AdderNet:
Do We Really Need Multiplications in Deep Learning?

Chen et al., CVPR 2020

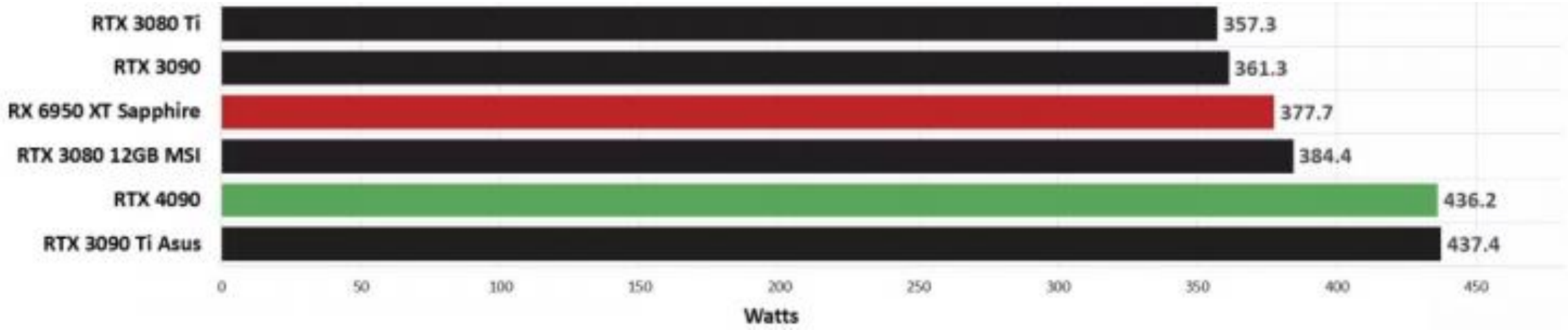
권순기

목차

1. Introduction

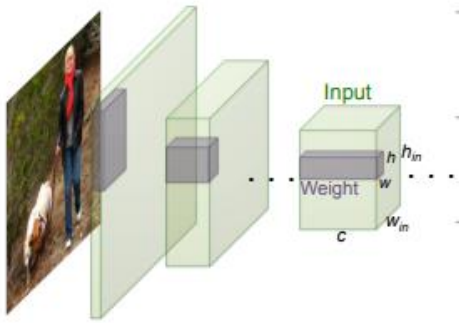
- GPU의 발달로 수십억개의 부동소수점 연산(floating point computation)을 하는 CNN network가 발전함
- GPU의 경우 많은 power가 필요해 mobile device에서 사용하기 힘들고 크기도 매우 큼

=> 모바일 기기에서 사용할 수 있는 **효율적인 Deep Neural Network**에 대한 연구가 필요



1. Introduction

- 일반적으로 multiplication이 addition에 비해 느림
- 그러나, Deep Neural Network에서는 forward inference시 float-valued weights와 float-valued activations간 multiplication이 수행됨
- floating-point 연산, multiplication 연산을 줄이기 위한 연구들이 많이 진행됨
  - Binary Connect
  - Binarized Neural Networks
  - XNOR-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

1. Introduction

- Binarizing filters를 사용하는 것은 computational cost를 획기적으로 줄였지만, 원래의 accuracy는 보장하지 못함
- Binary Network는 학습이 불안정하고 작은 learning rate의 사용으로 천천히 수렴함
- CNN은 input과 convolution filter간 similarity를 측정하는 것

  ⇒ 이 similarity measure를 바꾸기 위한 시도는 없었음

따라서, CNN의 multiplication 연산을 addition 연산으로 바꾸고자 함

2. Related works

- CNN의 computational complexity를 줄이기 위한 연구

1. Pruning based methods
   - remove redundant weights to compress model
   - Deep Compression

2. Designing novel blocks or operations
   - Squeezenet
   - MobileNet
   - ShuffleNet
   - SENet

3. Knowledge distillation

<u>기존 연구들은 여전히 multiplication 연산을 사용</u>

3. Networks without Multiplication

$$F \in \mathbb{R}^{d \times d \times c_{in} \times c_{out}}$$

$$X \in \mathbb{R}^{H \times W \times c_{in}}$$

$$Y(m,n,t) = \sum_{i=0}^{d} \sum_{j=0}^{d} \sum_{k=0}^{c_{in}} S\big(X(m+i, n+j, k), F(i,j,k,t)\big), \tag{1}$$

✓ Y : Output Feature(Filter와 Input Feature간 similarity)

✓ $S(\cdot,\cdot)$: pre-defined similarity measure

- cross-correlation : $S(x,y) = x \times y$
- d=1 : fully-connected layer

Input Feature와 Filter간 similarity를 어떻게 정의할것인가?

3. Networks without Multiplication

3.1. Adder Networks

- similarity measure : L1-norm

$$Y(m,n,t) = -\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} |X(m+i,n+j,k) - F(i,j,k,t)|. \tag{2}$$

✓ Conv Filter의 출력은 양수 또는 음수

✓ Adder Filter의 출력은 항상 음수

=> Batch Normalization을 사용하여 출력을 normalize하여 기존의 활성화 함수를 사용할 수 있도록 함

(Batch Normalization엔 multiplication이 포함되지만 convolution에 비하면 무시할만 함)

✓ Computational cost

$O(d^2 c_{in} c_{out} HW) + O(c_{out} H'W')$ => ?? $+ O(c_{out} H'W')$

3. Networks without Multiplication

3.2. Optimization

- F에 대한 Y의 partial derivative
  - ➢ CNN

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = X(m+i,n+j,k), \qquad (3)$$

  - ➢ AdderNet

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = \mathrm{sgn}(X(m+i,n+j,k)-F(i,j,k,t)), \quad (4)$$

✓ (4)에서 gradient는 -1, 0, 1 값만 가지므로, signSGD*를 이용해 최적화를 해야함

✓ signSGD는 대부분의 경우 가장 가파른 기울기를 가지는 방향을 취하지 않으며 이는 차원이 커질수록 더 심해지므로 수많은 파라미터를 가진 네트워크를 학습하기 어려움

*signSGD : SGD 각각의 배치에 대해 gradient를 업데이트할 때 full precision의 gradient 대신 gradient의 부호(sign)를 활용하는 최적화 알고리즘

3. Networks without Multiplication

3.2. Optimization

- Derivative of L2-norm

$$\frac{\partial Y(m,n,t)}{\partial F(i,j,k,t)} = X(m+i,n+j,k) - F(i,j,k,t), \quad (5)$$

✓ L2-norm의 derivative의 signSGD update는 Eq.(4)와 같음

✓ 따라서, Eq.(5)에 따라 full-precision gradient를 활용하여 F와 X의 gradient를 정확하게 업데이트 가능

3. Networks without Multiplication

3.2. Optimization

- Full-precision gradient를 사용하면 gradient가 [-1, 1]의 범위를 넘어 gradient exploding이 발생 가능
  - ✓ 레이어 i의 필터와 입력을 F_i, X_i라 하면, F_i의 기울기에만 영향을 주는 $\frac{\partial Y}{\partial F_i}$와 달리 $\frac{\partial Y}{\partial X_i}$는 chain rule에 의해 i 이전 레이어의 기울기에도 영향을 주기 때문
  - ✓ 따라서, <u>X의 기울기에 대해 gradient clipping을 사용</u>

$$\frac{\partial Y(m,n,t)}{\partial X(m+i,n+j,k)} = \mathrm{HT}(F(i,j,k,t) - X(m+i,n+j,k)). \tag{6}$$

where $\mathrm{HT}(\cdot)$ denotes the HardTanh function:

$$\mathrm{HT}(x) = \begin{cases} x & \text{if} \quad -1 < x < 1, \\ 1 & x > 1, \\ -1 & x < -1. \end{cases} \tag{7}$$

3. Networks without Multiplication

3.3. Adaptive Learning Rate Scaling

- F와 X가 i.i.d Normal distribution 이라 가정

$$Var[Y_{CNN}] = \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} Var[X \times F] \\ = d^2 c_{in} Var[X] Var[F]. \quad (8)$$

$$Var[Y_{AdderNet}] = \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} Var[|X - F|] \\ = (1 - \frac{2}{\pi}) d^2 c_{in} (Var[X] + Var[F]), \quad (9)$$

✓ 일반적으로 $Var[F]$는 CNN에서 매우 작음(10^{-3} or 10^{-4})

✓ 따라서, CNN보다 AdderNet에서 $Var[Y]$가 더 큼

3. Networks without Multiplication

3.3. Adaptive Learning Rate Scaling

- AdderNet의 output이 가지는 큰 분산의 효과

  ✓ mini-batch $B = \{x_1, \cdots, x_m\}$에 대해 batch normalization layer는 다음과 같음

$$y = \gamma \frac{x - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}} + \beta, \qquad (10)$$

  ✓ mini-batch에서 x에 대한 $loss$의 기울기는 다음과 같음

$$\frac{\partial \ell}{\partial x_i} = \sum_{j=1}^{m} \frac{\gamma}{m^2 \sigma_{\mathcal{B}}} \left\{ \frac{\partial \ell}{\partial y_i} - \frac{\partial \ell}{\partial y_j} [1 + \frac{(x_i - x_j)(x_j - \mu_{\mathcal{B}})}{\sigma_{\mathcal{B}}}] \right\}. \qquad (11)$$

  ✓ $Var[Y] = \sigma_B$가 크다면 x에 대한 기울기의 크기는 CNN보다 AdderNet이 훨씬 작음

  ✓ 결국, chain rule에 의해 F의 gradient 크기가 감소할 것

3. Networks without Multiplication

3.3. Adaptive Learning Rate Scaling

- LeNet-5-BN with MNIST -> F의 gradient의 L2-norm

Table 1. The ℓ_2-norm of gradient of weight in each layer using different networks at 1st iteration.

Model	Layer 1	Layer 2	Layer 3
AdderNet	0.0009	0.0012	0.0146
CNN	0.2261	0.2990	0.4646

✓ AdderNet의 gradient가 CNN보다 작음

✓ 이로 인해 update 속도가 느려짐

✓ 단순히 AdderNet에서 높은 learning rate를 사용하면 해결될 것 같지만,
layer마다 gradient의 norm값이 다름

=> Layer 마다 adaptive learning rate 적용

3. Networks without Multiplication

3.3. Adaptive Learning Rate Scaling

- Adaptive Learning Rate

$$\Delta F_l = \gamma \times \alpha_l \times \Delta L(F_l), \tag{12}$$

$$\alpha_l = \frac{\eta\sqrt{k}}{\|\Delta L(F_l)\|_2}, \tag{13}$$

- γ : global learning rate (adder layer, BN layer)
- $\Delta L(F_l)$: F의 gradient
- α_l : local learning rate (adder layer)
- k : F의 element 수
- η : adder filter의 learning rate를 제어하는 hyper-parameter

4. Experiment

4.1. MNIST

Method	#Add.	#Mul.	Accuracy	Latency
CNN	~435K	~435K	99.4%	~2.6M
AddNN	~870K	~0	99.4%	~1.7M

Table 4. The impact of parameter η using LeNet-5-BN on the MNIST dataset.

η	1	0.5	0.2	0.1	0.05
Acc. (%)	99.26	99.30	99.35	99.40	99.32

✓ η : 0.1 선택

4. Experiment

4.2. CIFAR

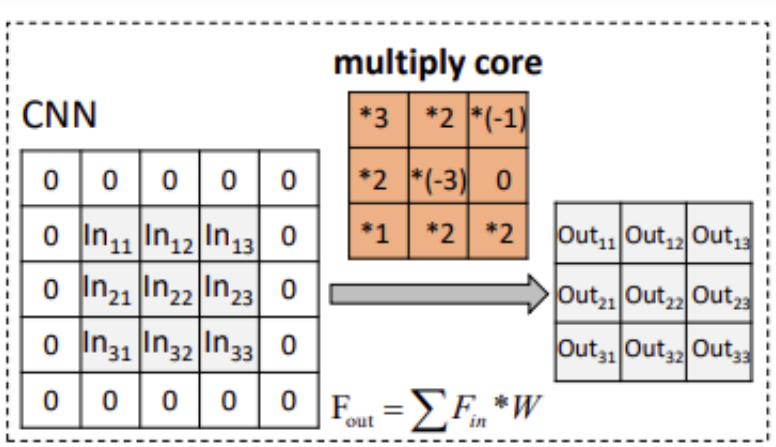
Table 2. Classification results on the CIFAR-10 and CIFAR-100 datasets.

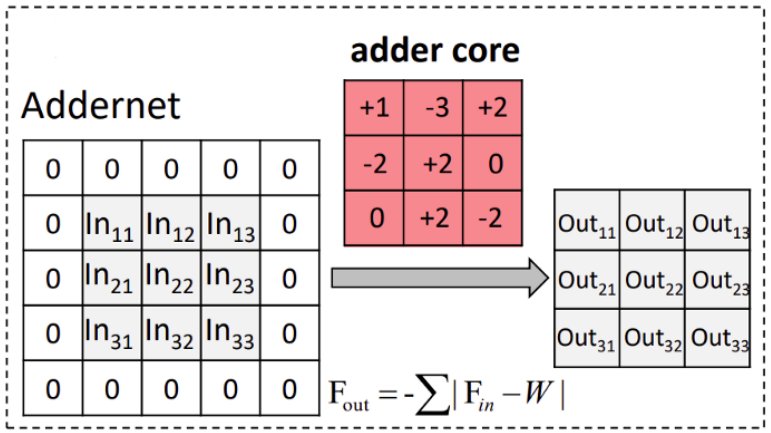
Model	Method	#Mul.	#Add.	XNOR	CIFAR-10	CIFAR-100
VGG-small	BNN	0	0.65G	0.65G	89.80%	65.41%
	AddNN	0	1.30G	0	93.72%	72.64%
	CNN	0.65G	0.65G	0	93.80%	72.73%
ResNet-20	BNN	0	41.17M	41.17M	84.87%	54.14%
	AddNN	0	82.34M	0	91.84%	67.60%
	CNN	41.17M	41.17M	0	92.25%	68.14%
ResNet-32	BNN	0	69.12M	69.12M	86.74%	56.21%
	AddNN	0	138.24M	0	93.01%	69.02%
	CNN	69.12M	69.12M	0	93.29%	69.74%

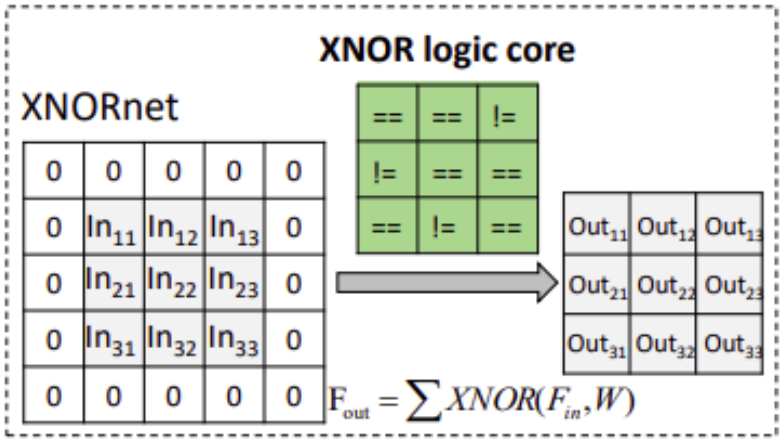
- ✓ First, last layer convolution은 유지
- ✓ Batch Nomalization Layer와 first, last layer의 convolution에 대한 multiplication cost는 다른 layer에 비해 작아 최종 FLOPs 계산에 포함 안함
- ✓ AdderNet이 CNN과 유사한 성능을 보였음

4. Experiment

4.2. CIFAR







4. Experiment

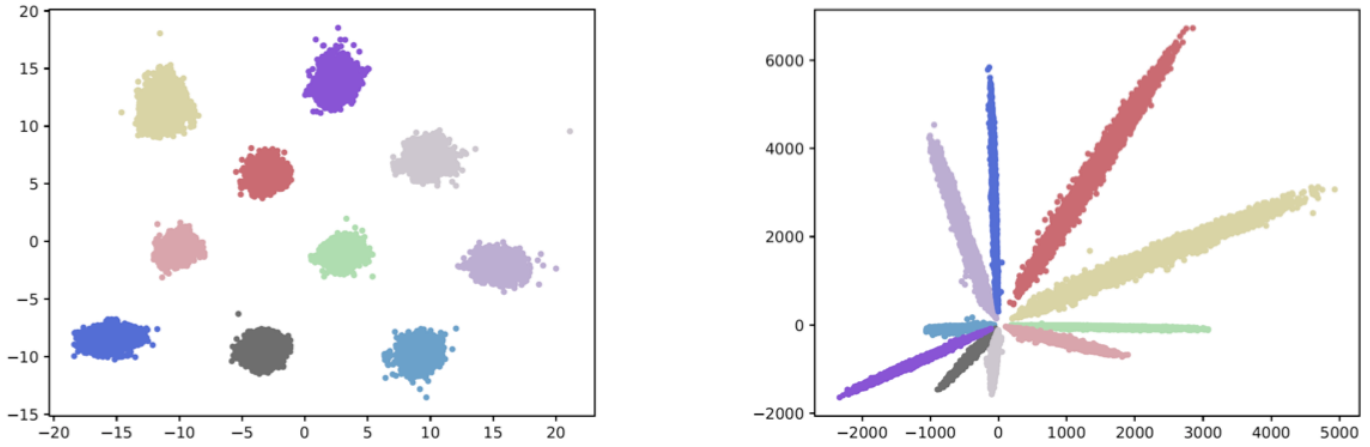
4.3. ImageNet

Table 3. Classification results on the ImageNet datasets.

Model	Method	#Mul.	#Add.	XNOR	Top-1 Acc.	Top-5 Acc.
ResNet-18	BNN	0	1.8G	1.8G	51.2%	73.2%
	AddNN	0	3.6G	0	67.0%	87.6%
	CNN	1.8G	1.8G	0	69.8%	89.1%
ResNet-50	BNN	0	3.9G	3.9G	55.8%	78.4%
	AddNN	0	7.7G	0	74.9%	91.7%
	CNN	3.9G	3.9G	0	76.2%	92.9%

4. Experiment

4.4. Visualization on features



(a) Visualization of features in AdderNets

(b) Visualization of features in CNNs

Figure 1. Visualization of features in AdderNets and CNNs. Features of CNNs in different classes are divided by their angles. In contrast, features of AdderNets tend to be clustered towards different class centers, since AdderNets use the ℓ_1-norm to distinguish different classes. The visualization results suggest that ℓ_1-distance can served as a similarity measure the distance between the filter and the input feature in deep neural networks

✓ CNN : 각도로 class를 분류함

- 필터와 입력 간의 cross correlation이 정규화되면 두 벡터 간의 cosine distance와 같음

✓ AddNN : 군집화된 점의 좌표로 class를 분류함

- L1-distance도 필터와 입력 간의 유사도로써 사용할 수 있음

4. Experiment

4.5. Visualization on filters

(a) Visualization of filters of AdderNets

(b) Visualization of filters of CNNs

Figure 2. Visualization of filters in the first layer of LeNet-5-BN on the MNIST dataset. Both of them can extract useful features for image classification.

✓ CNN과 AdderNet이 비슷한 패턴을 가짐

4. Experiment

4.6. Visualization on distribution of weights

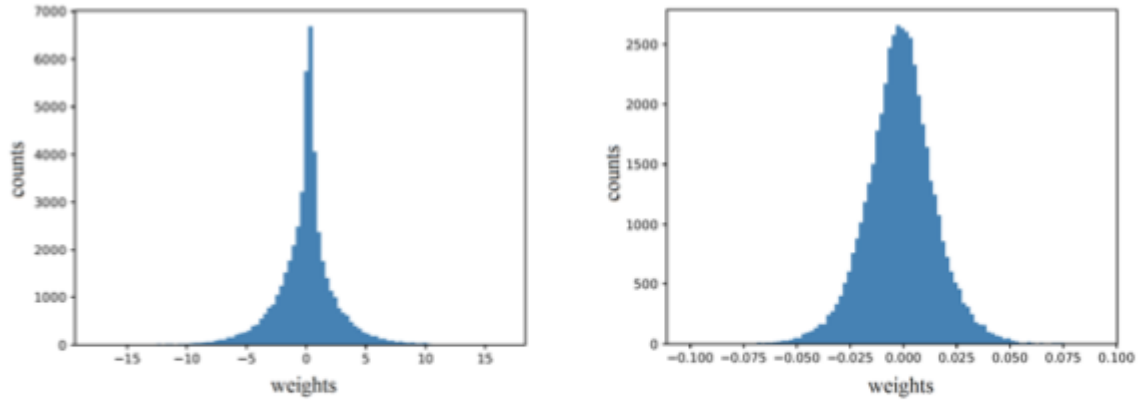


Figure 4. Histograms over the weights with AdderNet (left) and CNN (right). The weights of AdderNets follow Laplace distribution while those of CNNs follow Gaussian distribution.

- ✓ AdderNet : Laplace distribution

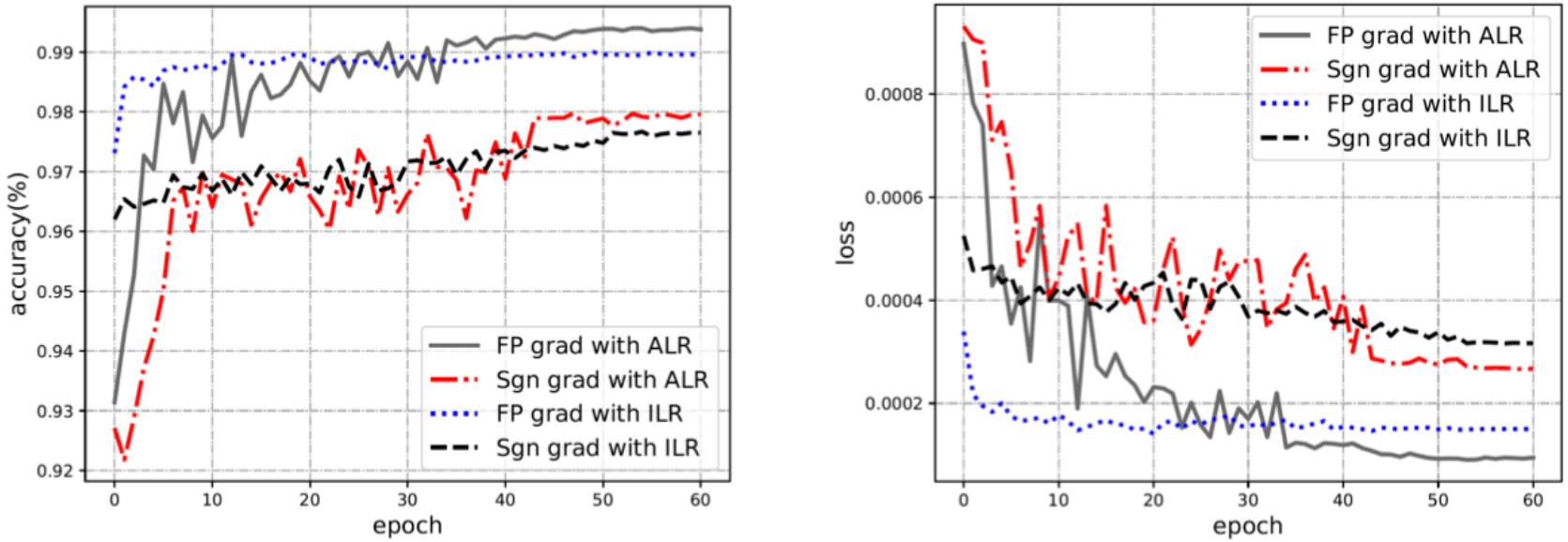
  - L1-norm의 prior distribution이 Laplace distribution을 따르기 때문

- ✓ CNN : Gaussian distribution

  - L2-norm은 cross-correlation과 같고, 이의 prior distribution은 Gaussian distribution을 따름

4. Experiment

4.7. Ablation Study



(a) Accuracy (b) Loss

Figure 3. Learning curve of AdderNets using different optimization schemes. FP and Sgn gradient denotes the full-precision and sign gradient. The proposed adaptive learning rate scaling with full-precision gradient achieves the highest accuracy (99.40%) with the smallest loss.

- ✓ learning rate를 바꾸지 않았을 때, small gradient로 인해 network를 학습하기 어려웠음
- ✓ filter의 learning rate를 100으로 키웠을 때(ILR) 가장 좋은 성능을 보였음
- ✓ sign gradient를 사용했을 때 CNN의 성능(99.40%)보다 ALR(97.99%), ILR(97.72%) 모두 성능이 낮았음
- ✓ full-precision gradient를 사용했을 때 ALR(99.40%), ILR(98.99%)로 CNN과 비슷한 성능을 보임

5. Conclusions

- CNN의 역할은 feature와 filter간 similarity를 측정하는 것, L1-distance도 가능
- convolution에서의 multiplication을 addition으로 바꿔 효율적인 연산이 가능함
- AdderNet이 CNN의 성능과 근접함을 benchmark dataset을 통해 확인함

- Image classification 뿐만 아니라 detection이나 segmentation task에서도 검증이 필요함
- 논문에서는 BNN의 실험과 유사한 세팅을 위해 ResNet의 첫번째, 마지막 convolution layer를 그대로 사용했지만, 실제로 이를 adder layer로 바꿨을 시 성능이 매우 하락했다고 함
- 모바일 디바이스의 Inference time을 위한 연구임을 언급했음에도 해당 부분에 대한 실험 결과가 없음

Do We Really Need Multiplications in Deep Learning?

=> Yes.