Review

- Object detection = Classification and **localization** of multiple objects
    - Box proposal, IoU, NMS

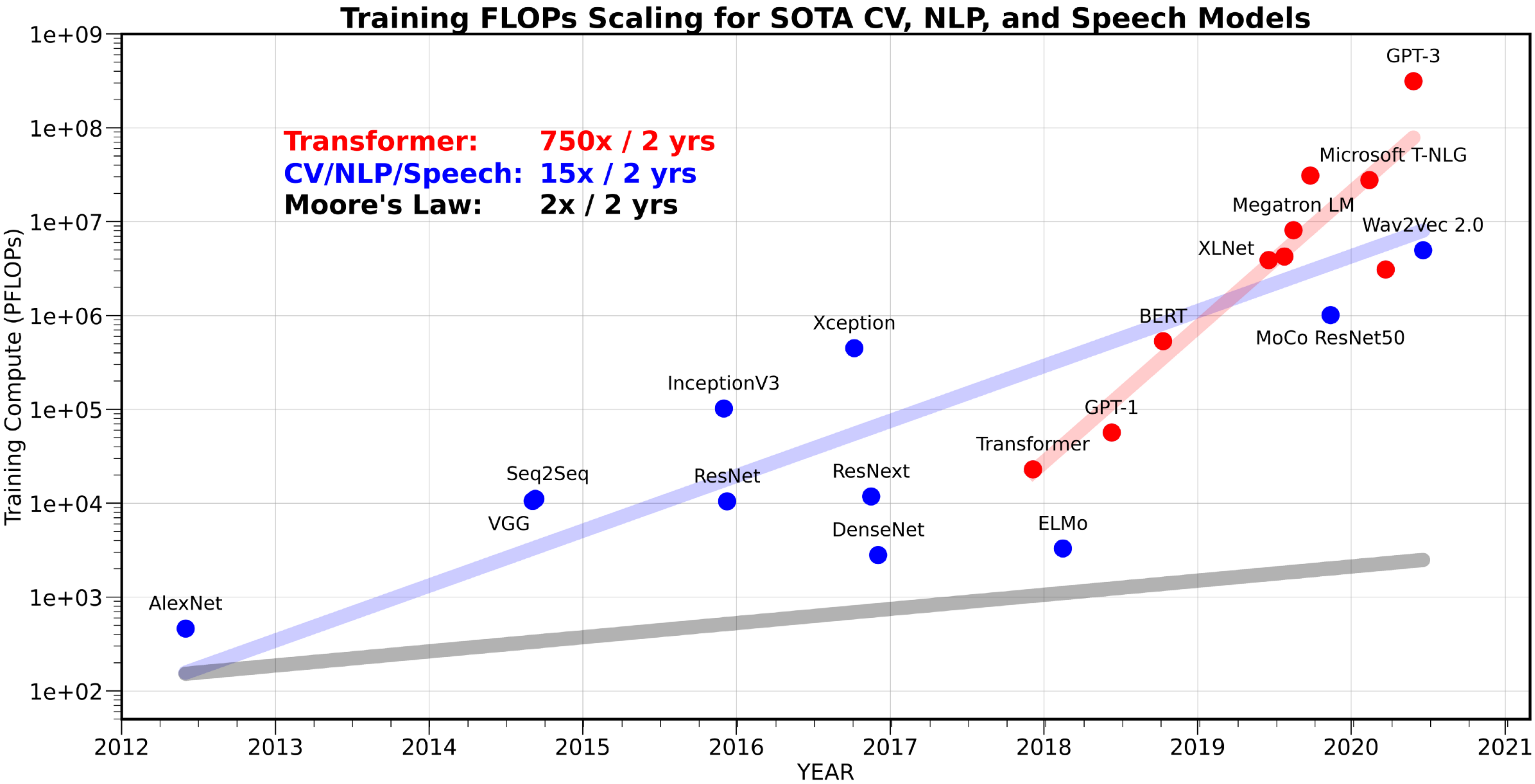
- Sliding windows
    - Intuitive but extremely slow (Run CNN for each of too many boxes)
- R-CNN
    - Run CNN for each of (only) 2000 boxes (selective search) but still slow
- SPPnet, Fast R-CNN, and Faster R-CNN
    - Much more advanced compared to R-CNN
- YoLo
    - Run CNN once and propose 98 boxes, fast but not accurate
- SSD
    - Run CNN once without FC layers and propose 8732 boxes, fast and accurate

DNN Quantization (1)

Lecture 6

Hyung-Sin Kim

SNU Graduate School of Data Science



[The figure comes from RISELab blog post at https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8]

On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜

Emily M. Bender*
ebender@uw.edu
University of Washington
Seattle, WA, USA

Timnit Gebru*
timnit@blackinai.org
Black in AI
Palo Alto, CA, USA

Angelina McMillan-Major
aymm@uw.edu
University of Washington
Seattle, WA, USA

Shmargaret Shmitchell
shmargaret.shmitchell@gmail.com
The Aether

ABSTRACT

The past 3 years of work in NLP have been characterized by the development and deployment of ever larger language models, especially for English. BERT, its variants, GPT-2/3, and others, most recently Switch-C, have pushed the boundaries of the possible both through architectural innovations and through sheer size. Using these pretrained models and the methodology of fine-tuning them for specific tasks, researchers have extended the state of the art on a wide array of tasks as measured by leaderboards on specific benchmarks for English. In this paper, we take a step back and ask: How big is too big? What are the possible risks associated with this technology and what paths are available for mitigating those risks? We provide recommendations including weighing the environmental and financial costs first, investing resources into curating and carefully documenting datasets rather than ingesting everything on the web, carrying out pre-development exercises evaluating how the planned approach fits into research and development goals and supports stakeholder values, and encouraging research directions beyond ever larger language models.

alone, we have seen the emergence of BERT and its variants [39, 70, 74, 113, 146], GPT-2 [106], T-NLG [112], GPT-3 [25], and most recently Switch-C [43], with institutions seemingly competing to produce ever larger LMs. While investigating properties of LMs and how they change with size holds scientific interest, and large LMs have shown improvements on various tasks (§2), we ask whether enough thought has been put into the potential risks associated with developing them and strategies to mitigate these risks.

We first consider environmental risks. Echoing a line of recent work outlining the environmental and financial costs of deep learning systems [129], we encourage the research community to prioritize these impacts. One way this can be done is by reporting costs and evaluating works based on the amount of resources they consume [57]. As we outline in §3, increasing the environmental and financial costs of these models doubly punishes marginalized communities that are least likely to benefit from the progress achieved by large LMs and most likely to be harmed by negative environmental consequences of its resource consumption. At the scale we are discussing (outlined in §2), the first consideration should be the

Google fires prominent AI ethicist Timnit Gebru

Gebru says the decision came from Google's head of AI, Jeff Dean

By Zoe Schiffer | @ZoeSchiffer | Dec 3, 2020, 1:11pm EST

Listen to this article

SHARE

Photo by Kimberly White / Getty Images for TechCrunch

Google fires top ethical AI expert Margaret Mitchell

The tech giant claims Mitchell violated staff codes of conduct.

Google has fired the co-lead of the company's ethical AI unit, Margaret Mitchell, on the heels of the removal of Timnit Gebru.

Mitchell, an ethical artificial intelligence (AI) expert who has previously worked on machine learning bias, race and gender diversity, and language models for image capture, was hired by Google to co-lead the firm's Ethical AI team with Gebru -- a post that has lasted roughly two years, as noted by Reuters.

Margaret Mitchell [right], was fired on the heels of the removal of Timnit Gebru.

Motivation – CNN vs. MLP

- CNN

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

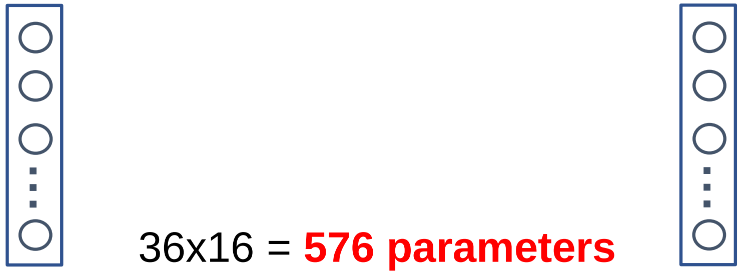
6x6

3x3 = **9 parameters**

4x4

Parameter sharing (less memory)

- Fully connected layer



36x16 = **576 parameters**

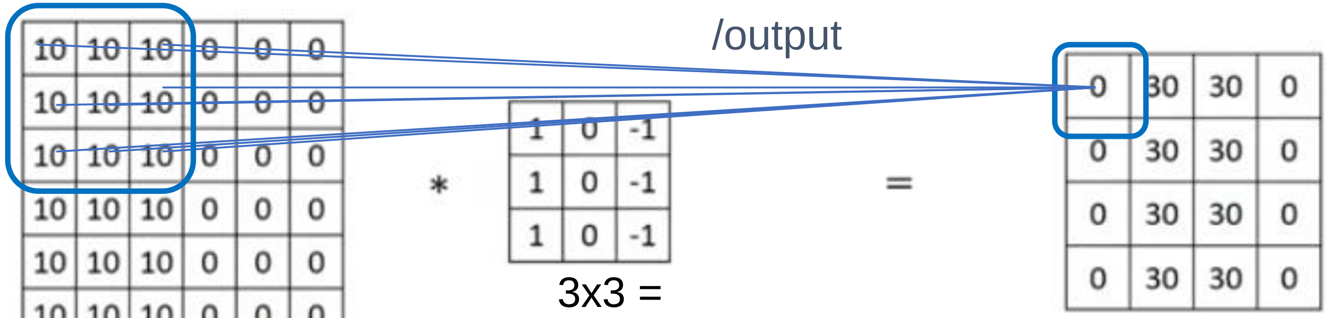
36

16

Motivation – CNN vs. MLP

- CNN

9 connections
/output



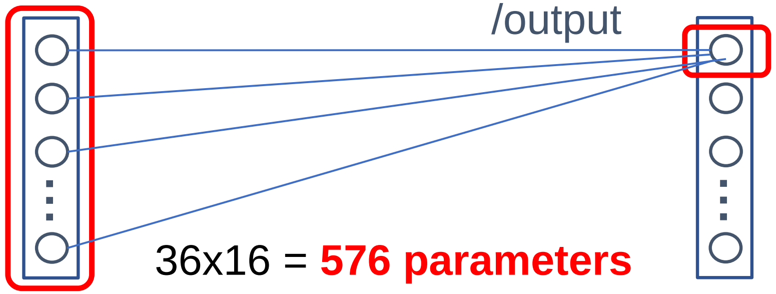
6x6

3x3 =
9 parameters

4x4

- Fully connected layer

36 connections
/output



36x16 = **576 parameters**

36

16

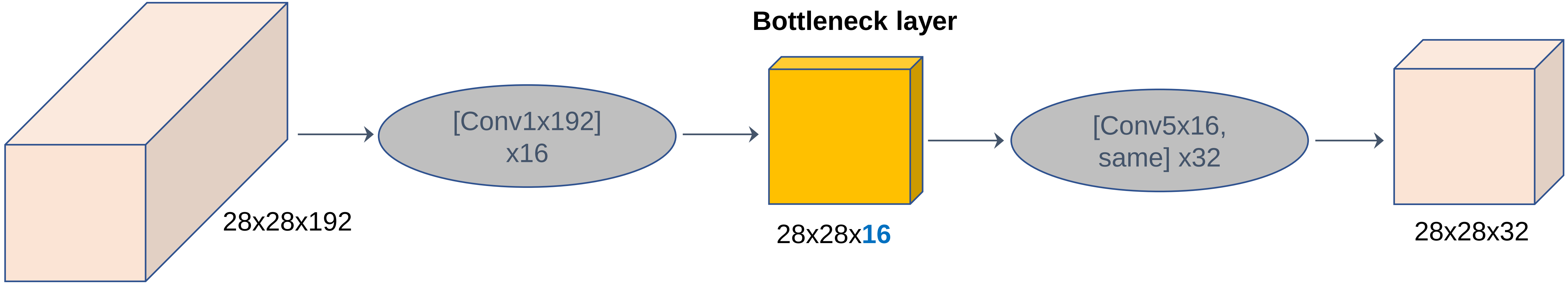
Parameter sharing
(less memory)

Sparsity of connections
(less computation)

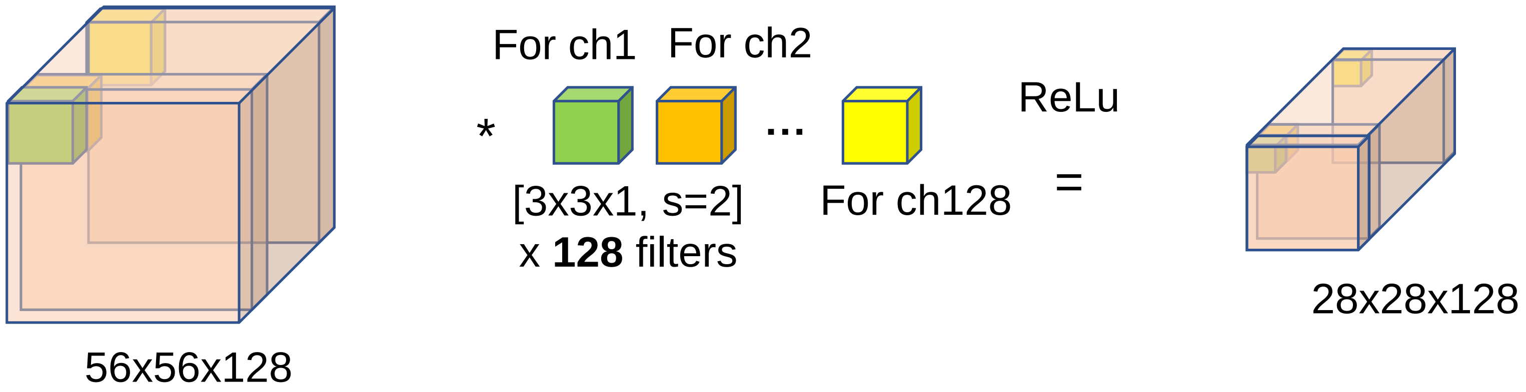
Regularization
(less overfitting)

Motivation – Lightening Techniques for CNN

- Bottleneck architecture



- Depthwise convolution



Lots of computation and memory that a DNN incurs might be ***unnecessary*** *actually…*

Quantization Question

Is it necessary to represent every weight and activation with ***32-bit*** *float?*

BinaryConnect [NIPS'15] - Concept

- How about using **<u>1-bit (1 or -1) binary</u>** value for weights?
  - Not only lower memory but eliminating multiplications!

- Deterministic binarization

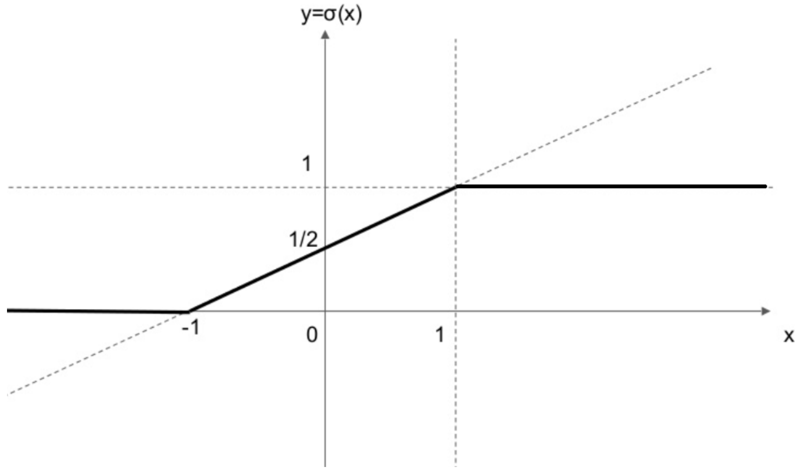
$$w_b = \begin{cases} +1 & \text{if } w \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

- Stochastic binarization

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

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

Hard Sigmoid (for simplicity)



[The figures are from M. Courbariaux et al., "Binaryconnect: Training deep neural networks with binary weights during propagations."]

BinaryConnect [NIPS'15] - Training

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:

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

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

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 $\frac{\partial C}{db_{t-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and a_{k-1}

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

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

***Binarize** the previous weight*

$z_k = a_{k-1} \cdot \boldsymbol{w_k^b} + b_k$

$a_k = g(z_k)$

Activations and biases are real values

[The figures are from M. Courbariaux et al., "Binaryconnect: Training deep neural networks with binary weights during propagations."]

BinaryConnect [NIPS'15] - Training

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 $\boxed{\frac{\partial C}{\partial a_{k-1}}}$ 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}

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

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

$$z_k = a_{k-1} \cdot \boldsymbol{w_k^b} + b_k$$

$$a_k = g(z_k)$$

$$\frac{\partial C}{\partial a_{k-1}} = \frac{\partial C}{\partial a_k} \cdot \frac{\partial a_k}{\partial z_k} \cdot \frac{\partial z_k}{\partial a_{k-1}} = \frac{\partial C}{\partial a_k} \cdot g(z_k)' \cdot \boldsymbol{w_k^b}$$

$$\frac{\partial C}{\partial w_{k-1}^b} = \frac{\partial C}{\partial a_k} \cdot \frac{\partial a_k}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_{k-1}^b} = \frac{\partial C}{\partial a_k} \cdot g(z_k)' \cdot a_{k-1}$$

[The figures are from M. Courbariaux et al., "Binaryconnect: Training deep neural networks with binary weights during propagations."]

BinaryConnect [NIPS'15] - Training

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

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

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

$z_k = a_{k-1} \cdot \boldsymbol{w_k^b} + b_k$

$a_k = g(z_k)$

$$\frac{\partial C}{\partial a_{k-1}} = \frac{\partial C}{\partial a_k} \cdot \frac{\partial a_k}{\partial z_k} \cdot \frac{\partial z_k}{\partial a_{k-1}} = \frac{\partial C}{\partial a_k} \cdot g(z_k)' \cdot \boldsymbol{w_k^b}$$

$$\frac{\partial C}{\partial w_{k-1}^b} = \frac{\partial C}{\partial a_k} \cdot \frac{\partial a_k}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_{k-1}^b} = \frac{\partial C}{\partial a_k} \cdot g(z_k)' \cdot a_{k-1}$$

Gradients of binarized weights are real values

[The figures are from M. Courbariaux et al., "Binaryconnect: Training deep neural networks with binary weights during propagations."]

BinaryConnect [NIPS'15] - Training

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}

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

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

*Update the <u>real value</u> weight by using gradients of <u>binarized</u> weight… (**not perfect**)*

***Clip** the updated weight for **regularization**!*

<u>mitigation of weight explosion</u>
– we want real weight updates to impact binary weights

[The figures are from M. Courbariaux et al., "Binaryconnect: Training deep neural networks with binary weights during propagations."]

BinaryConnect [NIPS'15] - Results

- MLP for MNIST
  - 3 hidden layers of 1024 nodes, ReLU, and L2-SVM for the output layer
- CNN for CIFAR-10 and SVHN
  - $(2\times128C3)-MP2-(2\times256C3)-MP2-(2\times512C3)-MP2-(2\times1024FC)-10SVM$

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%**

It works very well!
Slightly worse than using dropout

[The figures are from M. Courbariaux et al., "Binaryconnect: Training deep neural networks with binary weights during propagations."]

BinaryConnect [NIPS'15] - Results

- Potentially 3 times faster training (1/3 multiplications)
- 16 times less memory to store a model (16 bit to 1 bit)
- Zero multiplication at test time!

Why not quantizing activations too?

BNN [NIPS'16] – Training

- Forward propagation with binary weights and **activations**

a_{k-1}^{b} *Binary value*

{1. Computing the gradients:}
{1.1. Forward propagation:}
for $k = 1$ to L **do**
  $\boxed{W_k^b \leftarrow \text{Binarize}(W_k)}$, $s_k \leftarrow a_{k-1}^b W_k^b$
  $a_k \leftarrow \text{BatchNorm}(s_k, \theta_k)$
  if $k < L$ **then** $a_k^b \leftarrow \text{Binarize}(a_k)$

[The figure is from I. Hubara et al., "Binarized neural networks."]

BNN [NIPS’16] – Training

- Forward propagation with binary weights and **activations**

{1. Computing the gradients:}
{1.1. Forward propagation:}
for $k = 1$ to L **do**
  $W_k^b \leftarrow \mathrm{Binarize}(W_k)$, $\boxed{s_k \leftarrow a_{k-1}^b W_k^b}$
  $a_k \leftarrow \mathrm{BatchNorm}(s_k, \theta_k)$
  if $k < L$ **then** $a_k^b \leftarrow \mathrm{Binarize}(a_k)$

a_{k-1}^b *Binary value*

Binary sum $\downarrow$

s_k *Real value*

[The figure is from I. Hubara et al., “Binarized neural networks.”]

BNN [NIPS'16] – Training

- Forward propagation with binary weights and **activations**

{1. Computing the gradients:}
{1.1. Forward propagation:}
for $k = 1$ to L **do**
$W_k^b \leftarrow \text{Binarize}(W_k)$, $s_k \leftarrow a_{k-1}^b W_k^b$
$\boxed{a_k \leftarrow \text{BatchNorm}(s_k, \theta_k)}$
if $k < L$ **then** $a_k^b \leftarrow \text{Binarize}(a_k)$

a_{k-1}^b *Binary value*

Binary sum $\downarrow$

s_k *Real value*

BatchNorm $\downarrow$

a_k *Real value*

[The figure is from I. Hubara et al., "Binarized neural networks."]

BNN [NIPS'16] – Training

- Forward propagation with binary weights and **activations**
  - Binarize after batch normalization

{1. Computing the gradients:}
{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** $\boxed{a_k^b \leftarrow \text{Binarize}(a_k)}$

a_{k-1}^b — *Binary value*

↓ *Binary sum*

s_k — *Real value*

↓ *BatchNorm*

a_k — *Real value*

↓ *Binarization*

a_k^b — *Binary value*

[The figure is from I. Hubara et al., "Binarized neural networks."]

BNN [NIPS'16] – Training

- Back propagation

{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** $g_{a_k} \leftarrow g_{a_k^b} \circ 1_{|a_k| \leq 1}$
  $(g_{s_k}, g_{\theta_k}) \leftarrow \text{BackBatchNorm}(g_{a_k}, s_k, \theta_k)$
  $\boxed{g_{a_{k-1}^b} \leftarrow g_{s_k} W_k^b,}\ g_{W_k^b} \leftarrow g_{s_k}^\top a_{k-1}^b$

$\frac{\partial C}{\partial a_{k-1}^b}$ *Gradient of binary value*

Binary sum $\uparrow$

$\frac{\partial C}{\partial s_k}$ *Gradient of real value*

$\uparrow$

$\frac{\partial C}{\partial a_k}$ *Gradient of real value*

$\uparrow$

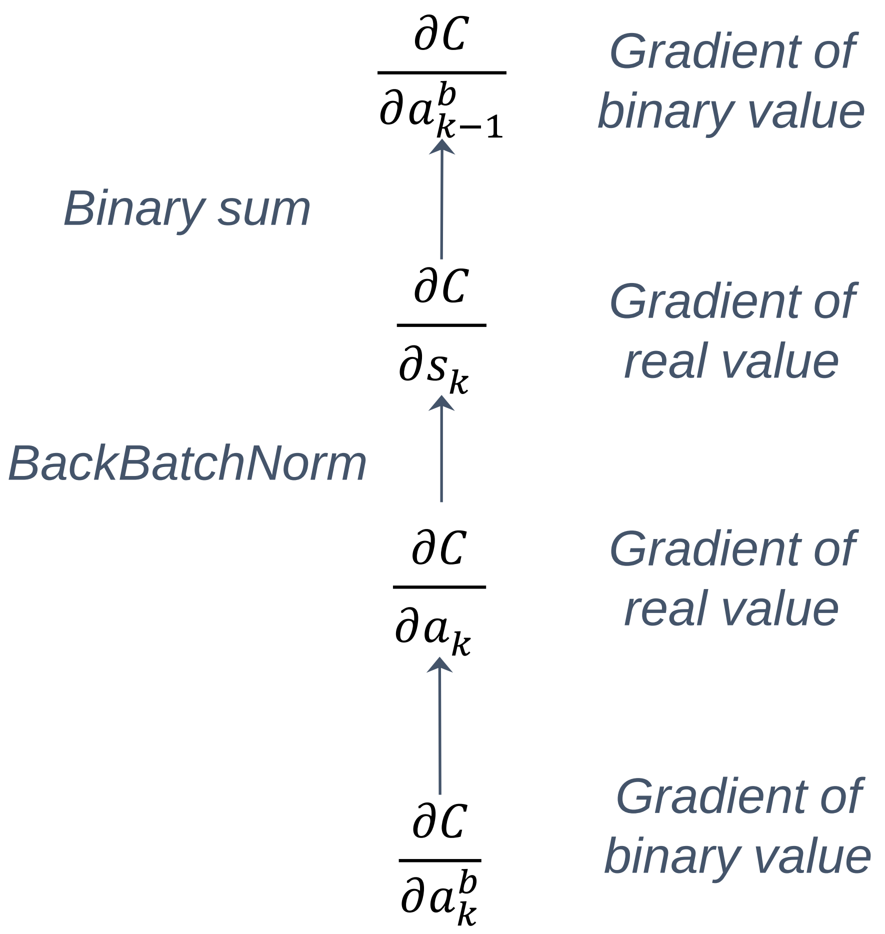
$\frac{\partial C}{\partial a_k^b}$ *Gradient of binary value*

[The figure is from I. Hubara et al., "Binarized neural networks."]

BNN [NIPS'16] – Training

- Back propagation

{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** $g_{a_k} \leftarrow g_{a_k^b} \circ 1_{|a_k| \leq 1}$
  $\boxed{(g_{s_k}, g_{\theta_k}) \leftarrow \text{BackBatchNorm}(g_{a_k}, s_k, \theta_k)}$
  $g_{a_{k-1}^b} \leftarrow g_{s_k} W_k^b$, $g_{W_k^b} \leftarrow g_{s_k}^\top a_{k-1}^b$



[The figure is from I. Hubara et al., "Binarized neural networks."]

BNN [NIPS'16] – Training

- Back propagation
  - Need gradient of **sign function**

{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** $\boxed{g_{a_k} \leftarrow g_{a_k^b} \circ 1_{|a_k| \leq 1}}$??
  $(g_{s_k}, g_{\theta_k}) \leftarrow \text{BackBatchNorm}(g_{a_k}, s_k, \theta_k)$
  $g_{a_{k-1}^b} \leftarrow g_{s_k} W_k^b$, $g_{W_k^b} \leftarrow g_{s_k}^\top a_{k-1}^b$

$\frac{\partial C}{\partial a_{k-1}^b}$ *Gradient of binary value*

Binary sum

$\frac{\partial C}{\partial s_k}$ *Gradient of real value*

BackBatchNorm

$\frac{\partial C}{\partial a_k}$ *Gradient of real value*

$a_k^b = \text{sign}(a_k)$ $\frac{\partial a_k^b}{\partial a_k}$??

$\frac{\partial C}{\partial a_k^b}$ *Gradient of binary value*

[The figure is from I. Hubara et al., "Binarized neural networks."]

BNN [NIPS’16] – Training

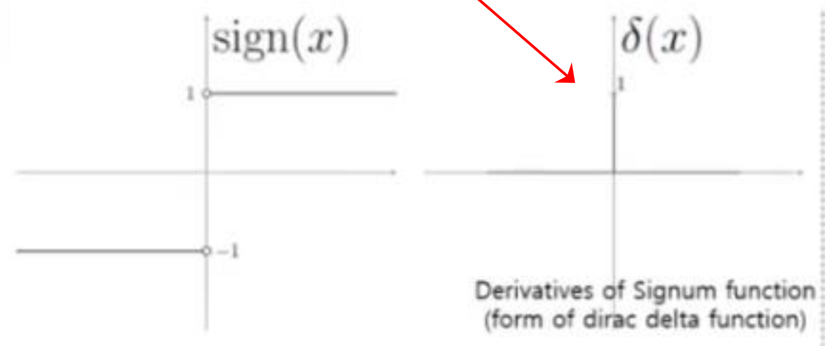
- Back propagation
  - Need gradient of **sign function**

{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** $\boxed{g_{a_k} \leftarrow g_{a_k^b} \circ 1_{|a_k| \leq 1}}$
  $(g_{s_k}, g_{\theta_k}) \leftarrow \text{BackBatchNorm}(g_{a_k}, s_k, \theta_k)$
  $g_{a_{k-1}^b} \leftarrow g_{s_k} W_k^b,\ g_{W_k^b} \leftarrow g_{s_k}^\top a_{k-1}^b$

$$\frac{\partial C}{\partial a_k} = \frac{\partial C}{\partial a_k^b} \cdot \frac{\partial a_k^b}{\partial a_k}$$
$$= \frac{\partial C}{\partial a_k^b} \cdot sign(a_k)'$$
$$= \frac{\partial C}{\partial a_k^b} \cdot \boldsymbol{\delta\ (a_k)}$$

$a_k^b = \text{sign}(a_k)$

Gradient vanishing!



[The figure is from I. Hubara et al., “Binarized neural networks.”]

BNN [NIPS'16] – Training

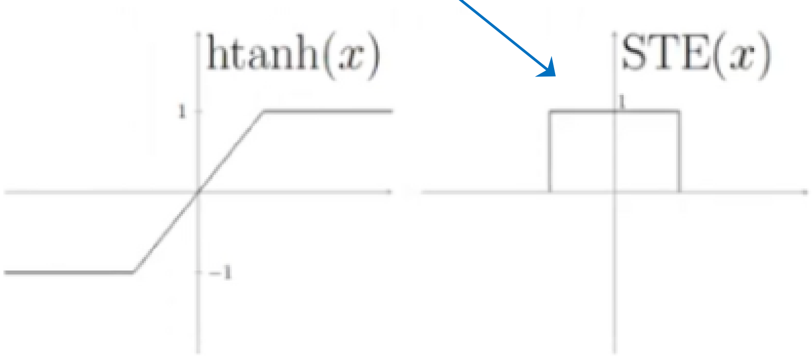
- Back propagation
  - Need gradient of **sign function**

{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** $\boxed{g_{a_k} \leftarrow g_{a_k^b} \circ 1_{|a_k| \leq 1}}$
  $(g_{s_k}, g_{\theta_k}) \leftarrow \text{BackBatchNorm}(g_{a_k}, s_k, \theta_k)$
  $g_{a_{k-1}^b} \leftarrow g_{s_k} W_k^b,\ g_{W_k^b} \leftarrow g_{s_k}^\top a_{k-1}^b$

$$\frac{\partial C}{\partial a_k} = \frac{\partial C}{\partial a_k^b} \cdot \frac{\partial a_k^b}{\partial a_k} = \frac{\partial C}{\partial a_k^b} \cdot \boldsymbol{htanh(a_k)'} = \frac{\partial C}{\partial a_k^b} \cdot \mathbf{1}_{|a_k| \leq 1}$$

$a_k^b = htanh(a_k)$

Fake function only for backprop…



Straight-through estimator (empirical finding…)

[The figure is from I. Hubara et al., "Binarized neural networks."]

BNN [NIPS'16] – Training

- Parameter updates

{2. Accumulating the gradients:}
for $k = 1$ to L **do**
$\theta_k^{t+1} \leftarrow \text{Update}(\theta_k, \eta^t, g_{\theta_k}), \eta^{t+1} \leftarrow \lambda\eta^t$
$\boxed{W_k^{t+1} \leftarrow \text{Clip}(\text{Update}(W_k, \gamma_k\eta^t, g_{W_k^b}), -1, 1)}$

= *BinaryConnect*

[The figure is from I. Hubara et al., "Binarized neural networks."]

BNN [NIPS'16] –Results

- Performance
  - Slightly worse than BinaryConnect

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%

[The figure is from I. Hubara et al., "Binarized neural networks."]

Let’s step forward!
Not just for MNIST but for ImageNet classification!

XNOR-Net [ECCV'16] – BWN Concept

- Applying BNN or BinaryConnect to a $\boldsymbol{c} \times \boldsymbol{w} \times \boldsymbol{h}$ Conv filter
  - $\mathbf{W} \cong \mathbf{B}$ where $\mathbf{B} \in \{-1,1\}^{\boldsymbol{c} \times \boldsymbol{w} \times \boldsymbol{h}}$
  - But only 1 and -1 seem **not enough**…

- Let's add a real value scaling factor for more precision!
  - $\mathbf{W} \cong \boldsymbol{\alpha}\mathbf{B}$ where $\mathbf{B} \in \{-1,1\}^{\boldsymbol{c} \times \boldsymbol{w} \times \boldsymbol{h}}$
    - **Memory**: One real value per conv filter (not too bad…)
  - One conv filter operation for a $\boldsymbol{c} \times \boldsymbol{w} \times \boldsymbol{h}$ input $\mathbf{I}$ becomes $\mathbf{I} * \mathbf{W} \cong (\mathbf{I} \oplus \mathbf{B})\boldsymbol{\alpha}$
    - $\oplus$ is a convolution without any multiplication (only additions and subtractions)
    - **Computation:** One multiplication per one output element (not too bad…)

XNOR-Net [ECCV'16] – Binary Weight Estimation

- Optimization problem
  - $\alpha^*, \mathbf{B}^* = argmin_{\alpha,\mathbf{B}}\ J(\mathbf{B},\alpha)$ where $J(\mathbf{B},\alpha) = \|\mathbf{W} - \alpha\mathbf{B}\|^2$
  - $J(\mathbf{B},\alpha) = (\mathbf{W} - \alpha\mathbf{B})^{\mathrm{T}}(\mathbf{W} - \alpha\mathbf{B})$
    $= \alpha^2\mathbf{B}^{\mathrm{T}}\mathbf{B} - 2\alpha\mathbf{W}^{\mathrm{T}}\mathbf{B} + \mathbf{W}^{\mathrm{T}}\mathbf{W}$
    $= \alpha^2 n - 2\alpha\mathbf{W}^{\mathrm{T}}\mathbf{B} + \mathbf{W}^{\mathrm{T}}\mathbf{W}$ where $n = c \times w \times h$
- Solution for $\mathbf{B}^*$
  - $\alpha^2 n$ and $\mathbf{W}^{\mathrm{T}}\mathbf{W}$ are constant values in the perspective of $\mathbf{B}$
  - $\mathbf{B}^* = argmax_{\mathbf{B}}\ \{\mathbf{W}^{\mathrm{T}}\mathbf{B}\} = sign(\mathbf{W})$ (same as BinaryConnect…)
- Solution for α^*
  - $\frac{\partial J(\mathbf{B},\alpha)}{\partial\alpha} = 2n\alpha - 2\mathbf{W}^{\mathrm{T}}\mathbf{B}$
  - $\alpha^* = \frac{\mathbf{W}^{\mathrm{T}}\mathbf{B}^*}{n} = \frac{\mathbf{W}^{\mathrm{T}}sign(\mathbf{W})}{n} = \frac{1}{n}\|\mathbf{W}\|_{L1}$

$$\mathbf{W}_b = \frac{1}{n}\|\mathbf{W}\|_{L1} \cdot sign(\mathbf{W})$$

XNOR-Net [ECCV'16] – BWN Training

Algorithm 1 Training an L-layers CNN with binary weights:

Input: A minibatch of inputs and targets $(\mathbf{I}, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weight $\mathcal{W}^t$ and current learning rate η^t.

Output: updated weight $\mathcal{W}^{t+1}$ and updated learning rate η^{t+1}.

Binarizing weight filters:

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

  for k^{th} filter in l^{th} layer **do**

    $\mathcal{A}_{lk} = \frac{1}{n} \|\mathcal{W}^t_{lk}\|_{\ell 1}$

    $\mathcal{B}_{lk} = \text{sign}(\mathcal{W}^t_{lk})$

    $\widetilde{\mathcal{W}}_{lk} = \mathcal{A}_{lk}\mathcal{B}_{lk}$

Binarize all conv filters

$\hat{\mathbf{Y}} =$ **BinaryForward**$(\mathbf{I}, \mathcal{B}, \mathcal{A})$ // standard forward propagation except that convolutions are computed using equation 1 or 11

$\frac{\partial C}{\partial \widetilde{\mathcal{W}}} =$ **BinaryBackward**$(\frac{\partial C}{\partial \hat{\mathbf{Y}}}, \widetilde{\mathcal{W}})$ // standard backward propagation except that gradients are computed using $\widetilde{\mathcal{W}}$ instead of $\mathcal{W}^t$

$\mathcal{W}^{t+1} =$ **UpdateParameters**$(\mathcal{W}^t, \frac{\partial C}{\partial \widetilde{\mathcal{W}}}, \eta_t)$ // Any update rules (*e.g.*,SGD or ADAM)

$\eta^{t+1} =$ **UpdateLearningrate**(η^t, t) // Any learning rate scheduling function

[The figure is from M. Rastegari et al., "Xnor-net: Imagenet classification using binary convolutional neural networks."]

XNOR-Net [ECCV'16] – BWN Training

Algorithm 1 Training an L-layers CNN with binary weights:

Input: A minibatch of inputs and targets $(\mathbf{I}, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weight $\mathcal{W}^t$ and current learning rate η^t.

Output: updated weight $\mathcal{W}^{t+1}$ and updated learning rate η^{t+1}.

```
Binarizing weight filters:
for l = 1 to L do
    for k^th filter in l^th layer do
        A_lk = (1/n) ||W^t_lk||_ℓ1
        B_lk = sign(W^t_lk)
        W~_lk = A_lk B_lk
Ŷ = BinaryForward(I, B, A)   // standard forward propagation except that convolutions are computed using equation 1 or 11
∂C/∂W~ = BinaryBackward(∂C/∂Ŷ, W~)   // standard backward propagation except that gradients are computed using W~ instead of W^t
W^{t+1} = UpdateParameters(W^t, ∂C/∂W~, η_t)   // Any update rules (e.g.,SGD or ADAM)
η^{t+1} = UpdateLearningrate(η^t, t)   // Any learning rate scheduling function
```

$\mathcal{A}_{lk} = \frac{1}{n}\|\mathcal{W}^t_{lk}\|_{\ell 1}$, $\mathcal{B}_{lk} = \operatorname{sign}(\mathcal{W}^t_{lk})$, $\widetilde{\mathcal{W}}_{lk} = \mathcal{A}_{lk}\mathcal{B}_{lk}$

$\boxed{\hat{\mathbf{Y}} = \mathbf{BinaryForward}(\mathbf{I}, \mathcal{B}, \mathcal{A})}$

$\frac{\partial C}{\partial \widetilde{\mathcal{W}}} = \mathbf{BinaryBackward}\left(\frac{\partial C}{\partial \hat{\mathbf{Y}}}, \widetilde{\mathcal{W}}\right)$

$\mathcal{W}^{t+1} = \mathbf{UpdateParameters}\left(\mathcal{W}^t, \frac{\partial C}{\partial \widetilde{\mathcal{W}}}, \eta_t\right)$

$\eta^{t+1} = \mathbf{UpdateLearningrate}(\eta^t, t)$

Forward propagation with binarized weights

[The figure is from M. Rastegari et al., "Xnor-net: Imagenet classification using binary convolutional neural networks."]

XNOR-Net [ECCV'16] – BWN Training

Algorithm 1 Training an L-layers CNN with binary weights:

Input: A minibatch of inputs and targets $(\mathbf{I}, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weight $\mathcal{W}^t$ and current learning rate η^t.

Output: updated weight $\mathcal{W}^{t+1}$ and updated learning rate η^{t+1}.

1: Binarizing weight filters:

2: **for** $l = 1$ to L **do**

3: **for** k^{th} filter in l^{th} layer **do**

4: $\mathcal{A}_{lk} = \frac{1}{n}\|\mathcal{W}^t_{lk}\|_{\ell 1}$

5: $\mathcal{B}_{lk} = \text{sign}(\mathcal{W}^t_{lk})$

6: $\widetilde{\mathcal{W}}_{lk} = \mathcal{A}_{lk}\mathcal{B}_{lk}$

7: $\hat{\mathbf{Y}} = \textbf{BinaryForward}(\mathbf{I}, \mathcal{B}, \mathcal{A})$ // standard forward propagation except that convolutions are computed using equation 1 or 11

8: $\boxed{\frac{\partial C}{\partial \widetilde{\mathcal{W}}} = \textbf{BinaryBackward}(\frac{\partial C}{\partial \hat{\mathbf{Y}}}, \widetilde{\mathcal{W}})}$ // standard backward propagation except that gradients are computed using $\widetilde{\mathcal{W}}$ instead of $\mathcal{W}^t$

9: $\mathcal{W}^{t+1} = \textbf{UpdateParameters}(\mathcal{W}^t, \frac{\partial C}{\partial \widetilde{\mathcal{W}}}, \eta_t)$ // Any update rules (*e.g.*,SGD or ADAM)

10: $\eta^{t+1} = \textbf{UpdateLearningrate}(\eta^t, t)$ // Any learning rate scheduling function

Compute gradients for binarized weights
(as described in BinaryConnect)

[The figure is from M. Rastegari et al., "Xnor-net: Imagenet classification using binary convolutional neural networks."]

XNOR-Net [ECCV'16] – BWN Training

Algorithm 1 Training an L-layers CNN with binary weights:

Input: A minibatch of inputs and targets $(\mathbf{I}, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weight $\mathcal{W}^t$ and current learning rate η^t.

Output: updated weight $\mathcal{W}^{t+1}$ and updated learning rate η^{t+1}.

Binarizing weight filters:
for $l = 1$ to L **do**
  for k^{th} filter in l^{th} layer **do**
    $\mathcal{A}_{lk} = \frac{1}{n} \|\mathcal{W}_{lk}^t\|_{\ell 1}$
    $\mathcal{B}_{lk} = \text{sign}(\mathcal{W}_{lk}^t)$
    $\widetilde{\mathcal{W}}_{lk} = \mathcal{A}_{lk} \mathcal{B}_{lk}$
$\hat{\mathbf{Y}} = \mathbf{BinaryForward}(\mathbf{I}, \mathcal{B}, \mathcal{A})$ // standard forward propagation except that convolutions are computed using equation 1 or 11
$\frac{\partial C}{\partial \widetilde{\mathcal{W}}} = \mathbf{BinaryBackward}(\frac{\partial C}{\partial \hat{\mathbf{Y}}}, \widetilde{\mathcal{W}})$ // standard backward propagation except that gradients are computed using $\widetilde{\mathcal{W}}$ instead of $\mathcal{W}^t$
$\boxed{\mathcal{W}^{t+1} = \mathbf{UpdateParameters}(\mathcal{W}^t, \frac{\partial C}{\partial \widetilde{\mathcal{W}}}, \eta_t)}$ // Any update rules (*e.g.*,SGD or ADAM)
$\eta^{t+1} = \mathbf{UpdateLearningrate}(\eta^t, t)$ // Any learning rate scheduling function

Update real value weight using gradients
(but differently from BinaryConnect)

[The figure is from M. Rastegari et al., "Xnor-net: Imagenet classification using binary convolutional neural networks."]

XNOR-Net [ECCV'16] – BWN Training

- Weight update in BinaryConnect
  - $W^{t+1} = W^t - \eta \frac{\partial C}{\partial \boldsymbol{W_b^t}}$ (using gradients of **binarized** weights)

- Weight update in BWN
  - $W^{t+1} = W^t - \eta \frac{\partial C}{\partial W^t}$ (using gradients of **real** weights)
  - $\frac{\partial C}{\partial W^t} = \frac{\partial C}{\partial W_b^t} \cdot \frac{\partial W_b^t}{\partial W^t}$ where $W_b^t = \frac{1}{n}\|\mathbf{W}^t\|_{L1} \cdot sign(W^t)$
  - $\frac{\partial W_b^{t,i}}{\partial W^{t,i}} = \frac{1}{n} + \frac{\|\mathbf{W}^t\|_{L1}}{n} \cdot \frac{\partial \mathrm{sign}(W^{t,i})}{\partial W^{t,i}}$

$$= \frac{1}{n} + \frac{\|\mathbf{W}^t\|_{L1}}{n} \cdot \boxed{\mathbf{1}_{|W^{t,i}| \le 1}}$$

Straight-through estimator

Fine-grained training!

XNOR-Net [ECCV'16] – Input Binarization

CNN with binarization

$\mathbf{X}_{k-1}$ *Real value*

Convolution (Binary weights) $\downarrow$

$\mathbf{X}_k^c$ *Real value*

BatchNorm $\downarrow$

$\mathbf{X}_k^{BNorm}$ *Real value*

Binarization $\downarrow$

$\mathbf{X}_k^b$ *Binary value + scaling*

Pooling $\downarrow$

$\mathbf{X}_k$ *Real value*

Huge information loss (all 1s… for max pooling)

XNOR-Net

$\mathbf{X}_{k-1}$ *Real value*

BatchNorm $\downarrow$

$\mathbf{X}_k^{BNorm}$ *Real value*

Binarization $\downarrow$

$\mathbf{X}_k^b$ *Binary value + scaling*

Convolution (Binary weights) $\downarrow$

$\mathbf{X}_k^c$ *Real value*

Pooling $\downarrow$

$\mathbf{X}_k$ *Real value*

Maintaining zero-mean

Less information loss

XNOR-Net [ECCV'16] – Input Binarization

- Goal: $\mathbf{X}^\mathsf{T}\mathbf{W} \approx \beta\mathbf{H}^\mathsf{T}\alpha\mathbf{B}$, where $\mathbf{H}, \mathbf{B} \in \{+1, -1\}^n$ and $\beta, \alpha \in \mathbb{R}^+$
  - $\mathbf{H}^T\mathbf{B} = (\text{\# of XNOR–bits}) - (\boldsymbol{n} - (\text{\# of XNOR–bits}))$
    $= (\text{\# of XNOR–bits}) \ll 1 \;\; - n$

- Problem formulation: $\alpha^*, \mathbf{B}^*, \beta^*, \mathbf{H}* = \underset{\alpha,\mathbf{B},\beta,\mathbf{H}}{\operatorname{argmin}} \|\mathbf{X} \odot \mathbf{W} - \beta\alpha\mathbf{H} \odot \mathbf{B}\|$
  - $\odot$: element-wise (Frobenius) product

- Re-formulation: $\gamma^*, \mathbf{C}^* = \underset{\gamma,\mathbf{C}}{\operatorname{argmin}} \|\mathbf{Y} - \gamma\mathbf{C}\|$

  $\mathbf{Y} \in \mathbb{R}^n$ such that $\mathbf{Y}_i = \mathbf{X}_i\mathbf{W}_i$

  $\mathbf{C} \in \{+1, -1\}^n$ such that $\mathbf{C}_i = \mathbf{H}_i\mathbf{B}_i$

  $\gamma \in \mathbb{R}^+$ such that $\gamma = \beta\alpha$.

XNOR-Net [ECCV'16] – Input Binarization

- Solution
  - $\mathbf{C}^* = \text{sign}(\mathbf{Y}) = \text{sign}(\mathbf{X}) \odot \text{sign}(\mathbf{W}) = \mathbf{H}^* \odot \mathbf{B}^*$ (same as binary weight prob)
  - $\gamma^* = \frac{\sum |\mathbf{Y}_i|}{n} = \frac{\sum |\mathbf{X}_i||\mathbf{W}_i|}{n}$ Since $|\mathbf{X}_i|, |\mathbf{W}_i|$ are independent, knowing that $\mathbf{Y}_i = \mathbf{X}_i \mathbf{W}_i$ then, $\mathbf{E}\left[|\mathbf{Y}_i|\right] = \mathbf{E}\left[|\mathbf{X}_i||\mathbf{W}_i|\right] = \mathbf{E}\left[|\mathbf{X}_i|\right]\mathbf{E}\left[|\mathbf{W}_i|\right]$ therefore,

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

$$\mathbf{X}_b = \frac{1}{n}\|\mathbf{X}\|_{L1} \cdot sign(\mathbf{X})$$

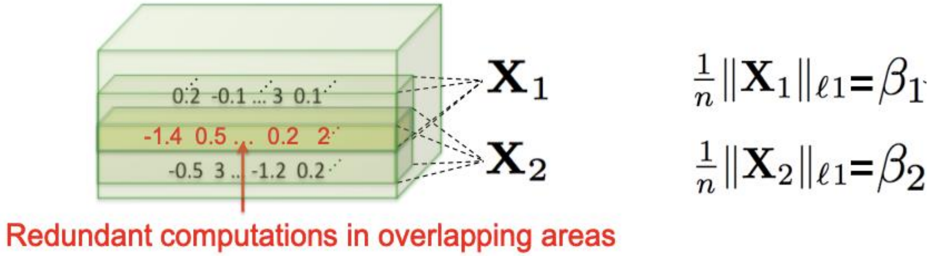
$$\mathbf{W}_b = \frac{1}{n}\|\mathbf{W}\|_{L1} \cdot sign(\mathbf{W})$$

XNOR-Net [ECCV'16] – Convolution

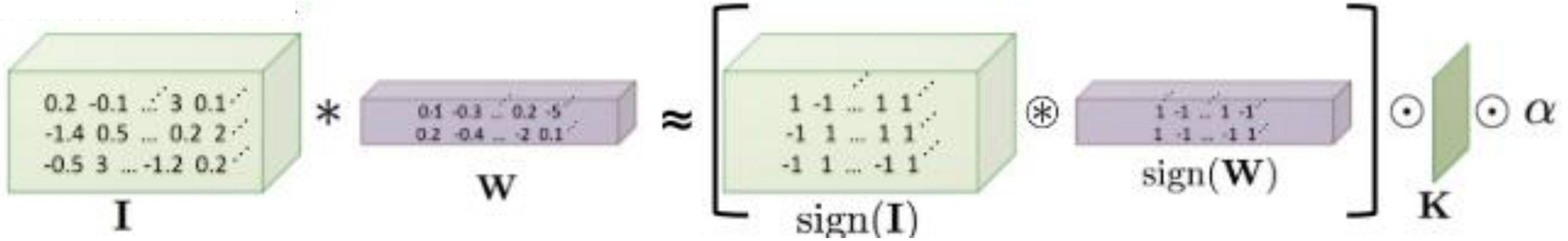
- **Problem:** Calculating $\boldsymbol{\beta}$ for each input separately incurs redundant computations
  - $(w_f \times h_f \times c_{in}) \times \boxed{(w_{out} \times h_{out})}$ additions

of $\boldsymbol{\beta}$

Inefficient



- **Solution** (3 steps)
  - 1) Average input across all channels: $\mathbf{A} = \frac{\sum |\mathbf{I}_{:,:,i}|}{c}$ $\quad \mathbf{K} = \mathbf{A} * \mathbf{k}$, where $\forall ij \ \ \mathbf{k}_{ij} = \frac{1}{w \times h}$ $\quad \mathbf{k} \in \mathbb{R}^{w \times h}$
  - 2) Calculate $\boldsymbol{\beta}$ for all sub-tensors in the input: ,
    - **K is a** $(w_{out} \times h_{out})$ matrix that contains <u>all $\boldsymbol{\beta}$ needed for each output element</u>
    - $c_{in} + (w_f \times h_f) \times (w_{out} \times h_{out})$ additions (***c* times** reduction)
  - 3) Convolution using binary operations: $\mathbf{I} * \mathbf{W} \approx (\operatorname{sign}(\mathbf{I}) \circledast \operatorname{sign}(\mathbf{W})) \odot \mathbf{K}\alpha$



[The figure is from M. Rastegari et al., "Xnor-net: Imagenet classification using binary convolutional neural networks."]

XNOR-Net [ECCV'16] – Results

- ImageNet classification

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-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

[The figure is from M. Rastegari et al., "Xnor-net: Imagenet classification using binary convolutional neural networks."]

Thanks!