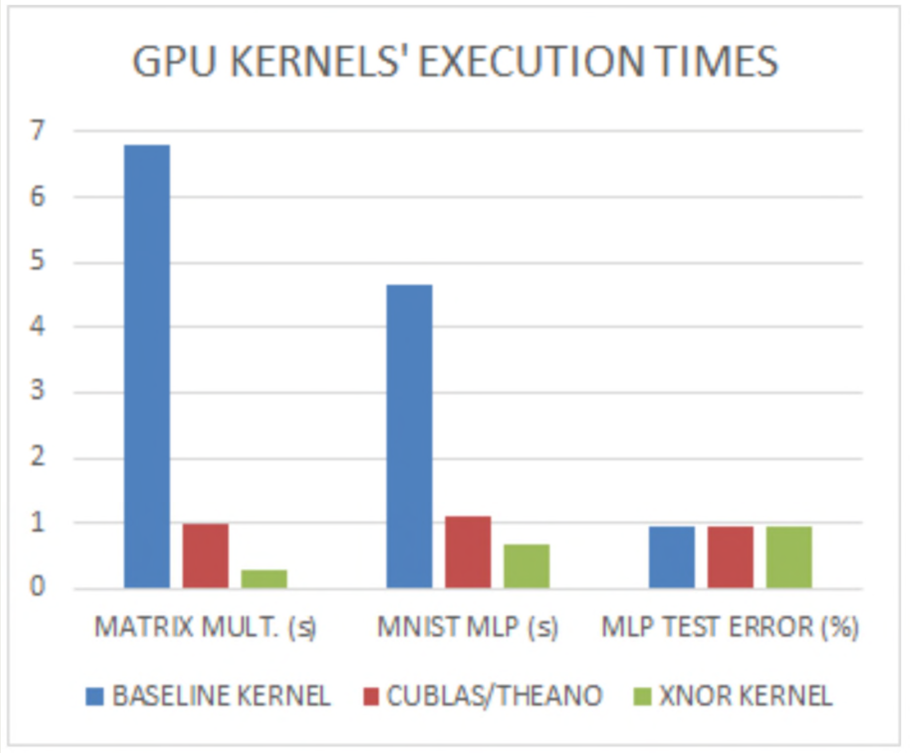
Data set	MNIST	SVHN	CIFAR-10
Binarized activations+weights, during training and test			
BNN (Torch7)	1.40%	2.53%	10.15%
BNN (Theano)	0.96%	2.80%	11.40%
Committee Machines' Array (Baldassi et al., 2015)	1.35%	-	-
Binarized weights, during training and test			
BinaryConnect (Courbariaux et al., 2015)	1.29± 0.08%	2.30%	9.90%
Binarized activations+weights, during test			
EBP (Cheng et al., 2015)	2.2± 0.1%	-	-
Bitwise DNNs (Kim & Smaragdis, 2016)	1.33%	-	-
Ternary weights, binary activations, during test			
(Hwang & Sung, 2014)	1.45%	-	-
No binarization (standard results)			
Maxout Networks (Goodfellow et al.)	0.94%	2.47%	11.68%
Network in Network (Lin et al.)	-	2.35%	10.41%
Gated pooling (Lee et al., 2015)	-	1.69%	7.62%

Quantization with full-precision copy: Binarized Neural Networks

Operation	MUL	ADD
8bit Integer	0.2pJ	0.03pJ
32bit Integer	3.1pJ	0.1pJ
16bit Floating Point	1.1pJ	0.4pJ
32bit Floating Point	3.7pJ	0.9pJ

Memory size	64-bit memory access
8k	10pJ
32k	20pJ
1M	100pJ
DRAM	1.3-2.6nJ

Quantization with full-precision copy: Binarized Neural Networks



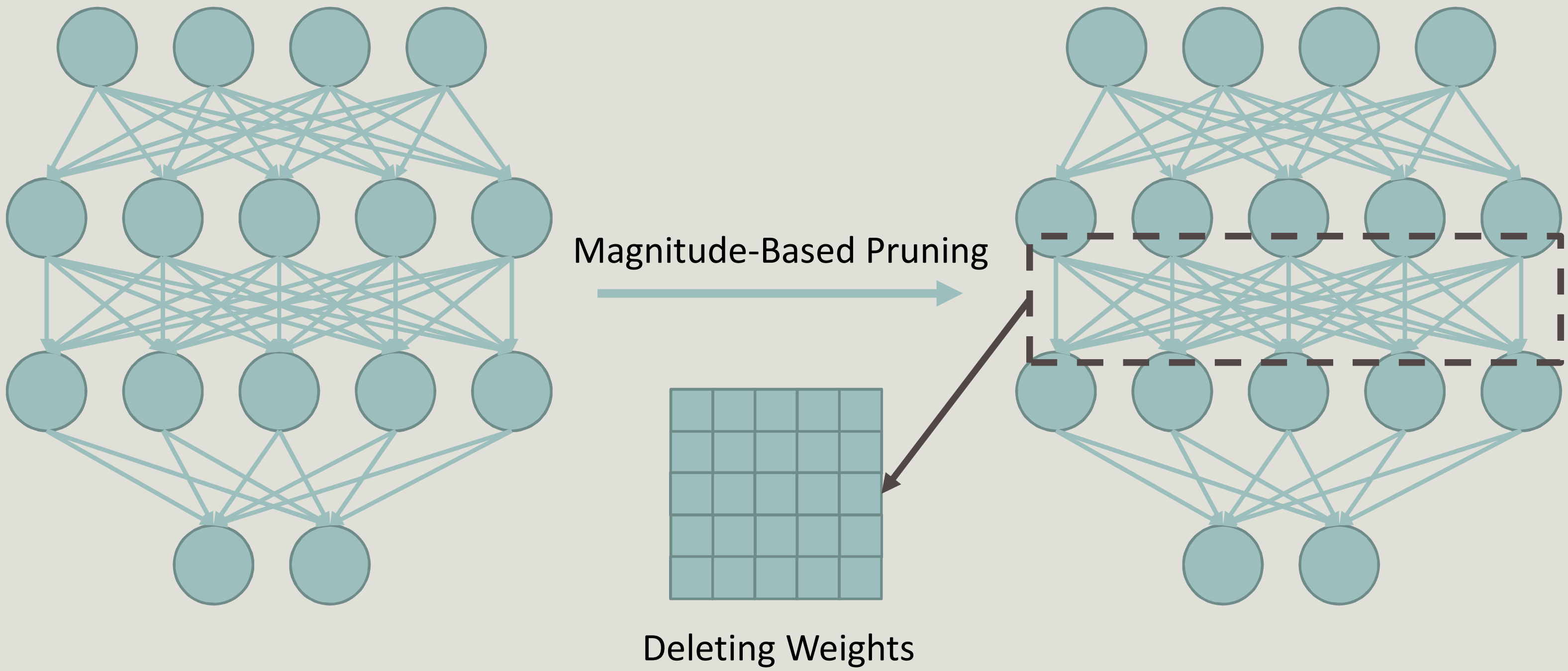
Existing solutions

Weight Pruning

Quantization method

DeepIoT: Structure Compression

Previous Solutions



Problem: Inefficiency in Spare Matrix

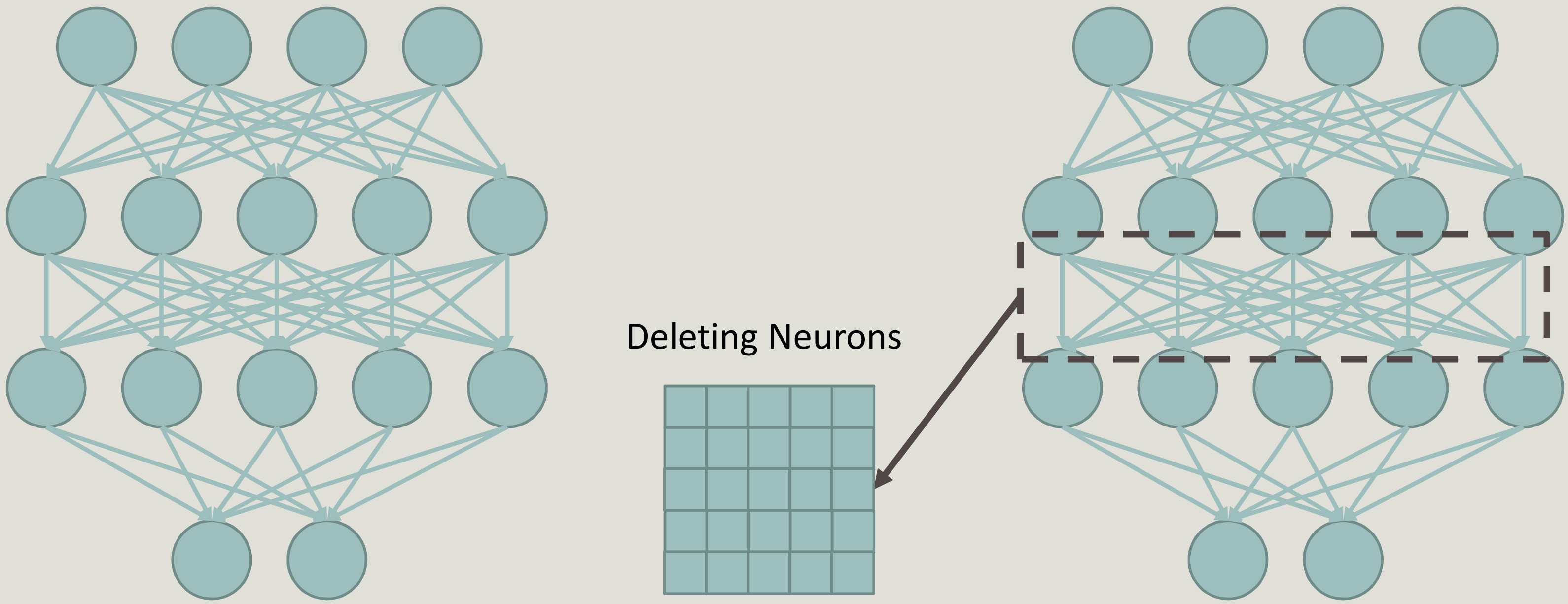
1. Need to record both indices and values for non-zero elements (≥✕3 memory consumption).

2. The multiplication between a matrix (m✕k) with ***1%*** of nonzero elements and a vector (k✕1)

m	k	Time(Sparse_matmul/Dense_matmul)
100	100	51.7%
100	1000	29.1%
1000	100	33.6%
1000	1000	11.7%

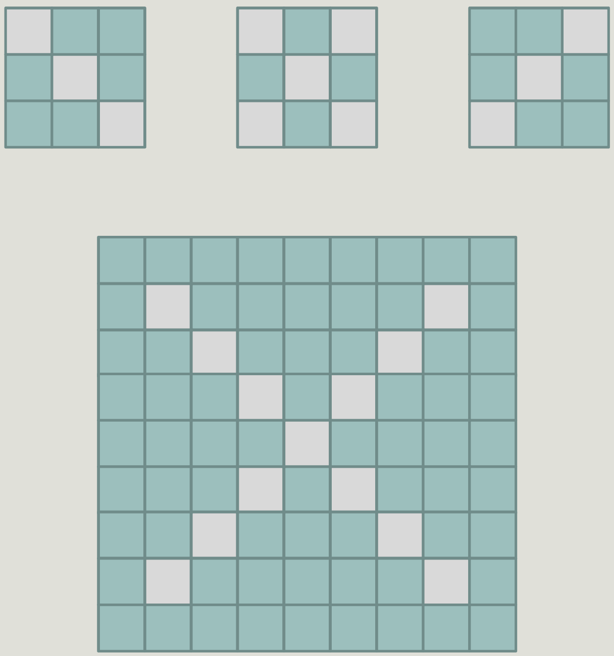
How to efficiently convert "theoretical" reduction in the number of parameters into "practical" system improvements?

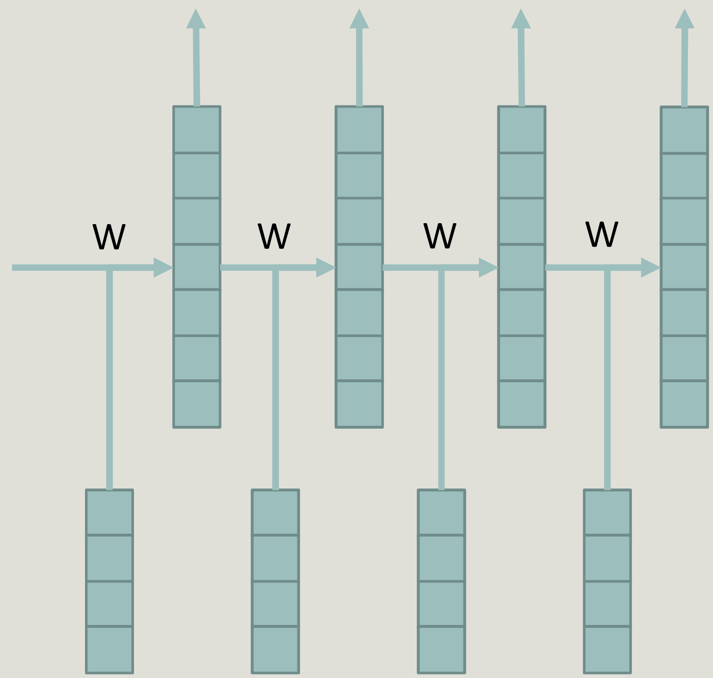
DeepIoT: Intuition



Deciding the optimal number of elements in each layer (structure).

DeepIoT: CNNs and RNNs





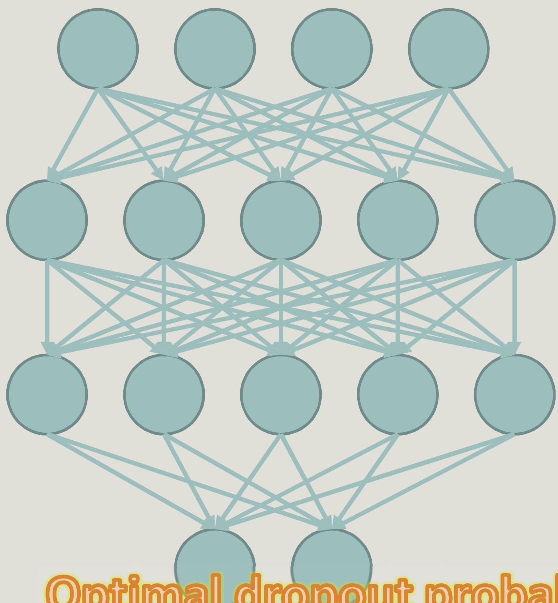
DeepIoT: The Properties We Prefer

1. Can recover the previous pruned elements.

2. Not prune elements just based on magnitudes but on parameter redundancies.

3. Have a global view of parameter redundancies.

DeepIoT: Ability to Recover



Optimal dropout probabilities for each element

~~Dropout probability: 0.5~~

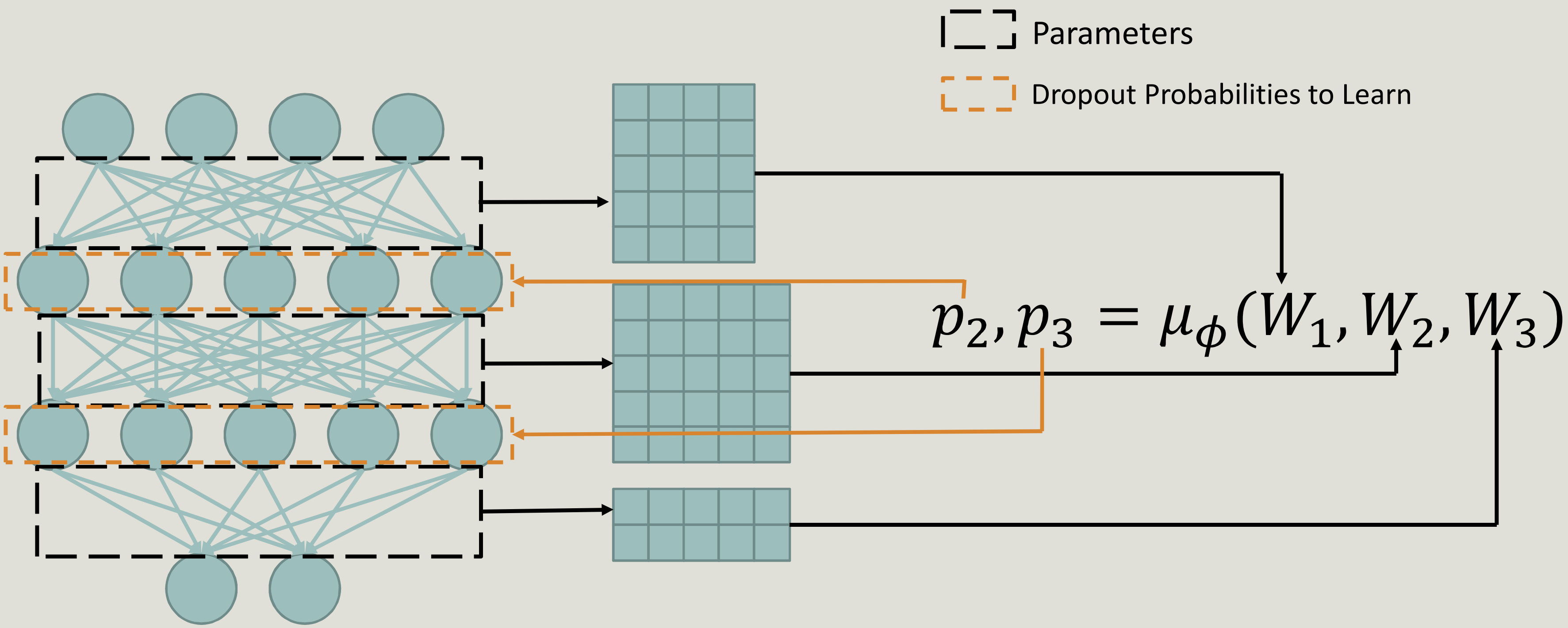
Dropout

Stochastically prune hidden elements based on pre-defined dropout probability to generate a “thinned” structure during training.

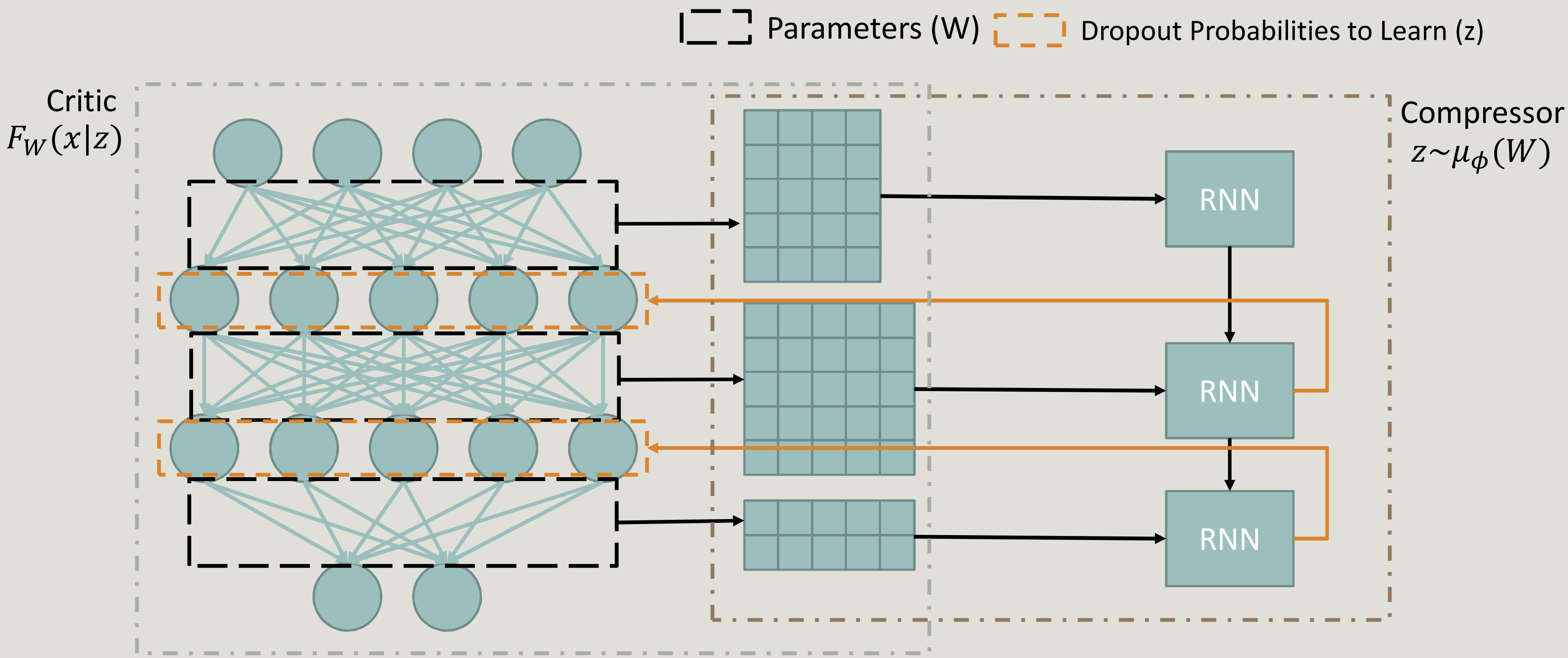
If we have the “optimal” dropout probabilities for each element, we can obtain the “optimal” slim structure for compression.

The stochastical pruning during the compression process provides DeepIoT the ability to “recover” based on the learnt dropout probability.

DeepIoT: Learning Parameter Redundancies



DeepIoT: Global Views of Parameter Redundancies



DeepIoT: Compressor-Critic Framework

$$\mathcal{L} = \sum_{z\sim\{0,1\}^{|z|}} \mu_\phi(W) \times l(y, F_W(x|z))$$

Iteratively train Compressor and Critic neural networks

$$\nabla_W \mathcal{L} = \sum_z \mu_\phi(W) \times \nabla_W l(y, F_W(x|z))$$
$$= \mathbb{E}_{z\sim\mu_\phi}[\nabla_W l(y, F_W(x|z))]$$
$$\widehat{\nabla_W \mathcal{L}} = \nabla_W l(y, F_W(x|z)) \quad z\sim\mu_\phi(W)$$

$$\nabla_\phi \mathcal{L} = \sum_z \nabla_\phi \mu_\phi(W) \times l(y, F_W(x|z))$$
$$= \sum_z \mu_\phi(W) \nabla_\phi log\big(\mu_\phi(W)\big) \times l(y, F_W(x|z))$$
$$= \mathbb{E}_{z\sim\mu_\phi}\big[\nabla_\phi log\big(\mu_\phi(W)\big) \times l(y, F_W(x|z))\big]$$
$$\widehat{\nabla_\phi \mathcal{L}} = \nabla_\phi log\big(\mu_\phi(W)\big) \times l(y, F_W(x|z)) \quad z\sim\mu_\phi(W)$$

DeepIoT: Evaluation

Intel Edison Platform.

Run solely on CPU.

No additional runtime optimization.

All models use 32-bit floats without any quantization.

DeepIoT: Performance Overview

Original/Compressed/Compression Ratio

Model	Size (MB)	Time (ms)	Energy (mJ)
LeNet5	1.72/0.04/97.6%	50.2/14.2/71.4%	47.1/12.5/73.5%
VGGNet	118.8/2.9/97.6%	1.5K/82.2/94.5%	1.7K/74/95.6%
Bi-LSTM	76.0/7.59/90.0%	71K/9.6K/86.5%	62.9K/8.1K/87.1%
DeepSense-HHAR	1.89/0.12/93.7%	130/36.7/71.8%	99.6/27.7/72.2%
DeepSense-UserID	1.89/0.02/98.9%	130/25.1/80.7%	105.1/18.1/82.8%

DeepIoT: Baseline Algorithms

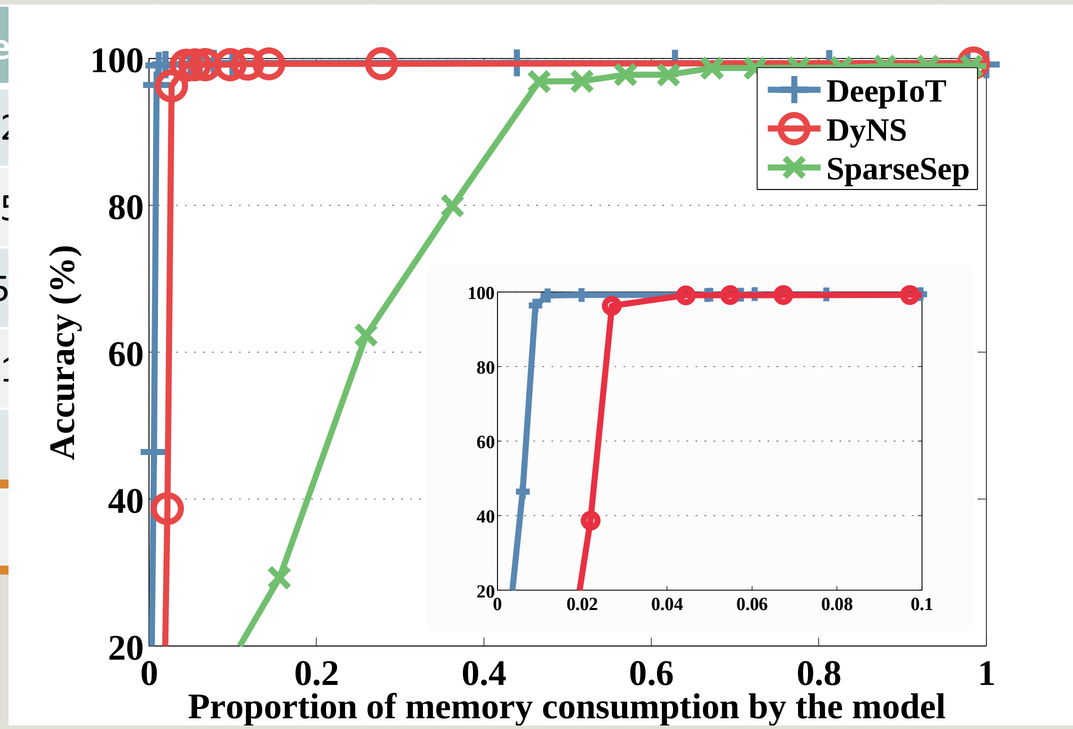
1. DyNS：
   - A magnitude-based network pruning algorithm.
   - It retrains the network connections after each pruning step and has the ability to recover the pruned weights.

2. SparseSep:
   - A sparse-coding and factorization based algorithm.
   - Simplifies the fully-connected layer by finding the optimal code-book and code based on a sparse coding technique.
   - Simplifies convolutional layer by matrix factorization

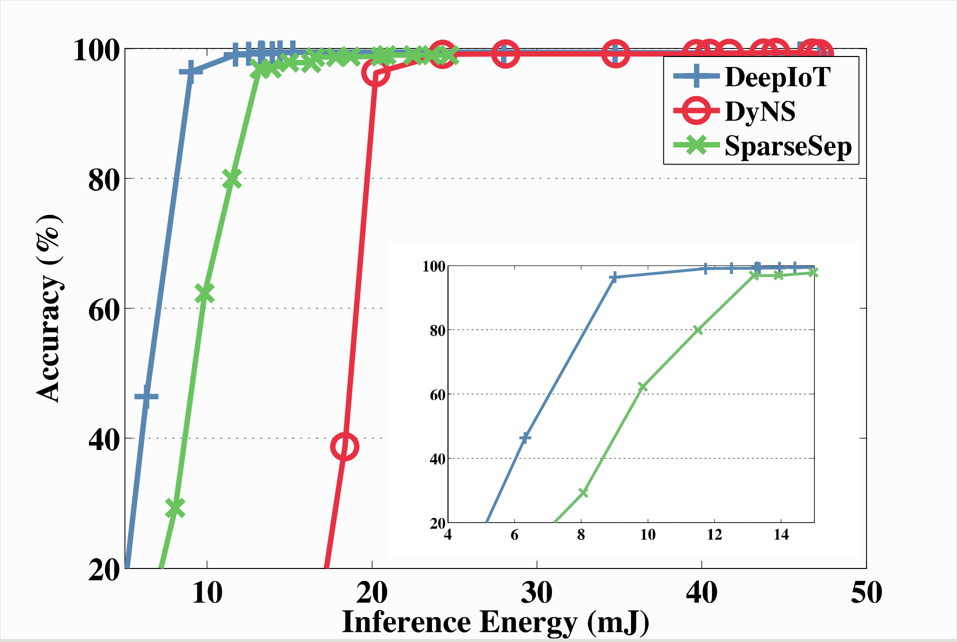
3. DyNS-Ext:
   - Enhance and extend the magnitude-based method used in DyNS to recurrent layers.

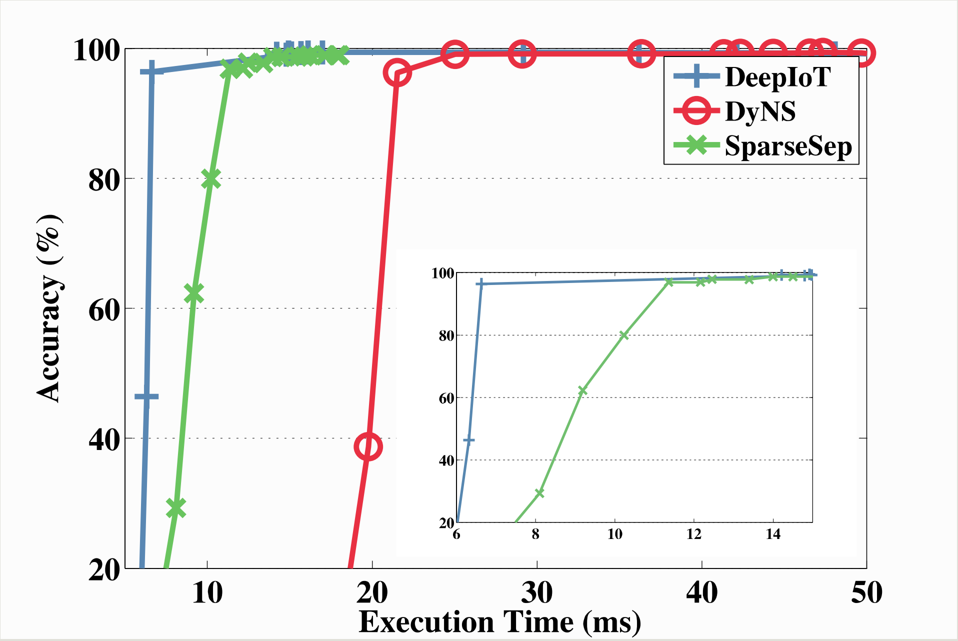
DeepIoT: Handwritten digits recognition with LeNet5

Layer	Hidde	NS	SparseSep
conv1 (5 × 5)	2	2%	84%
conv2 (5 × 5)	5	7%	91%
fc1	5	%	78.75%
fc2	1	4%	70.28%
total		5%	72.39%
Test Error		5%	1.05%

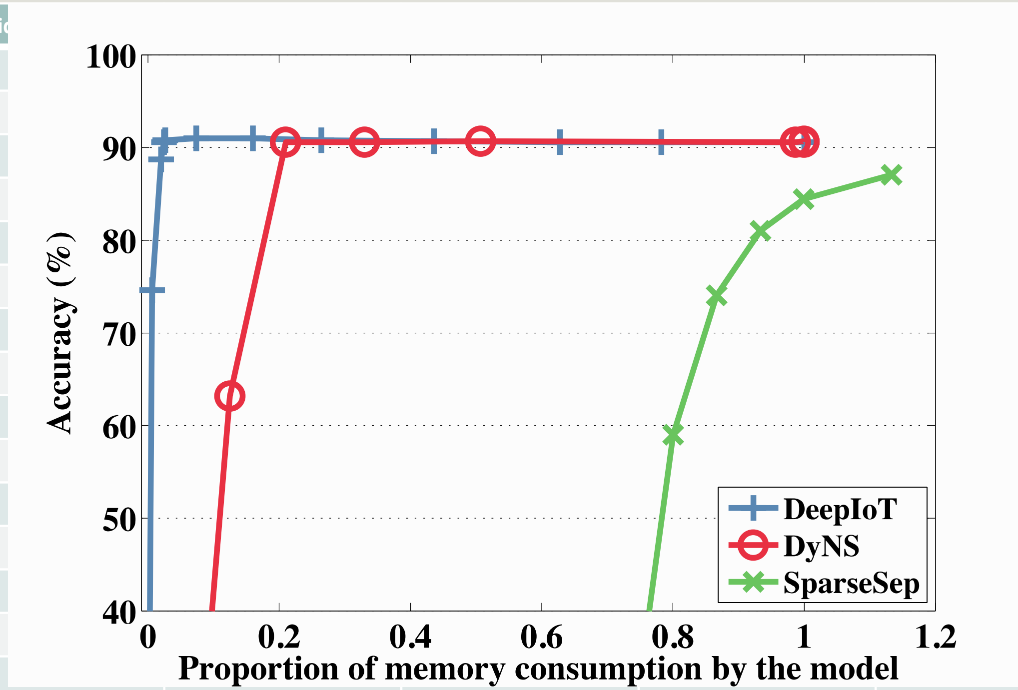


DeepIoT: Handwritten digits recognition with LeNet5



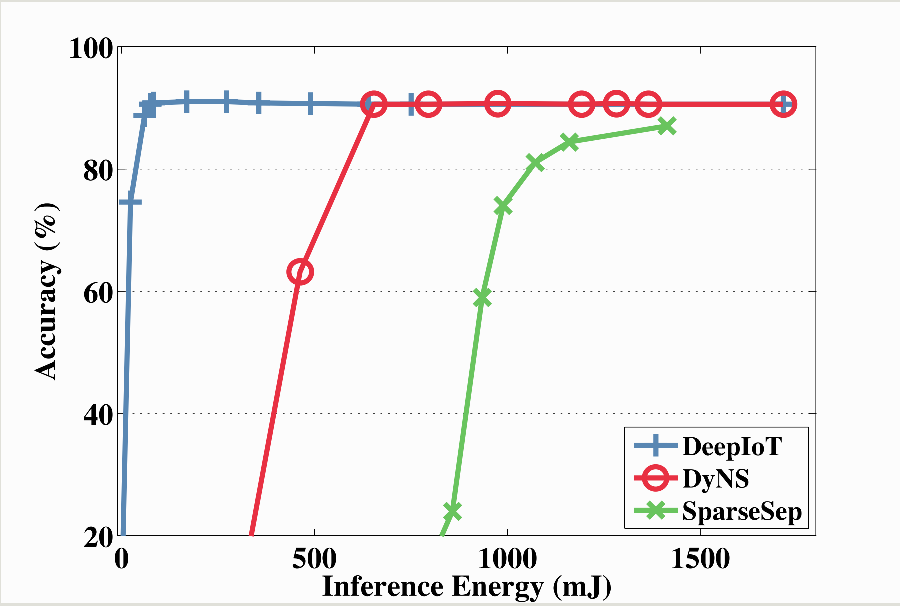


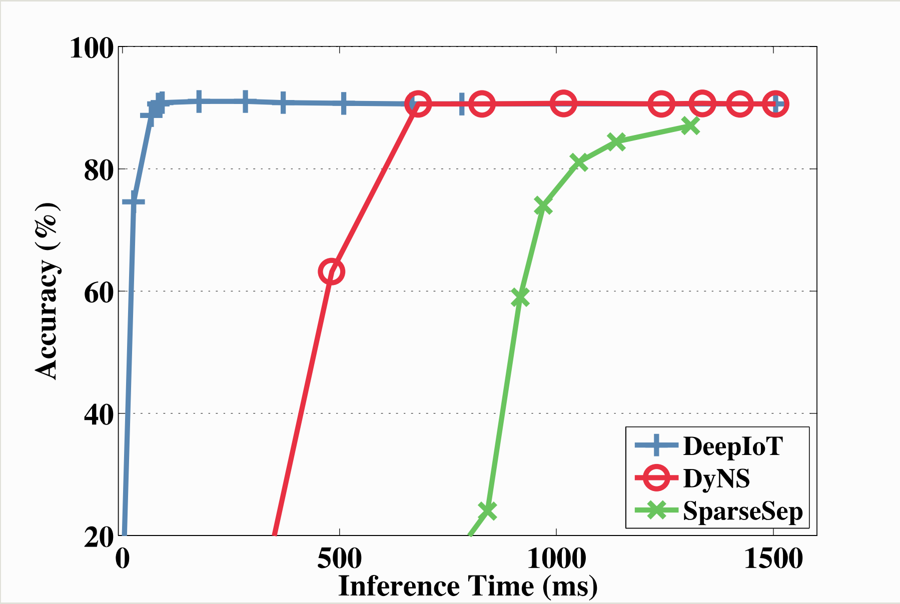
DeepIoT: Image recognition with VGGNet



Layer	Hi					SparseSep
conv1 (3 × 3)						93.1%
conv2 (3 × 3)						57.3%
conv3 (3 × 3)						85.1%
conv4 (3 × 3)						56.8%
conv5 (3 × 3)						85.1%
conv6 (3 × 3)						56.8%
conv7 (3 × 3)						56.8%
conv8 (3 × 3)						85.2%
conv9 (3 × 3)						56.8%
conv10 (3 × 3)						56.8%
conv11 (2 × 2)						85.2%
conv12 (2 × 2)						85.2%
conv13 (2 × 2)						85.2%
fc1						95.8%
fc2						95.8%
fc3	10	41K	10	9.1%	18.5%	90.2%
total		29.7M		2.44%	7.05%	112%
Test Accuracy	90.06%		90.06%		90.06%	87.1%

DeepIoT: Image recognition with VGGNet





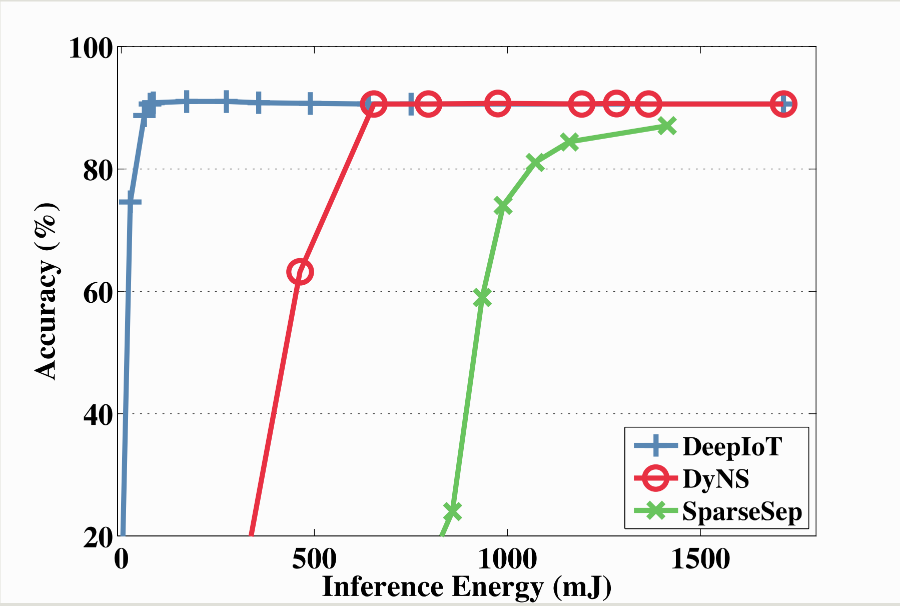
DeepIoT: Image recognition with VGGNet

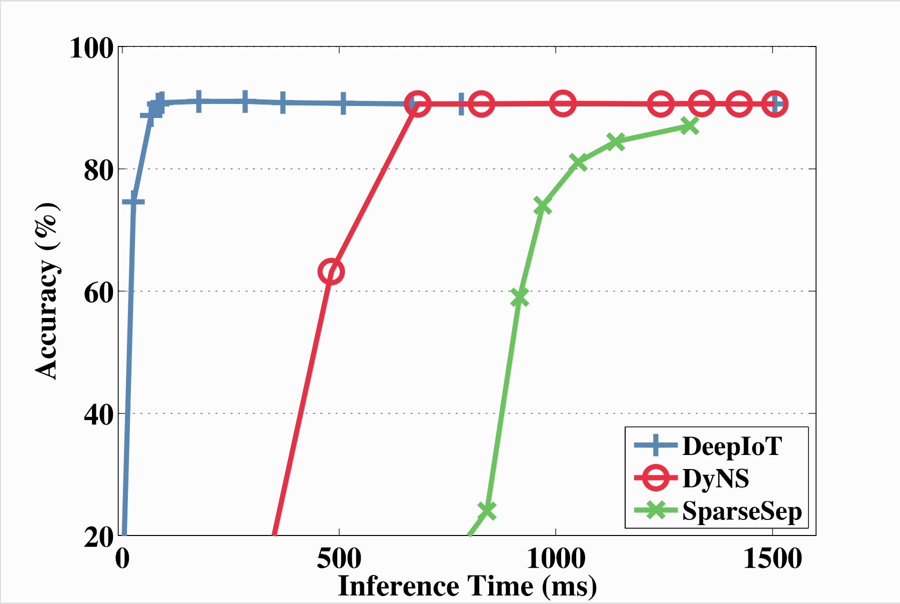
Layer	Hidden Units	Params	DeepIoT (Hidden Units/ Params)		DyNS	SparseSep
conv1 (3 × 3)	64	1.7K	27	42.2%	53.9%	93.1%
conv2 (3 × 3)	64	36.9K	47	31.0%	40.1%	57.3%
conv3 (3 × 3)	128	73.7K	53	30.4%	52.3%	85.1%
conv4 (3 × 3)	128	147.5K	68	22.0%	67.0%	56.8%
conv5 (3 × 3)	256	294.9K	104	21.6%	71.2%	85.1%
conv6 (3 × 3)	256	589.8K	97	15.4%	65.0%	56.8%
conv7 (3 × 3)	256	589.8K	89	13.2%	61.2%	56.8%
conv8 (3 × 3)	512	1.179M	122	8.3%	36.5%	85.2%
conv9 (3 × 3)	512	2.359M	95	4.4%	10.6%	56.8%
conv10 (3 × 3)	512	2.359M	64	2.3%	3.9%	56.8%
conv11 (2 × 2)	512	1.049M	128	3.1%	3.0%	85.2%
conv12 (2 × 2)	512	1.049M	112	5.5%	1.7%	85.2%
conv13 (2 × 2)	512	1.049M	149	6.4%	2.4%	85.2%
fc1	4096	2.097M	27	0.19%	2.2%	95.8%
fc2	4096	16.777M	371	0.06%	0.39%	95.8%
fc3	10	41K	10	9.1%	18.5%	90.2%
total		29.7M		2.44%	7.05%	112%acc_mem.eps
Test Accuracy	90.06%		90.06%		90.06%	87.1%

DeepIoT: Handwritten digits recognition with LeNet5

Layer	Hidden Units	Params	DeepIoT (Hidden Units/ Params)		DyNS	SparseSep
conv1 (5 × 5)	20	0.5k	10	50%	24.2%	84%
conv2 (5 × 5)	50	25k	20	20%	20.7%	91%
fc1	500	400k	10	0.8%	1.0%	78.75%
fc2	10	5k	10	2.0%	16.34%	70.28%
total		431k		1.98%	2.35%	72.39%
Test Error	0.85%		0.85%		0.85%	1.05%

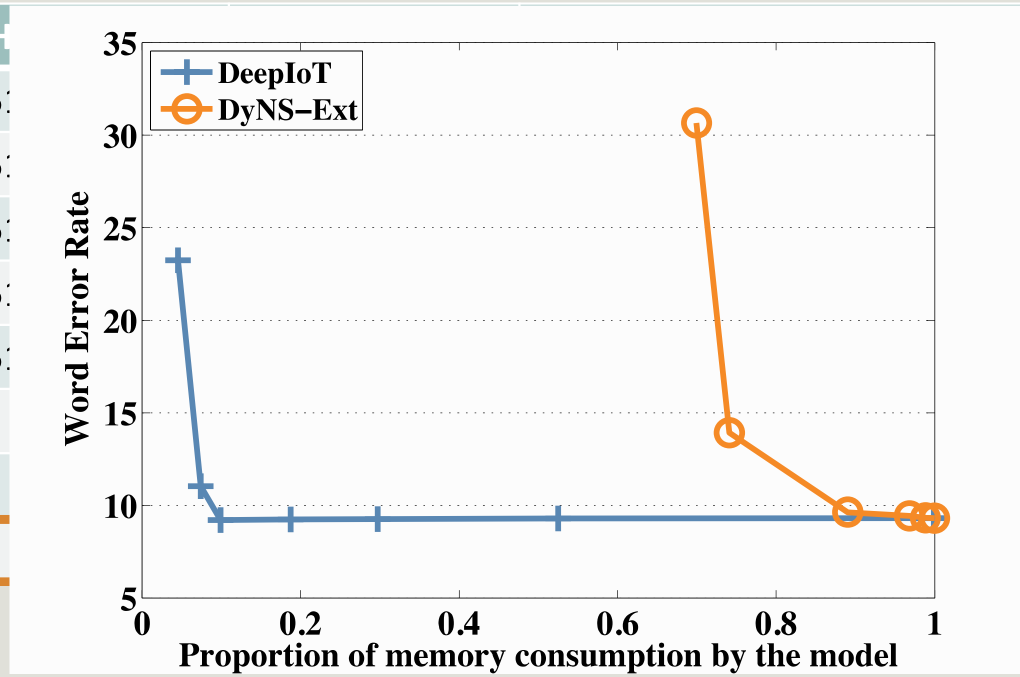
DeepIoT: Image recognition with VGGNet



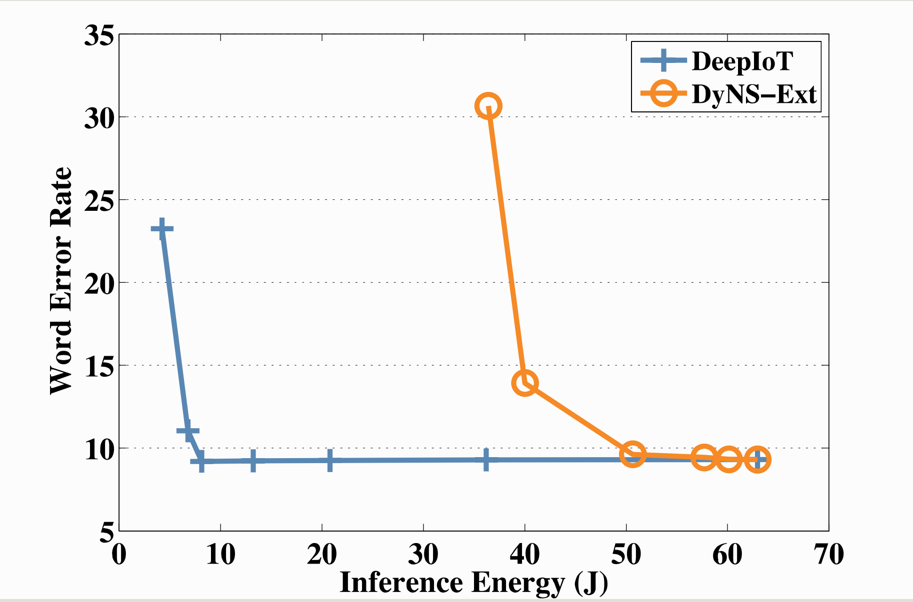


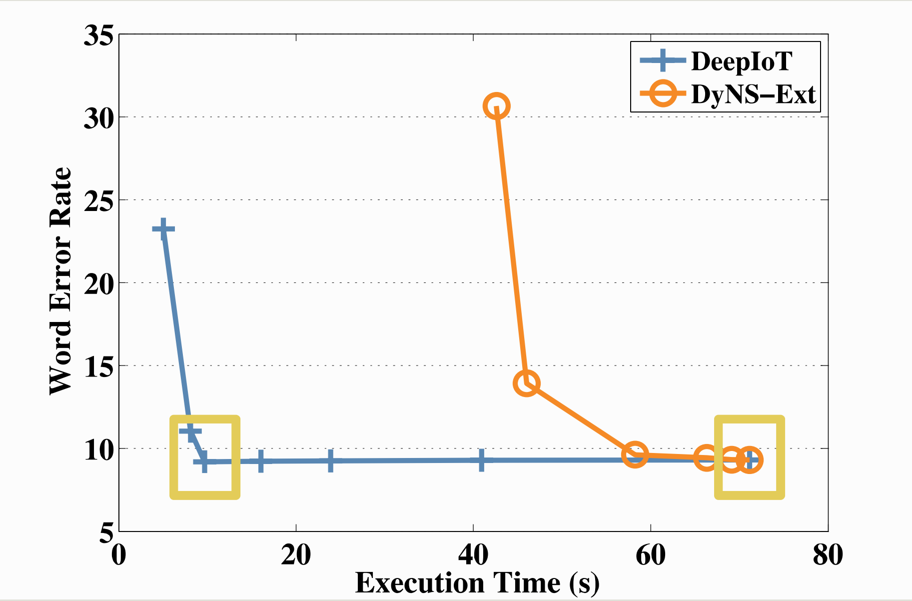
DeepIoT: Speech recognition with deep Bidirectional LSTM

Layer		H	ns)	DyNS-Ext	
LSTMf1	LSTMb1	5	1%	34.9%	18.2%
LSTMf2	LSTMb2	5	4%	37.2%	23.1%
LSTMf3	LSTMb3	5	6%	43.1%	27.9%
LSTMf4	LSTMb4	5	5%	52.3%	40.2%
LSTMf5	LSTMb5	5	8%	72.6%	61.8%
fc1				69.0%	
total				37.1%	
Word error rate				9.62	



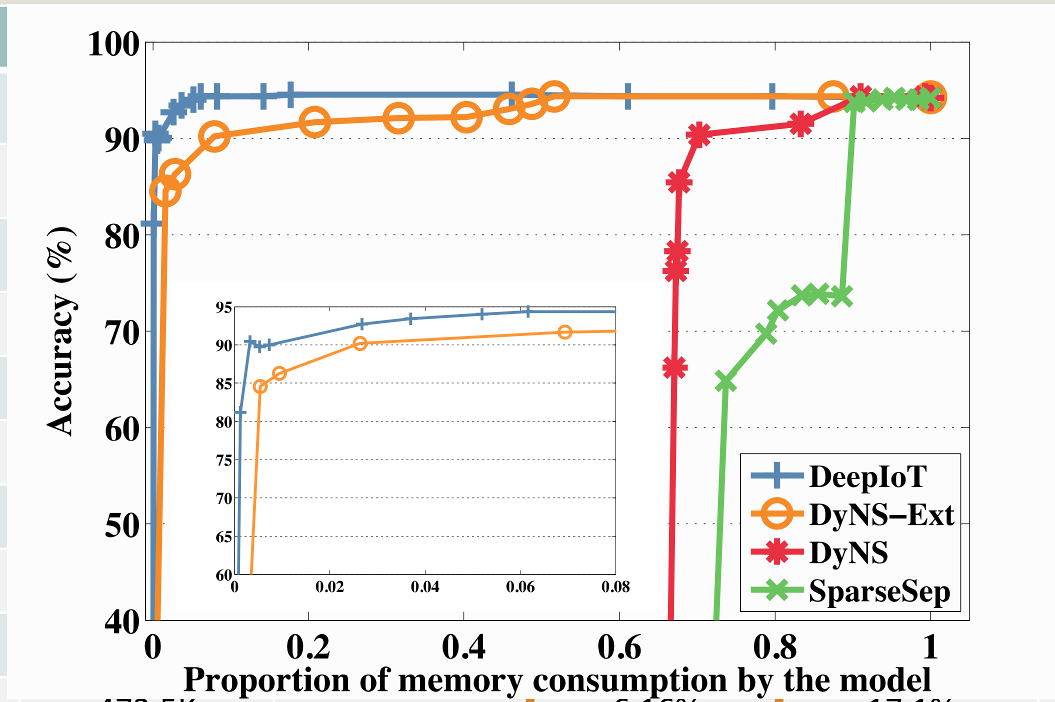
DeepIoT: Speech recognition with deep Bidirectional LSTM





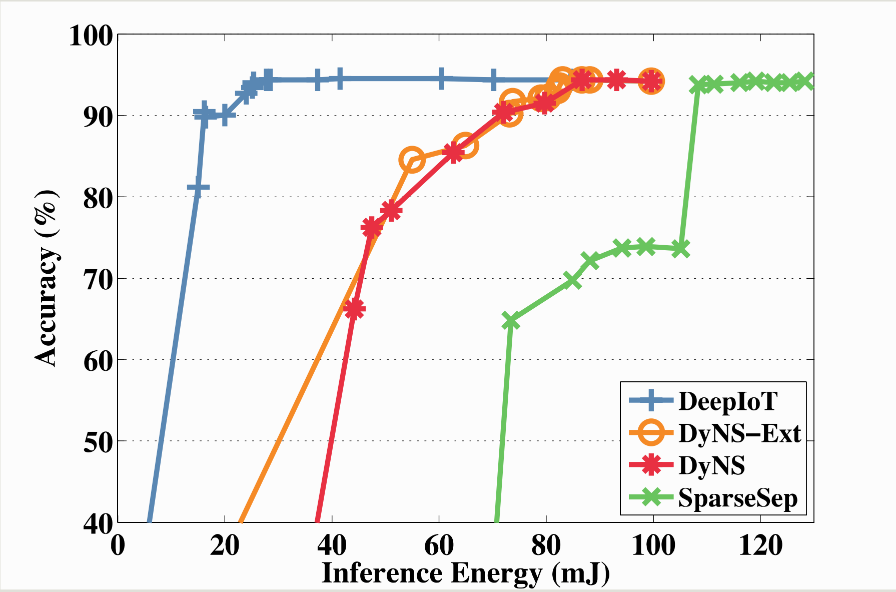
DeepIoT: Heterogeneous Human Activity Recognition with DeepSense

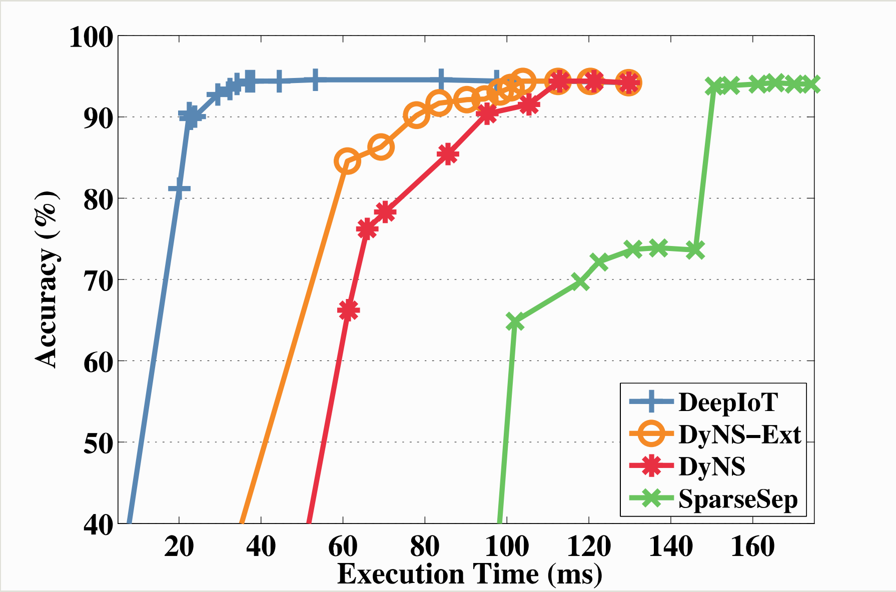
Layer		Hidden Unit						DyNS		SparseSep	
conv1a	conv1b (2 × 9)	64	64					…0.3%	60.0%	100%	100%
conv2a	conv2b (1 × 3)	64	64					…5.3%	40.5%	114%	114%
conv3a	conv3b (1 × 3)	64	64					…2.1%	35.1%	114%	114%
conv4 (2 × 8)		64						20.4%		53.7%	
conv5 (1× 6)		64						18.2%		100%	
conv6 (1 × 4)		64						12.0%		100%	
gru1		120						100%		100%	
gru2		120						100%		100%	
fc		6						99%		70%	
total				472.5K		6.16%	17.1%	74.5%		95.3%	
Test Accuracy		94.6%			94.7%		94.6%	94.6%		93.7%	



Yao, Shuochao, et al. "DeepSense: A unified deep learning framework for time-series mobile sensing data processing." *Proceedings of the 26th International Conference on World Wide Web*, 2017.

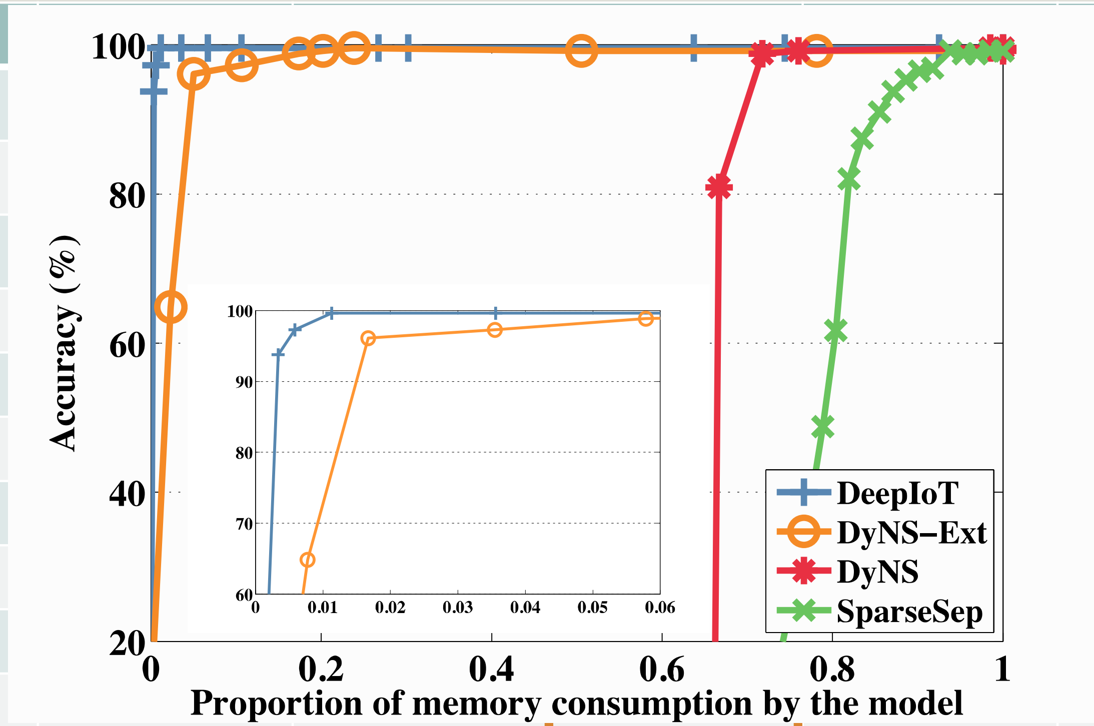
DeepIoT: Heterogeneous Human Activity Recognition with DeepSense



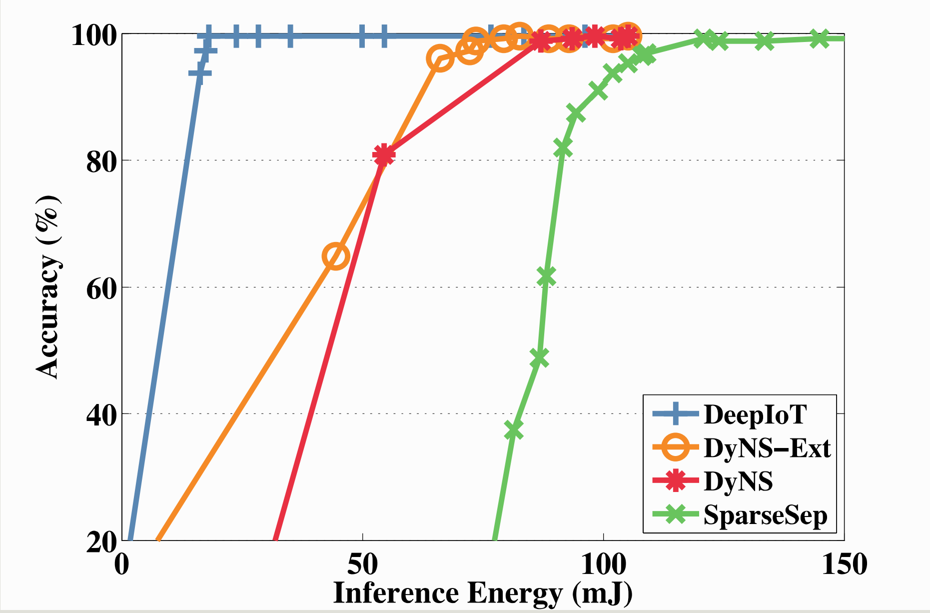


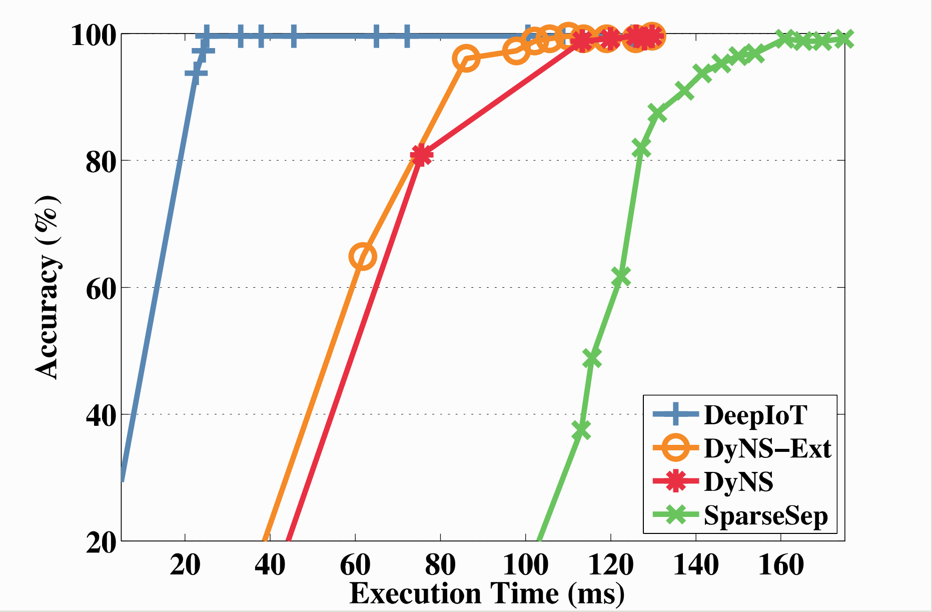
DeepIoT: Biometric Motion Analysis for User identification with DeepSense

Layer		Hidden Unit					DyNS		SparseSep	
conv1a	conv1b (2 × 9)	64	64				.8%	65.6%	100%	100%
conv2a	conv2b (1 × 3)	64	64				.6%	48.0%	114%	114%
conv3a	conv3b (1 × 3)	64	64				.4%	43.5%	114%	114%
conv4 (2 × 8)		64					29.2%		53.7%	
conv5 (1× 6)		64					23.3%		100%	
conv6 (1 × 4)		64					16.0%		100%	
gru1		120					100%		100%	
gru2		120					100%		100%	
fc		9					98%		88%	
total							77.0%		95.4%	
Test Accuracy		99.6%			99.6%	99.6%	99.6%		98.8%	



DeepIoT: Biometric Motion Analysis for User identification with DeepSense





Discussion

1. The tradeoff between compression granularity and system efficiency.

2. Compress more complex network structures.

3. Theoretical analysis or measures for the degree of compression.

Code available

https://github.com/yscacaca/DeepIoT