

Network Binarization

3-29
uoguelph-mlrg
He Ma

Hi everyone, Today I'm going to give a quick review of recent progress on network binarization of Convolutional Neural Networks. This presentation is mainly based on two papers:

XNOR-Net
BinaryNetworks

Contents

- Why binarizing a CNN?
- How to binarize a CNN?
 - related works and performance
- Summary (three levels of binarization)

First, let's take a quick review of how standard full precision CNNs are inference and updated. Since CNNs are stacked by Convolutional and Pooling layers. Let's look at a Convolutional Layer first.

Why binarizing networks?

- To run CNNs on target portable devices
 - low power, low memory
- To speed up CNNs run-time (inference time)
- To explore to what extent CNN depends on high precision
- To explore binary as a kind of regularization
 - a variant of Dropout(BinaryConnect)

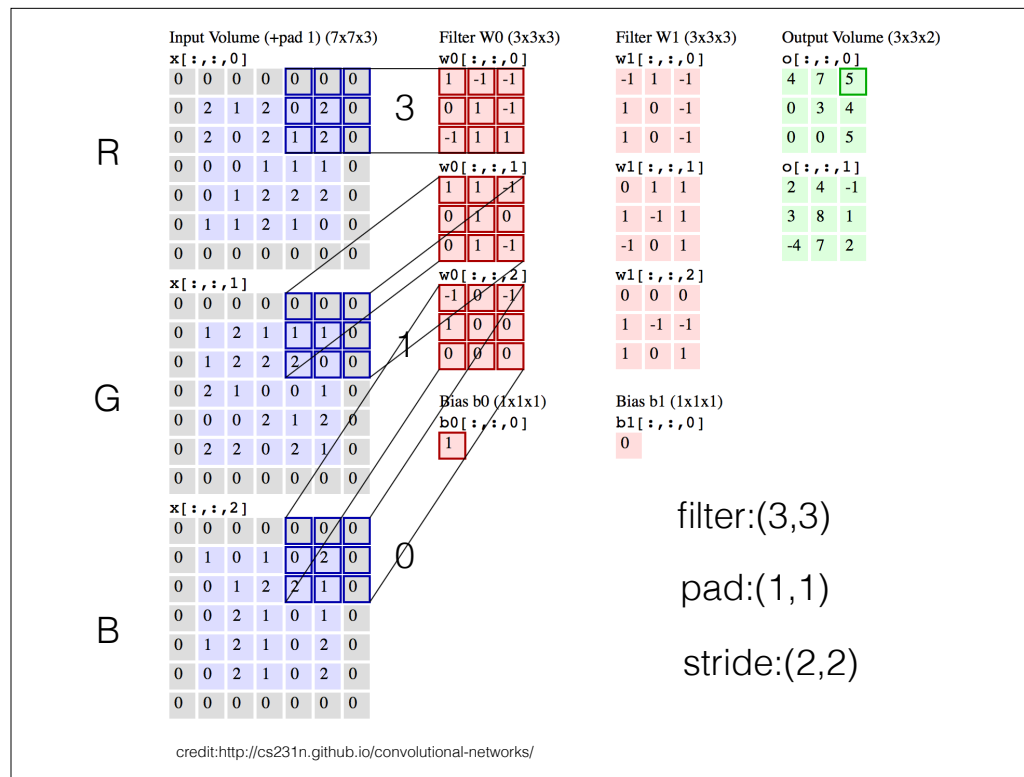
Recent progress in Convolutional Neural Networks shows that they can be trained and do a good job in some complicated tasks, like large scale image recognition. And we want deploy this growing ability of CNN into robots and portable devices. However, the device memory, power and time needed to inference those model is too much for robotic systems and portable devices, let alone to train the model on those systems. To be able to do this, we need to either increase the computing capability of those portable devices or lower the power, memory and time usage of the CNN model.

Recent progress shows that it is possible to lower those three usage of CNNs by binarizing it without losing much recognition accuracy.

How: A ConvOp Example

- input image: (7,7)
- channels: 3
- filter:(3,3)
- pad:(1,1)
- stride:(2,2)

First let us review some basics of Convolution. In the next slide, I will show an example of a full precision Convolution Operation with the following size.

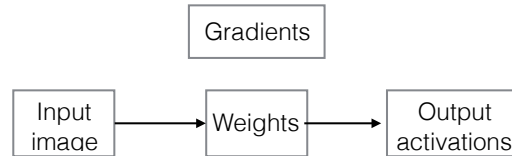


Here shows how a standard Conv layer works on full precision parameters.

This demo is taken from the Stanford ConvNet course website

The weights in real CNNs are usually float32 numbers which produce float numbers in the output.

Full precision Conv Op



full precision number in the graph:

- input features (or images)
- weights (and biases)
- output activation
- gradients

sub operations:

- matrix multiply
- sum

The standard full precision Conv Operation consists of two sub operations:
matrix multiplication and summation

The computing graph requires all operands to be float32. Literature shows that it is important to keep those parameters in high precision to ensure the accuracy of the model. However, which parameter is more important to keep in high accuracy is not clear.

BNN paper explains that “SGD explores the space of parameters in small and noisy steps and that noise is averaged out by stochastic gradient contributions accumulated in each weight. Because of which, it is important to keep sufficient resolution for these accumulators.”

Related Works

- M Courbariaux et al 2015 Nov: **BinaryConnect**: Training Deep Neural Networks with binary weights during propagations
- Z Lin et al 2016 Feb: Neural Networks with Few Multiplications
- M Courbariaux et al 2016 Mar: **Binarized Neural Networks**: Training Neural Networks with Weights and Activations Constrained to +1 or -1
- M Rastegari et al 2016 Mar: **XNOR-Net**: ImageNet Classification Using Binary Convolutional Neural Networks

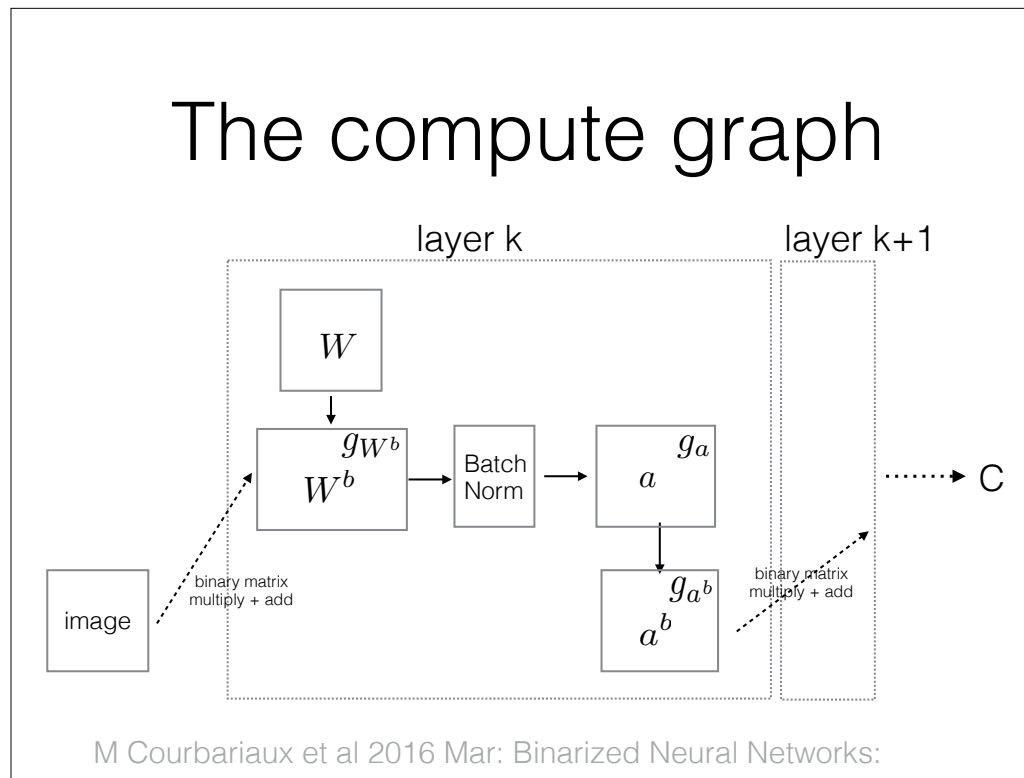
In recent two years there has been some progress in reducing the precision of those parameters.

BinaryConnect shows it is possible to make weights in binary and model can still perform well on small image datasets.

BNN shows that the output activation can also be binarized to further speed up the training and inference of CNN.

XNOR-Net extends BNN's idea, binarized weights and activations with an additional scale factor and developed faster approximation of the Binary version of Convolutional Operator. Also they adjusted the order of operations inside a ConvPoolBNLayer and get better accuracy than BNNs and can approximate well even on large image datasets.

The compute graph



Now let me show their proposed method to binarize weight and activations. In this graph, the dotted box is...
 training time: forward pass use W^b , backward pass get g_{W^b} , use g_{W^b} to update W .

run-time only use W^b

fixed point dedicated hardware can reduce time by 60%

BinaryConnect only use W^b , BNN uses W^b and a^b . In the BNN paper they developed an efficient Binary Convolution kernel, the binary matrix multiplication kernel is further improved by XNOR-bitcount, XNOR also rearrange the order to BN, BinActiv, BinConv AND Pool.

Binarize weight

- Using a sign() function

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

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

$$W_t = \text{update}\left(\frac{\partial C}{\partial W_t^b}, W_{t-1}\right)$$

- Clip W within (-1, 1) right after update

M Courbariaux et al 2015 Nov: BinaryConnect

Here shows the details of how they binarize their weight.

Where the sign function can be a stochastic rather than deterministic.

its gradient can be calculated in back-propagation.

Update W with Wb's gradient and W from last iteration with Momentum SGD or ADAM.

Binarize activation

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

$$\frac{\partial C}{\partial a} = g_a = g_a^b \circ 1_{|a| \leq 1}$$

Y Bengio 2013: Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation

M Courbariaux et al 2016 Mar: Binarized Neural Networks:

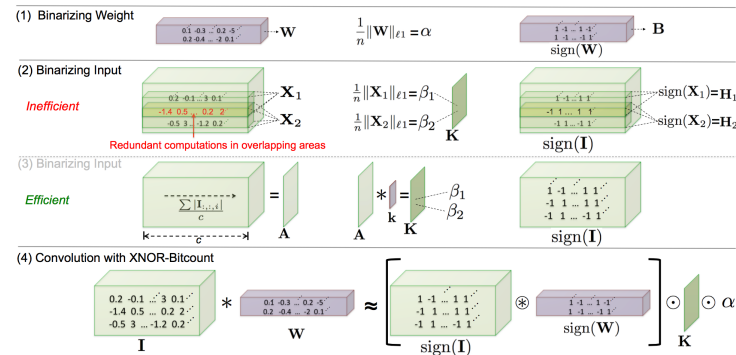
in his new paper this year, he proposed binarizing activations as well making the whole network computing in binary except that the input image at the first layer is not and convolution operates on binary numbers which is not efficient.

straight through estimator was introduced in Hinton's lecture in 2012:

1 if the argument is positive, 0 otherwise

Binarize input image

- Avoiding redundant computations
- Efficient for a **Binary Conv Op**



M Rastegari et al 2016 Mar: XNOR-Net

The difference between XNOR-Net and the BNN is XNOR-Net tries too further optimize the Binary Convolution Operation, and to do this they first estimate the operand of this operation

Approximate Conv Op with Binary Op

$$I * W \approx I * W^b \quad (\text{BC})$$

$$I * W \approx I^b * W^b \quad (\text{BNN})$$

M Courbariaux et al 2016 Mar: Binarized Neural Networks:

A coarse binary approximation, a regularization
use XNOR kernel, 7 times faster to train MNIST, without accuracy loss
32 times memory saving during inference

Approximate Conv Op with Binary Op

$$I * W \approx (I \oplus B)\alpha$$

$$I * W \approx (I \oplus \text{sign}(W))\alpha \quad (\text{BWN})$$

$$I * W \approx (\text{sign}(I) \circledast \text{sign}(W)) \odot K\alpha \quad (\text{XNOR-NET})$$

M Rastegari et al 2016 Mar: XNOR-Net

First Op is a convolution without any multiplication but input operand is not binary

Binary Conv Op is an better approximated Conv Op without any multiplication

B is binary tensor proved to be $\text{sign}(W)$

alpha is the scaling factors for W, real scalar proved to be average of W

K is average of I over channels convolved with k, an 2d filter. Basically K contains the scaling factors for all sub-tensors in input I.

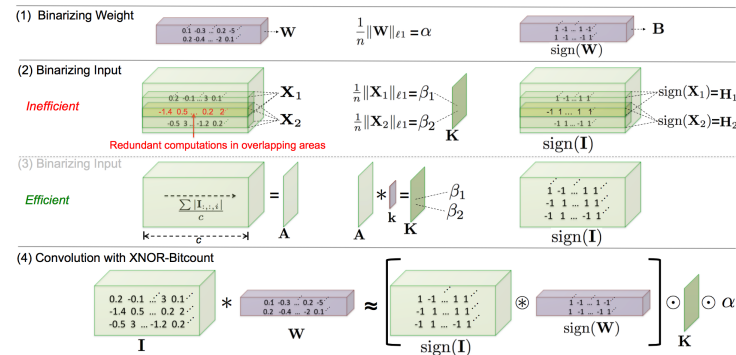
use XNOR and bit count

and proof why $\text{sign}(W)$ is a reasonable approximation of W

32 times memory saving, 58times CPU time saving during inference

Binarize input image

- Avoiding redundant computations
- Efficient for a **Binary Conv Op**



M Rastegari et al 2016 Mar: XNOR-Net

$$n = c * w * h$$

Convolving W with I involves calculating $W \cdot X_1, W \cdot X_2 \dots$ producing a bunch of vectors and forms a 3-d input feature map matrix for the next layer.

like the procedure of approximating W to a $\text{sign}(W)$ times α , each input image local trunk X is approximated to $\text{sign}(X)$ times β .

$$\text{So } X^T W \approx \beta \text{sign}(X)^T \cdot \alpha \text{sign}(W)$$

Since there are many trunk X in input I , for each trunk there's a corresponding β , and all these β gathered into a matrix K .

$$\text{So } I * W \approx \text{sign}(I) \cdot \text{sign}(W) \cdot K \cdot \alpha$$

But preparing each β individually consists some redundant computation. An equivalent and efficient way is to averaging input I across channel and convolving with a smaller $k=1/(\text{width} * \text{height})$.

Performance

- **Speed up**
- **Memory**
- **Accuracy**

Performance: inference speed up

- on GPU[1]:

The XNOR kernel runs 3.4 times faster than cuBlas MM

- on CPU[2]:

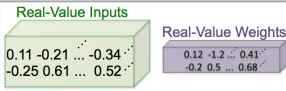
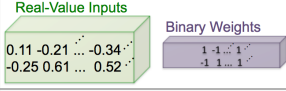
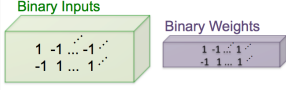
The XNOR kernel runs 58 times faster when channel larger than 3 and kernel are larger than 1x1 .

[1] M Courbariaux et al 2016 Mar: Binarized Neural Networks:

[2] M Rastegari et al 2016 Mar: XNOR-Net

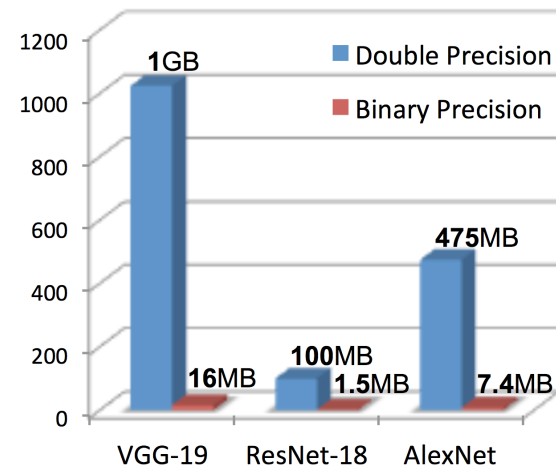
This means when floating point network is binarized, CPU can be fast for inference as well. suitable for portable devices.

Performance: inference memory saving

	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Time Saving on CPU (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	 <p>Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52</p> <p>Real-Value Weights 0.12 -1.2 ... 0.41 -0.2 0.5 ... 0.68</p>	+ , - , ×	1x	1x	%56.7
Binary Weight	 <p>Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52</p> <p>Binary Weights 1 -1 1 -1 1 -1</p>	+ , -	~32x	~2x	%53.8
BinaryWeight Binary Input (XNOR-Net)	 <p>Binary Inputs 1 -1 -1 -1 1 1</p> <p>Binary Weights 1 -1 1 -1 1 -1</p>	XNOR , bitcount	~32x	~58x	%44.2

M Rastegari et al 2016 Mar: XNOR-Net

Performance: inference memory saving



M Rastegari et al 2016 Mar: XNOR-Net

may include the memory of feature maps

Performance: accuracy

- small dataset:

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%

M Courbariaux et al 2016 Mar: Binarized Neural Networks:

Performance: accuracy

- large dataset:

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
53.8	77.0	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

M Rastegari et al 2016 Mar: XNOR-Net

Resolved and remaining issue

- Resolved:
 - weight binarization (BinaryConnect)
 - activation binarization (BinaryNet)
 - taking gradient of sign() function (BinaryNet)
 - input feature or image binarization (XNOR-Net)
 - efficient convolution on binary weight(XNOR-Net)
- Remaining
 - save full precision weights during training
 - full precision gradient

Here I show made a simple summary of those resolved problems and unresolved.

Taking gradient of sign() function I did

not understood well

Summary

- three levels of binarization

Summary

- main contribution