Cours ENSL:
Big Data – Streaming, Sketching, Compression, Learning
Today: Convolutional Neural Networks, computational and memory issues

---

Olivier Beaumont, Inria Bordeaux Sud-Ouest
Olivier.Beaumont@inria.fr

Outline of the talk

- Crash course in CNNs
  - very little about usage

Outline of the talk

- Crash course in CNNs
  - very little about usage no cat pictures

Outline of the talk

- Crash course in CNNs
  - very little about usage no cat pictures
  - very little about origins of CNNs

Outline of the talk

- Crash course in CNNs
  - very little about usage no cat pictures
  - very little about origins of CNNs no (cat) neuron picture

Outline of the talk

- Crash course in CNNs
  - very little about usage no cat pictures
  - very little about origins of CNNs no (cat) neuron picture
  - what type of task graphs, what type of operations, what type of constraints
- Computational Issues
  - During the inference phase:
    - Pruning, Quantization (weights and activations), Low Rank Decompositions (SVD based and TT decomposition)
  - During the training phase:
    - Parallelism (hyper-parameters, data parallelism, spatial parallelism, model parallelism), Mixed Precision arithmetic, Memory issues (checkpointing and offloading)

Crash Course in DL

Outline

Crash Course in DL (1): a little bit of vocabulary

- CNNs have been designed to solve very difficult problems
  - "Is there a cat in this picture?"

Crash Course in DL (1): a little bit of vocabulary

- CNNs have been designed to solve very difficult problems
  - "Is there a cat in this picture?" no obvious algorithm...
- Idea: data-driven approach (supervised learning)
  - we start with a network architecture with weights to be determined (a gigantic family of computable functions)
  - we start with a gigantic set of annotated data (input, result).
- Many times:
  - we randomly choose a pair (input, result)
  - we compute an error between f(input) and result
  - we modify the parameters to minimize this error
- We look for the network weights that minimize this error on all pairs
- We hope that this will work also for the other pairs

Crash Course in DL (1): a little bit of vocabulary

- CNNs have been designed to solve very difficult problems
  - "Is there a cat in this picture?" no obvious algorithm...
- Idea: data-driven approach (supervised learning)
  - we start with a network architecture with weights to be determined (a gigantic family of computable functions) Hyper Parameters (HP)
  - we start with a gigantic set of annotated data (input, result).
- Many times:
  - we randomly choose a pair (input, result)
  - we compute an error between f(input) and result
  - we modify the parameters to minimize this error
- We look for the network weights that minimize this error on all pairs
- We hope that this will work also for the other pairs

Crash Course in DL (1): a little bit of vocabulary

- CNNs have been designed to solve very difficult problems
  - "Is there a cat in this picture?" no obvious algorithm...
- Idea: data-driven approach (supervised learning)
  - we start with a network architecture with weights to be determined (a gigantic family of computable functions) Hyper Parameters (HP)
  - we start with a gigantic set of annotated data (input, result). https://www.mturk.com
- Many times:
  - we randomly choose a pair (input, result)
  - we compute an error between f(input) and result
  - we modify the parameters to minimize this error
- We look for the network weights that minimize this error on all pairs
- We hope that this will work also for the other pairs

Crash Course in DL (1): a little bit of vocabulary

- CNNs have been designed to solve very difficult problems
  - "Is there a cat in this picture?" no obvious algorithm...
- Idea: data-driven approach (supervised learning)
  - we start with a network architecture with weights to be determined (a gigantic family of computable functions) Hyper Parameters (HP)
  - we start with a gigantic set of annotated data (input, result). https://www.mturk.com
- Many times:
  - we randomly choose a pair (input, result) or a mini-batch of pairs (HP)
  - we compute an error between f(input) and result
  - we modify the parameters to minimize this error
- We look for the network weights that minimize this error on all pairs
- We hope that this will work also for the other pairs

Crash Course in DL (1): a little bit of vocabulary

- CNNs have been designed to solve very difficult problems
  - "Is there a cat in this picture?" no obvious algorithm...
- Idea: data-driven approach (supervised learning)
  - we start with a network architecture with weights to be determined (a gigantic family of computable functions) Hyper Parameters (HP)
  - we start with a gigantic set of annotated data (input, result). https://www.mturk.com
- Many times:
  - we randomly choose a pair (input, result) or a mini-batch of pairs (HP)
  - we compute an error between f(input) and result choice of loss L (HP) is crucial, returns a single number, regularization, overfitting
  - we modify the parameters to minimize this error
- We look for the network weights that minimize this error on all pairs
- We hope that this will work also for the other pairs

Crash Course in DL (1): a little bit of vocabulary

- CNNs have been designed to solve very difficult problems
  - "Is there a cat in this picture?" no obvious algorithm...
- Idea: data-driven approach (supervised learning)
  - we start with a network architecture with weights to be determined (a gigantic family of computable functions) Hyper Parameters (HP)
  - we start with a gigantic set of annotated data (input, result). https://www.mturk.com
- Many times:
  - we randomly choose a pair (input, result) or a mini-batch of pairs (HP)
  - we compute an error between f(input) and result choice of loss L (HP) is crucial, returns a single number, regularization, overfitting
  - we modify the parameters to minimize this error Stochastic Gradient Descent, estimate $\frac{\partial L}{\partial W_i}$, update W_i with learning rate (HP)
- We look for the network weights that minimize this error on all pairs
- We hope that this will work also for the other pairs

Crash Course in DL (1): a little bit of vocabulary

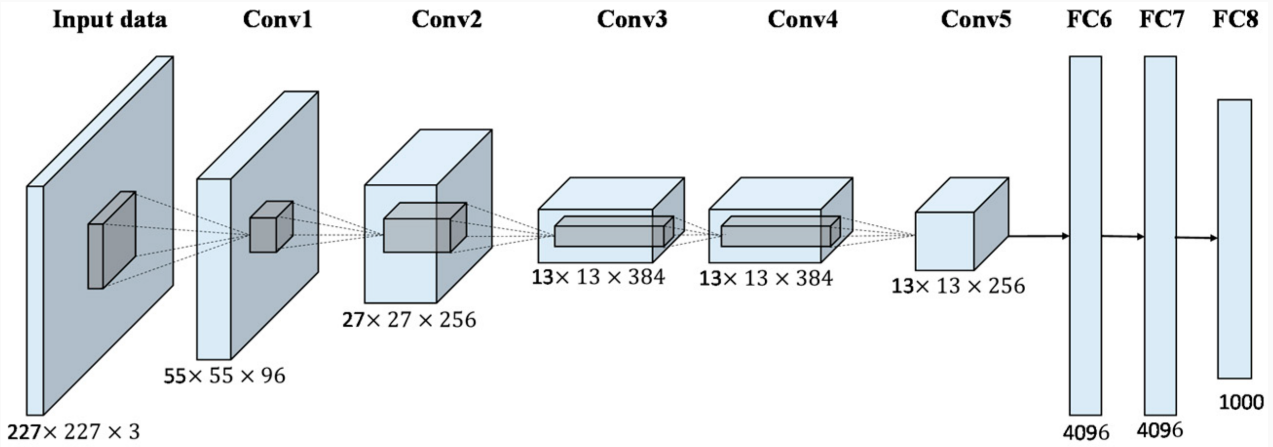
- CNNs have been designed to solve very difficult problems
  - "Is there a cat in this picture?" no obvious algorithm...
- Idea: data-driven approach (supervised learning)
  - we start with a network architecture with weights to be determined (a gigantic family of computable functions) Hyper Parameters (HP)
  - we start with a gigantic set of annotated data (input, result). https://www.mturk.com
- Many times:
  - we randomly choose a pair (input, result) or a mini-batch of pairs (HP)
  - we compute an error between f(input) and result choice of loss L (HP) is crucial, returns a single number, regularization, overfitting
  - we modify the parameters to minimize this error Stochastic Gradient Descent, estimate $\frac{\partial L}{\partial W_i}$, update W_i with learning rate (HP)
- We look for the network weights that minimize this error on all pairs Optimization Strategy, validation set and error
- We hope that this will work also for the other pairs

Crash Course in DL (1): a little bit of vocabulary

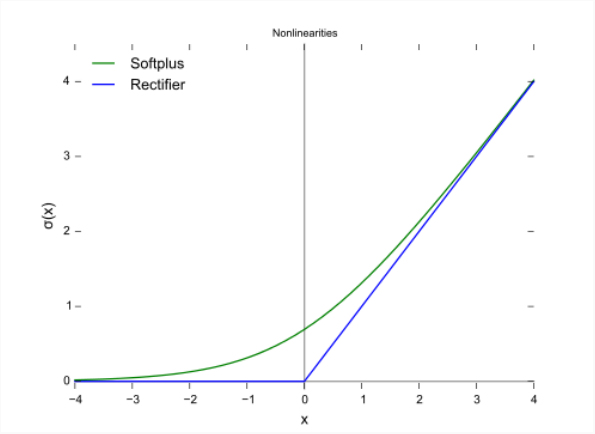
- CNNs have been designed to solve very difficult problems
  - "Is there a cat in this picture?" no obvious algorithm...
- Idea: data-driven approach (supervised learning)
  - we start with a network architecture with weights to be determined (a gigantic family of computable functions) Hyper Parameters (HP)
  - we start with a gigantic set of annotated data (input, result). https://www.mturk.com
- Many times:
  - we randomly choose a pair (input, result) or a mini-batch of pairs (HP)
  - we compute an error between f(input) and result choice of loss L (HP) is crucial, returns a single number, regularization, overfitting
  - we modify the parameters to minimize this error Stochastic Gradient Descent, estimate $\frac{\partial L}{\partial W_i}$, update W_i with learning rate (HP)
- We look for the network weights that minimize this error on all pairs Optimization Strategy, validation set and error
- We hope that this will work also for the other pairs test set and error, training and inference phases

Crash Course in DL (2): What do networks look like?

AlexNet (2012): 60 million parameters



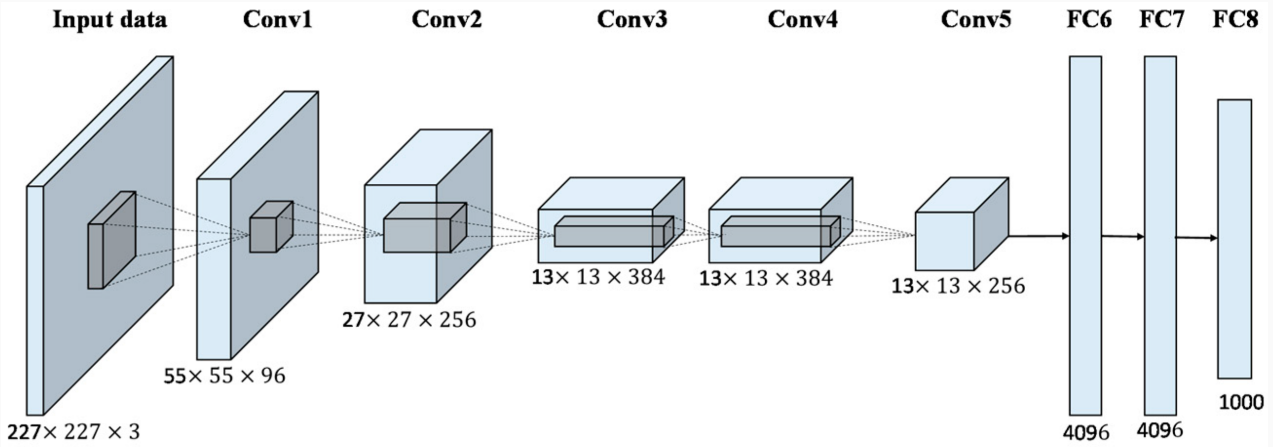
- Convolutional (next slides) and Fully Connected Layers (matrix)



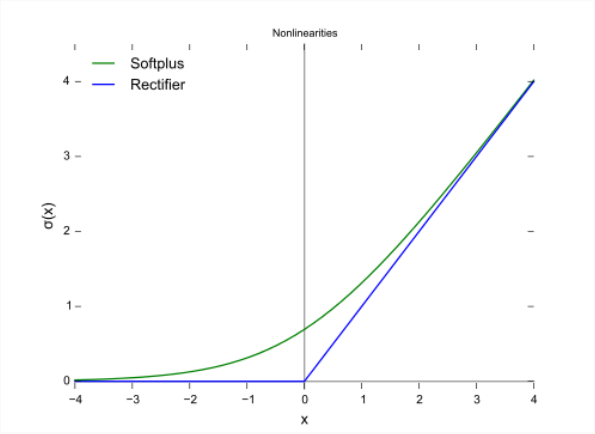
- Conv should be read Conv + RELU Why?

Crash Course in DL (2): What do networks look like?

AlexNet (2012): 60 million parameters



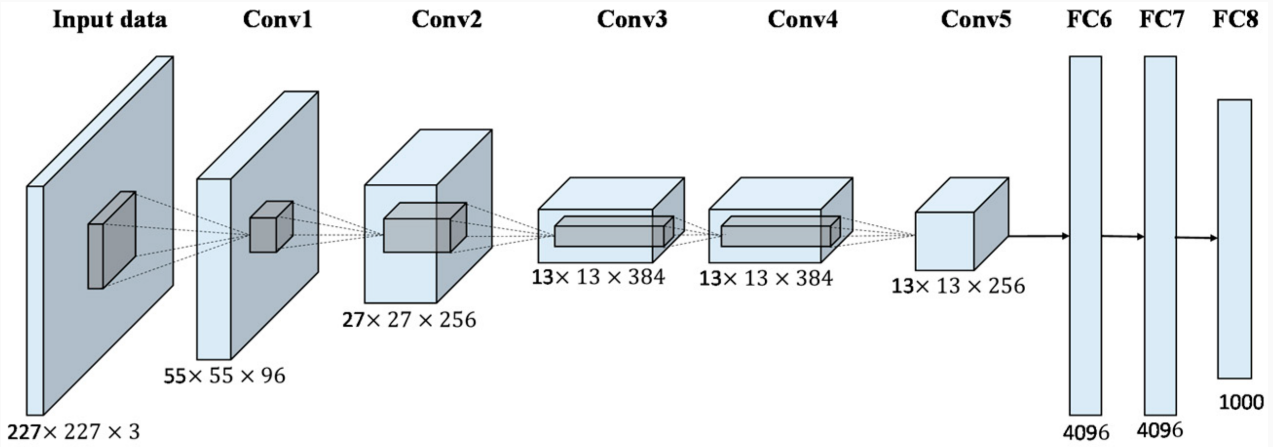
- Convolutional (next slides) and Fully Connected Layers (matrix)



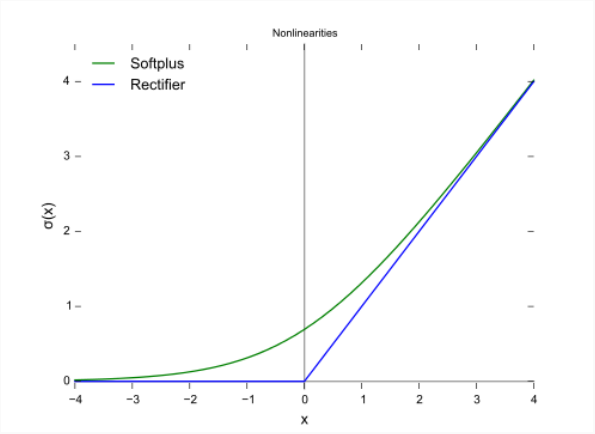
- Conv should be read Conv + RELU Why?
- Convolutions reshape 3D volume (spatial + RGB locality)
- smaller "image sizes" (width + height), larger number of channels

Crash Course in DL (2): What do networks look like?

AlexNet (2012): 60 million parameters



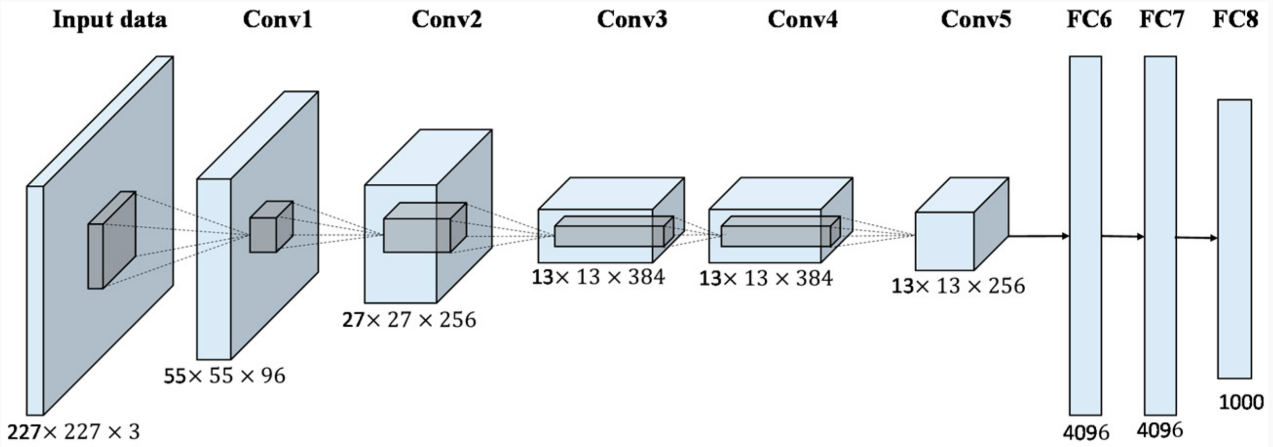
- Convolutional (next slides) and Fully Connected Layers (matrix)



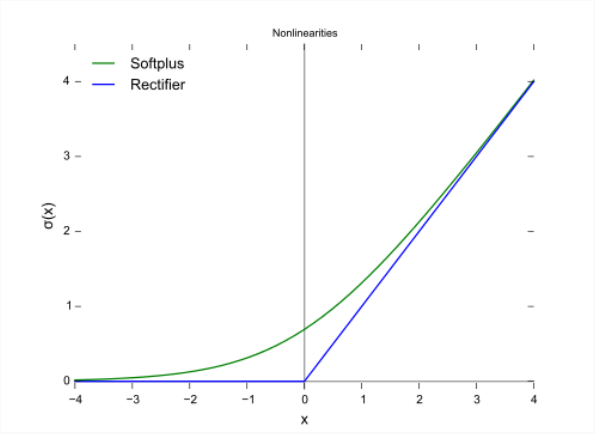
- Conv should be read Conv + RELU Why?

Crash Course in DL (2): What do networks look like?

AlexNet (2012): 60 million parameters



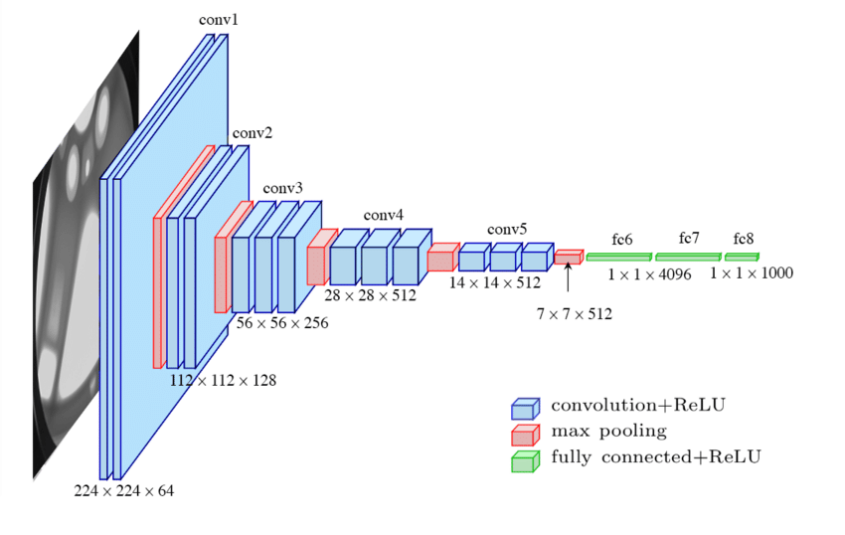
- Convolutional (next slides) and Fully Connected Layers (matrix)



- Conv should be read Conv + RELU Why?
- Convolutions reshape 3D volume (spatial + RGB locality)
- smaller "image sizes" (width + height), larger number of channels

Crash Course in DL (2): What do networks look like?

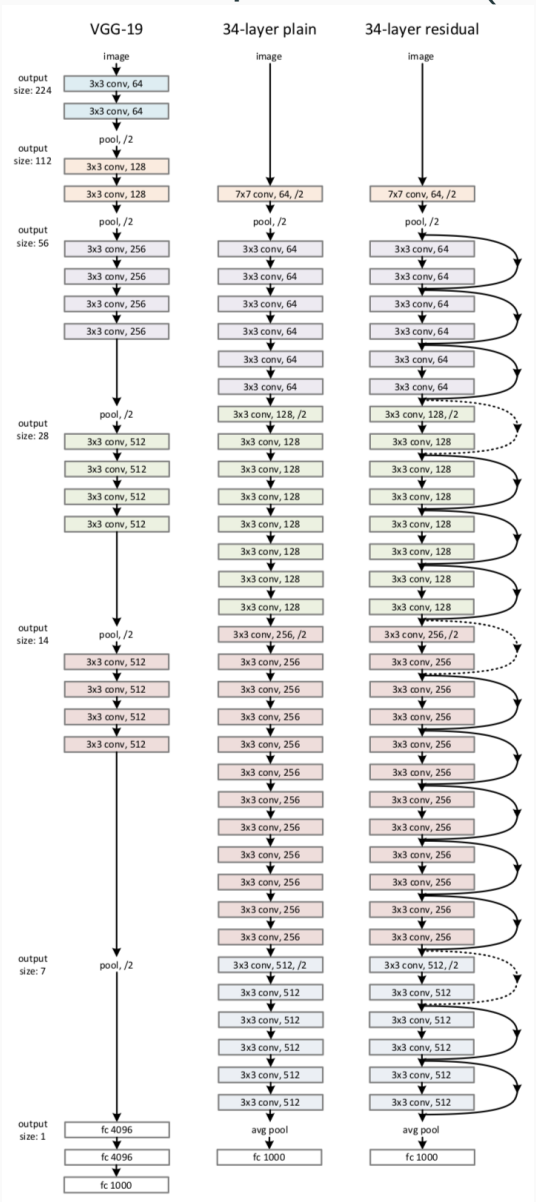
VGG16: 138 million parameters

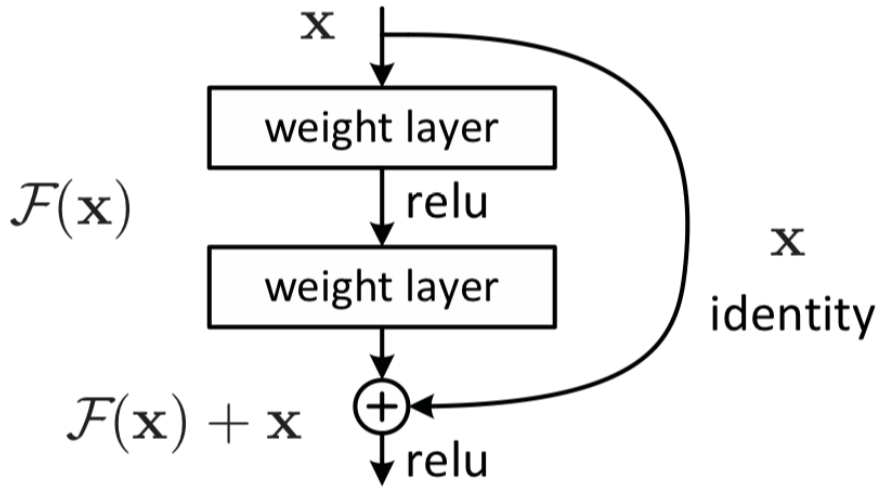


- max pooling layers to decrease image size
- smaller convolutional filters (3x3 and 1x1 instead of 3, 7 and even 11)
- deeper network (13 vs 5)

Crash Course in DL (2): What do networks look like?

ResNet: 60M parameters (same as AlexNet) for 152 layers.



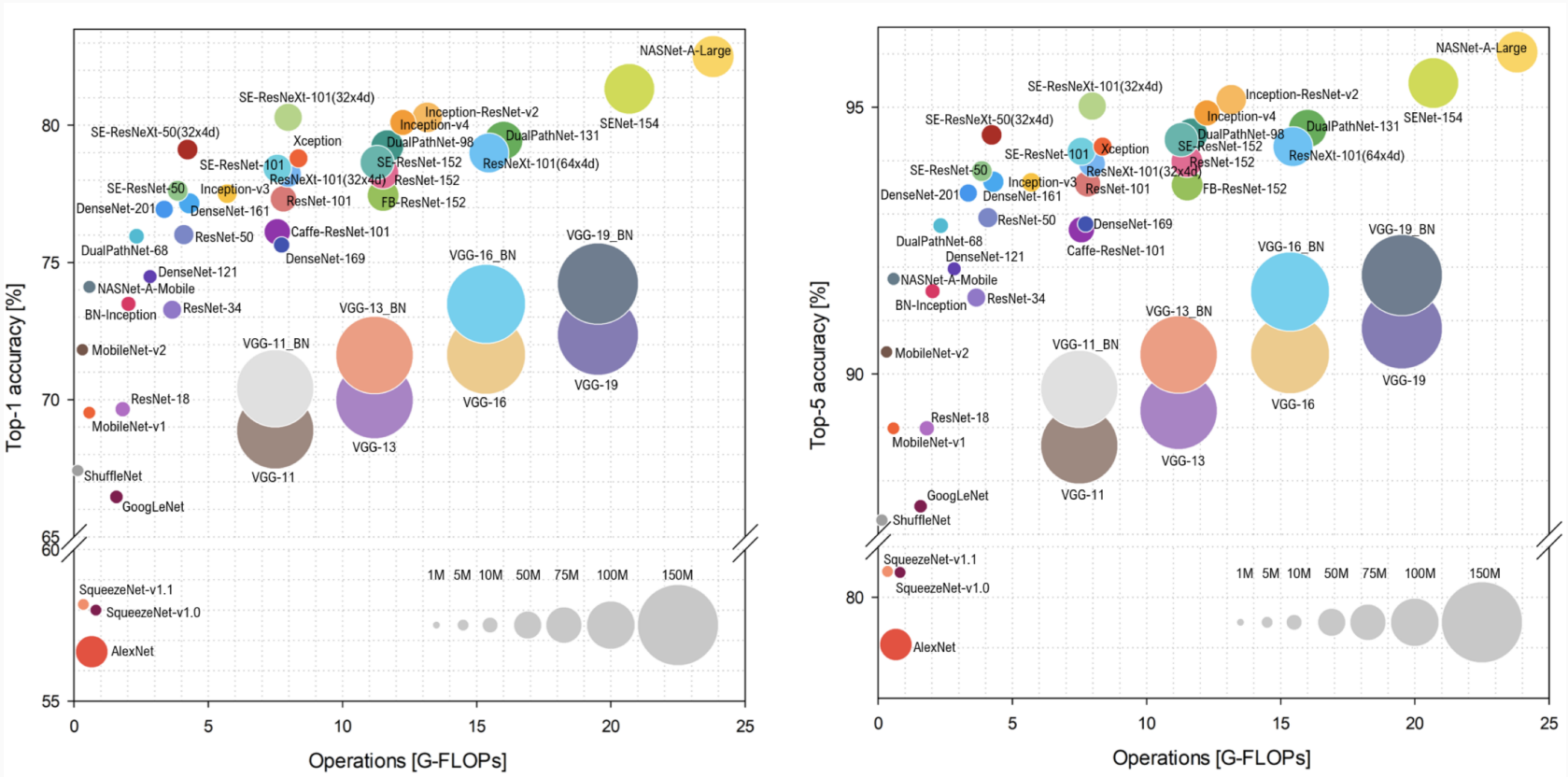


- use of skip connections to mitigate the problem of vanishing gradients
- state-of-the-art

CNNs Zoo

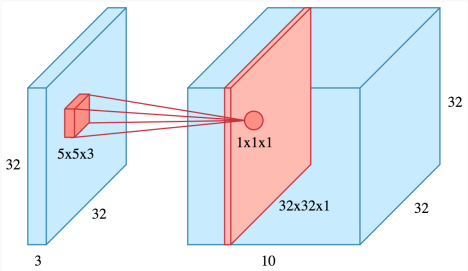
Performance on ImageNet

- Top1: find the the best label
- Top5: find the best label while proposing 5
- size: number of weights
- higher: better accuracy
- right: higher computation cost

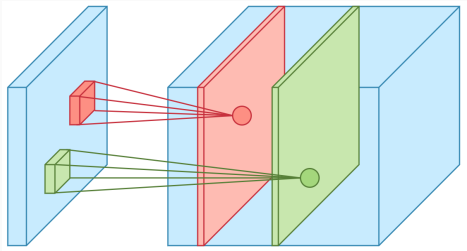


Outline

Convolution (description)

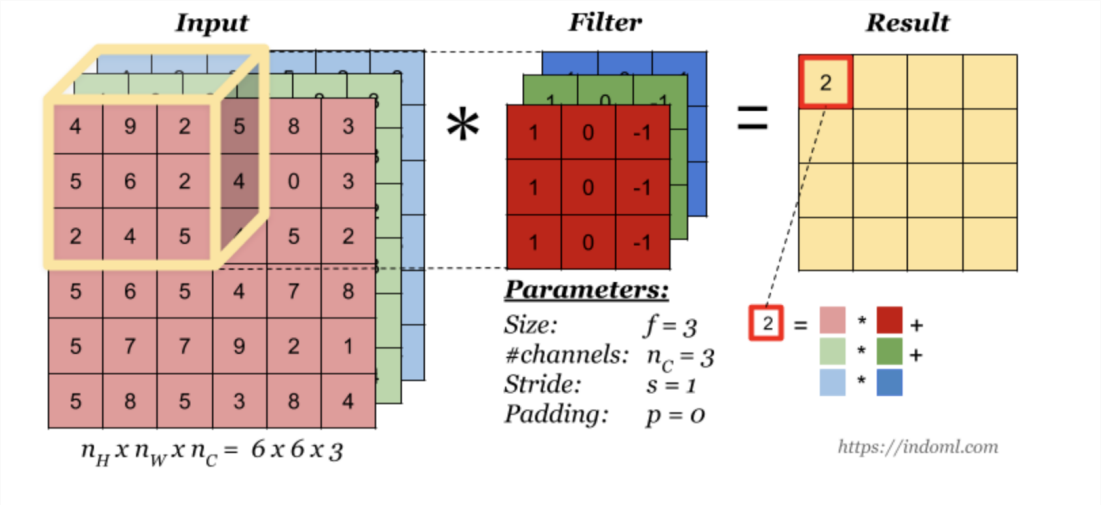


(a) Filter $5 \times 5 \times 3$



(b) 2 filters $5 \times 5 \times 3$ that contribute to different output feature maps

- input Image: $w \times h \times c$
- Filter $k \times k \times c$
- D Filters Output image $w \times h \times d$ (not completely true, padding, stride, dilation)
- operation:



Convolution (VGG example)

Case Study: VGGNet

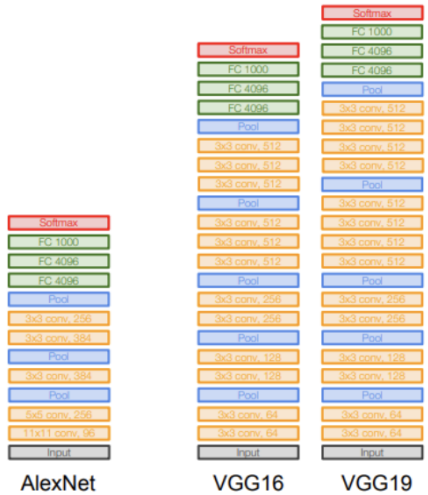
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14



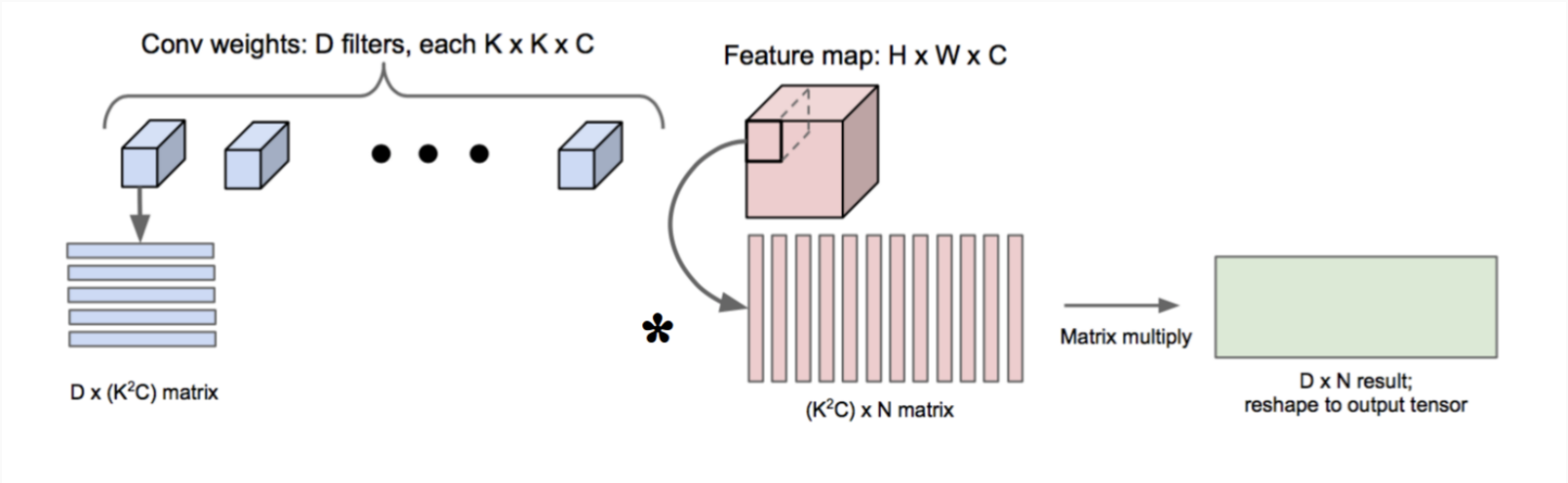
(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

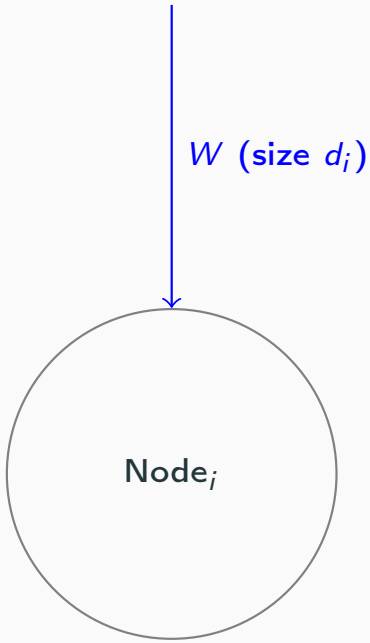
Cost and Implementation

- d times $k \times k \times c$ filters
- arranged into a $d \times k^2 c$ matrix
- input images are arranged into a $k^2 c \times n$ matrix ($n = wh$ with no striding or dilation)
- convolution is performed with a GEMM (GPU friendly)
- Note that there is a lot of redundancy in the input image matrix (see later)

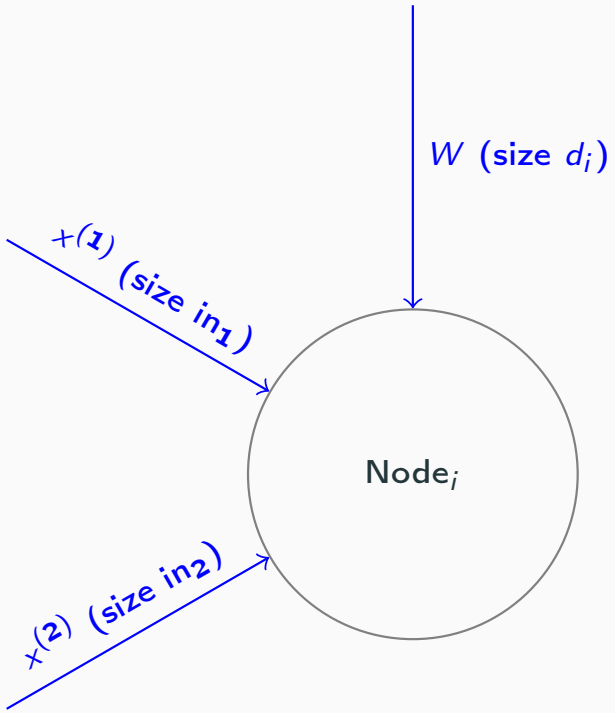


Outline

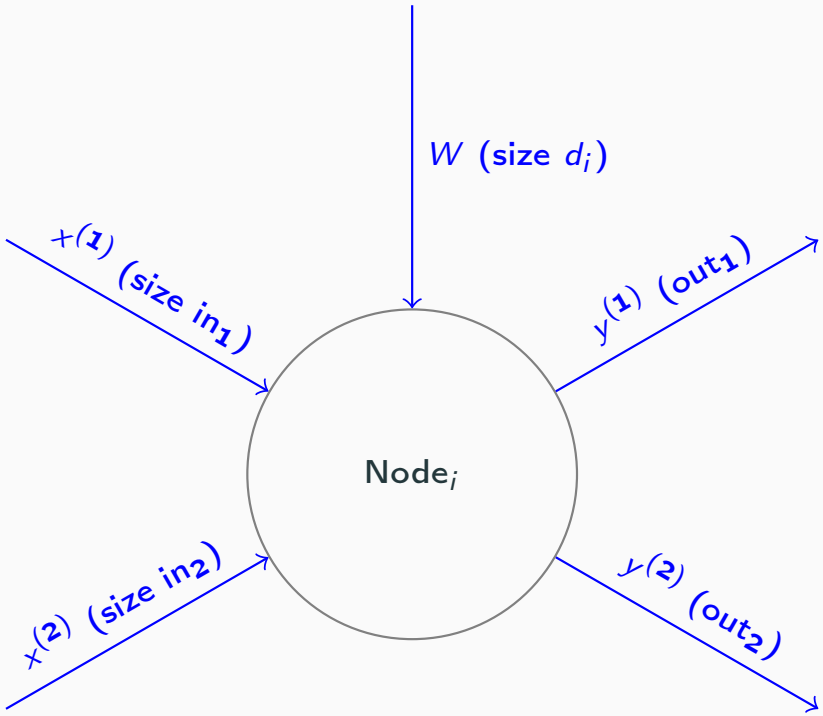
SGD: forward propagation and backward propagation



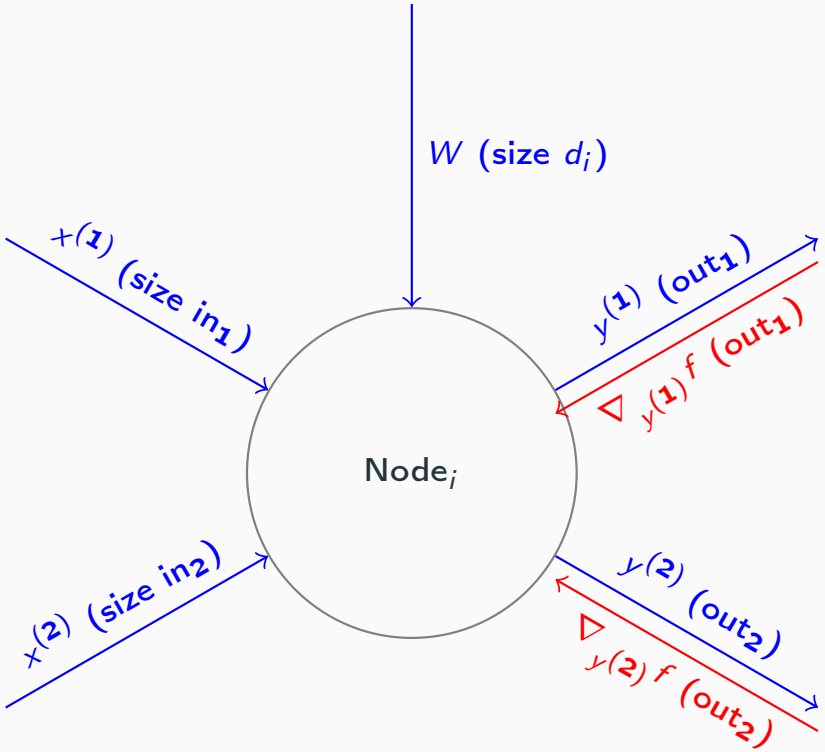
SGD: forward propagation and backward propagation



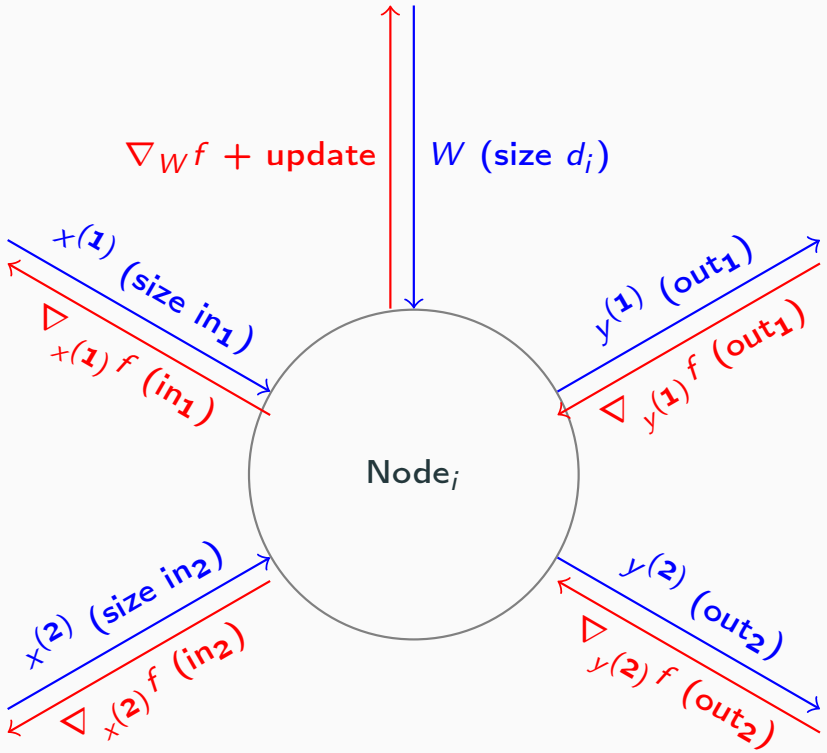
SGD: forward propagation and backward propagation



SGD: forward propagation and backward propagation



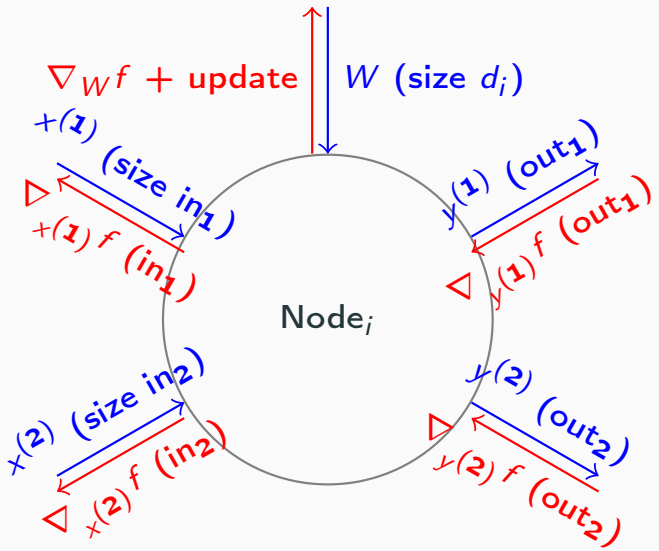
SGD: forward propagation and backward propagation



- $\frac{\partial f}{\partial x_i^{(1)}} = \frac{\partial f}{\partial y^{(1)}} \frac{\partial y^{(1)}}{\partial x_i^{(1)}} + \frac{\partial f}{\partial y^{(2)}} \frac{\partial y^{(2)}}{\partial x_i^{(1)}}$
- $\frac{\partial f}{\partial x_i^{(2)}} = \frac{\partial f}{\partial y^{(1)}} \frac{\partial y^{(1)}}{\partial x_i^{(2)}} + \frac{\partial f}{\partial y^{(2)}} \frac{\partial y^{(2)}}{\partial x_i^{(2)}}$
- $\frac{\partial f}{\partial W_i} = \frac{\partial f}{\partial y^{(1)}} \frac{\partial y^{(1)}}{\partial W_i} + \frac{\partial f}{\partial y^{(2)}} \frac{\partial y^{(2)}}{\partial W_i}$

- size($\nabla_{x^{(1)}}$)=size($x^{(1)}$)
- the costs of forward and backward phases are approximately the same for CONV and FC layers

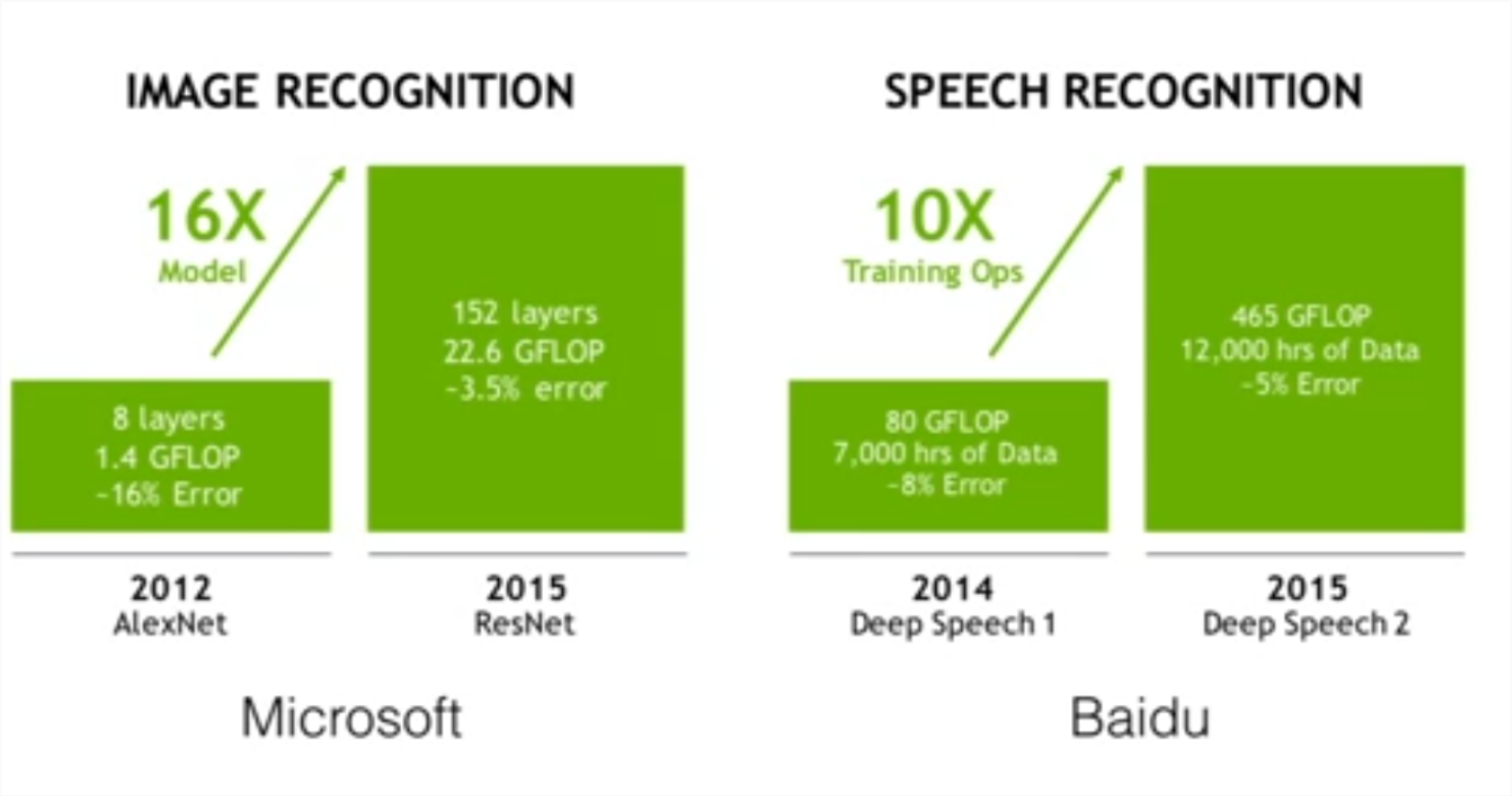
SGD: forward propagation and backward propagation



Storage

- we must keep W all the time (and update it)
- we must remember
  - $x^{(1)}$, $x^{(2)}$ until the corresponding backward operation
  - (or to recompute them when using Checkpointing)
- Forward: compute $y^{(1)}$ et $y^{(2)}$ and then wait...
- Receive $\nabla_{y^{(1)}} f$ et $\nabla_{y^{(2)}} f$ (same size as $y^{(1)}$ and $y^{(2)}$)
- compute and trasmit $\nabla_{x^{(1)}} f$ and $\nabla_{x^{(2)}} f$ (same size as $x^{(1)}$ and $x^{(2)}$)
- compute $\nabla_W f$ et update W
- computational DAG: original DAG + loss + returned DAG

Models are getting larger and deeper



Models are getting larger and deeper

- Size issues
  - size problem for inference: must fit into the phone's memory
    - difficult to convince users to download and store a model of 100MB or more
  - size problem for training: it must fit into the GPU's memory
    - all network weights (hyper-parameter search, data parallelism)
    - all activations during training (and for a long time)

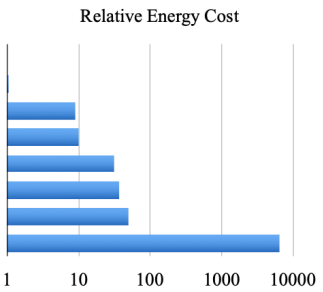
Models are getting larger and deeper

- Size issues
  - size problem for inference: must fit into the phone's memory
    - difficult to convince users to download and store a model of 100MB or more
  - size problem for training: it must fit into the GPU's memory
    - all network weights (hyper-parameter search, data parallelism)
    - all activations during training (and for a long time)
- Time issues
  - Training time for Resnet PyTorch with 4 M40 GPUs

	Error rate	Training time
ResNet18:	10.76%	2.5 days
ResNet50:	7.02%	5 days
ResNet101:	6.21%	1 week
ResNet152:	6.16%	1.5 weeks

  - a productivity problem
- Energy Issues
  - per game, AlphaGo (1920 CPUs and 280 GPUs) 3000$ electric bill
  - critical situation when used on mobile phone, risk of draining the battery
  - the main source of energy consumption is memory accesses

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**



Computational Issues: Inference Phase

Outline

Pruning: removing weights (1)

Source: Learning both Weights and Connections for Efficient Neural Networks. Han et al. NIPS'15

Goal: how to dramatically prune the model weights (ie to sparsify matrices) ?

Pruning: removing weights (1)

Source: Learning both Weights and Connections for Efficient Neural Networks. Han et al. NIPS'15

Goal: how to dramatically prune the model weights (ie to sparsify matrices) ?

- Loop
  - Train the network (full network)
  - Prune edges (weights forced to be 0)
  - Train remaining weights

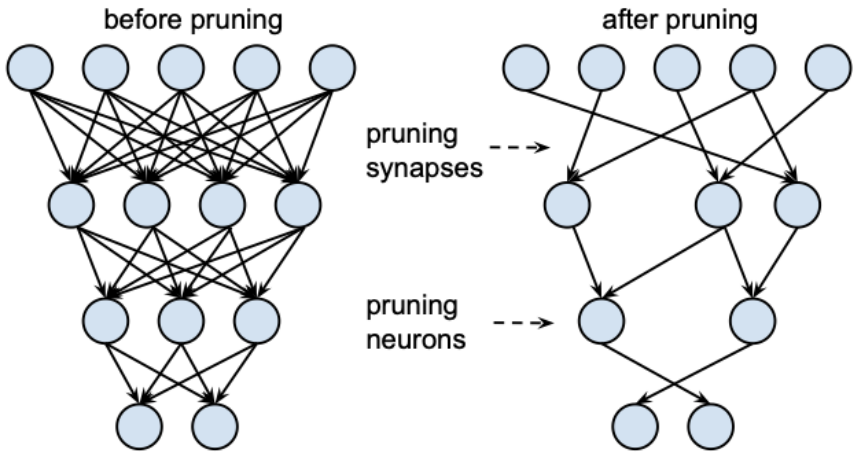


Figure 3: Synapses and neurons before and after pruning.

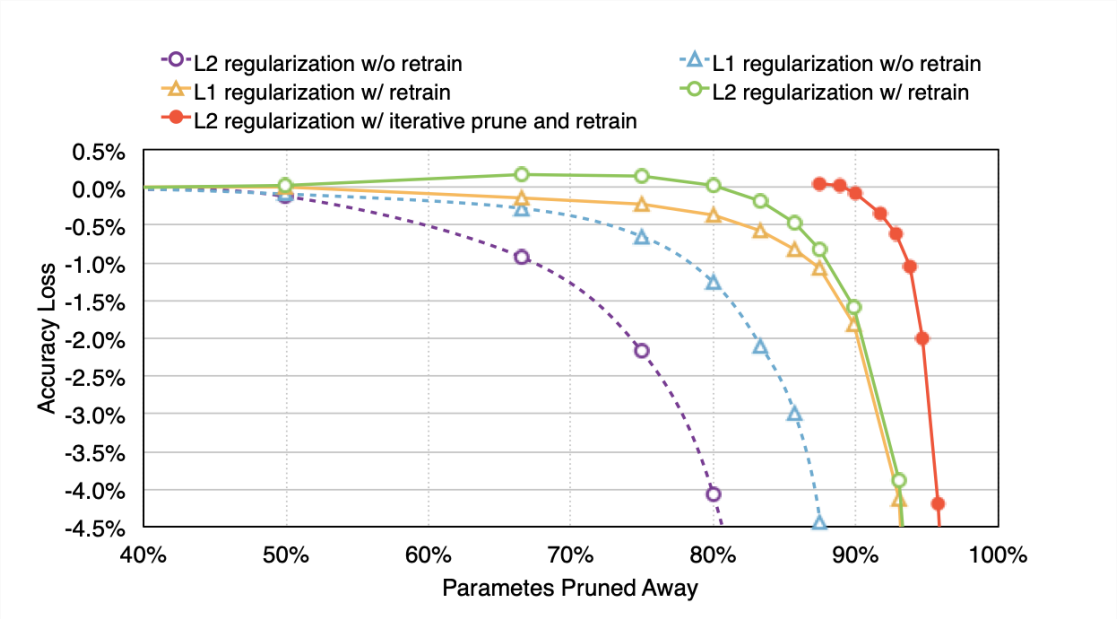
- It works!
- Could this (fix the sparsity pattern) work to build preconditioners?

Pruning: removing weights (2)

- Loop Retraining is crucial:
  - train + prune for AlexNet: -4% accuracy with 80% pruning
  - train + prune + retrain for AlexNet: 80% pruning improves accuracy! Why?

Pruning: removing weights (2)

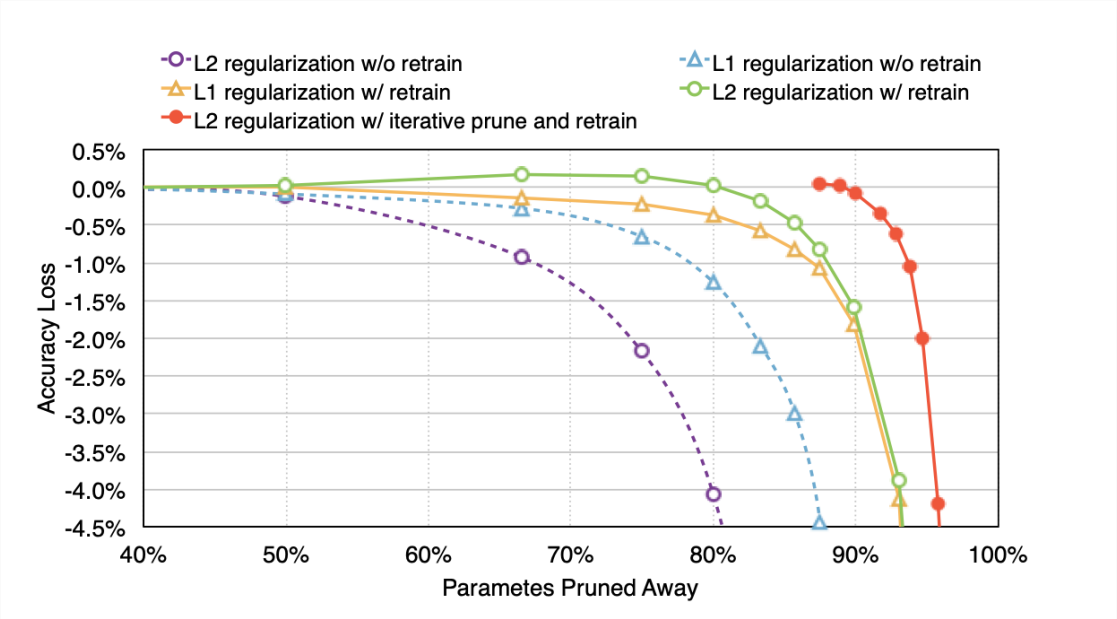
- Loop Retraining is crucial:
  - train + prune for AlexNet: -4% accuracy with 80% pruning
  - train + prune + retrain for AlexNet: 80% pruning improves accuracy! Why?
  - (train + prune)* same accuracy up to 90% pruning
  - AlexNet (ImageNet) from 60M parameters to 6M with the same accuracy



??

Pruning: removing weights (2)

- Loop Retraining is crucial:
  - train + prune for AlexNet: -4% accuracy with 80% pruning
  - train + prune + retrain for AlexNet: 80% pruning improves accuracy! Why?
  - (train + prune)* same accuracy up to 90% pruning
  - AlexNet (ImageNet) from 60M parameters to 6M with the same accuracy



??

- Something even more surprising:
  - source DSD: Dense-Sparse-Dense Training for Deep Neural Networks, Song Han, ICLR 2017
  - Retraining with all removed weights improves the accuracy (by 1 to 2%) ??

??

Outline

Quantization of the weights (1)

source: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, Song Han ICLR'16

Goal: discretize model weights... make 2.09, 2.12, 1.92 the same value

- How to perform quantization?

Quantization of the weights (1)

source: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, Song Han ICLR'16

Goal: discretize model weights... make 2.09, 2.12, 1.92 the same value

- How to perform quantization?
  - use of clustering algorithms on the weights
  - associate each weight to its closest representative

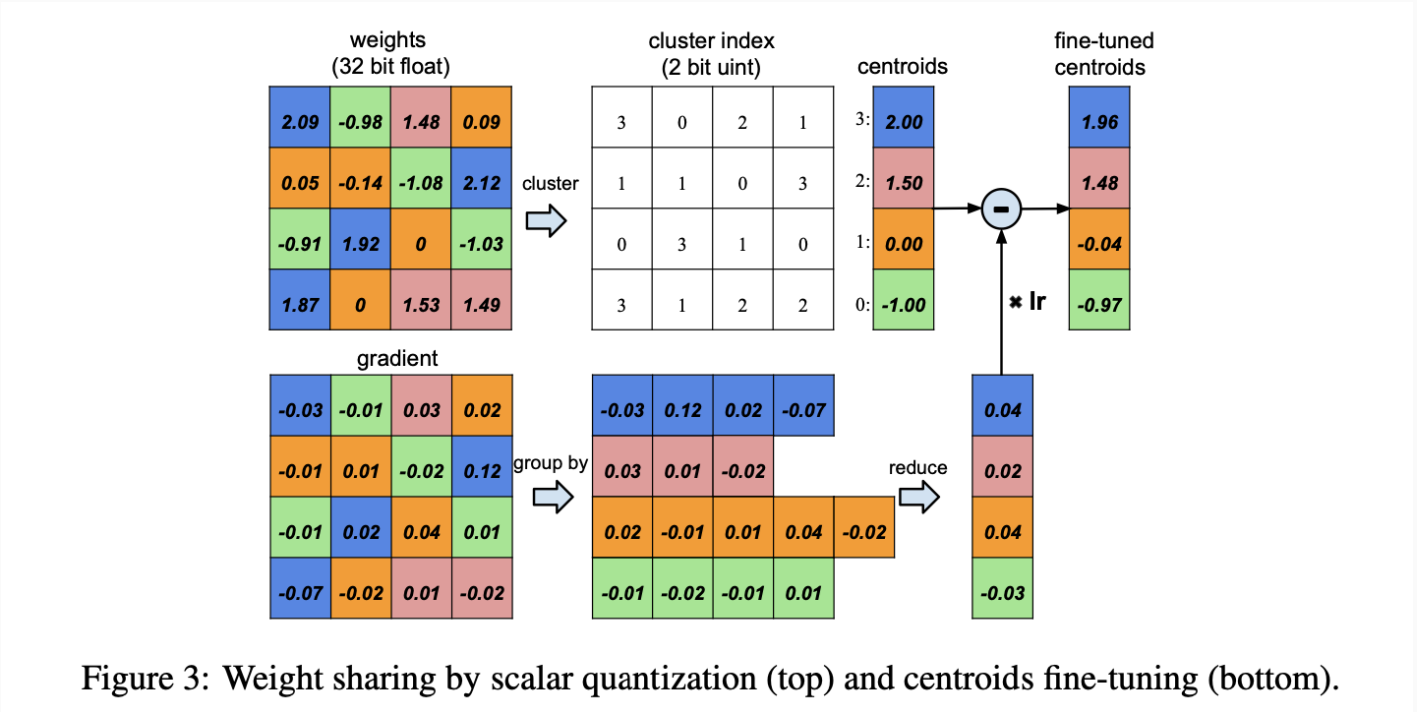


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

- Does it work?

Quantization of the weights (1)

source: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, Song Han ICLR'16

Goal: discretize model weights... make 2.09, 2.12, 1.92 the same value

- How to perform quantization?
  - use of clustering algorithms on the weights
  - associate each weight to its closest representative

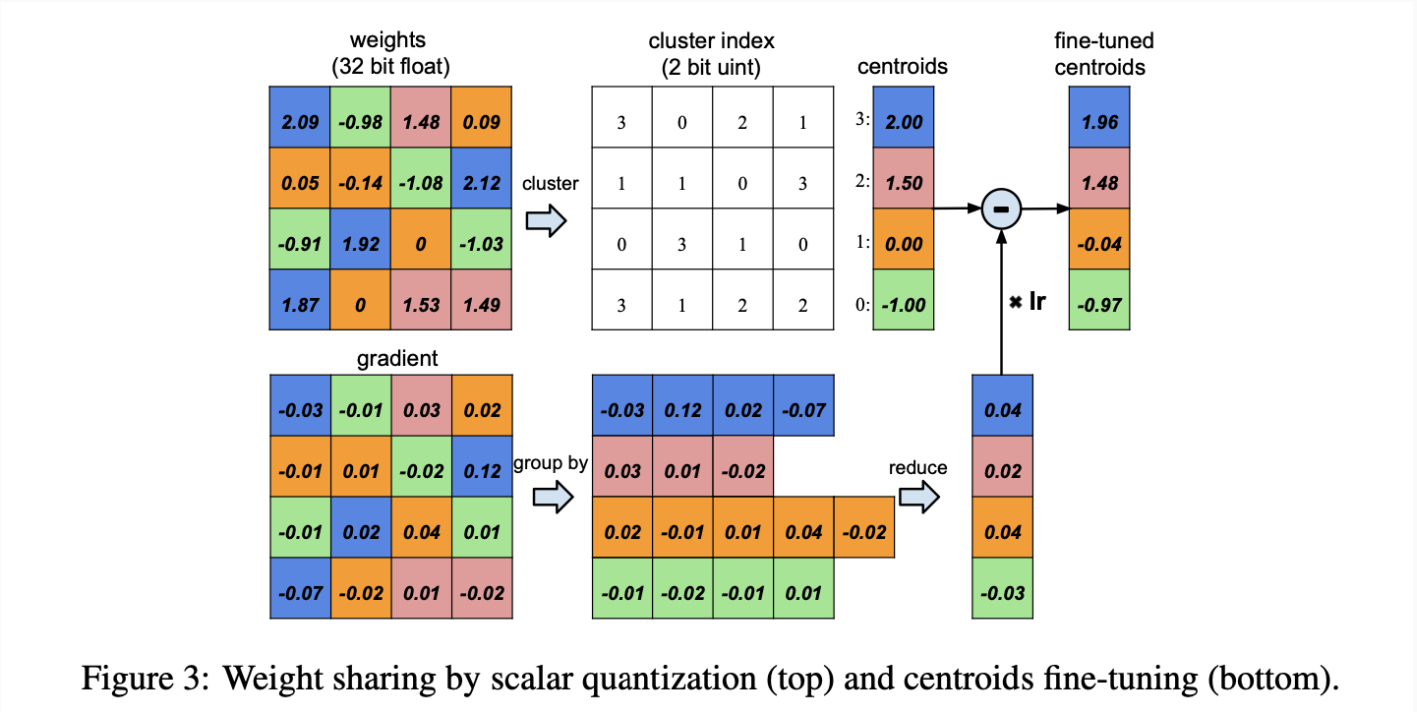


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

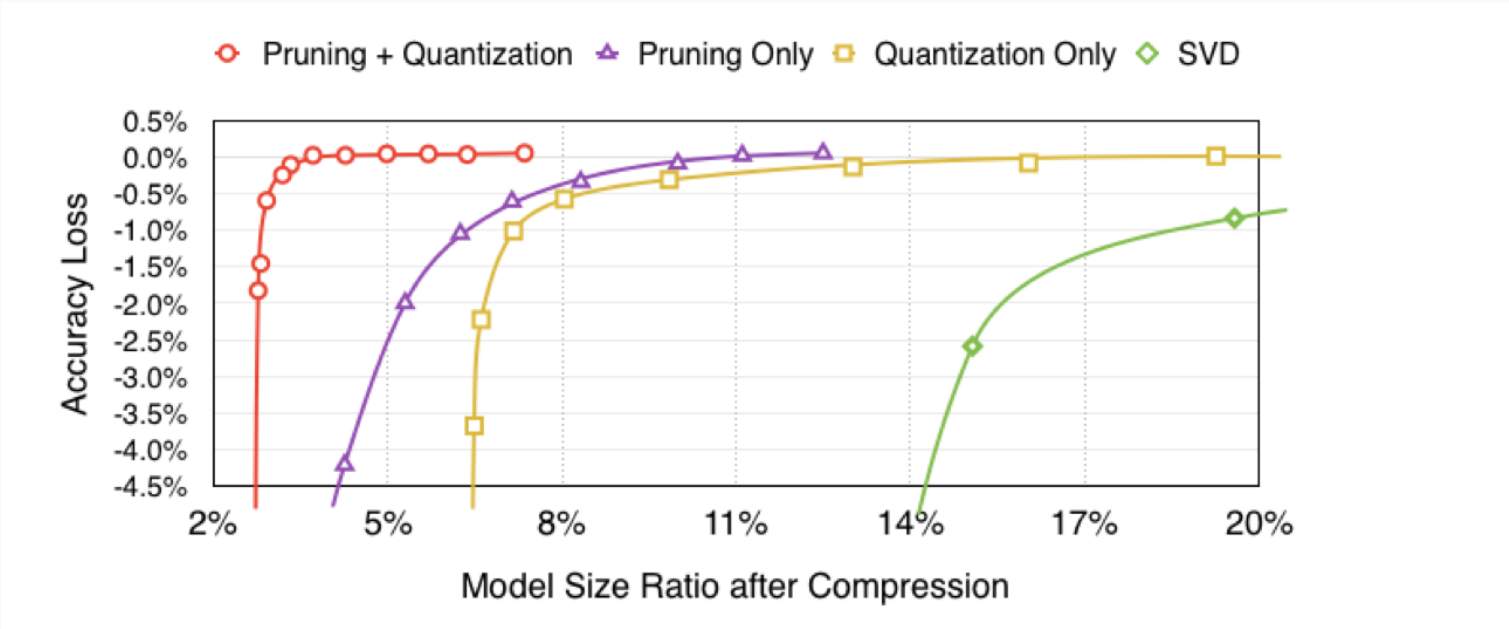
- Does it work? basically, there are averaging mechanisms, but loss of accuracy

Quantization of the weights (2)

- How to make it work?

Quantization of the weights (2)

- How to make it work? use retraining again !
  - How to retrain? train, cluster the weights, generate a dictionary of weights, assign of word in the dictionary to each weight , retrain the dictionary (but not the assignment)!
  - How to do that ? compute the gradients (as usual), average the gradients corresponding the same number in the dictionary, then update the weights!
- It works!



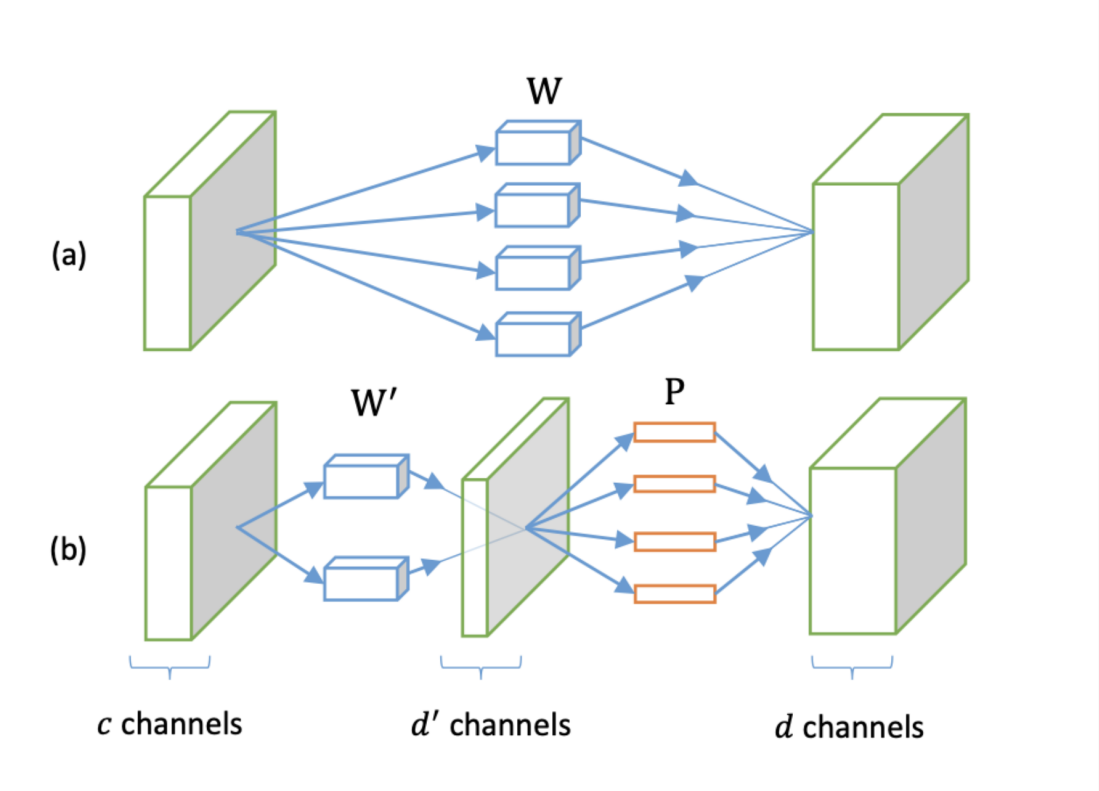
Outline

Approximation of Convolutional Layers (1)

source: Efficient and Accurate Approximations of Nonlinear Convolutional Networks Xiangyu Zhang et al. CVPR'14

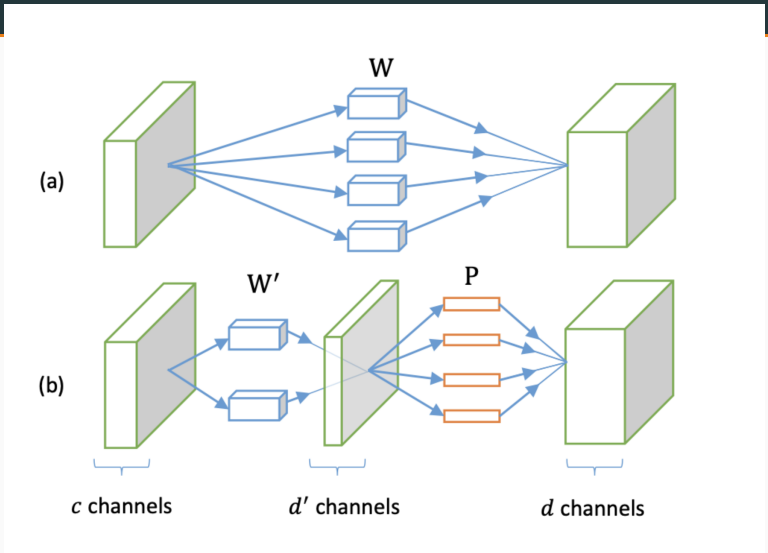
goal: find a more compact (low rank) representation of the convolutional layers

More specifically, the goal is to find such a decomposition



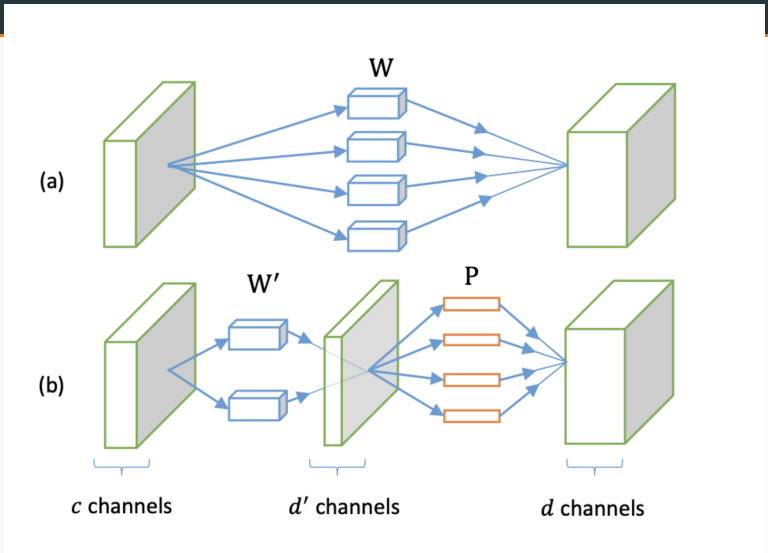
- exchange d times $k \times k \times c$ filters (storage k^2dc, flops $wh \times dk^2c$)
- against d' times $k \times k \times c$ filters and d times $1 \times 1 \times d'$ filters (storage $k^2d'c + dd'$, flops $wh \times (dk^2c + d'd)$)

Approximation of Convolutional Layers (2)



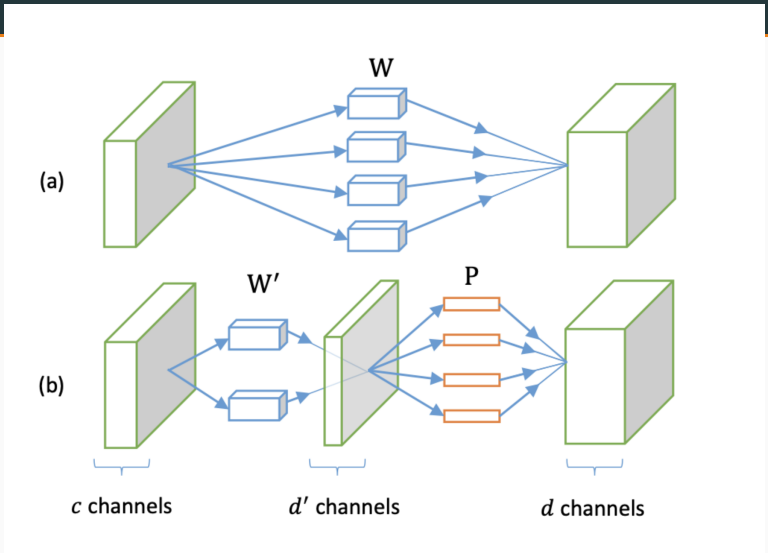
- Difficulty: what is the rank of W after the training process (SGD)?

Approximation of Convolutional Layers (2)



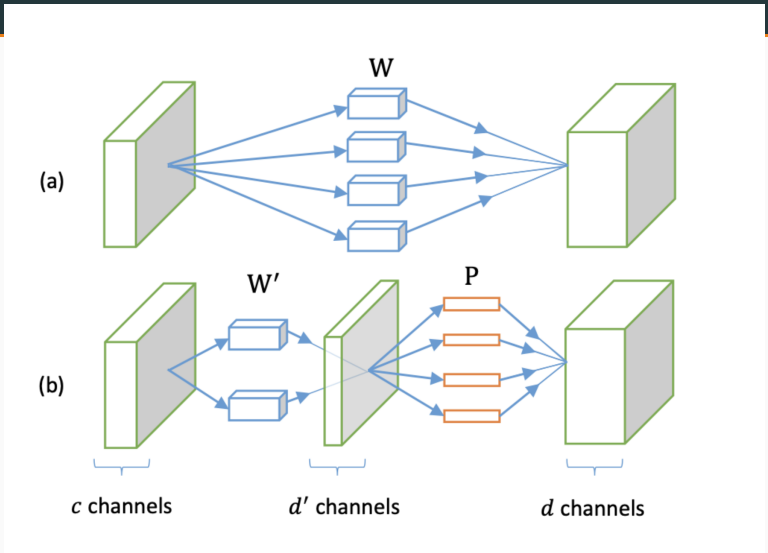
- Difficulty: what is the rank of W after the training process (SGD)? full
- What has a chance of being low rank ?

Approximation of Convolutional Layers (2)



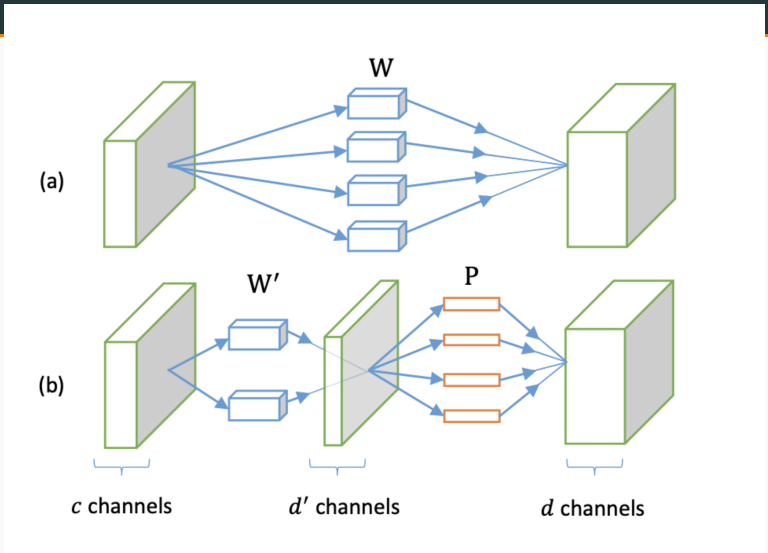
- Difficulty: what is the rank of W after the training process (SGD)? full
- What has a chance of being low rank ? $y = Wx \in R^d$ because x contains information about data! How to do this ?

Approximation of Convolutional Layers (2)



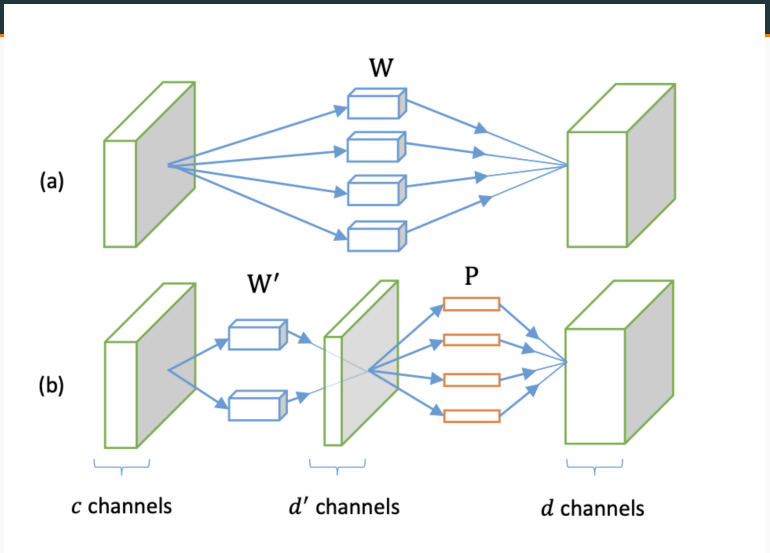
- Difficulty: what is the rank of W after the training process (SGD)? full
- What has a chance of being low rank ? $y = Wx \in R^d$ because x contains information about data! How to do this ?
  - Collect (through sampling) plenty (N) of y_i's (different locations and training images) (Y is a $N \times d$ matrix)
  - Compute the SVD of Y, write $y = M(y - \bar{y}) + \bar{y}$ where $M = P \times Q^T$ is low rank (d')

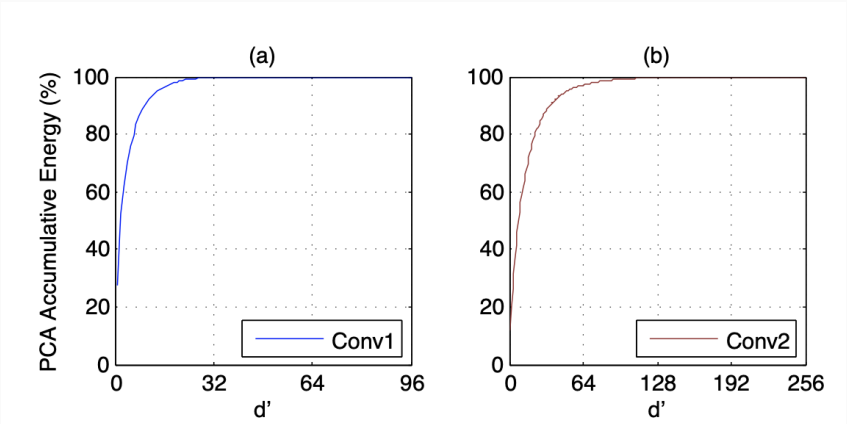
Approximation of Convolutional Layers (2)



- Difficulty: what is the rank of W after the training process (SGD)? full
- What has a chance of being low rank ? $y = Wx \in R^d$ because x contains information about data! How to do this ?
  - Collect (through sampling) plenty (N) of y_i's (different locations and training images) (Y is a $N \times d$ matrix)
  - Compute the SVD of Y, write $y = M(y - \bar{y}) + \bar{y}$ where $M = P \times Q^T$ is low rank (d')
  - And finally $y = P \times (Q' = Q^T W)x + b$
  - Q' is a $d' \times k^2 c$ matrix than can be used as W'

Approximation of Convolutional Layers (2)





- Difficulty: what is the rank of W after the training process (SGD)? full
- What has a chance of being low rank ? $y = Wx \in R^d$ because x contains information about data! How to do this ?
  - Collect (through sampling) plenty (N) of y_i's (different locations and training images) (Y is a $N \times d$ matrix)
  - Compute the SVD of Y, write $y = M(y - \bar{y}) + \bar{y}$ where $M = P \times Q^T$ is low rank (d')
  - And finally $y = P \times (Q' = Q^T W)x + b$
  - Q' is a $d' \times k^2 c$ matrix than can be used as W'
- It works!

Generalization TT Decompositions

source: Tensorizing Neural Networks, Novikov et al, NIPS'15

goal: generalization

- d filters of size $k \times k \times c$: $(k^2 c, d) \longrightarrow (k^2, d')$ and (d', d) (previous slide)
- this is only one possible option!
- Tensor Train Decompositions introduced by Oseledets et al.:
  - A is a tensor of dimension d. A in TT tensor format if
  - $A(j_1, j_2, \ldots, j_d) = G^{j_1} \times G^{j_d}$ where
  - G^{j_k} is a $r_{k-1} \times r_k$ matrix ($r_0 = r_d = 1$) and $r = \max r_k$ is the rank of the TT decomposition.
  - Decomposition is not unique and we look for low rank decompositions.
- Basic example:

  - Consider a tensor:

$$A(i_1, i_2, i_3) := i_1 + i_2 + i_3,$$
$$i_1 \in \{1,2,3\}, \quad i_2 \in \{1,2,3,4\}, \quad i_3 \in \{1,2,3,4,5\}.$$

  - Its TT-format:

$$A(i_1, i_2, i_3) = G_1[i_1] G_2[i_2] G_3[i_3],$$

  where

$$G_1[i_1] := \begin{bmatrix} i_1 & 1 \end{bmatrix}, \quad G_2[i_2] := \begin{bmatrix} 1 & 0 \\ i_2 & 1 \end{bmatrix}, \quad G_3[i_3] := \begin{bmatrix} 1 \\ i_3 \end{bmatrix}$$

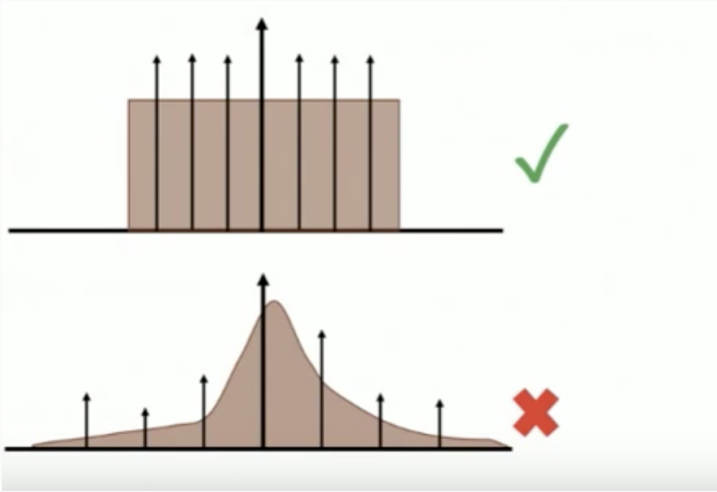
  - Check:

$$A(i_1, i_2, i_3) = \begin{bmatrix} i_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ i_2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ i_3 \end{bmatrix} =$$
$$= \begin{bmatrix} i_1 + i_2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ i_3 \end{bmatrix} = i_1 + i_2 + i_3.$$

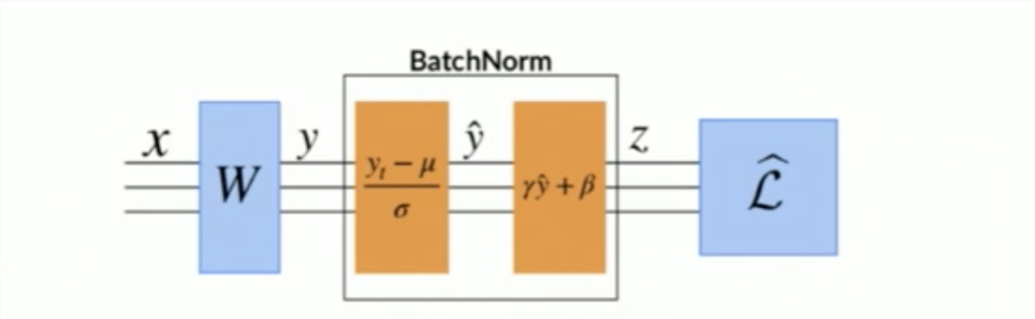
Quantization of the activations

source: Elaina Chai, Quantization Error in Neural Networks
goal: quantize activations (must be extremely cheap)

- compute the activations in FP32, use quantization to store them in INT8
- fixed intervals... Pb: if the order of magnitude of the different components are very different, it will not work!

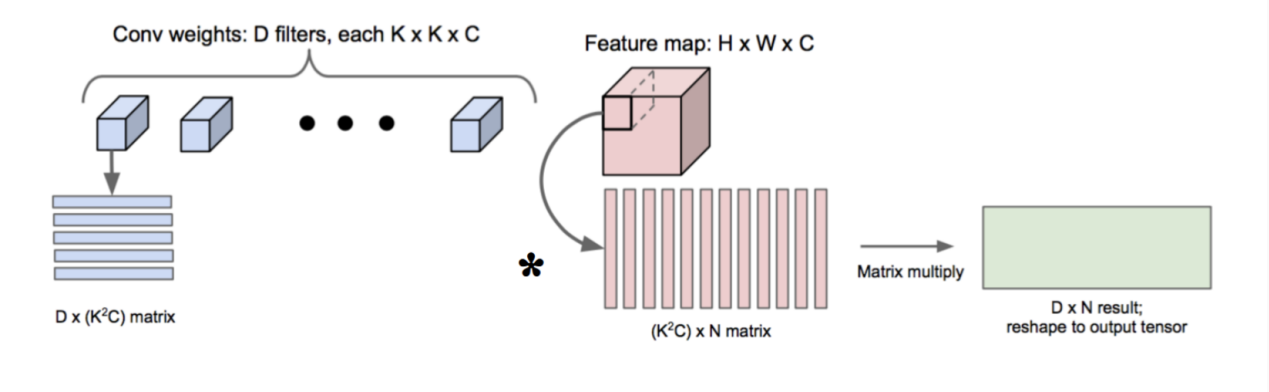


- Solution: change the network so that it does not happen
- Batch Normalization tends to homogenize the components (train γ and β)



Winograd Transformation

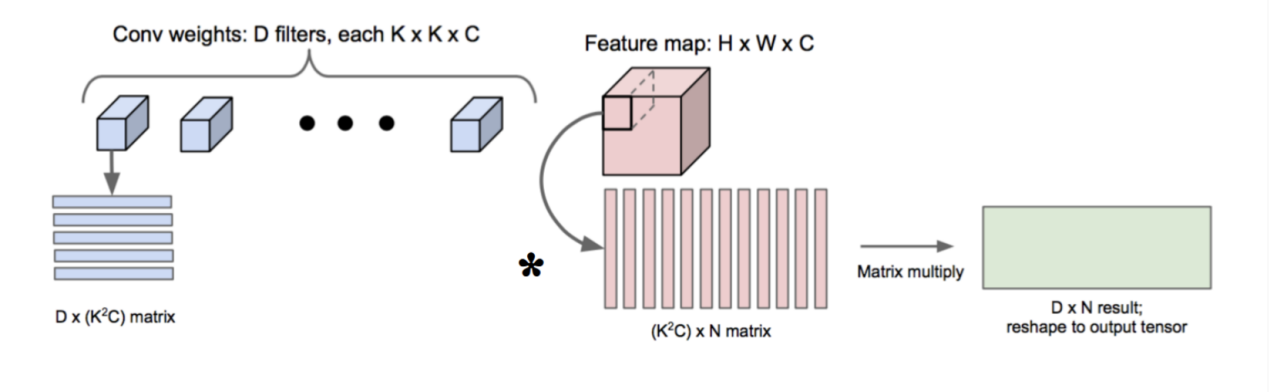
Fast Algorithms for Convolutional Neural Networks, A. Lavin et al., CVPR'16
Goal: take advantage of the particular structure of the activation matrix in convolutions



- Goal is to keep GPUs happy by doing GEMMs
- Toy example with an image 4×1 and a filter 3×1

Winograd Transformation

Fast Algorithms for Convolutional Neural Networks, A. Lavin et al., CVPR'16
Goal: take advantage of the particular structure of the activation matrix in convolutions



- Goal is to keep GPUs happy by doing GEMMs
- Toy example with an image 4×1 and a filter 3×1 , we obtain: ?

$$result = \begin{bmatrix} d0 & d1 & d2 \\ d1 & d2 & d3 \end{bmatrix} \begin{bmatrix} g0 \\ g1 \\ g2 \end{bmatrix} = \begin{bmatrix} m1+m2+m3 \\ m2-m3-m4 \end{bmatrix}$$

where ?

$$m_1 = (d_0 - d_2)g_0 \qquad m_2 = (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}$$
$$m_4 = (d_1 - d_3)g_2 \qquad m_3 = (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2}$$

- so we end up with 4 Mult (and 8 Add) instead of 6 Mult
- In general the improvement is from 36 Mult to 16 Mult (2.25x improvement) using Winograd transformation
- implemented in cuDNN since version 5.0

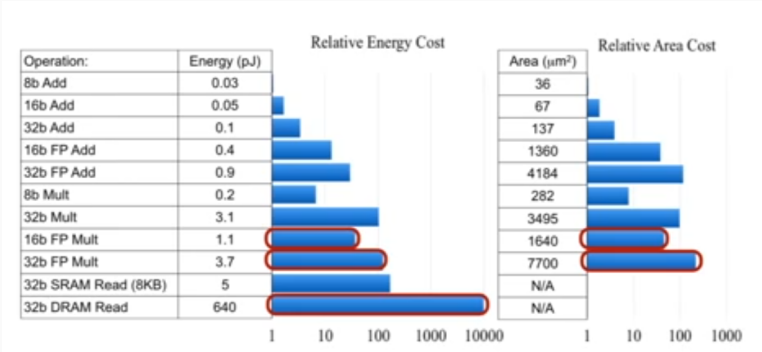
Computational Issues: Training Phase

Outline

Mixing FP32 and FP16 precisions

Source: Mixed Precision Training, Paulius Micikevicius et al., ICLR'18

Goal: consume less energy during the training phase by using mixed precision



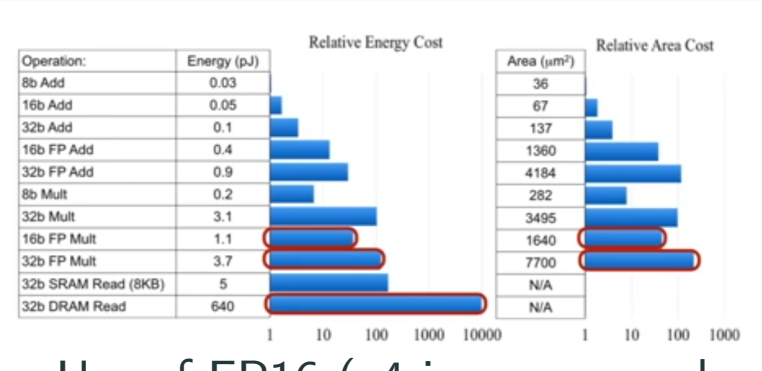
?

- Use of FP16 (x4 in energy and area)
- Algorithm:
-

Mixing FP32 and FP16 precisions

Source: Mixed Precision Training, Paulius Micikevicius et al., ICLR'18

Goal: consume less energy during the training phase by using mixed precision



?

- Use of FP16 (x4 in energy and area)
- Algorithm: store the weights in FP32, do forward and backward in FP16
-

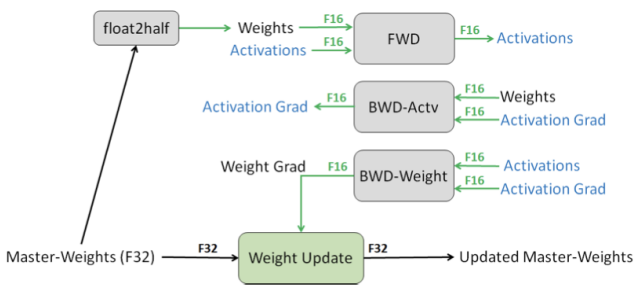
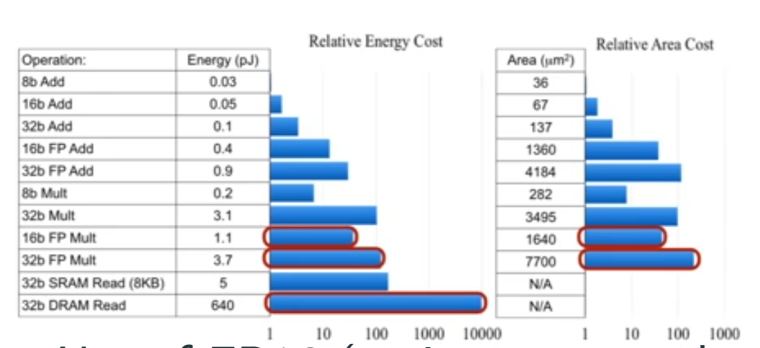


Figure 1: Mixed precision training iteration for a layer.

Mixing FP32 and FP16 precisions

Source: Mixed Precision Training, Paulius Micikevicius et al., ICLR'18

Goal: consume less energy during the training phase by using mixed precision



Operation:	Energy (pJ)	Area (μm²)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A

?
- Use of FP16 (x4 in energy and area)
- Algorithm: store the weights in FP32, do forward and backward in FP16
- It works!

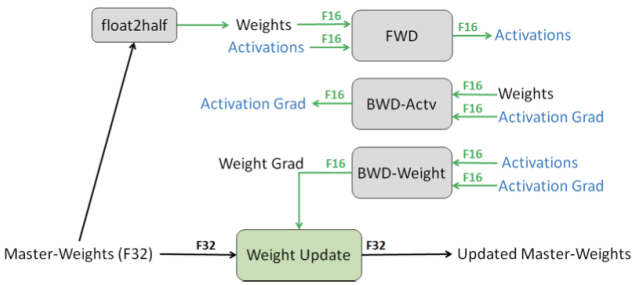
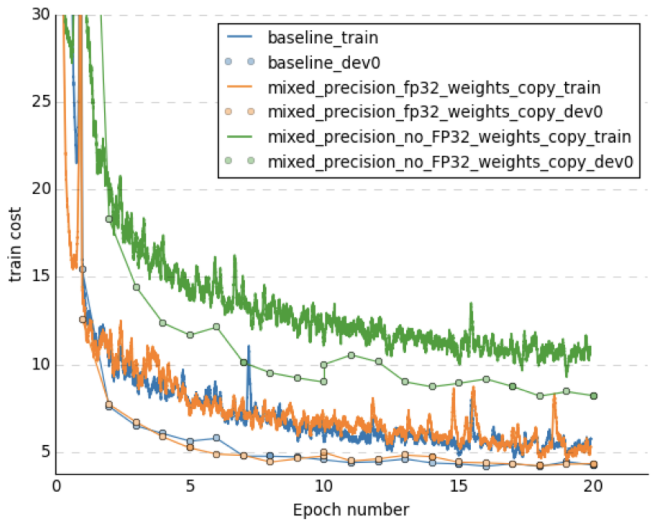


Figure 1: Mixed precision training iteration for a layer.



Outline

Parallelism

- Plenty of potential sources of parallelism
  1. Hyper Parameter search (shape of the network, batch size, loss function, learning rate and optimizer)
  2. Data, Spatial, Channel and Kernel Parallelisms: the input of the network is 4 dimensional: batch x channels x height x width, plenty of ways to split it and generate parallelisme (communication intensive, optimized `MPI_Allreduce` to exchange weights updates)
  3. Model Parallelism: the network itself can be deep (10s to 100s layers) and it can be split in turn: be careful, due to backward phase, a priori much much parallelism !
  4. Kernel (GPU) Parallelism: optimization of this special GEMM operation between the weight matrix (2D conversion of the convolutional filters) and the activation matrix (equivalent 2D conversion): special because of the redundancies in the activation matrix.

Parallelism

- Plenty of potential sources of parallelism
  1. Hyper Parameter search (shape of the network, batch size, loss function, learning rate and optimizer)
  2. Data, Spatial, Channel and Kernel Parallelisms: the input of the network is 4 dimensional: batch x channels x height x width, plenty of ways to split it and generate parallelisme (communication intensive, optimized `MPI_Allreduce` to exchange weights updates)
  3. Model Parallelism: the network itself can be deep (10s to 100s layers) and it can be split in turn: be careful, due to backward phase, a priori much much parallelism !
  4. Kernel (GPU) Parallelism: optimization of this special GEMM operation between the weight matrix (2D conversion of the convolutional filters) and the activation matrix (equivalent 2D conversion): special because of the redundancies in the activation matrix.
- Additional Difficulty / Opportunity w.r.t. Numerical Linear Algebra

Parallelism

- Plenty of potential sources of parallelism
  1. Hyper Parameter search (shape of the network, batch size, loss function, learning rate and optimizer)
  2. Data, Spatial, Channel and Kernel Parallelisms: the input of the network is 4 dimensional: batch x channels x height x width, plenty of ways to split it and generate parallelisme (communication intensive, optimized `MPI_Allreduce` to exchange weights updates)
  3. Model Parallelism: the network itself can be deep (10s to 100s layers) and it can be split in turn: be careful, due to backward phase, a priori much much parallelism !
  4. Kernel (GPU) Parallelism: optimization of this special GEMM operation between the weight matrix (2D conversion of the convolutional filters) and the activation matrix (equivalent 2D conversion): special because of the redundancies in the activation matrix.
- Additional Difficulty / Opportunity w.r.t. Numerical Linear Algebra
  - Data and Spatial parallelism influence the SGD (batch size in particular) and thus modify both the convergence speed and the accuracy at the end.

Parallelism

- Plenty of potential sources of parallelism
  1. Hyper Parameter search (shape of the network, batch size, loss function, learning rate and optimizer)
  2. Data, Spatial, Channel and Kernel Parallelisms: the input of the network is 4 dimensional: batch x channels x height x width, plenty of ways to split it and generate parallelisme (communication intensive, optimized `MPI_Allreduce` to exchange weights updates)
  3. Model Parallelism: the network itself can be deep (10s to 100s layers) and it can be split in turn: be careful, due to backward phase, a priori much much parallelism !
  4. Kernel (GPU) Parallelism: optimization of this special GEMM operation between the weight matrix (2D conversion of the convolutional filters) and the activation matrix (equivalent 2D conversion): special because of the redundancies in the activation matrix.
- Additional Difficulty / Opportunity w.r.t. Numerical Linear Algebra
  - Data and Spatial parallelism influence the SGD (batch size in particular) and thus modify both the convergence speed and the accuracy at the end.
  - Model parallelism does not induce any speedup, and data and spatial parallelism induce to many communications when done at large scale...

Parallelism

- Plenty of potential sources of parallelism
  1. Hyper Parameter search (shape of the network, batch size, loss function, learning rate and optimizer)
  2. Data, Spatial, Channel and Kernel Parallelisms: the input of the network is 4 dimensional: batch x channels x height x width, plenty of ways to split it and generate parallelisme (communication intensive, optimized `MPI_Allreduce` to exchange weights updates)
  3. Model Parallelism: the network itself can be deep (10s to 100s layers) and it can be split in turn: be careful, due to backward phase, a priori much much parallelism !
  4. Kernel (GPU) Parallelism: optimization of this special GEMM operation between the weight matrix (2D conversion of the convolutional filters) and the activation matrix (equivalent 2D conversion): special because of the redundancies in the activation matrix.
- Additional Difficulty / Opportunity w.r.t. Numerical Linear Algebra
  - Data and Spatial parallelism influence the SGD (batch size in particular) and thus modify both the convergence speed and the accuracy at the end.
  - Model parallelism does not induce any speedup, and data and spatial parallelism induce to many communications when done at large scale...
  - but you can change the rules, what will once again change the convergence and the accuracy, sometimes for the better and sometimes for the worse!

Gradient Descent Revisited: Stochastic Gradient Descent (1)

General Idea

- Start with one example (image, class) (x, y)
- Forward propagation for $\texttt{net}(x)$ as for a classical task graph
- Evaluation of $\text{loss}(\texttt{net}(x), y)$
- Backpropagation of loss to determine its sensitivity to the different parameters as for a classical task graph
- Update the weights $W_i(t+1) = W_i(t) - \epsilon \frac{\partial \text{loss}}{\partial W_i}$

Consequences

- efficiency: depending on the size of the image, GPU usage might not be optimal (increasing batch size increases the size of the GEMMs)
- general belief: smaller training accuracy but better ability to generalize to other examples (better test accuracy)

Gradient Descent Revisited: Mini Batch Stochastic Gradient Descent (2)

General Idea

- Start with a set of B examples (image, class) $(x^{(k)}, y^{(k)})$
- Forward propagation for $\texttt{net}(x^{(k)})$ as for a classical task graph
- Evaluation of $L_k = \text{loss}(\texttt{net}(x^{(k)}), y^{(k)})$
- Backpropagation of L_k to determine its sensitivity to the different parameters as for a classical task graph
- Update the weights $W_i(t+1) = W_i(t) - \epsilon \sum_j \frac{\partial L_j}{\partial W_i}$

Consequences

- efficiency: with a batch size of B,
  - same size for network weights, but activation sizes x B
  - cost is x B (even better for GPUs)
  - everything (including activation stored for the backward phase) must fit into memory
- general belief: a large batch slows down convergence (less frequent updates) and affects generalization ability.

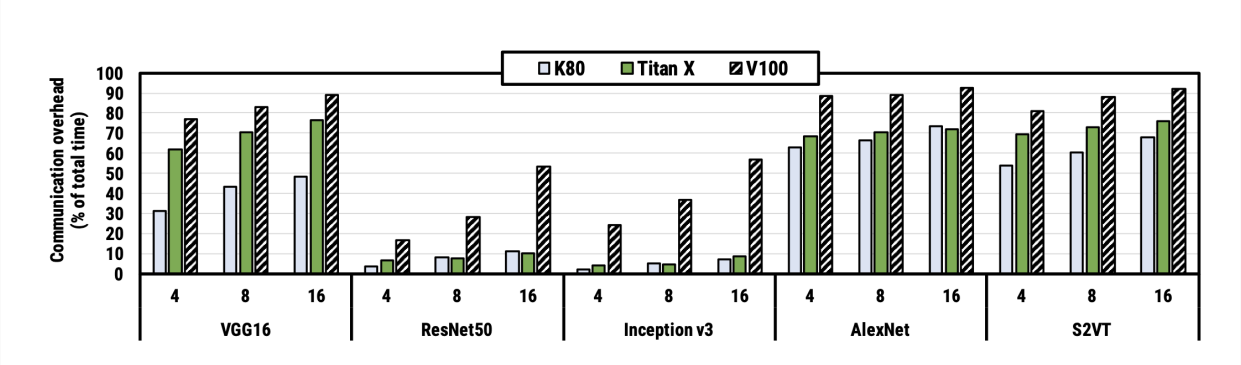
Data Parallelism: Mini Batch Parallelism

Principe

- Let us suppose we have N (identical) GPUs
- Each GPU can perform SGD with a batch size of B
- We train in parallel a batch size of size BN (B on each GPU)
- Each resource computes a gradient (size of all weights)
- MPI `Allreduce` is used to compute a global gradient

Limitations

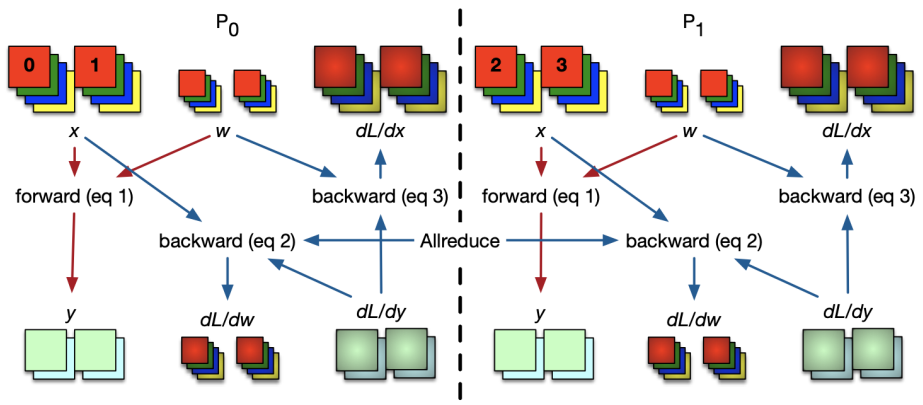
- MPI `Allreduce` is expensive when N becomes large



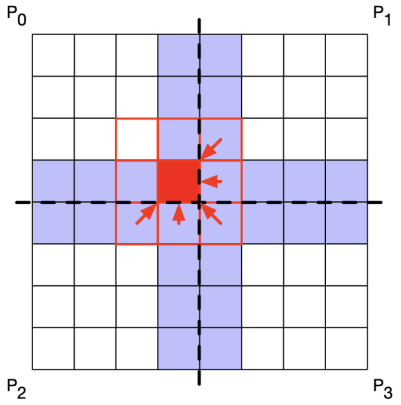
- N large, NB very large $\longrightarrow$ slow convergence and poor generalization
- there is a strict barrier at the end of each `Allreduce` operation, bad when N is large

Spatial Parallelism: Split Images

Source: "Improving Strong-Scaling of CNN Training by Exploiting Finer-Grained Parallelism" Nikoli Dryden et al.



(a) Sample parallelism



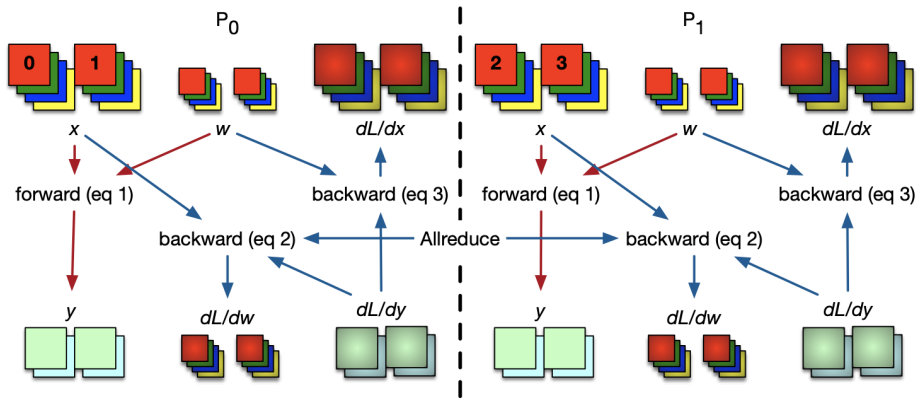
(b) Spatial parallelism halo exchange

Principe

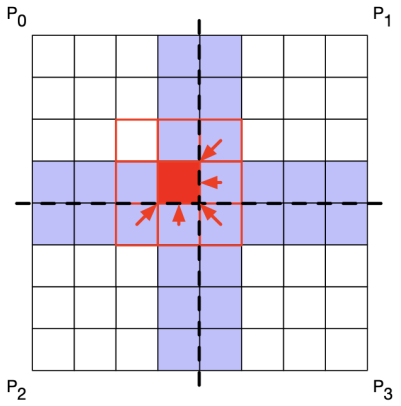
- Split the input image into 4 (slightly overlapping) parts
- Perform the forward phase independently on 4 GPUs
- Perform the backward phase (almost independently, halo communications) on the 4 GPUs

Spatial Parallelism: Split Images

Source: "Improving Strong-Scaling of CNN Training by Exploiting Finer-Grained Parallelism" Nikoli Dryden et al.



(a) Sample parallelism



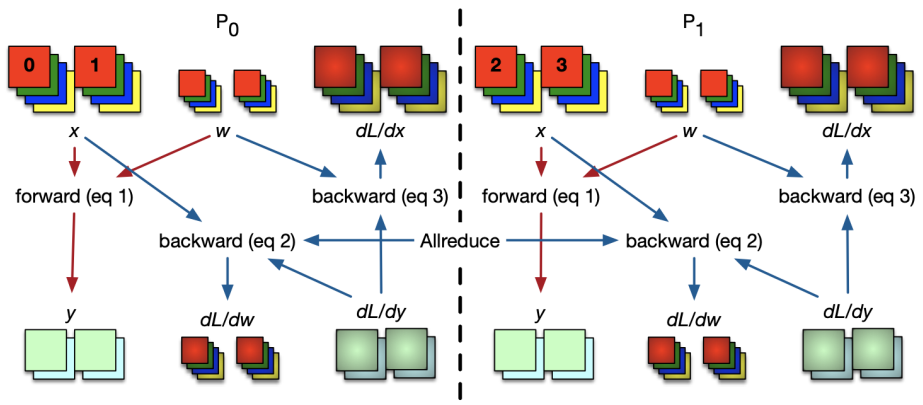
(b) Spatial parallelism halo exchange

Principe

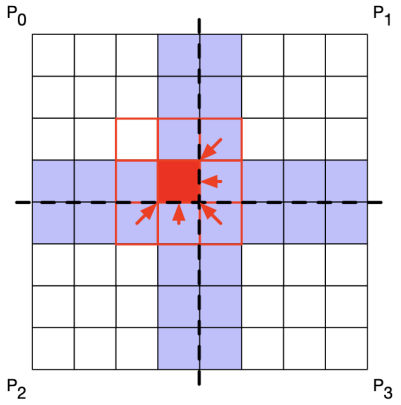
- Split the input image into 4 (slightly overlapping) parts
- Perform the forward phase independently on 4 GPUs
- Perform the backward phase (almost independently, halo communications) on the 4 GPUs
- It looks a priori great, but then you need to update the weights...

Spatial Parallelism: Split Images

Source: "Improving Strong-Scaling of CNN Training by Exploiting Finer-Grained Parallelism" Nikoli Dryden et al.



(a) Sample parallelism



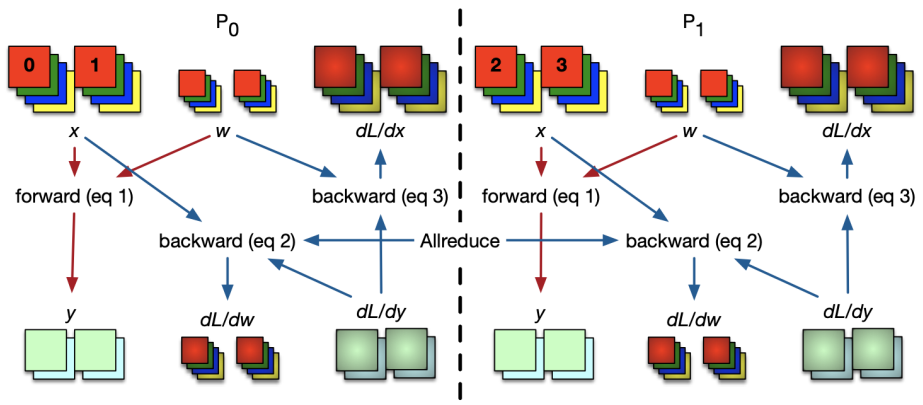
(b) Spatial parallelism halo exchange

Principe

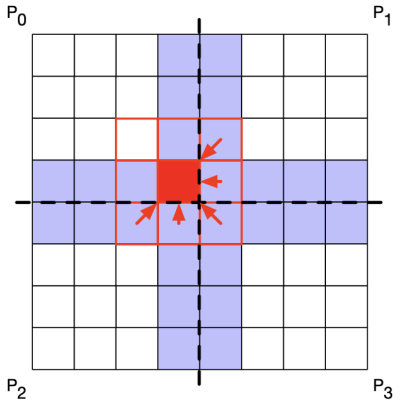
- Split the input image into 4 (slightly overlapping) parts
- Perform the forward phase independently on 4 GPUs
- Perform the backward phase (almost independently, halo communications) on the 4 GPUs
- It looks a priori great, but then you need to update the weights...
  - for fully connected layers, it is great

Spatial Parallelism: Split Images

Source: "Improving Strong-Scaling of CNN Training by Exploiting Finer-Grained Parallelism" Nikoli Dryden et al.



(a) Sample parallelism



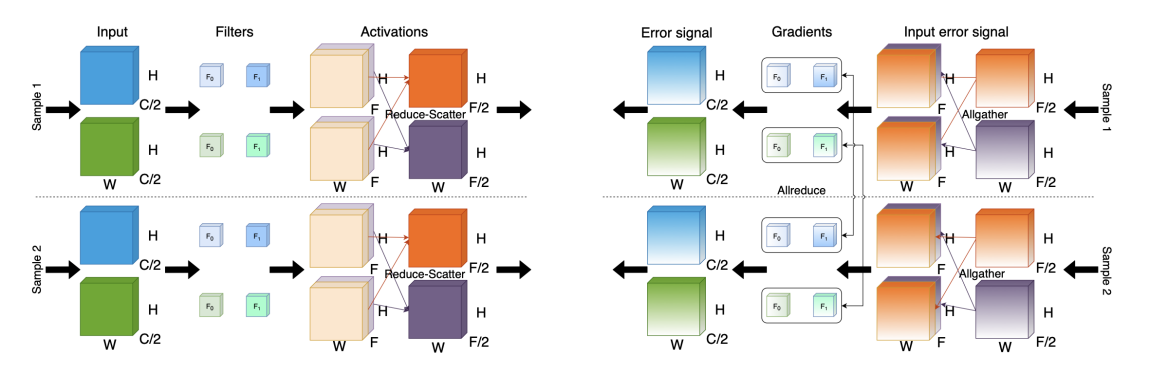
(b) Spatial parallelism halo exchange

Principe

- Split the input image into 4 (slightly overlapping) parts
- Perform the forward phase independently on 4 GPUs
- Perform the backward phase (almost independently, halo communications) on the 4 GPUs
- It looks a priori great, but then you need to update the weights...
  - for fully connected layers, it is great
  - but for convolutional layers, it has the same cost as data parallelism!

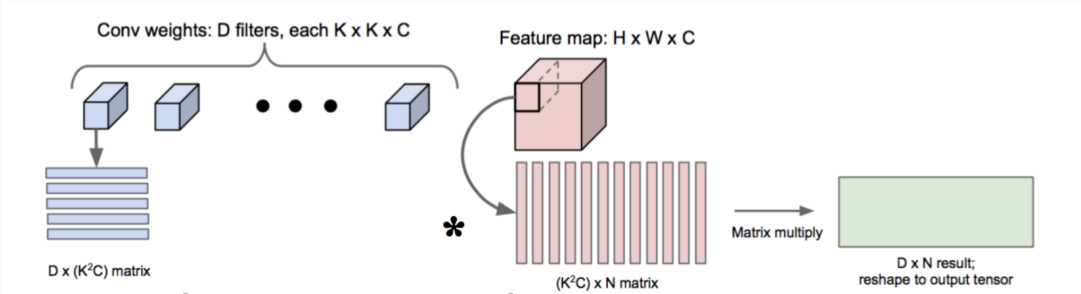
Filter Parallelism: Split Filters

"Channel and Filter Parallelism for Large-Scale CNN Training" Dryden, SC19



General Idea

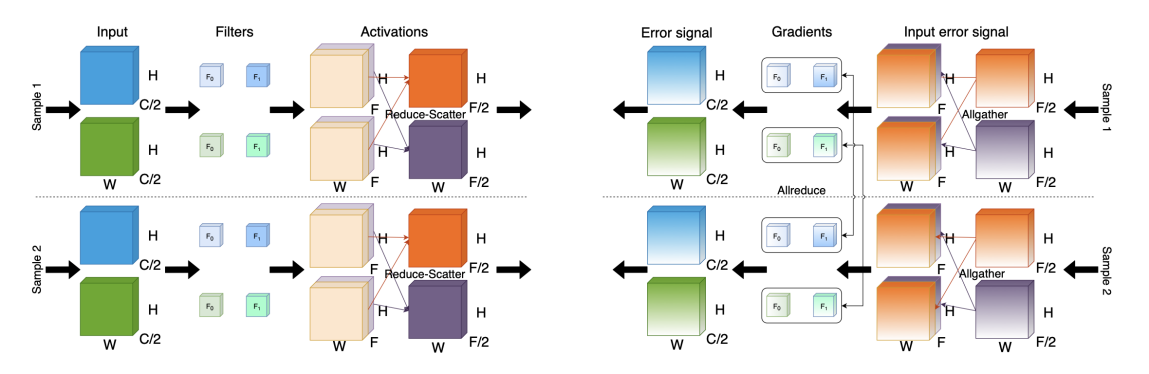
- start with c channels, produces d channels with d $k \times k \times c$ filters



- plenty of opportunities for parallelism with 2 GPUs
  - take $D/2$ filters on each GPU

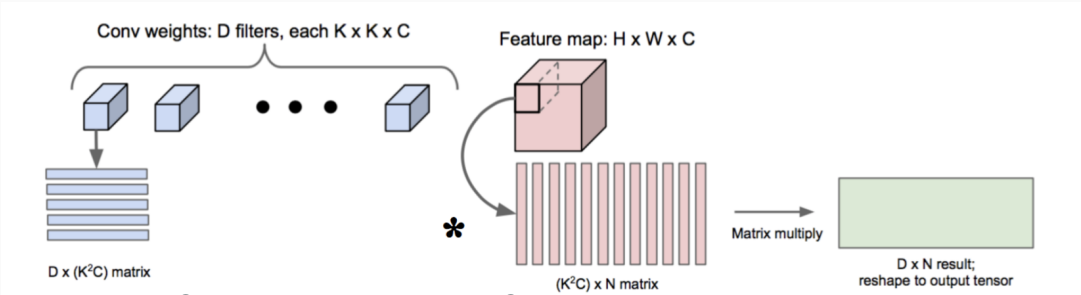
Filter Parallelism: Split Filters

"Channel and Filter Parallelism for Large-Scale CNN Training" Dryden, SC19



General Idea

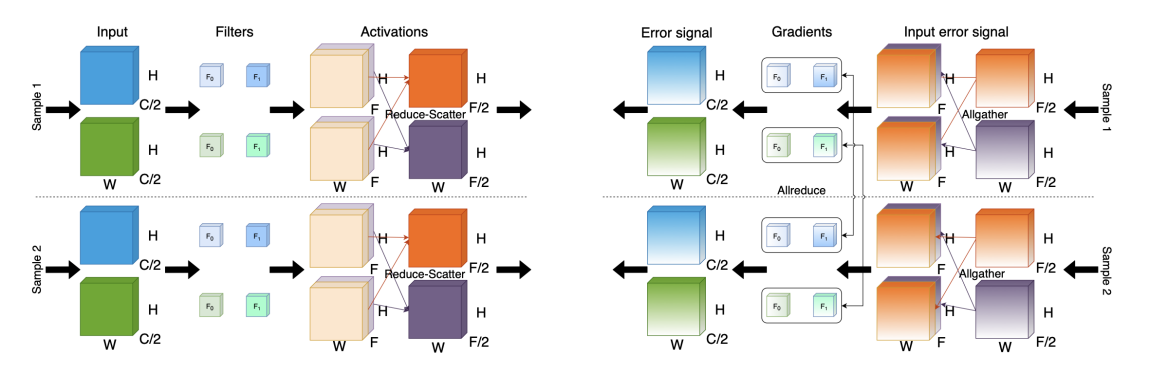
- start with c channels, produces d channels with d $k \times k \times c$ filters



- plenty of opportunities for parallelism with 2 GPUs
  - take $D/2$ filters on each GPU the input feature map (activation) must be replicated on both and the output feature map must be rebuilt
  - take $C/2$ channels on each GPU

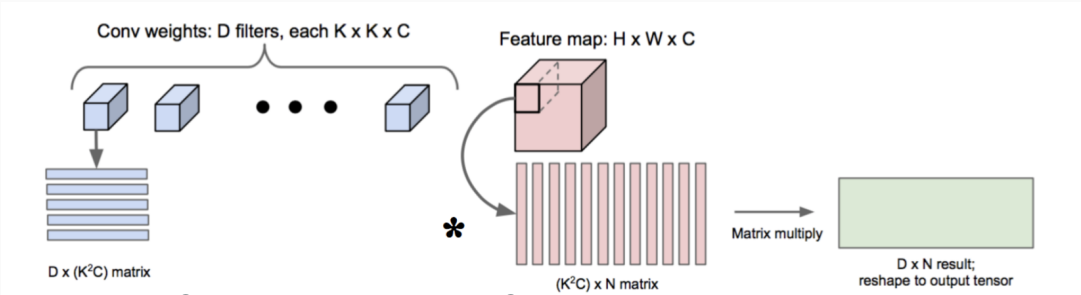
Filter Parallelism: Split Filters

"Channel and Filter Parallelism for Large-Scale CNN Training" Dryden, SC19



General Idea

- start with c channels, produces d channels with d $k \times k \times c$ filters



- plenty of opportunities for parallelism with 2 GPUs
  - take $D/2$ filters on each GPU the input feature map (activation) must be replicated on both and the output feature map must be rebuilt
  - take $C/2$ channels on each GPU the input feature map must be split and the output feature map must be "reduced"
- Ideas are close to 2.5D algorithms by Demmel et al.

Outline

Data Parallelism – Opportunities

General Idea

- N GPUs
- P_k processes the mini-batch of size B $(X^{(k)}, Y^{(k)})$ and computes $\Delta W_i^{(k)} = \frac{\alpha}{\sum_k |X^{(k)}|} \sum_j \frac{\partial L_j}{\partial W_i}$
- to do the same thing as what would be achieved with a batch size of NB
  - we need to update the weights when all NB batches have been processed
  - ie to perform $\Delta W_i = \sum_k \Delta W_i^{(k)}$ after mini-batch
  - inducing a lot of communications (`Allreduce`
  - a lot of synchronizations (BSP)

Opportunities

1. to limit synchronizations:

Data Parallelism – Opportunities

General Idea

- N GPUs
- P_k processes the mini-batch of size B $(X^{(k)}, Y^{(k)})$ and computes $\Delta W_i^{(k)} = \frac{\alpha}{\sum_k |X^{(k)}|} \sum_j \frac{\partial L_j}{\partial W_i}$
- to do the same thing as what would be achieved with a batch size of NB
  - we need to update the weights when all NB batches have been processed
  - ie to perform $\Delta W_i = \sum_k \Delta W_i^{(k)}$ after mini-batch
  - inducing a lot of communications (`Allreduce`
  - a lot of synchronizations (BSP)

Opportunities

1. to limit synchronizations: consider more asynchronous versions

Data Parallelism – Opportunities

General Idea

- N GPUs
- P_k processes the mini-batch of size B $(X^{(k)}, Y^{(k)})$ and computes $\Delta W_i^{(k)} = \frac{\alpha}{\sum_k |X^{(k)}|} \sum_j \frac{\partial L_j}{\partial W_i}$
- to do the same thing as what would be achieved with a batch size of NB
  - we need to update the weights when all NB batches have been processed
  - ie to perform $\Delta W_i = \sum_k \Delta W_i^{(k)}$ after mini-batch
  - inducing a lot of communications (`Allreduce`
  - a lot of synchronizations (BSP)

Opportunities

1. to limit synchronizations: consider more asynchronous versions
2. to limit communications:

Data Parallelism – Opportunities

General Idea

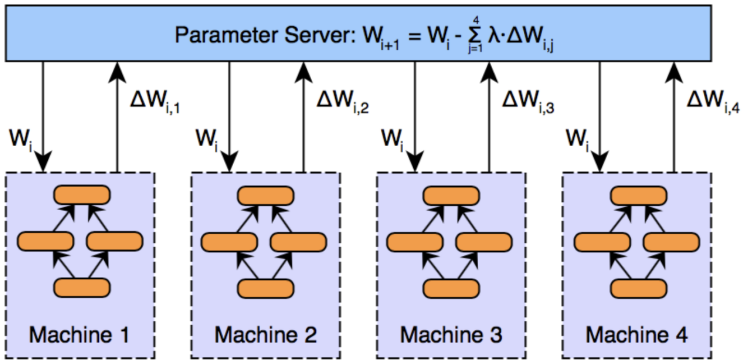
- N GPUs
- P_k processes the mini-batch of size B $(X^{(k)}, Y^{(k)})$ and computes $\Delta W_i^{(k)} = \frac{\alpha}{\sum_k |X^{(k)}|} \sum_j \frac{\partial L_j}{\partial W_i}$
- to do the same thing as what would be achieved with a batch size of NB
  - we need to update the weights when all NB batches have been processed
  - ie to perform $\Delta W_i = \sum_k \Delta W_i^{(k)}$ after mini-batch
  - inducing a lot of communications (`Allreduce`
  - a lot of synchronizations (BSP)

Opportunities

1. to limit synchronizations: consider more asynchronous versions
2. to limit communications: consider compacting weight updates

Data Parallelism – Asynchronous Versions (1)

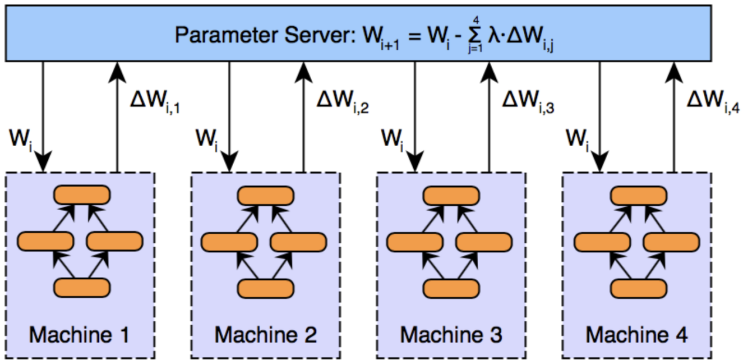
Source: Large Scale Distributed Deep Networks, Jeffrey Dean et al., NIPS'12



- use a centralized server for updates
- update the weights each time a contribution is received by one P_k
- the server sends back the new weight to P_k
- Advantages:

Data Parallelism – Asynchronous Versions (1)

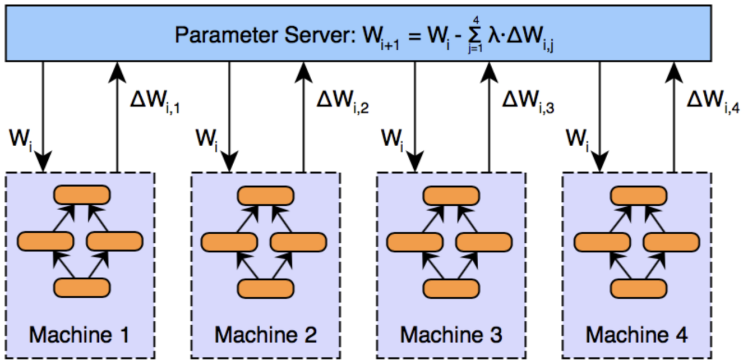
Source: Large Scale Distributed Deep Networks, Jeffrey Dean et al., NIPS'12

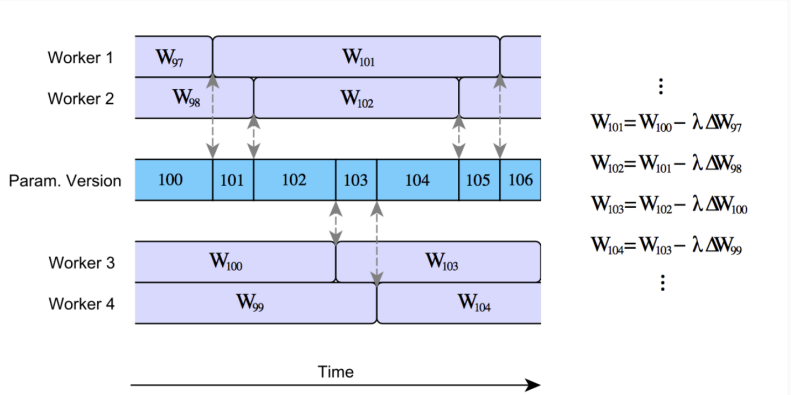


- use a centralized server for updates
- update the weights each time a contribution is received by one P_k
- the server sends back the new weight to P_k
- Advantages: no more synchronization
- Drawback:

Data Parallelism – Asynchronous Versions (1)

Source: Large Scale Distributed Deep Networks, Jeffrey Dean et al., NIPS'12

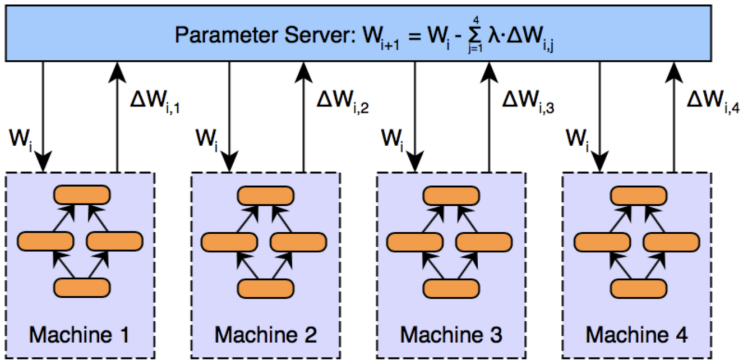


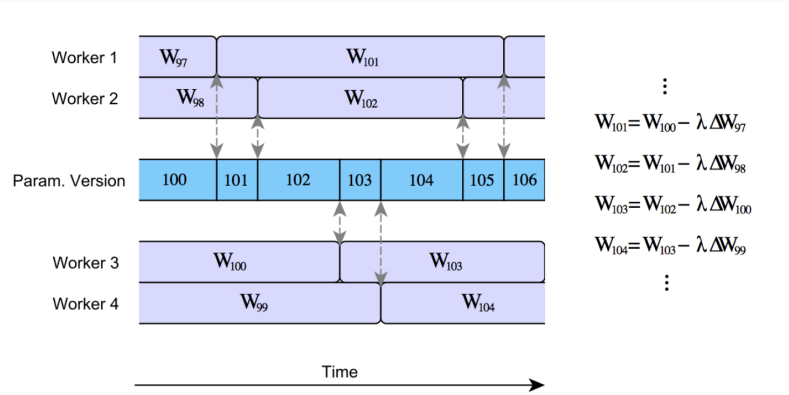


- use a centralized server for updates
- update the weights each time a contribution is received by one P_k
- the server sends back the new weight to P_k
- Advantages: no more synchronization
- Drawback: weight updates are done w.r.t. to non-consistent weights)-;
- called average gradient staleness

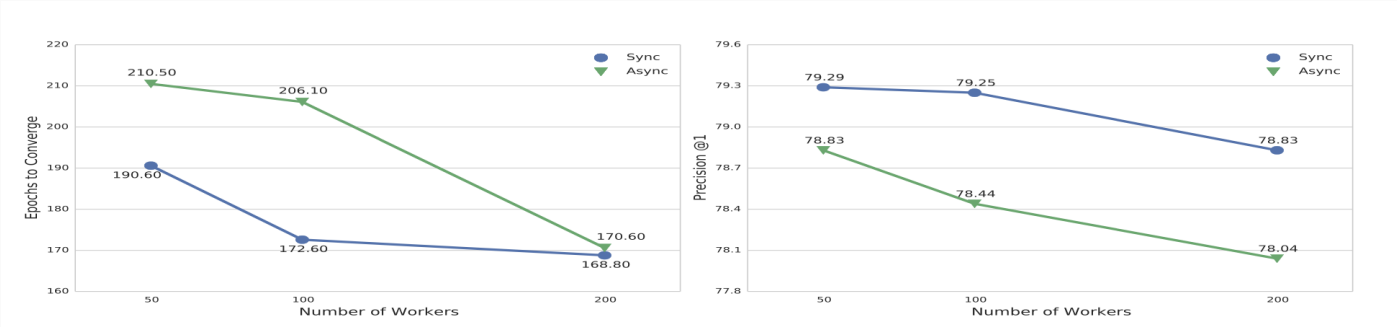
Data Parallelism – Asynchronous Versions (1)

Source: Large Scale Distributed Deep Networks, Jeffrey Dean et al., NIPS'12





- use a centralized server for updates
- update the weights each time a contribution is received by one P_k
- the server sends back the new weight to P_k
- Advantages: no more synchronization
- Drawback: weight updates are done w.r.t. to non-consistent weights)-;
- called average gradient staleness
- In practice affects accuracy at the end (but parallelism Ok)



Data Parallelism – Asynchronous Versions (2)

Source: Revisiting Distributed Synchronous SGD, Jianmin Chen et al., ICLR'16

- reserve 5% extra processors $N' = 1.05N$
- use the synchronized version, but wait only for N updates

Data Parallelism – Asynchronous Versions (2)

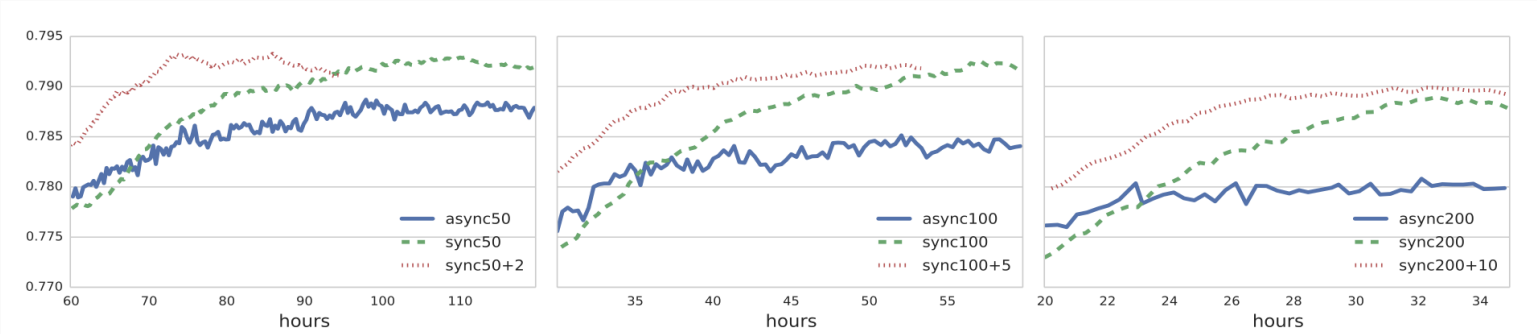
Source: Revisiting Distributed Synchronous SGD, Jianmin Chen et al., ICLR'16

- reserve 5% extra processors $N' = 1.05N$
- use the synchronized version, but wait only for N updates
- remember that mini-batches are built with randomly chosen images and have all the exact same complexity

Data Parallelism – Asynchronous Versions (2)

Source: Revisiting Distributed Synchronous SGD, Jianmin Chen et al., ICLR'16

- reserve 5% extra processors $N' = 1.05N$
- use the synchronized version, but wait only for N updates
- remember that mini-batches are built with randomly chosen images and have all the exact same complexity
- 5% of spare processors is enough to make homogeneous GPUs actually homogeneous



Data Parallelism – Compressed Versions

Source: Scalable distributed DNN training using commodity gpu cloud computing, Nikko Strom, INTERSPEECH'15

- Amazon AWS
- `Allreduce` operations are expected to have low performance
- Solution: send smaller updates (synchronous or even P2P based in the paper)
  - pruning:

Data Parallelism – Compressed Versions

Source: Scalable distributed DNN training using commodity gpu cloud computing, Nikko Strom, INTERSPEECH'15

- Amazon AWS
- `Allreduce` operations are expected to have low performance
- Solution: send smaller updates (synchronous or even P2P based in the paper)
  - pruning: if the gradient update is smaller than a threshold, make it zero

Data Parallelism – Compressed Versions

Source: Scalable distributed DNN training using commodity gpu cloud computing, Nikko Strom, INTERSPEECH'15

- Amazon AWS
- `Allreduce` operations are expected to have low performance
- Solution: send smaller updates (synchronous or even P2P based in the paper)
  - pruning: if the gradient update is smaller than a threshold, make it zero
  - quantization:

Data Parallelism – Compressed Versions

Source: Scalable distributed DNN training using commodity gpu cloud computing, Nikko Strom, INTERSPEECH'15

- Amazon AWS
- `Allreduce` operations are expected to have low performance
- Solution: send smaller updates (synchronous or even P2P based in the paper)
  - pruning: if the gradient update is smaller than a threshold, make it zero
  - quantization: gradients can only take very few values
  - delayed update:

Data Parallelism – Compressed Versions

Source: Scalable distributed DNN training using commodity gpu cloud computing, Nikko Strom, INTERSPEECH'15

- Amazon AWS
- `Allreduce` operations are expected to have low performance
- Solution: send smaller updates (synchronous or even P2P based in the paper)
  - pruning: if the gradient update is smaller than a threshold, make it zero
  - quantization: gradients can only take very few values
  - delayed update: keep (locally) of copy of what should have been committed to push it later!

Compression	**Update Size**	**Reduction**
None (32-bit floating point)	58.4 MB	-
16-bit floating point	29.2 MB	50%
Quantized, $\tau=2$	0.21 MB	99.6%

- Drawbacks
  - experiments with too little quantization
  - on small models where the extra cost of quantization is significant
- there are both accuracy and performance issues in the paper...

Conclusion

Conclusion and perspectives

- Inference phase: performed with under strong memory / energy constraints
- The training phase induces a lot of calculations and memory (peak) usage
- Many original techniques for inference
  - Train with full precision
  - then Prune, Quantize, Compress
  - and then use retraining !
- Many opportunities to find parallelism in training phase
  - but most of them induce large communications
  - again, pruning, quantization and compression can be used!
  - and original techniques can be designed to trade memory / communications agains computations

Conclusion and perspectives

- Inference phase: performed with under strong memory / energy constraints
- The training phase induces a lot of calculations and memory (peak) usage
- Many original techniques for inference
  - Train with full precision
  - then Prune, Quantize, Compress
  - and then use retraining !
- Many opportunities to find parallelism in training phase
  - but most of them induce large communications
  - again, pruning, quantization and compression can be used!
  - and original techniques can be designed to trade memory / communications agains computations
- Difficulties
  - This is a field where practice is ahead of theory (by far)
  - based on a very experimental approach: "it's a good idea because it works"