FIXED-POINT DEEP NEURAL NETWORK DESIGN

Wonyong Sung
Signal Processing Systems Lab.,
Electrical and Computer Engineering
Seoul National University

This talk is based on three papers

- “X1000 real-time phoneme recognition VLSI using feed-forward deep neural networks,” ICASSP 2014
- “Fixed-point feedforward deep neural network design using weights+ 1, 0, and− 1,” SiPS Workshop 2014
- “Resiliency of deep neural networks under quantization,” ICLR2016, submitted.

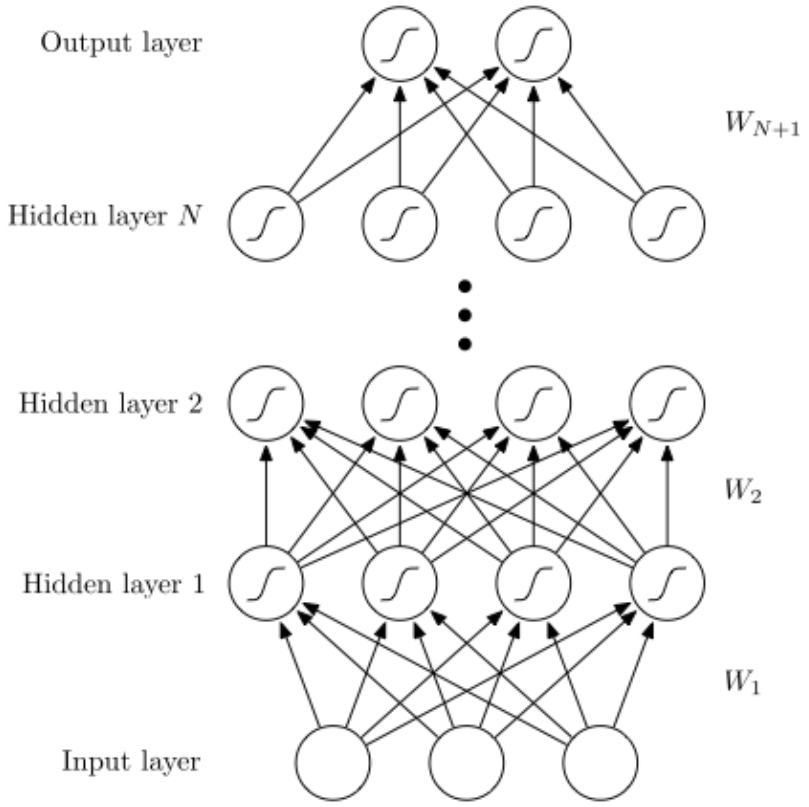
Fixed Point Feedforward Deep Neural Network Design Using Weights +1, 0, and -1 (SiPS 2014)

Kyuyeon Hwang and Wonyong Sung

Signal Processing Systems Lab.
Seoul National University, Korea

Feedforward deep neural network

- Neural network with multiple hidden layers
- Millions of parameters needed



Activation Function Bias vector

$$\mathbf{y}_{k+1} = \phi_{k+1}\left(\mathbf{W}_{k+1}\,\mathbf{y}_k + \mathbf{b}_{k+1}\right)$$

Signal vector Weight matrix

Why fixed point design?

- Too many parameters (weights) for embedded systems or hardware (VLSI or FPGA) implementations
  - Ex) acoustic modeling for speech recognition systems employs about **20 million weights**

    (4 hidden layers, 2048 units per layer)

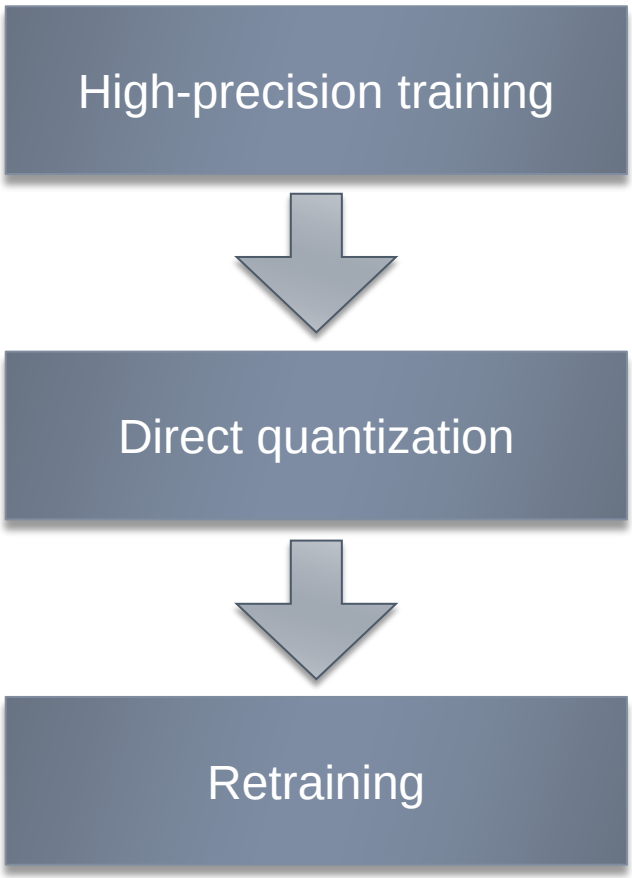
- Hardware complexity

- Solutions:
  - Reduce the network size
  - Fixed point design of weights and signals

Previous works on small or shallow networks

- Direct quantization: demands 8 ~ 16 bits for near floating-point performance
- Retraining: conducts retraining on directly quantized weights, show much improved performance even when ternary weights (+1, 0, -1) are employed, show still some performance loss.

Retrain based fixed point design of DNNs



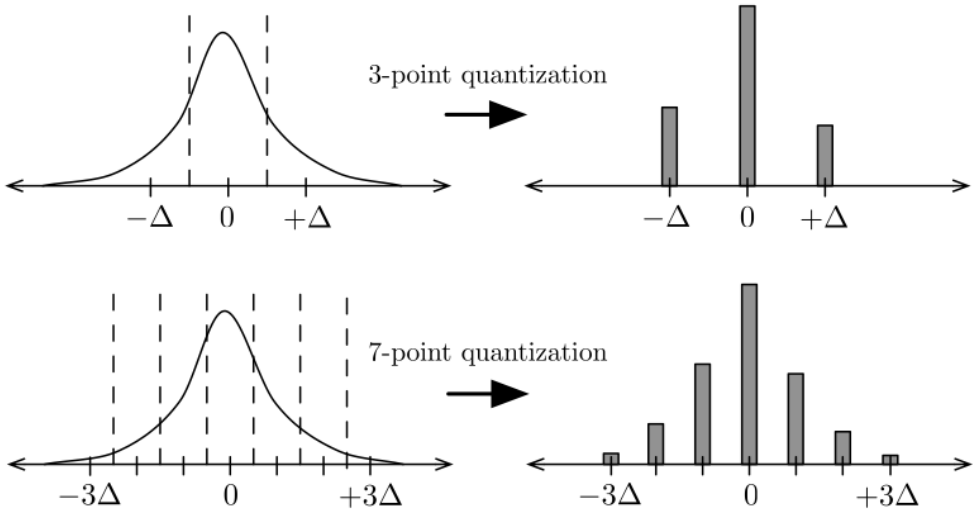
1. Pre-training (RBM)
2. Fine-tuning (error backpropagation)

Uniform quantization of weights and signals

Error backpropagation with quantized weights and signals

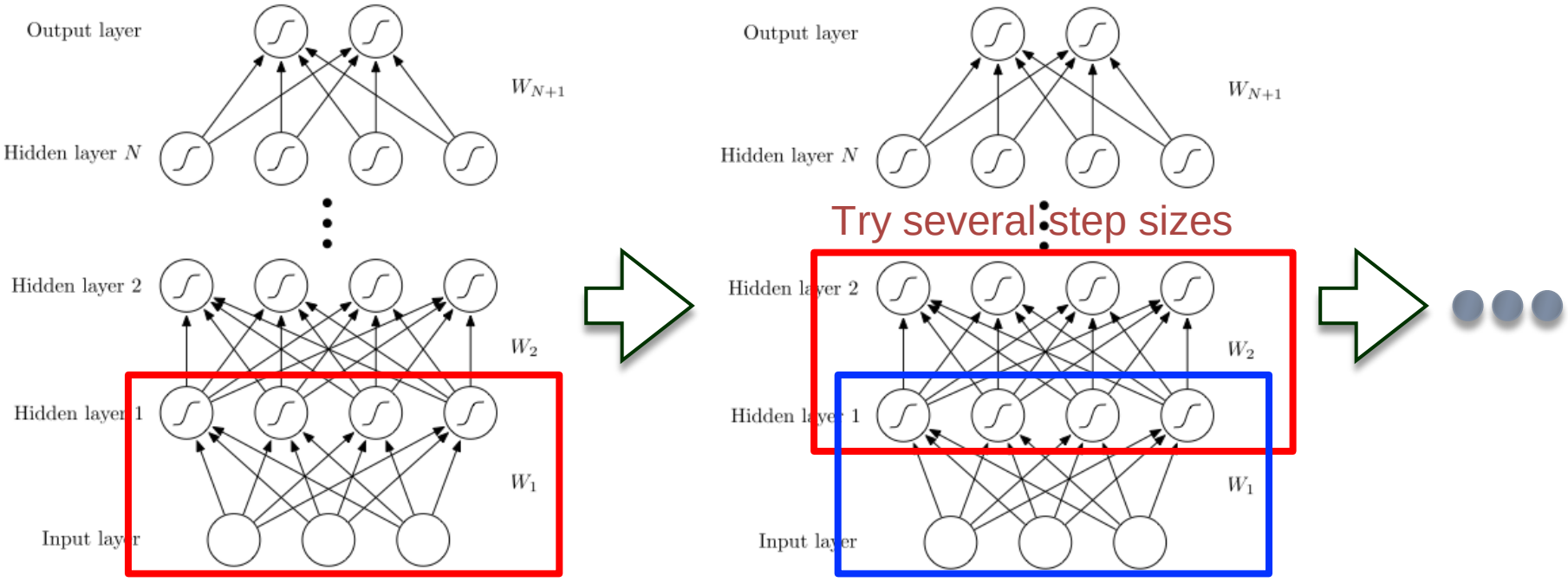
Direct quantization

- Quantize weights and signals after floating-point training
- Perform uniform quantization
- Each weight matrix has its own quantization step size, Δ
- The step sizes are determined by exhaustive search



Exhaustive search of the quantization step size, Δ

- Step by step quantization to reduce the search space



Try several step sizes to find the optimal value

Fix the step size

Direct quantization - limitation

- Induces lots of error due to quantization
- Errors are accumulated through multiple layers

- Solution:
  - Retraining the network after direct quantization
  - Retraining should be performed with quantized weights and signals

Retraining

- Modified **error backpropagation**

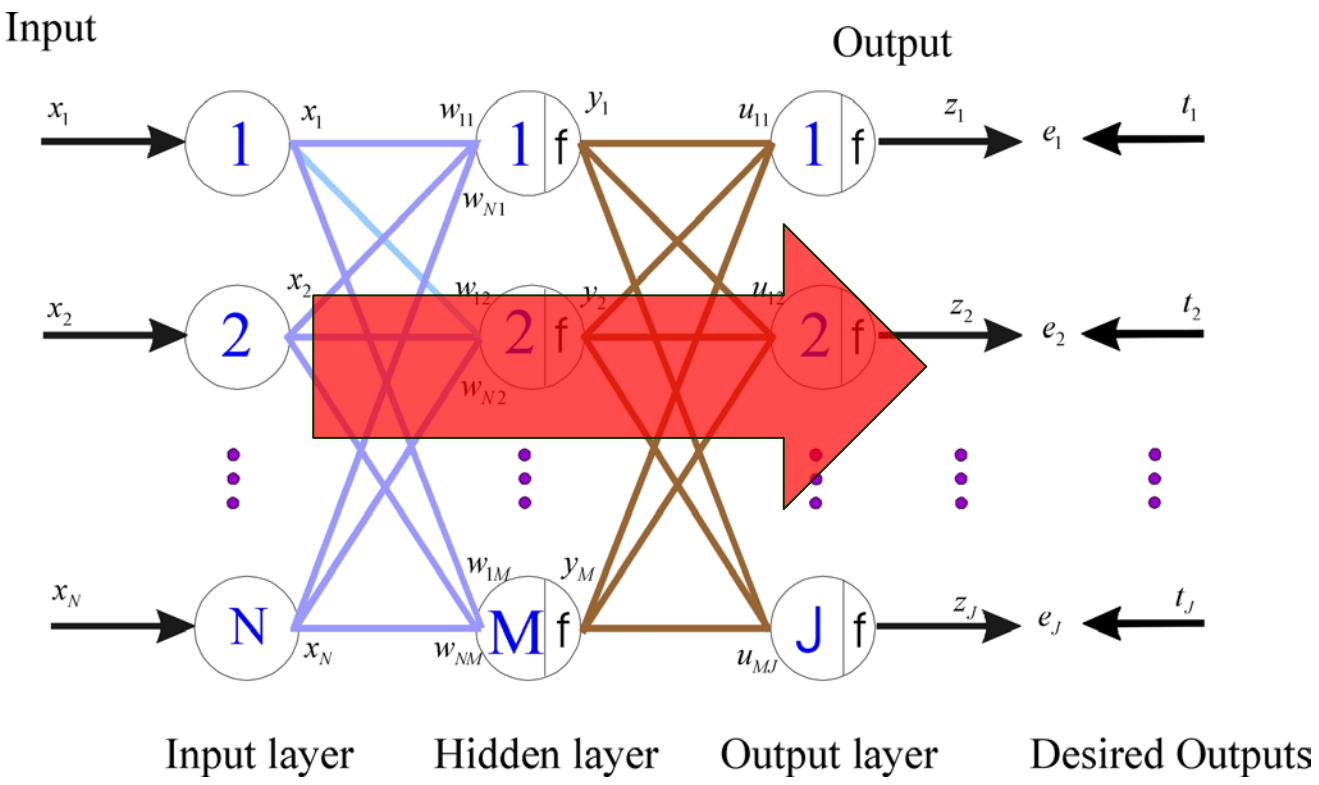
- Main idea:
  - Maintain not only quantized weights but also high-precision weights
  - Update these high precision weights, not the quantized ones, to accumulates small error gradients
  - After updating the high precision weights, quantization is performed to get new quantized weights

Backprop with quantized weights and signals

- Forward pass with quantized weights and signals

$$net_i = \sum_{j \in A_i} w_{ij}^{(q)} y_j^{(q)} \qquad y_i^{(q)} = R_i\left(\phi_i\left(net_i\right)\right)$$

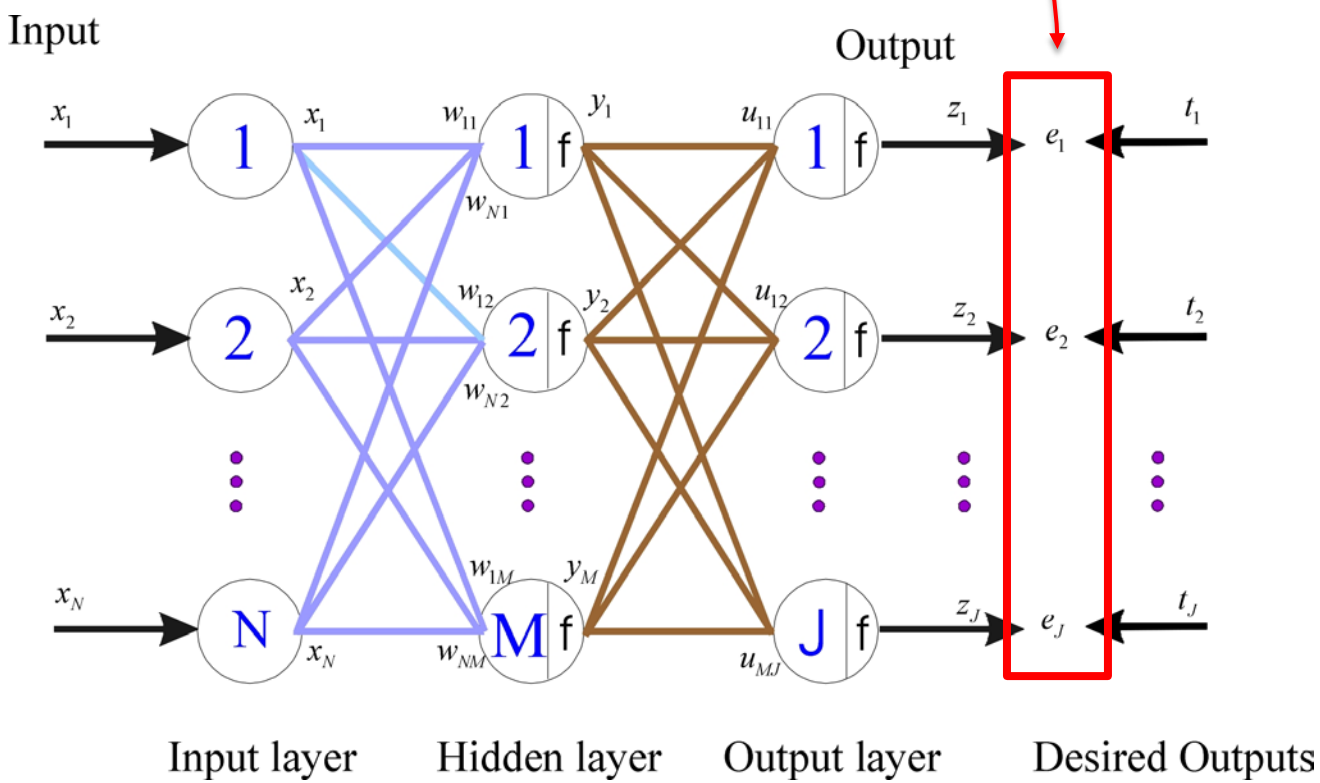
(q) denotes a quantized value
R is a quantization function



Backprop with quantized weights and signals

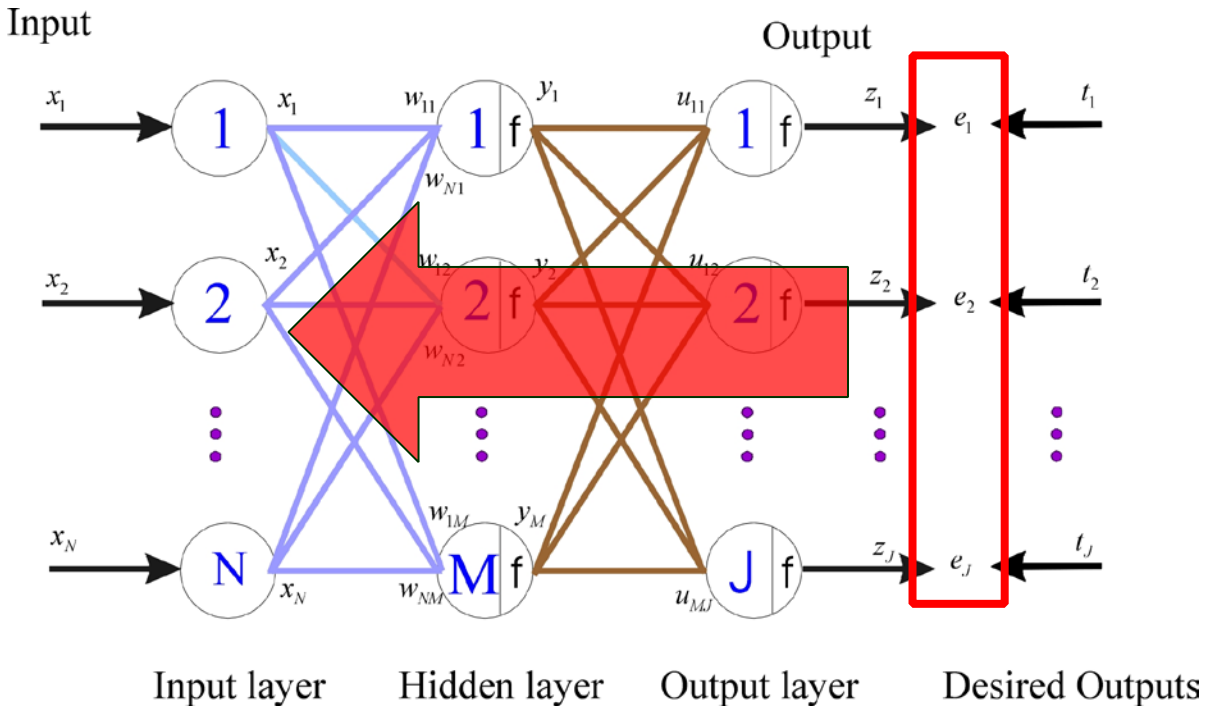
- Use **high-precision error signals**

- error signal = desired output – current output



Backprop with quantized weights and signals

- Error is backpropagated to the input layer
- Error gradient is calculated and **high-precision weights are updated**
- **Finally, quantize the updated high-precision weights**



Retraining - overall algorithm

Definition of error signal:

$$\delta_i = -\frac{\partial E}{\partial net_i}$$

Forward step:

$$net_i = \sum_{j \in A_i} w_{ij}^{(q)} y_j^{(q)}$$

$$y_i^{(q)} = R_i\left(\phi_i\left(net_i\right)\right)$$

Backward step:

$$\delta_j = \phi_j'\left(net_j\right) \sum_{i \in P_j} \delta_i w_{ij}^{(q)}$$

Gradient calculation:

$$\frac{\partial E}{\partial w_{ij}} = -\delta_i y_j^{(q)}$$

Weight update:

$$w_{ij,new} = w_{ij} - \alpha \left\langle \frac{\partial E}{\partial w_{ij}} \right\rangle$$

$$w_{ij,new}^{(q)} = Q_{ij}\left(w_{ij,new}\right)$$

- (q) denotes a quantized value
- R, Q: quantization function

Forward step with quantized weights and signals

Backward step with quantized weights and high precision errors

1. Update high precision weights
2. Quantize the new weights

Experimental results on MNIST

- Handwritten digit recognition
- Pre-trained with RBM
- DNN architecture: 784-500-500-2000-10
- Miss classification rate before quantization: 0.97 %

MISS CLASSIFICATION RATE (%) ON THE TEST SET WITH THE MNIST HANDWRITTEN DIGIT RECOGNITION EXAMPLE USING M-POINT WEIGHT QUANTIZATION.

Approach	Signal word-length	$M = 3$	$M = 7$	$M = 15$
Direct	1 bit	7.60	1.72	1.38
	2 bits	4.85	1.28	1.06
	3 bits	4.28	1.20	0.99
	8 bits	4.20	1.21	0.95
Retrain	1 bit	1.45	1.27	1.10
	2 bits	1.11	0.99	1.00
	3 bits	1.08	0.95	0.94
	8 bits	1.11	0.95	0.95

Improved performance due to regularization effect

Weight distribution

- Sparsity: about 85% of the quantized weights are zero
- Only few weights are changed after retraining

WEIGHT DISTRIBUTION OF THE MNIST HANDWRITTEN DIGIT RECOGNITION EXAMPLE BEFORE AND AFTER RETRAINING WITH 3-POINT WEIGHT QUANTIZATION AND 3-BIT SIGNAL QUANTIZATION.

Weight distribution (%)		After retraining			Sum
		$-\Delta$	0	$+\Delta$	
Before retraining	$-\Delta$	7.7	0.4	0.0	8.1
	0	1.2	85.2	1.2	87.6
	$+\Delta$	0.0	0.2	4.1	4.3
Sum		8.9	85.8	5.3	100.0

Experimental results on TIMIT

- Phoneme recognition (acoustic modeling for speech recognition)
- Pre-trained with RBM
- DNN architecture: 429-N-N-N-N-61 (N = 128, 256, 512, 1024)
- Input: 11 frames of 39-dim MFCC features
- Output: 61 phoneme labels

COMPARISON OF FRAME-LEVEL PHONE ERROR RATES AND THE PERCENTAGES OF NONZERO WEIGHTS AFTER QUANTIZATION WITH VARYING NUMBER OF UNITS PER HIDDEN LAYER WHEN 3-POINT WEIGHT AND 3-BIT SIGNAL QUANTIZATION IS APPLIED.

Hidden layer size	Frame-level phone error rate (%)		Nonzeros (%)
	Floating-point	Fixed-point	
128	30.38	37.35 (+6.97)	56.26
256	28.19	32.53 (+4.34)	47.56
512	26.84	28.91 (+2.07)	40.04
1024	26.24	27.63 (+1.39)	37.19

Combined with dropout

- Dropout: regularization method to prevent overfitting by randomly dropping out certain fraction of units
- With dropout, the performance gap between floating-point and fixed-point implementations can be reduced

- Results on TIMIT:

COMPARISON OF FRAME-LEVEL PHONE ERROR RATES WITH AND WITHOUT DROPOUT WHEN THE HIDDEN LAYER SIZE IS 1024. FOR FIXED-POINT EXPERIMENTS, 3-POINT WEIGHT AND 3-BIT SIGNAL QUANTIZATION IS APPLIED.

Dropout	Frame-level phone error rate (%)	
	Floating-point	Fixed-point
No	26.24	27.63 (+1.39)
Yes	22.04	22.70 (+0.66)

Resiliency of deep neural networks under quantization (ICLR 2016 submitted)

Wonyong Sung, Sungho Shin, and Kyuyeon Hwang

Overview

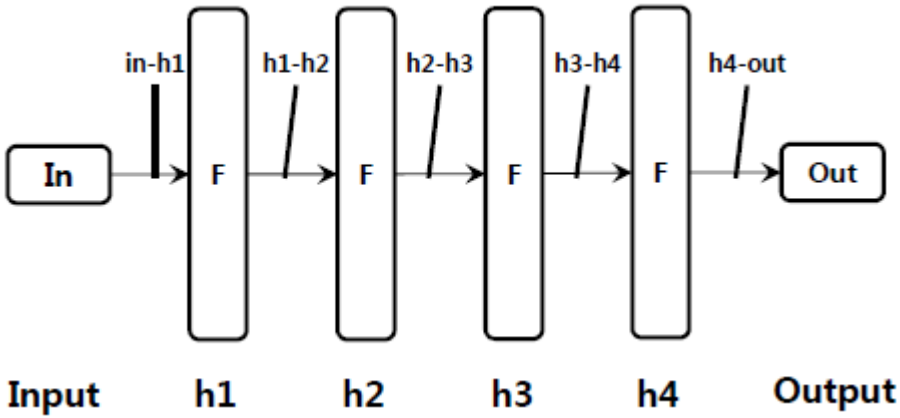
- To know the effects of network complexity on the performance of quantized DNNs
  - Can the network complexity be traded with the precision?

  In other words, can we implement DNNs showing near floating-point performance using only ternary weights?

- Conduct experiments on feed-forward DNN and CNN while varying the network complexity. We also have results on RNN (though not shown in this paper)

Feed-forward DNN

- Reference design: four hidden layers, phoneme recognition
- The input layer has 1,353 units to for Fourier-transform-based filter-bank
- N_h changed from 32, 64, 128, 256, 512, and 1024



CNN

- CNN for CIFAR-10 classification
- Reference design is 32-32-64
- The complexity is changed to: 8-8-16, 16-16-32, 32-32-64, 64-64-128, 96-96-192, 128-128-256

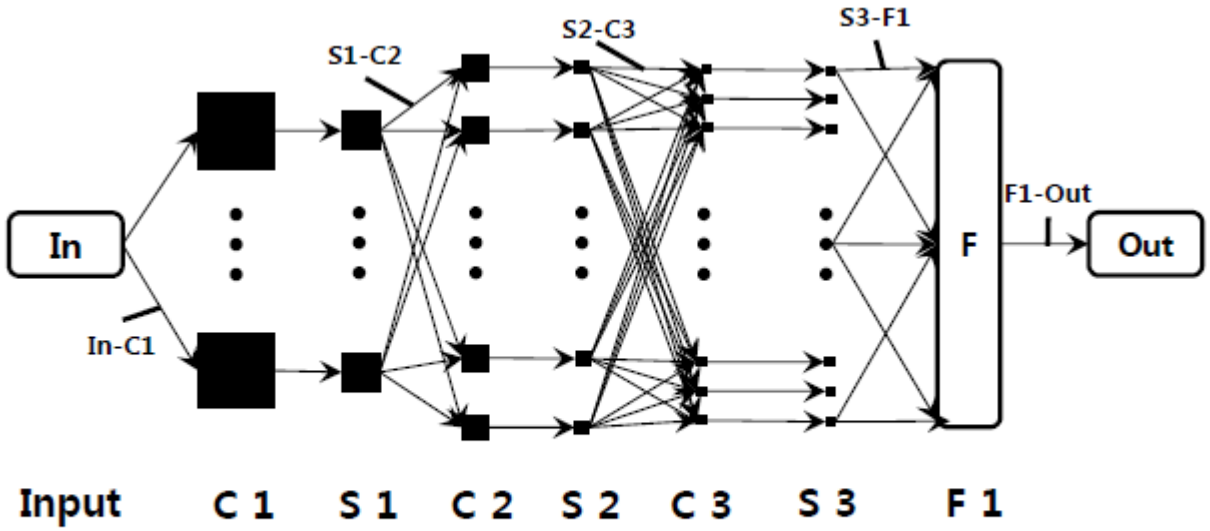
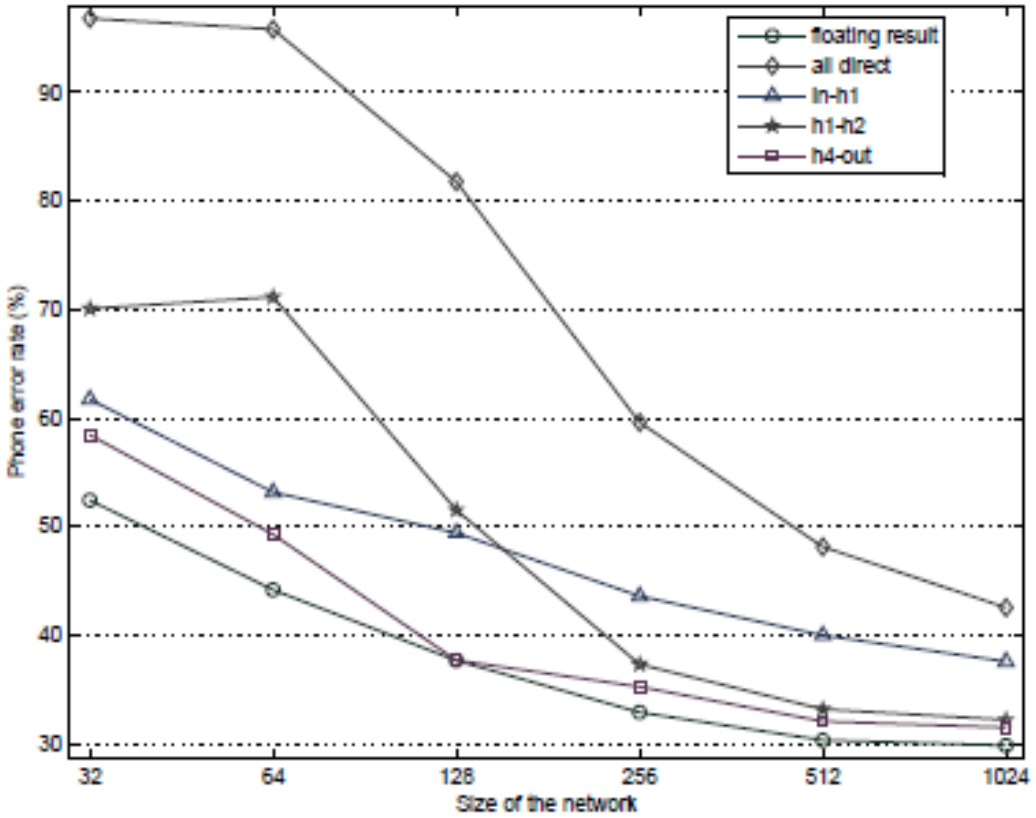


Figure 2: CNN structure with 3 convolution layers and 1 fully-connected layers.

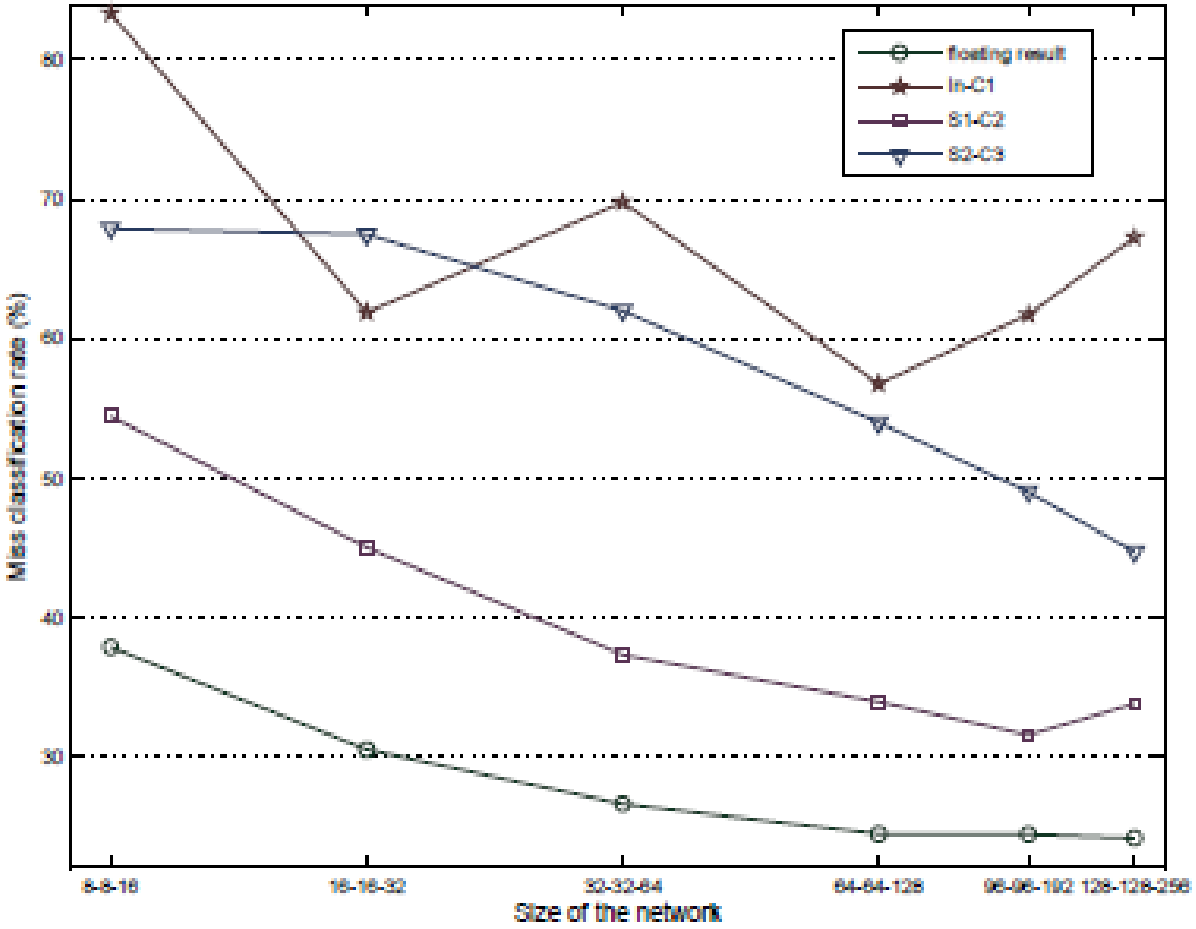
FFDNN – direct quantization

- 2bit (ternary level) quantization applied to each weight group for sensitivity analysis.
- In-h1 layer does not improve
- h1-h2 layer, h4-out improves
- More than 10 dB difference compared to floating-point



CNN – direct quantization

- In-C1 shows the highest sensitivity
- S1-C2, S2-C3 decreases, but slowly



Direct quantization analysis

- Quantization induced distortion is decreasing as the number of inputs are increasing.
- The number of input nodes does not increase when the complexity is changed. As a result, ‘In-h1’ and ‘In-C1’ do not improves.

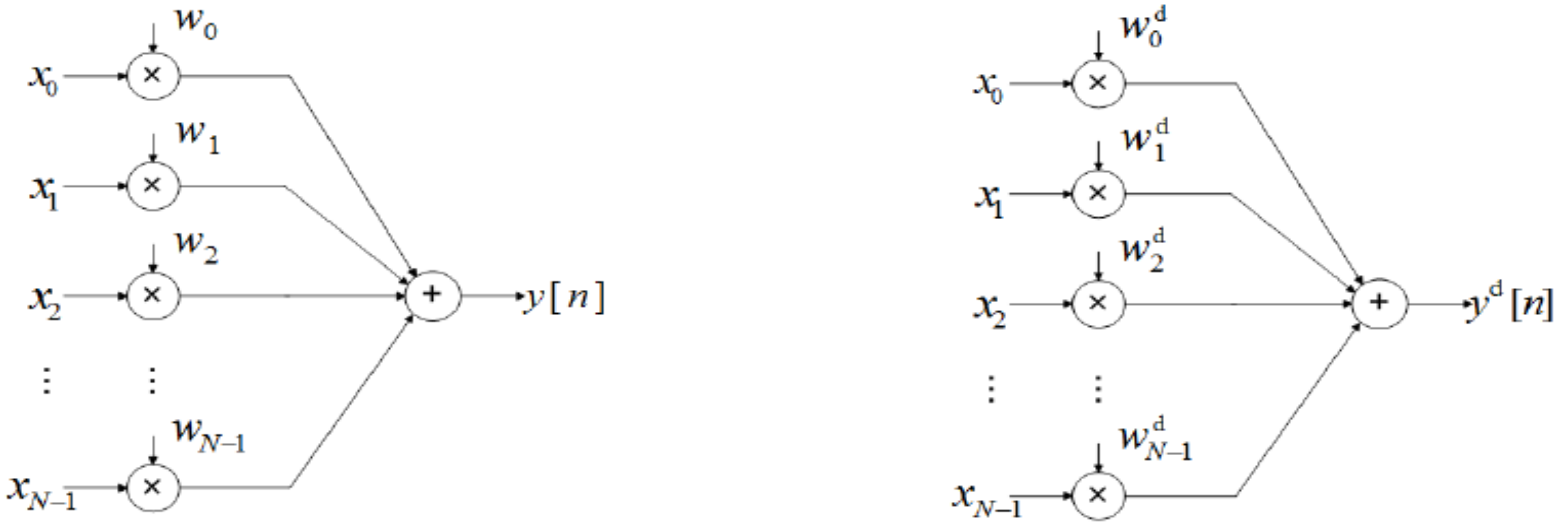


Figure 3: Computation model for a unit in the hidden layer j (left: floating-point, right: distortion).

Direct quantization summary

- About 6-bit is needed

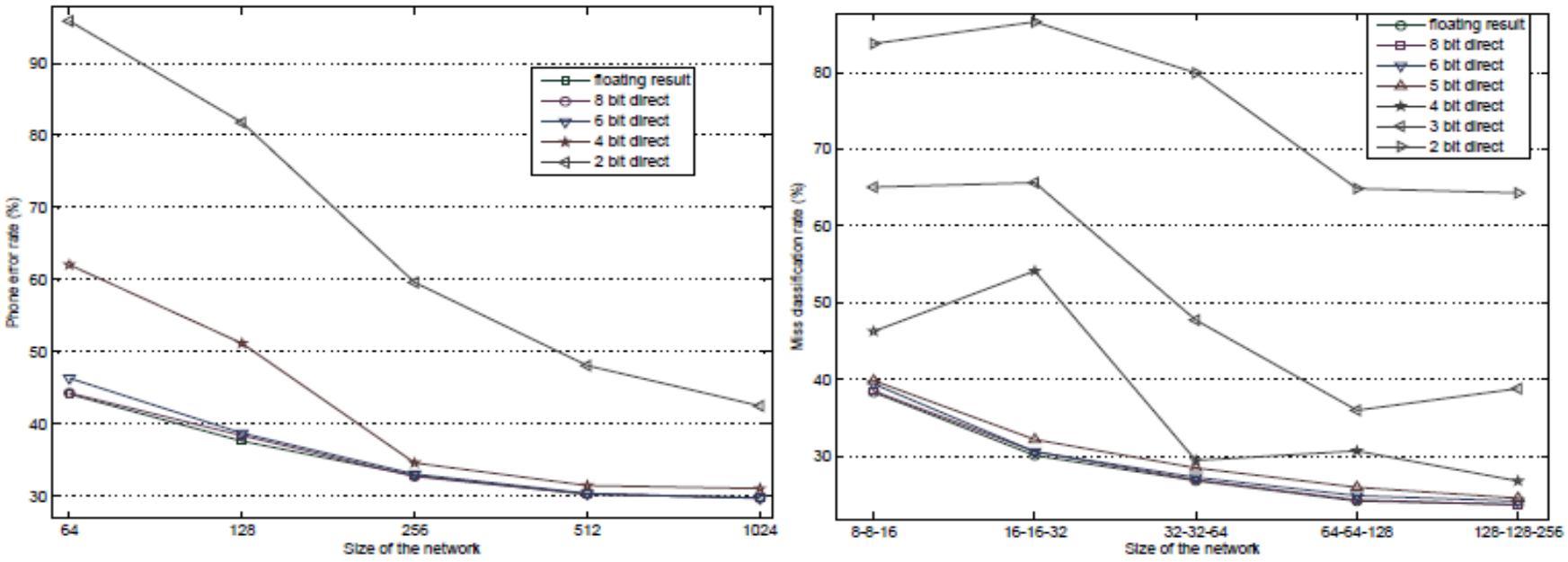


Figure 5: Performance of direct quantization with multiple precision (left:FFDNN, right:CNN).

Retraining on quantized weights

- The gap disappears when the network size is large (512 or bigger). 2bit (7-level) quantization shows good performance. For small networks, even 3bit shows some difference.

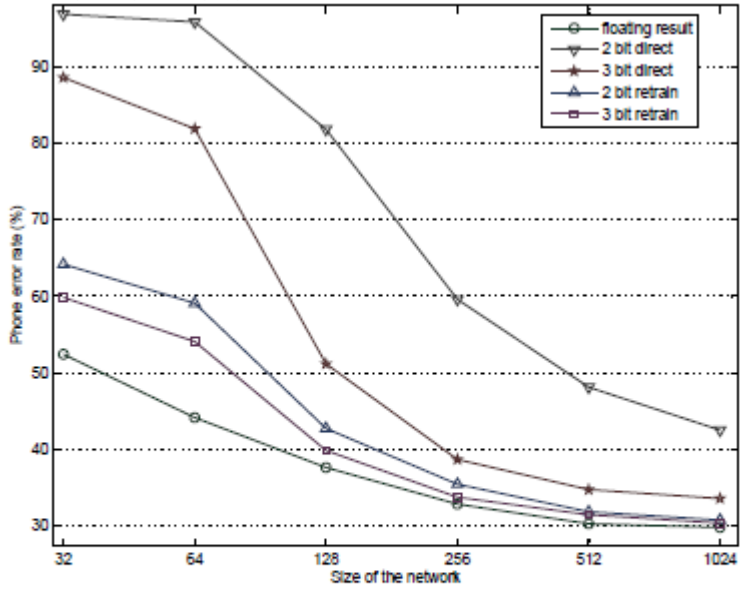


Figure 6: Comparison of retrain-based and direct quantization for DNN. All the weights are quantized with ternary and 7-level weights.

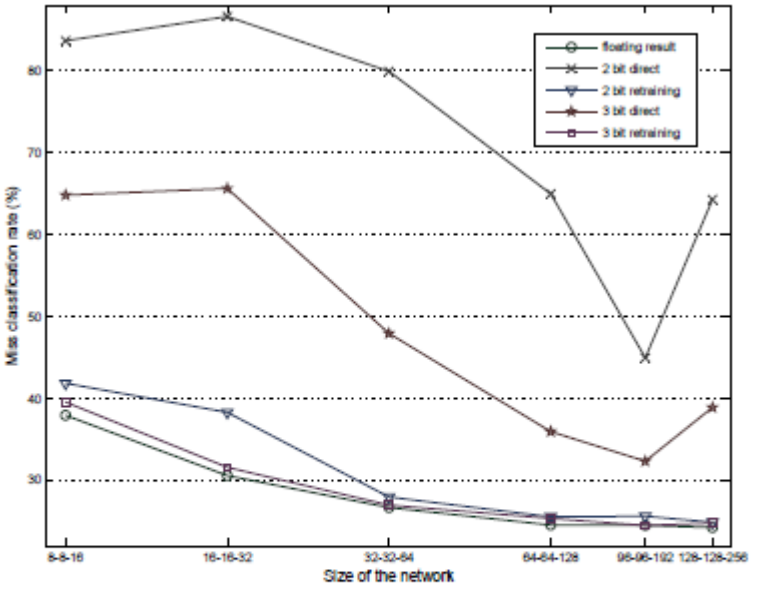


Figure 7: Comparison of retrain-based and direct quantization for CNN. All the weights are quantized with ternary and 7-level weights.

Retraining results when the number of levels reduced

Table 1: Depth change in DNN

Number of layers (Floating-point result)	Quant Level	Direct	Retraining	Difference
4 (30.31%)	3-level	48.13%	31.86%	1.55%
	7-level	34.77%	31.49%	1.18%
3 (30.81%)	3-level	49.27%	33.05%	2.24%
	7-level	36.58%	31.72%	0.91%
2 (31.51%)	3-level	47.74%	33.89%	2.38%
	7-level	36.99%	33.04%	1.53%
1 (34.67%)	3-level	69.88%	38.58%	3.91%
	7-level	56.81%	36.57%	1.90%

Table 2: Depth change in CNN

Layer (Floating-point result)	Quant Level	Direct	Retraining	Difference
64 (34.19%)	3-level	72.95%	35.37%	1.18%
	7-level	46.60%	34.15%	-0.04%
32-64 (29.29%)	3-level	55.30%	29.51%	0.22%
	7-level	39.80%	29.32%	0.03%
32-32-64 (26.87%)	3-level	79.88%	27.94%	1.07%
	7-level	47.91%	26.95%	0.08%

Summary

- The performance gap between the floating-point and the fixed-point networks with ternary weights (+1, 0, -1) almost vanishes when the DNNs are in the performance saturation region for the given training data.
- When the complexity of DNNs are reduced, by lowering either the number of units, feature maps, or hidden layers, the performance gap between them increases.

- For design with limited hardware resources, when the size of the reference DNN is relatively small, it is advised to employ a very low-precision arithmetic and, instead, increase the network complexity as much as the hardware capacity allows. But, when the DNNs are in the performance saturation region, increasing the arithmetic precision gains slightly more because growing the ’already-big’ network size brings almost no performance advantages.

X1000 real-time phoneme recognition VLSI using feed-forward deep neural networks (ICASSP 2014)

Jonghong Kim, Kyuyeon Hwang, Wonyong Sung

FFDNN for phoneme recognition

- At each frame (10 msec), the monophone likelihood is obtained (61 phoneme)
- The hidden layer size is 1024
- Each hidden layer contains about 1Mega weights.

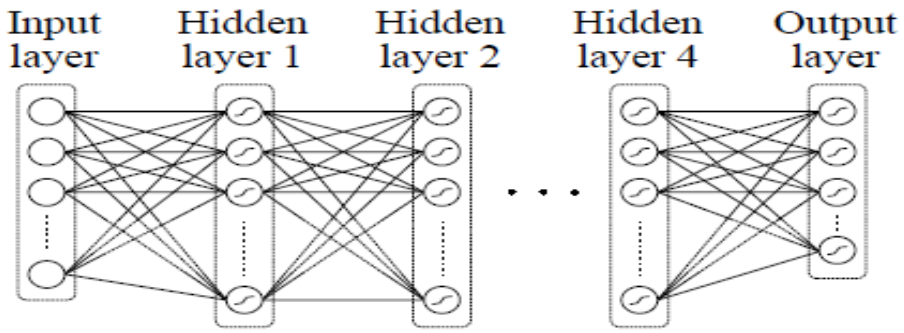


Fig. 1: A deep neural network with four hidden layers.

Fixed-point optimization

- 3-level (2 bits) weight quantization with retraining
- Total number of weights is about 8Mbits
- DNN architecture: 429-1024-1024-1024-1024-61
- Signal representation 4 bits, accumulator 8 bits

Table 1: Frame level phoneme recognition error rate according to the training method with floating-point and fixed-point arithmetic.

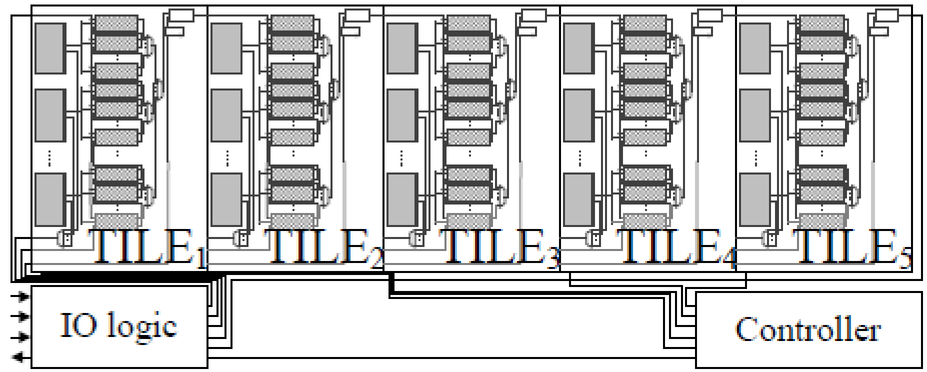
Hidden layer size and training method	Error rate (%)	
	Floating-point	Fixed-point
1,024-unit-layer with conv. training	26.24	27.63
1,024-unit-layer with drop-out training	23.71	24.93

Architecture

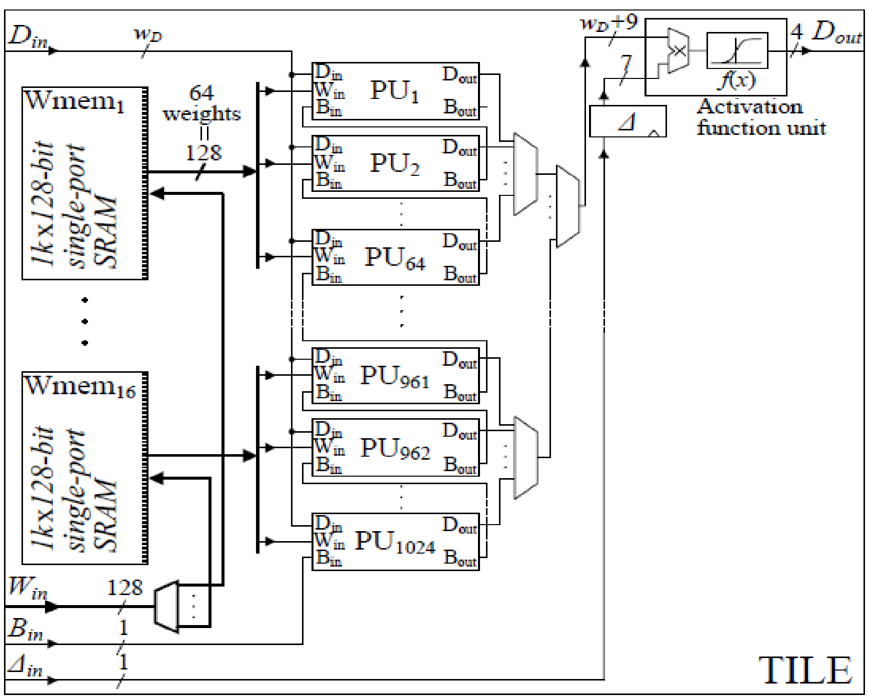
- For each layer, 1024 processing elements are implemented, it demands 1024 clocks to process one frame (outer product method).
- With 100MHz clock, it takes only 10μsec, which is x1000 faster than the real-time needed (10msec frame)

- Outer product method

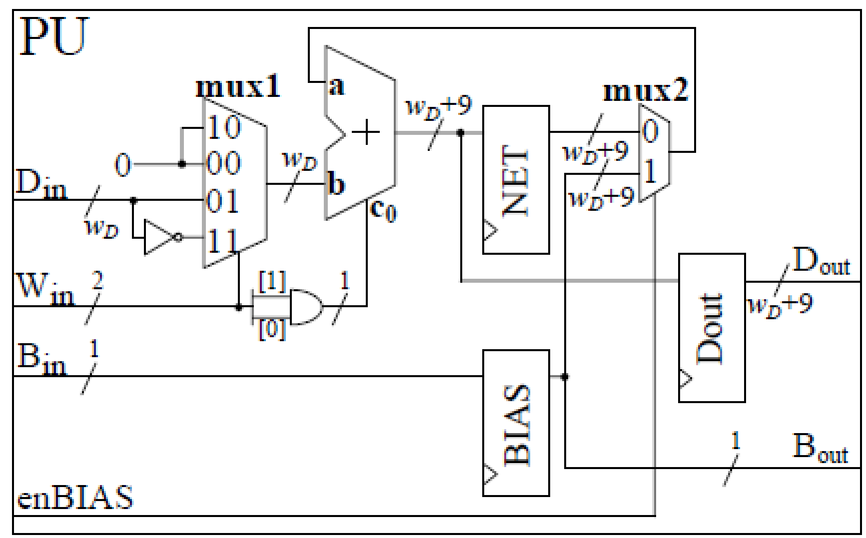
```
Parallel for PE = 0 to 1023
    Y[PE] = 0.0
    Serial for i = 0 to 1023
        Y[PE] +=W[PE,i]*x[i]
```


TILE1
TILE2
TILE3
TILE4
TILE5
IO logic
Controller

(a)


Din
wD
Wmem1
1kx128-bit single-port SRAM
64 weights = 128
Wmem16
1kx128-bit single-port SRAM
PU1
PU2
PU64
PU961
PU962
PU1024
Din Win Bin
Dout Bout
wD+9
7
4
Dout
f(x)
Activation function unit
Δ
Win
128
Bin
1
Δin
1
TILE

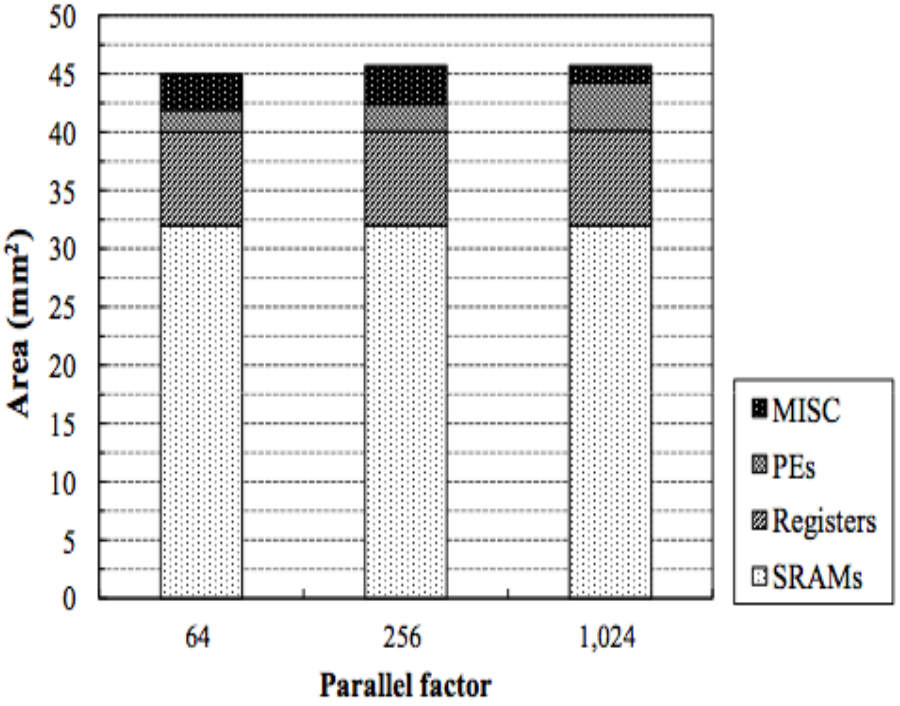
(b)

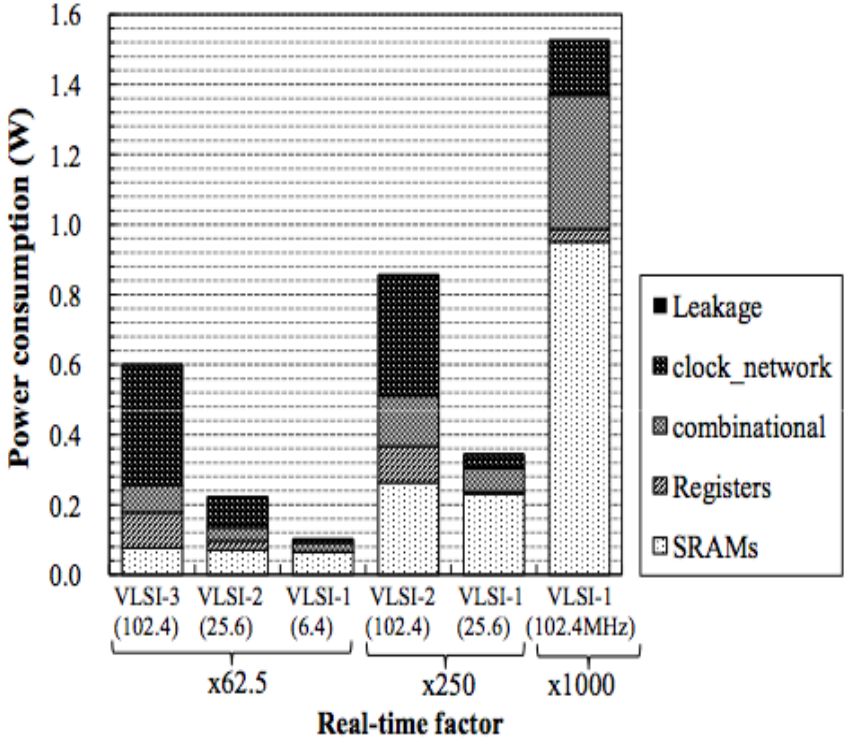

PU
mux1
mux2
a
b
c0
+
wD+9
wD
0
10
00
01
11
Din
wD
Win
2
[1]
[0]
1
Bin
1
NET
BIAS
Dout
wD+9
Dout
1
Bout
enBIAS

(c)

VLSI implementation

(0.13-μm CMOS technology)





- VLSIs with various parallel factors
- Very high throughput (x1000) or very low-power
  - VLSI-1: 1024 / VLSI-2: 256 / VLSI-3: 64

FPGA based DNN

- Xilinx ZC706 evaluation board with 218K LUT
- Parallel factor of 512 used, with 3bit weight precision, for MNIST
- 66K image recognition per sec., with about 5W
  (GPU: 250K images per sec, about 150W)
- Will scale up using Virtex Ultrascale (BRAM 132Mb)

Table 2. FPGA resource utilization for digit recognition with different weight precision.

Resource	FF	LUT	BRAM	DSP
3-bit without DSP	138095	168344	323	0
3-bit with DSP	130802	121173	323	900
8-bit fixed point	136677	213593	750.5	900
Available	437200	218600	545	900

Summary

- We have developed a VLSI for DNN, achieving x1000 times throughput (vs real-time) with approximately 4,000 processing elements.
- The proposed retraining procedure yields superior results compared to the direct quantization method especially when only 3-point weights (+1, 0, and -1) are used
- The signal word length can also be reduced to 3 bits without sacrificing the performance much
- This research is useful not only for hardware based implementations but also real-time software development
- The area for weight storage takes the largest portion.

Conclusion

- For highly complex networks (DNN, CNN, RNN), it seems possible to trade the network complexity and the precision. We can design near floating-point performance networks with ternary weights, which yields hardware efficient design.
- VLSI or FPGA based implementations can provide very low-power and high-throughput solutions for DNN, CNN, and RNN.

Thank You !!