Binarized Neural Networks

outlines

- **Binarized Neural Network**
- **XNOR Neural Network**
- Brief introduction
  - FINN
  - Residual Binarized Neural Network （ReBenet）
  - BI-Real NETWORK
  - DOREFA-NET

Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to $+1$ or -1

Matthieu Courbariaux*[1] MATTHIEU.COURBARIAUX@GMAIL.COM
Itay Hubara*[2] ITAYHUBARA@GMAIL.COM
Daniel Soudry[3] DANIEL.SOUDRY@GMAIL.COM
Ran El-Yaniv[2] RANI@CS.TECHNION.AC.IL
Yoshua Bengio[1,4] YOSHUA.UMONTREAL@GMAIL.COM

[1]Université de Montréal
[2]Technion Israel Institute of Technology
[3]Columbia University
[4]CIFAR Senior Fellow
*Indicates equal contribution. Ordering determined by coin flip.

Binarized： deterministic & stochastic

$$x^b = \text{Sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

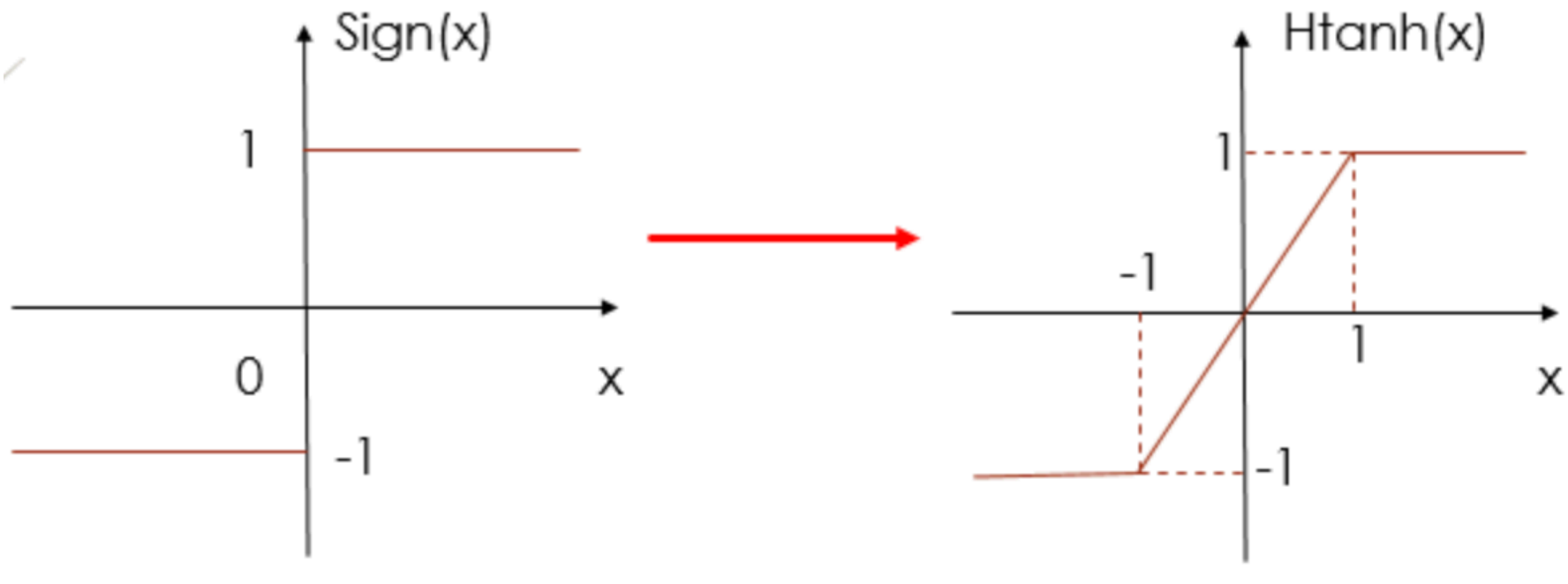
Used

$$x^b = \begin{cases} +1 & \text{with probability } p = \sigma(x), \\ -1 & \text{with probability } 1 - p. \end{cases}$$

$$\sigma(x) = \text{clip}(\frac{x+1}{2}, 0, 1) = \max(0, \min(1, \frac{x+1}{2}))$$

hard tanh

$$q = \mathrm{Sign}(r)$$



Weight： Binarized
Gradient： ?

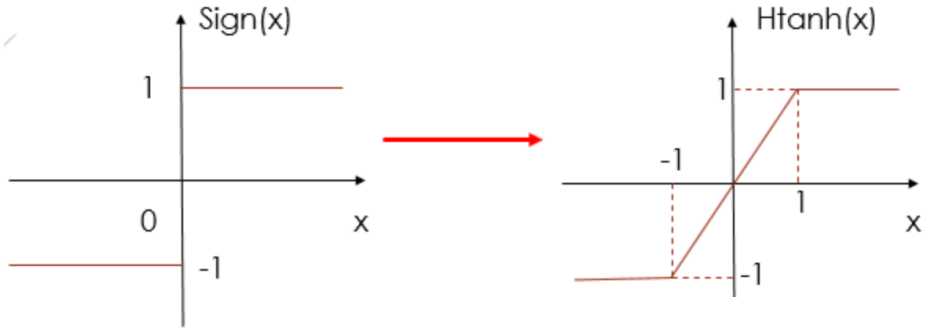
- The **magnitude** of the gradient is very **small**

- Additive effect：the noise is generally considered to be a normal distribution, so the cumulative gradient can be used to **average the noise.**

Gradient： real value （high precision）

How to do the propagation （forward & backward）

$q = \mathrm{Sign}(r)$



$$q = \mathrm{Sign}(r)$$

$$\mathrm{Htanh}(x) = \mathrm{Clip}(x, -1, 1) = \max(-1, \min(1, x))$$

$$\frac{\partial C}{\partial r} \qquad \frac{\partial C}{\partial q}$$

$$g_r = g_q 1_{|r| \leq 1}.$$

Tips：

- **For weight：**
- **clipping the weights during training**

- When using a weight w^r, quantize it using $w^b = \text{Sign}(w^r)$.

1.Computing the parameters' gradient
1.1 Forward propagation

for $k = 1$ to L **do**
  $W_k^b \leftarrow \text{Binarize}(W_k)$
  $s_k \leftarrow a_{k-1}^b W_k^b$
  $a_k \leftarrow \text{BatchNorm}(s_k, \theta_k)$
  if $k < L$ **then**
    $a_k^b \leftarrow \text{Binarize}(a_k)$
  end if
end for

1.2. Backward propagation

{Please note that the gradients are not binary.}

Compute $g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing a_L and a^*

for $k = L$ to 1 **do**

  if $k < L$ **then**

    $g_{a_k} \leftarrow g_{a_k^b} \circ 1_{|a_k| \leq 1}$

  end if

  $(g_{s_k}, g_{\theta_k}) \leftarrow \text{BackBatchNorm}(g_{a_k}, s_k, \theta_k)$

  $g_{a_{k-1}^b} \leftarrow g_{s_k} W_k^b$

  $g_{W_k^b} \leftarrow g_{s_k}^\top a_{k-1}^b$

end for

{2. Accumulating the parameters' gradient:}

for $k = 1$ to L **do**

  $\theta_k^{t+1} \leftarrow \text{Update}(\theta_k, \eta, g_{\theta_k})$

  $W_k^{t+1} \leftarrow \text{Clip}(\text{Update}(W_k, \gamma_k \eta, g_{W_k^b}), -1, 1)$

  $\eta^{t+1} \leftarrow \lambda \eta$

end for

$\text{Htanh}(x) = \text{Clip}(x, -1, 1) = \max(-1, \min(1, x))$

talk later

Weight is binary

Shift BN

Algorithm 2 Shift based Batch Normalizing Transform, applied to activation x over a mini-batch. $AP2(x) = \text{sign}(x) \times 2^{\text{round}(\log 2|x|)}$ is the approximate power-of-2, and $\ll\gg$ stands for **both** left and right binary shift.

Require: Values of x over a mini-batch: $B = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Ensure: $\{y_i = \text{BN}(x_i, \gamma, \beta)\}$

$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ {mini-batch mean}

$C(x_i) \leftarrow (x_i - \mu_B)$ {centered input}

$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(C(x_i) \ll\gg AP2(C(x_i)))$ {apx variance}

$\hat{x}_i \leftarrow C(x_i) \ll\gg AP2((\sqrt{\sigma_B^2 + \epsilon})^{-1})$ {normalize}

$y_i \leftarrow AP2(\gamma) \ll\gg \hat{x}_i$ {scale and shift}

trick

Estimate the value using the base 2 exponential function

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$$

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

denominator>0

Shift AdaMax

Algorithm 3 Shift based AdaMax learning rule (Kingma & Ba, 2014). g_t^2 indicates the elementwise square $g_t \circ g_t$. Good default settings are $\alpha = 2^{-10}$, $1-\beta_1 = 2^{-3}$, $1-\beta_2 = 2^{-10}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: Previous parameters θ_{t-1} and their gradient g_t, and learning rate α.

Ensure: Updated parameters θ_t

{Biased 1st and 2nd raw moment estimates:}

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$

$v_t \leftarrow \max(\beta_2 \cdot v_{t-1}, |g_t|)$

{Updated parameters:}

$\theta_t \leftarrow \theta_{t-1} - (\alpha \ll\gg (1 - \beta_1)) \cdot \hat{m} \ll\gg v_t^{-1})$

1 SGD

2 Momentum

3 Nesterov

4 Adagrad

5 Adadelta

6 RMSprop

7 Adam

8 AdaMax

9 Nadam

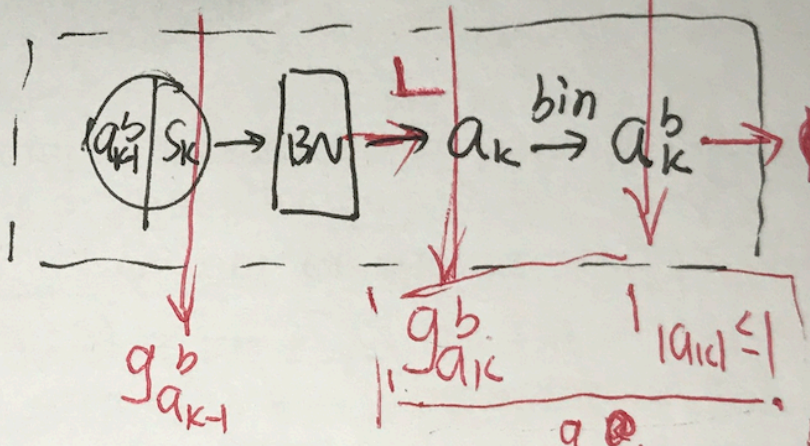
https://blog.csdn.net/fishmai/article/details/52510826

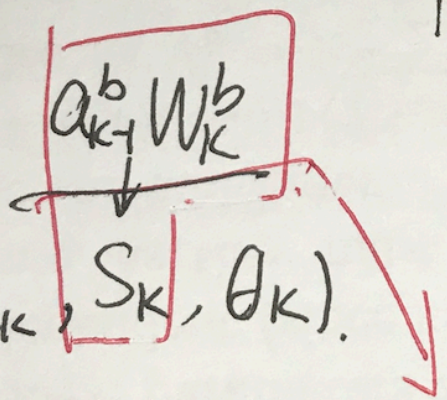
(Htanh())

$$g_{a_k} = g_{a_k^b} \circ 1_{|a_k| \le 1}$$

$$\frac{\partial C}{\partial a_k} = \frac{\partial C}{\partial a_k^b} \cdot \frac{\partial a_k^b}{\partial a_k}$$

$$\frac{\partial C}{\partial a_{k-1}^b} = \frac{\partial C}{\partial a_k^b} \frac{\partial a_k^b}{\partial a_k} \frac{\partial a_k}{\partial s_k} \frac{\partial s_k}{\partial a_{k-1}^b}$$





~~Backk~~ BackBN(g_{a_k}, s_k, θ_k).

$$\frac{\partial L}{\partial a_{k-1}^b} = \frac{\partial L}{\partial a_k^b} \cdot \frac{\partial a_k^b}{\partial a_{k-1}^b} = g_{s_k} w_k^b$$

$$\frac{\partial L}{\partial w_k^b} = \frac{\partial L}{\partial s_k^b} \frac{\partial s_k^b}{\partial w_k^b} = g_{s_k}^T a_{k-1}^b$$

s_k

$y = \gamma x + \beta$ → g_{θ_k}

$\{\gamma, \beta\}$

Algorithm 4 Running a BNN. L is the number of layers.

Require: a vector of 8-bit inputs a_0, the binary weights W^b and the BatchNorm parameters θ.

Ensure: the MLP output a_L.

$\{1.$ First layer:$\}$

$a_1 \leftarrow 0$

for $n = 1$ to 8 **do**

$a_1 \leftarrow a_1 + 2^{n-1} \times \text{XnorDotProduct}(\mathrm{a_0^n}, \mathrm{W_1^b})$

end for

$a_1^b \leftarrow \text{Sign}(\text{BatchNorm}(a_1, \theta_1))$

$\{2.$ Remaining hidden layers:$\}$

for $k = 2$ to $L - 1$ **do**

$a_k \leftarrow \text{XnorDotProduct}(a_{k-1}^b, W_k^b)$

$a_k^b \leftarrow \text{Sign}(\text{BatchNorm}(a_k, \theta_k))$

end for

$\{3.$ Output layer:$\}$

$a_L \leftarrow \text{XnorDotProduct}(a_{L-1}^b, W_L^b)$

$a_L \leftarrow \text{BatchNorm}(a_L, \theta_L)$

Image pixels to binary value: it uses 8 bit to represent one pixel.

For example, cifar-10: 32*32=1024
Inputs is a 1024 8 bit vector.

Core trick：**bit manipulation**
talk later

Also can consider it as
Binary activation function

XnorDotProduct

- XNOR -----> ~ XOR

a=[1,-1, 1, 1, -1] **a=[1,0,1,1,0]**

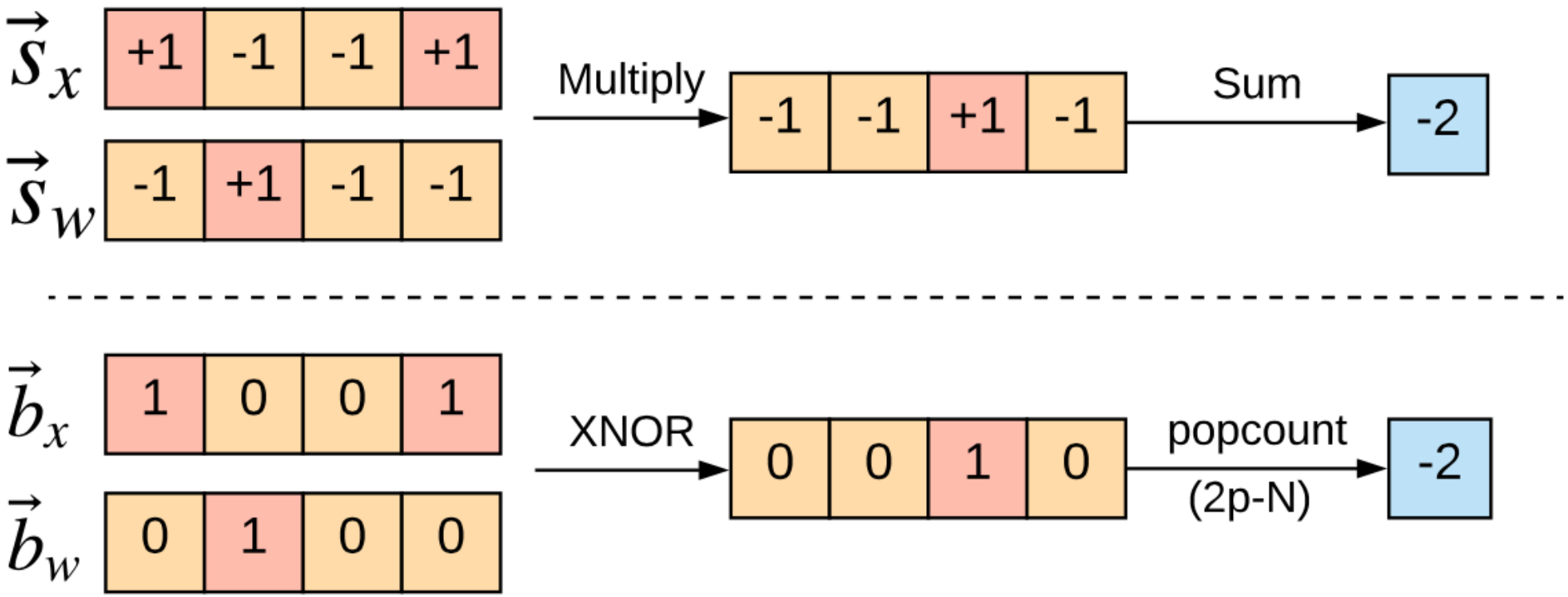
W=[-1,1,1,-1,-1] **W=[0,1,1,0,0]**

a1*w1+a2*w2+a3*w3+a4*w4+a5*w5=1*-1+-1*1+1*1+1*-1+-1*-1=-1

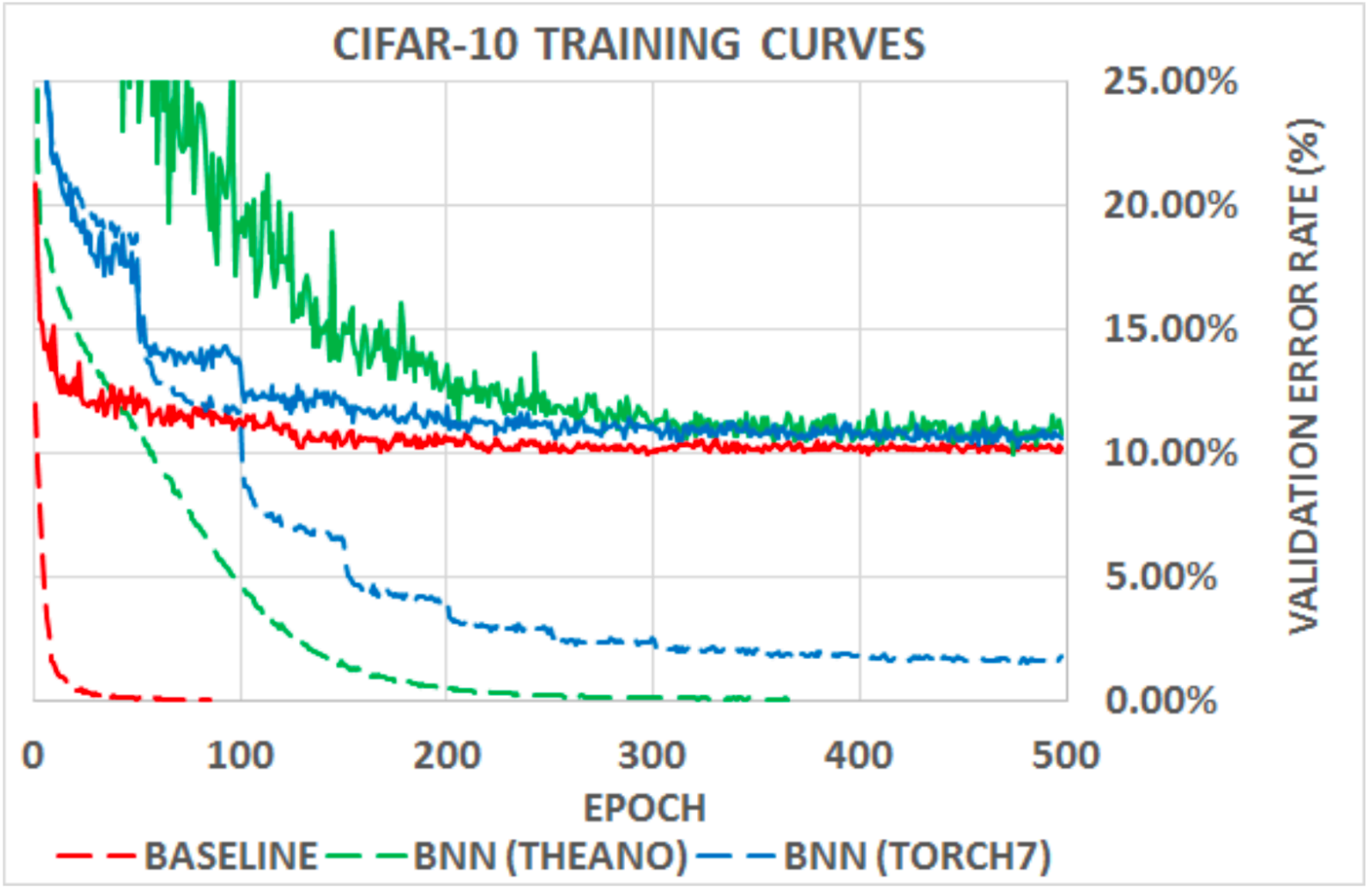
a^W=[1^0,0^1,1^1,1^0,0^0]=11010

Popcount(a^w)=3

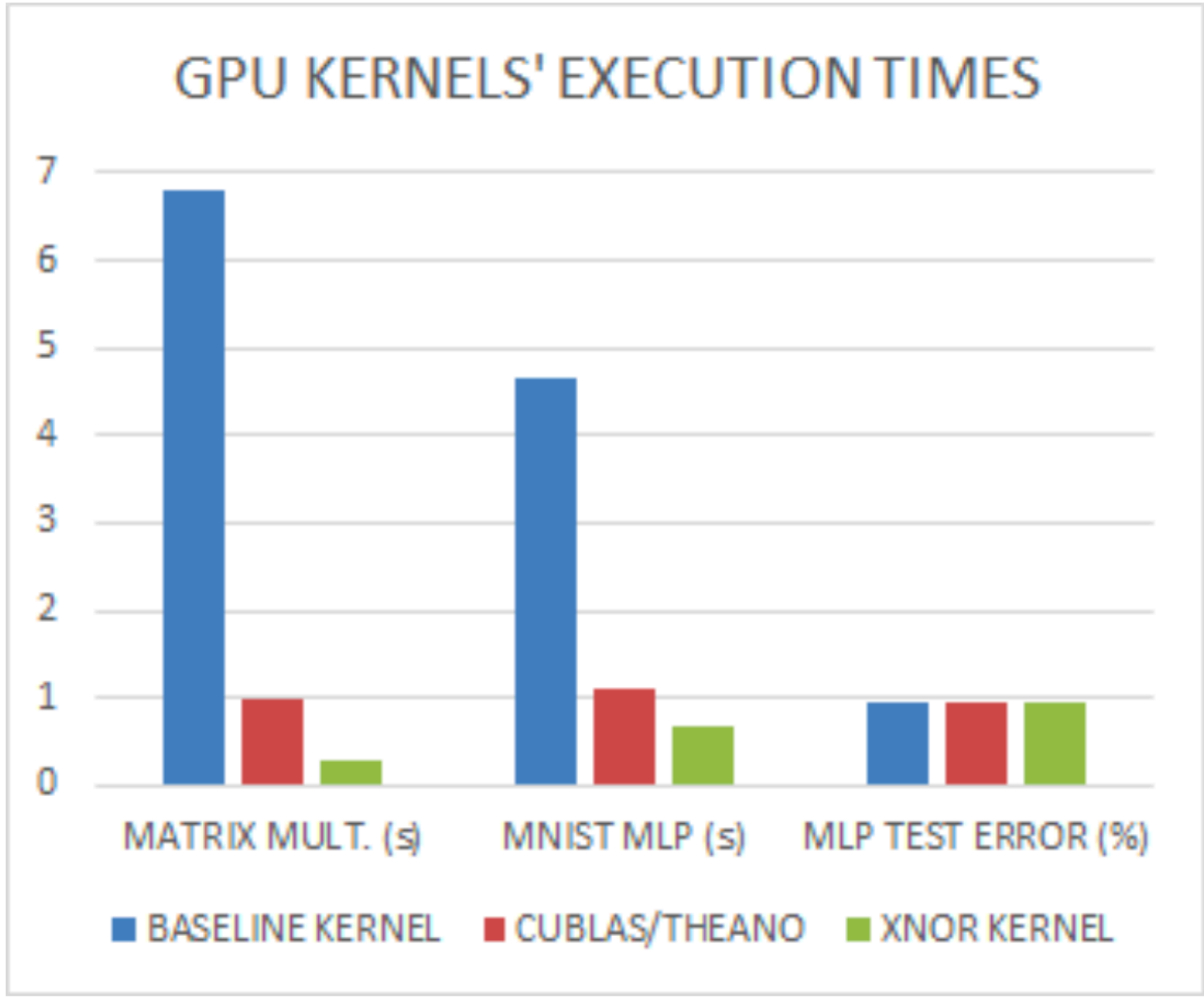
-(2*Popcount(a^w)-5) = -1


$\vec{s}_x$
+1
-1
-1
+1
$\vec{s}_w$
-1
+1
-1
-1
Multiply
-1
-1
+1
-1
Sum
-2
$\vec{b}_x$
1
0
0
1
$\vec{b}_w$
0
1
0
0
XNOR
0
0
1
0
popcount
(2p-N)
-2

Experiments: CIFAR-10 Training



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%

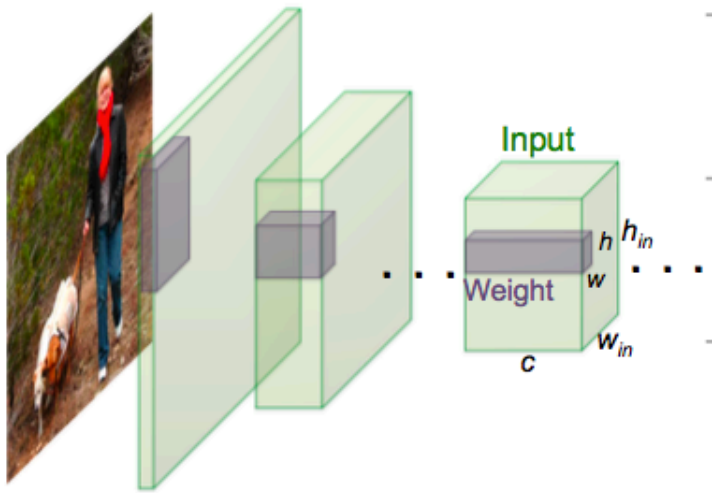

GPU KERNELS' EXECUTION TIMES
7
6
5
4
3
2
1
0
MATRIX MULT. (s)
MNIST MLP (s)
MLP TEST ERROR (%)
BASELINE KERNEL
CUBLAS/THEANO
XNOR KERNEL

Summary

- 1. “fast”
- 2. low memory, light network
- 3. mobile

- 1. “slow”
- 2. lower accuracy
- 3. bottleneck ： gradient still needs real high precision

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input **(XNOR-Net)**	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

$$\mathbf{W} \approx \alpha \mathbf{B} \quad \alpha \in \mathbb{R}^+$$
$$\mathbf{B} \in \{+1, -1\}^{c \times w \times h}$$

$$\mathbf{I} * \mathbf{W} \approx (\mathbf{I} \oplus \mathbf{B})\, \alpha$$

$\oplus$ indicates a convolution without any multiplication

$$J(\mathbf{B}, \alpha) = \|\mathbf{W} - \alpha \mathbf{B}\|^2$$
$$\alpha^*, \mathbf{B}^* = \underset{\alpha, \mathbf{B}}{\operatorname{argmin}} J(\mathbf{B}, \alpha)$$

$$J(\mathbf{B}, \alpha) = \alpha^2 \mathbf{B}^\mathsf{T} \mathbf{B} - 2\alpha \mathbf{W}^\mathsf{T} \mathbf{B} + \mathbf{W}^\mathsf{T} \mathbf{W}$$

$$J(\mathbf{B}, \alpha) = \alpha^2 n - 2\alpha \mathbf{W}^\mathsf{T} \mathbf{B} + \mathbf{c}$$

$$\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmax}} \{\mathbf{W}^\mathsf{T} \mathbf{B}\} \quad s.t. \ \mathbf{B} \in \{+1, -1\}^n$$

vector

$$\mathbf{B}^* = \operatorname{sign}(\mathbf{W})$$

$$\alpha^* = \frac{\mathbf{W}^\mathsf{T} \operatorname{sign}(\mathbf{W})}{n} = \frac{\sum |\mathbf{W}_i|}{n} = \frac{1}{n} \|\mathbf{W}\|_{\ell 1}$$

$\mathbf{B} \in \{+1, -1\}^n$,

$\mathbf{B}^\mathsf{T} \mathbf{B} = n$

$\mathbf{c} = \mathbf{W}^\mathsf{T} \mathbf{W}$

$$\alpha^* = \frac{\mathbf{W}^\mathsf{T} \mathbf{B}^*}{n}$$

$$\mathbf{X}^\top \mathbf{W} \approx \beta \mathbf{H}^\top \alpha \mathbf{B}, \text{ where } \mathbf{H}, \mathbf{B} \in \{+1, -1\}^n \text{ and } \beta, \alpha \in \mathbb{R}^+$$

$$\alpha^*, \mathbf{B}^*, \beta^*, \mathbf{H}* = \underset{\alpha, \mathbf{B}, \beta, \mathbf{H}}{\operatorname{argmin}} \left\| \mathbf{X} \odot \mathbf{W} - \beta \alpha \mathbf{H} \odot \mathbf{B} \right\|$$

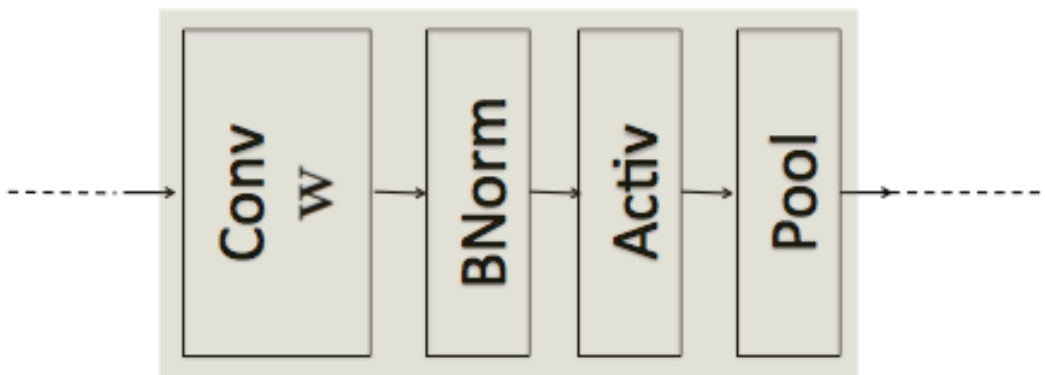
$$\gamma^*, \mathbf{C}^* = \underset{\gamma, \mathbf{C}}{\operatorname{argmin}} \left\| \mathbf{Y} - \gamma \mathbf{C} \right\|$$

$$\mathbf{C}^* = \operatorname{sign}(\mathbf{Y}) = \operatorname{sign}(\mathbf{X}) \odot \operatorname{sign}(\mathbf{W}) = \mathbf{H}^* \odot \mathbf{B}^*$$

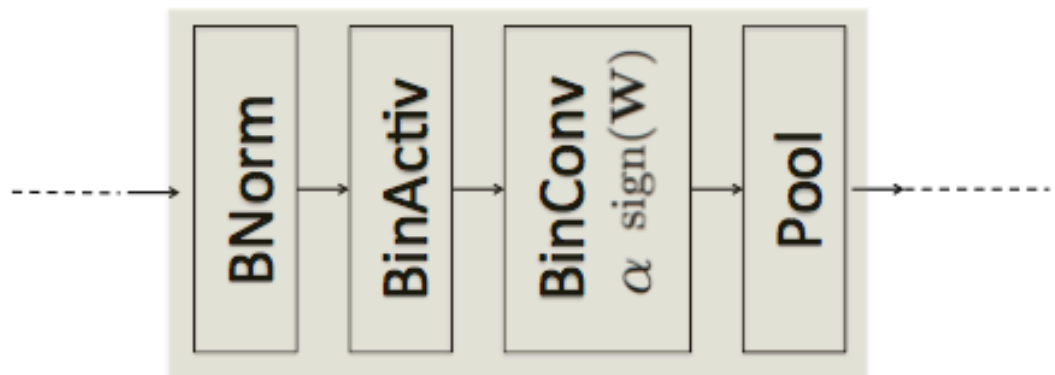
$$\gamma^* = \frac{\sum |\mathbf{Y}_i|}{n} = \frac{\sum |\mathbf{X}_i||\mathbf{W}_i|}{n} \approx \left(\frac{1}{n} \|\mathbf{X}\|_{\ell 1} \right) \left(\frac{1}{n} \|\mathbf{W}\|_{\ell 1} \right) = \beta^* \alpha^*$$

$$\mathbf{E}\left[|\mathbf{Y}_i|\right] = \mathbf{E}\left[|\mathbf{X}_i||\mathbf{W}_i|\right] = \mathbf{E}\left[|\mathbf{X}_i|\right] \mathbf{E}\left[|\mathbf{W}_i|\right]$$

$$\mathbf{I} * \mathbf{W} \approx (\text{sign}(\mathbf{I}) \circledast \text{sign}(\mathbf{W})) \odot \mathbf{K}\alpha$$

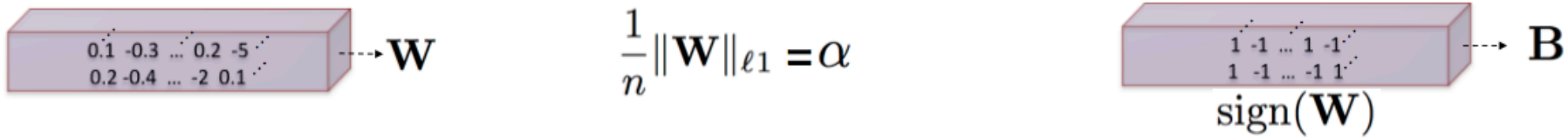


A typical block in CNN



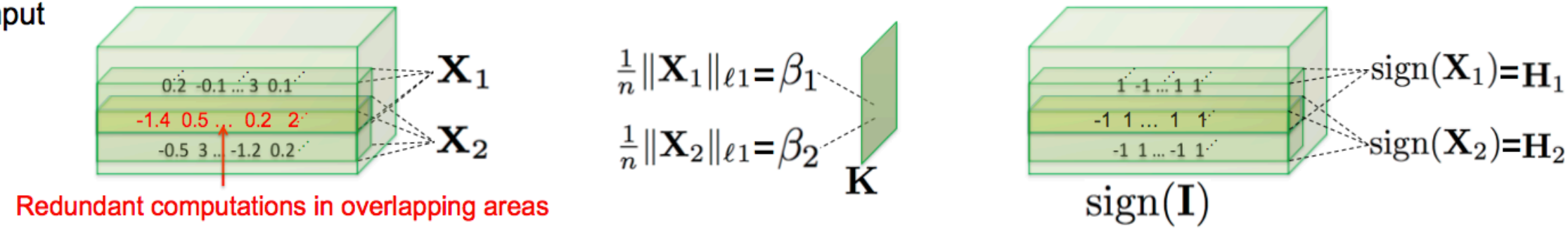
A block in XNOR-Net

(1) Binarizing Weight



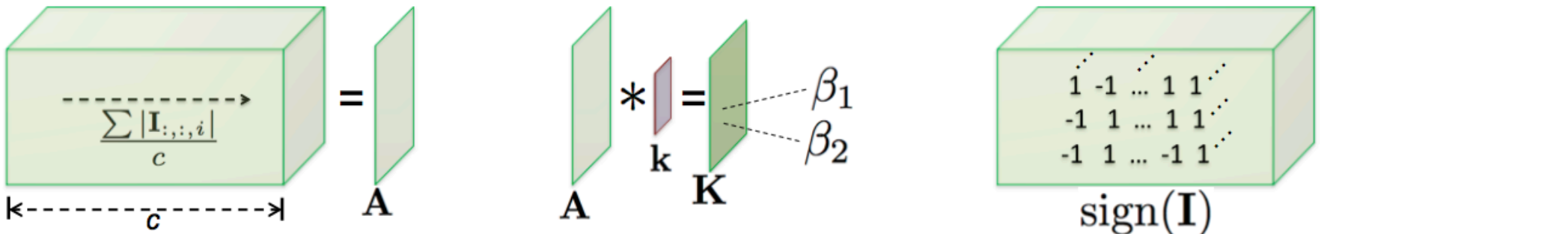
(2) Binarizing Input

Inefficient

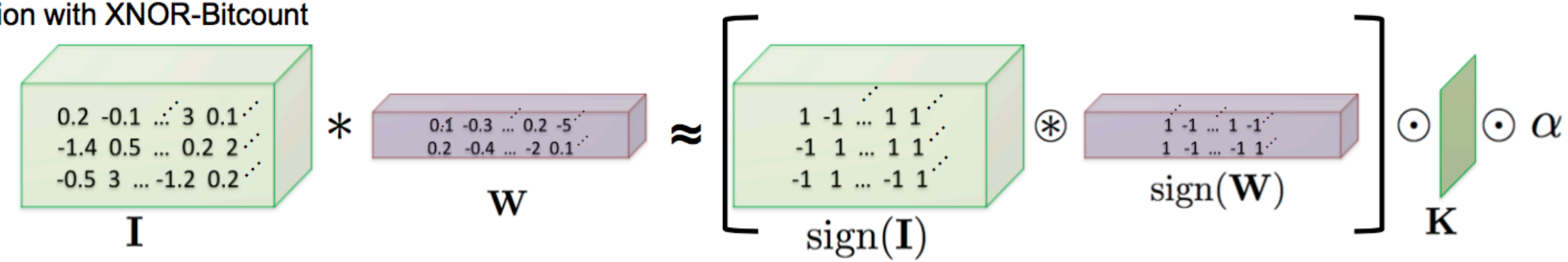


(3) Binarizing Input

Efficient



(4) Convolution with XNOR-Bitcount



Input

$$\bar{\bar{y}}_1^{\ell-1} = \begin{bmatrix} 3 & 2 & 1 \\ 0 & -1 & -2 \\ -3 & 1 & -1 \end{bmatrix}, \quad \bar{\bar{w}}_{1,1}^{\ell} = \begin{bmatrix} 1 & 1 \\ -1 & -2 \end{bmatrix}$$

$$\bar{\bar{y}}_2^{\ell-1} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ -3 & 1 & -1 \end{bmatrix}, \quad \bar{\bar{w}}_{2,1}^{\ell} = \begin{bmatrix} -1 & -1 \\ 2 & 1 \end{bmatrix}, \quad b_1^{\ell} = 1$$

Convolution computation

$$\bar{\bar{y}}_1^{\ell-1} * \bar{\bar{w}}_{1,1}^{\ell} = \begin{bmatrix} 7 & 8 \\ 0 & -2 \end{bmatrix}, \quad \bar{\bar{y}}_2^{\ell-1} * \bar{\bar{w}}_{2,1}^{\ell} = \begin{bmatrix} -3 & -9 \\ -4 & 4 \end{bmatrix}$$

$$\bar{\bar{s}}_n^{\ell} = \bar{\bar{y}}_1^{\ell-1} * \bar{\bar{w}}_{1,1}^{\ell} + \bar{\bar{y}}_2^{\ell-1} * \bar{\bar{w}}_{2,1}^{\ell} + b_n^{\ell} = \begin{bmatrix} 5 & 0 \\ -3 & 3 \end{bmatrix}$$

BWN Convolution

$$\bar{y}_1^{\ell-1} = \begin{bmatrix} 3 & 2 & 1 \\ 0 & -1 & -2 \\ -3 & 1 & -1 \end{bmatrix}, \quad \bar{w}_{1,1}^{\ell} = \begin{bmatrix} 1 & 1 \\ -1 & -2 \end{bmatrix}$$

$$\bar{y}_2^{\ell-1} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ -3 & 1 & -1 \end{bmatrix}, \quad \bar{w}_{2,1}^{\ell} = \begin{bmatrix} -1 & -1 \\ 2 & 1 \end{bmatrix}, \quad b_1^{\ell} = 1$$

$$\alpha_1^{\ell} = \frac{\sum_{i,j,m \in V_1^{\ell}} |\bar{w}_{m,1}^{\ell}|}{\mathrm{Ch}_1^{\ell} \times w_{\ell} \times h_{\ell}}$$

$$= \frac{1+1+1+2+1+1+1+2+1}{8} = \frac{5}{4}$$

$$\bar{B}_{1,1}^{\ell} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, \quad \bar{B}_{2,1}^{\ell} = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$$

$$\bar{s}_n^{\ell} = \alpha_1^{\ell}(\bar{y}_1^{\ell-1} \otimes \bar{B}_{1,1} + \bar{y}_2^{\ell-1} \otimes \bar{B}_{2,1}^{\ell}) + b_1^{\ell}$$

$$= \frac{5}{4}\left(\begin{bmatrix} 6 & 4 \\ 1 & -3 \end{bmatrix} + \begin{bmatrix} -4 & -8 \\ -1 & 3 \end{bmatrix}\right) + b_1^{\ell} = \begin{bmatrix} \frac{7}{2} & \frac{-3}{2} \\ 1 & 1 \end{bmatrix}$$

XNOR Convolution

$$\bar{y}_1^{\ell-1} = \begin{bmatrix} 3 & 2 & 1 \\ 0 & -1 & -2 \\ -3 & 1 & -1 \end{bmatrix}, \quad \bar{w}_{1,1}^{\ell} = \begin{bmatrix} 1 & 1 \\ -1 & -2 \end{bmatrix}$$

$$\bar{y}_2^{\ell-1} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ -3 & 1 & -1 \end{bmatrix}, \quad \bar{w}_{2,1}^{\ell} = \begin{bmatrix} -1 & -1 \\ 2 & 1 \end{bmatrix}, \quad b_1^{\ell} = 1$$

$$\bar{y}_{c,1}^{\ell-1} = \bar{y}_1^{\ell-1} + \bar{y}_2^{\ell-1} = \begin{bmatrix} 4 & 4 & 4 \\ 0 & -2 & -4 \\ -6 & 2 & -2 \end{bmatrix}$$

$$\beta_1^{\ell-1} = \bar{y}_{c,1}^{\ell-1} \otimes \text{ones}(h_\ell, w_\ell) \times \frac{1}{\text{Ch}_1^{\ell} \times h_\ell \times w_\ell}$$

$$= \frac{1}{2 \times 2 \times 2} \begin{bmatrix} 6 & 2 \\ -6 & -6 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 3 & 1 \\ -3 & -3 \end{bmatrix}$$

$$H_1^{\ell-1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & -1 \end{bmatrix}, \quad H_2^{\ell-1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & -1 \end{bmatrix}$$

BWN

$$\alpha_1^{\ell} = \frac{\sum_{i,j,m \in V_1^{\ell}} |\bar{w}_{m,1}^{\ell}|}{\text{Ch}_1^{\ell} \times w_\ell \times h_\ell}$$

$$= \frac{1+1+1+2+1+1+1+2+1}{8} = \frac{5}{4}$$

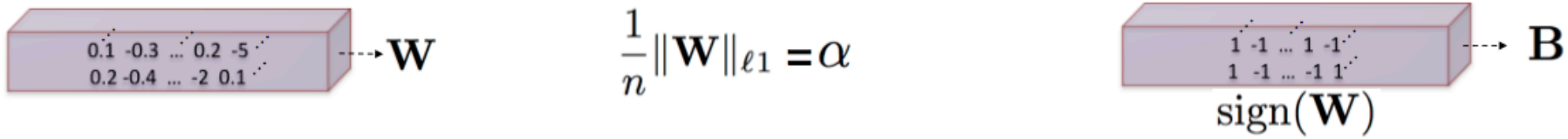
$$\bar{B}_{1,1}^{\ell} = \begin{vmatrix} 1 & 1 \\ -1 & -1 \end{vmatrix}, \quad \bar{B}_{2,1}^{\ell} = \begin{vmatrix} -1 & -1 \\ 1 & 1 \end{vmatrix}$$

$$\bar{s}_n^{\ell} = \alpha_1^{\ell} \times \beta_1^{\ell-1} (\bar{H}_1^{\ell-1} \otimes \bar{B}_{1,1}^{\ell} + \bar{H}_2^{\ell-1} \otimes \bar{B}_{2,1}^{\ell}) + b_1^{\ell}$$

$$= \frac{5}{4} \times \frac{1}{4} \begin{bmatrix} 3 & 1 \\ -3 & -3 \end{bmatrix} \odot \left(\begin{bmatrix} 2 & 4 \\ 0 & -2 \end{bmatrix} + \begin{bmatrix} -2 & -4 \\ 0 & 2 \end{bmatrix} \right) + 1$$

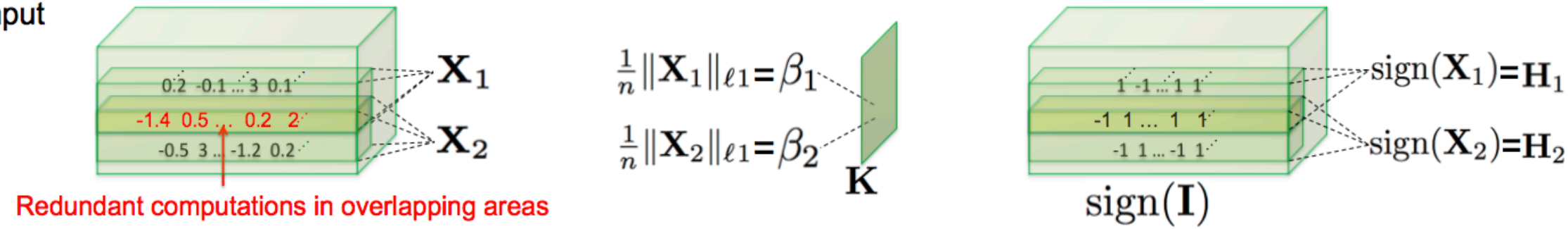
$$= \frac{5}{16} \begin{bmatrix} 3 & 1 \\ -3 & -3 \end{bmatrix} \odot \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + 1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

(1) Binarizing Weight



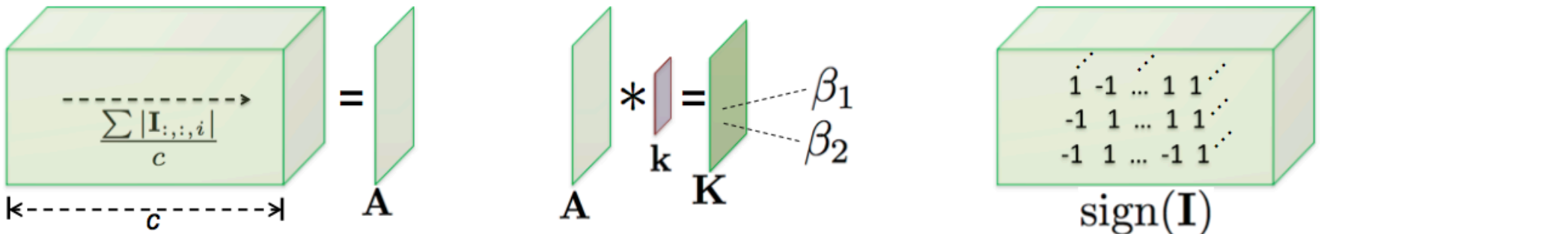
(2) Binarizing Input

Inefficient

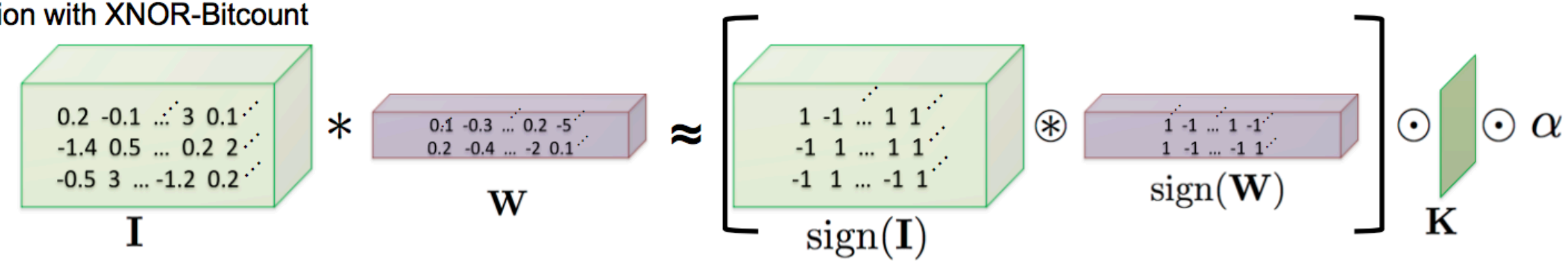


(3) Binarizing Input

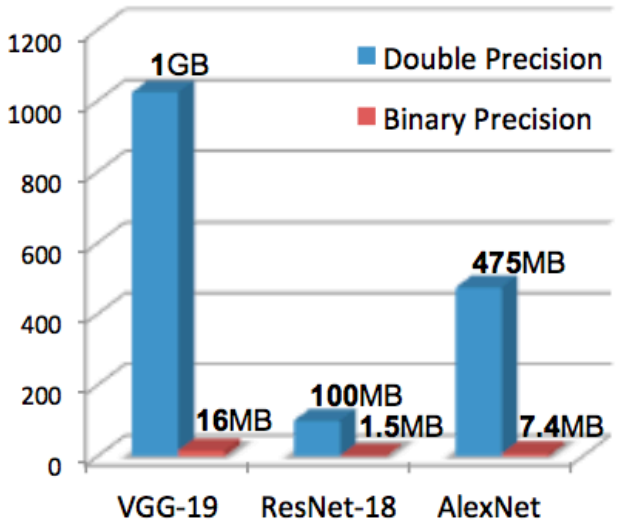
Efficient



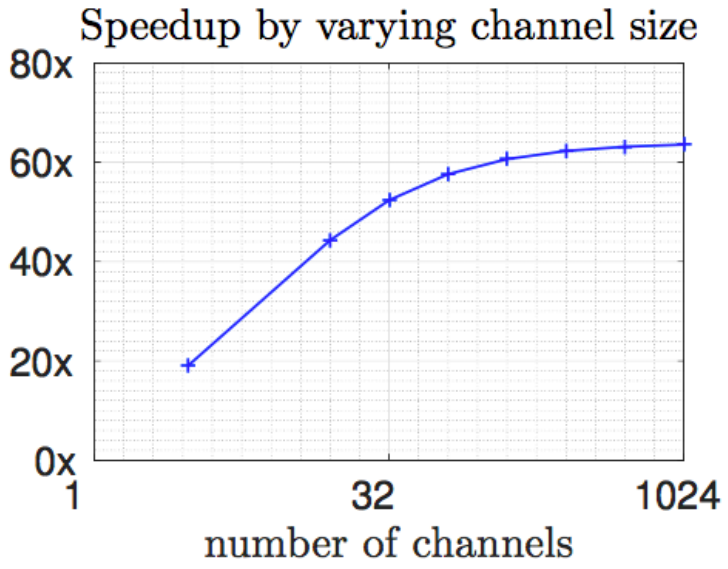
(4) Convolution with XNOR-Bitcount



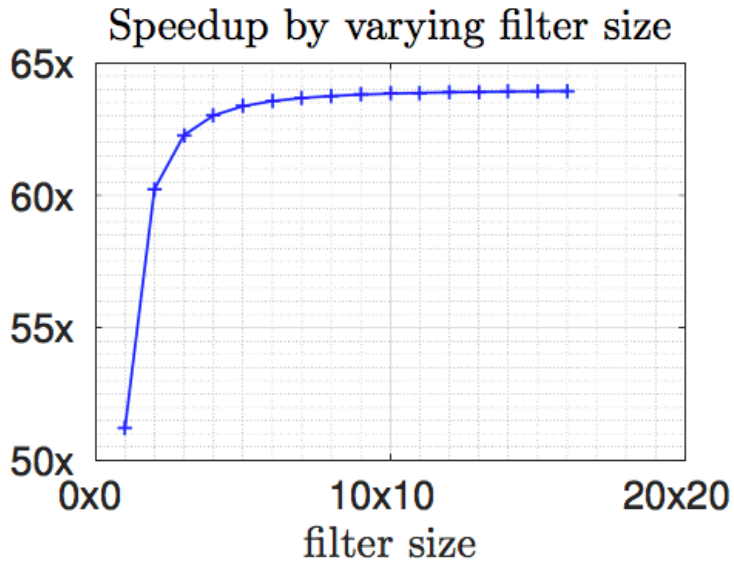
experiments



(a)



(b)



(c)

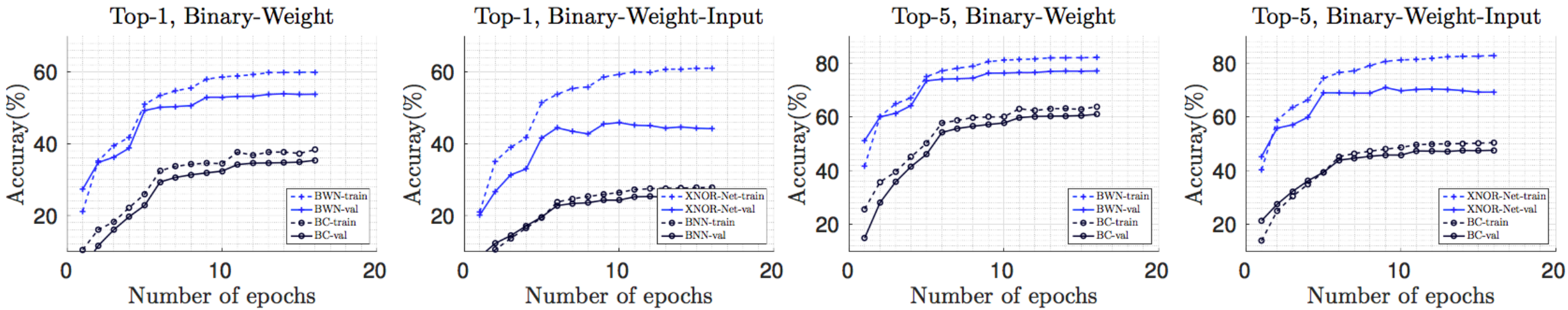
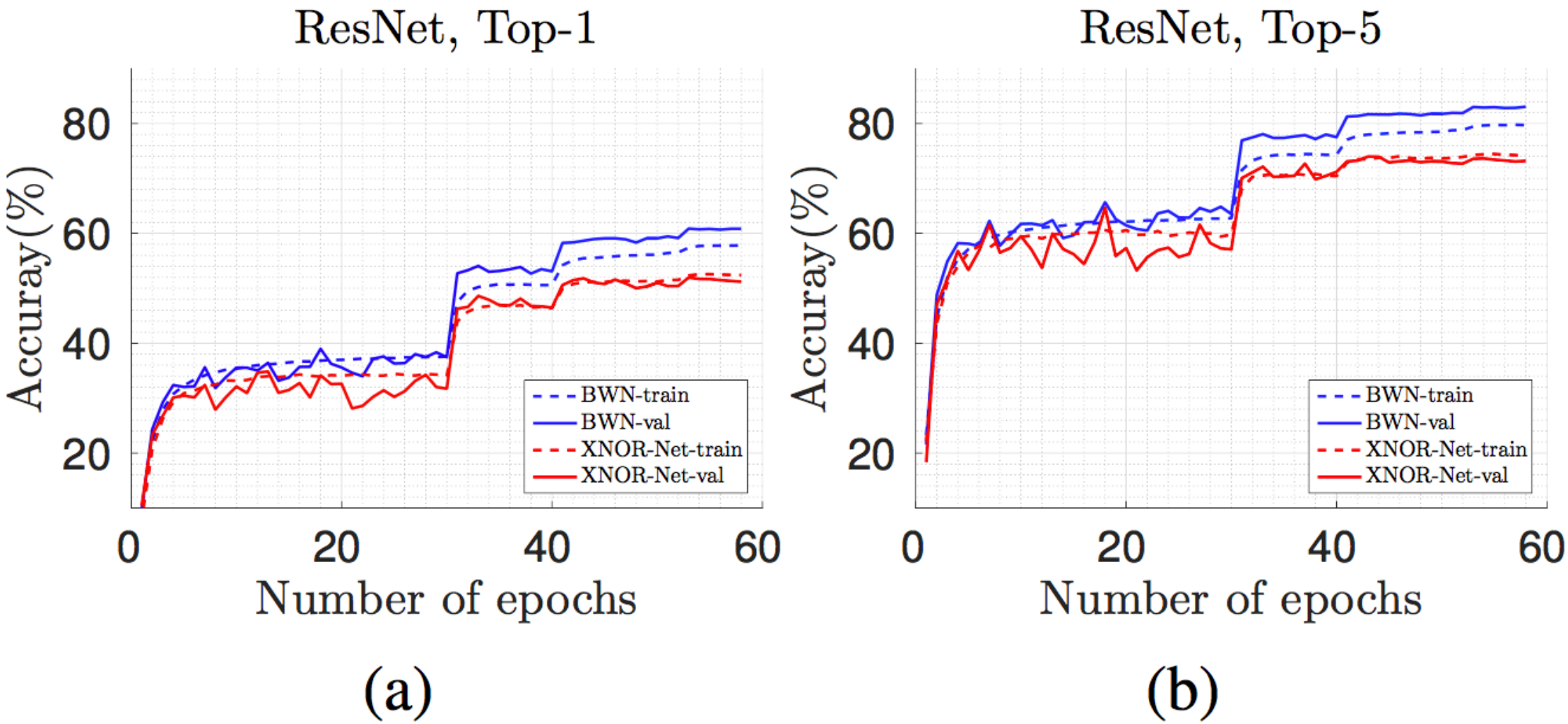


Fig. 5: This figure compares the imagenet classification accuracy on Top-1 and Top-5 across training epochs. Our approaches BWN and XNOR-Net outperform BinaryConnect(BC) and BinaryNet(BNN) in all the epochs by large margin(∼17%).

Classification Accuracy(%)										
Binary-Weight				Binary-Input-Binary-Weight				Full-Precision		
BWN		BC[11]		XNOR-Net		BNN[11]		AlexNet[1]		
Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	
56.8	**79.4**	35.4	61.0	**44.2**	**69.2**	27.9	50.42	56.6	80.2	


ResNet, Top-1
Accuray(%)
80
60
40
20
0
20
40
60
Number of epochs
BWN-train
BWN-val
XNOR-Net-train
XNOR-Net-val
(a)
ResNet, Top-5
Accuray(%)
80
60
40
20
0
20
40
60
Number of epochs
BWN-train
BWN-val
XNOR-Net-train
XNOR-Net-val
(b)

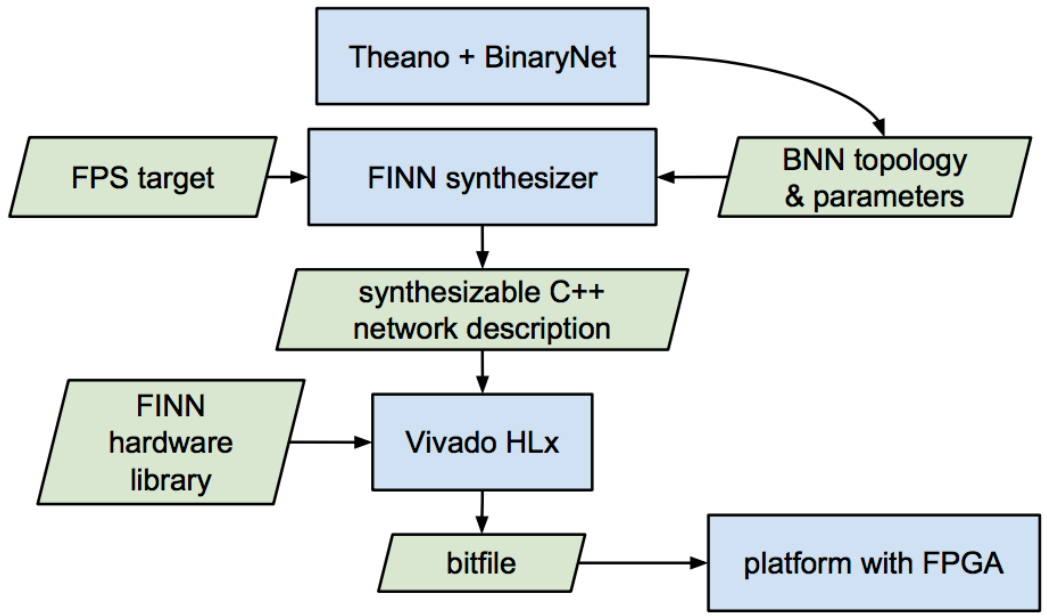
	ResNet-18		GoogLenet	
Network Variations	top-1	top-5	top-1	top-5
Binary-Weight-Network	60.8	83.0	65.5	86.1
XNOR-Network	51.2	73.2	N/A	N/A
Full-Precision-Network	69.3	89.2	71.3	90.0

Binary-Weight-Network		
Strategy for computing α	top-1	top-5
Using equation 6	56.8	79.4
Using a separate layer	46.2	69.5

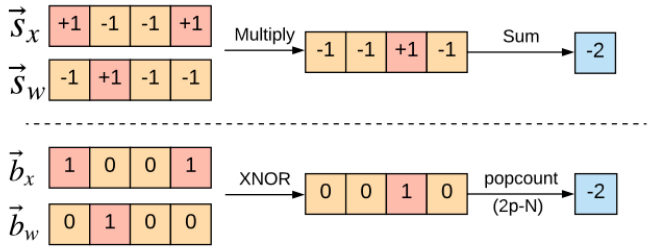
XNOR-Network		
Block Structure	top-1	top-5
C-B-A-P	30.3	57.5
B-A-C-P	44.2	69.2

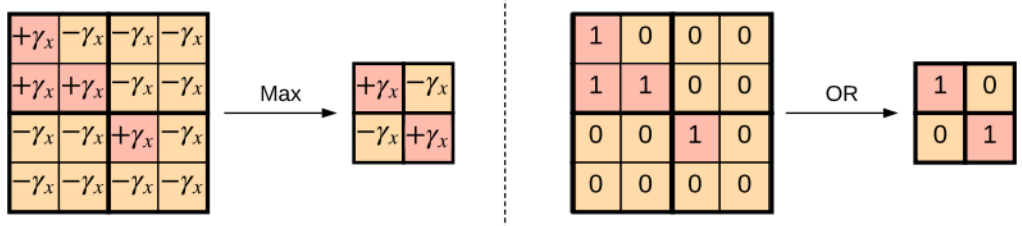
FINN: A Framework for Fast, Scalable Binarized Neural Network Inference

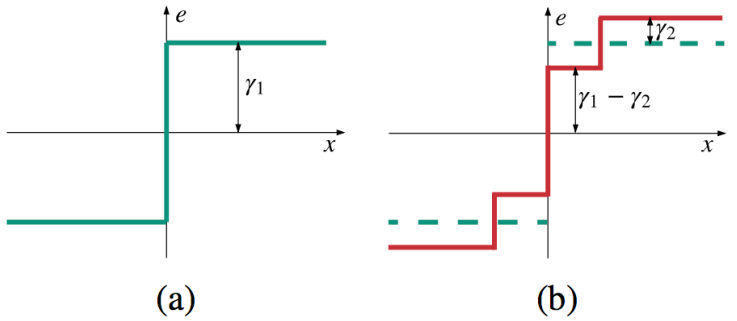
Yaman Umuroglu[*†], Nicholas J. Fraser[*‡], Giulio Gambardella[*], Michaela Blott[*],
Philip Leong[‡], Magnus Jahre[†] and Kees Vissers[*]
[*]Xilinx Research Labs; [†]Norwegian University of Science and Technology; [‡]University of Sydney
yamanu@idi.ntnu.no

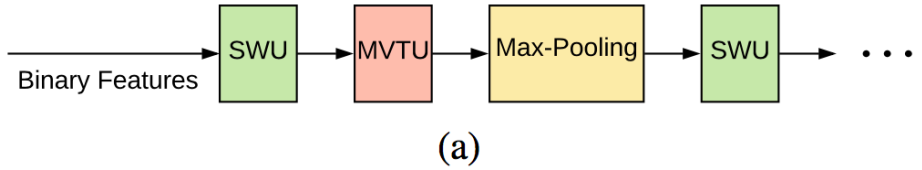


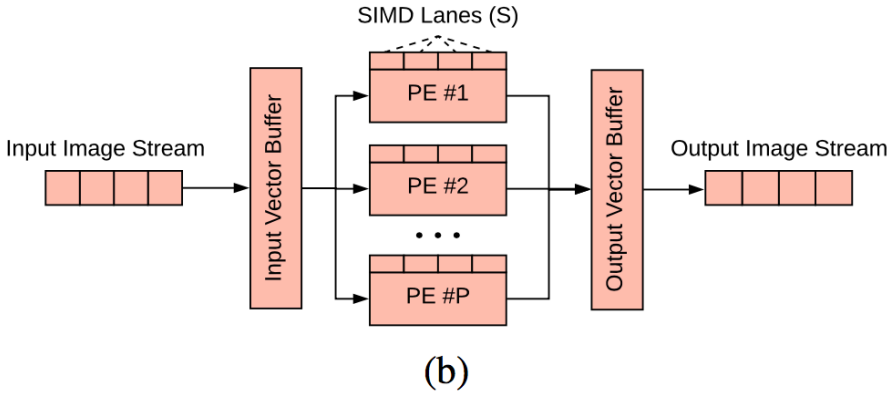
ReBNet: Residual Binarized Neural Network

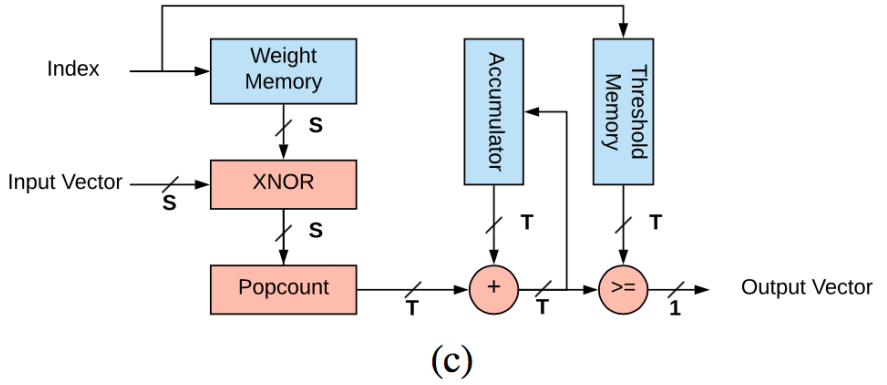




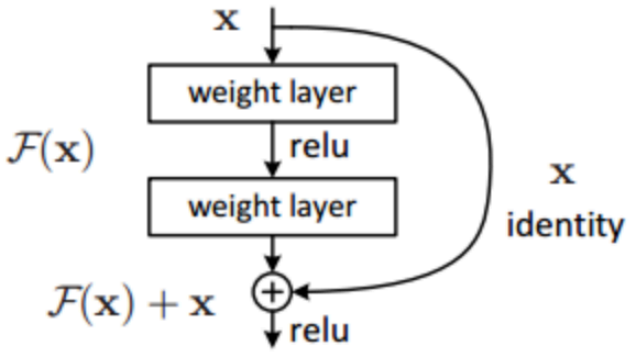








F'(5)=5.1 H(5)=5.1 H(5)=F(5)+5, F(5)=0.1

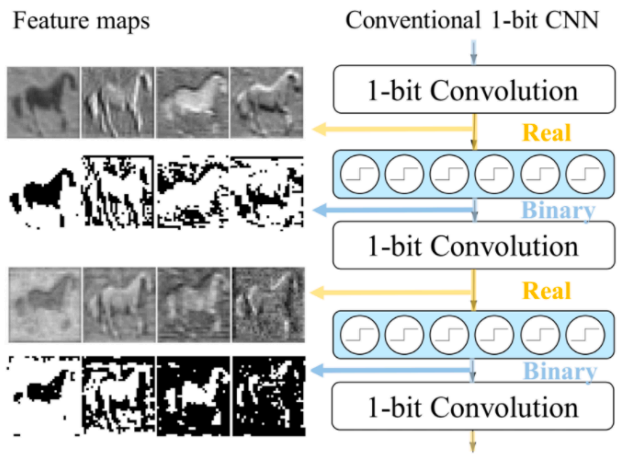


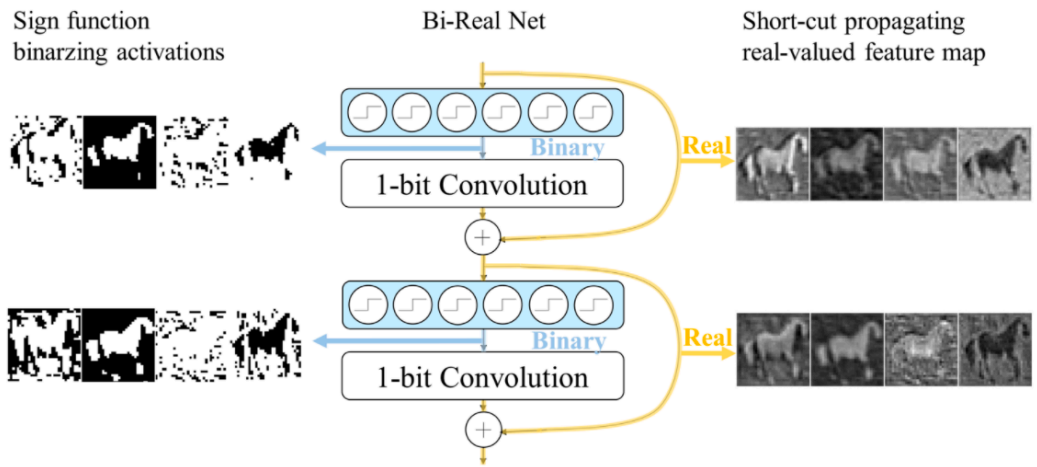
1/51=2% (0.2-0.1)/0.1 = 1

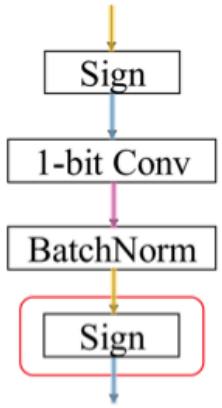
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

			MNIST	CIFAR10	SVHN	Imagenet
FINN		Accuracy (%)	95.83	80.1	94.9	27.9
		Throughput (samples/sec)	6.3×10^5	6×10^3	6×10^3	5.2×10^2
Ours	M=1	Accuracy (%)	97.87	80.59	95.46	40.89
		Throughput (samples/sec)	6.4×10^5	6×10^3	6×10^3	5.3×10^2
	M=2	Accuracy (%)	98.29	85.94	96.82	41.37
		Throughput (samples/sec)	3.3×10^5	3×10^3	3×10^3	2.6×10^2
	M=3	Accuracy (%)	98.25	86.98	97.00	41.43
		Throughput (samples/sec)	2.3×10^5	2×10^3	2×10^3	1.7×10^2

Bi-Real Net: Enhancing the Performance of 1-bit CNNs With Improved Representational Capability and Advanced Training Algorithm

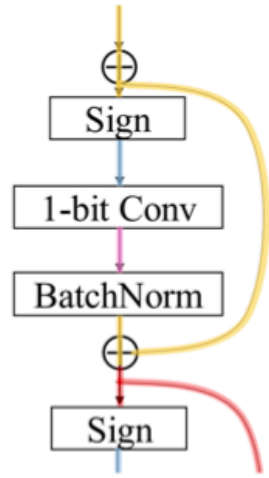






	Map size	Value range	Different value	$\mathbb{R}$
A_b^l	14x14x32	{-1, 1}	2	$2^{14x14x32}$
A_m^l	14x14x32	{-288,-286,...,288}	289	$289^{14x14x32}$
A_r^{l+1}	14x14x32	{a certain range}	289	$289^{14x14x32}$
A_b^{l+1}	14x14x32	{-1, 1}	2	$2^{14x14x32}$

(a)



	Map size	Value range	Different value	$\mathbb{R}$
A_r^l	14x14x32	{a certain range}	289	$289^{14x14x32}$
A_b^l	14x14x32	{-1, 1}	2	$2^{14x14x32}$
A_m^l	14x14x32	{-288,-286,...,288}	289	$289^{14x14x32}$
A_r^{l+1}	14x14x32	{a certain range}	289	$289^{14x14x32}$
A_{add}^{l+1}	14x14x32	{a certain range}	289*289	$289^{2x14x14x32}$

(b)

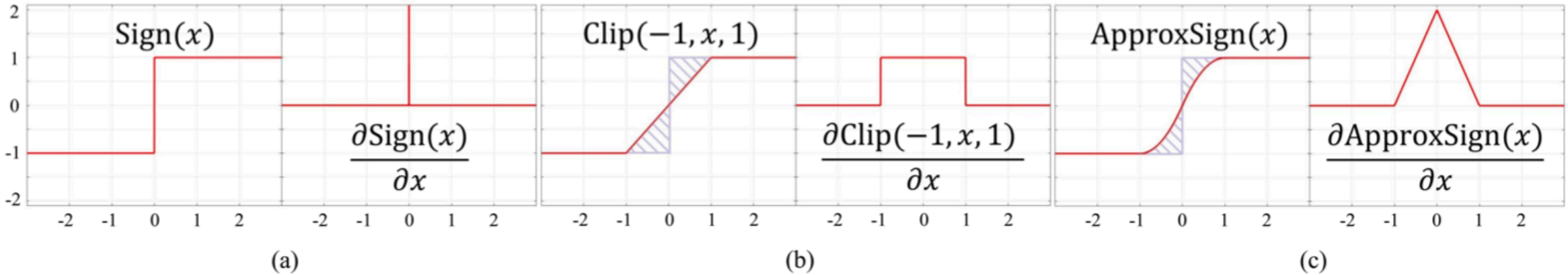


Fig. 5. (a) Sign function and its derivative, (b) Clip function and its derivative for approximating the derivative of the sign function, proposed in [7], (c) Proposed differentiable piecewise polynomial function and its triangle-shaped derivative for approximating the derivative of the sign function in gradients computation.

$$\mathbf{x} \cdot \mathbf{y} = N - 2 \times \text{bitcount}(\text{xnor}(\mathbf{x}, \mathbf{y})), x_i, y_i \in \{-1, 1\} \forall i.$$

DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients

Binarized LSTM Language Model