Quantization in Neural Networks
Advantages and limitations

Emilio Paolini, PhD Student

Outline

- Introduction
- Post-training quantization
- Quantization-Aware training
- State-of-the-art
- Hands-on

Introduction

A gentleman otter in a 19th century portrait

Image generated with Stable Diffusion

- 1943 - Pitts and McCulloch created a computer model based on the neural networks of the human brain
- 1960s - Back-propagation model basics
- 1970s - AI winter: promises that couldn't be kept
- 1980s - Convolution emerges, LeNet performs Digit Recognition
- 1988-90s - Second AI winter: the "immediate" potential of AI was exaggerated. AI = pseudoscience status
- 2000-2010 - Big data introduction, first big datasets (ImageNet)
- 2010-2020 - Computational power, GAN appears
- Present - DL boom. AI is pervasive and influences the creation of new business models

Factors that led to DL explosion

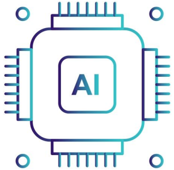
Since 2012 investment in AI has grow exponentially global startup funding:

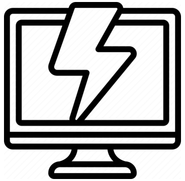
- $670 million in 2011
- $36 billion U.S. dollars in 2020
- $77 billion in 2021

AI systems have been around since the 1950s, so why are we suddenly seeing breakthroughs in so many diverse areas?

Three main factors:

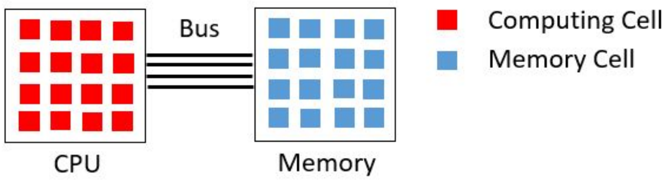
- Enormously increased data (5G, IoT)
- Significantly improved algorithms and models
- Higher computing power



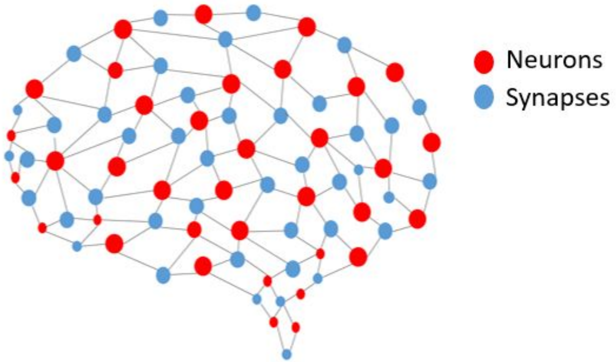


Modern Deep Learning Issues (1/2)

- Von Neumann vs Neural Network (NN) architecture
  - The main source of latency and power consumption comes from data movement even in very optimized architectures
  - Computing units and memory elements are physically separate chips in computers



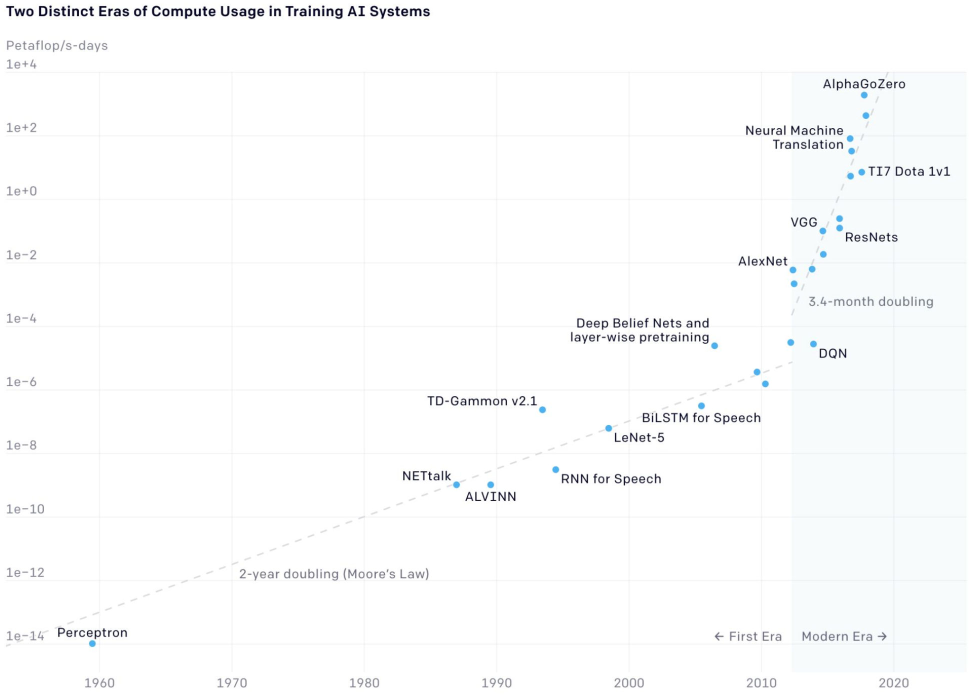
(a) Von Neumann Computing System



(b) Brain Computing System

Modern Deep Learning Issues (2/2)

- Core speeds have stopped to grow because of physical limits in power dissipation



OpenAI part “AI and Compute”
https://openai.com/blog/ai-and-compute/

Possible solutions

HARDWARE

Goal: change the underlying hardware

- Specialized digital electronic architecture (e.g., tensor core)
- Analog electronic circuits
- Photonic hardware

SOFTWARE

Goal: reduce the size of the model

- Pruning
- Knowledge distillation
- **Quantization**

Quantization

Def. It is the process of constraining an input from a continuous or otherwise large set of values (such as the real numbers) to a discrete set (such as the integers) (Wikipedia)

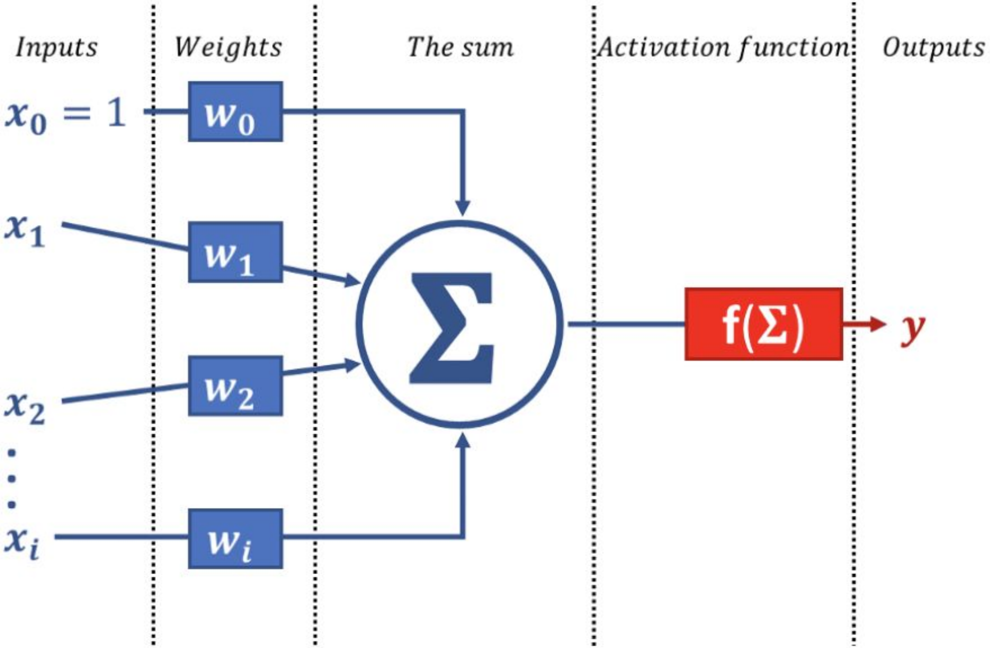
Our case

Computation in Neural Networks (NNs) use Floating Point numbers (32 bits)

Goal: performing computations and storing tensors at lower bitwidths

Floating Point (32 bits) → Lower bitwidths

Why Quantization in NNs? (1/4)



Artificial Neuron

Network	Model size (MB)	GFLOPS
AlexNet*	233	0.7
VGG-16*	528	15.5
VGG-19*	548	19.6
ResNet-50*	98	3.9
ResNet-101*	170	7.6
ResNet-152*	230	11.3
GoogleNet#	27	1.6
InceptionV3#	89	6
MobileNet#	38	0.58
SequeezeNet#	30	0.84

*: Characterization and Benchmarking of Deep Learning, Natalia Vassilieva
#: https://github.com/albanie/convnet-burden

Why Quantization in NNs? (2/4)

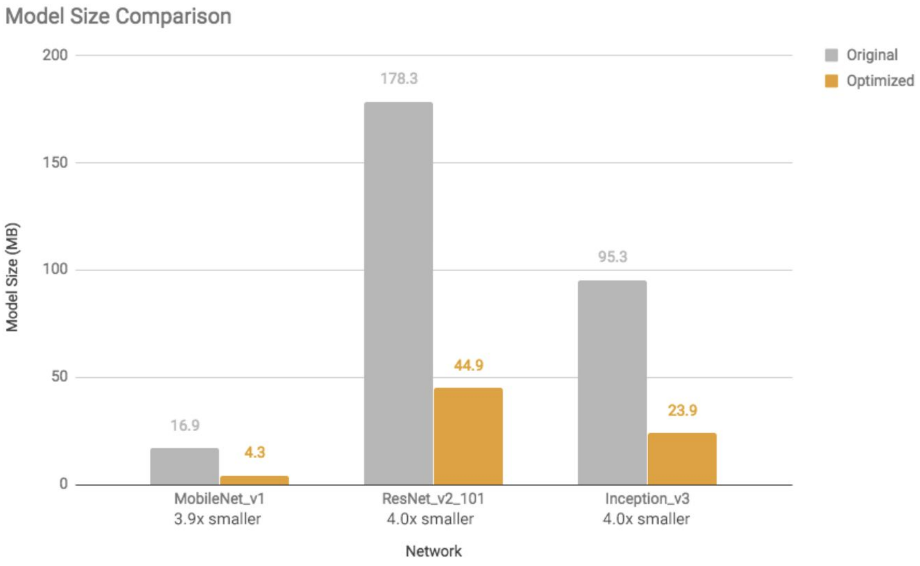
- Reducing the number of bits for representing the neural network's parameters results in less memory storage
- Using the lower-bit quantized data requires less data movement, which reduces memory bandwidth and saves significant energy
- Lower-precision mathematical operations, such as an 8-bit integer multiply versus a 32-bit floating point multiply, consume less energy and increase compute efficiency, thus reducing power consumption

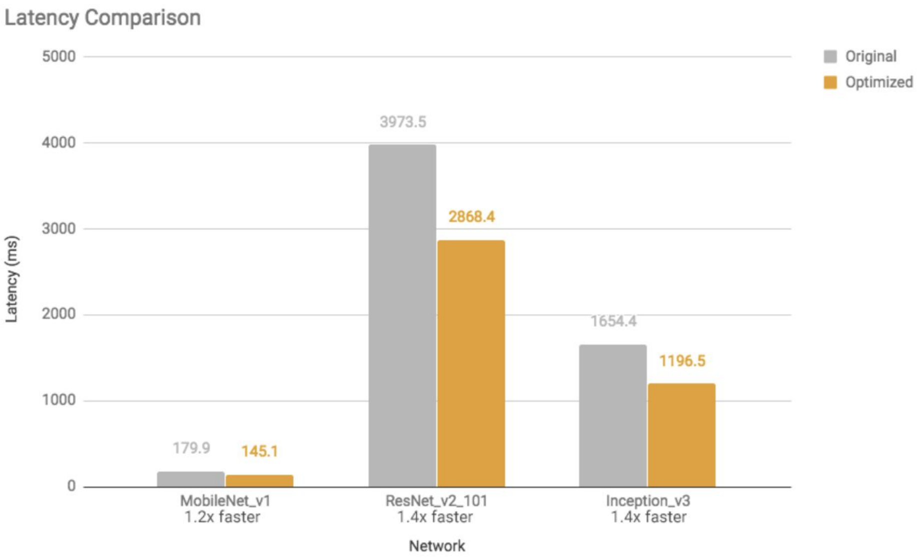
Why Quantization in NNs? (3/4)

- Three components that can be quantized in a NN
  - Weights
  - Activations
  - Gradients
- By quantizing weights and activations, we can achieve smaller model size
- Quantization of gradients can be used for example where the training environment is distributed to save communication cost
- Generally it is more difficult to quantize the gradients than quantizing weights and activations since high-precision gradients are needed to perform backpropagation

Why Quantization in NNs? (4/4)

- Quantization converts floating-point arithmetic of neural networks into low precision arithmetic and makes real time inference possible on mobile phones as well as benefits cloud applications





Quantization drawbacks

- Direct quantization of NNs architectures results in a severe loss of accuracy (see later in lab session)
- Quantization is an approximation
    - The closer the approximation, the less performance decay one can expect
    - Quantize everything to *float16*: cut the memory in half, probably no accuracy loss
    - But won't really gain speedup
    - Quantizing with *int8* can result in much faster inference
    - But the performance will probably be worse. Extreme scenario: it won't even work

Quantization in practice

- How to quantize NN models and reduce accuracy loss?
- Avoid Direct Quantization!

Post-Training Quantization (PTQ)	Quantization-Aware Training (QAT)
How: train the model using *float32*, then quantize it • It can result in accuracy loss	**How**: quantize model during training, trying to compensate for the quantization-related errors • Best accuracy results

Post-Training Quantization

- Fastest and easiest way to get a quantized model
- It can lead to significant accuracy deviation in some cases
- Several PTQ options:
  - Dynamic range quantization
    - 4x smaller, 2x-3x speedup
  - Full integer quantization
    - 4x smaller, 3x+ speedup
  - Float16 quantization
    - 2x smaller, GPU acceleration

PTQ - Dynamic Range Quantization

- It provides reduced memory usage and faster computation without having to provide a representative dataset for calibration
- Statically quantize the weights from floating point to 8-bits of precision and dynamically quantize the activations at inference
- Activations are always stored in float 32
- But they are converted to 8-bit integers while processing and back to floating point after the processing is done
- Provides latencies close to fully fixed-point inferences

PTQ - Full Integer Quantization

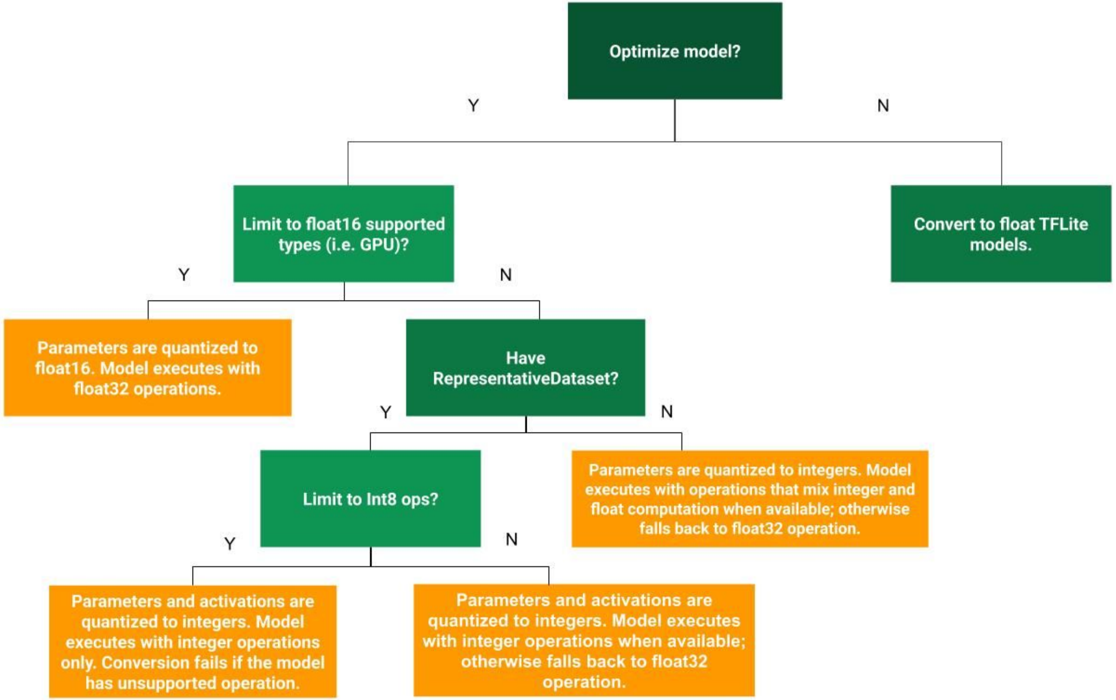
- Further latency improvements, reductions in peak memory usage, and compatibility with integer-only hardware devices or accelerators by making sure all model math is integer quantized
- Statically quantize all weights and activations of the model to 8 bit integers
- Need to calibrate or estimate the range, i.e, (min, max) of all floating-point tensors in the model
  - Constant tensors: weights, biases
  - Variable tensors: model input, activations (outputs of intermediate layers) and model output

- Cannot be calibrated unless a representative dataset is used to estimate the range
- Dataset can be a subset of training/test

PTQ - Float16 quantization

- Reduce the size of a floating point model by quantizing the weights to float16
- Reduce model size by up to half
- Cause minimal loss in accuracy
- Supports some hardware which can operate directly on float16 data, resulting in faster execution than float32 computations
- Disadvantages
  - Does not reduce latency as much as a quantization to fixed point math
  - By default, a float16 quantized model will "dequantize" the weights values to float32 when run on the CPU
    - CPUs upscale float16 back to float32 before processing

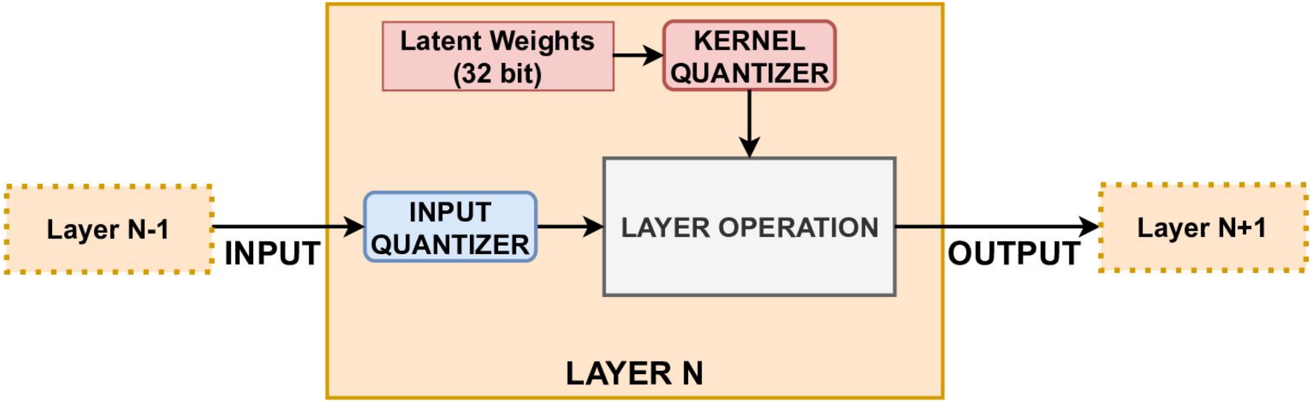
Post-Training Quantization



Quantization-Aware Training

- Quantization during training: take the effect of quantization loss into account during training
- Typically provides higher accuracies as compared to PTQ
- QAT is achieved by adding fake quantization nodes
- Simulates low precision behavior in the forward pass, while the backward pass remains in float32

Quantization-Aware Training



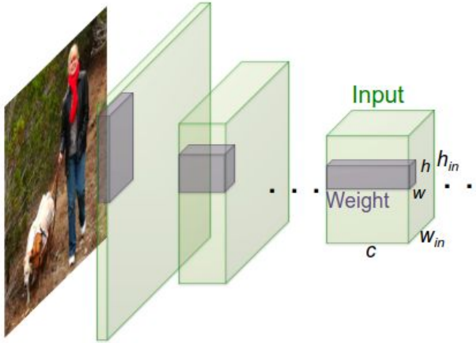
- Quantizer: defines the way of transforming a full precision input to a quantized output
- All the weight adjustments during training are made while “aware” of the fact that the model will ultimately be quantized

PTQ vs QAT

Model	Floating-point baseline model	QAT model	Delta	Post-training full integer quantized model
MobileNet v1 1.0 224	71.03%	71.06%	0.04%	69.57%
MobileNet v2 1.0 224	70.77%	70.01%	-1.07%	70.2%
ResNet v1 50	76.3%	76.1%	-0.26%	75.95%

XNOR-Net

- Both weights and input activations of convolutional layers are binarized



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+, −, ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+, −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR, bitcount	~32x	~58x	%44.2

DoReFa-Net

- Further extends the method of binarized neural networks to create a NN that has arbitrary bitwidths for weights and activations

W	A	G	Training Complexity	Inference Complexity	Storage Relative Size	AlexNet Accuracy
1	1	32	-	1	1	0.279 (BNN)
1	1	32	-	1	1	0.442 (XNOR-Net)
1	1	32	-	1	1	0.436
1	2	4	6	2	1	0.471
1	2	4	6	2	1	0.507 (initialized)
1	2	8	10	2	1	0.456
1	2	32	-	2	1	0.498 (initialized)
1	4	32	-	4	1	0.530 (initialized)
32	32	32	-	-	32	0.559

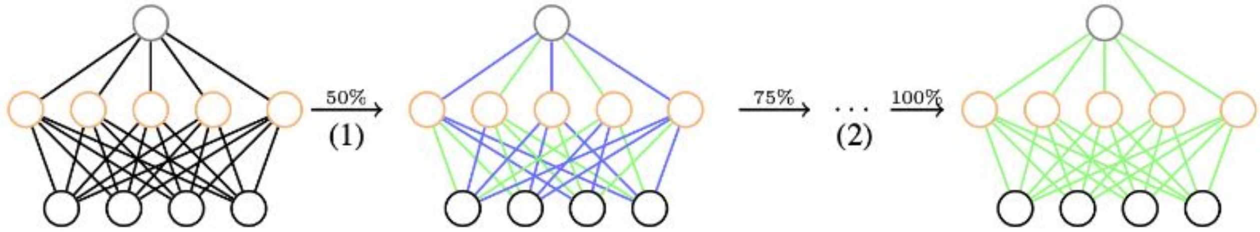
Incremental Network Quantization

- Method to efficiently convert any pre-trained full-precision NN into a low-precision version whose weights are constrained to be either powers of two or zero

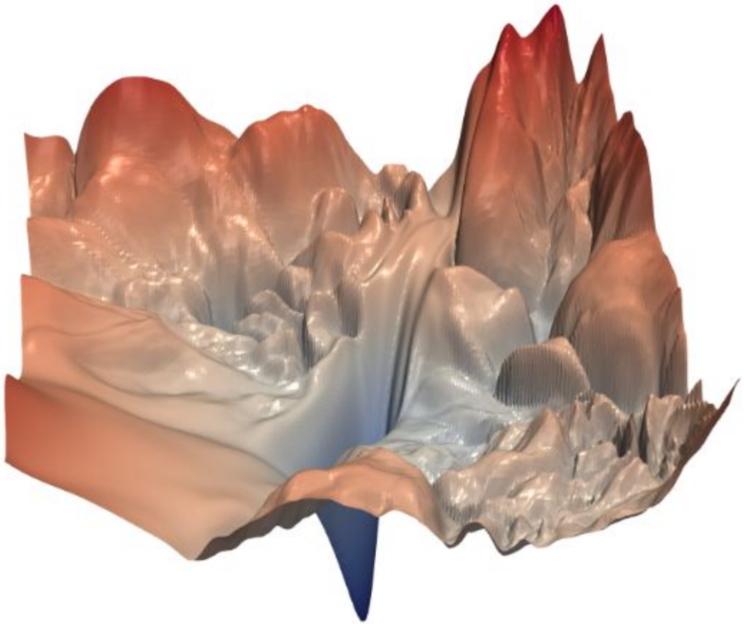
- Three operations
  - Weight partitioning
  - Group-wise quantization
  - Re-training

Incremental Network Quantization

- Weights in the first group are quantized to be either powers of two or zero by a variable-length encoding method, forming a low-precision base for the original model
- Weights in the second group are re-trained while keeping the weights in the first group fixed, in order to compensate the accuracy loss resulted from the quantization
- These operations are repeated on the weights of the second group in an iterative manner until all the weights are quantized



Dangers of quantization



Visualizing the Loss Landscape of Neural Nets, Hao Li et al

- Loss landscape of a ResNet56
- The independent variables represent the weights of the model, while the the dependent variable is the loss
- Changing the weights just a bit, the differences in loss can be enormous

There is no guarantee that it won't totally mess up the model

Hands-on

https://github.com/emiliopaolini/ICTP_2022