Advanced Machine Learning (Semester 1 2023)

Architecture of Machine Learning Projects

Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

July 10, 2023

Recap

Machine learning methods we saw so far fall either into the category of **supervised** or **unsupervised** or **reinforcement** machine learning.

neural networks (NN, CNN, Autoencoder)

Random Forest

SVM

regression

...

Hierarchical Clustering

PCA

...

Reinforcement Learning

actor-critic

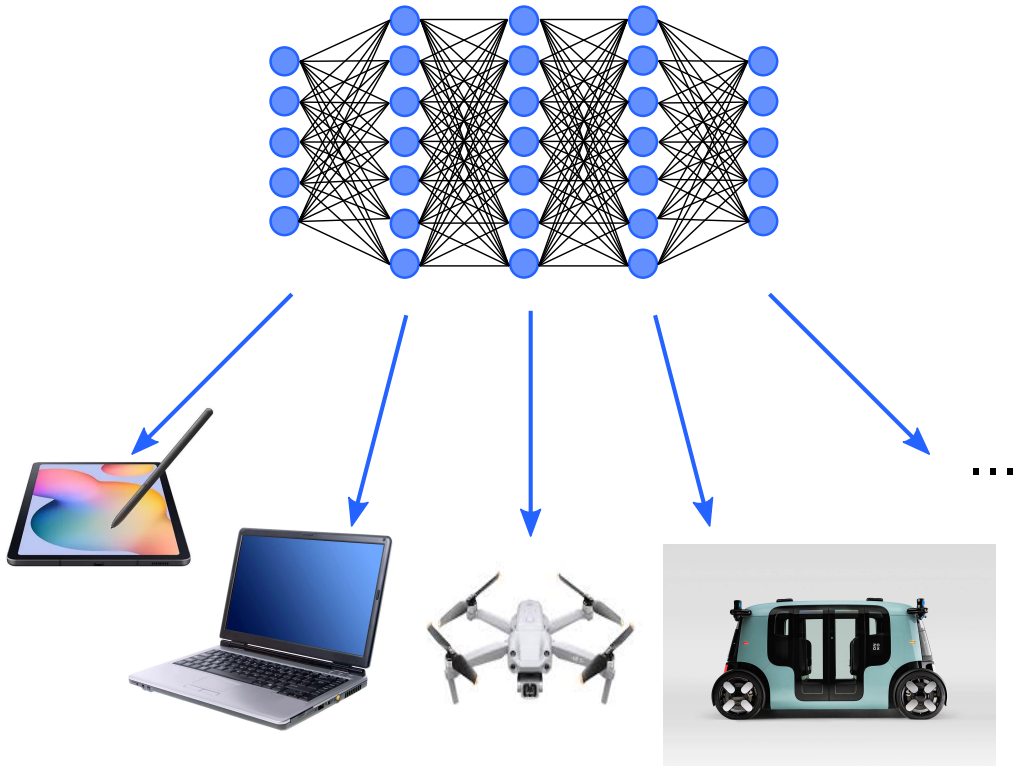
Deep Reinforcement Learning

...

Motivation

Challenges with large machine learning models

Size: We train and then pickle the model. Model sizes can be large and hard to distribute.



Motivation

Challenges with large machine learning models

Speed: Increasing training time limits productivity. We need a reasonable trade-off.

validation error rate on 224 $\times$ 224 images from `https://image-net.org/`, benchmarked with `fb.resnet.torch` using four M40 GPUs

Network	Top-1 error	Top-5 error	training time
ResNet-18	30.43%	10.76%	2.5 days
ResNet-50	24.01%	7.02%	5 days
ResNet-101	22.44%	6.21%	1 week
ResNet-152	22.16%	6.16%	1.5 weeks

The Top-1 error indicates whether the top class (the one with the highest probability) matches the target label.
The Top-5 error indicates whether one of the top 5 predictions is matching the target label.

Motivation

Challenges with large machine learning models

Energy Efficiency: this limits the application of machine learning models **even after training**

on mobile devices: drains battery
on data centers: increases the Total Cost of Ownership (TCO)

example: a single game of AlphaGo, involving 1920 CPUs and 280 GPUs, produces about a 3000 USD electricity bill

Motivation

analyzing the problem:

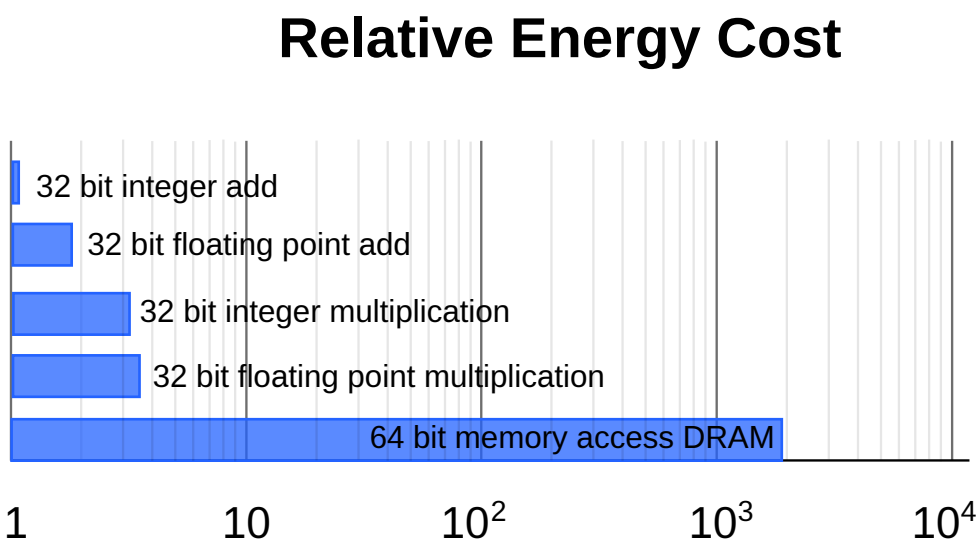
larger models $\rightarrow$ more memory references $\rightarrow$ more energy

Arithmetic operation	ADD	MUL
8-bit Integer	0.03 pJ	0.2 pJ
16-bit Floating point	0.4 pJ	1.1 pJ
32-bit Integer	0.1 pJ	3.1 pJ
32-bit Floating point	0.9 pJ	3.7 pJ

TABLE I: Approximate energy costs for different arithmetic operations in 45nm 0.9V.

Memory size	64-bit Memory access
8KB	10 pJ
32KB	20 pJ
1 MB	100 pJ
DRAM	1.3-2.6 nJ

TABLE II: Memory access energy expenditure (consumption) in 45nm 0.9V.



1 pJ (pico Joule) $= 10^{-12}$ J

tables from: M. Krouka (2021), Published in: 2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications

Motivation

analyzing the problem:

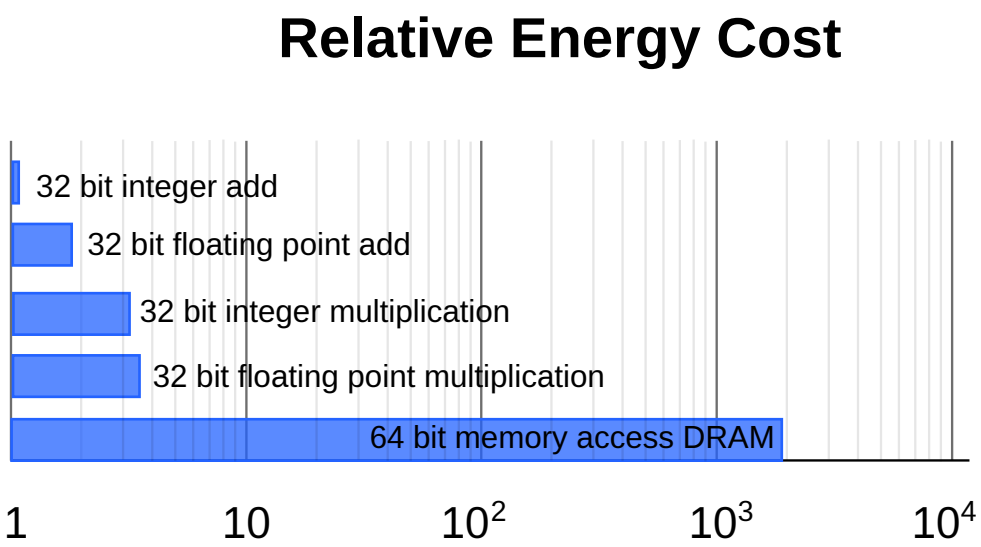
larger models $\rightarrow$ more memory references $\rightarrow$ more energy

Arithmetic operation	ADD	MUL
8-bit Integer	0.03 pJ	0.2 pJ
16-bit Floating point	0.4 pJ	1.1 pJ
32-bit Integer	0.1 pJ	3.1 pJ
32-bit Floating point	0.9 pJ	3.7 pJ

TABLE I: Approximate energy costs for different arithmetic operations in 45nm 0.9V.

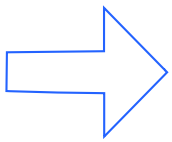
Memory size	64-bit Memory access
8KB	10 pJ
32KB	20 pJ
1 MB	100 pJ
DRAM	1.3-2.6 nJ

TABLE II: Memory access energy expenditure (consumption) in 45nm 0.9V.



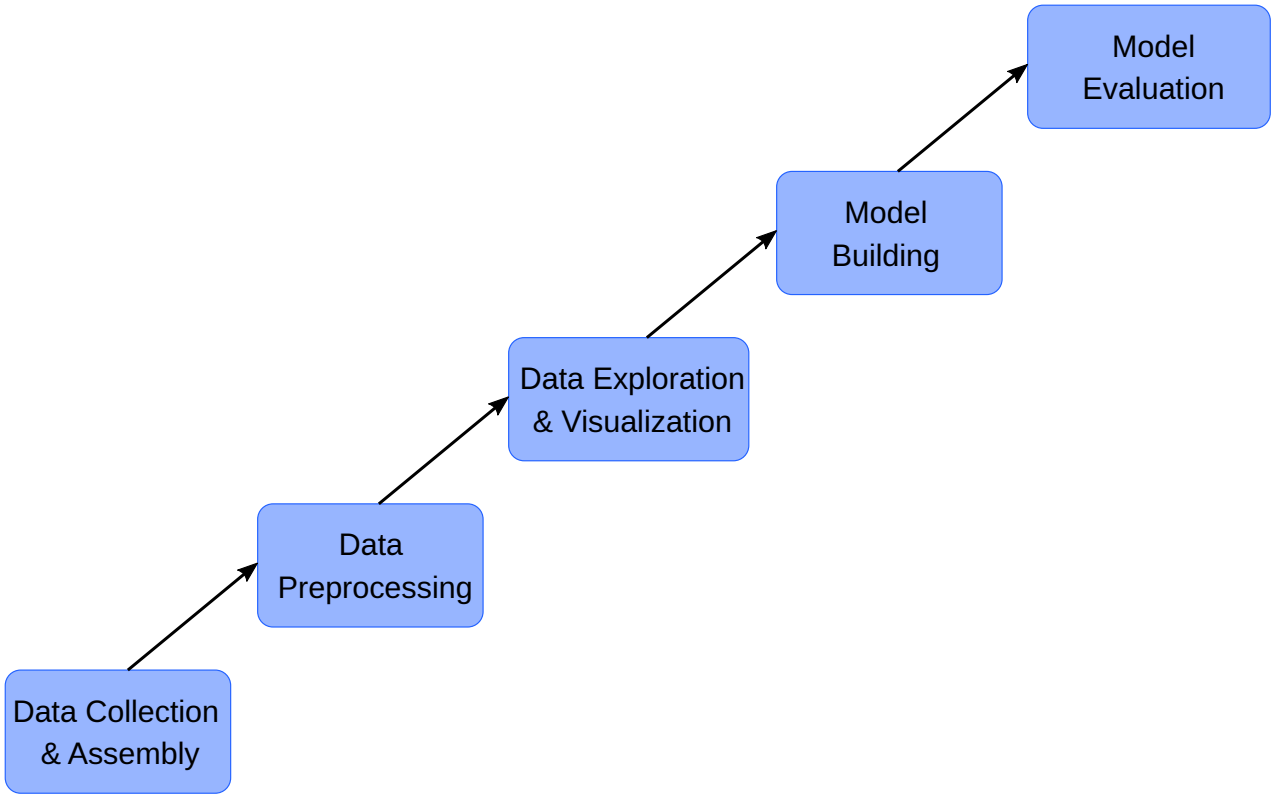
1 pJ (pico Joule) $= 10^{-12}$ J

tables from: M. Krouka (2021), Published in: 2021 IEEE 32nd Annual International Symposium on Personal, Indoor and Mobile Radio Communications

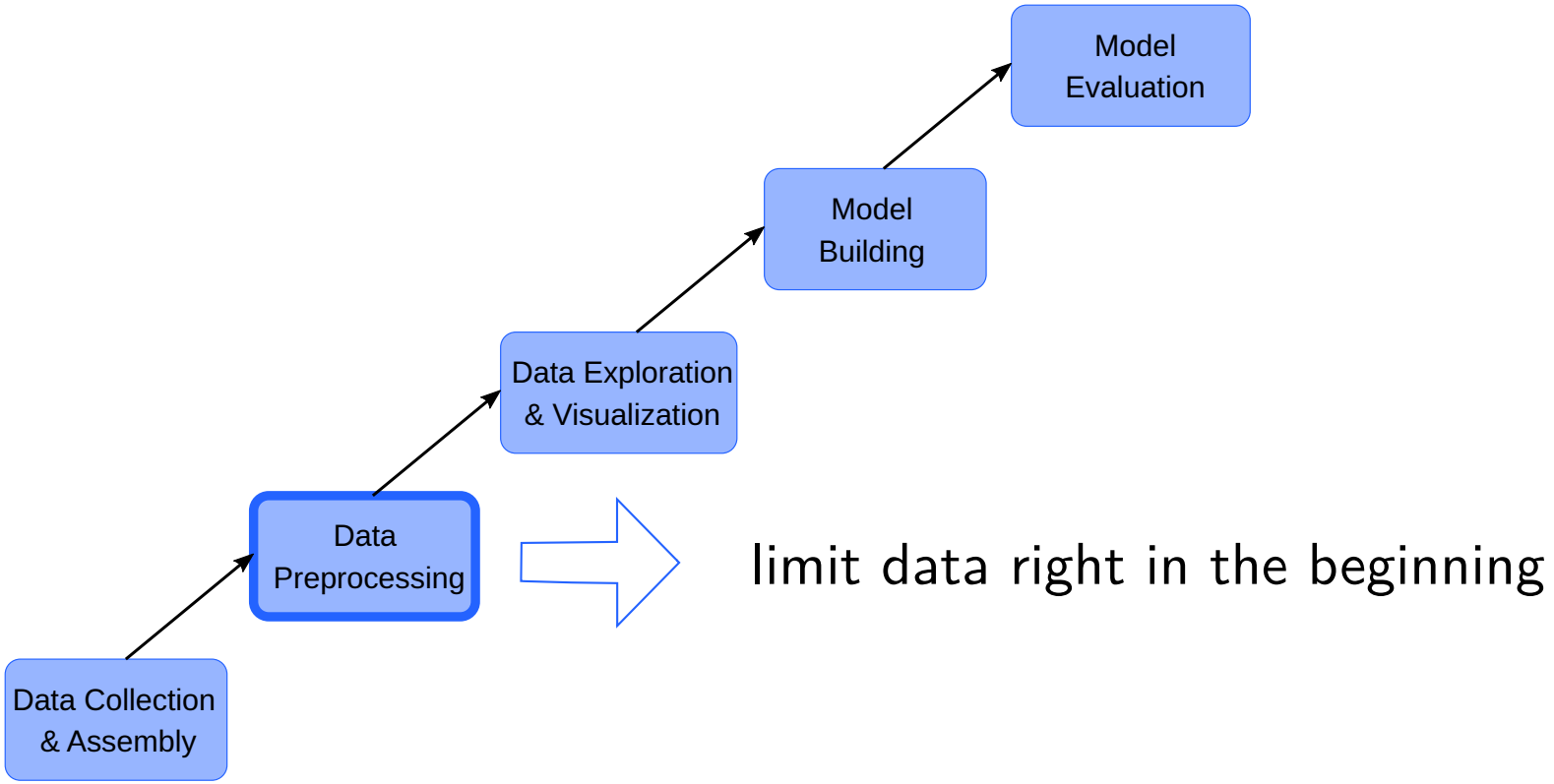


how to make Deep Learning more efficient?

Data Preprocessing



Data Preprocessing



Data Preprocessing

Data Preprocessing transforms data so that it may be easily processed by the machine learning algorithm.

Why is Data Preprocessing important?

Data Preprocessing

Data Preprocessing transforms data so that it may be easily processed by the machine learning algorithm.

Why is Data Preprocessing important?

The main agenda for a model to be accurate and precise in predictions is that the algorithm should be able to easily interpret the data's features.

Many real-world datasets are highly susceptible to be missing, inconsistent, and noisy due to their heterogeneous origin.

- Duplicate or missing values may give an incorrect view of the overall statistics of data.
- Outliers and inconsistent data points often tend to disturb the model's overall learning, leading to false predictions.

Data Preprocessing

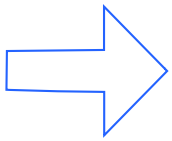
Data Preprocessing transforms data so that it may be easily processed by the machine learning algorithm.

Why is Data Preprocessing important?

The main agenda for a model to be accurate and precise in predictions is that the algorithm should be able to easily interpret the data's features.

Many real-world datasets are highly susceptible to be missing, inconsistent, and noisy due to their heterogeneous origin.

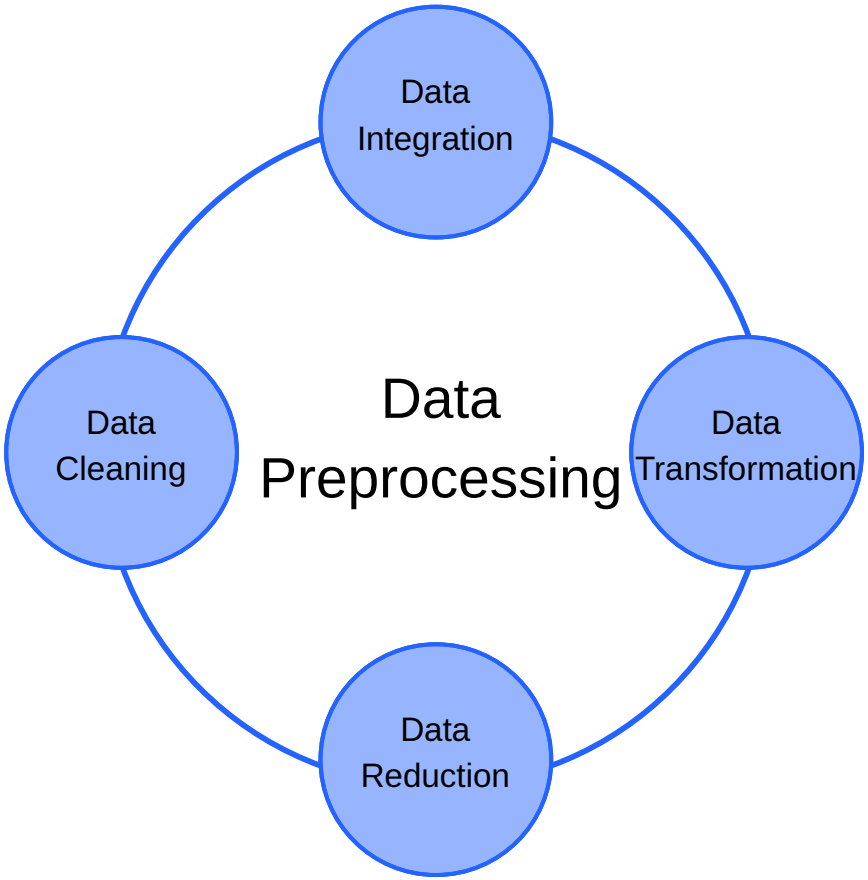
- Duplicate or missing values may give an incorrect view of the overall statistics of data.
- Outliers and inconsistent data points often tend to disturb the model's overall learning, leading to false predictions.



Quality decisions must be based on quality data.

Data Preprocessing

Data Preprocessing consists of four steps:



Data Preprocessing

Data Cleaning as typically the first step of data processing involves:

Filling missing values:

- Ignore those tuples: This method should be considered when the dataset is huge and numerous missing values are present within a tuple. Caution: Can lead to an imbalanced data set!
- Fill in the missing values (imputation): Possible due to regression. Caution: Can lead to false predictions!

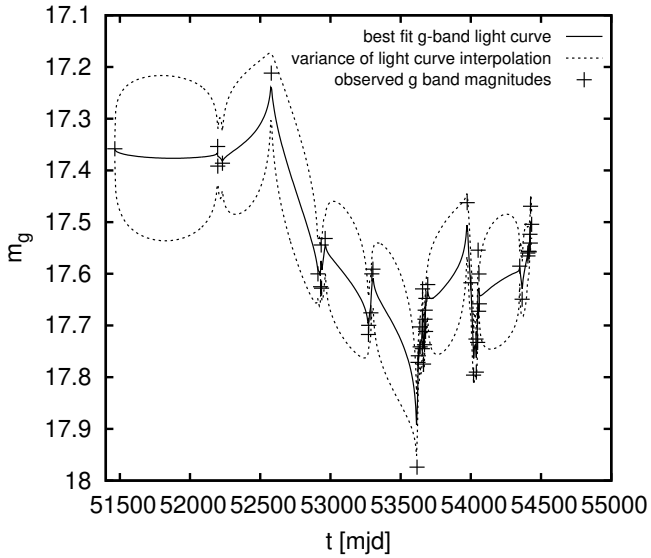
Data Preprocessing

Data Cleaning as typically the first step of data processing involves:

Filling missing values:

- Ignore those tuples: This method should be considered when the dataset is huge and numerous missing values are present within a tuple. Caution: Can lead to an imbalanced data set!
- Fill in the missing values (imputation): Possible due to regression. Caution: Can lead to false predictions!

Sometimes a better choice:
Keeping missing values as they are.
Build your model in a way so it can deal with missing data.
E.g.: Interpolation, regression doesn't happen at the data level, but internally at the feature extractor or classifier level.



Data Preprocessing

Data Cleaning as typically the first step of data processing involves:

Dealing with noisy data:
Techniques involve removing a random error or variance in a measured variable, using the following techniques:

- Measurement errors: If measurement errors are given (often for astronomical data), they can be incorporated into a fitting function.
- Binning: This technique works on data that can be sorted; all data in a bin are replaced by its mean, median or boundary values.
- Regression: This technique is generally used for prediction. It helps to smoothen data by fitting all the data points in a regression function.
- Clustering: Creation of groups/clusters from data having similar values. Values outside of the cluster can be treated as noisy data and can be removed.

Data Preprocessing

Data Cleaning as typically the first step of data processing involves:

Dealing with noisy data:
Techniques involve removing a random error or variance in a measured variable, using the following techniques:

- Measurement errors: If measurement errors are given (often for astronomical data), they can be incorporated into a fitting function.
- Binning: This technique works on data that can be sorted; all data in a bin are replaced by its mean, median or boundary values.
- Regression: This technique is generally used for prediction. It helps to smoothen data by fitting all the data points in a regression function.
- Clustering: Creation of groups/clusters from data having similar values. Values outside of the cluster can be treated as noisy data and can be removed.

Caution: This noise could be where the interesting stuff is - rare events!

Data Preprocessing

Data Cleaning as typically the first step of data processing involves:

Removing outliers:

- Statistical methods like 3σ clipping.
- Clustering techniques group together similar data points. The tuples that lie outside the cluster are outliers/inconsistent data.

Data Preprocessing

Data Cleaning as typically the first step of data processing involves:

Removing outliers:

- Statistical methods like 3σ clipping.
- Clustering techniques group together similar data points. The tuples that lie outside the cluster are outliers/inconsistent data.

Caution: Again, ouliers could be where the interesting stuff is - rare events!

Sometimes a better choice:
Keeping possible outliers as they are. Build your model in a way so it can deal with missing data. E.g.: building an outlier model that can account for complex outliers, without clipping too much data points.

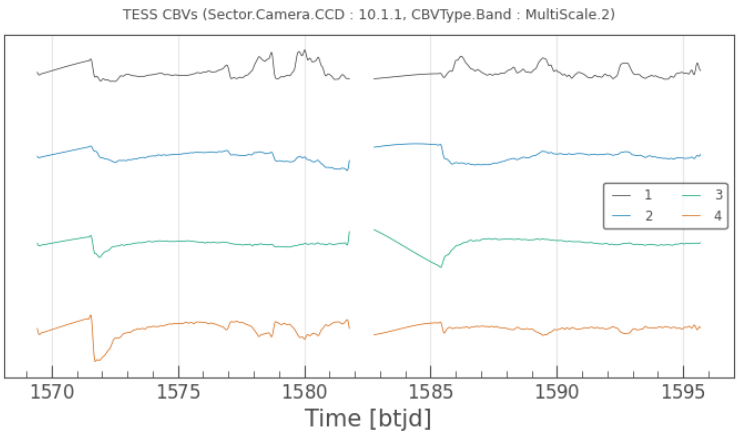
Data Preprocessing

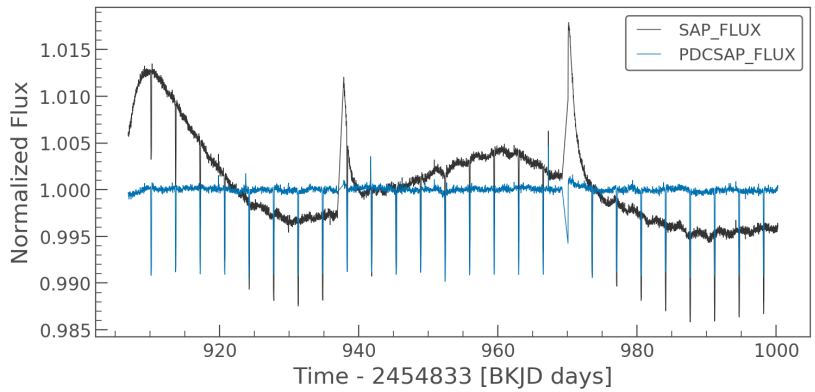
Data Cleaning as typically the first step of data processing involves:

Removing trends:

- General statistical methods like moving-average detrending.
- Specific models that make use of e.g. spacecraft telemetry or statistics of carefully selected subsamples like periodic variable stars.

Cotrending Basis Vectors (CBVs) are generated in the Kepler/K2/TESS pipeline to remove systematic trends in light curves.





Data Preprocessing

Data Integration merges data from multiple sources.

With large astronomical surveys, follow-up surveys, multi-messenger astronomy... having access to multiple sources of data is nowadays very common in astronomy.

typical **issues** we encounter during data integration include:

- Schema integration and object matching: The data can be present in different formats, and attributes that might cause difficulty in data integration.
- Detection and resolution of data value conflicts. Example: How to deal with slightly different positions for the same astronomical source?

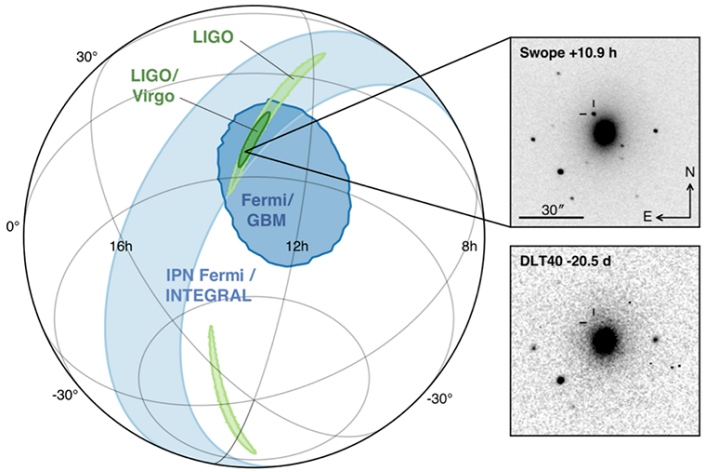


image credit: LIGO

Data Preprocessing

Data Transformation changes the value, structure, or format of data using methods like the following ones:

Generalization:
The low-level data are converted to high-level information by using concept hierarchies. Example: observations are grouped to light curves or objects.

Normalization:
Numerical attributes are scaled to fit within a specified range.
Normalization can be done in multiple ways, which are highlighted here:

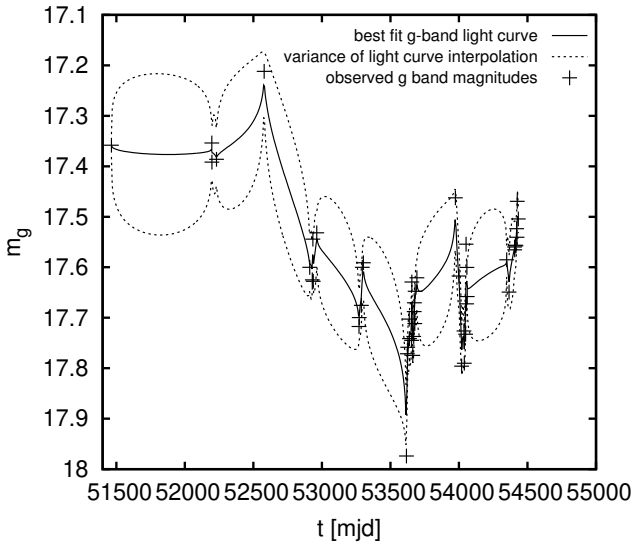
- min-max normalization
- Z-Score normalization
- decimal scaling normalization

Data Preprocessing

Data Transformation changes the value, structure, or format of data using methods like the following ones:

Strategies especially common in time-domain astronomy are:

Feature extraction:
The algorithms used can be rather simple, as well as complex (machine-learning).



This light-curve fit leads to feature extraction: the features are parameters of a structure function model.

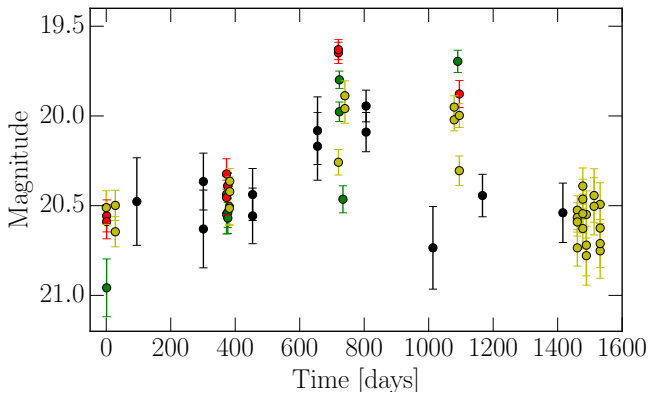
Data Preprocessing

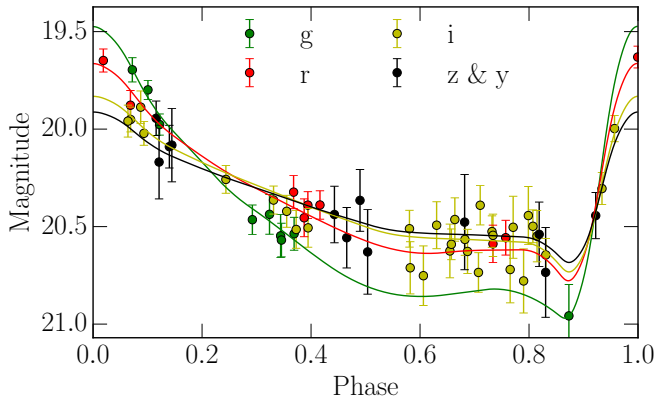
Data Transformation changes the value, structure, or format of data using methods like the following ones:

Strategies especially common in time-domain astronomy are:

Phase-folding: Phase-folding requires knowledge about the periodicity of a signal (such as a light curve). Period detection can involve algorithms such as Lomb-Scargle as well as complex machine-learning algorithms capable of dealing with data with noise and less-than-optimal cadence.
Caution: Previously measured periodicity information might have changed!

example: RR Lyrae period fitting with light curve templates from SDSS





Data Preprocessing

Data Reduction obtains a reduced and better manageable data set that produces the same quality of analytical results. Common Data Reduction strategies include:

Dimensionality reduction:
Dimensionality reduction techniques are used to perform (abstract) feature extraction by using algorithms such as Principal Component Analysis.

Numerosity reduction:
The data can be represented as a model or equation like a regression model. This saves the burden of storing huge datasets instead of a model.

Attribute subset selection:
Selecting only specific features, attributes (table columns) is very important to reduce model sizes. Only attributes that add more value towards model training should be considered, and the rest all can be discarded for the problem at hand.

Data Quality Assessment

A high-quality data set should fulfill the following conditions:

- data validity
- accuracy and reliability in terms of information
- consistency in all features
- completeness with no missing attribute values
- no redundancy

Data Quality Assessment

A high-quality data set should fulfill the following conditions:

- data validity
- accuracy and reliability in terms of information
- consistency in all features
- completeness with no missing attribute values
- no redundancy

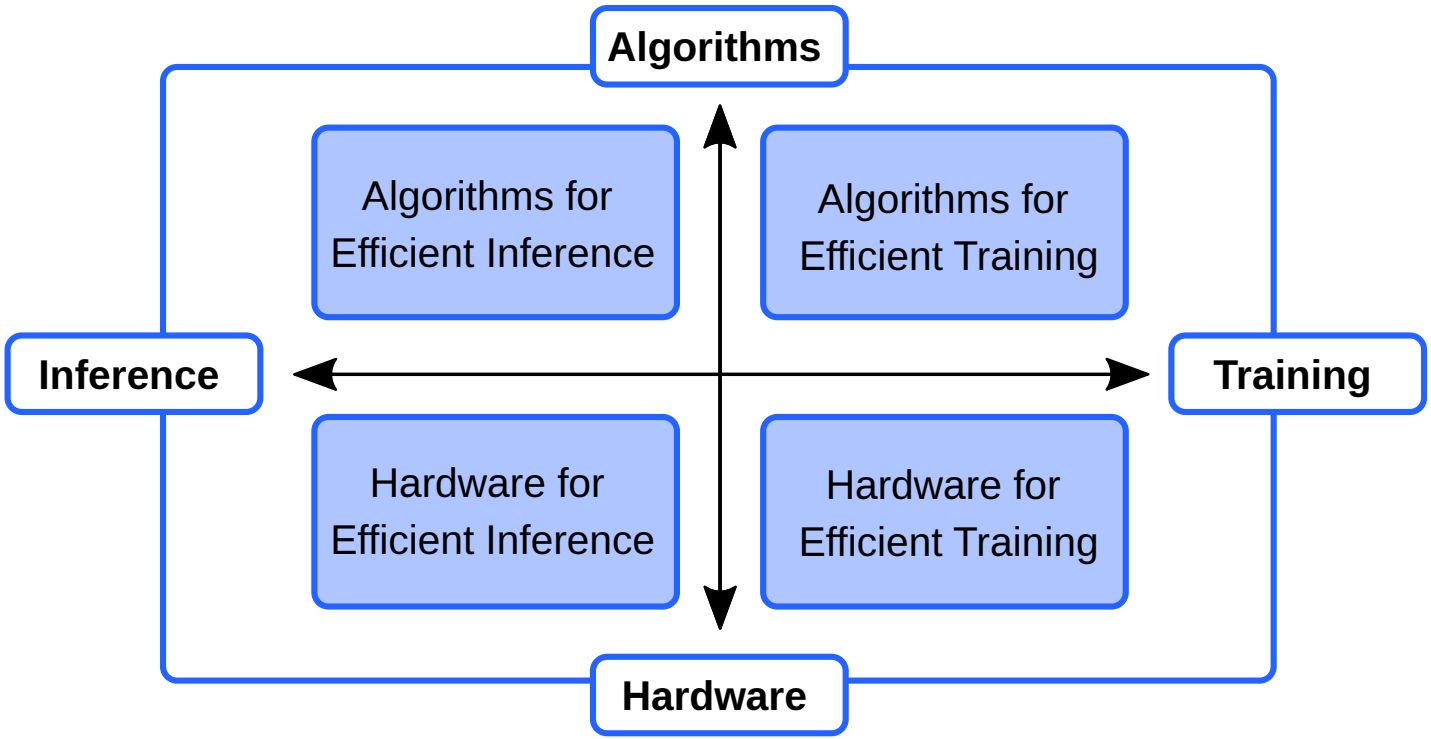
A common redundancy problem in astronomical data sets: multiple object IDs refer to the same astronomical object/ a light curve is split into multiple object IDs due to problems in light curve extraction.

Data Preprocessing

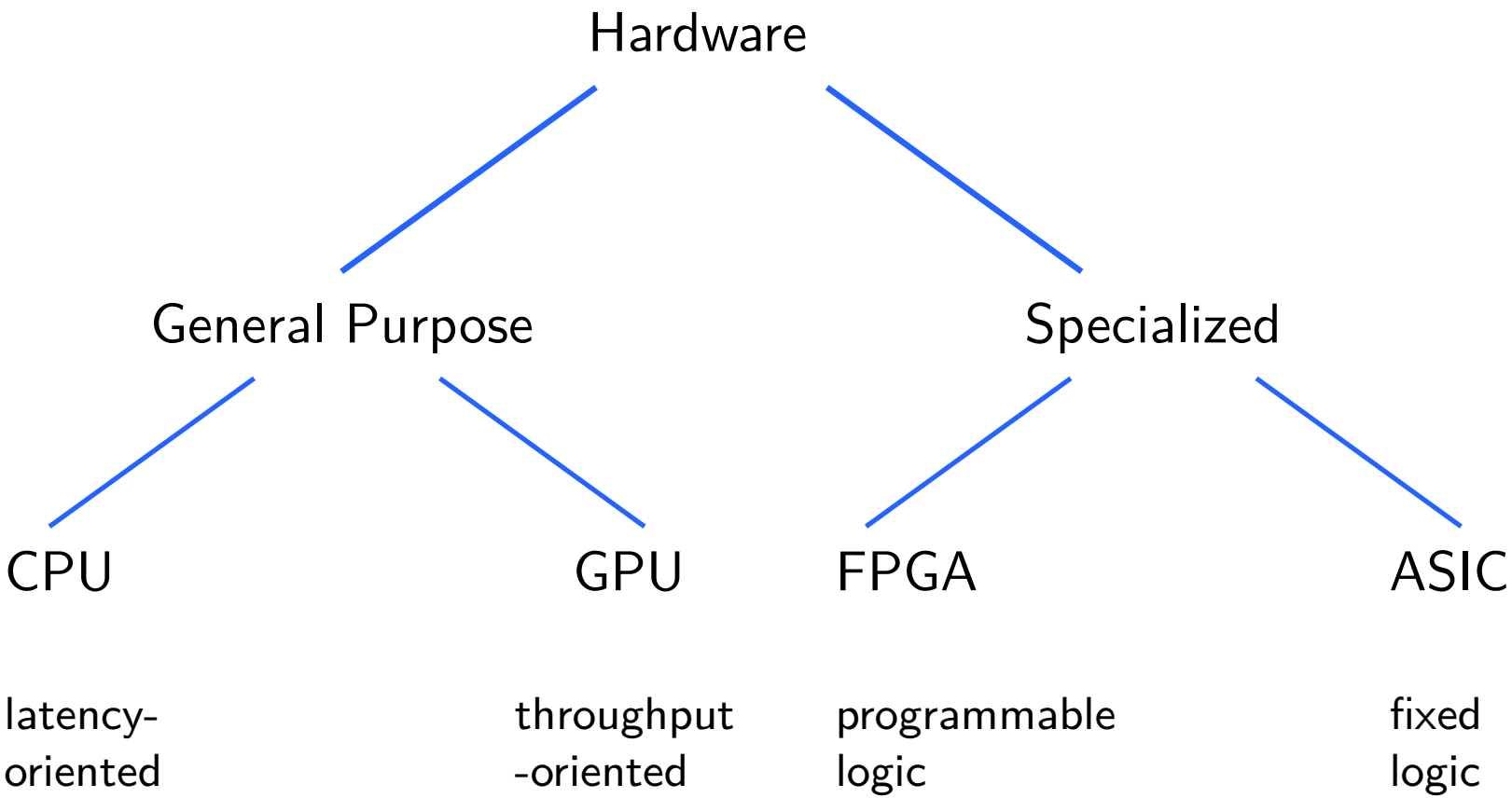
Best practices for Data Preprocessing:

- The first step in Data Preprocessing is to understand your data. Just looking at your dataset can give you an intuition of what things you need to focus on.
- Use statistical methods or pre-built libraries that help you visualize the dataset and give a clear image of how your data looks in terms of class distribution.
- Summarize your data in terms of the number of duplicates, missing values, and outliers present in the data.
- Drop the fields you think have no use for the modeling or are closely related to other attributes. Dimensionality reduction is one of the very important aspects of Data Preprocessing.
- Do some feature engineering and figure out which attributes contribute most towards model training.

Improving Efficiency



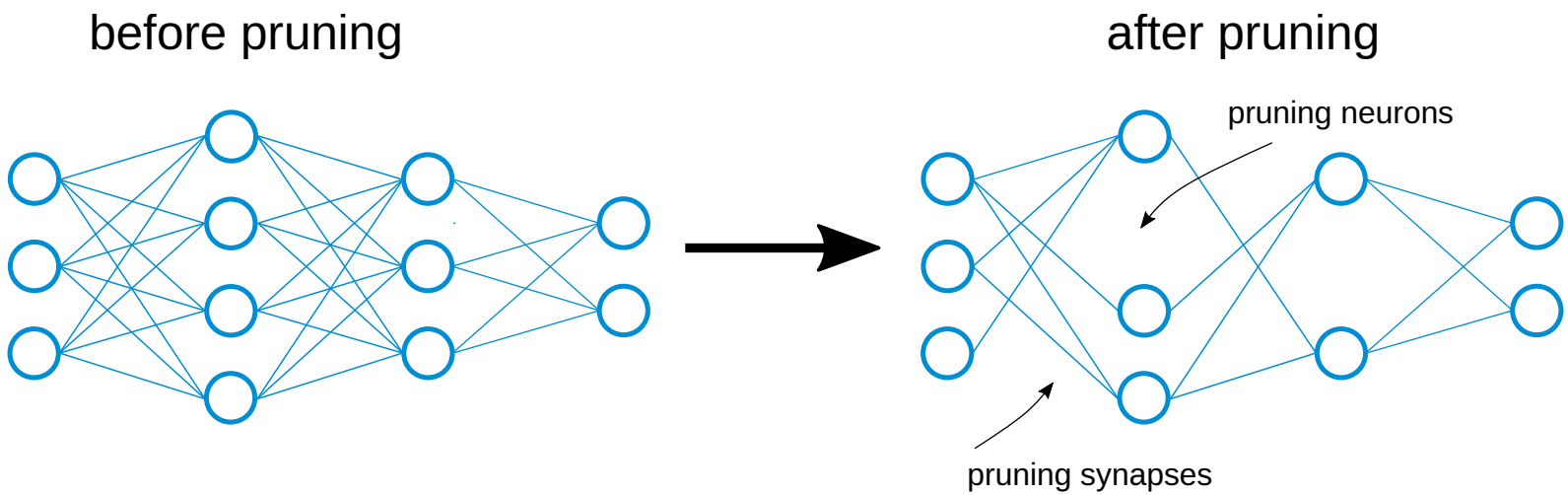
Hardware



Algorithms

we care here about **algorithms** to make machine learning more efficient

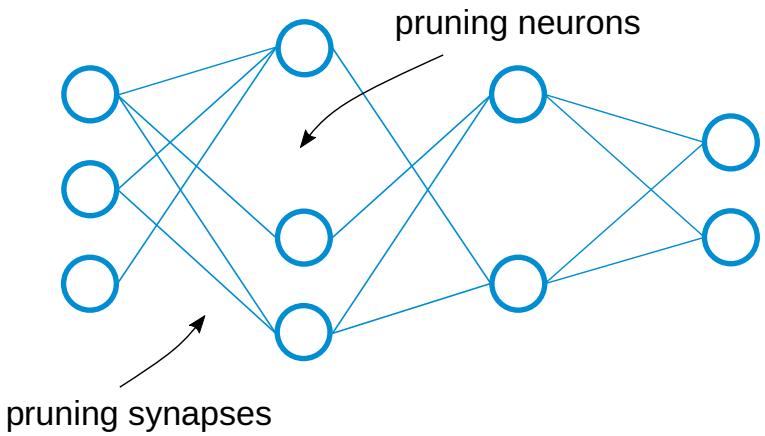
Pruning Neural Networks

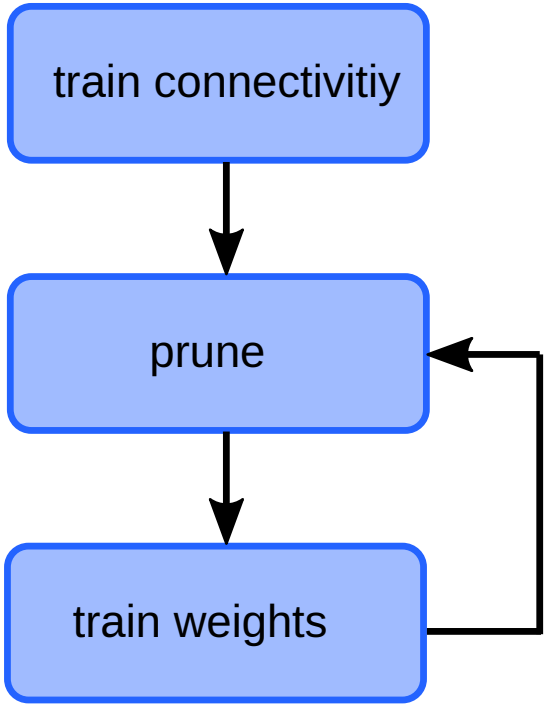


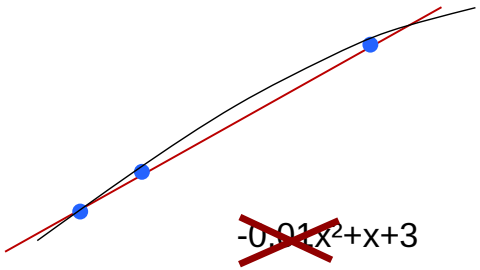
credit: Lecun et al. (1989), Han et al. (2015)

Pruning Neural Networks

What happens to the weights?





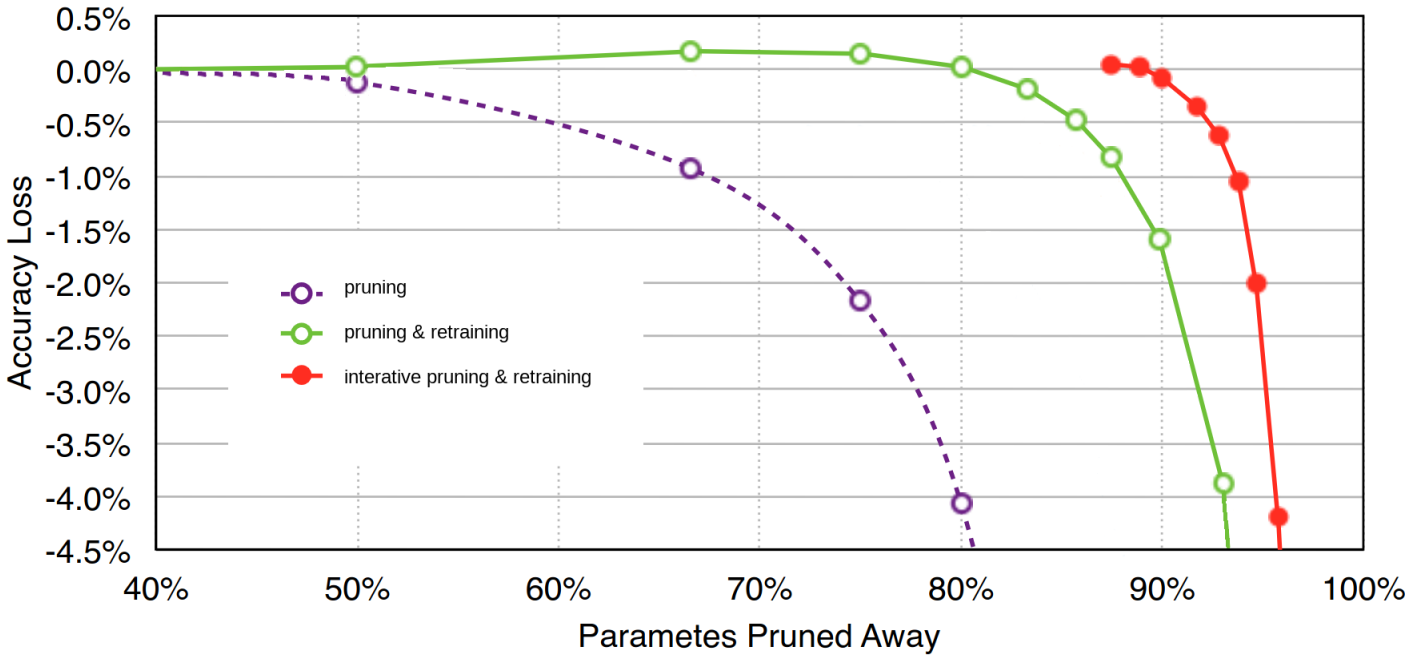


reduces number of connections e.g. by a factor of 10

credit: Lecun et al. (1989), Han et al. (2015)

Pruning Neural Networks

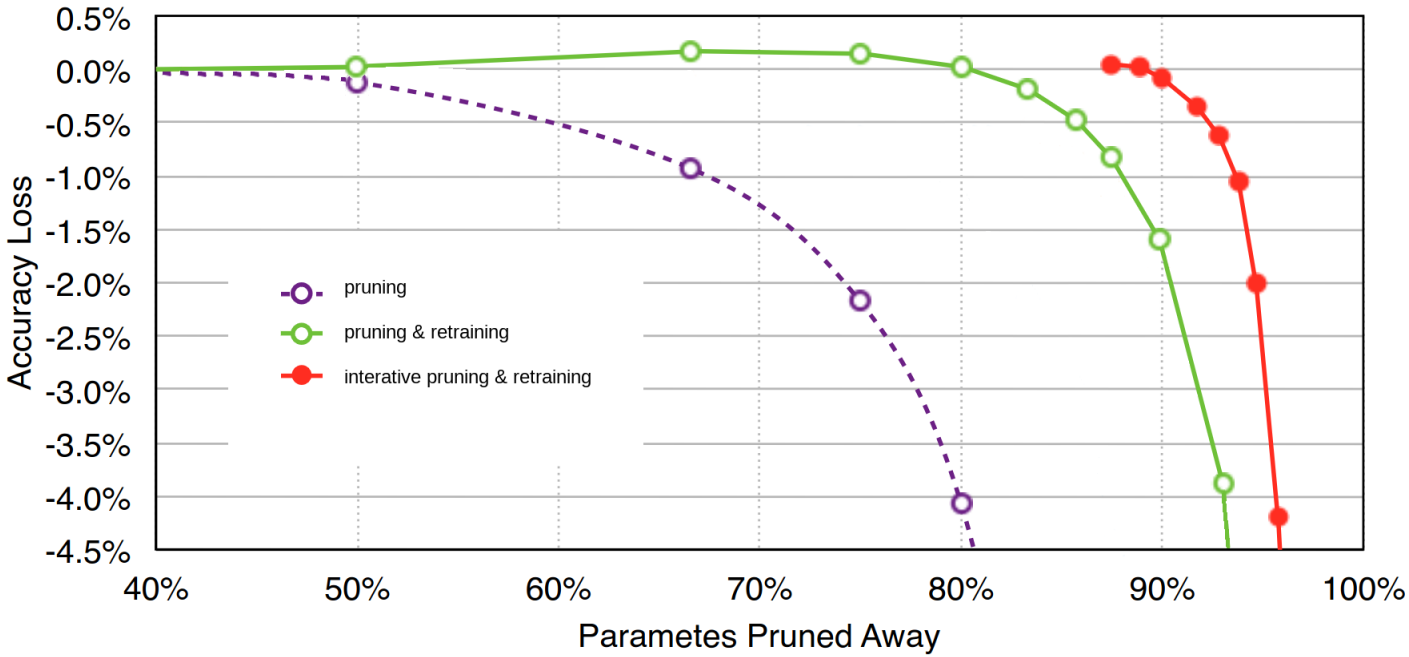
What happens to the accuracy?



credit: Han et al. (2015)

Pruning Neural Networks

What happens to the accuracy?



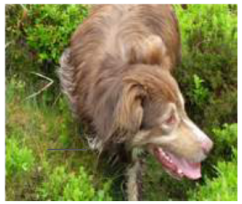
credit: Han et al. (2015)

Retraining after pruning recovers accuracy - **iterative** pruning & retraining performs better

Pruning Neural Networks

example results: Pruning NeuralTalk

- **Original**: a basketball player in a white uniform is playing with a ball
- **Pruned 90%**: a basketball player in a white uniform is playing with a basketball



- **Original** : a brown dog is running through a grassy field
- **Pruned 90%**: a brown dog is running through a grassy area

- **Original** : a man is riding a surfboard on a wave
- **Pruned 90%**: a man in a wetsuit is riding a wave on a beach

- **Original** : a soccer player in red is running in the field
- **Pruned 95%**: a man in a red shirt and black and white black shirt is running through a field

credit: Han et al. (2015), NIPS presentation

Pruning Neural Networks

Pruning also happens in the human (and animal) brain:

newborn human: 50 trillion synapses

human toddler: 1000 trillion synapses

human adolescent: 50 trillion synapses

(source: C.A. Walsh, P. Huttenlocher, Nature, 502(7470):172, (2013)

Pruning Neural Networks

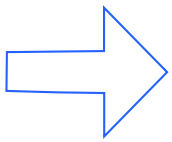
Pruning also happens in the human (and animal) brain:

newborn human: 50 trillion synapses

human toddler: 1000 trillion synapses

human adolescent: 50 trillion synapses

(source: C.A. Walsh, P. Huttenlocher, Nature, 502(7470):172, (2013)

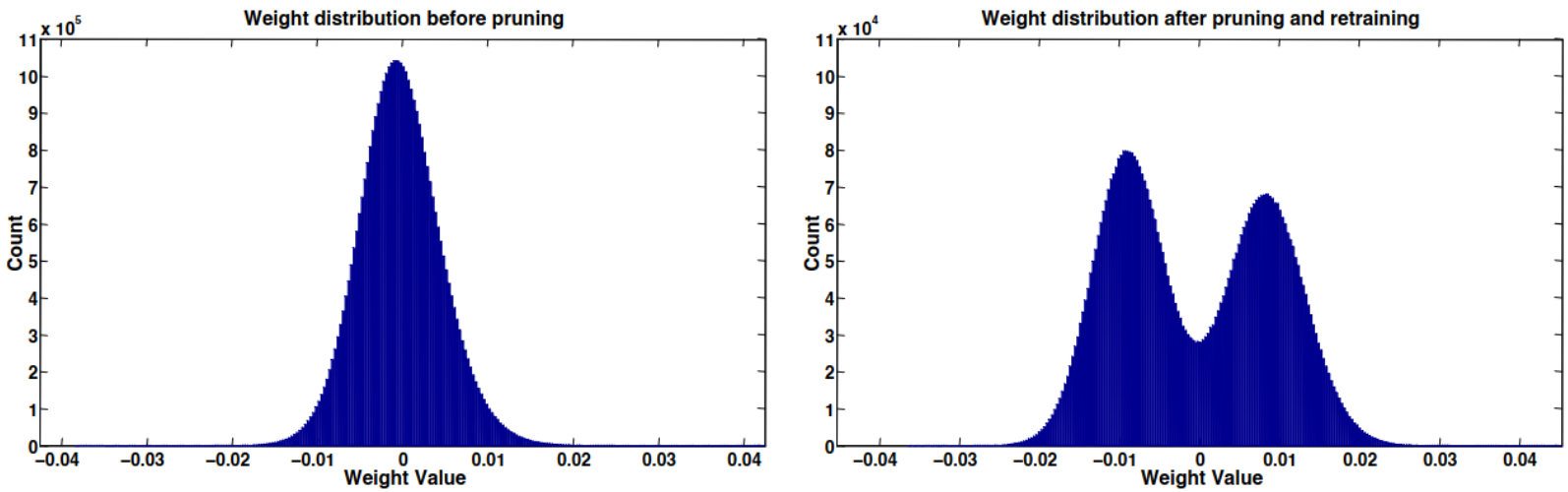


this allows for a lot of **extra flexilbility** until adapted to the environment

Pruning Neural Networks

What happens to the weights?

Their distribution changes:



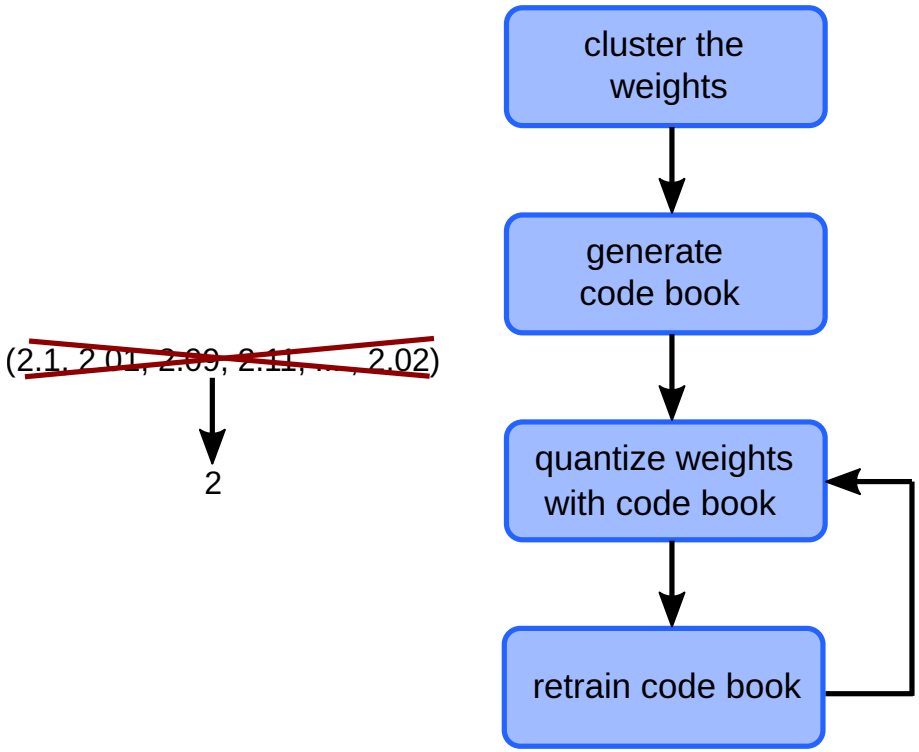
Weight distribution before and after parameter pruning in the Conv5 Layer of AlexNet. The right figure has 10× smaller scale than the left one. Representative also for other network layers.

credit: Han et al. (2015)

Weight Sharing with Trained Quantization

Do we really need very precise weights?

We can rather group similar weights and use a code book to save memory. That's called **weight quantization**.



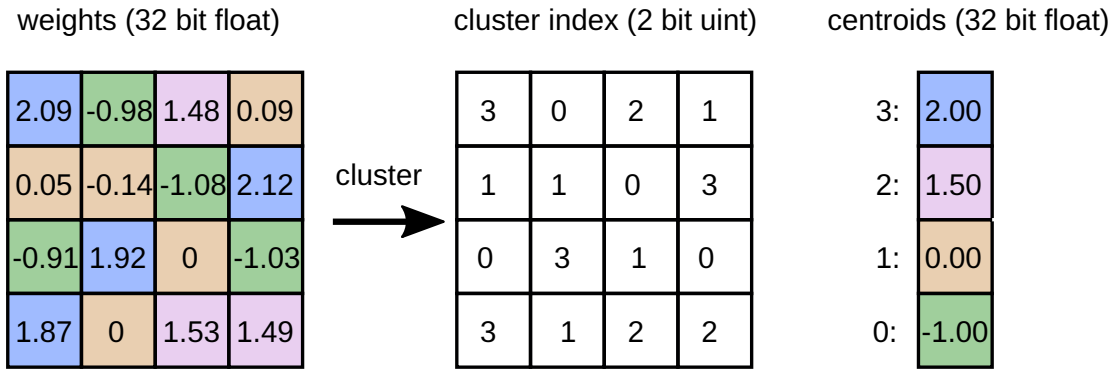
This reduces the memory footprint by a factor of ~ 8

Weight Sharing with Trained Quantization

Weight sharing by scalar quantization and centroids fine-tuning.

credit: Han et al. ICLR'16

k-means clustering is used to identify the shared weights, so that all the weights that fall into the same cluster will have the same weight.

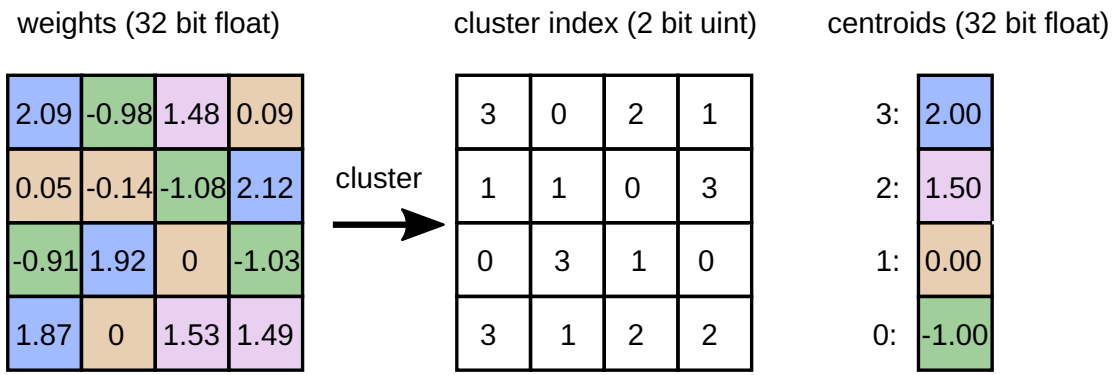


Weight Sharing with Trained Quantization

Weight sharing by scalar quantization and centroids fine-tuning.

credit: Han et al. ICLR'16

k-means clustering is used to identify the shared weights, so that all the weights that fall into the same cluster will have the same weight.



gradient

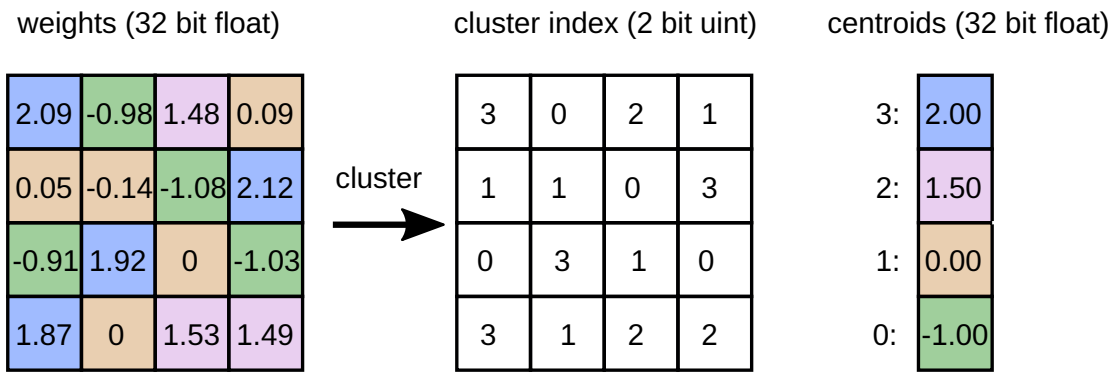
-0.03	-0.01	0.03	0.02
-0.01	0.01	-0.02	0.12
-0.01	0.02	0.04	0.01
-0.07	-0.02	0.01	-0.02

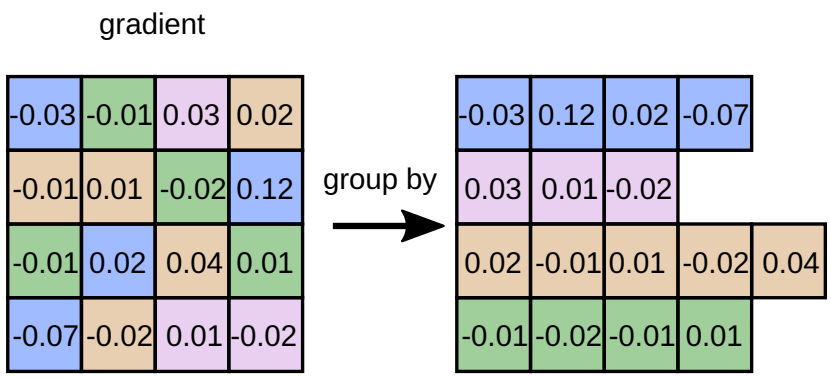
Weight Sharing with Trained Quantization

Weight sharing by scalar quantization and centroids fine-tuning.

credit: Han et al. ICLR'16

k-means clustering is used to identify the shared weights, so that all the weights that fall into the same cluster will have the same weight.



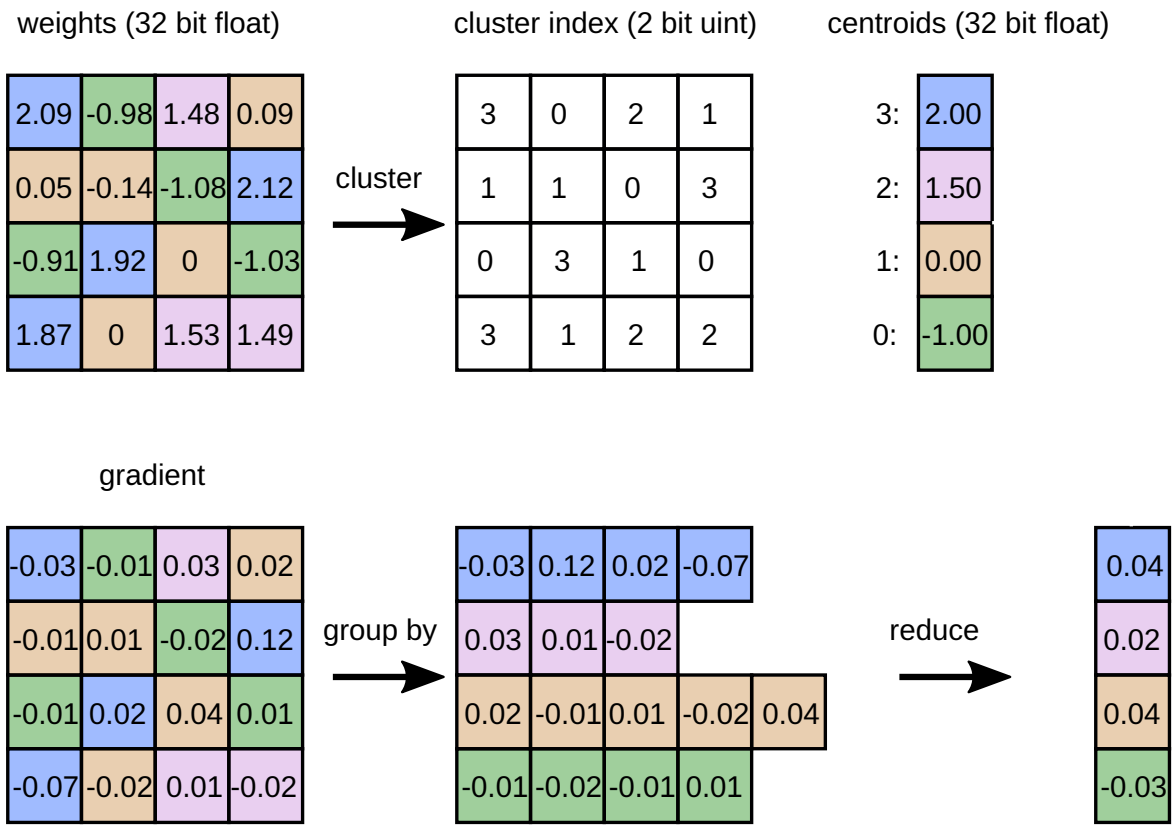


Weight Sharing with Trained Quantization

Weight sharing by scalar quantization and centroids fine-tuning.

credit: Han et al. ICLR'16

k-means clustering is used to identify the shared weights, so that all the weights that fall into the same cluster will have the same weight.

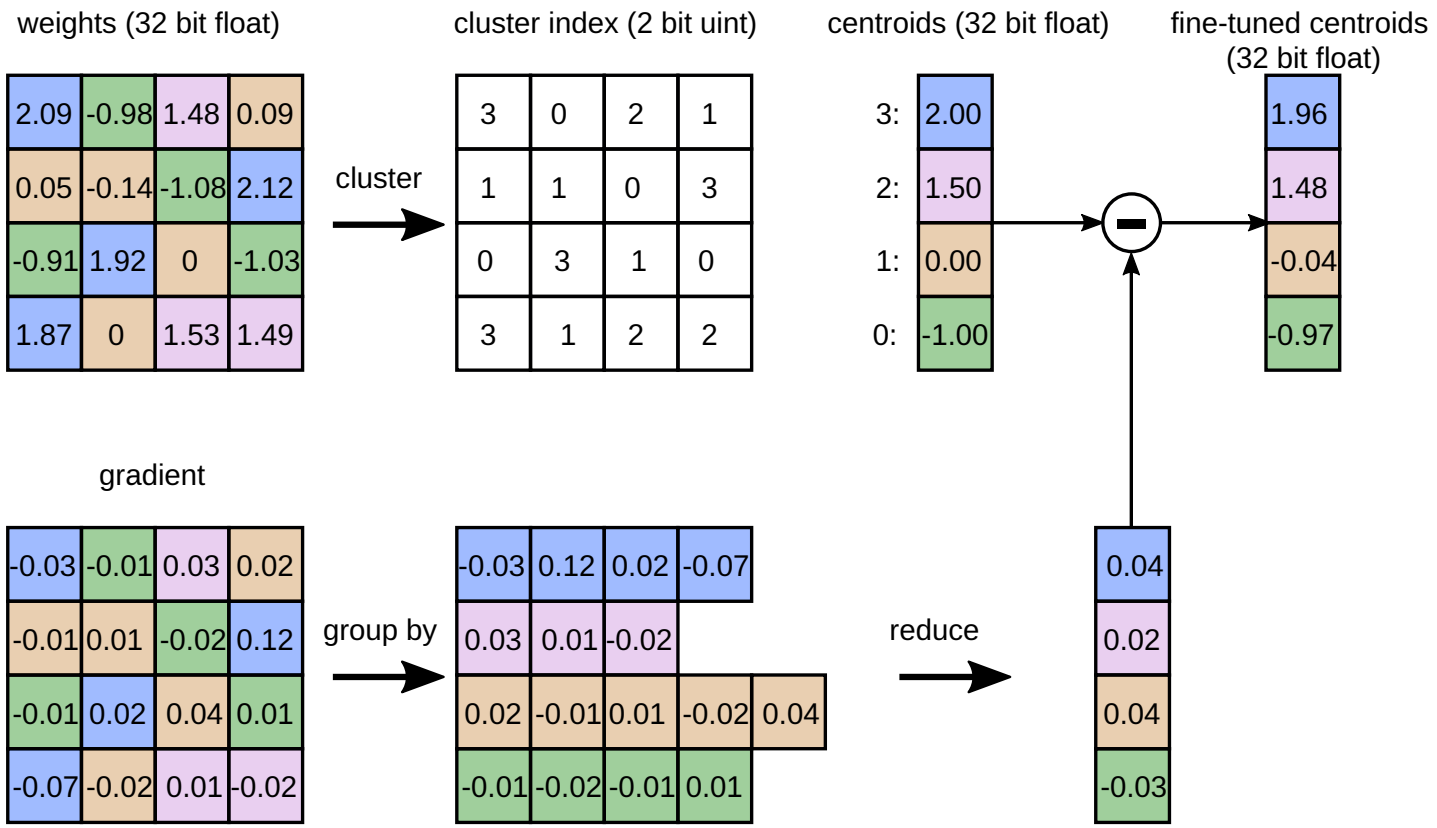


Weight Sharing with Trained Quantization

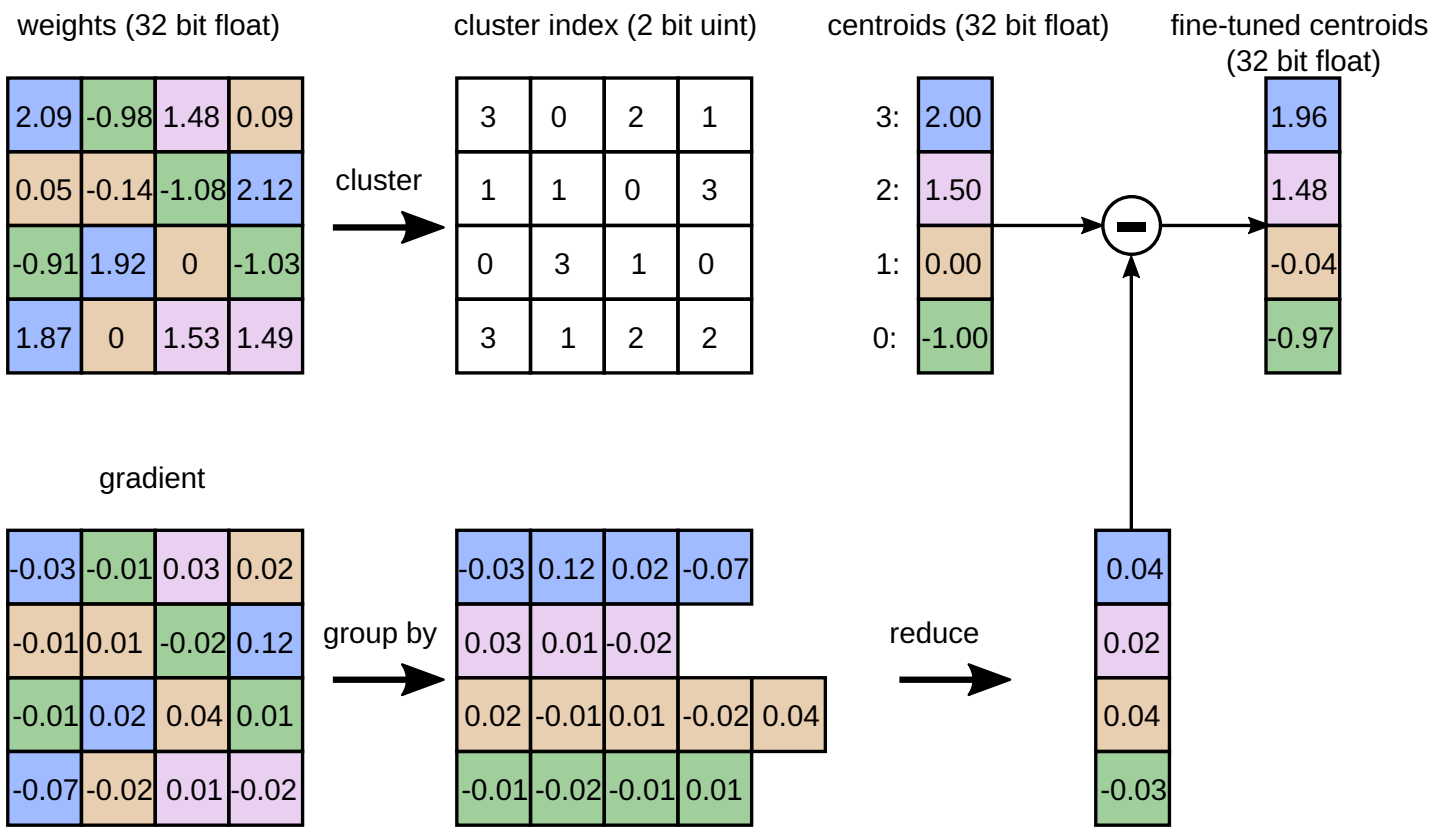
Weight sharing by scalar quantization and centroids fine-tuning.

credit: Han et al. ICLR'16

k-means clustering is used to identify the shared weights, so that all the weights that fall into the same cluster will have the same weight.



Weight Sharing with Trained Quantization



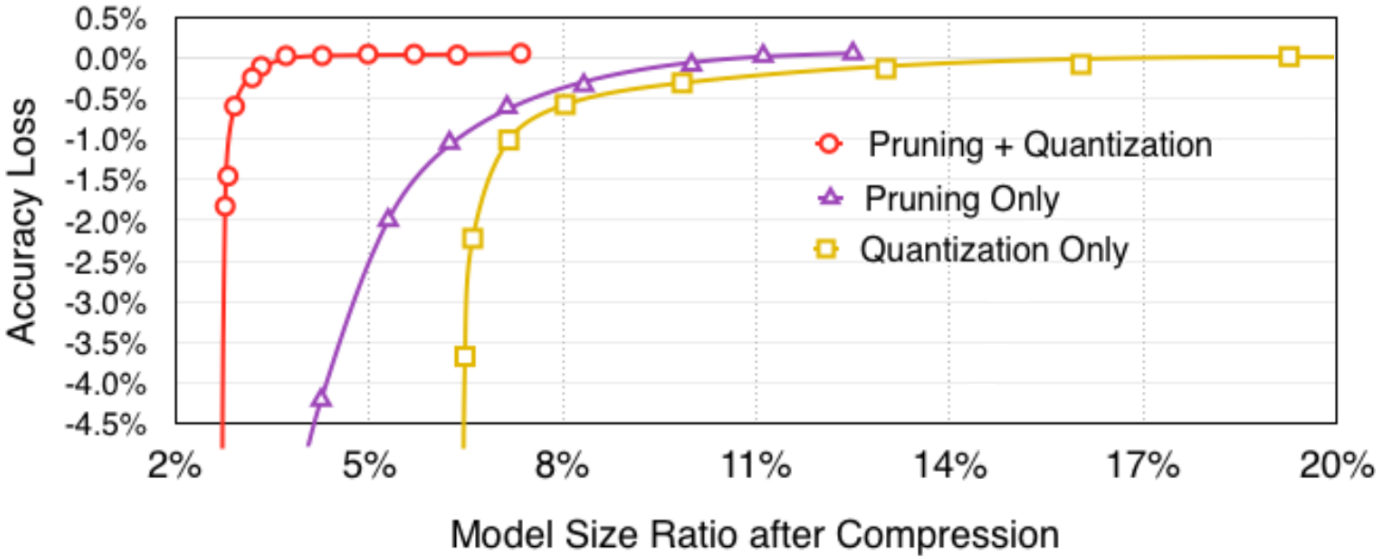
What is the compression rate?

Here we have the weights of a single layer neural network with four input units and four output units. There are $4 \times 4 = 16$ weights originally, where each has 32 bits. Now we have just 4 shared 32-bit weights. With 16 2-bit indices, we get a compression rate of $16 \times 32/(4 \times 32 + 2 \times 16) = 3.2$.

Weight Sharing with Trained Quantization

Pruning and Trained Quantization

These two algorithms work together:



Accuracy v.s. compression rate under different compression methods. Pruning and quantization works best when combined. Carried out with AlexNet on ImageNet.
credit: Han et al. ICLR'16

Weight Sharing with Trained Quantization

Pruning and Trained Quantization

We now have weights and weight indices - we can further encode them.

Huffman encoding is used to encode both the weights and indices.

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. It uses variable-length codewords to encode source symbols. The table is derived from the occurrence probability for each symbol. More common symbols are represented with fewer bits.

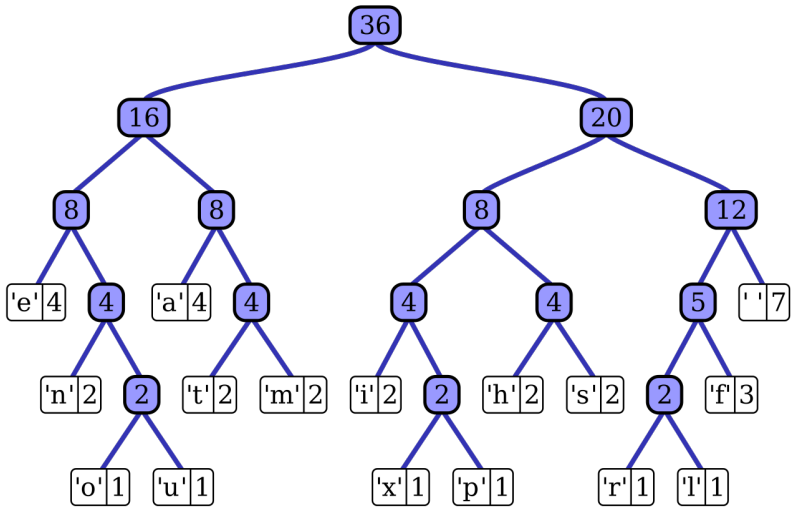
Weight Sharing with Trained Quantization

Pruning and Trained Quantization

We now have weights and weight indices - we can further encode them.

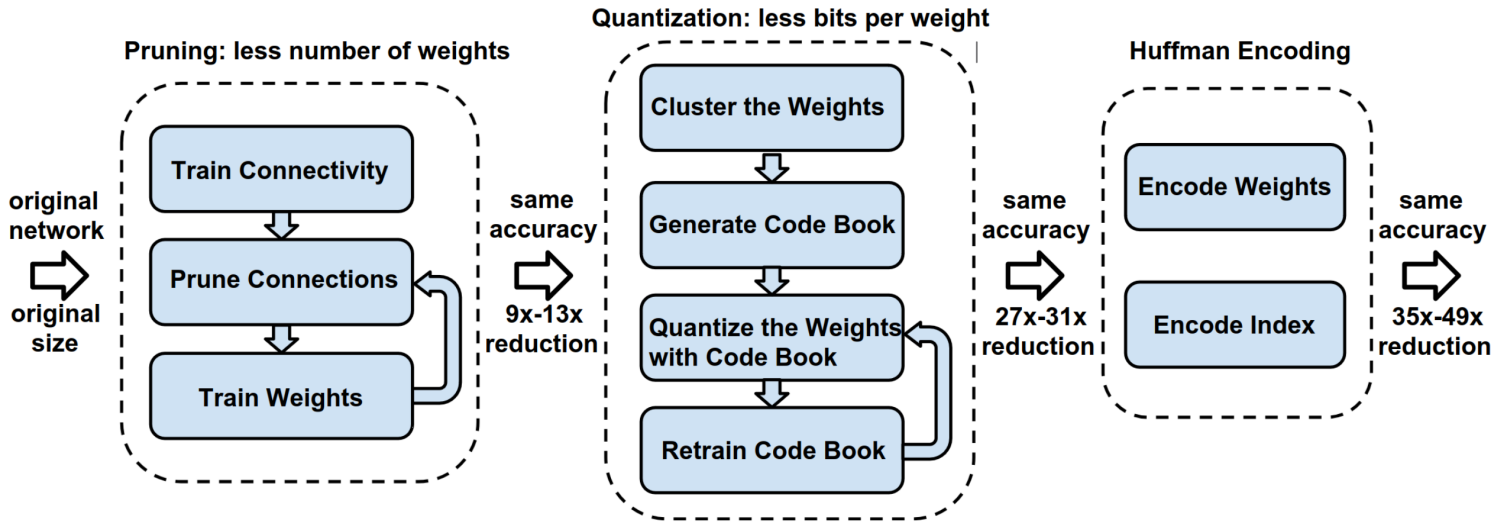
Huffman encoding is used to encode both the weights and indices.

example:
Huffman tree generated from the text "this is an example of a huffman tree". Encoding the sentence requires 135 bits, as opposed to 288 bits if 36 characters of 8 bits were used.



Weight Sharing with Trained Quantization

A Compression Pipeline: Deep Compression



The compression pipeline: pruning, quantization and Huffman coding. Pruning reduces the number of weights by a factor of 10, while quantization further improves the compression rate: between a factor of 27 and 31. Huffman coding gives a compression factor of 35 and 49. The compression rate already includes the meta-data for sparse representation. The compression scheme doesn't lead to any accuracy loss.

credit: Han et al. ICLR'16

Weight Sharing with Trained Quantization

Results: Compression Ratio

The compression pipeline can reduce the parameter storage by a factor of 35 to 49 with no loss of accuracy.

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	**27 KB**	**40**×
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	**44 KB**	**39**×
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	**6.9 MB**	**35**×
VGG-16 Ref	31.50%	11.32%	552 MB	
VGG-16 Compressed	31.17%	10.91%	**11.3 MB**	**49**×

credit: Han et al. ICLR'16

SqueezeNet

For a given accuracy level, it is typically possible to identify multiple CNN architectures that achieve that accuracy level.

Given equivalent accuracy, smaller CNN architectures offer at least three advantages:

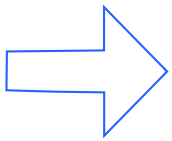
- Smaller CNNs require less communication across servers during distributed training.
- Smaller CNNs require less bandwidth to export a new model from the cloud to an autonomous car.
- Smaller CNNs are more feasible to deploy on hardware with limited memory.

SqueezeNet

For a given accuracy level, it is typically possible to identify multiple CNN architectures that achieve that accuracy level.

Given equivalent accuracy, smaller CNN architectures offer at least three advantages:

- Smaller CNNs require less communication across servers during distributed training.
- Smaller CNNs require less bandwidth to export a new model from the cloud to an autonomous car.
- Smaller CNNs are more feasible to deploy on hardware with limited memory.



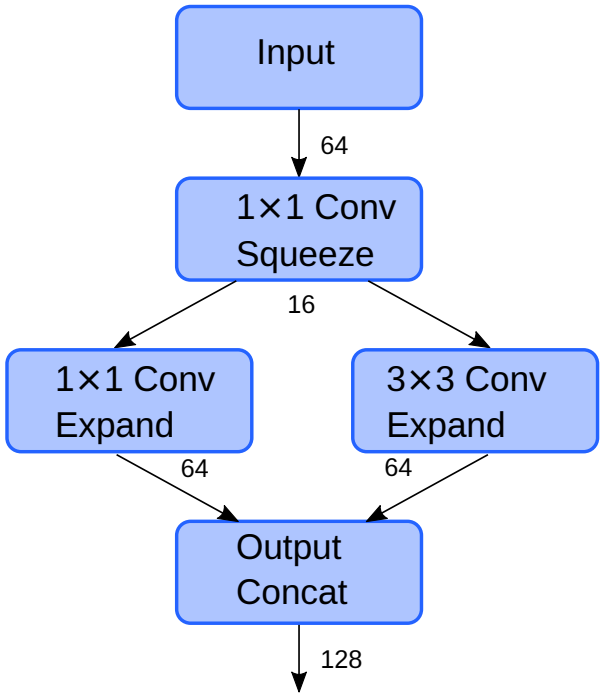
find a smaller neural network architecture

SqueezeNet

Iandola et al. (2016), *SqueezeNet: AlexNet-Level Accuracy with 50× fewer parameters and <0.5 MB Model Size**

* this is 510× smaller than AlexNet

The diagram shows one module (the *Fire module*) of that architecture which will be stacked.



SqueezeNet

What happens if we apply compression techniques to SqueezeNet?

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%

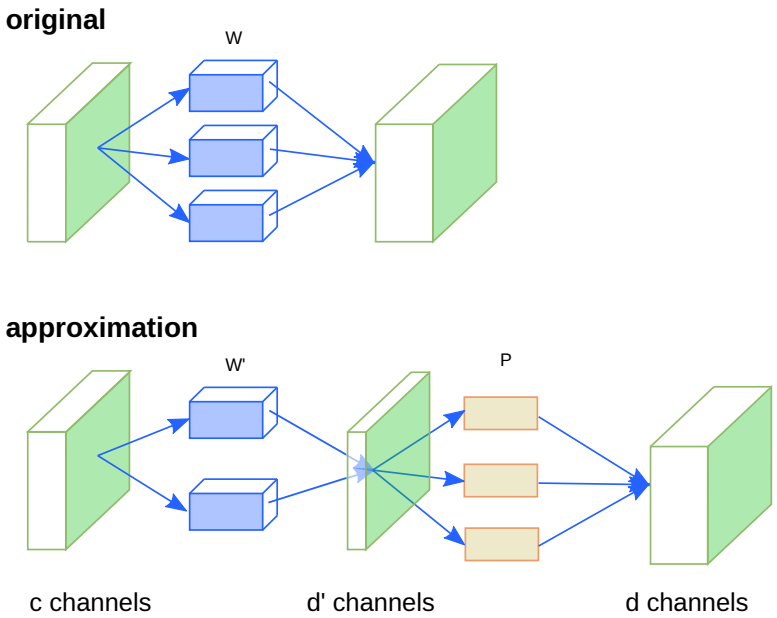
credit: Iandola et al. (2016)

Low Rank Approximation for Convolutional Layer

idea: Layer responses occupy a low-rank subspace

Decompose a convolutional layer with d filters each of size $k \times k \times c$ to:

- a layer with d' filters $(k \times k \times c)$
- a layer with d filters $1 \times 1 \times d'$



This is a Low-Rank Approximation by matrix/tensor decomposition.
Compression and acceleration: less storage, and less FLOP in computation.

Low Rank Approximation for Fully-Connected Layer

idea: Layer responses occupy a low-rank subspace

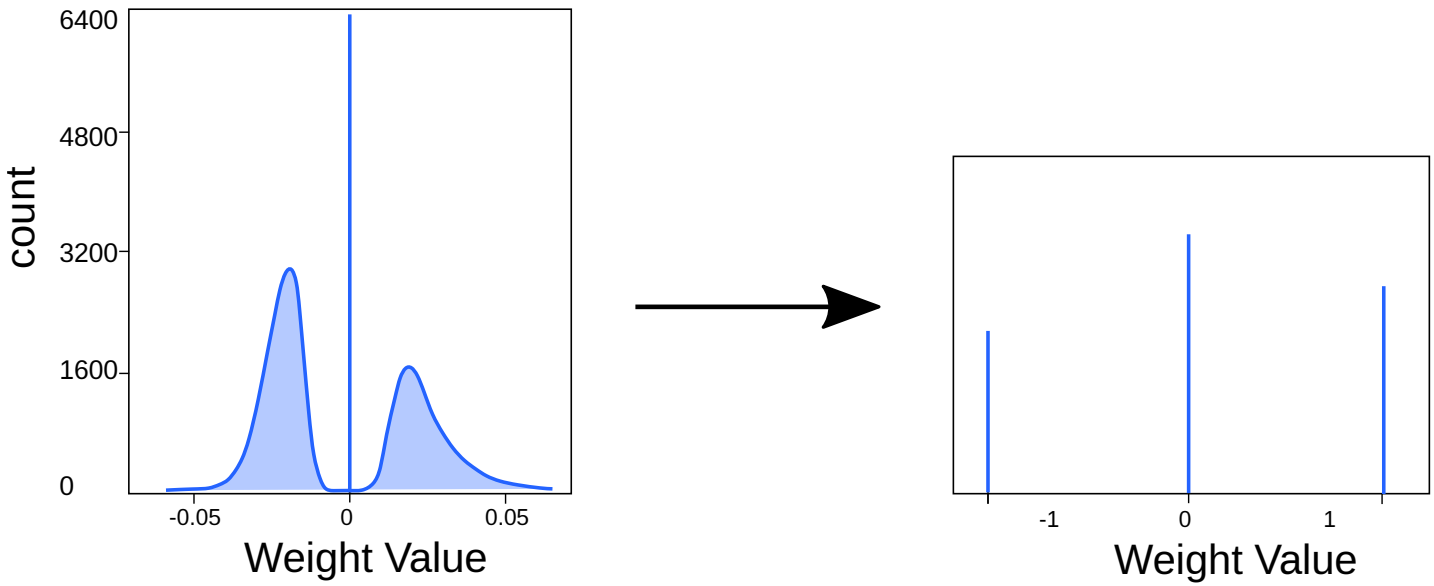
Build a mapping from indices of matrix $\mathbf{W} = [W(x, y)]$ to vectors i and j: $x \leftrightarrow i = (i_1, .., i_d)$ and $y \leftrightarrow j = (j_1, ..., j_d)$.

matrix $\mathbf{W}$:

$$\begin{aligned}\mathbf{W}(i_1, ..., i_d; j_1, ..., j_d) &= \mathbf{W}(x(i), y(j)) \\ &= \underbrace{G_1[i_1, j_1]}_{1\times r}\underbrace{G_2[i_2, j_2]}_{r\times r} ... \underbrace{G_d[i_d, j_d]}_{r\times 1}\end{aligned}$$

Binary/ Ternary Net

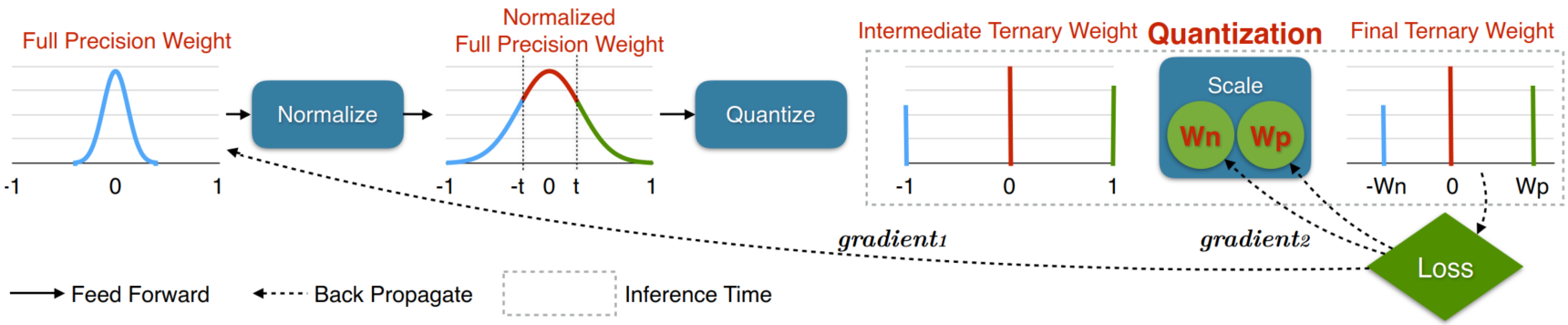
Motivation:



Ternary neural networks (TNNs) can accelerate neural networks by reducing the full-precision weights in network to ternary ones, e.g., -1,0,1.

Binary/ Ternary Net

Trained Ternary Quantization



credit: Zhu et al. ICLR'17

Binary/ Ternary Net

Weight Evolution during Training

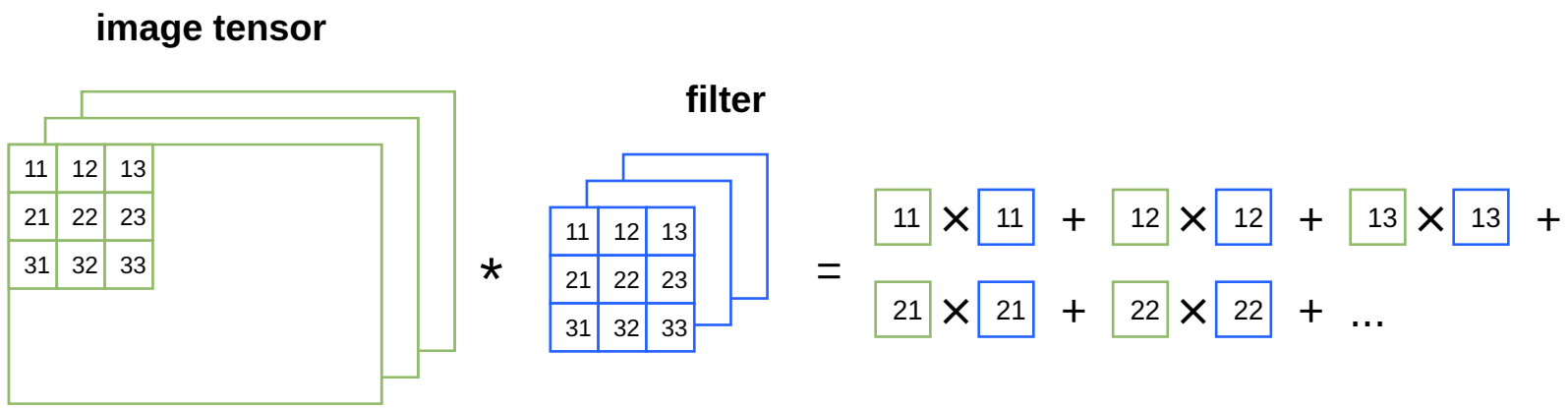
Ternary weights value (above) and distribution (below) with iterations for different layers of ResNet-20 on CIFAR-10.

credit: Zhu et al. ICLR'17

Winograd Convolution

CNNs require large amounts of computing resources for both training and inference: primarily because the convolution layers are computationally intensive.

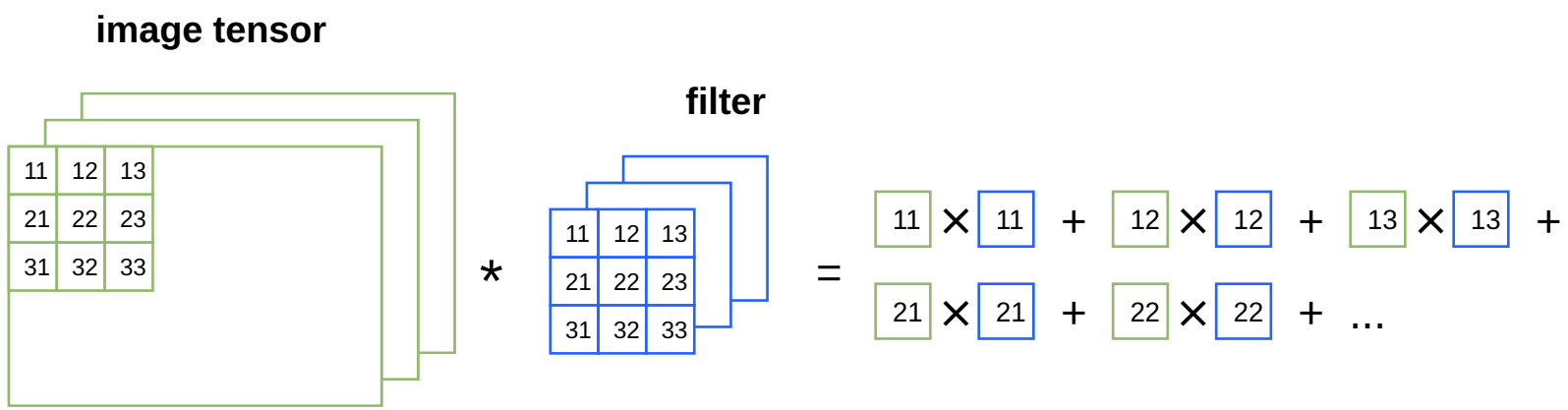
The computational cost of a CNN is primarily derived from convolution layers. A large array of input data is convolved with a large number of much smaller convolution kernels. A simple implementation of convolution of an input of size n with a size k kernel requires $\mathcal{O}(kn)$ operations.



Winograd Convolution

CNNs require large amounts of computing resources for both training and inference: primarily because the convolution layers are computationally intensive.

The computational cost of a CNN is primarily derived from convolution layers. A large array of input data is convolved with a large number of much smaller convolution kernels. A simple implementation of convolution of an input of size n with a size k kernel requires $\mathcal{O}(kn)$ operations.



reduce computational cost by using fast convolution algorithms

Winograd Convolution

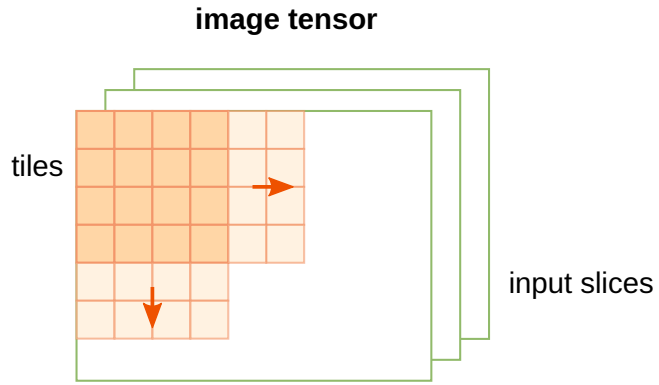
Winograd convolution (Lavin & Gray 2016) samples input and kernel at a given set of points using transformation matrices. This is followed by an element-wise multiplication, and a transformation back to the original space. Such algorithms are called *fast filtering algorithms*.

Winograd Convolution

Winograd convolution (Lavin & Gray 2016) samples input and kernel at a given set of points using transformation matrices. This is followed by an element-wise multiplication, and a transformation back to the original space. Such algorithms are called *fast filtering algorithms*.

Here's how to compute the Winograd convolution:

1. Divide the image tensor into overlapping tiles, as shown in the following diagram:



We here use 4×4 tiles with stride 2, but others are possible.

Winograd Convolution

2. Transform each tile using the following two matrix multiplications:

$$\mathbf{D}_t = (\mathbf{BD})\mathbf{B}^\mathsf{T}$$

$$\begin{bmatrix} \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \bullet \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \right) \bullet \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}^\mathsf{T}$$

In this equation, the matrix **D** is the input slice while **B** is a special matrix, which results from the specifics of the Winograd algorithm (more on this in the paper by Lavin & Gray 2016). A transformed tile $\mathbf{D}_t$ is the result.

Winograd Convolution

3. Transform the filter using the following two matrix multiplications:

$$\mathbf{F}_t = (\mathbf{G}\mathbf{F})\mathbf{G}^\mathsf{T}$$

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \right) \bullet \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}^\mathsf{T}$$

In this equation, the matrix $\mathbf{F}$ is the 3×3 convolution filter between one input and one output slice. $\mathbf{G}$ and its transpose are again special matrices which result from the specifics of the Winograd algorithm. Note that the transformed filter matrix, $\mathbf{F}_t$, has the same dimensions as the input tile, $\mathbf{D}_t$.

Winograd Convolution

4. Compute the transformed output as an element-wise multiplication of the transformed input and filter:

$$\mathbf{O}_t = \mathbf{F}_t \odot \mathbf{D}_t$$

$$\begin{bmatrix} \otimes & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes \end{bmatrix} = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \odot \begin{bmatrix} \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \end{bmatrix}$$

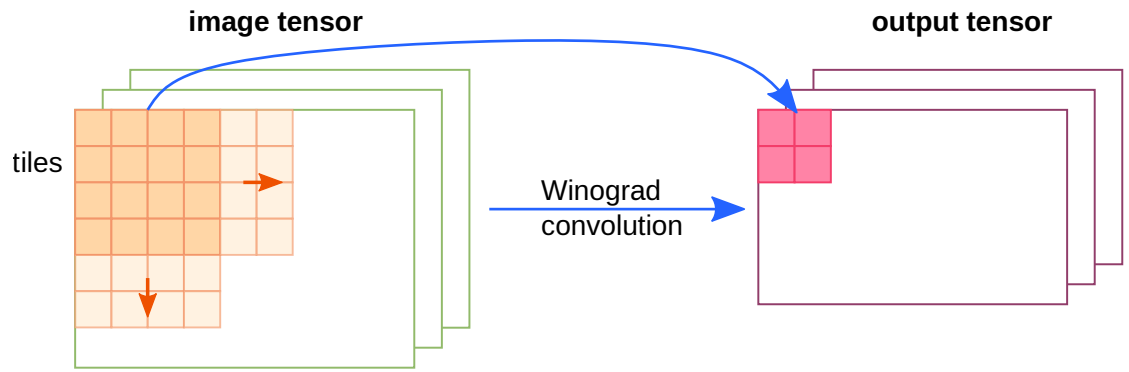
Winograd Convolution

5. Transform the output back into its original form:

$$\mathbf{O} = (\mathbf{A}\mathbf{O}_t)\mathbf{A}^\mathsf{T}$$

$$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} = \left(\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix} \bullet \begin{bmatrix} \otimes & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes \end{bmatrix} \right) \bullet \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}^\mathsf{T}$$

A is a transformation matrix, allowing to transform back into the form which would have resulted from a direct convolution. As shown in the equation and in the following diagram, the Winograd convolution allows us to compute 2×2 output tile simultaneously (four output cells):



Winograd Convolution

At first it looks like the Winograd algorithm performs a lot more operations than direct convolution.
Where is the speed-up?

Winograd Convolution

At first it looks like the Winograd algorithm performs a lot more operations than direct convolution.
Where is the speed-up?

Let's focus on the $\mathbf{D}_t \rightarrow \mathbf{D}$ transformation.

$$\mathbf{D}_t = (\mathbf{BD})\mathbf{B}^\mathsf{T}$$

$$\begin{bmatrix} \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \bullet \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \right) \bullet \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}^\mathsf{T}$$

The key here is that we have to perform $\mathbf{D}_t \rightarrow \mathbf{D}$ only once and then $\mathbf{D}_t$ can participate in the outputs of all K (following the notation) output slices. Therefore, $\mathbf{D}_t \rightarrow \mathbf{D}$ doesn't affect the performance much.

Winograd Convolution

At first it looks like the Winograd algorithm performs a lot more operations than direct convolution.
Where is the speed-up?

Next, let's take a look at the $\mathbf{F}_t \rightarrow \mathbf{F}$ transformation.

$$\mathbf{F}_t = (\mathbf{GF})\mathbf{G}^\mathsf{T}$$

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \right) \bullet \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}^\mathsf{T}$$

This one is even better because, once we compute $\mathbf{F}_t$, we can apply it $N \times P \times Q$ times (across all of the cells of the output slice and all of the images in the batch). Therefore, the performance penalty for this transformation is negligible. Similarly, the output transformation $\mathbf{O}_t \rightarrow \mathbf{O}$ is amortized over the number of input channels C.

Winograd Convolution

At first it looks like the Winograd algorithm performs a lot more operations than direct convolution.
Where is the speed-up?

Finally, we look at the element-wise multiplication, $\mathbf{O}_t = \mathbf{F}_t \odot \mathbf{D}_t$, which is applied $\mathbf{P} \times \mathbf{Q}$ times across all of the cells of the output slice.

$$\mathbf{O}_t = \mathbf{F}_t \odot \mathbf{D}_t$$

$$\begin{bmatrix} \otimes & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes \\ \otimes & \otimes & \otimes & \otimes \end{bmatrix} = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \odot \begin{bmatrix} \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \\ \circ & \circ & \circ & \circ \end{bmatrix}$$

It consists of 16 scalar multiplications and allows us to compute 2×2 output tile, which results in four multiplications per output cell.
We saw that the direct convolution requires $3 \times 3 = 9$ scalar multiplications (each filter element is multiplied by each receptive field input cell) per output. Therefore, the Winograd convolution requires $9/4 = 2.25$ fewer operations.

Winograd Convolution

Conclusion:

The Winograd algorithm can provide 2-3× speedup compared to the direct convolution.

The Winograd convolution has the most benefits when working with smaller filter sizes (for example, 3×3).

Convolutions with larger filters (for example, 11×11) can be efficiently implemented with Fast Fourier Transform (FFT) convolutions, which are beyond the scope of this book.

Training

Larger data sets and models lead to better accuracy but also increase computation time. Progress in deep neural networks is limited by how fast the networks can be computed.

Also the application of large CNN to low-latency inference problems can be critical: self-driving cars with pedestrian and car detection, robots working in dangerous environments.

Optimized training algorithms can lead to more efficient models.

Training

Acceleration

- Run a network faster (Performance, bit/s)
- Run a network more efficiently
  - Energy (bit/J)
  - Cost (bit/s$)

Training

- Running the network forward
- Back-propagation of gradient
- Update of parameters

Inference

- running the network forward

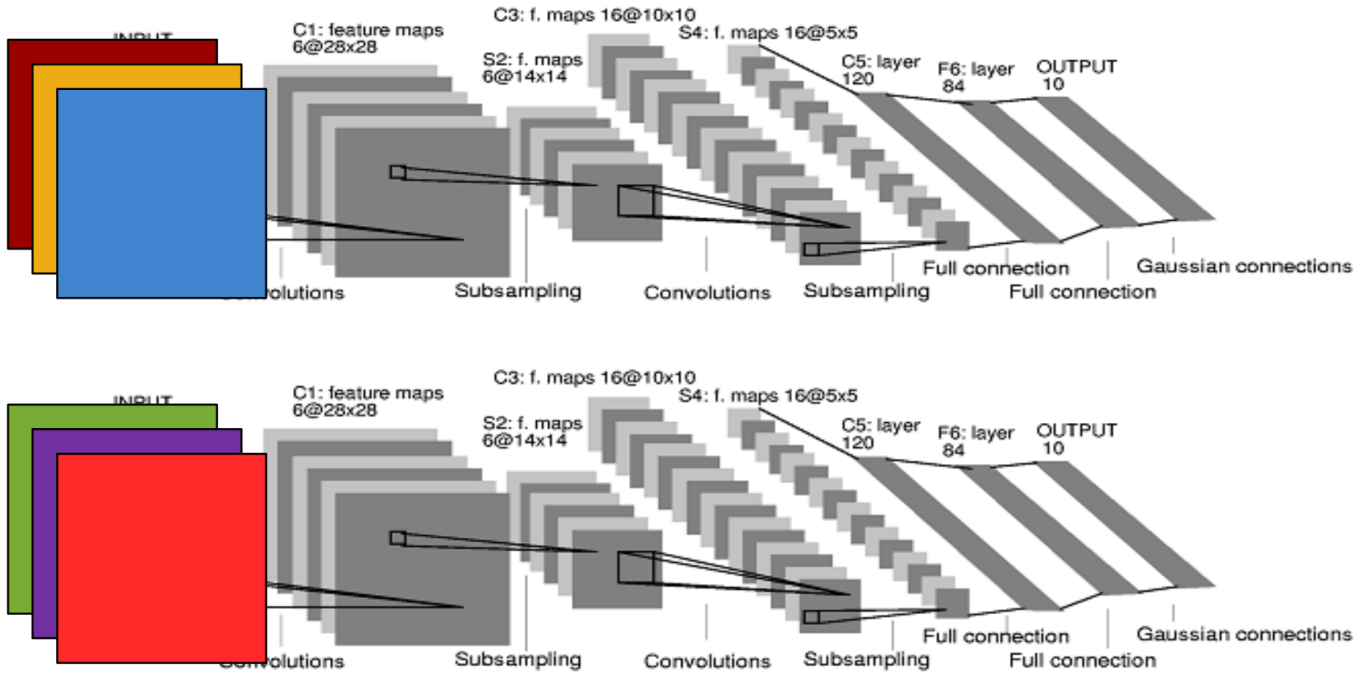
Parallelization

Lots of parallelism in a Deep Neural Network:

- Inputs
- Points of a feature map
- Filters
- Elements within a filter
- Multiplications within layer are independent
- Sums are reductions
- Only layers are dependent

Parallelization

Data Parallel - Run multiple inputs in parallel

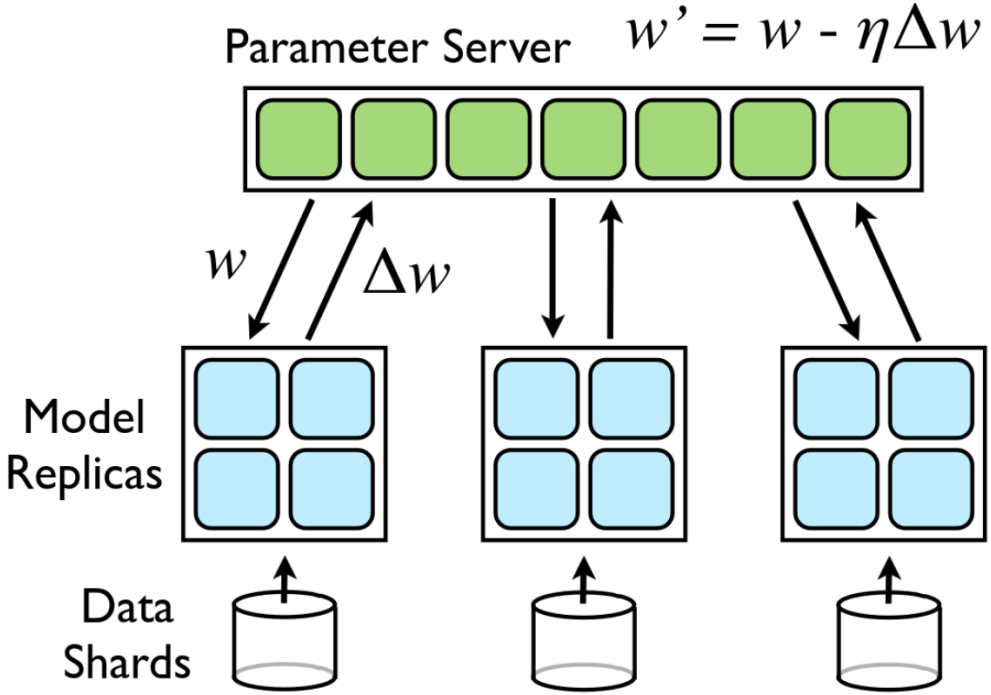


- doesn't affect latency for one input
- requires P-fold large batch size
- for training requires coordinated weight update

credit: Dally, High Performance Hardware for Learning, NIPS'2015

Parallelization

Parameter Update

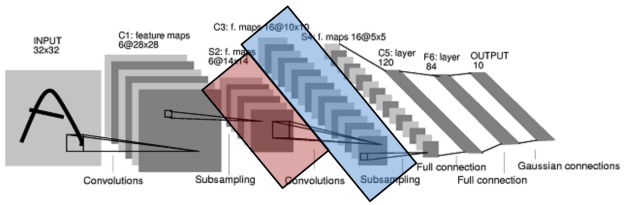


credit: J. Dean (2013), Large Scale Disributed Deep Networks

Parallelization

Model Parallel: Split up the Model i.e. the network

The model-parallel convolution outputs per region (x, y)



Kernels
Multiple 3D
K_{uvkj}

x

A_{xyk}

B_{xyj} B_{xyj}
B_{xyj} B_{xyj}

Input maps
A_{xyk}

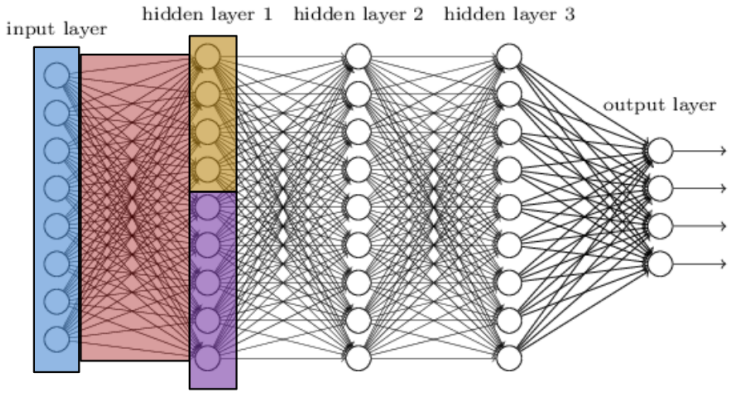
Output maps
B_{xyj}

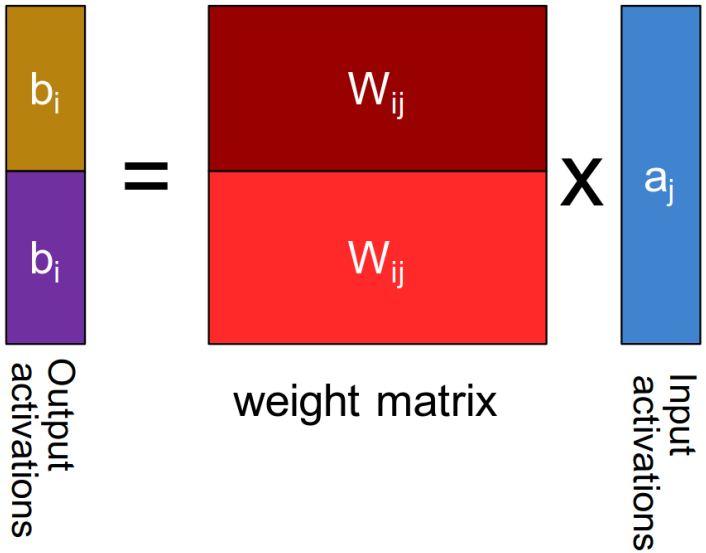
6D Loop
Forall region XY
  For each output map j
    For each input map k
      For each pixel x,y in XY
        For each kernel element u,v
          B_{xyj} += $A_{(x-u)(y-v)k}$ x K_{uvkj}

credit: Dally, High Performance Hardware for Learning, NIPS'2015

Parallelization

Model Parallel: Split up the Model i.e. the network





Parallelization

Summary

A lot of parallelism in Deep Neural Networks

Data parallel

- run multiple training examples in parallel
- limited by batch size

Model parallel

- split model over multiple processors
- by layer
- convolutional layers by map region
- fully connected layers by output activation

Mixed Precision Training

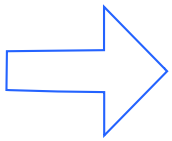
Increasing the size of a neural network typically improves accuracy but also increases the memory and compute requirements for training the model.

Mixed Precision Training involves the usage of half-precision floating point numbers durng training, without losing model accuracy or having to modify hyper- parameters.

Mixed Precision Training

Increasing the size of a neural network typically improves accuracy but also increases the memory and compute requirements for training the model.

Mixed Precision Training involves the usage of half-precision floating point numbers durng training, without losing model accuracy or having to modify hyper- parameters.



This nearly halves memory requirements and, on recent GPUs, speeds up arithmetic.

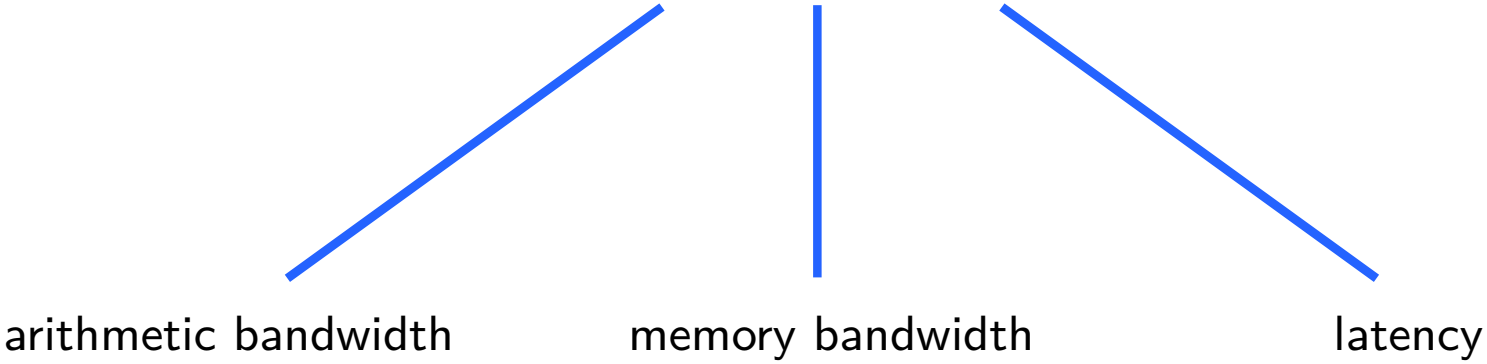
Mixed Precision Training

To overcome the narrower of half-precision, Micikevicius et al. (2017) propose three techniques for preventing the loss of critical information:

- maintaining a single-precision copy of weights that accumulates the gradients after each optimizer step (this copy is rounded to half-precision for the forward- and back-propagation)
- loss-scaling to preserve gradient values with small magnitudes
- half-precision arith-etic that accumulates into single-precision outputs, which are converted to half- precision before storing to memory.

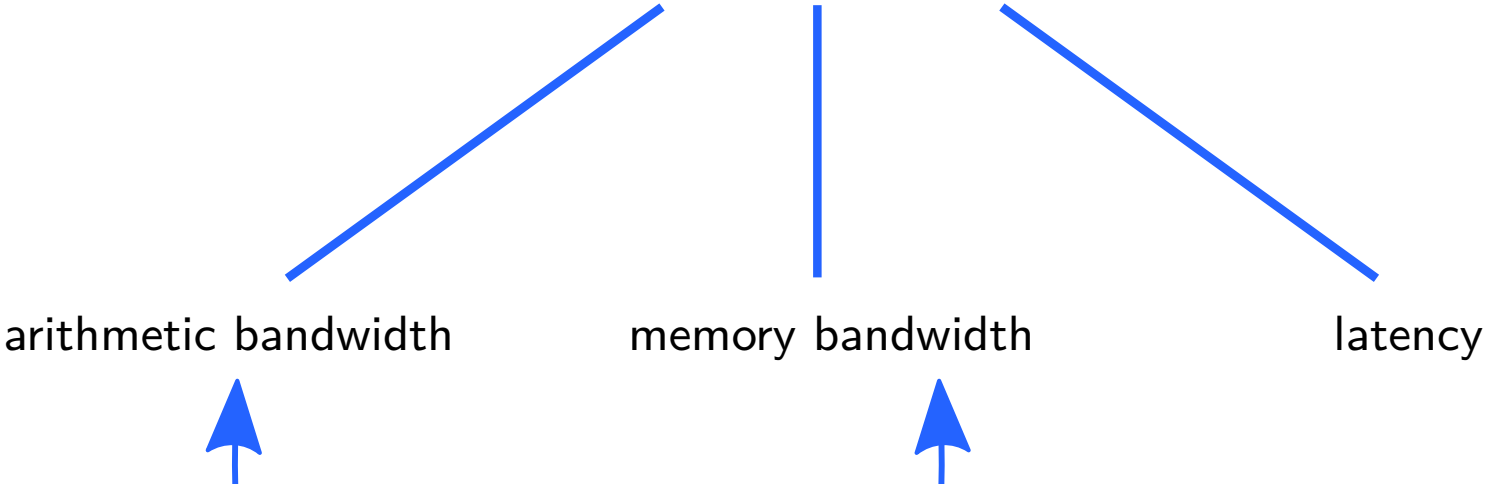
Mixed Precision Training

Performance of any program, including neural network training and inference, is limited by one of three factors:

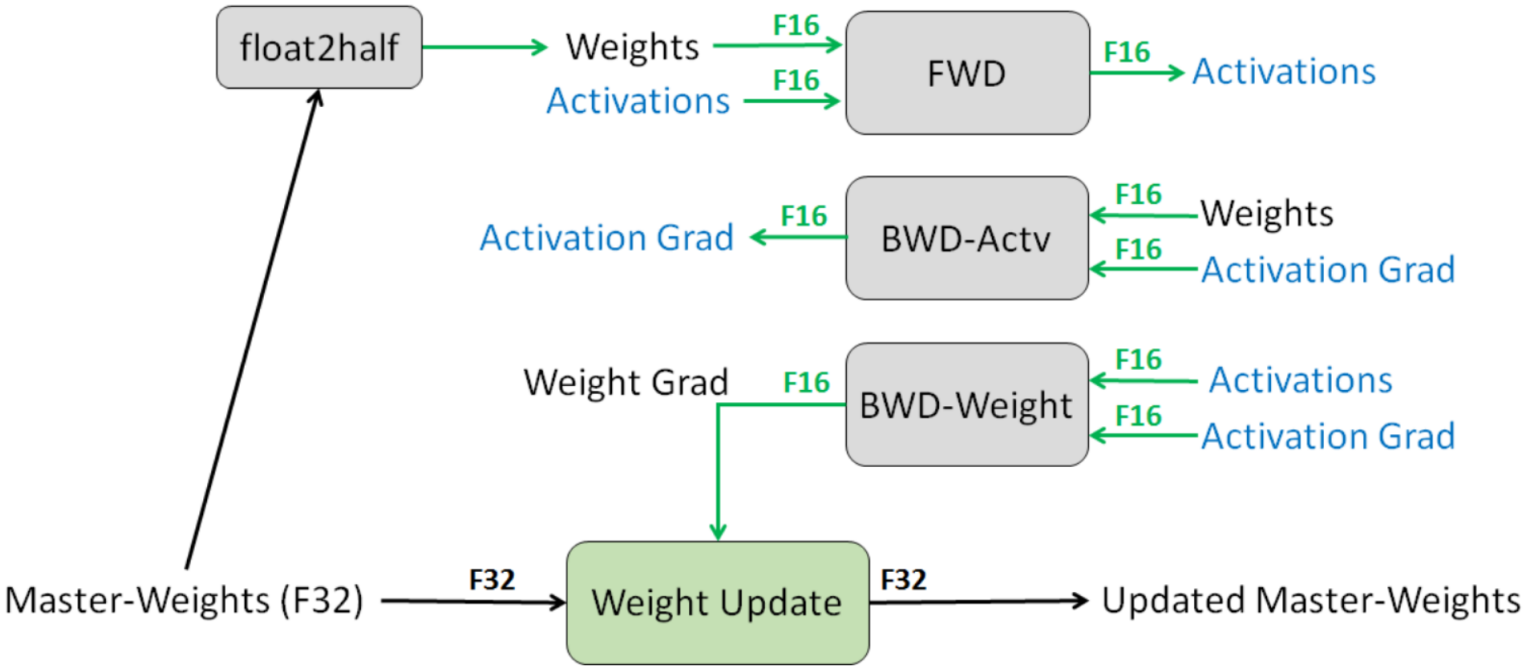


Mixed Precision Training

Performance of any program, including neural network training and inference, is limited by one of three factors:



Mixed Precision Training



Training iteration for a layer. Credit: Micikevicius et al. (2017)

Weights, activations and gradients are stored as FP16. To match the accuracy of the FP32 networks, an FP32 master copy of weights is maintained and updated with the weight gradient during optimization. In each iteration a FP16 copy of the master weights is used in the forward and backward pass, halving the storage and bandwidth w.r.t. FP32 training.

Model Distillation

idea: The easiest way to extract a lot of knowledge from the training data is to learn many different models in parallel.

Model Distillation

idea: The easiest way to extract a lot of knowledge from the training data is to learn many different models in parallel.

concept:

- We want to make the models as different as possible to minimize the correlations between their errors.
- We can use different initializations or different architectures or different subsets of the training data.
- It is helpful to over-fit the individual models.

At test time we average the predictions of all the models or of a selected subset of good models that make different errors.

Model Distillation

Hinton et al. (2015) uses **soft targets** as a way to transfer the function

If the output is a big N-way softmax, the targets are usually a single 1 and a whole lot of 0's.

On average each target puts $\leq \log N$ bits of constraint on the function.

If we have the ensemble, we can divide the averaged logits from the ensemble by a *temperature* to get a much softer distribution:

$$p_i = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_i}{T}\right)}$$

An example of hard and soft targets:

cow	dog	cat	car	
0	1	0	0	original hard targets
10^{-6}	.9	.1	10^{-9}	output of geometric ensemble
.05	.3	.2	.005	softened output of ensemble

Summary

Subtle but important design choices within a given Neural Network architecture (CNN, RNN...) can lead to dramatic improvements in computing time, memory and disk space usage and energy efficiency.

recommendation: after a trained model works well, look into ways to improve its efficiency without losing accuracy