Deep Compression: Compressing Deep Neural Networks With Pruning, Trained Quantization and Huffman Coding

Song Han, Huizi Mao, William J. Dally

First presented at: International Conference on Learning Representations (ICLR) 2016

Presented by: Hong Chul Nam

Seminar in Computer Architecture

Executive Summary

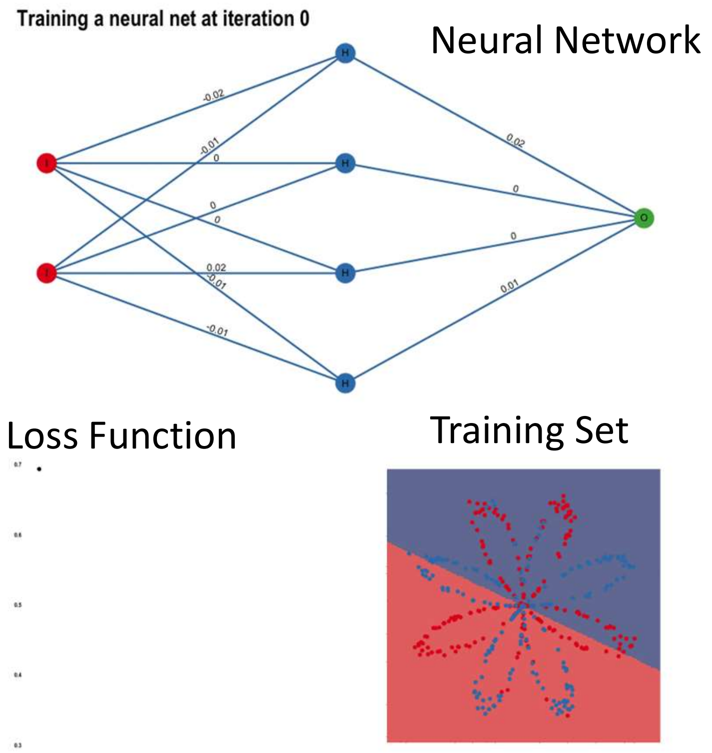
- Problem
  - Large DNNs are hard to be fitted into a **resource-restraint** environment
  - Current DNNs are mostly **too large**
- Goal
  - **Compress** large DNNs into a smaller one such that memory fetching is minimized
- Key idea
  - **Adaptive pruning** – use less weights
  - **Adaptive quantization** – use less bits per weight
  - **Huffman encoding** – use less bits per character sequence
- Results
  - A **much smaller** network able to be fitted into the mobile platform

Background, Problem & Goal

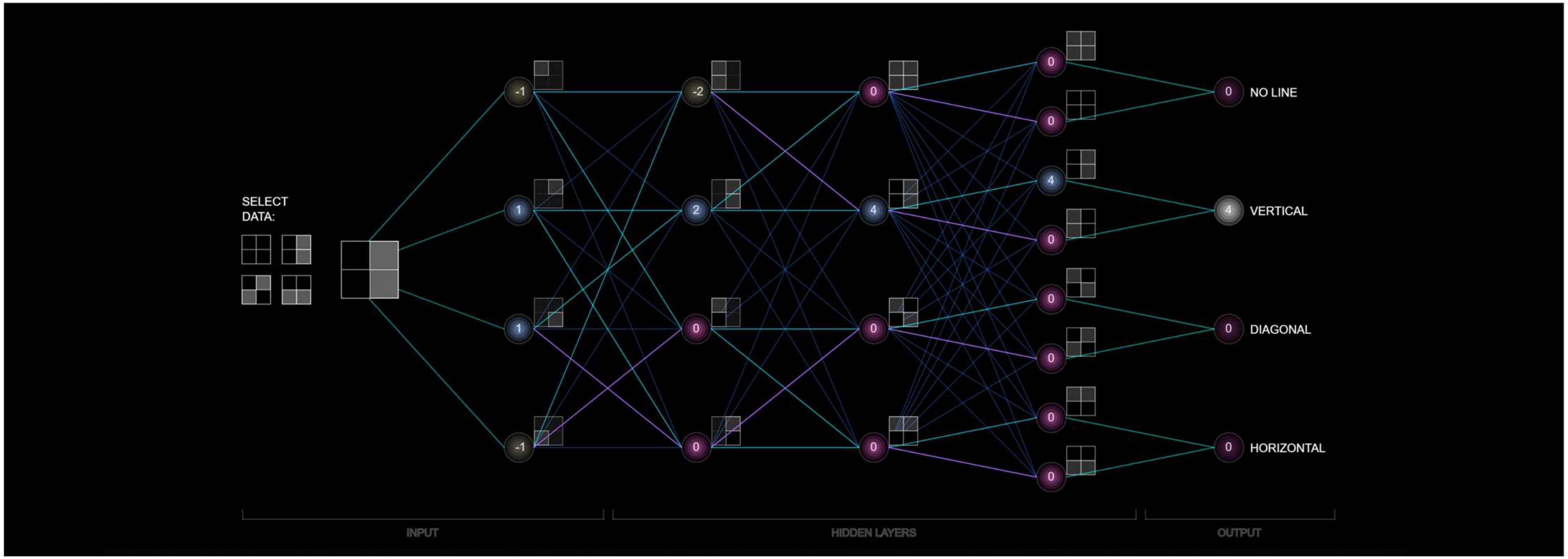
Neural Network: Theory

- What is a neural network?

Neural Network: Training Phase (I)

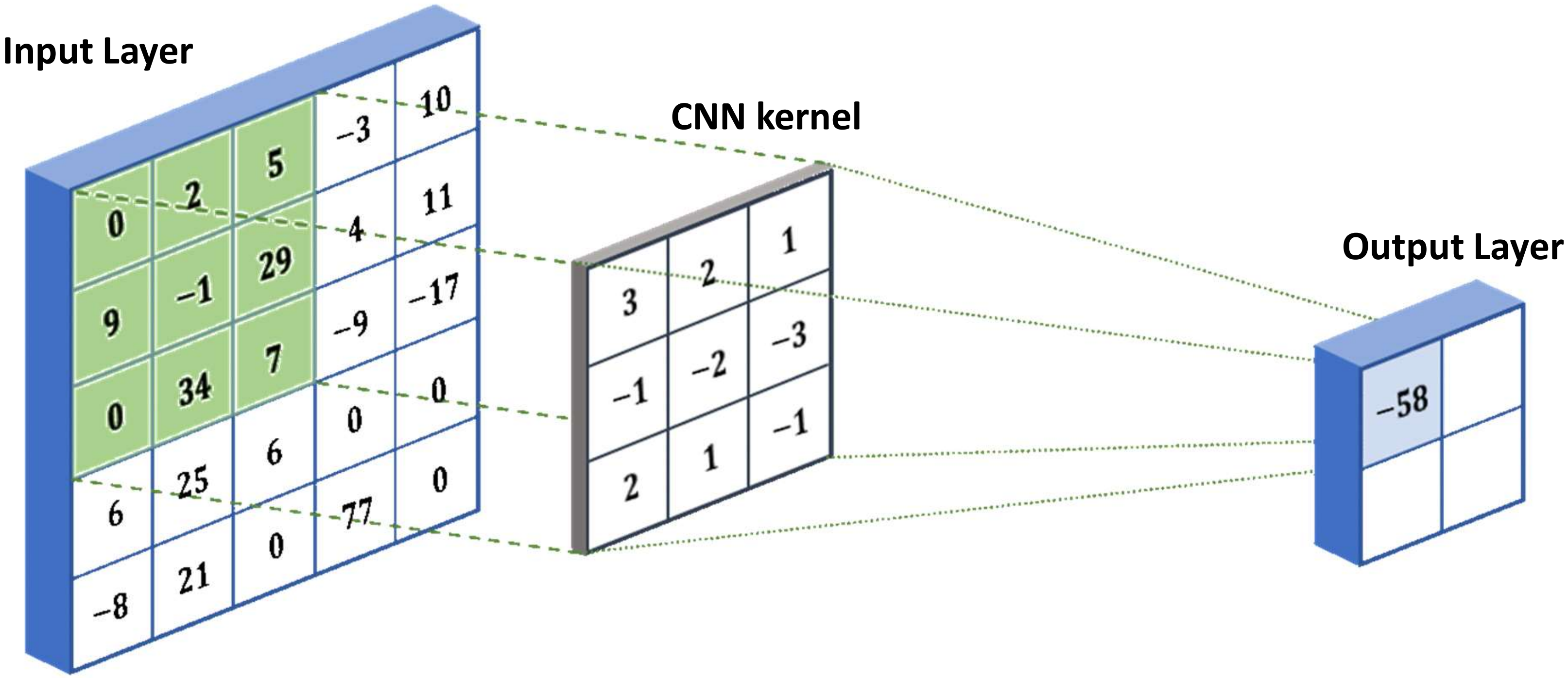


Neural Network: Inference Phase (II)

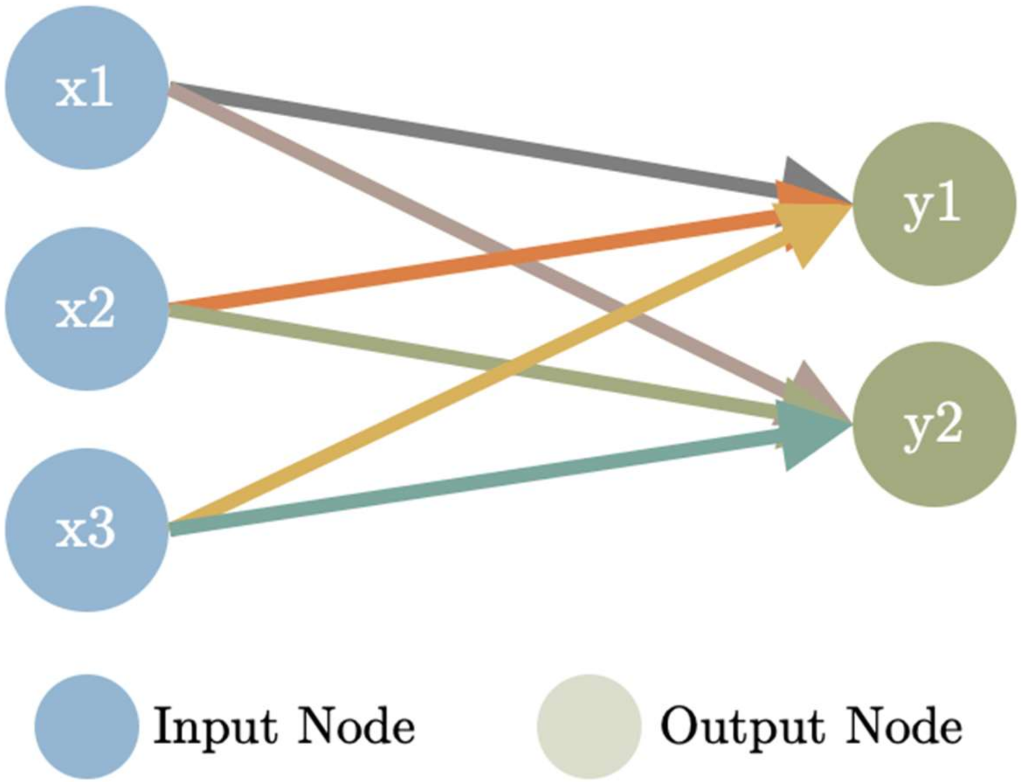


Source: https://medium.com/colaberry-labs/toy-neural-network-classifies-orientation-of-line-acf143b89c22

Possible Layers: CNN Layer



Possible Layers: Dense Layer



Huffman Coding

- A widely-used lossless compression algorithm
- Idea:
  - Shorter sequence for frequently appearing object
  - Longer sequence for rarely appearing object
  - Sequence length determined by appearing frequency

Huffman Coding Example (I)

Frequency	1	6	5	3
Vocabulary	B	C	A	D

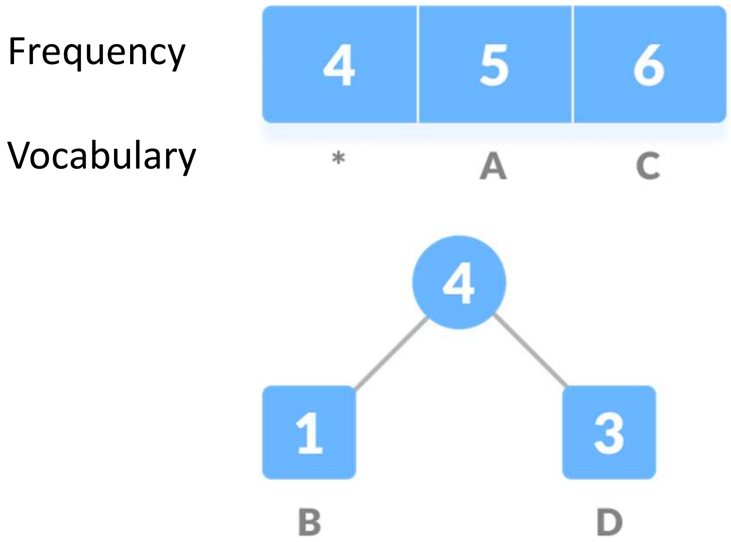
Source: https://www.programiz.com/dsa/huffman-coding

Huffman Coding Example (II)

Frequency	1	3	5	6
Vocabulary	B	D	A	C

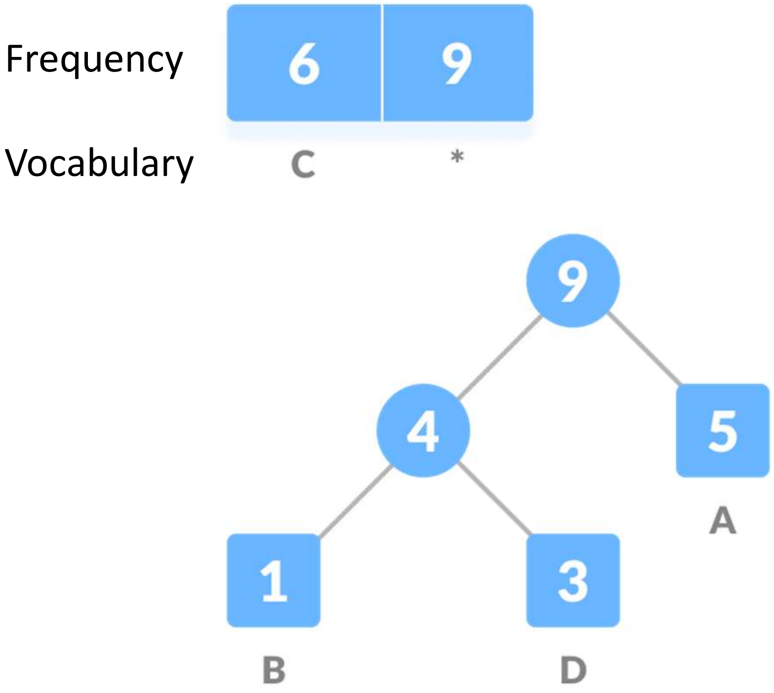
Source: https://www.programiz.com/dsa/huffman-coding

Huffman Coding Example (III)



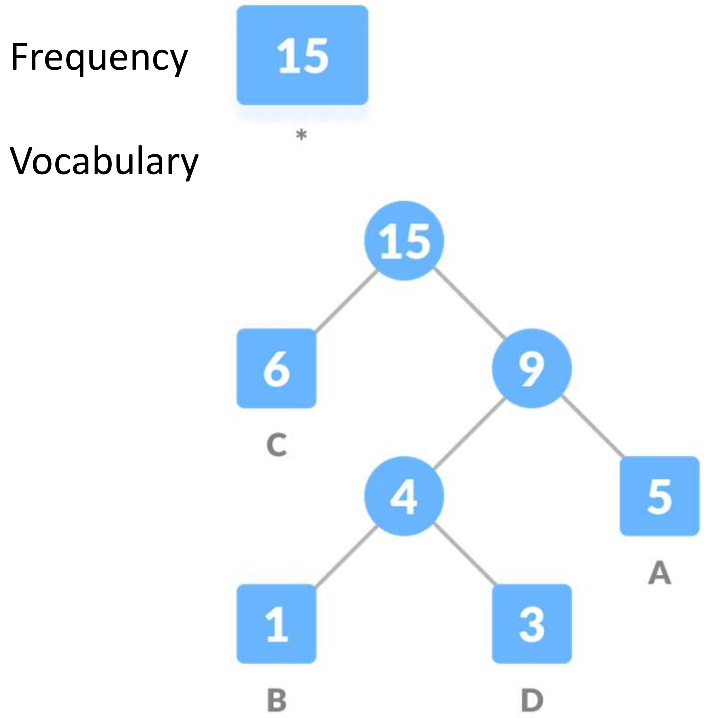
Source: https://www.programiz.com/dsa/huffman-coding

Huffman Coding Example (IV)



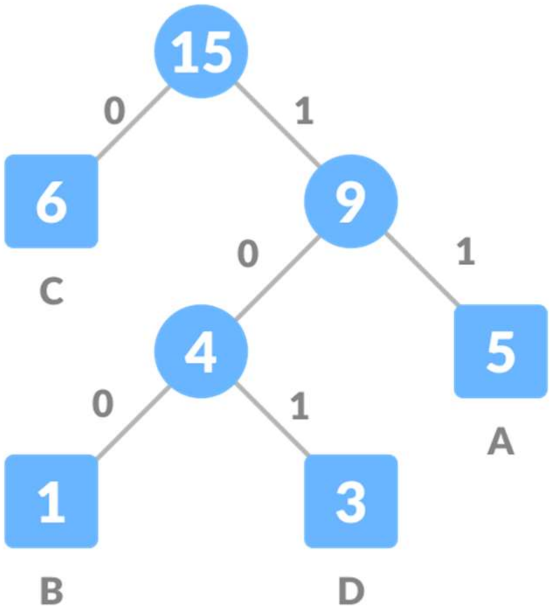
Source: https://www.programiz.com/dsa/huffman-coding

Huffman Coding Example (V)



Source: https://www.programiz.com/dsa/huffman-coding

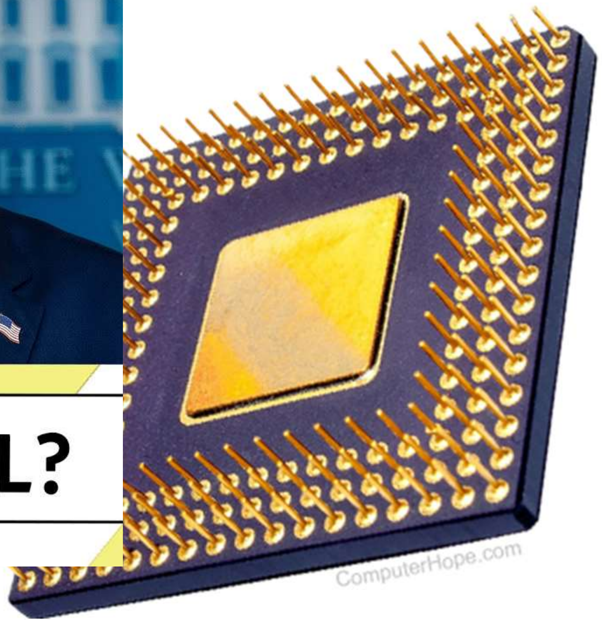
Huffman Coding Example (VI)



Source: https://www.programiz.com/dsa/huffman-coding

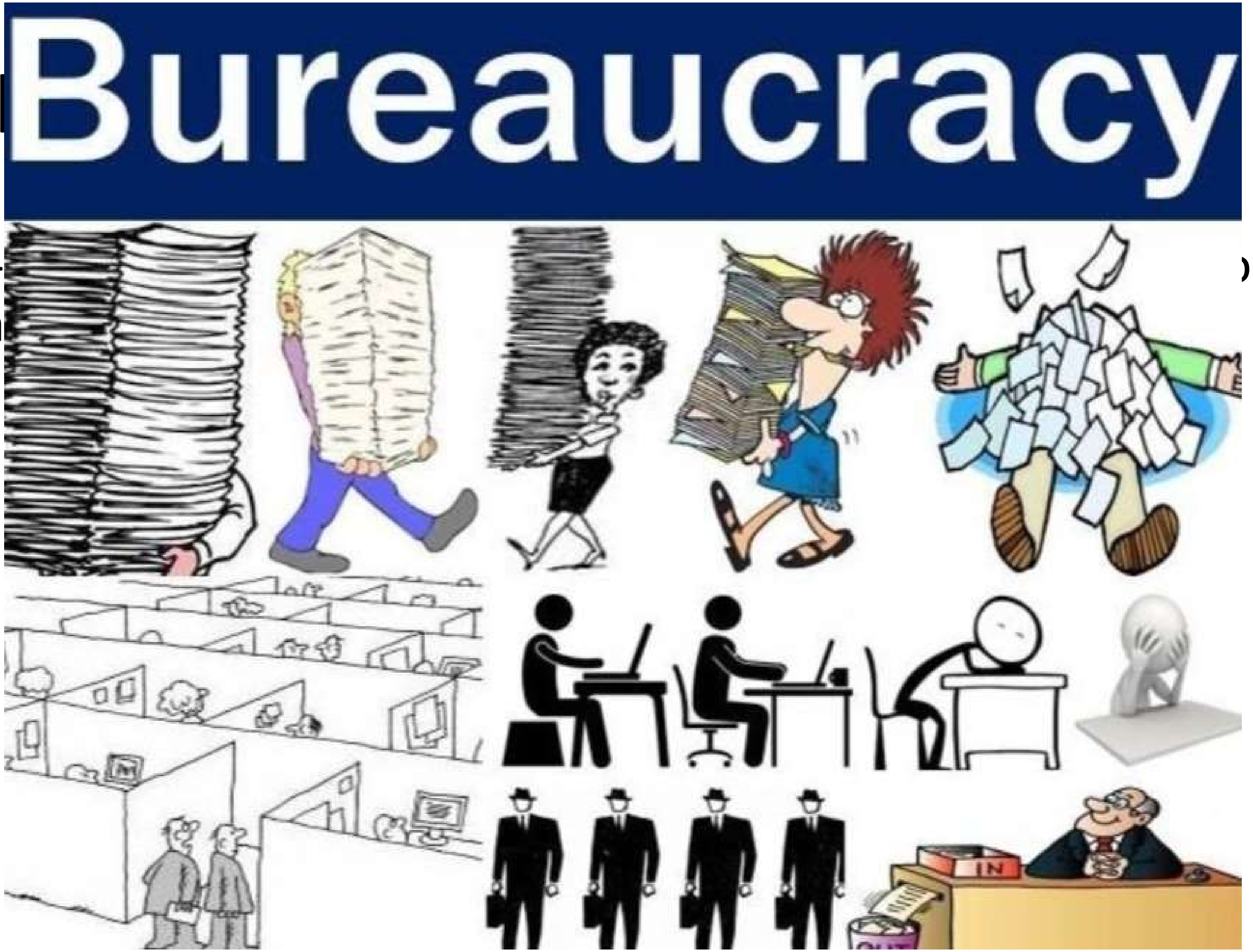
Problem I: Large File Sizes (I)

- Memory banc



Source: https://www.politifact.com/truth-o-meter/promises/trumpometer/promise/1397/build-wall-and-make-mexico-pay-it/

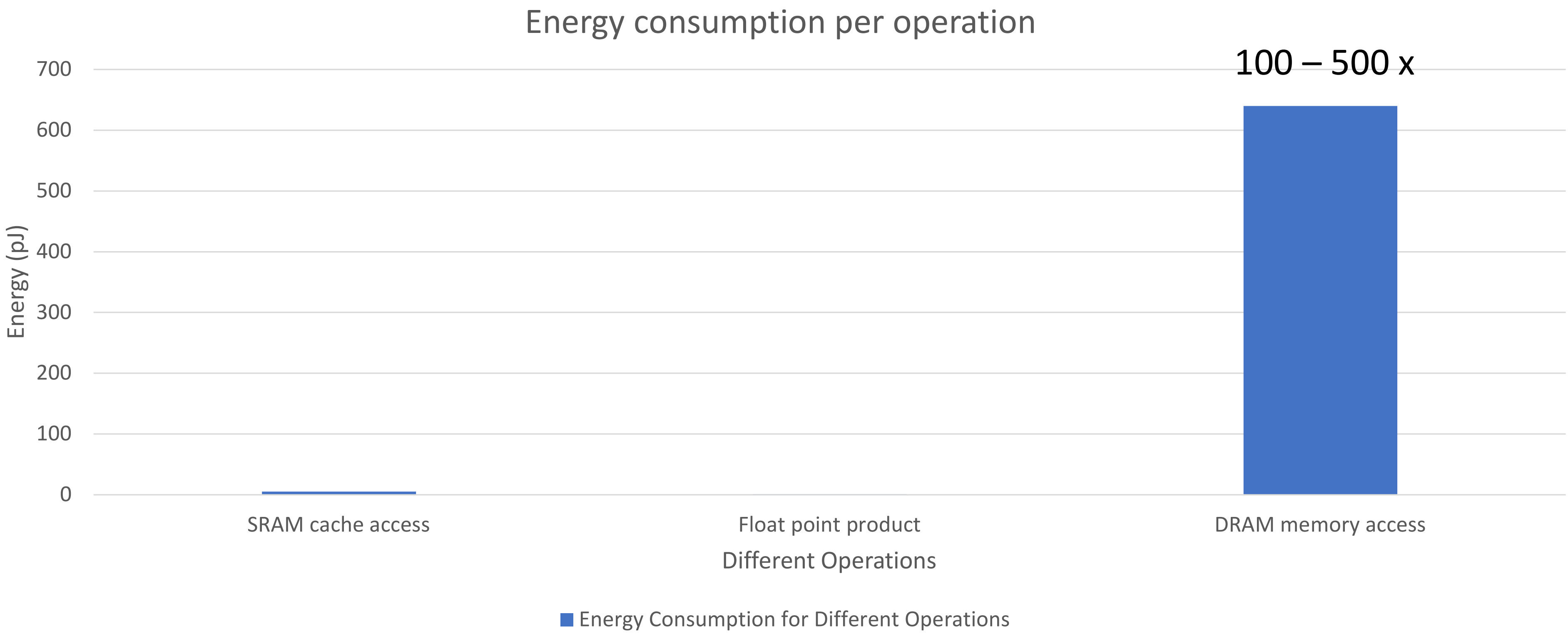
Probl



- Large-
  appea

Problem II: Energy Consumption

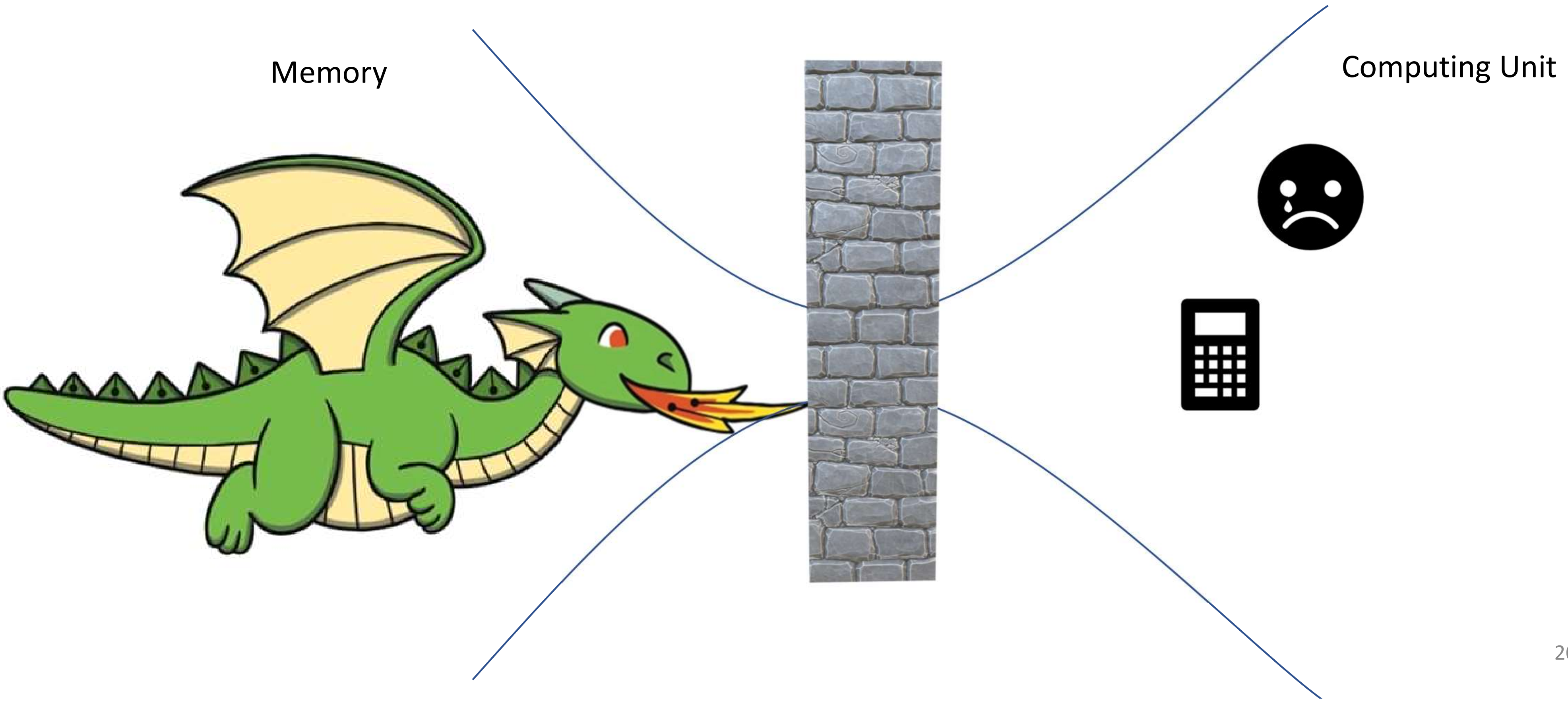
- Too many memory fetching for weights!



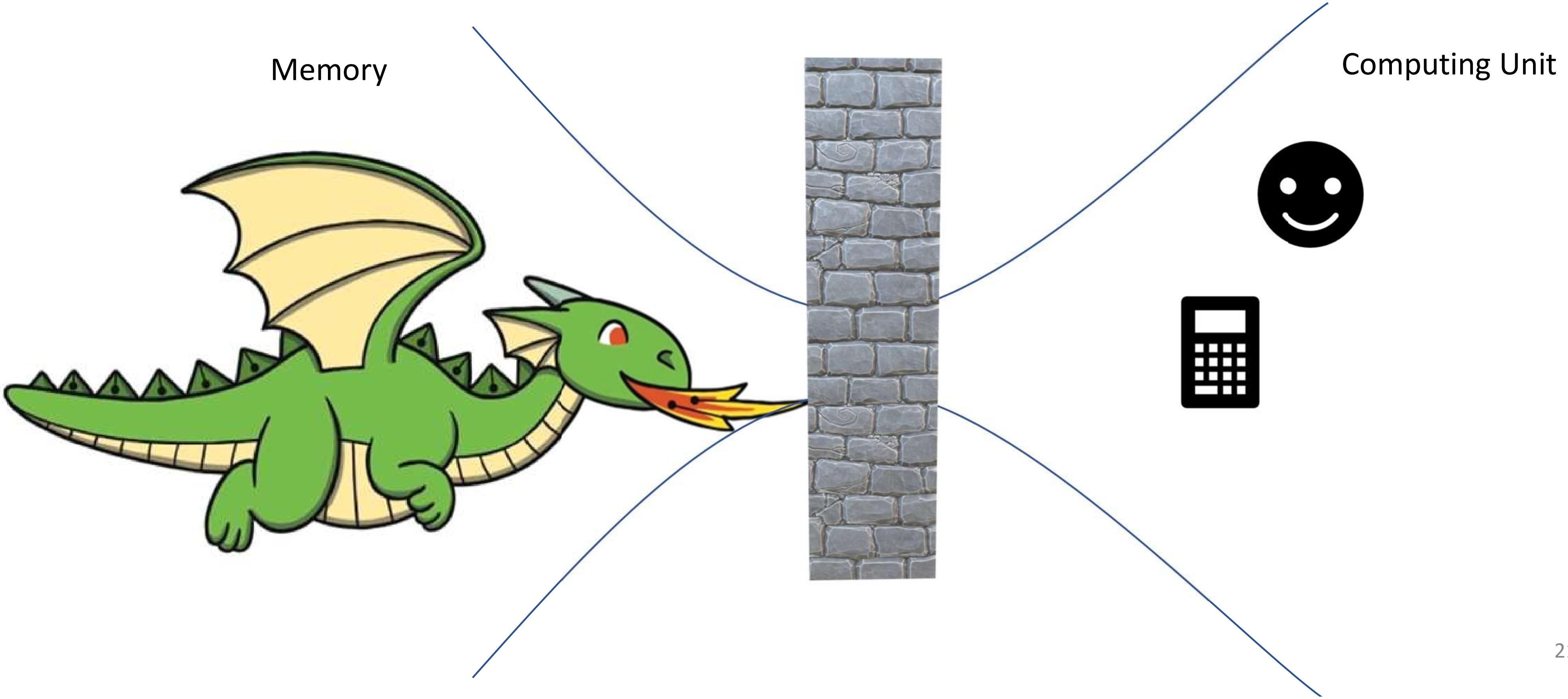
Goal

- Compress large deep neural networks such that access to DRAM for fetching weights could be minimized
- Enable running the DNN directly on mobile devices

Goal

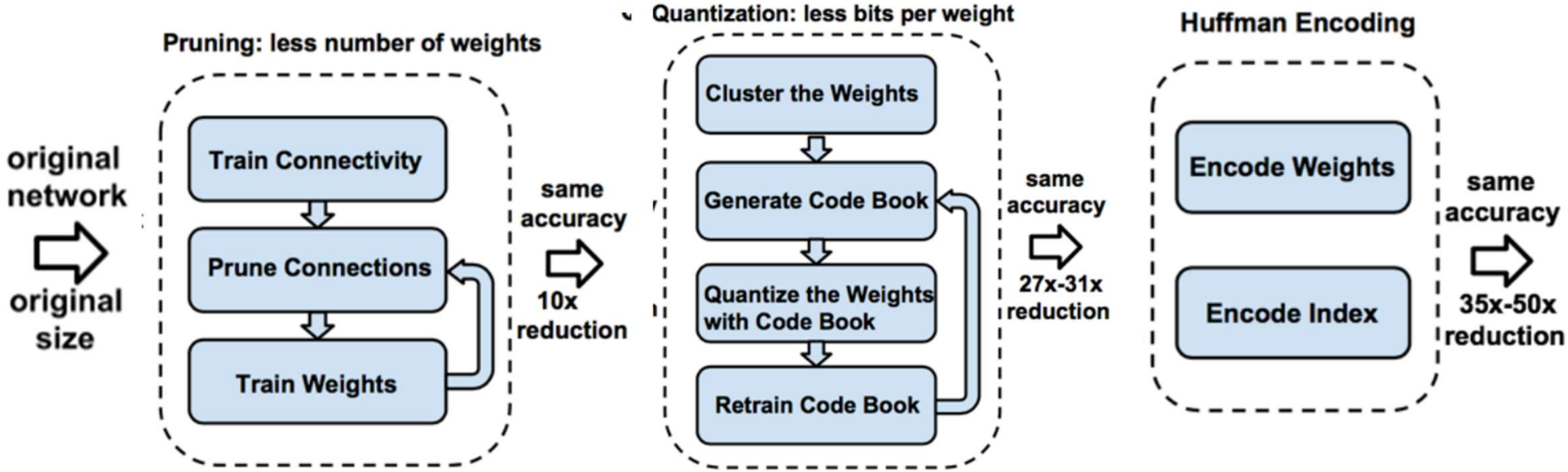


Goal

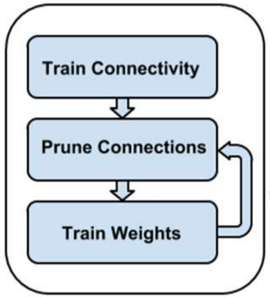


Implementation

Overview: Three-staged Compression



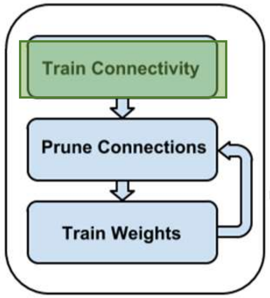
Pruning: Idea



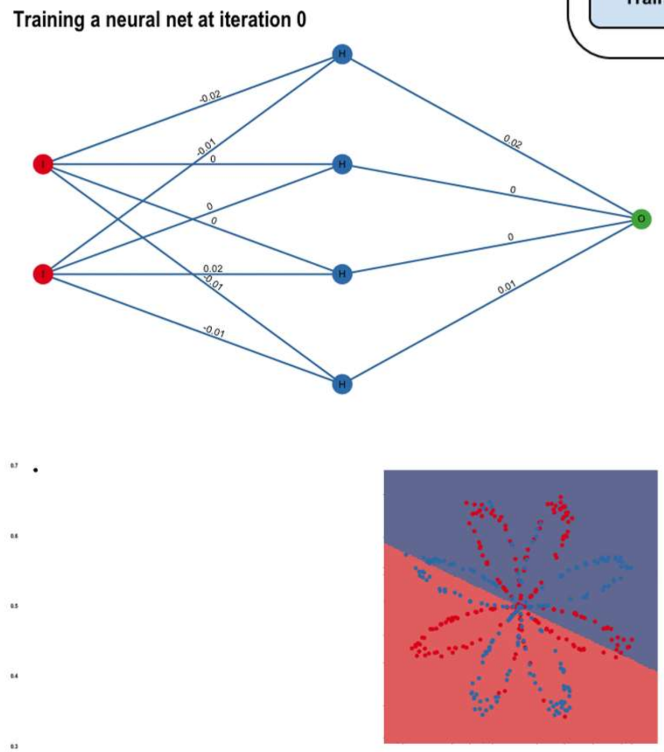
- A lot of neural networks are overparametrized
- Many weights are either zero or close to zero
- These zero-ish weights do not contribute much to the result

IF ALMOST ZERO, WHY NOT SET THEM ZERO AT ALL?

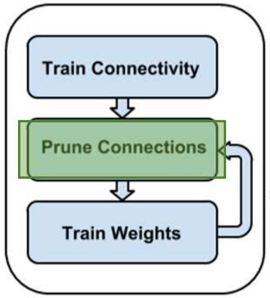
Pruning: Implementation (I)



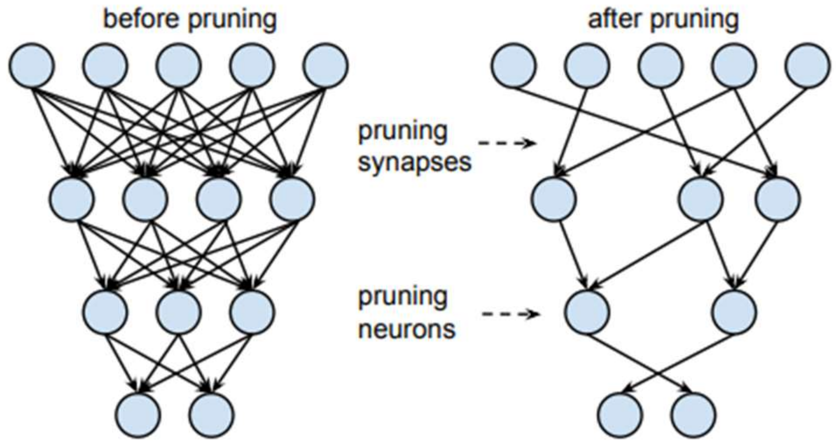
- Train connectivity
  - Train the original dense network
- Prune Connections
  - $\theta_{i,j} = 1_{|\theta_{i,j}| \leq T} \theta_{i,j}$ with T = threshold
- Train Weights
  - Retrain the network
- Storage in CSR format
  - Compressed Sparse Row
  - A storing format for sparse matrix



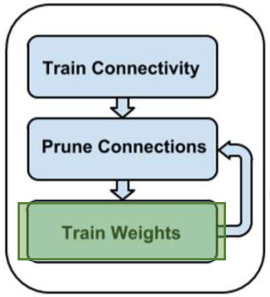
Pruning: Implementation (II)



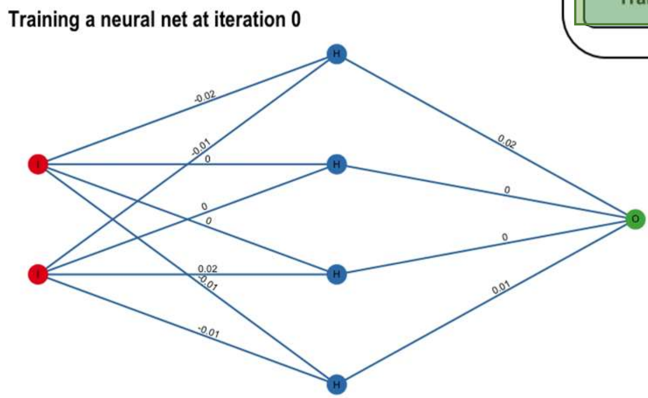
- Train connectivity
  - Train the original dense network
- Prune Connections
  - $\theta_{i,j} = 1_{|\theta_{i,j}| \geq T} \theta_{i,j}$ with T = threshold
- Train Weights
  - Retrain the network
- Storage in CSR format
  - Compressed Sparse Row
  - A storing format for sparse matrix

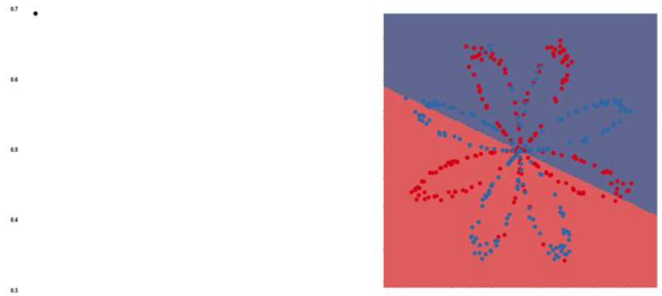


Pruning: Implementation (III)

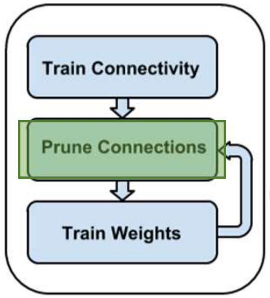


- Train connectivity
  - Train the original dense network
- Prune Connections
  - $\theta_{i,j} = 1_{|\theta_{i,j}| \le T}\theta_{i,j}$ with T = threshold
- Train Weights
  - Retrain the network
- Storage in CSR format
  - Compressed Sparse Row
  - A storing format for sparse matrix

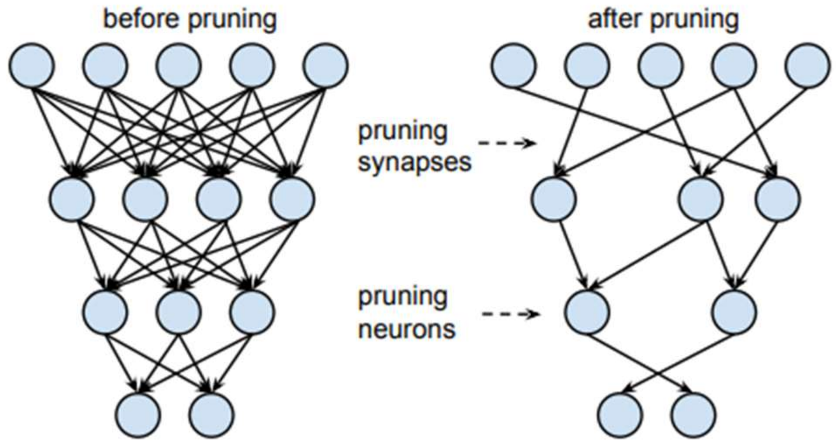




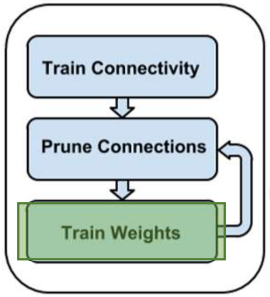
Pruning: Implementation (II)



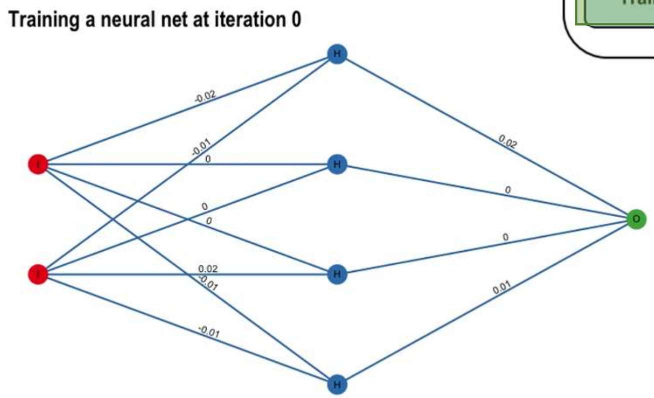
- Train connectivity
  - Train the original dense network
- Prune Connections
  - $\theta_{i,j} = 1_{|\theta_{i,j}| \leq T} \theta_{i,j}$ with T = threshold
- Train Weights
  - Retrain the network
- Storage in CSR format
  - Compressed Sparse Row
  - A storing format for sparse matrix

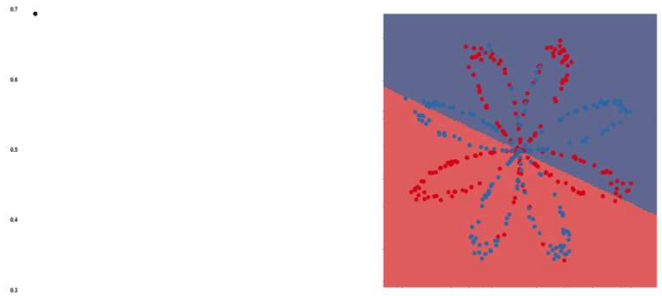


Pruning: Implementation (III)

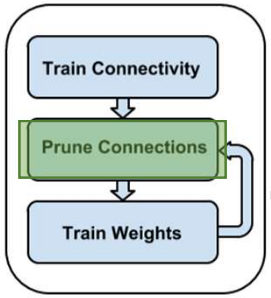


- Train connectivity
  - Train the original dense network
- Prune Connections
  - $\theta_{i,j} = 1_{|\theta_{i,j}| \leq T} \theta_{i,j}$ with T = threshold
- Train Weights
  - Retrain the network
- Storage in CSR format
  - Compressed Sparse Row
  - A storing format for sparse matrix

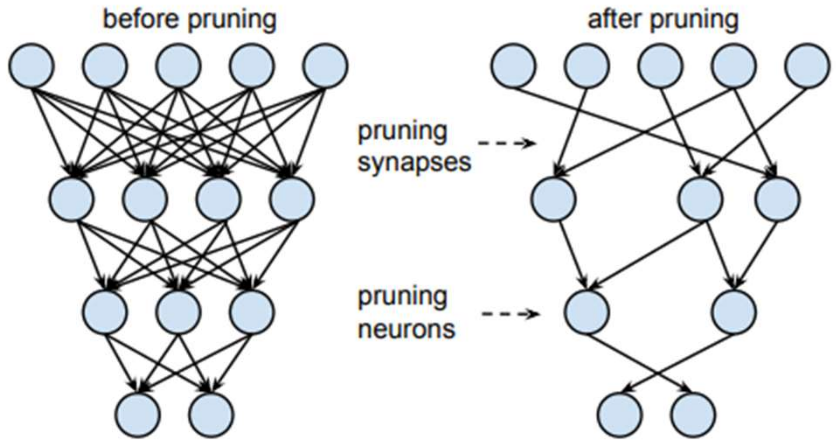




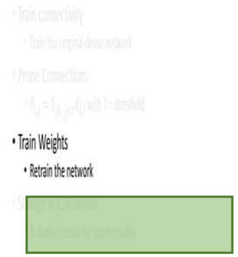
Pruning: Implementation (II)



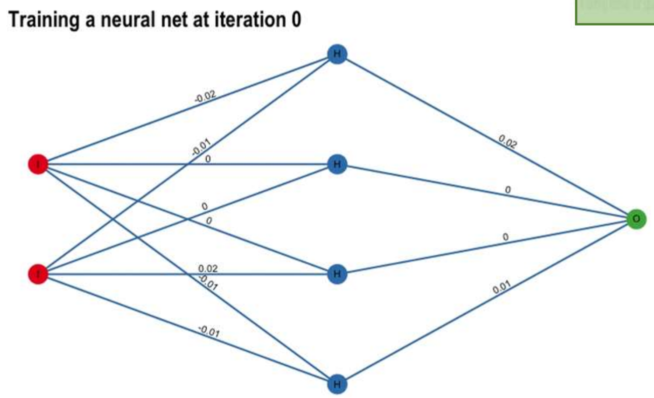
- Train connectivity
  - Train the original dense network
- Prune Connections
  - $\theta_{i,j} = 1_{|\theta_{i,j}| \leq T} \theta_{i,j}$ with T = threshold
- Train Weights
  - Retrain the network
- Storage in CSR format
  - Compressed Sparse Row
  - A storing format for sparse matrix

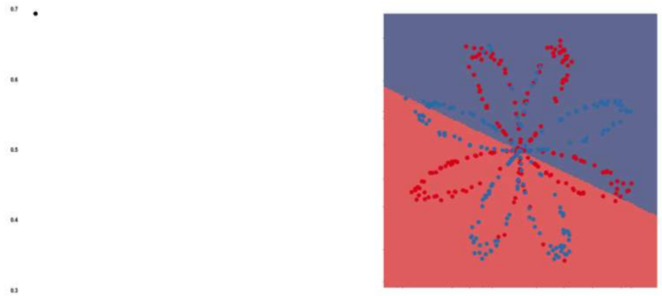


Pruning: Implementation (III)

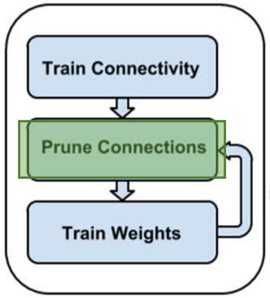


- Train connectivity
  - Train the original dense network
- Prune Connections
  - $\theta_{i,j} = 1_{|\theta_{i,j}| \le T}\theta_{i,j}$ with T = threshold
- Train Weights
  - Retrain the network
- Storage in CSR format
  - Compressed Sparse Row
  - A storing format for sparse matrix

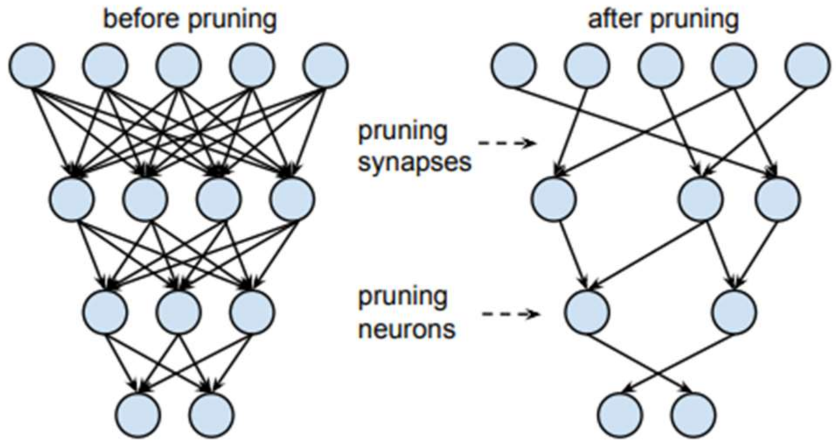




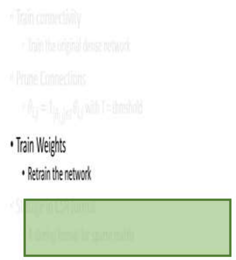
Pruning: Implementation (II)



- Train connectivity
  - Train the original dense network
- Prune Connections
  - $\theta_{i,j} = 1_{|\theta_{i,j}| \leq T} \theta_{i,j}$ with T = threshold
- Train Weights
  - Retrain the network
- Storage in CSR format
  - Compressed Sparse Row
  - A storing format for sparse matrix



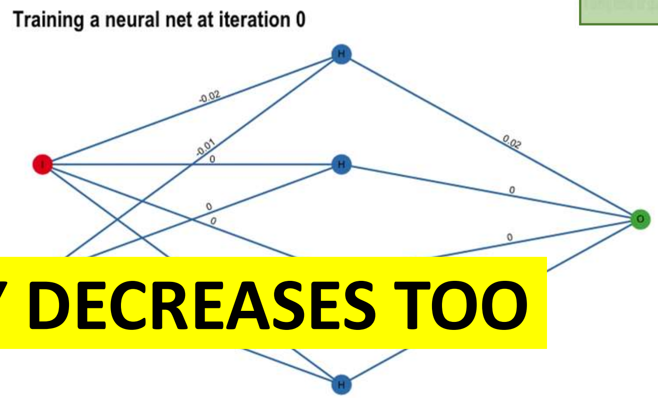
Pruning: Implementation (III)

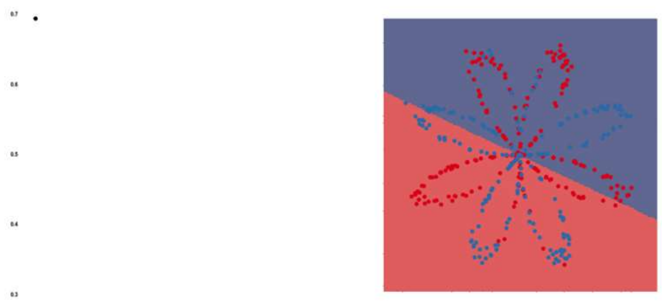


- Train connectivity
  - Train the original dense network
- Prune Connections
  - $\theta_{i,j}$

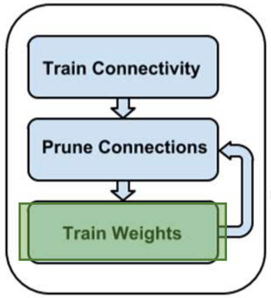
STOP WHEN THE ACCURACY DECREASES TOO MUCH

- Train W
  - Retrain the network
- Storage in CSR format
  - Compressed Sparse Row
  - A storing format for sparse matrix

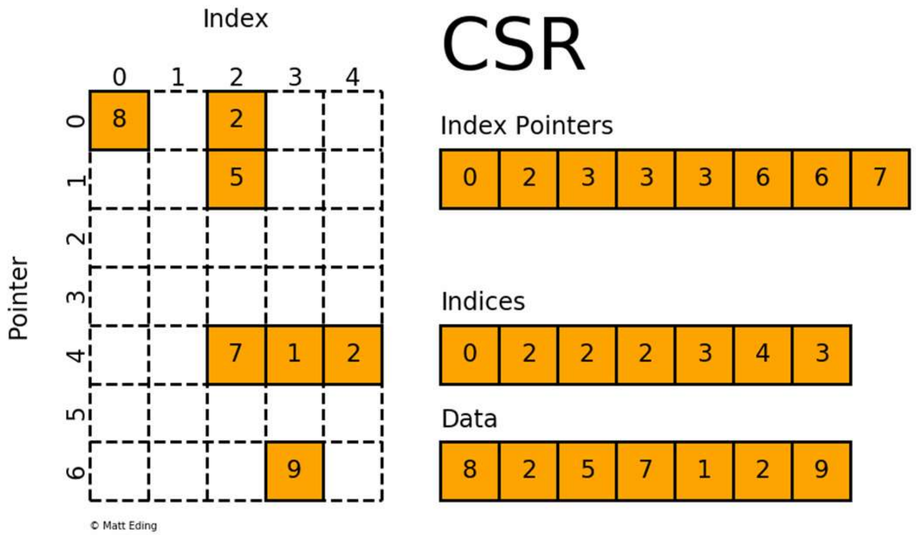




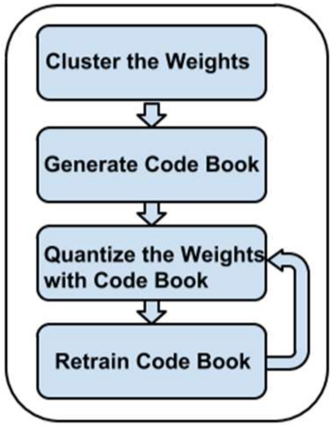
Pruning: Implementation



- Train connectivity
  - Train the original dense network
- Prune Connections
  - $\theta_{i,j} = 1_{|\theta_{i,j}| \le T}\theta_{i,j}$ with T = threshold
- Train Weights
  - Retrain the network
- Storage in CSR format
  - Compressed Sparse Row
  - A storing format for sparse matrix



Quantization: Idea



High energy efficiency

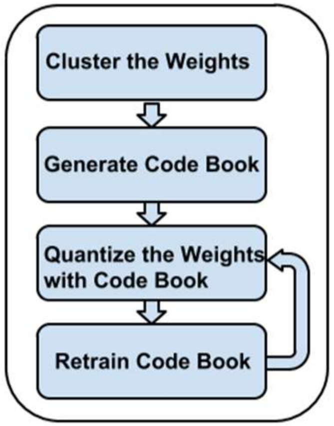
Low precision

A MIDDLE GROUND

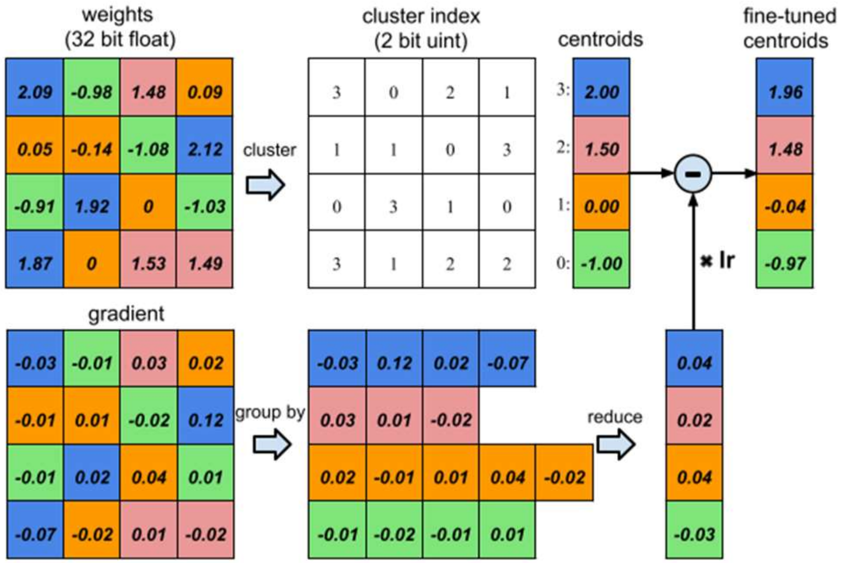
Low energy efficiency

High precision

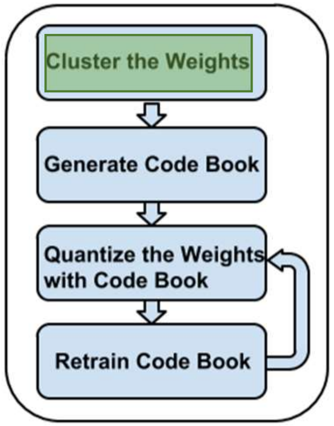
Quantization: Implementation



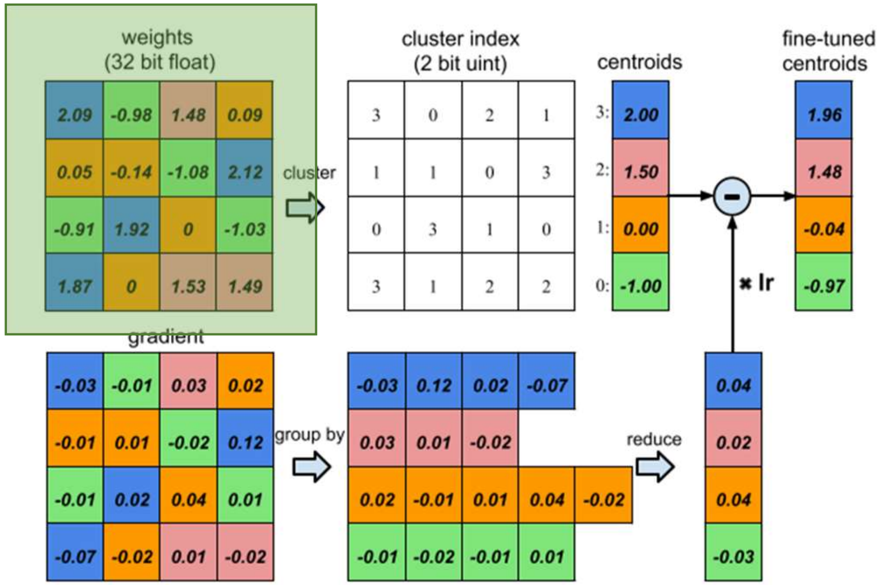
- K-means clustering on weights to find centroids
- Match all weights into the corresponding centroids

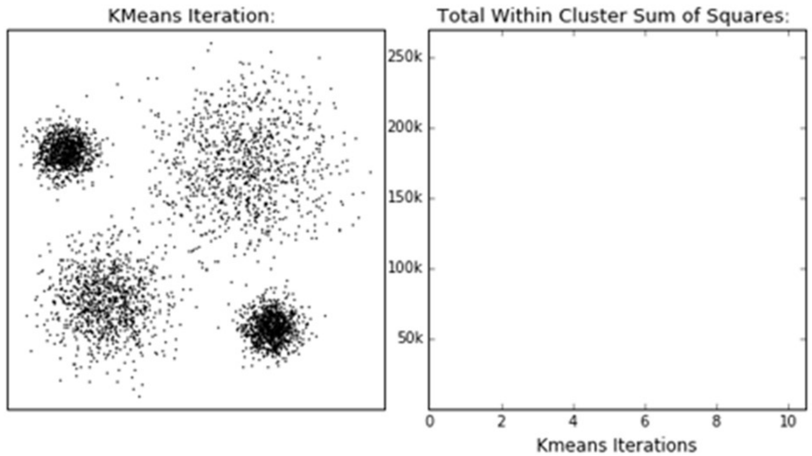


Quantization: Implementation (I)

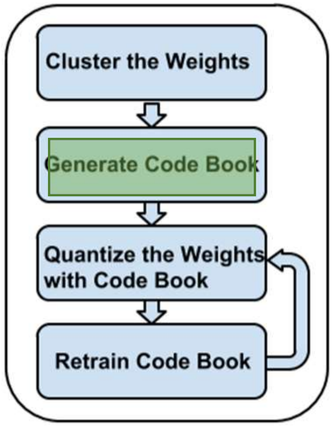


- K-means clustering on weights to find centroids

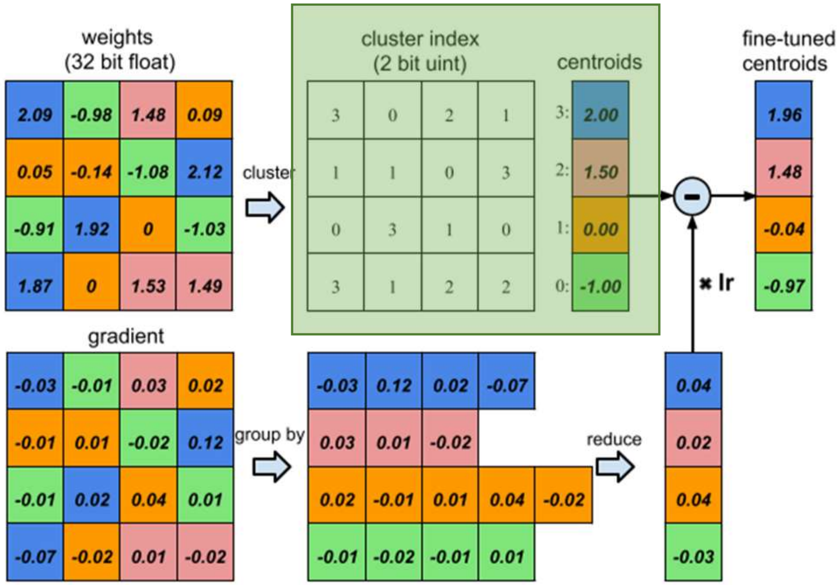




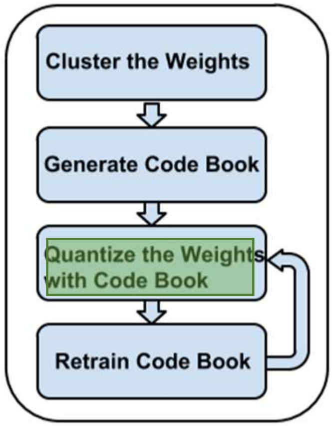
Quantization: Implementation (II)



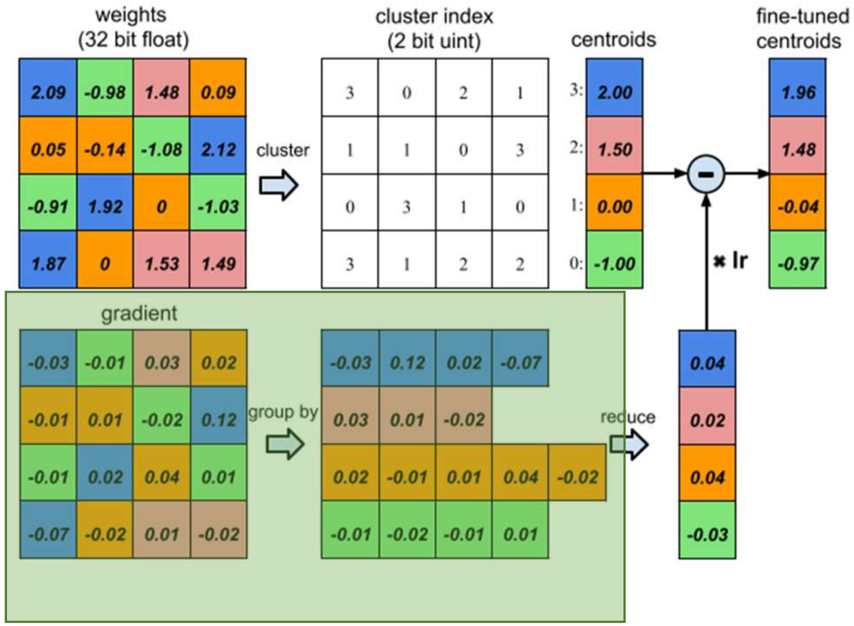
- Use index on centroids/bins as the weight value



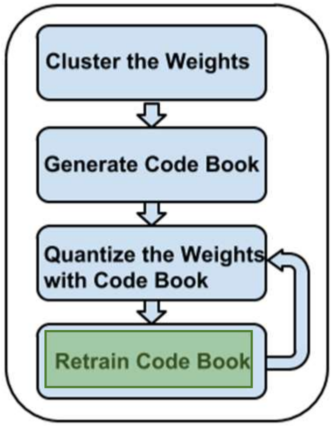
Quantization: Implementation (III)



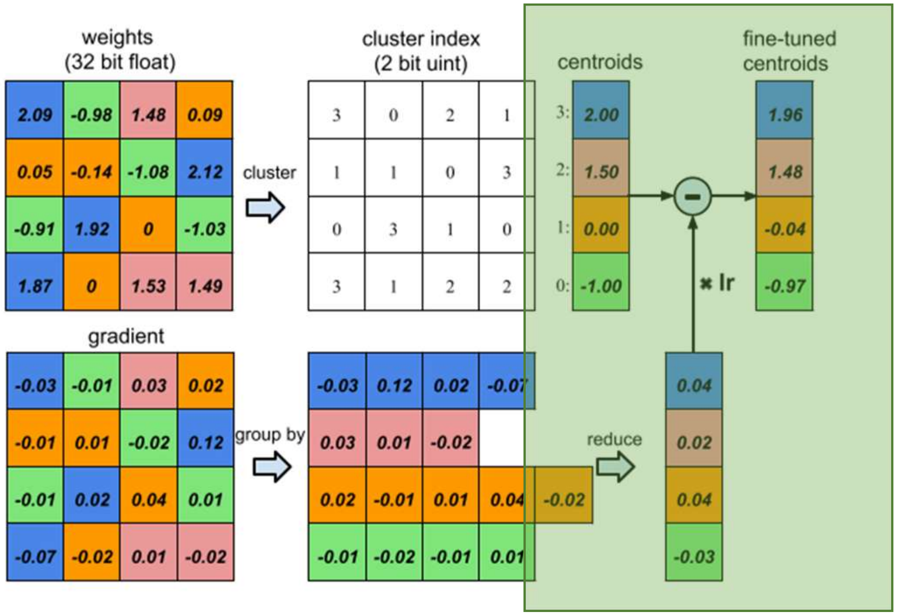
- Use index on centroids/bins as gradient value



Quantization: Implementation (IV)

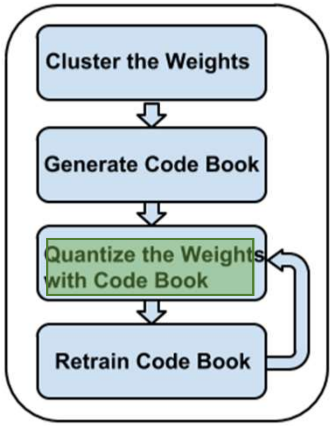


- Backpropagation to fine-tune the centroid

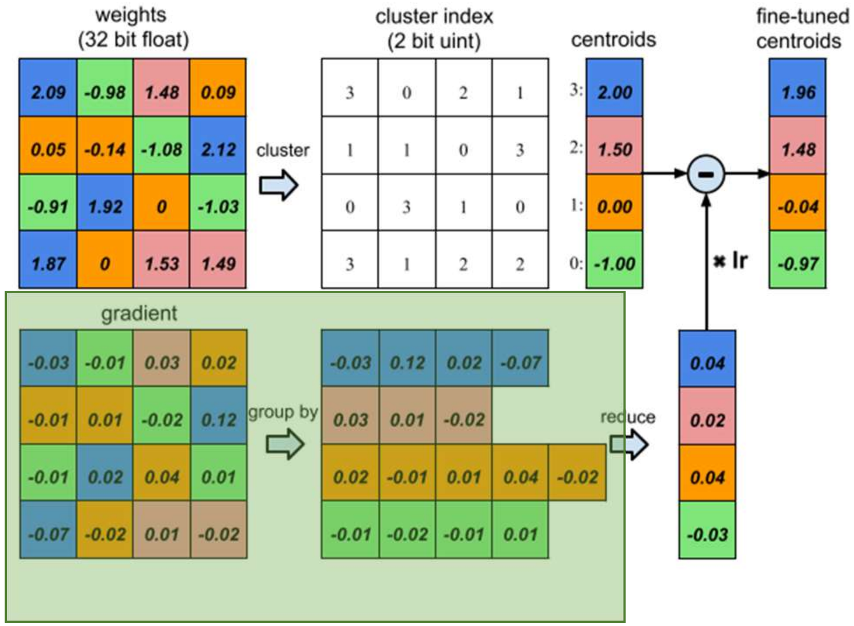


$$\frac{\partial \mathcal{L}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \frac{\partial W_{ij}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \mathbb{1}(I_{ij} = k)$$

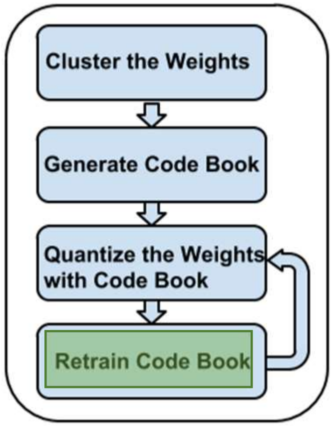
Quantization: Implementation (III)



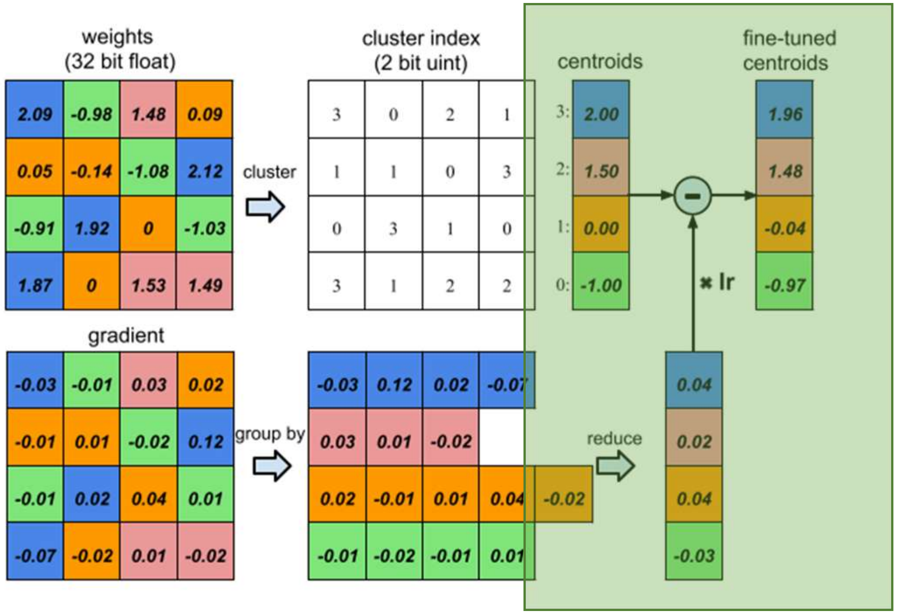
- Use index on centroids/bins as gradient value



Quantization: Implementation (IV)

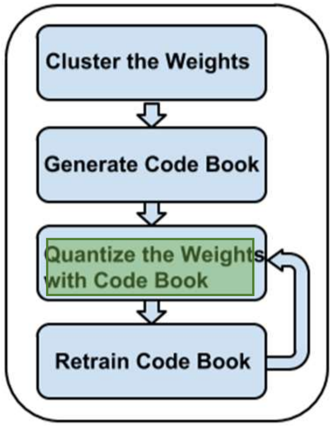


- Backpropagation to fine-tune the centroid

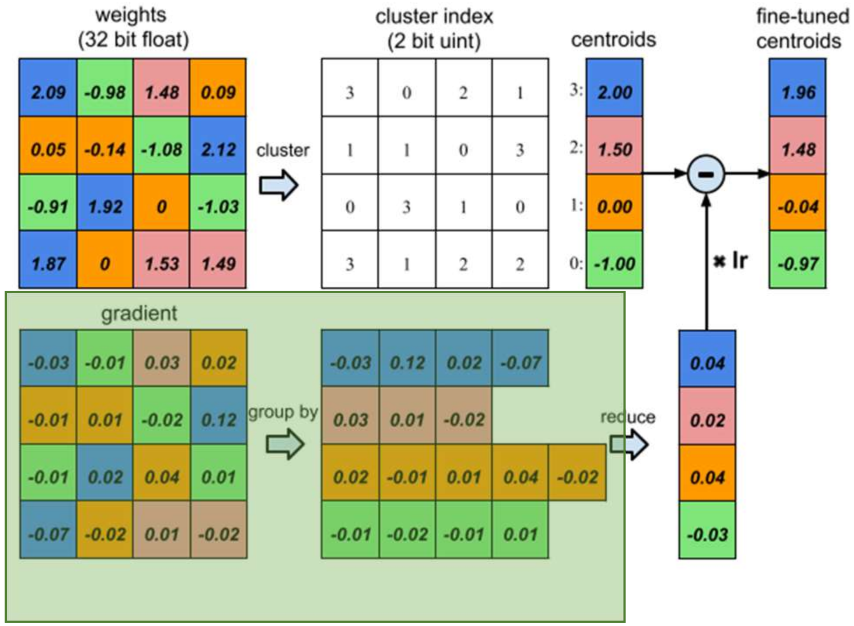


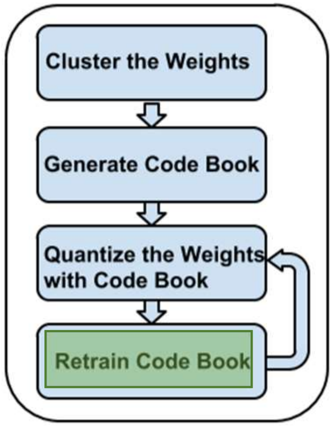
$$\frac{\partial \mathcal{L}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \frac{\partial W_{ij}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \mathbb{1}(I_{ij} = k)$$

Quantization: Implementation (III)



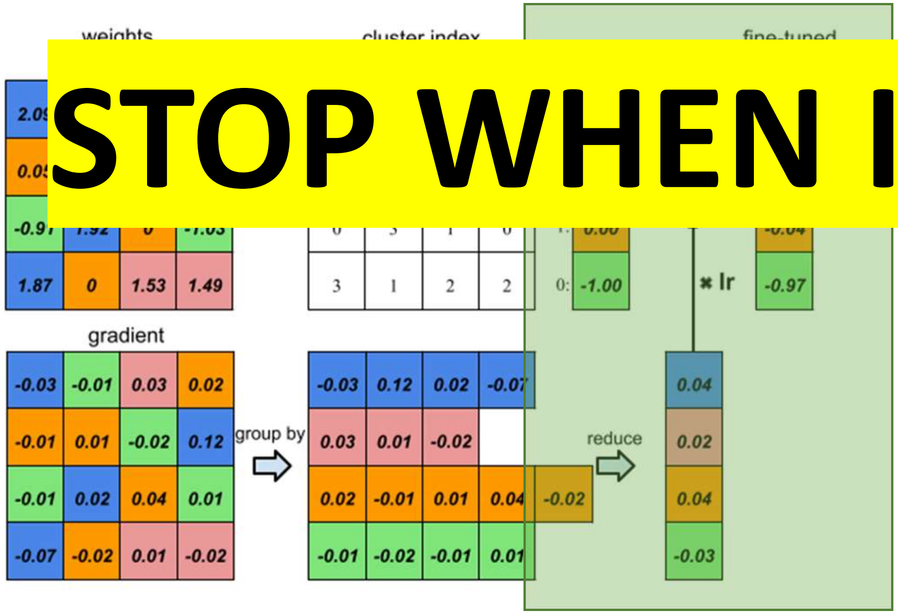
- Use index on centroids/bins as gradient value





Quantization: Implementation (IV)

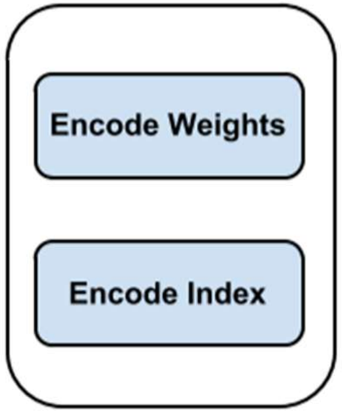
- Backpropagation to fine-tune the centroid



STOP WHEN IT CONVERGES

$$\frac{\partial \mathcal{L}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \frac{\partial W_{ij}}{\partial C_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} \mathbb{1}(I_{ij} = k)$$

Huffman Encoding



- Each weight is represented by the index of the centroid
- A fixed vocabulary (bins/centroids for weights) with indices

LONG FOR SHORT!

SHORT FOR LONG!

Distribution of weights/indices

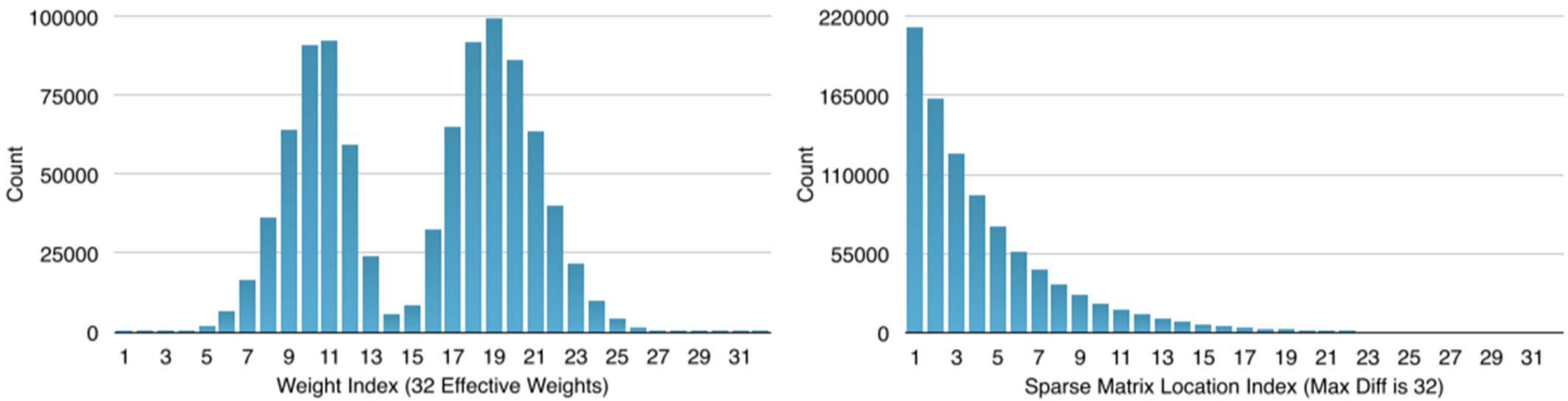
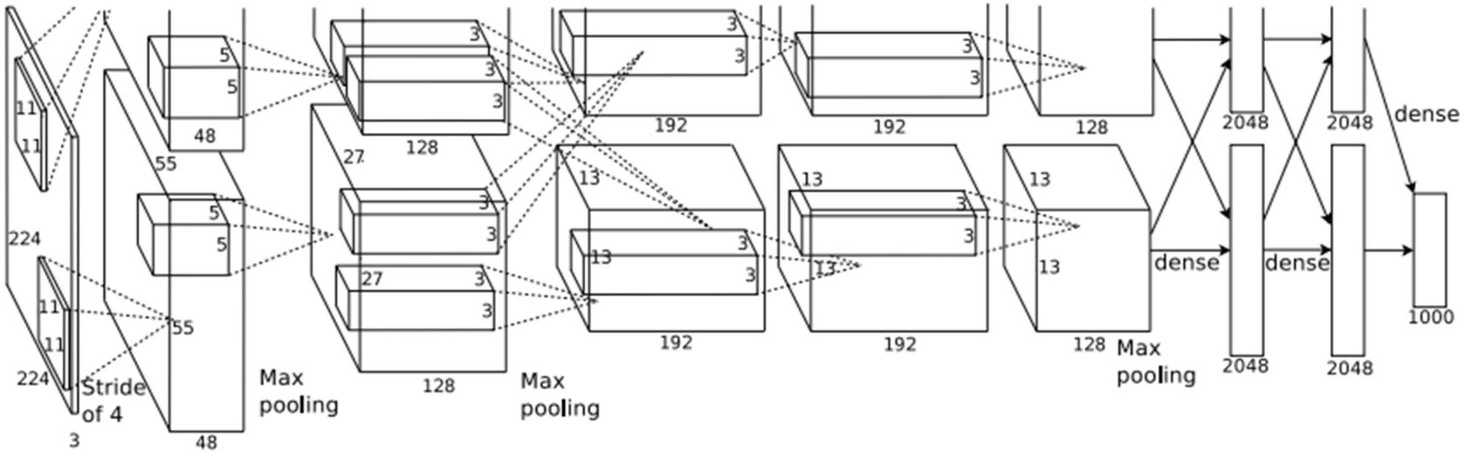


Figure 5: Distribution for weight (Left) and index (Right). The distribution is biased.

Sample Model

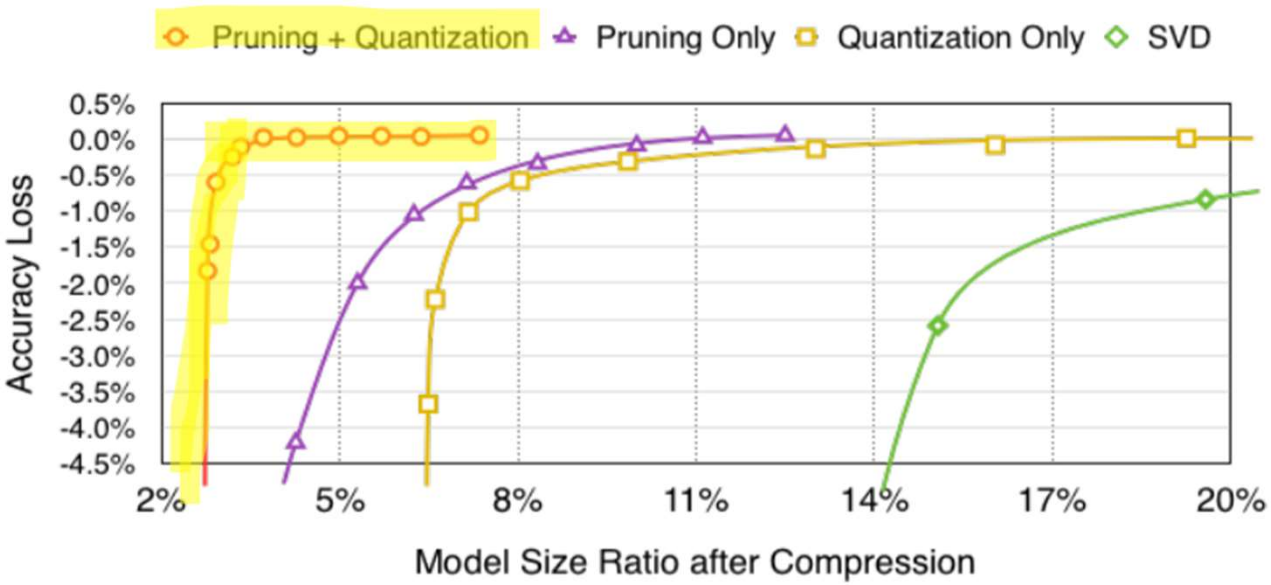
- AlexNet



Sample Model

Results

Results (I): Compression Ratio



- Pruning + Quantization reaches the maximum of **3% model size** without accuracy loss

Results (II): Compression Ratio

Table 4: Compression statistics for AlexNet. P: pruning, Q: quantization, H:Huffman coding.

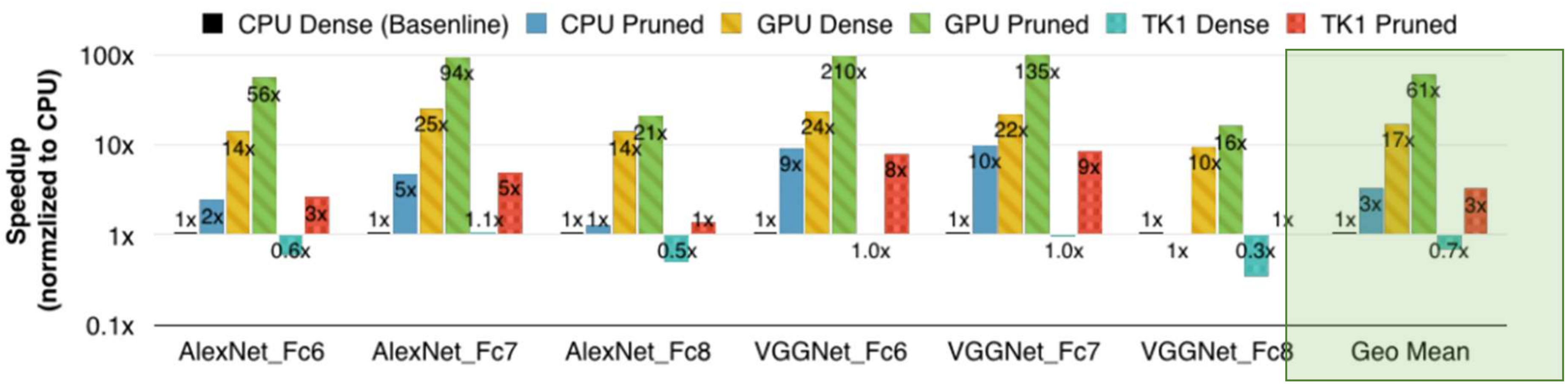
Layer	#Weights	Weights% (P)	Weight bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1	35K	84%	8	6.3	4	1.2	32.6%	20.53%
conv2	307K	38%	8	5.5	4	2.3	14.5%	9.43%
conv3	885K	35%	8	5.1	4	2.6	13.1%	8.44%
conv4	663K	37%	8	5.2	4	2.5	14.1%	9.11%
conv5	442K	37%	8	5.6	4	2.5	14.0%	9.43%
fc6	38M	9%	5	3.9	4	3.2	3.0%	2.39%
fc7	17M	9%	5	3.6	4	3.7	3.0%	2.46%
fc8	4M	25%	5	4	4	3.2	7.3%	5.85%
Total	61M	11%(9×)	5.4	4	4	3.2	3.7% (**27×**)	2.88% (**35×**)

Results (III): Compression Ratio

Table 5: Compression statistics for VGG-16. P: pruning, Q:quantization, H:Huffman coding.

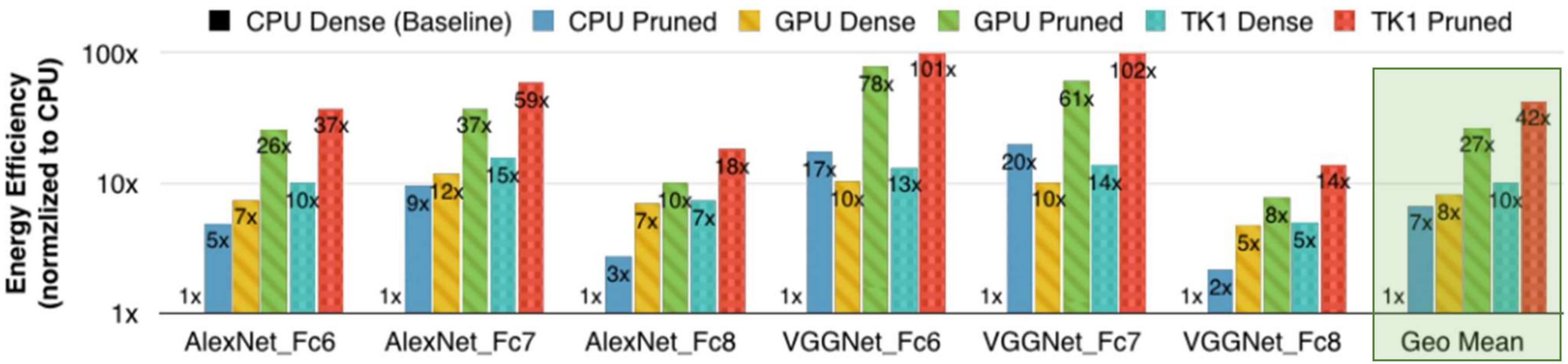
Layer	#Weights	Weights% (P)	Weigh bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1_1	2K	58%	8	6.8	5	1.7	40.0%	29.97%
conv1_2	37K	22%	8	6.5	5	2.6	9.8%	6.99%
conv2_1	74K	34%	8	5.6	5	2.4	14.3%	8.91%
conv2_2	148K	36%	8	5.9	5	2.3	14.7%	9.31%
conv3_1	295K	53%	8	4.8	5	1.8	21.7%	11.15%
conv3_2	590K	24%	8	4.6	5	2.9	9.7%	5.67%
conv3_3	590K	42%	8	4.6	5	2.2	17.0%	8.96%
conv4_1	1M	32%	8	4.6	5	2.6	13.1%	7.29%
conv4_2	2M	27%	8	4.2	5	2.9	10.9%	5.93%
conv4_3	2M	34%	8	4.4	5	2.5	14.0%	7.47%
conv5_1	2M	35%	8	4.7	5	2.5	14.3%	8.00%
conv5_2	2M	29%	8	4.6	5	2.7	11.7%	6.52%
conv5_3	2M	36%	8	4.6	5	2.3	14.8%	7.79%
fc6	103M	4%	5	3.6	5	3.5	1.6%	1.10%
fc7	17M	4%	5	4	5	4.3	1.5%	1.25%
fc8	4M	23%	5	4	5	3.4	7.1%	5.24%
Total	138M	7.5%(13×)	6.4	4.1	5	3.1	3.2% (**31**×)	2.05% (**49**×)

Results (IV): Speedup



- 3x speedup on CPU, 4.2x on mobile GPU and 3.5x on GPU

Results (V): Energy Efficiency



- 7x less energy on CPU, 3.3x less energy on GPU and 4.2x less energy on mobile GPU in average

Results (IV): Quantization Error

#CONV bits / #FC bits	Top-1 Error	Top-5 Error	Top-1 Error Increase	Top-5 Error Increase
32bits / 32bits	42.78%	19.73%	-	-
8 bits / 5 bits	42.78%	19.70%	0.00%	-0.03%
8 bits / 4 bits	42.79%	19.73%	0.01%	0.00%
4 bits / 2 bits	44.77%	22.33%	1.99%	2.60%

- Quantization error depends on the number of bits needed to represent each centroid bin
- Critical to find the sweet spot or choose depending on the requirement

Strengths & Weaknesses

Strengths

- First paper to use Huffman encoding to code weight books and indices
- First paper to use adaptive quantization
- First paper to implement the network-wise pruning (the method was proposed in another paper from the same year from the same author)
- Applicable to not only mobile platforms but also general platforms to reduce the energy/space consumption
- Clear demonstration

Weaknesses

- Latency of computation is ignored -> CNN is essentially the bottleneck
- Performance methodology is biased
- Unstructured sparsity could hinder parallel computation
- The problem intentionally chose the neural network architecture famous for being overparametrized and sparsifiable (AlexNet)
- One must first train a densely connected DNN to operate on this network

Related Works & Current Development

DNN Compression Is a HOT HOT Topic

- Parameter pruning and quantization
    - Redundancy reduction
- Low-rank factorization
    - Low-rank decomposition/approximation SVD
- Transferred/compact convolutional filters
    - Structured convolutional filters
- Knowledge distillation
    - Train a smaller network based on the larger network

Source: A Survey of Model Compression and Acceleration for Deep Neural Networks – Cheng et al.

Pruning

Sparsification

- S. J. Hanson on ***Comparing bi*** ***al network construction with back-propagation*** introduce
- Srinivas and Babu et al. on ***De*** ***ter pruning for deep neural networks*** introduced pruning
- Han et al. on ***Learning both v*** ***nections for efficient neural networks*** introduced pruning etwork
- Chen et al. on ***Compre*** ***e hashing trick*** introduced pruning with eights for parameter sharing
- Lebedev et al. on ***Fast convne*** ***-wise brain damage*** introduced pruning while trainin ying sparsity constraints

Quantization

Approximation

- Gong et al. **Compressing deep convolutional networks using vector quantization** and Wu et al. applied k-means
- Vanhoucke et al. applied 8-bit quantization
- Han et al. applied Huffman coding to the quantized the link weights
- Choi et al. applied Hessian weight to measure the importance of weights
- BinaryConnect, BinaryNet and XNOR attempt to use only 1-bit representation
- Hou et al., Lin et al. and Cai et al. attempt to adjust the loss of precision due to binarization

Overparameterization of DNN

Predicting Parameters in Deep Learning

Misha Denil[1] Babak Shakibi[2] Laurent Dinh[3]
Marc'Aurelio Ranzato[4] Nando de Freitas[1,2]

DNNs ARE UNNECESSARILY TOO LARGE

[4]Facebook Inc., USA
{misha.denil,nando.de.freitas}@cs.ox.ac.uk
laurent.dinh@umontreal.ca
ranzato@fb.com

Abstract

We demonstrate that there is significant redundancy in the parameterization of several deep learning models. Given only a few weight values for each feature it is possible to accurately predict the remaining values. Moreover, we show that not only can the parameter values be predicted, but many of them need not be learned at all. We train several different architectures by learning only a small number of weights and predicting the rest. In the best case we are able to predict more than 95% of the weights of a network without any drop in accuracy.

Source: Denil et al., Predicting Parameters in Deep Learning, NISP 2013

Lottery Ticket Hypothesis

THE LOTTERY TICKET HYPOTHESIS:
FINDING SPARSE, TRAINABLE NEURAL NETWORKS

Jonathan Frankle
MIT CSAIL
jfrankle@csail.mit.edu

Michael Carbin
MIT CSAIL
mcarbin@csail.mit.edu

EXISTENCE OF SPARSE SUBNETS MAY BE GUARANTEED

Neural network pruning techniques can reduce the parameter counts of trained networks by over 90%, decreasing storage requirements and improving computational performance of inference without compromising accuracy. However, contemporary experience is that the sparse architectures produced by pruning are difficult to train from the start, which would similarly improve training performance.

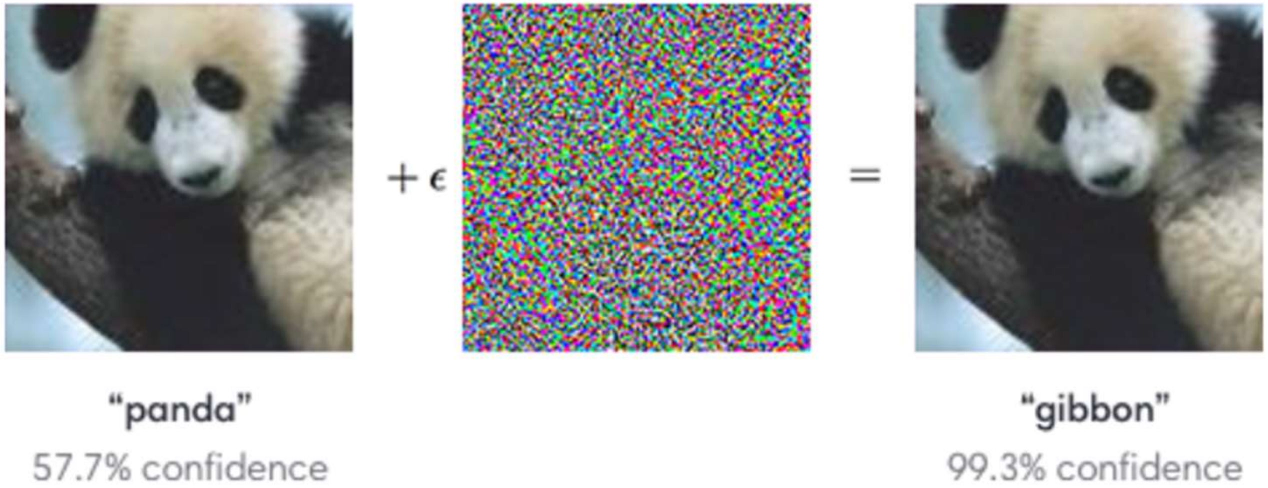
We find that a standard pruning technique naturally uncovers subnetworks whose initializations made them capable of training effectively. Based on these results, we articulate the *lottery ticket hypothesis*: dense, randomly-initialized, feed-forward networks contain subnetworks (*winning tickets*) that—when trained in isolation—reach test accuracy comparable to the original network in a similar number of iterations. The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective.

We present an algorithm to identify winning tickets and a series of experiments that support the lottery ticket hypothesis and the importance of these fortuitous initializations. We consistently find winning tickets that are less than 10-20% of the size of several fully-connected and convolutional feed-forward architectures for MNIST and CIFAR10. Above this size, the winning tickets that we find learn faster than the original network and reach higher test accuracy.

Source: Frankle et al., The Loyalty Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, ICLR 2019

Robustness

- DNNs are often **vulnerable** to intentionally perturbed data



Panda or Gibbon? That is a big question!

Source: https://openai.com/blog/adversarial-example-research/

Robustness and Generalization

Robustness and Generalization

Huan Xu HUAN.XU@MAIL.UTEXAS.EDU
Department of Electrical and Computer Engineering
the University of Texas at Austin, TX, USA

ROBUST MODELS ARE MORE GENERALIZABLE

Editor: n/a

Abstract

We derive generalization bounds for learning algorithms based on their robustness: the property that if a testing sample is "similar" to a training sample, then the testing error is close to the training error. This provides a novel approach, different from the complexity or stability arguments, to study generalization of learning algorithms. We further show that a weak notion of robustness is both sufficient and necessary for generalizability, which implies that robustness is a fundamental property for learning algorithms to work.

Source: Xu et al., Robustness and Generalization, ArXiv 2010

Robustness-Redundancy Hypothesis

ROBUSTNESS AND/OR REDUNDANCY EMERGE IN OVERPARAMETRIZED DEEP NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

MODEL SIZE DOES NOT GUARANTEE ROBUSTNESS

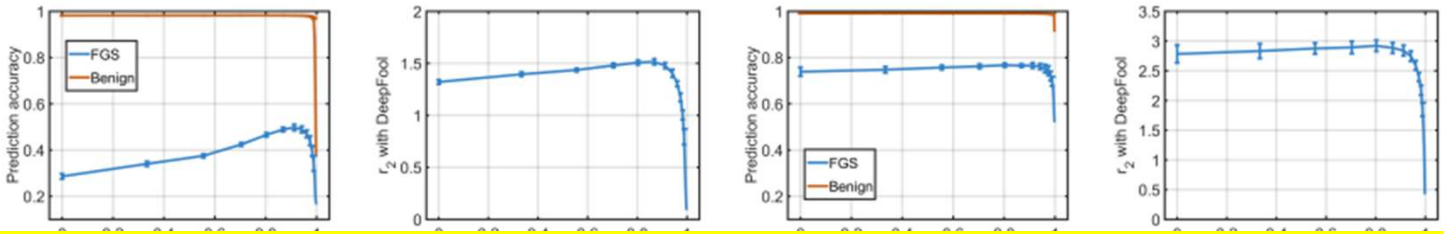
Deep neural networks (DNNs) perform well on a variety of tasks despite the fact

HOWEVER, THEY SEEM TO INCREASE ALONGSIDE EACH OTHER

networks to the task at hand and avoiding overfitting (Arora et al., 2018; Zhou et al., 2018). In this paper, we provide new empirical evidence that supports this hypothesis, identifying two independent mechanisms that emerge when the network's width is increased: robustness (having units that can be removed without affecting accuracy) and redundancy (having units with similar activity). In a series of experiments with AlexNet, ResNet and Inception networks in the CIFAR-10 and ImageNet datasets, and also using shallow networks with synthetic data, we show that DNNs consistently increase either their robustness, their redundancy, or both at greater widths for a comprehensive set of hyperparameters. These results suggest that networks in the deep learning regime adjust their effective capacity by developing either robustness or redundancy.

Source: Anonymous, Robustness and/or Redundancy Emerge in Overparametrized Deep Neural Networks, ArXiv 2020

Accuracy of Pruned DNN over Adversarial Attacks



ROBUSTNESS DROPS ONCE PRUNING IS DONE TOO MUCH

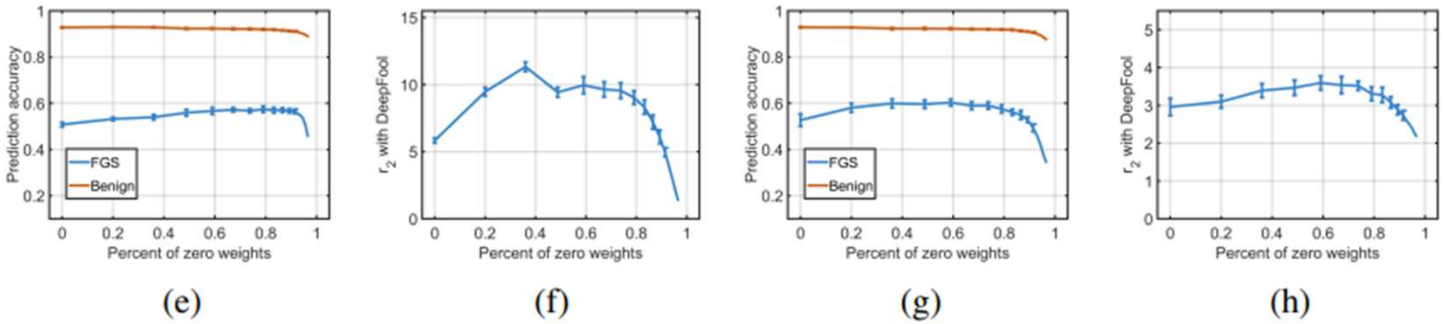
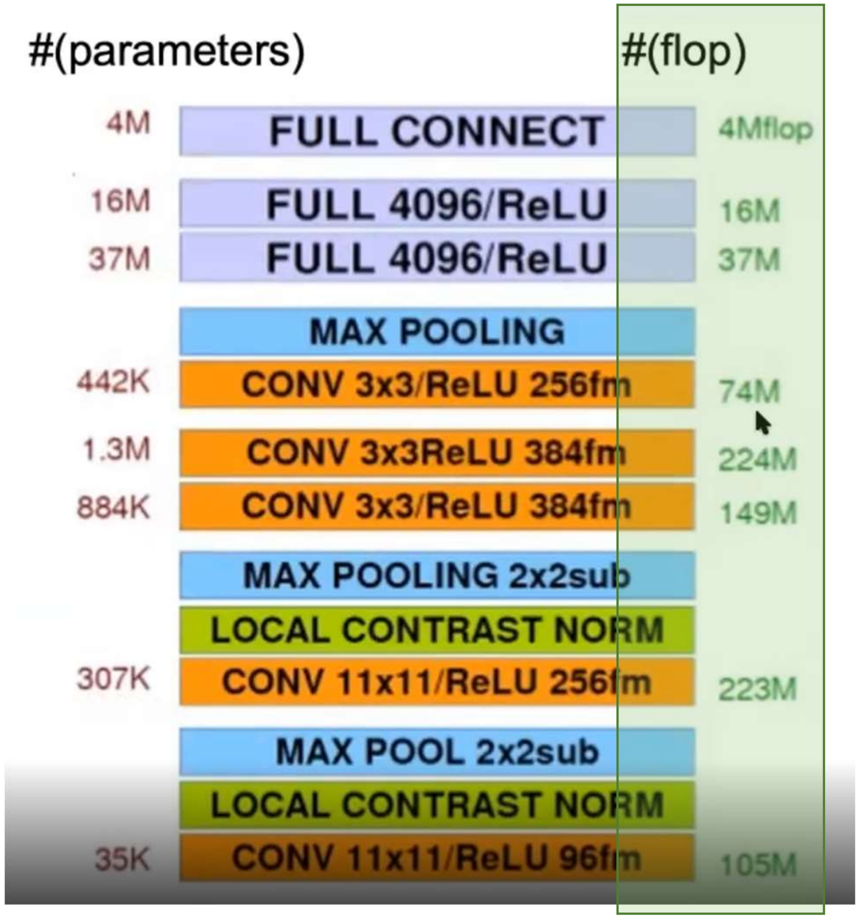


Figure 2: The robustness of nonlinear DNNs with varying weight sparsity. (a)-(b): LeNet-300-100, (c)-(d): LeNet-5, (e)-(f): the VGG-like network, (g)-(h): ResNet-32.

Discussion

AlexNet Dimension



Source: Lecture Slide from Deep Learning for Autonomous Driving

Discussion (I): End-to-End Performance

- Latency/throughput is not mentioned by the paper
  - Critical for real-time processing as was targeted by the paper
- Speedup is actually… **not true**… (in my opinion)
  - Only **densely connected layers** are measured to have a significant speedup
  - Overheads are mostly in **CNN layers**
  - The overall throughput **does not** increase if the bottleneck layer is not boosted much (and so is latency)
  - How do you think that it would be fairer methodology to measure the speedup? What would you expect really from throughput by using this approach? What kind of benchmarks would make sense?

Discussion (II): Scalability/Applicability

- This is not fundamentally solving the issue of memory wall
- File sizes would **eventually increase** with current trend of increasing large/deep neural networks (e.g., GPT3)
- Same memory wall would still occur since larger models are coming in
- A lot of larger networks are becoming less sparse -> fundamental assumption in pruning
- Quantization has fundamentally inevitable information loss
- Would **near-data processing** be a better candidate for scalability?

Discussion (III): Unstructured Sparsity and Overheads

- Pruning makes DNN **unstructuredly sparse**
- Existing accelerators become inefficient because it must still perform lots of unnecessary operations on zero points in the sparse matrix
- Any remedy for it?
- Furthermore, pruning has proven to be a very expensive operation
  - (both from literature review and first-hand experience)
  - Any idea if we could create a hardware accelerator to boost it?

Discussion (IV): Quantization

- Quantization often uses **fixed bits** for each value
  - High precision requires more bits per value
- How could one improve the precision while using minimal bits per value?
- How can one enable a hardware optimization to reduce the access time for quantized values?

Discussion (V): Tradeoff between Robustness and Compactness

- As shown earlier, pruning could **harm** the robustness after a threshold
- A metrics to compensate for both accuracy loss and robustness loss is urgently needed
- Under what metrics should one prune the network?
  - Accuracy loss over the original data?
  - Accuracy loss over the adversarial data?
  - Both?

Discussions (VI): Overparameterization

- More evidences are showing that overparameterization has mysterious relationships with generalization
  - Even more with current interpretation of double gradient descent phenomenon occurring in a largely overparametrized models
- **Trade-off** between generalization and compactness must be made
  - How would you think of doing it?

Discussions (I): Sparsification

- Current solutions are only able to sparsify a neural network **after** it has been densely trained
- Can you think of any solution to **directly** prune a network without having to train the dense one first?

Backup Slides

Source:

EIE Accelerator

(2) becomes

$$b_i = ReLU\left(\sum_{j \in X_i \cap Y} S[I_{ij}]a_j\right)$$

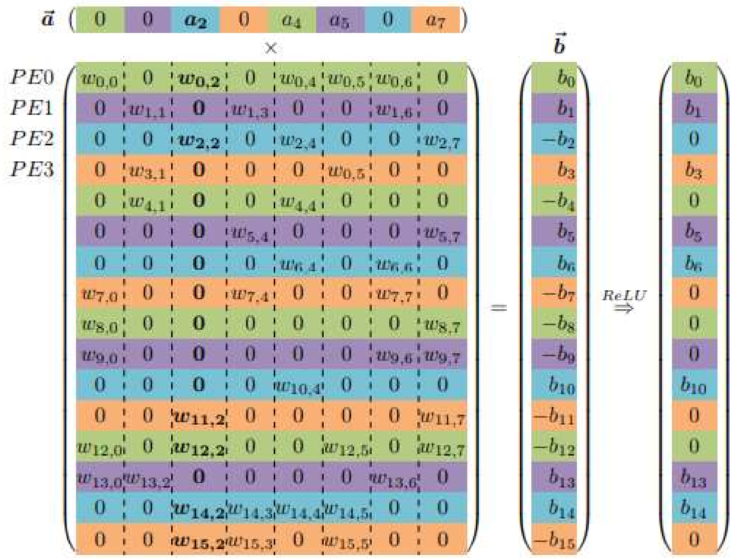


Figure 2. Matrix W and vectors a and b are interleaved over 4 PEs. Elements of the same color are stored in the same PE.

Virtual Weight	$W_{0,0}$	$W_{8,0}$	$W_{12,0}$	$W_{4,1}$	$W_{0,2}$	$W_{12,2}$	$W_{0,4}$	$W_{4,4}$	$W_{0,5}$	$W_{12,5}$	$W_{0,6}$	$W_{8,7}$	$W_{12,7}$
Relative Row Index	0	1	0	1	0	2	0	0	0	2	0	2	0
Column Pointer	0	3	4	6	6	8	10	11	13				

Figure 3. Memory layout for the relative indexed, indirect weighted and interleaved CSC format, corresponding to PE_0 in Figure 2.

Source:

EIE HW Architecture

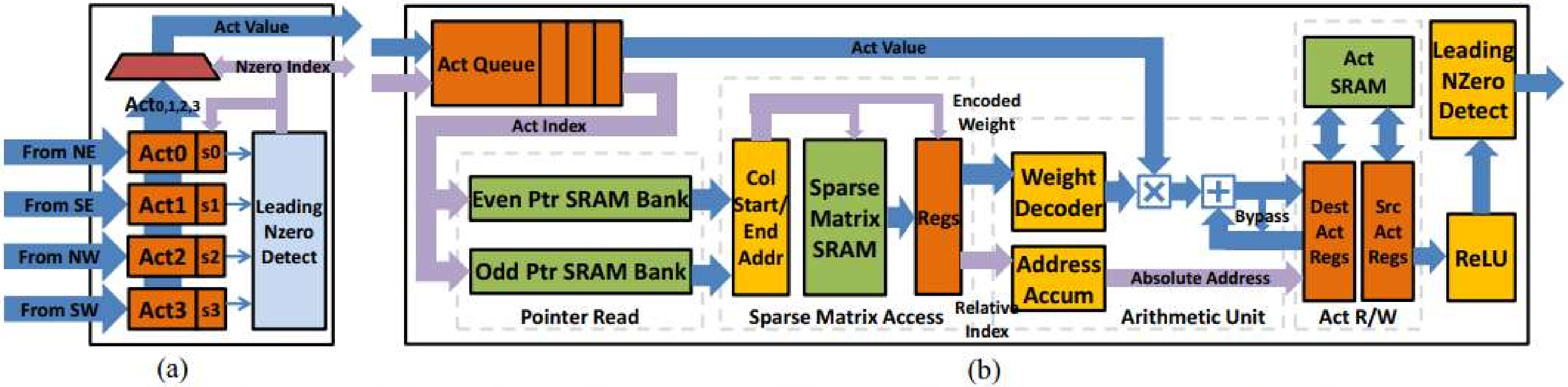


Figure 4. (a) The architecture of Leading Non-zero Detection Node. (b) The architecture of Processing Element.

Adversarial Training + Pruning

$$\min_{\boldsymbol{\theta}} \quad \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\boldsymbol{\Delta}} L(\boldsymbol{\theta}, x+\delta, y)\right]$$

$$\begin{aligned}
\min_{\boldsymbol{\theta}_i} \quad & \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\boldsymbol{\Delta}} L(\boldsymbol{\theta}, x+\delta, y)\right] + \sum_{i=1}^{N} g_i(\mathbf{z}_i), \\
s.t. \quad & \boldsymbol{\theta}_i = \mathbf{z}_i,\ i = 1, \ldots, N.
\end{aligned}$$

$$g_i(\boldsymbol{\theta}_i) = \begin{cases} 0 & \text{if } \boldsymbol{\theta}_i \in S_i \\ +\infty & \text{otherwise} \end{cases}$$

Lagrangian Multiplier

$$\mathcal{L}(\{\boldsymbol{\theta}_i\}, \{\mathbf{z}_i\}, \{\mathbf{u}_i\}) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta \in \boldsymbol{\Delta}} L(\boldsymbol{\theta}, x + \delta, y) \right]$$
$$+ \sum_{i=1}^{N} g_i(\mathbf{z}_i) + \sum_{i=1}^{N} \mathbf{u}_i^T (\boldsymbol{\theta}_i - \mathbf{z}_i) + \frac{\rho}{2} \sum_{i=1}^{N} \|\boldsymbol{\theta}_i - \mathbf{z}_i\|_2^2,$$

Source: