TYPES OF MODEL COMPRESSION

Soham Saha,
MS by Research in CSE,
CVIT, IIIT Hyderabad

1. Pruning
2. Quantization
3. Architectural Modifications

PRUNING

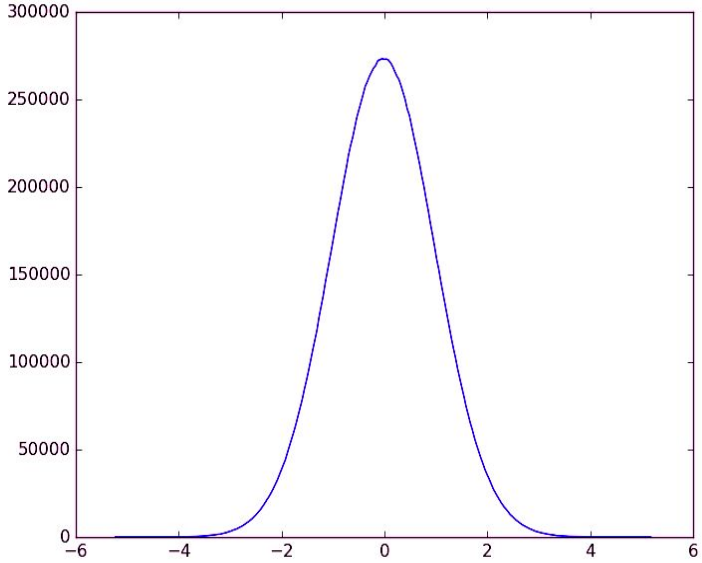
WHY PRUNING ?

Deep Neural Networks have redundant parameters.

Such parameters have a negligible value and can be ignored.

Removing them does not affect performance.

Figure: Distribution of weights after Training

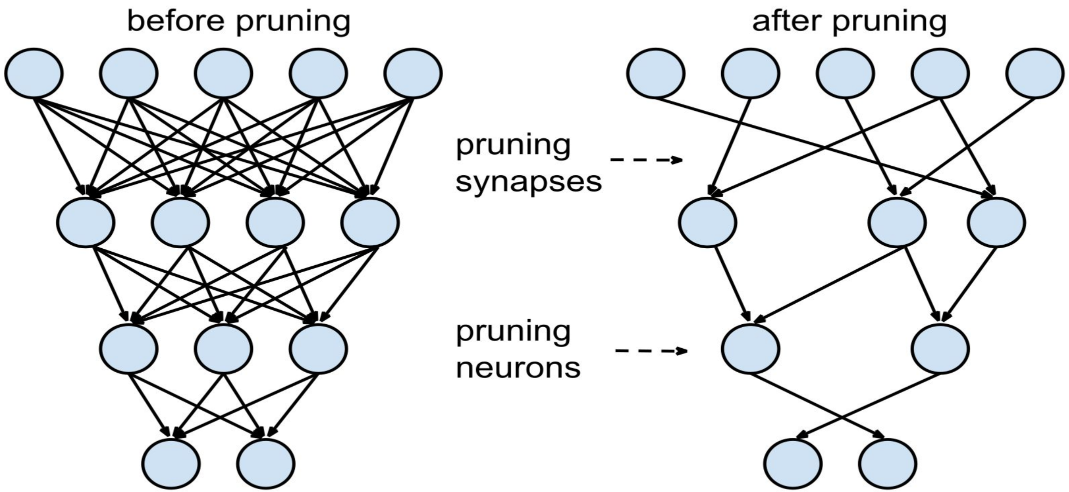


TYPES OF PRUNING

- Fine Pruning
  -- Prune the weights
- Coarse Pruning
  -- Prune neurons and layers
- Static Pruning
  -- Pruning after training
- Dynamic Pruning
  -- Pruning during training time

Weight Pruning

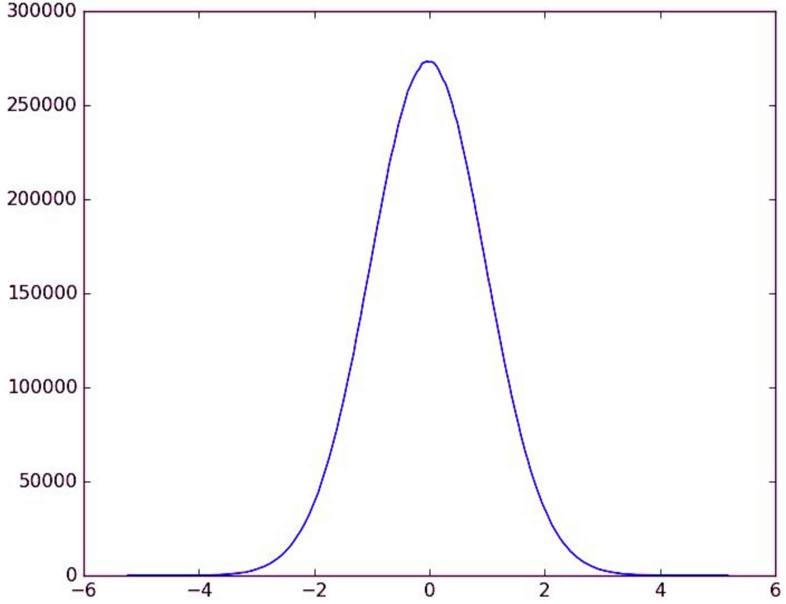
(After Training)

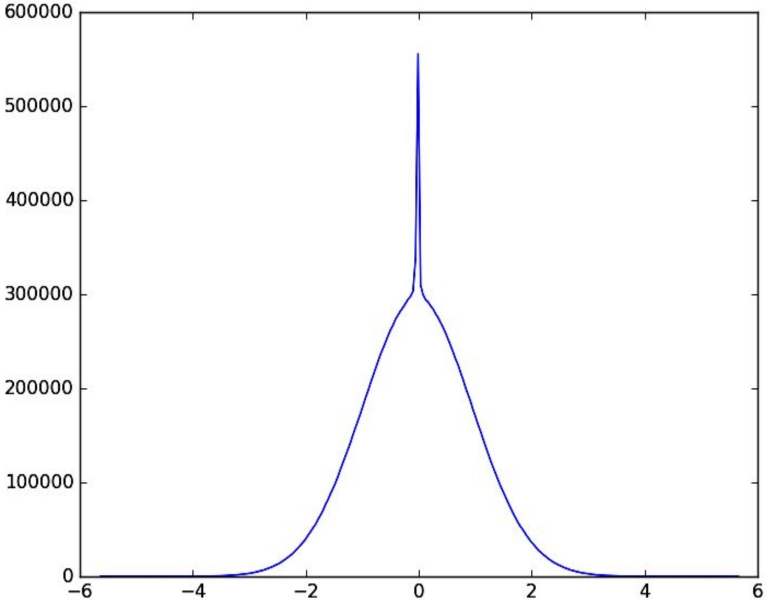


- The matrices can be made sparse. A naive method is to drop those weights which are 0 after training.
- Drop the weights below some threshold.
- Can be stored in optimized way if matrix becomes sparse.
- Sparse Matrix Multiplications are faster.

Ensuring Sparsity

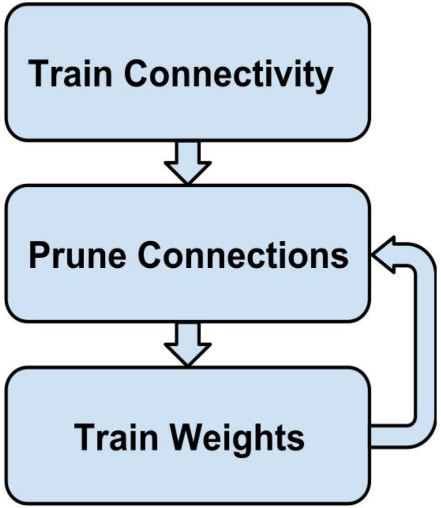
Addition of L1 regulariser to ensure sparsity

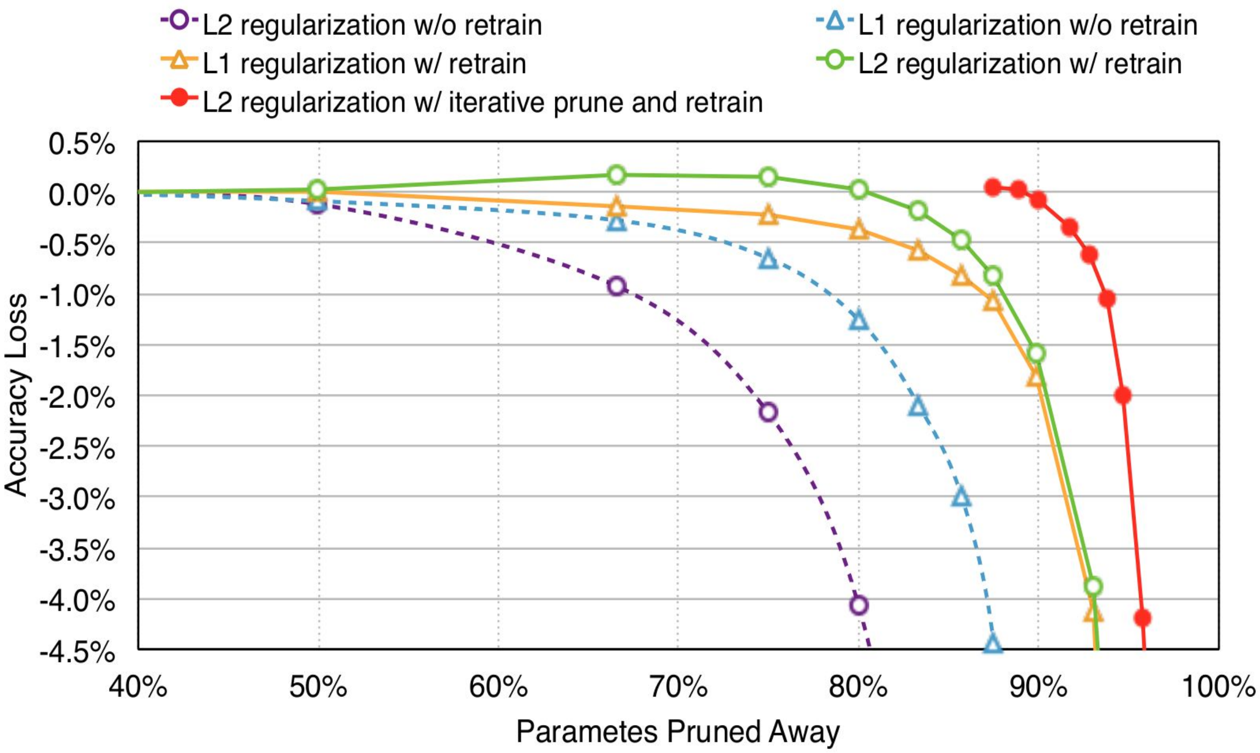




Sparsify at Training Time

Iterative pruning and retraining





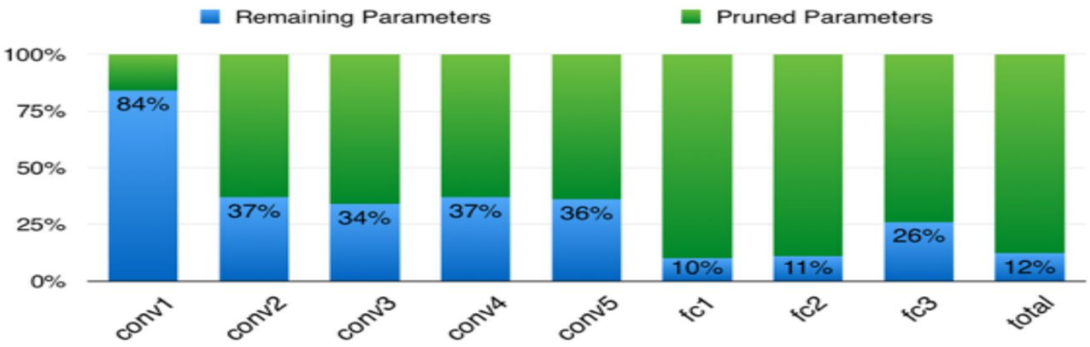
DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING Song Han, Huizi Mao, William J. Dally

Results reported by Deep Compression

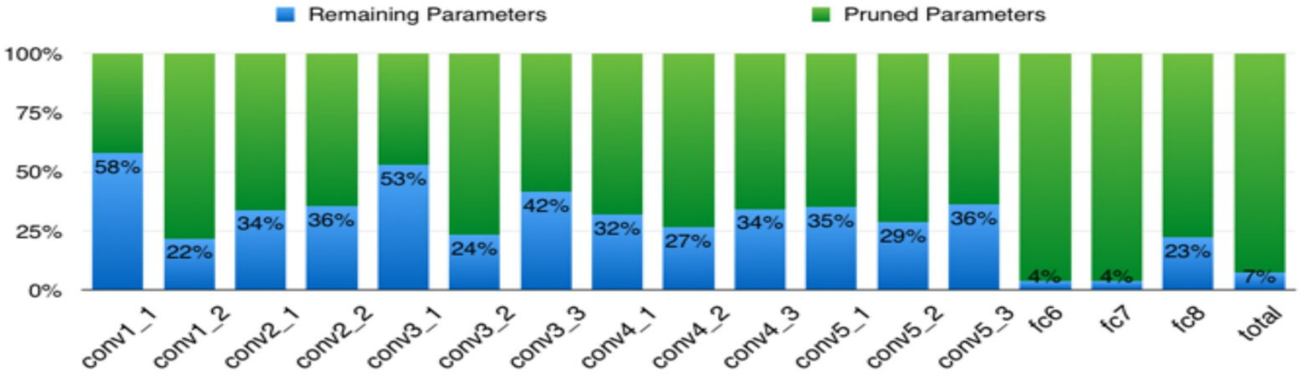
Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12**$\times$
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12**$\times$
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9**$\times$
VGG16 Ref	31.50%	11.32%	138M	
VGG16 Pruned	31.34%	10.88%	**10.3M**	**13**$\times$

Table 1: Network pruning can save 9$\times$ to 13$\times$ parameters with no drop in predictive performance

Remaining parameters in Different Layers



ALEXNET



VGG16

DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING Song Han, Huizi Mao, William J. Dally

Comments on Weight Pruning

1. Matrices become sparse. Storage in HDD is efficient.
2. Same memory in RAM is occupied by the weight matrices.
3. Matrix multiplication is not faster since each 0 valued weight occupies as much space as before.
4. Optimized Sparse matrix multiplication algorithms need to be coded up separately even for a basic forward pass operation.

Neuron Pruning

- ➔ Previously, we had a sparse weight matrix.
- ➔ Now, we will be effectively removing rows and columns in a weight matrix.
- ➔ Matrix multiplication will be faster improving test time.
- ➔ Drop Neuron uses custom regularizers to prune neurons.
- ➔ Use thresholding to remove all connections of a neuron.

DropNeuron: Simplifying the Structure of Deep Neural Networks Wei Pan, Hao Dong, Yike Guo

Dropping Neurons by Regularization

$$\texttt{li\_regulariser} := \lambda_{\ell_i} \sum_{\ell=1}^{L} \sum_{j=1}^{n^\ell} \|\mathbf{W}^\ell_{:,j}\|_2 = \lambda_{\ell_i} \sum_{\ell=1}^{L} \sum_{j=1}^{n^\ell} \sqrt{\sum_{i=1}^{n^{\ell-1}} \left(W^\ell_{ij}\right)^2}$$

$$\texttt{lo\_regulariser} := \lambda_{\ell_o} \sum_{\ell=1}^{L} \sum_{i=1}^{n^{\ell-1}} \|\mathbf{W}^\ell_{i,:}\|_2 = \lambda_{\ell_o} \sum_{\ell=1}^{L} \sum_{i=1}^{n^{\ell-1}} \sqrt{\sum_{j=1}^{n^\ell} \left(W^\ell_{ij}\right)^2}$$

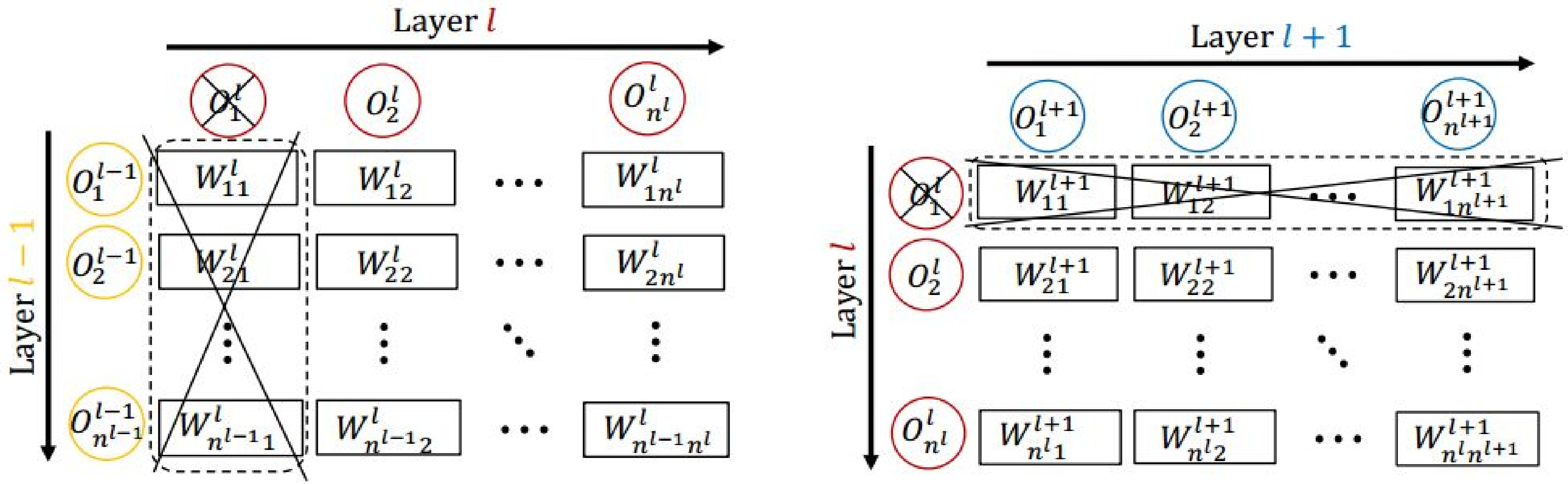
DropNeuron: Simplifying the Structure of Deep Neural Networks Wei Pan, Hao Dong, Yike Guo

Dropping principles

- All input connections to a neuron is forced to be 0 or as close to 0 as possible. (force li_regulariser to be small)
- All output connections of a neuron is forced to be 0 or as close to zero as possible. (force lo_regulariser to be small)
- Add regularisers to the loss function and train.
- Remove all connections less than threshold after training.
- Discard neuron with no connection.

DropNeuron: Simplifying the Structure of Deep Neural Networks Wei Pan, Hao Dong, Yike Guo

Effect of neuron pruning on weight matrices



(c) Removal of incoming connections to neuron O_1^ℓ, i.e., the group of weights in the dashed box are all zeros

(d) Removal of outgoing connections from neuron O_1^ℓ, i.e., the group of weights in the dashed box are all zeros

DropNeuron: Simplifying the Structure of Deep Neural Networks Wei Pan, Hao Dong, Yike Guo

Results on FC Layer (MNIST)

Table 3: Summary of statistics for the fully connect layer of LeNet5 (average over 10 initialisations)

Regularisation	$\mathbf{W}^{FC1}\%$	$\mathbf{W}^{FC2}\%$	$\mathbf{W}^{total}\%$	Accuracy	Accuracy (no prune)
DO+P	55.15%	62.81%	55.17%	99.07%	99.12%
ℓ_1+DO+P	5.42%	51.66%	5.57%	99.01%	98.96%
ℓ_1+DN+P	1.44%	16.82%	1.49%	99.07%	99.14%
Regularisation	$\mathbf{O}^{FC1}\%$	$\mathbf{O}^{FC2}\%$	$\mathbf{O}^{output}\%$	$\mathbf{O}^{total}\%$	**Compression Rate**
DO+P	$\frac{3136}{3136} = 100\%$	$\frac{504}{512} = 98.44\%$	$\frac{10}{10} = 100\%$	$\frac{3650}{3658} = 99.78\%$	1.81
ℓ_1+DO+P	$\frac{1039}{3136} = 33.13\%$	$\frac{320}{512} = 62.5\%$	$\frac{10}{10} = 100\%$	$\frac{1369}{3658} = 37.42\%$	17.95 [4]
ℓ_1+DN+P	$\frac{907}{3136} = 28.92\%$	$\frac{110}{512} = 21.48\%$	$\frac{10}{10} = 100\%$	$\frac{1027}{3658} = 28.08\%$	67.04

DropNeuron: Simplifying the Structure of Deep Neural Networks Wei Pan, Hao Dong, Yike Guo

Neuron and Layer Pruning

- Can we learn hyperparameters by Backpropagation?
  - Hidden Layer / Filter size
  - Number of layers
- We would actually be learning the architecture
- Modifying the activation function
- ‘w’ and ‘d’ are binary variables in the equation below.

$$tsReLU(x) = \begin{cases} wx, & x \geq 0 \\ wdx, & otherwise \end{cases}$$

Learning Neural Network Architectures using Backpropagation Suraj Srinivas , R. Venkatesh Babu

Loss Function

$$\theta, \mathbf{w}, \mathbf{d} = \underset{\theta, w_{ij}, d_i : \forall i,j}{\arg\min} \; \ell(\hat{y}(\theta, \mathbf{w}, \mathbf{d}), y) + \lambda_1 \sum_{i=1}^{m} \sum_{j=1}^{n_i} w_{ij}(1 - w_{ij}) + \lambda_2 \sum_{i=1}^{m} d_i(1 - d_i)$$

$$w'_{ij} = \begin{cases} 1, & w_{ij} \geq 0.5 \\ 0, & otherwise \end{cases}$$

Learning Neural Network Architectures using Backpropagation Suraj Srinivas , R. Venkatesh Babu

Results

Method	λ_3	Layers Learnt	Architecture	AL (%)	NN (%)
Baseline	N/A	(0,x)-(0,x)-(0,0)	20-50-500-10	N/A	99.3
AL_1	$0.4\lambda_1$	(1,x)-(1,x)-(1,1)	16-26-10	99.07	99.08
AL_2	$0.4\lambda_1$	(1,x)-(1,x)-(1,0)	20-50-20-10	99.07	99.14
AL_3	$0.2\lambda_1$	(1,x)-(1,x)-(1,1)	16-40-10	99.22	99.25
AL_4	$0.2\lambda_1$	(1,x)-(1,x)-(1,0)	20-50-70-10	99.19	99.21

Learning Neural Network Architectures using Backpropagation Suraj Srinivas , R. Venkatesh Babu

QUANTIZATION

Binary Quantization

$$\hat{W}_{ij} = \begin{cases} 1 & \text{if } W_{ij} \geq 0, \\ -1 & \text{if } W_{ij} < 0. \end{cases}$$

Size Drop : 32X

Runtime : Much faster (7x) matrix multiplication for binary matrices.

Accuracy Drop : Classification error is about 20% on the top 5 accuracy on ILSVRC dataset.

COMPRESSING DEEP CONVOLUTIONAL NETWORKS USING VECTOR QUANTIZATION Yunchao Gong, Liu Liu , Ming Yang, Lubomir Bourdev

Binary Quantization while Training

- Add regularizer and round at the end of training

$$\sum_{i} W^{2}{}_{i}(1 - W^{2}{}_{i})$$

Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or −1 Matthieu Courbariaux, Itay Hubara , Daniel Soudry , Ran El-Yaniv, Yoshua Bengio

8-bit uniform quantization

- Divide the max and min weight values into 256 equal divisions uniformly.
- Round weights to the nearest point
- Store weights as 8 bit ints

Size Drop : 4X

Runtime : Much faster matrix multiplication for 8 bit matrices.

Accuracy Drop : Error is acceptable for classification for non critical tasks

https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/

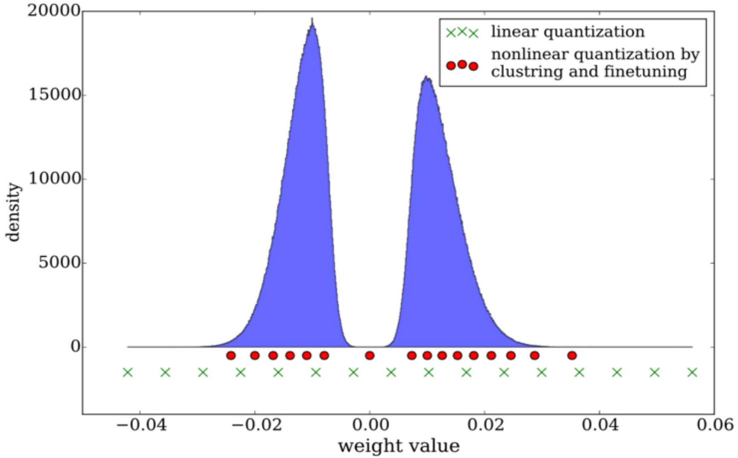
8 bit Uniform Quantization while Training

- Add L1, L2 regularizers to ensure that the min and max values are close.

$$\sum_i argmin\ d(W_i, I)$$

Non Uniform Quantization/ Weight Sharing

$$\min \sum_{i}^{mn} \sum_{j}^{k} \|w_i - c_j\|_2^2,$$



- perform k-means clustering on weights.

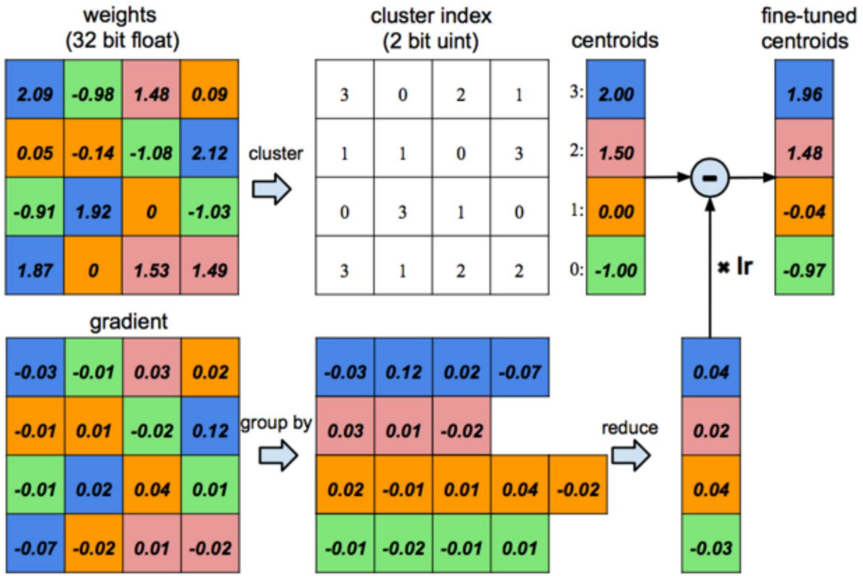
- Need to store mapping from integers to cluster centers. We only need log (k) bits to code the clusters which results in a compression factor rate of 32/ log (k). In this case the compression rate is 4.

DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING Song Han, Huizi Mao, William J. Dally

Weight Sharing while Training

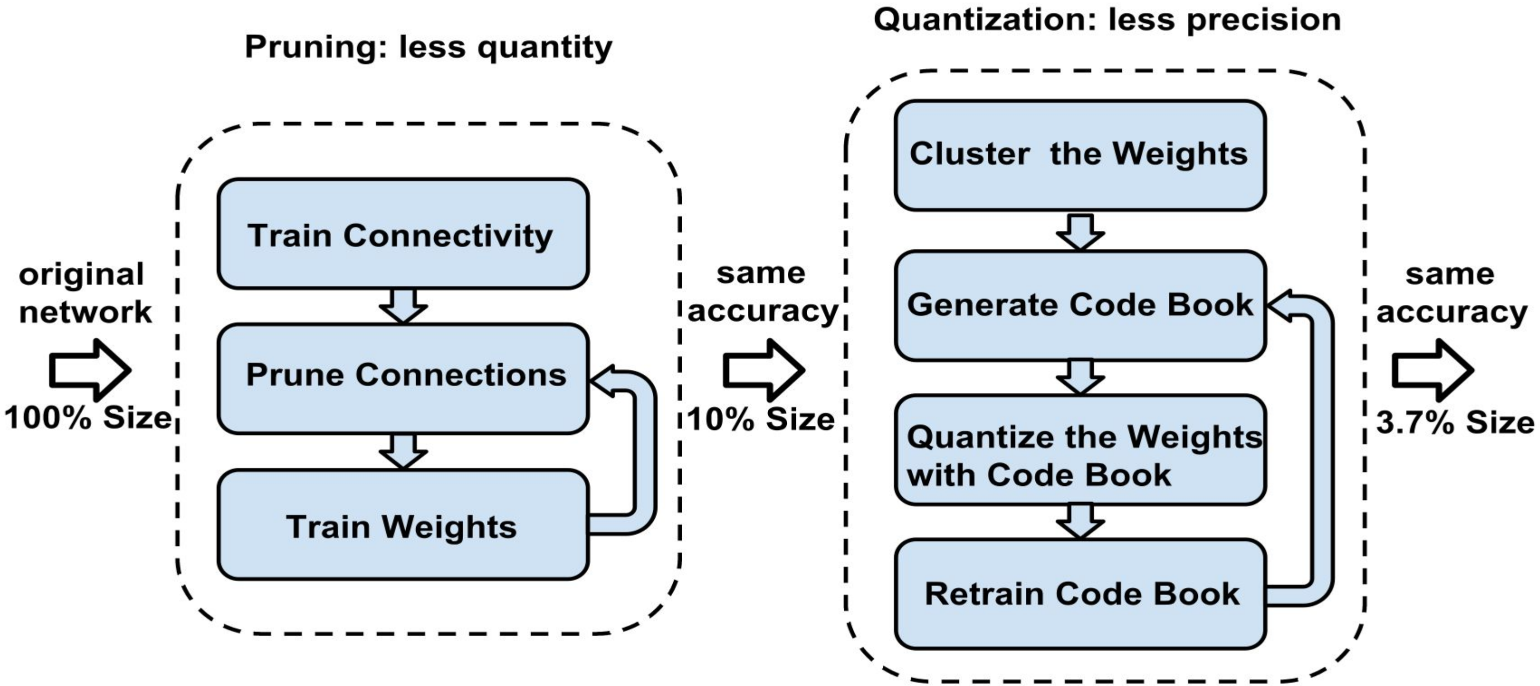
- Iterate
  - Train
  - Cluster weights
  - Make them same

- Need to ensure the gradients are updated with respect to the weight shared model.



DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING Song Han, Huizi Mao, William J. Dally

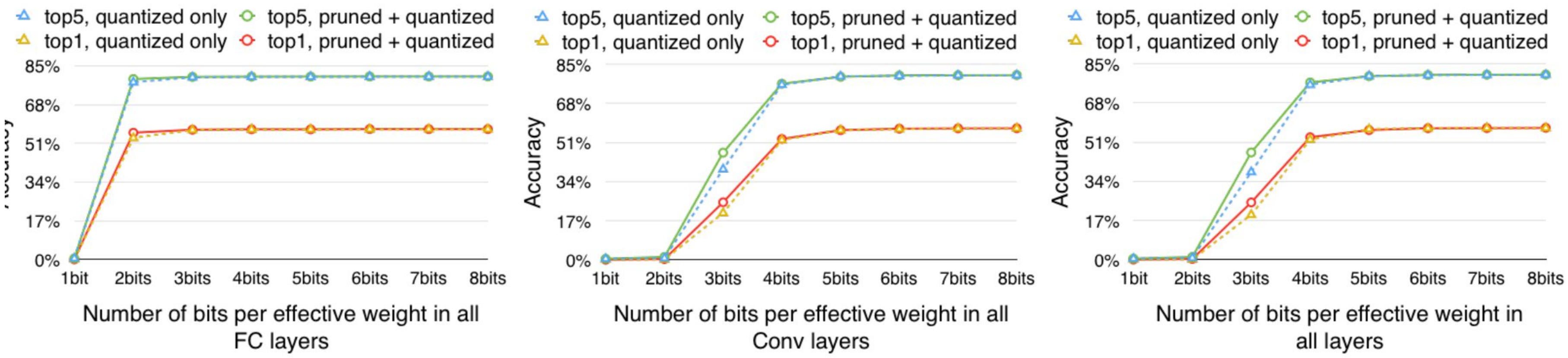
Deep Compression by Song Han



DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING Song Han, Huizi Mao, William J. Dally

Deep Compression by Song Han

Pruning and Quantization Works Well Together



DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING Song Han, Huizi Mao, William J. Dally

Product Quantization

Partition the given matrix into several submatrices and we perform k-means clustering for all of them.

$$W = [W^1, W^2, \ldots, W^s],$$

$$\min \sum_{z}^{m} \sum_{j}^{k} \|\boldsymbol{w}_z^i - \boldsymbol{c}_j^i\|_2^2,$$

$$\hat{W} = [\hat{W}^1, \hat{W}^2, \ldots, \hat{W}^s], \quad \text{where}$$

$$\hat{\boldsymbol{w}}_j^i = \boldsymbol{c}_j^i, \quad \text{where} \quad \min_j \|\boldsymbol{w}_z^i - \boldsymbol{c}_j^i\|_2^2.$$

COMPRESSING DEEP CONVOLUTIONAL NETWORKS USING VECTOR QUANTIZATION Yunchao Gong, Liu Liu , Ming Yang, Lubomir Bourdev

Residual Quantization

First quantize the vectors into k-centers.

$$\min \sum_{z}^{m} \sum_{j}^{k} \|\boldsymbol{w}_z - \boldsymbol{c}_j^1\|_2^2,$$

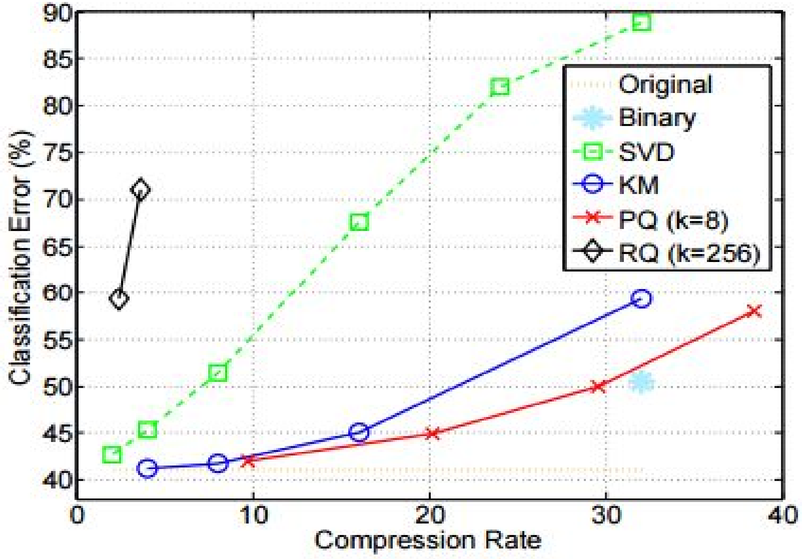
Next step is to find out the residuals for each data point(w-c) and perform k-means on the residuals

Then the resultant weight vectors are calculated as follows.

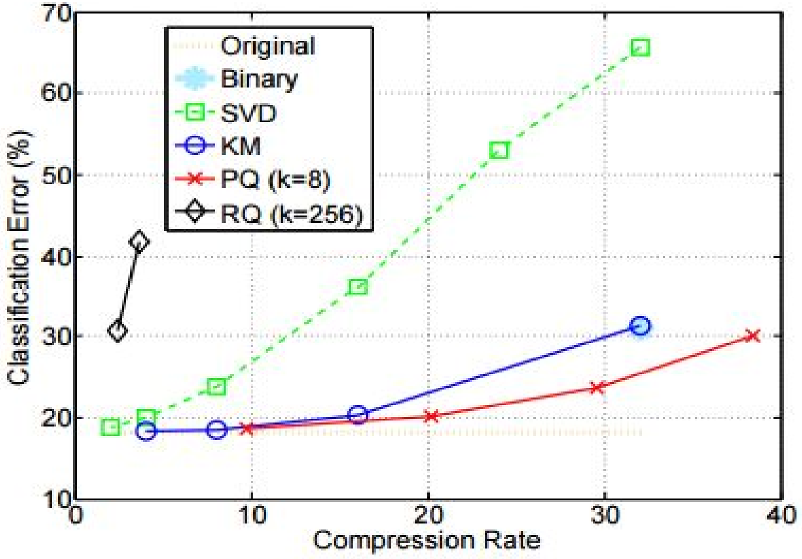
$$\hat{\boldsymbol{w}}_z = \boldsymbol{c}_j^1 + \boldsymbol{c}_j^2 + \dots, \boldsymbol{c}_j^t,$$

COMPRESSING DEEP CONVOLUTIONAL NETWORKS USING VECTOR QUANTIZATION Yunchao Gong, Liu Liu , Ming Yang, Lubomir Bourdev

Comparison of Quantization methods on Imagenet



(a) Accuracy@1



(b) Accuracy@5

Figure 3: Comparison of different compression methods on ILSVRC dataset.

COMPRESSING DEEP CONVOLUTIONAL NETWORKS USING VECTOR QUANTIZATION Yunchao Gong, Liu Liu , Ming Yang, Lubomir Bourdev

XNOR Net

❖ Binary Weight Networks :

- ➢ Estimate real time weight filter using a binary filter.
- ➢ Only the weights are binarized.
- ➢ Convolutions are only estimated with additions and subtractions (no multiplications required due to binarization).

❖ XNOR Networks:

- ➢ Binary estimation of both inputs and weights
- ➢ Input to the convolutions are binary.
- ➢ Binary inputs and weights ensure calculations using XNOR operations.

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks Mohammad Rastegari , Vicente Ordonez , Joseph Redmon , Ali Farhadi

Binary weight networks

Estimating binary weights:

Objective function :

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

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

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

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

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks Mohammad Rastegari , Vicente Ordonez , Joseph Redmon , Ali Farhadi

XNOR Networks

Objective function for dot product approximation:

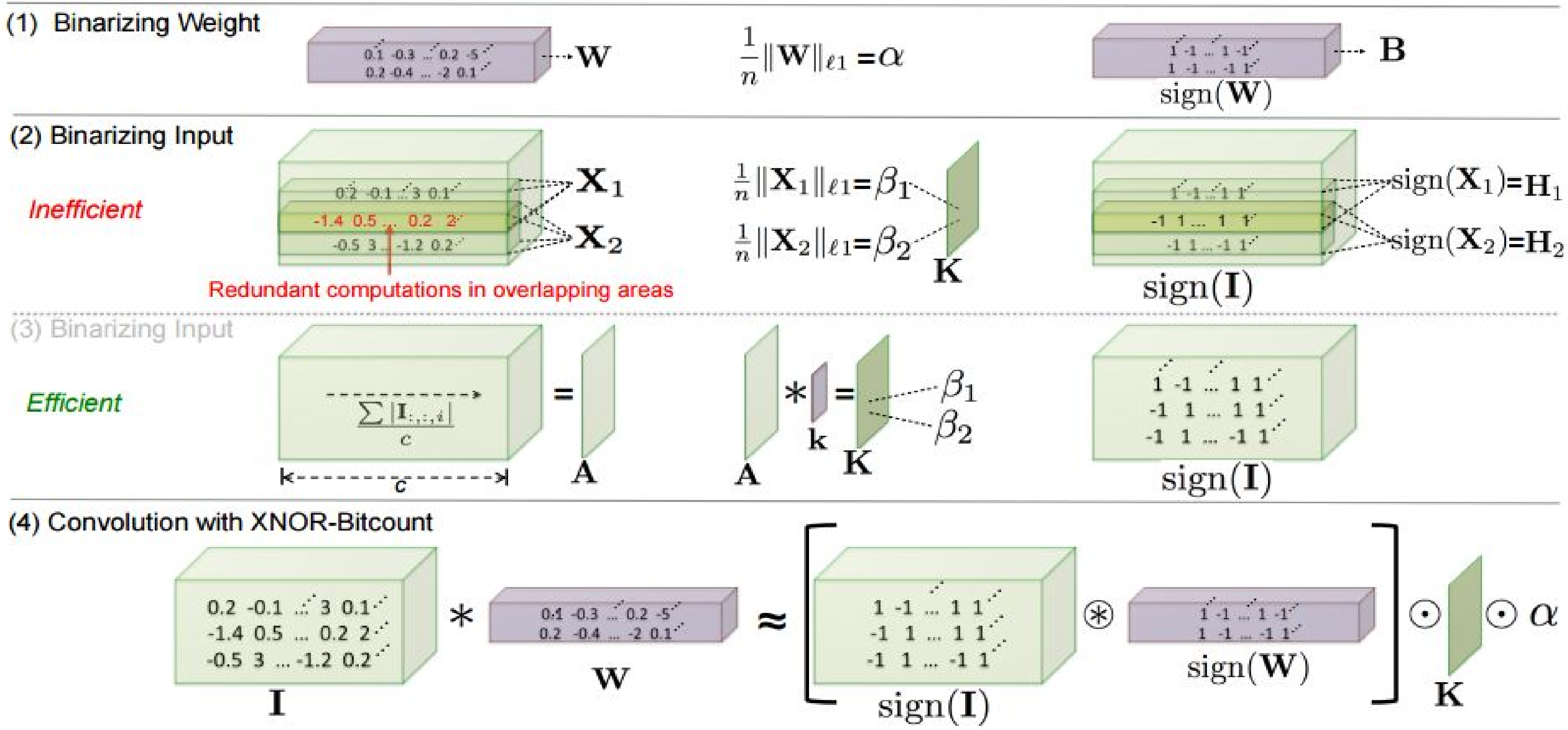
$$\alpha^*, \mathbf{B}^*, \beta^*, \mathbf{H}* = \underset{\alpha,\mathbf{B},\beta,\mathbf{H}}{\operatorname{argmin}} \|\mathbf{X} \odot \mathbf{W} - \beta\alpha\mathbf{H} \odot \mathbf{B}\|$$

We can approximate the input I and weight filter W by using the following binary operations:

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

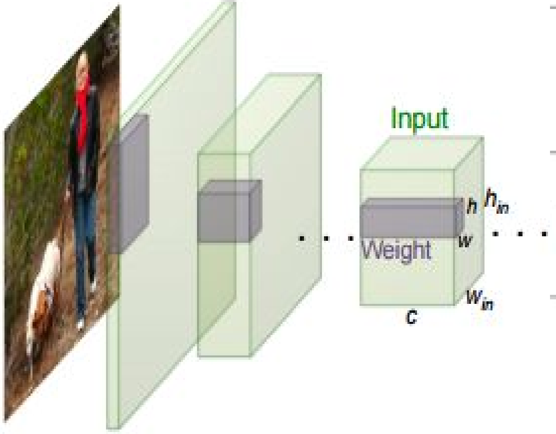
XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks Mohammad Rastegari , Vicente Ordonez , Joseph Redmon , Ali Farhadi

Approximating a convolution using binary operations



XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks Mohammad Rastegari , Vicente Ordonez , Joseph Redmon , Ali Farhadi

Results



	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

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks Mohammad Rastegari , Vicente Ordonez , Joseph Redmon , Ali Farhadi

Results

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

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks Mohammad Rastegari , Vicente Ordonez , Joseph Redmon , Ali Farhadi

FIXED POINT REPRESENTATION

FLOATING POINT VS FIXED POINT REPRESENTATION

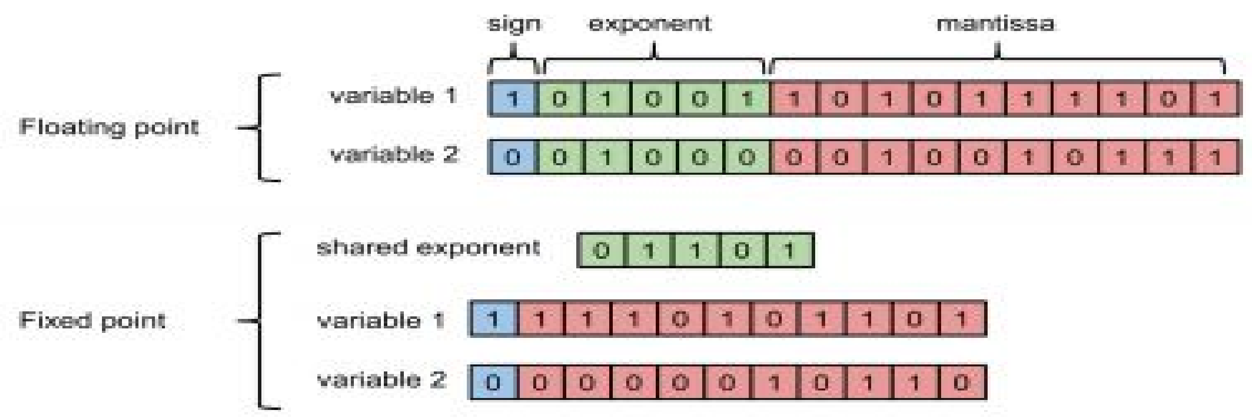


Figure 1: Comparison of the floating point and fixed point formats.

Fixed point

- ➔ Fixed point formats consist in a signed mantissa and a global scaling factor shared between all fixed point variables. It is usually ‘fixed’.
- ➔ Reducing the scaling factor reduces the range and augments the precision of the format.
- ➔ It relies on integer operations. It is hardware-wise cheaper than its floating point counterpart, as the exponent is shared and fixed.

Disadvantages of using fixed point

- ❖ When training deep neural networks :
  - ➢ Activations , gradients and parameters have very different ranges.
  - ➢ The ranges of the gradients slowly diminish during training.
  - ➢ Fixed point arithmetic is not optimised on regular hardware and specialised hardware such as FPGAs are required.

- ❖ As a result the fixed point format with its unique shared fixed exponent is ill-suited to deep learning.
- ❖ The dynamic fixed point format is a variant of the fixed point format in which there are several scaling factors instead of a single global one.

Summary

- Pruning weights and neurons
- Uniform Quantization
- Non Uniform Quantization / Weight Sharing

THANK YOU