Binary Deep Learning

Presented by

Roey Nagar and Kostya Berestizshevsky

Deep Learning Seminar, School of Electrical Engineering, Tel Aviv University
January 22nd 2017

Lecture Outline

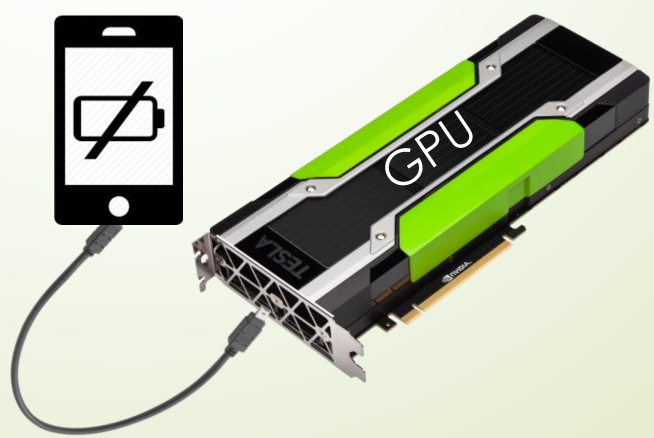
- Motivation and existing studies
- BinaryConnect
- XNOR-Net
- Bonus: Hardware for machine learning

Motivation

- Make neural computation suitable for low-power **dedicated hardware**
- Reduce computational/memory effort

  (training / inference/ both)

Alex Net:
- 61M parameters
- 1.5B high precision operations per image



Existing Methods To Reduce Over-parametrization

- Shallow-networks
- Compress a pre-trained network
- Compact Layers
- Quantize parameters (to several levels)
- Network binarization (to 2 values)

BinaryConnect: Training Deep Neural Networks with binary weights during propagations

First presented at NIPS 2015 by

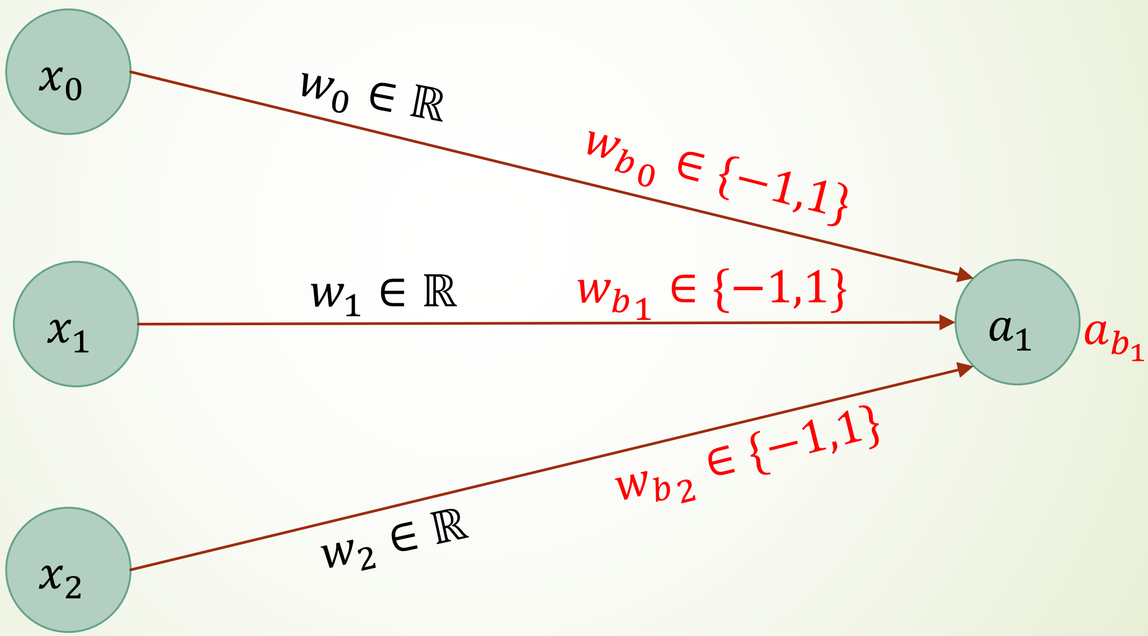
Matthieu Courbariaux
Yoshua Bengio
Jean-Pierre David

Today presented by

Roey Nagar

Idea

- Binarization of weights



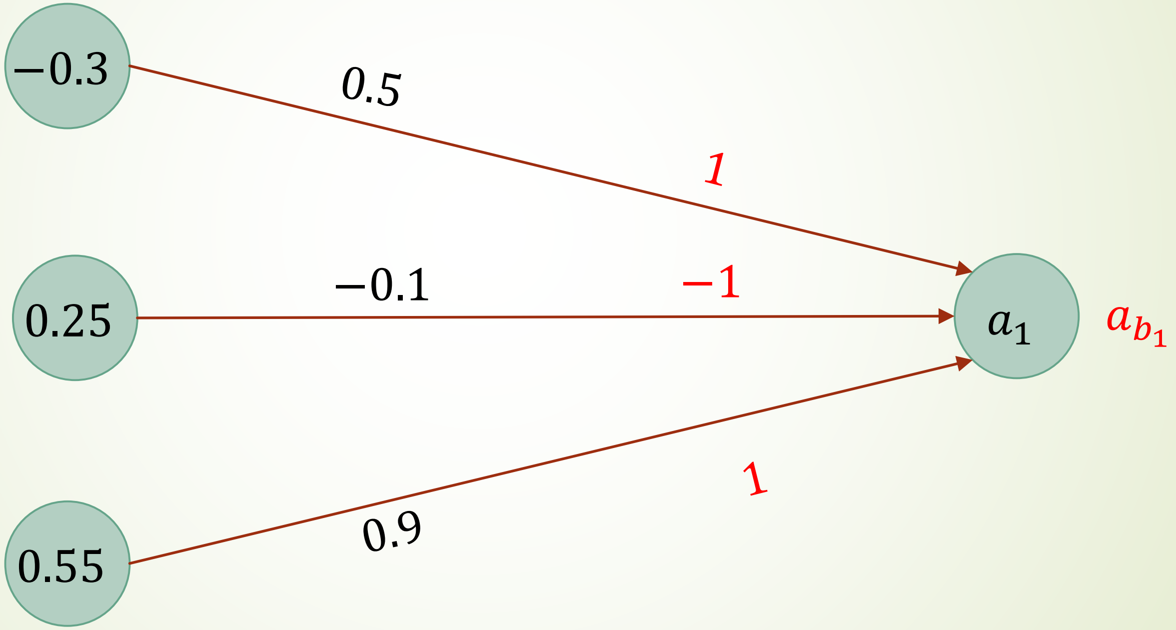
$$a_1 = ReLU(w_0 \cdot x_0 \;+ w_1 \cdot x_1 \;+ w_2 \cdot x_2 \;)$$

$$a_{b_1} = ReLU(\underbrace{w_{b_0}}_{\text{Sign}} x_0 + \underbrace{w_{b_1}}_{\text{Sign}} x_1 + \underbrace{w_{b_2}}_{\text{Sign}} x_2)$$

No multiplications required

Idea

- Binarization of weights



$$a_1 = ReLU(0.5 \cdot (-0.3) - 0.1 \cdot 0.25 + 0.9 \cdot 0.55 \quad) = 0.32$$

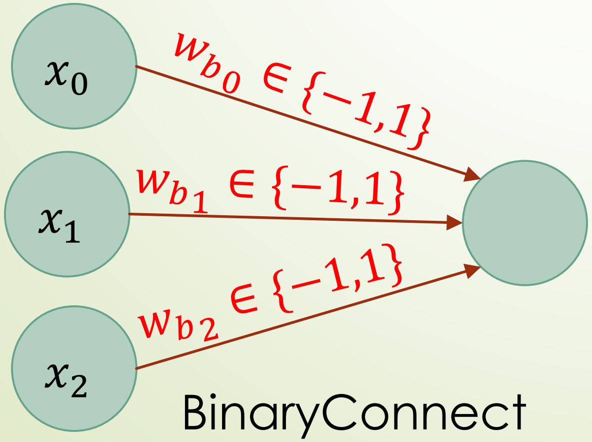
$$a_{b_1} = ReLU(-0.3 \; - 0.25 + 0.55\,) = 0$$

Ingredient #1 – Update w, not w_b

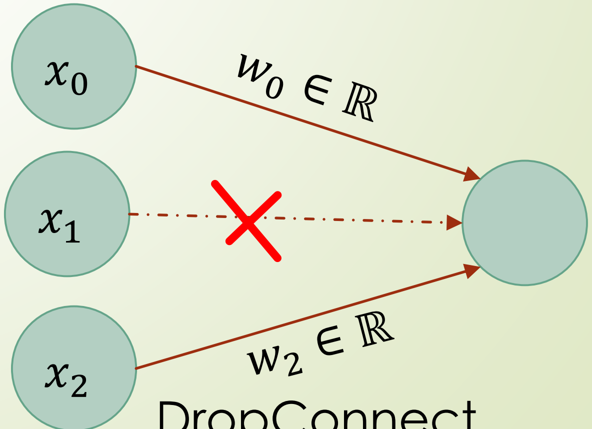
- $w_b \in \{-1,1\}$ are obtained from w
- Forward/backward pass uses $w_b \in \{-1,1\}$
- Only the real valued $w \in \mathbb{R}$ are updated
- **Important in order to catch the small contributions of the sub-gradients of SGD**

Ingredient #2 - Regularization

- Similarily to DropConnect, BinaryConnect alters the weights, but via binarizing them to $\{-1,1\}$ instead of zeroing part of the weights each time.
- This can be referred to as a “quantization noise” or a **regularization**



BinaryConnect



DropConnect

BinaryConnect

In detail

Deterministic vs stochastic binarization

- Deterministic:

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

- Stochastic

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

$$\sigma(w) = \max\left(0, \min\left(1, \frac{x+1}{2}\right)\right)$$

BinaryConnect 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 $\frac{\partial C}{\partial a_L}$

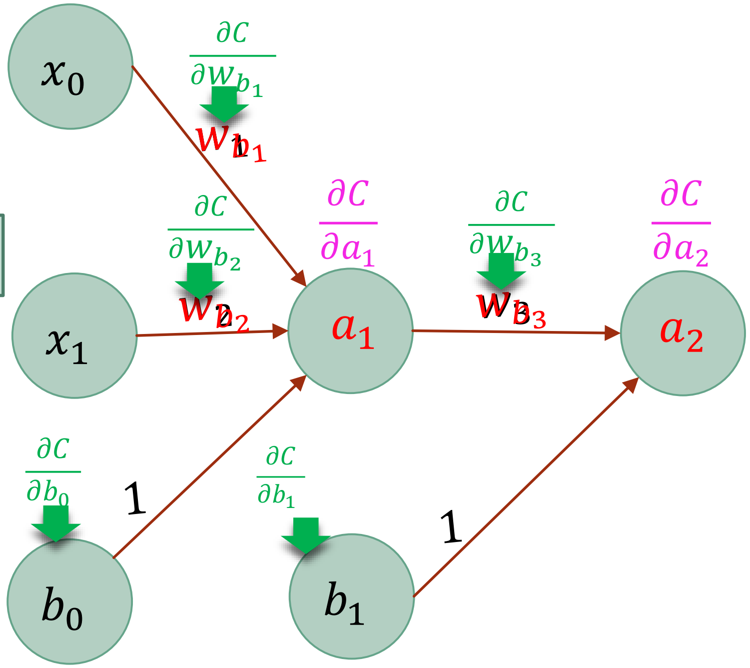
For $k = L$ to 2, compute $\frac{\partial C}{\partial a_{k-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and w_b

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}

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

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

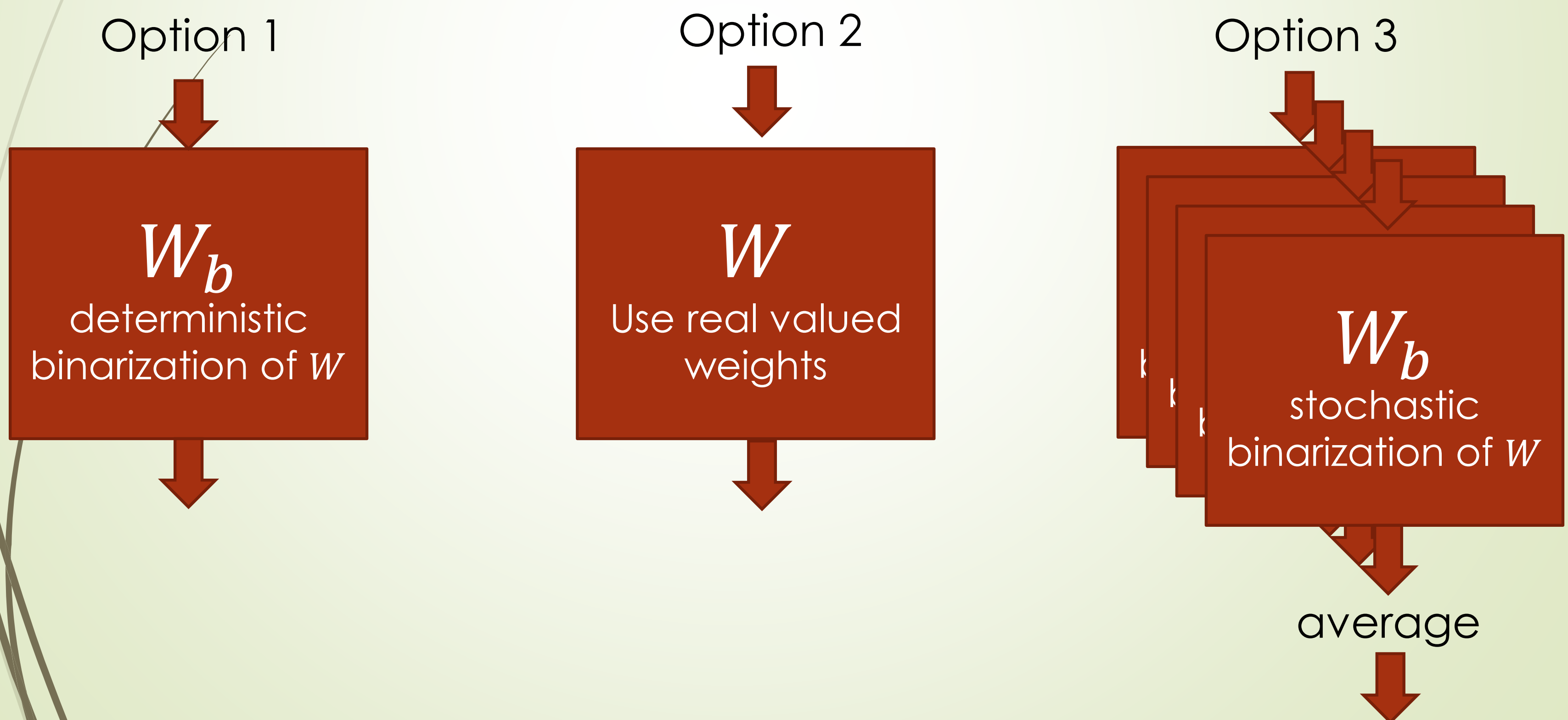


Training - Weight Clipping

- **Observation:** Binarization operation is not influenced by w with magnitude beyond 1.
- **Problem:** w might grow very large without influencing w_b
- **Solution:** The real valued w are clipped to $[-1, +1]$ upon updates
- This is yet another weight **regularization**

Test-Time Inference

- So far we trained DNN with on-the-fly binarization
- Result: we have the **trained real valued W**
- How do we test this DNN on **new examples**?



Experiments

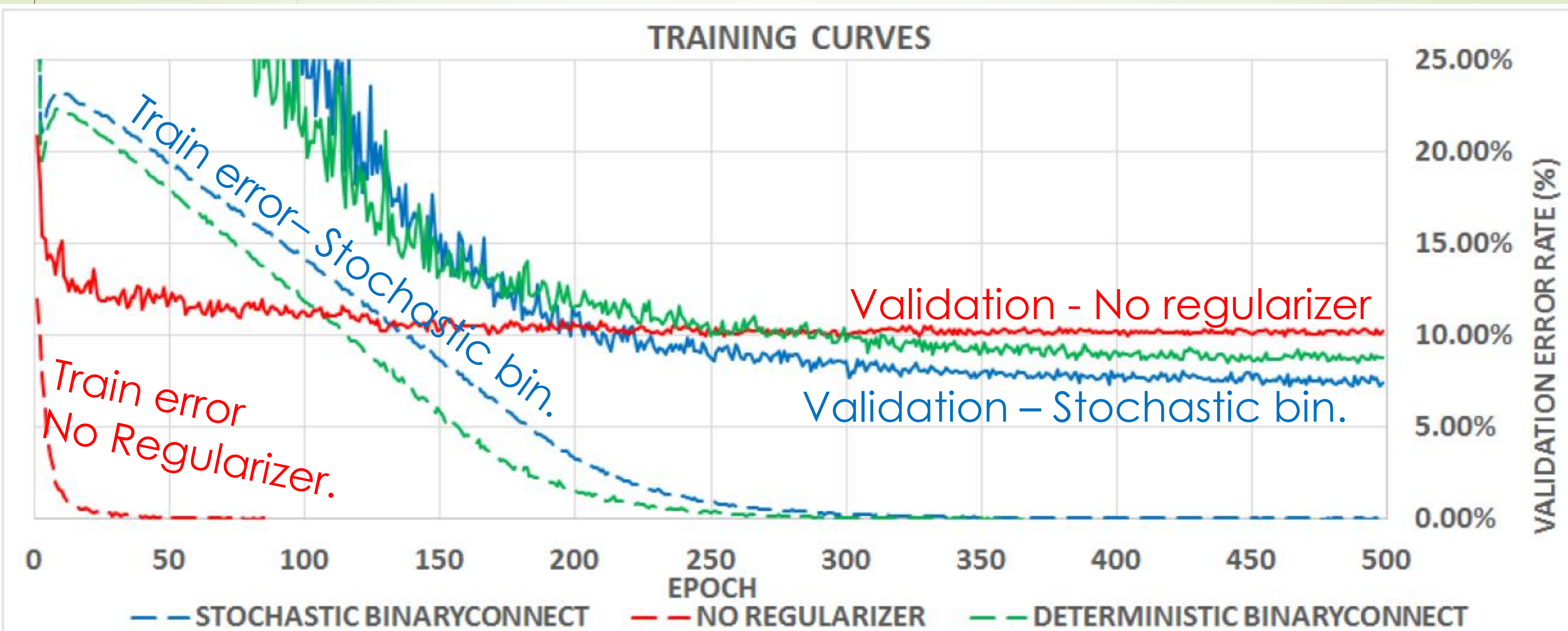
- **MNIST** – 3 FC layers + SVM
- **CIFAR-10, SVHN** – 6 CONV layers + 2 FC +SVM

W_b

$W \in \mathbb{R}$

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%**

Regularization contribution

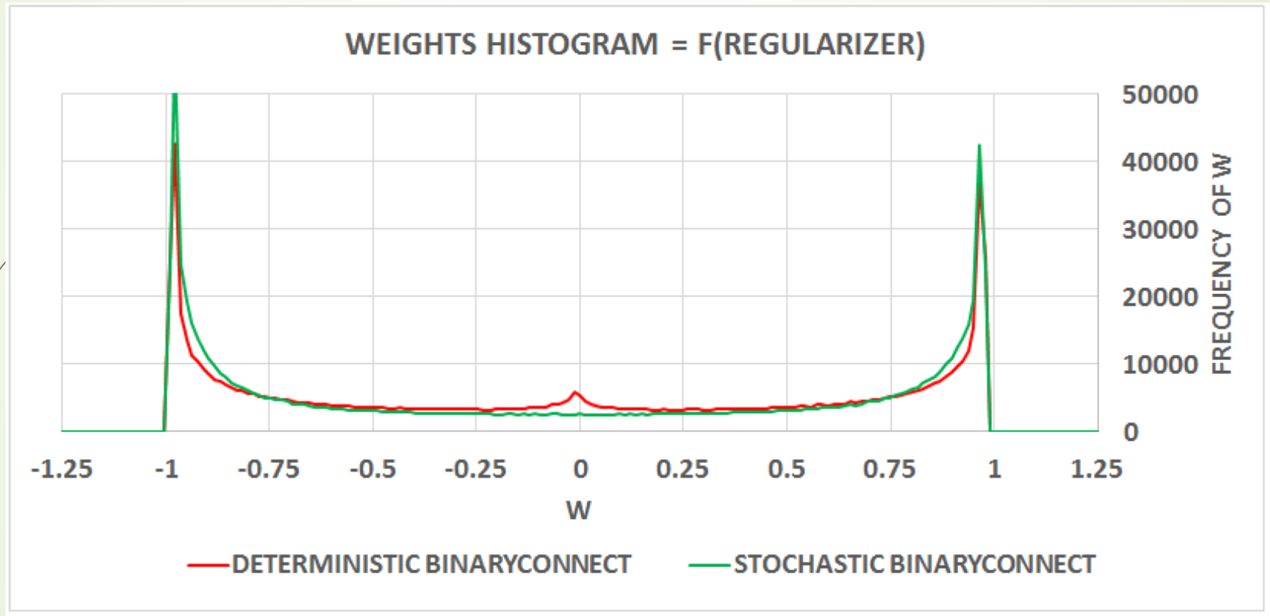


(CNN of CIFAR-10)

- Train Error and Time increases
- Validation Cost decreases
- Similar to the effect of Dropout

Training accomplished…

- How are the final weights distributed?



(FC network of MNIST)

- The weights are trying to become deterministic to reduce error

Similar Approaches

- **Ternary Connect** (SiPS 2014 Hwang et al.)
  - Train only using $w \in \mathbb{R}$
  - Ternarize the weights to $w_T \in \{-H, 0, H\}$ where H is adjusted to minimize error
  - Re-train using w_T during propagation and w for updates
- **BinaryNet (BNN)** (2016 Courbariaux *et al.*)
  - Binarize weights and activations

BinaryConnect Summary

- Weights are binarized during forward and backward propagations
- Acts as a regularizer, preserves accuracy
- Training - $\approx$2/3 of the multiplications removed
- Test time - getting rid of the multiplications altogether
- Test time – memory bandwidth reduction $\times$ 16 (16-bit floating point → 1-bit)
- Theano + Lasagne code available: https://github.com/MatthieuCourbariaux/BinaryConnect

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks

First presented at ECCV 2016 by

Mohammad Rastegariy

Vicente Ordonezy

Joseph Redmon

Ali Farhadi

Today presented by

Kostya Berestizshevsky

From Google Ngram-Viewer

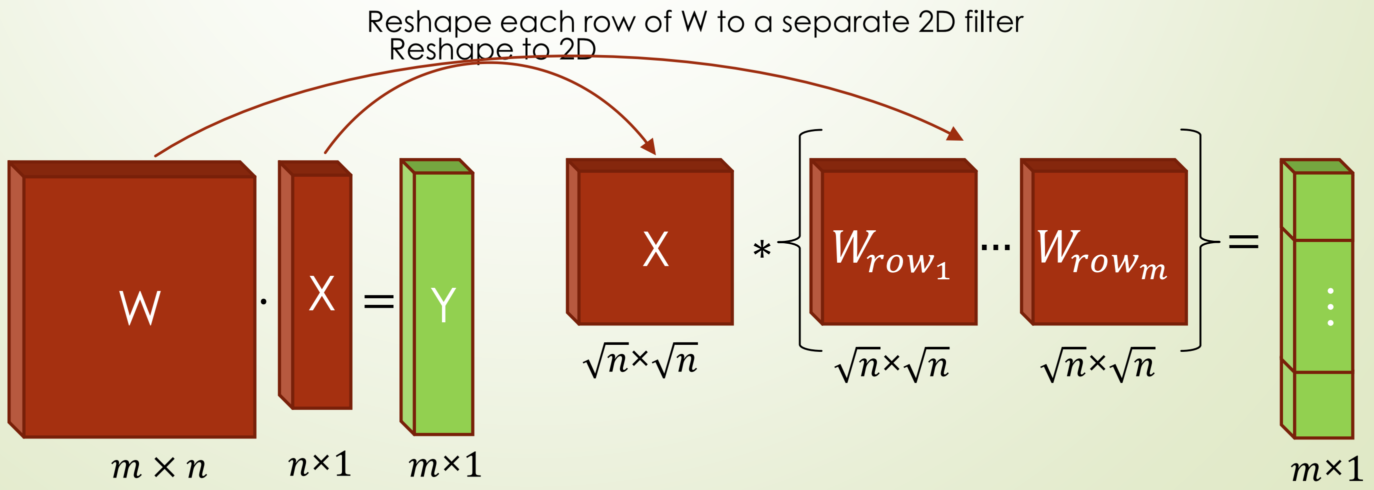
Highlights

- Try binarization on **large datasets**
- Two new variations tested on ImageNet:
  1. Binary-Weight-Nets <bin weights only>
  2. XNOR-Nets <bin. weights & inputs>
- Lower run-times and memory
  (relative to full-precision)
- Higher accuracy
  (relative to former binary methods)

Interesting Observation

- FC layer with n inputs and m outputs (weight matrix of a size $m \times n$)

- CONV layer with m filters of a size n



Binary CNN 1st variation: Binary Weight Networks

- W - real valued weights (filter)
- I - real valued input tensor
- * is a convolution operation

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

- B – binary weights (the sign of W)
- $\oplus$ is a convolution using only add/sub operations
- α – real valued scale factor (the average of $|W|$)

Proof

Why XNOR + bitcount approximates MAC ?

I	W	$I \odot W$	$XNOR(I,W)$
-1	-1	1	1
-1	1	-1	-1
1	-1	-1	-1
1	1	1	1

$I \cdot W$	$bitCount(XNOR(I,W))$
0	0

Binary CNN 2nd variation: XNOR Networks

- W - real valued weights (filter)
- I - real valued input tensor
- * is a convolution operation

$$\boxed{I * W \approx (H \circledast B) \odot \alpha K}$$

- H – input binary tensor (the sign of I)
- B – binary weights (the sign of W)
- $\circledast$ is a convolution using only XNOR and bit-counting
- α – real valued scale factors (the average of |W|)
- K – real valued scale factors of input conv-windows
- $\odot$ is elementwise multiplication

XNOR-Net Convolution Example

I – input tensor

0.1	0.8	0.7	0.5
0.3	-0.9	-0.9	-0.8
0.5	-0.2	-0.4	-0.5
0.9	-0.4	1	0.8

W filter

-0.4	-0.5
1	0.8

$\alpha = 0.225$

$K =$

0.525	0.825	0.725
0.475	0.6	0.65
0.5	0.5	0.675

$I * W$

-0.86	-2.29	-2.07
0.67	0.29	-0.04
0.48	0.68	2.05

H – binarized input tensor

1	1	1	1
1	-1	-1	-1
1	-1	-1	-1
1	-1	1	1

B binarized filter

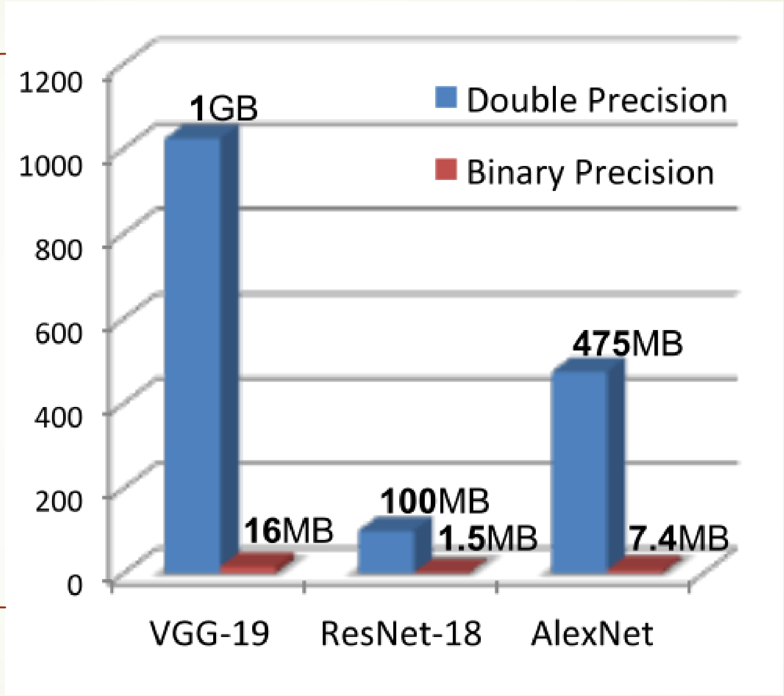
-1	-1
1	1

$H \circledast B \odot K\alpha$

-0.71	-2.23	-1.96
0	0	0
0	0.68	1.82

Binary weights reduce memory

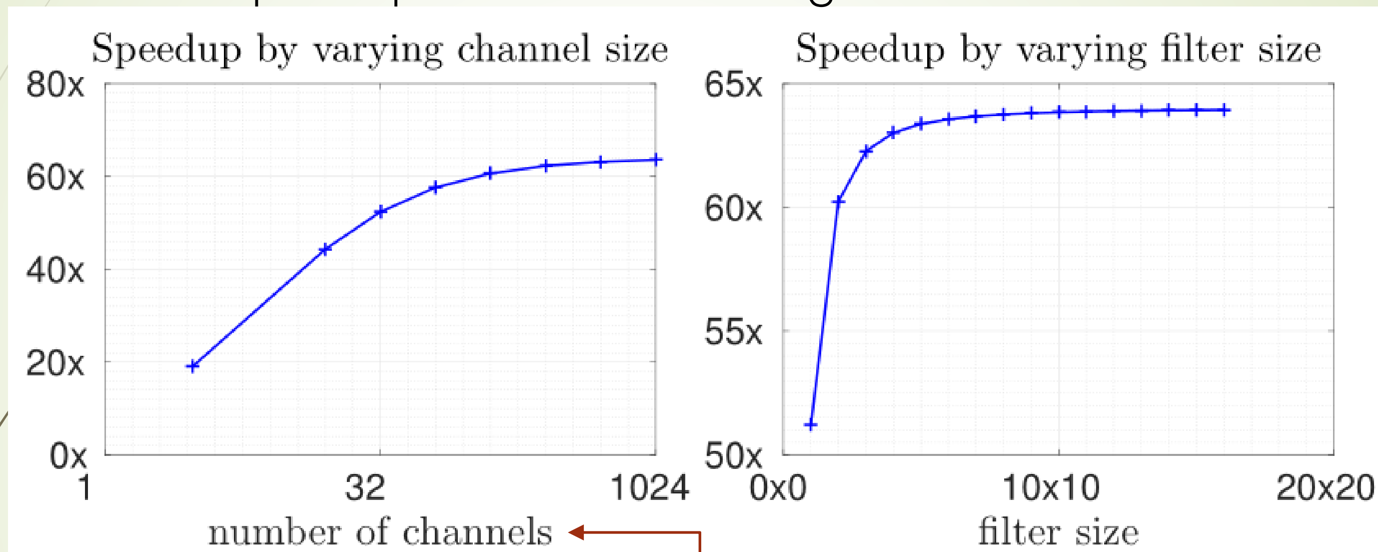
Required memory for weights storage [MB]



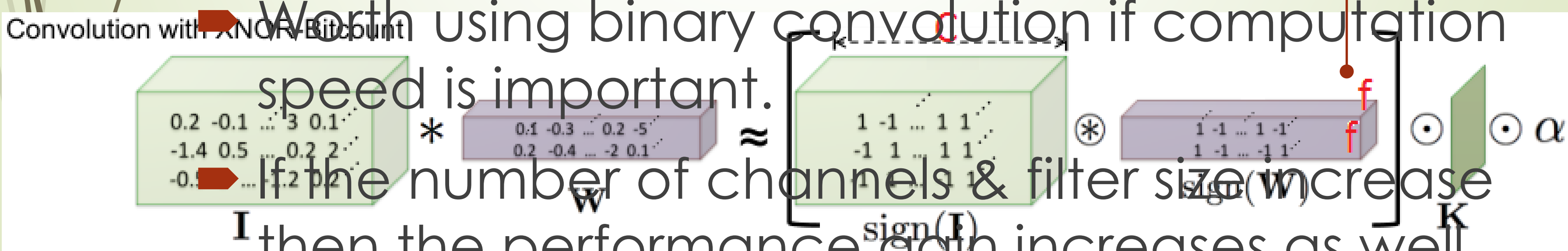
- Binary-weight-networks are so small that can be easily fitted into portable devices

Speedup using XNOR-bitcount

The speedup is relative to the regular convolution



- Worth using binary convolution if computation speed is important.
- If the number of channels & filter size increase then the performance gain increases as well.



Accuracy comparison

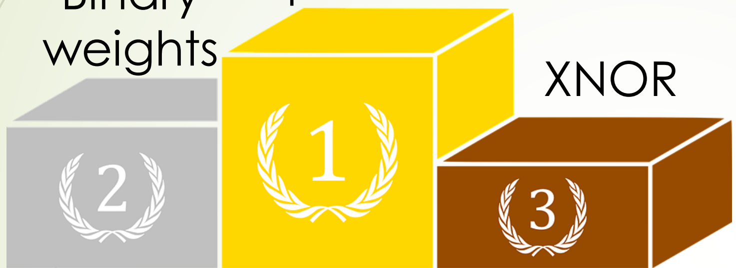
AlexNet Architecture

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

Other Architectures

	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

Accuracy / Tradeoff summary

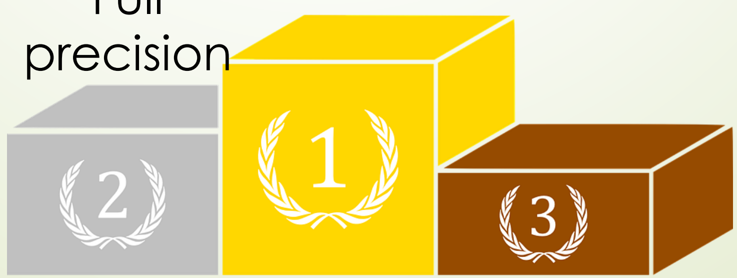


Other binary nets – poor performance

Speed

x2 x58 x1

Memory



x1 ~x60 (64 bit representation)

Conclusions

- Binary-Weight-Nets provide a good compromise of Memory & Speed vs. Accuracy
- XNOR-Nets provide supreme speedup at a cost of accuracy (The authors opened a startup!)

	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

- Torch 7 code available:
  `https://github.com/allenai/XNOR-Net`

Dedicated Hardware for Machine Learning

(Bonus)

ML Hardware Research

- Dedicate hardware for training or inference?
- Optimize Throughput? [Gop/sec]
- Optimize Area? [Gop/sec/mm^2]
- Optimize Power? [Gop/sec/Watt]
- ASIC / FPGA?

Intermediate Conclusions

- Memory is the bottleneck.
- Weights $\in \mathbb{R}$: 32-bits for training,16-bits for infer.
- The design must be general enough – since the ML algorithms are changing rapidly,

Academic works

- **CNP'09, Neuflow'11, nn-X'14 – FPGA** *[Farabet, Gokhale et al.]*

  Low power of 4W yet hight throughput of 200GOp/sec

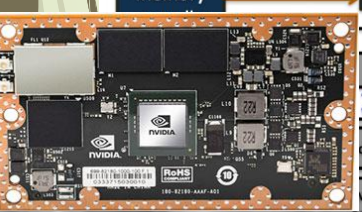
- **DaDianNao – ASIC** *[Chen, Temam et al.]*

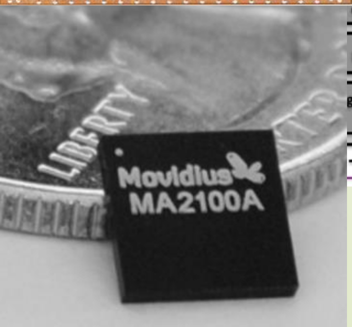
  Focused on a smart memory placement around processing elements

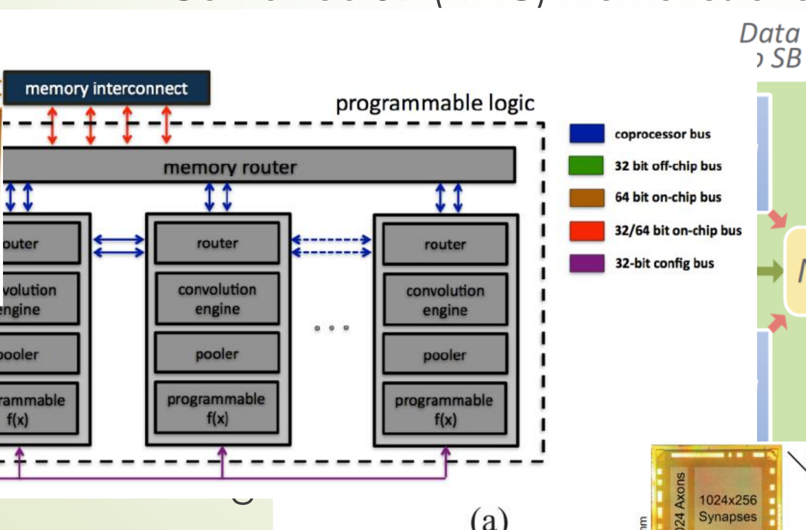
  x150 power reduction and x450 speedup relative to K20 GPU

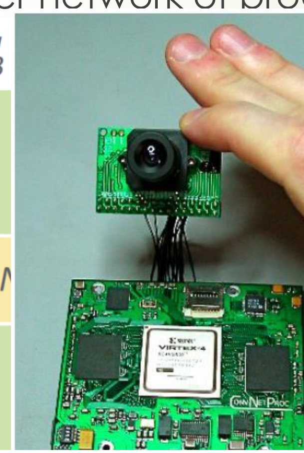
- **Neurocube'16 – ASIC** *[Kim et al.]*

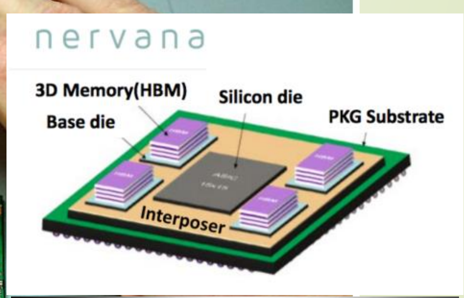
  Combined 3D (HMC) memories over network of processing elements

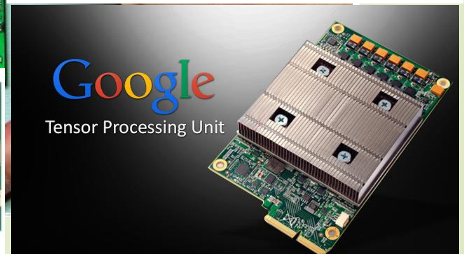




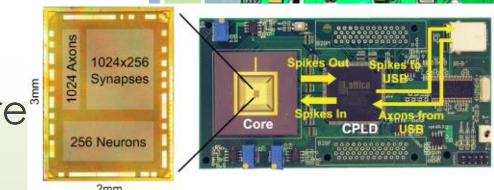








- IBM – Neurosynaptic core



Thank You!

Any Questions?

References – Binary DL

- Matthieu Courbariaux, Yoshua Bengio, Jean-Pierre David:
  BinaryConnect: Training Deep Neural Networks with binary weights during propagations. NIPS 2015: 3123-3131
- Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, Ali Farhadi:
  XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. ECCV (4) 2016: 525-542
- Matthieu Courbariaux, Yoshua Bengio:
  BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. CoRRabs/1602.02830 (2016)
- K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and −1," *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, Belfast, 2014, pp. 1-6.

References - Hardware

- C. Farabet, C. Poulet, J. Y. Han and Y. LeCun, "**CNP**: An FPGA-based processor for Convolutional Networks," *2009 International Conference on Field Programmable Logic and Applications*, Prague, 2009, pp. 32-37.
- C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello and Y. LeCun, "**NeuFlow**: A runtime reconfigurable dataflow processor for vision," *CVPR 2011 WORKSHOPS*, Colorado Springs, CO, 2011, pp. 109-116.
- Gokhale, Vinayak, et al. "A 240 g-ops/s mobile coprocessor for deep neural networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014.
- Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2014. **DaDianNao**: A Machine-Learning Supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*
- D. Kim, J. Kung, S. Chai, S. Yalamanchili and S. Mukhopadhyay, "**Neurocube**: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, 2016, pp. 380-392.