DL-specific deployment

Efficient model inference

Lecture plan

- Scope
- Ways to optimize models
- Efficient architectures
- Reducing number of model’s parameters
- Get the most out of training
- Relations with inference engines

Scope

- Task
- Data collection
- Architecture choice
- Train the model (parallelism, fast tools, etc.)
- ???
- Deployment
- Profit!

Our goal

Acquire the most efficient model for inference

Our goal

Acquire the most efficient model for inference

but…

1. Why?
2. What is efficient model?

Model size

What does the model size affect?

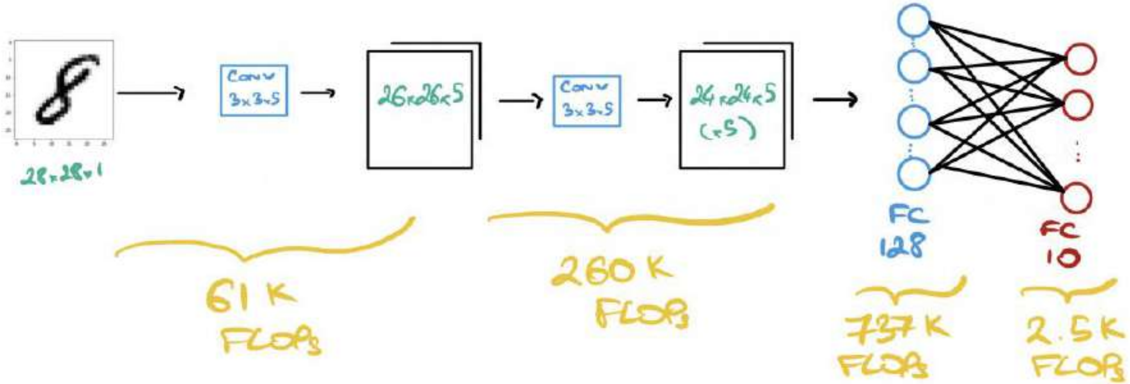
Loading speed, device choice (2 models per device / 3 models per device?), FLOPs

Model speed

- The inference time is **how long** is takes for a forward propagation
- Three core ideas:
  - **FLOPs** or **Floating Point Operations** are total number of calculations such as addition, subtraction, division, multiplication
  - **FLOPS** are the Floating Point Operations per Second
  - **MACs** or **Multiply-Accumulate Computations** are operations that perform addition and multiplication, that is, 2 operations

As a rule, we consider **1 MAC = 2 FLOPs**

Model speed [calculating FLOPs]



- The model will do **FLOPs = 60,840 + 259,200 + 737,280 + 2,560 = 1,060,400** operations
- Say we have a CPU that performs 1 GFLOPS
  FLOPs/FLOPS = (1,060,400)/(1,000,000,000) = **0,001 s or 1ms**

What slows down the model?

- Unnecessary / ineffective operations (attention examples, depthwise convs)
- Synchronisation costs (global pooling, squeeze-and-excitation blocks)
- Memory access (branches in ConvNets)

Ways to optimize models

- Efficient architectures
- Number of parameters reduction
  - Knowledge distillation
  - Pruning
  - Matrices decompositions
- Get the most from training
  - Quantization aware training
  - Stochastic weight averaging
- Take the best framework / engine / server solution

Ways to optimize models

- Efficient architectures
- Number of parameters reduction
  - Knowledge distillation
  - Pruning
  - Matrices decompositions
- Get the most from training
  - Quantization aware training
  - Stochastic weight averaging
- Take the best framework / engine / server solution

Efficient architectures [CV]

What’s the time of 256x256 image classification on CPU?

Efficient architectures [CV]

What’s the time of 256x256 image classification on CPU?

- MobileNets (2017-2019)
  - Convolutions → depthwise-separable convolutions
  - V3: ~ 1ms on IPhone 12, ImageNet accuracy 72%, 3.4kk params
- MobileOne (2022)
  - Reparametrization of branches
  - ~ 1ms on IPhone 12, ImageNet accuracy 78%, 4.5kk params

Efficient architectures [CV] — MobileNet

Main idea: replace ConvBlocks with depthwise convolutions + pointwise convolutions

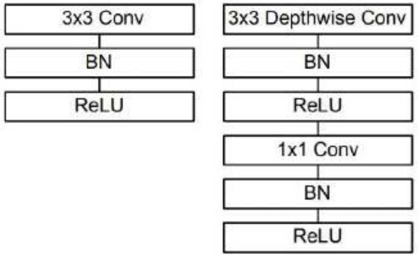
Input: $D_F \times D_F \times M$; Output: $D_F \times D_F \times N$

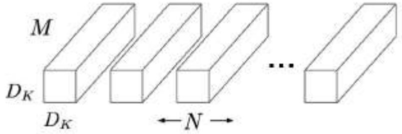
Kernel: $D_K \times D_K \times M \times N$

Standard convolution: $D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$

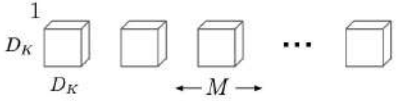
Depthwise convolution: $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$

Depthwise-separable: $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$

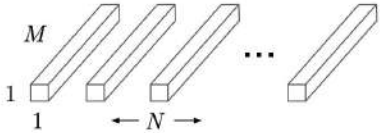




(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



Efficient architectures [CV] — MobileNet

Table 8. MobileNet Comparison to Popular Models

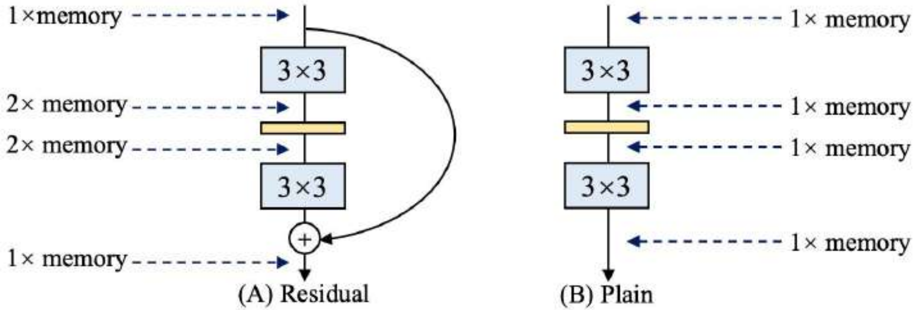
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

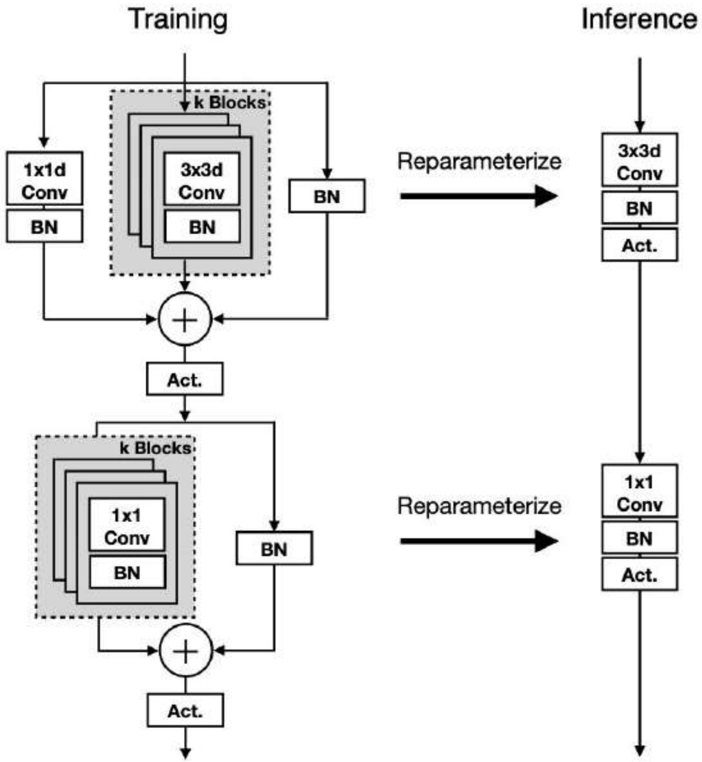
Table 13. COCO object detection results comparison using different frameworks and network architectures. mAP is reported with COCO primary challenge metric (AP at IoU=0.50:0.05:0.95)

Framework Resolution	Model	mAP	Billion Mult-Adds	Million Parameters
SSD 300	deeplab-VGG	21.1%	34.9	33.1
	Inception V2	22.0%	3.8	13.7
	MobileNet	19.3%	1.2	6.8
Faster-RCNN 300	VGG	22.9%	64.3	138.5
	Inception V2	15.4%	118.2	13.3
	MobileNet	16.4%	25.2	6.1
Faster-RCNN 600	VGG	25.7%	149.6	138.5
	Inception V2	21.9%	129.6	13.3
	Mobilenet	19.8%	30.5	6.1

Efficient architectures [CV] — MobileOne

Main idea: remove overparametrized branches from depthwise-separable ConvBlocks





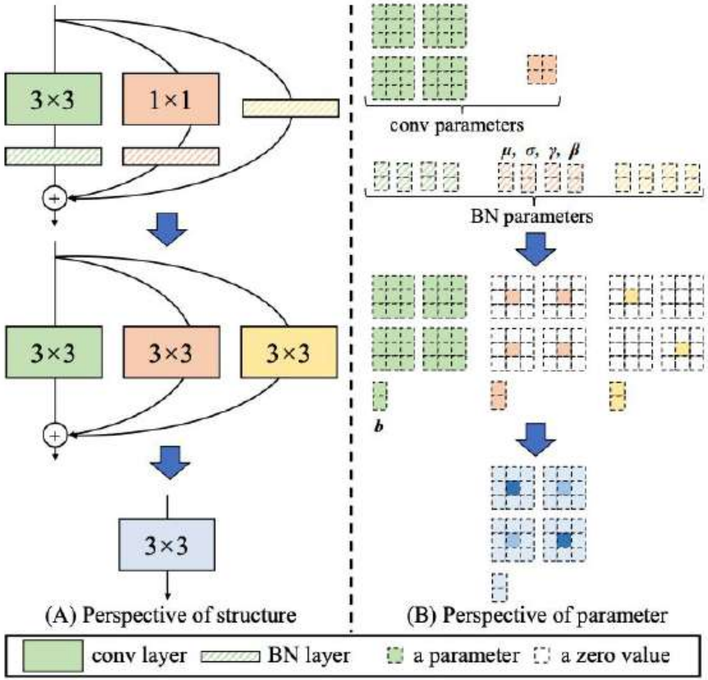
Efficient architectures [CV] — MobileOne

Advantages of branches removal:

- Fast
- Memory-economical
- Flexible architecture

BN and convolution fusion

$$\begin{pmatrix} \hat{F}_{1,i,j} \\ \hat{F}_{2,i,j} \\ \vdots \\ \hat{F}_{C-1,i,j} \\ \hat{F}_{C,i,j} \end{pmatrix} = \begin{pmatrix} \frac{\gamma_1}{\sqrt{\hat{\sigma}_1^2+\epsilon}} & 0 & \cdots & & 0 \\ 0 & \frac{\gamma_2}{\sqrt{\hat{\sigma}_2^2+\epsilon}} & & & \\ \vdots & & \ddots & & \vdots \\ & & & \frac{\gamma_{C-1}}{\sqrt{\hat{\sigma}_{C-1}^2+\epsilon}} & 0 \\ 0 & & \cdots & 0 & \frac{\gamma_C}{\sqrt{\hat{\sigma}_C^2+\epsilon}} \end{pmatrix} \cdot \begin{pmatrix} F_{1,i,j} \\ F_{2,i,j} \\ \vdots \\ F_{C-1,i,j} \\ F_{C,i,j} \end{pmatrix} + \begin{pmatrix} \beta_1 - \gamma_1 \frac{\hat{\mu}_1}{\sqrt{\hat{\sigma}_1^2+\epsilon}} \\ \beta_2 - \gamma_2 \frac{\hat{\mu}_2}{\sqrt{\hat{\sigma}_2^2+\epsilon}} \\ \vdots \\ \beta_{C-1} - \gamma_{C-1} \frac{\hat{\mu}_{C-1}}{\sqrt{\hat{\sigma}_{C-1}^2+\epsilon}} \\ \beta_C - \gamma_C \frac{\hat{\mu}_C}{\sqrt{\hat{\sigma}_C^2+\epsilon}} \end{pmatrix}$$



Efficient architectures [NLP] — ALBERT

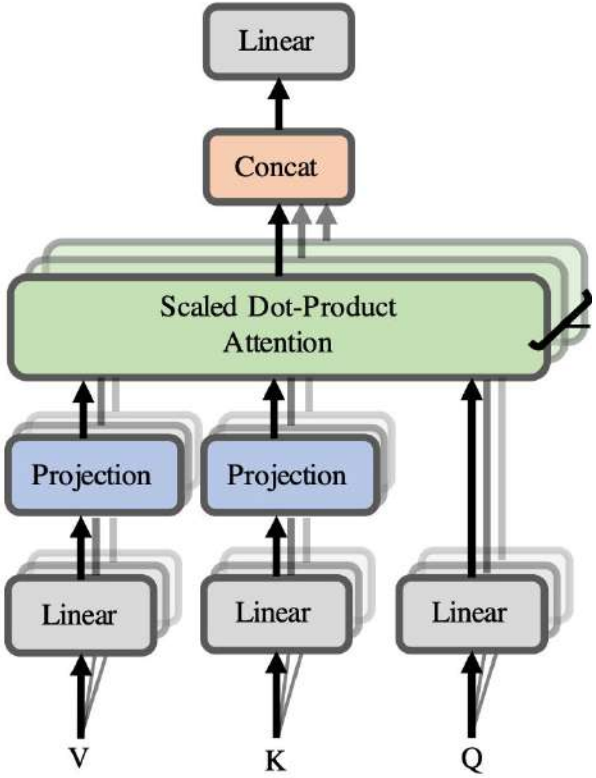
- Projections for embeddings
- Parameters sharing for layers

Model		Parameters	Layers	Hidden	Embedding	Parameter-sharing
BERT	base	108M	12	768	768	False
	large	334M	24	1024	1024	False
ALBERT	base	12M	12	768	128	True
	large	18M	24	1024	128	True
	xlarge	60M	24	2048	128	True
	xxlarge	235M	12	4096	128	True

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

Efficient architectures [NLP] — Linformer

$$\begin{aligned}\overline{\text{head}_i} &= \text{Attention}(QW_i^Q, E_iKW_i^K, F_iVW_i^V) \\ &= \underbrace{\text{softmax}\left(\frac{QW_i^Q(E_iKW_i^K)^T}{\sqrt{d_k}}\right)}_{\bar{P}:n\times k} \cdot \underbrace{F_iVW_i^V}_{k\times d}\end{aligned}$$



Efficient architectures [NLP] — Linear Transformer

- Linear time w.r.t. sequence length
- Constant memory
- Causal masking

$$V_i' = \frac{\sum_{j=1}^{N} \operatorname{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \operatorname{sim}\left(Q_i, K_j\right)}$$

$$V_i' = \frac{\sum_{j=1}^{N} \phi\left(Q_i\right)^T \phi\left(K_j\right) V_j}{\sum_{j=1}^{N} \phi\left(Q_i\right)^T \phi\left(K_j\right)}$$

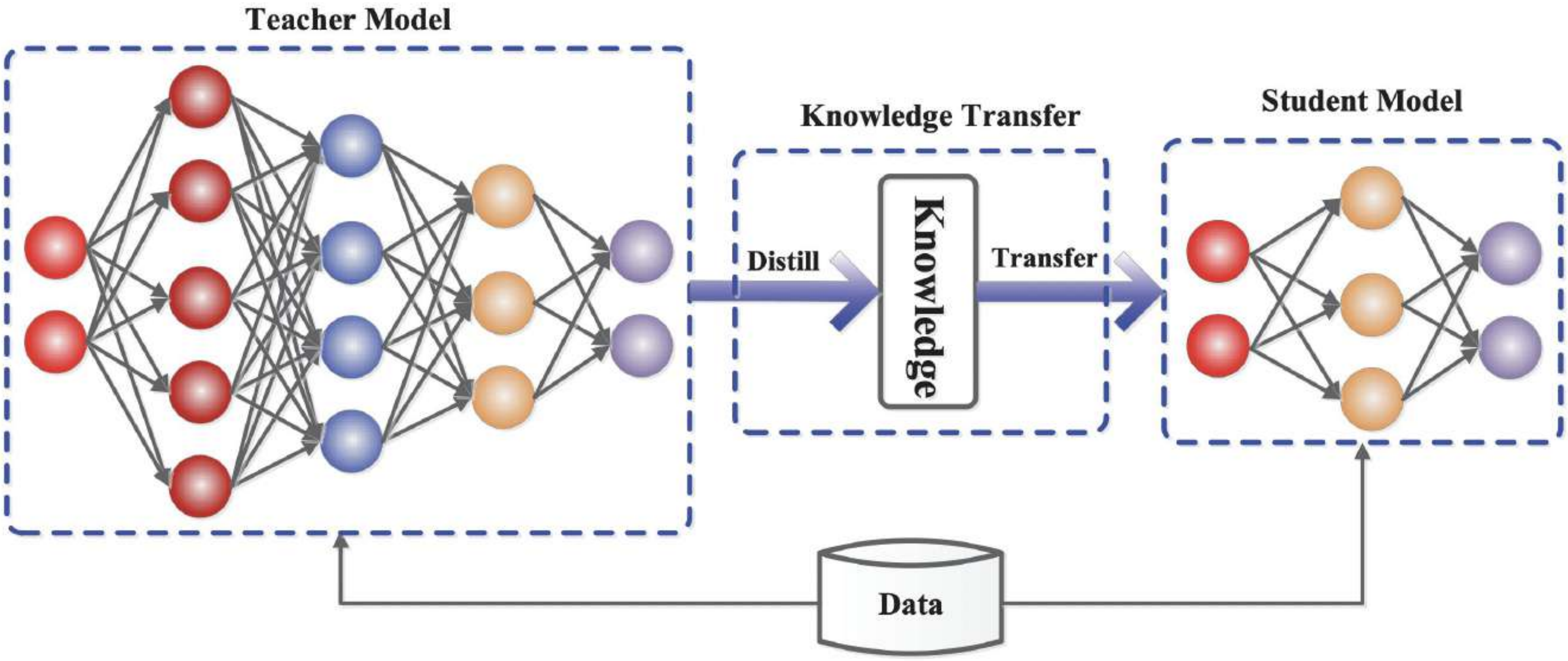
$$V_i' = \frac{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right)}$$

Ways to optimize models

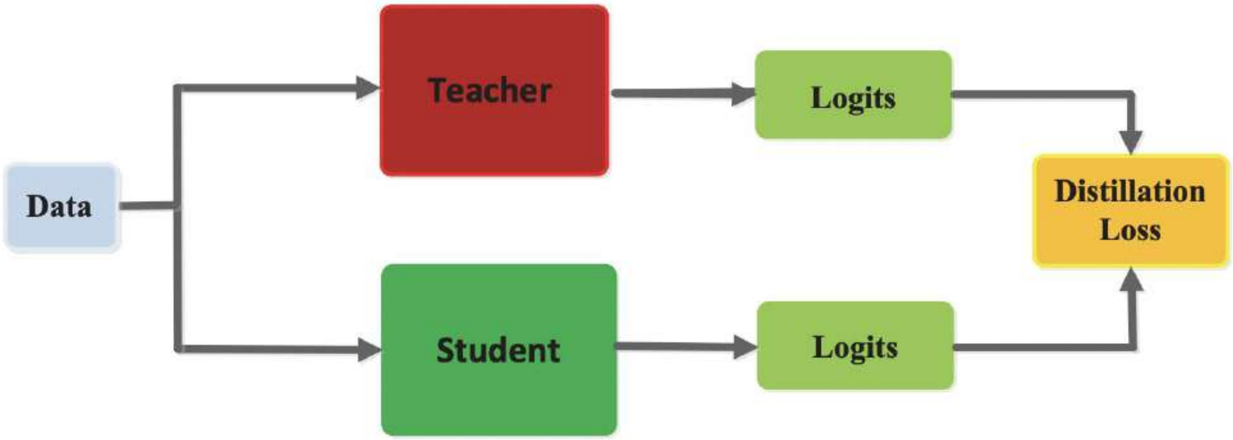
- Efficient architectures
- Number of parameters reduction
  - Knowledge distillation
  - Pruning
  - Matrices decompositions
- Get the most from training
  - Quantization aware training
  - Stochastic weight averaging
- Take the best framework / engine / server solution

Knowledge distillation

Knowledge distillation

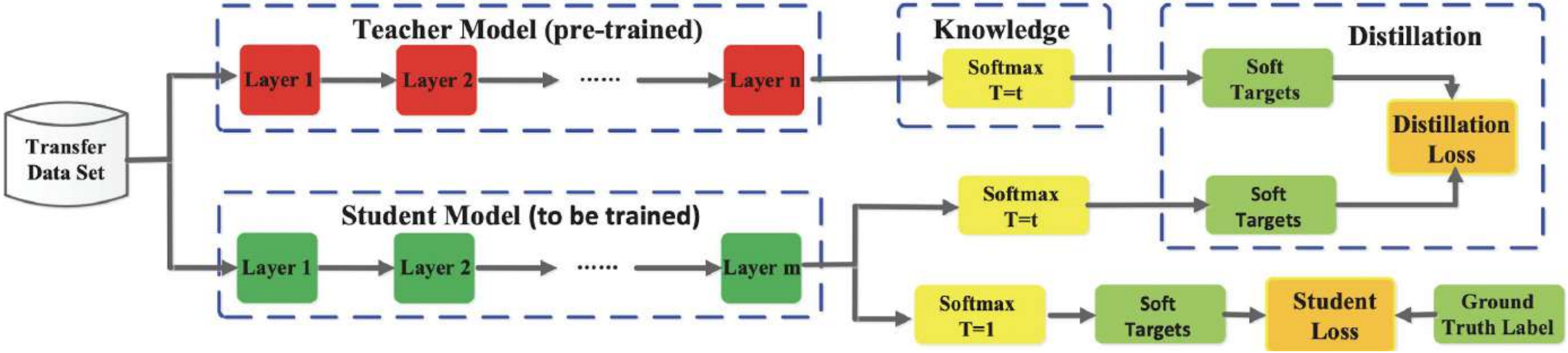


Knowledge distillation [response-based]



- Update **student** weights and freeze **teacher** weights
- The **responses** can be logits, offsets, heatmaps and so on

Knowledge distillation [response-based]



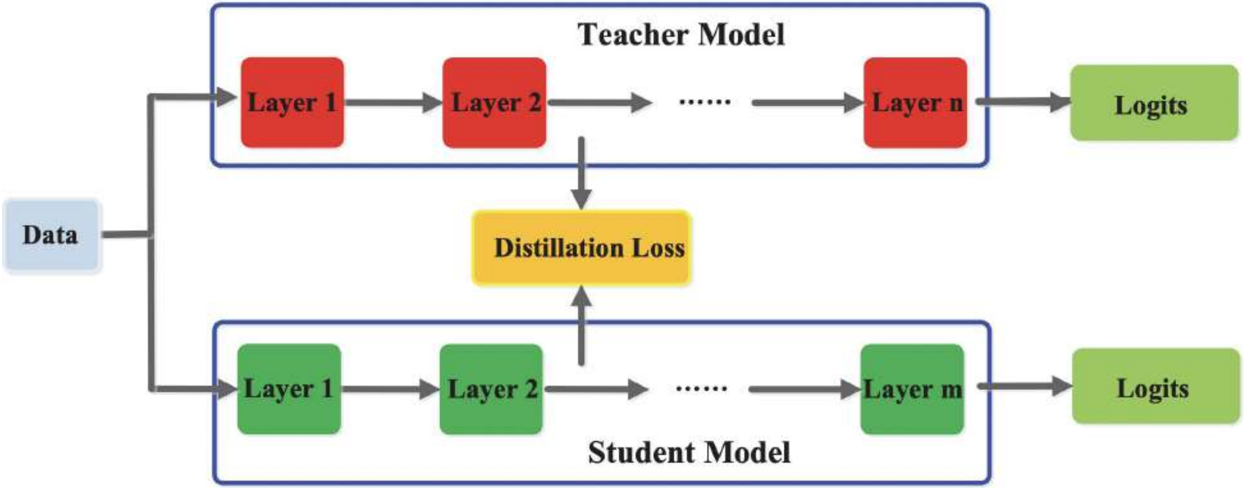
- Optimise weighted combination of **student** and **distillation** losses
- As usual, **student loss** is cross-entropy and **distillation loss** is Kullback-Leibler divergence

Knowledge distillation [response-based]

Temperature	Logits	Softmax Probabilities
1	$[30, 5, 2, 2]$	$[1e+0, 1.38e-11, 6.91e-13, 6.91e-13]$
10	$[3, 0.5, 0.2, 0.2]$	$[0.8308, 0.0682, 0.0505, 0.0505]$

- Soft targets contain the informative **dark knowledge** from the teacher model
- Higher temperatures produce softer probabilities which provides a stronger signal to the student

Knowledge distillation [feature-based]



- Directly match the feature activations of the teacher and the student

Knowledge distillation [feature-based]

$$L = \mathcal{L}_F(\Phi_t(f_t(x)), \Phi_s(f_s(x)))$$

Similarity Function
(L1, L2, MMD, CE)

Alignment Function
(MLP, Conv)

Feature Map

Knowledge distillation [feature-based] — TinyBERT

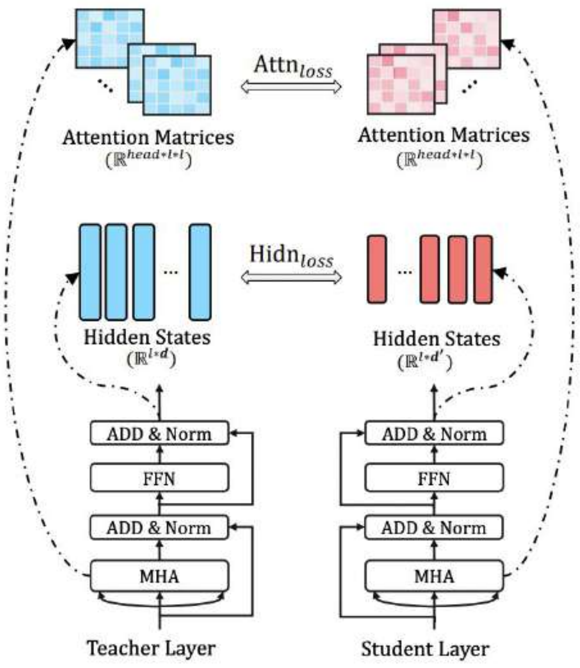
$$\mathcal{L}_{\text{attn}} = \frac{1}{h}\sum_{i=1}^{h} \text{MSE}(\boldsymbol{A}_i^S, \boldsymbol{A}_i^T)$$

$$\mathcal{L}_{\text{hidn}} = \text{MSE}(\boldsymbol{H}^S W_h, \boldsymbol{H}^T)$$

$$\mathcal{L}_{\text{embd}} = \text{MSE}(\boldsymbol{E}^S \boldsymbol{W}_e, \boldsymbol{E}^T)$$

$$\mathcal{L}_{\text{pred}} = \text{CE}(\boldsymbol{z}^T/t, \boldsymbol{z}^S/t)$$

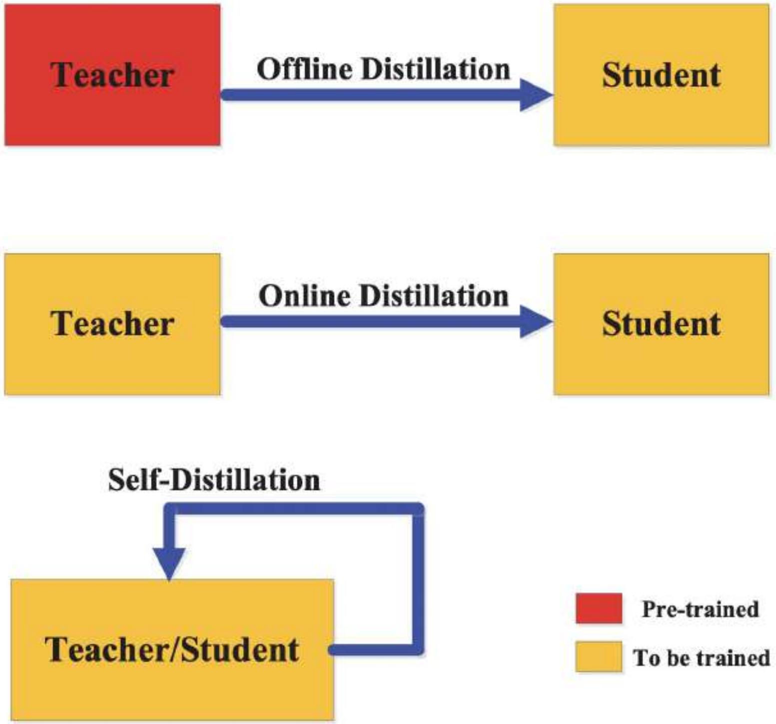
$$\mathcal{L}_{\text{model}} = \sum_{x\in\mathcal{X}} \sum_{m=0}^{M+1} \lambda_m \mathcal{L}_{\text{layer}}\left(f_m^S(x), f_{g(m)}^T(x)\right)$$



Knowledge distillation [a good teacher]

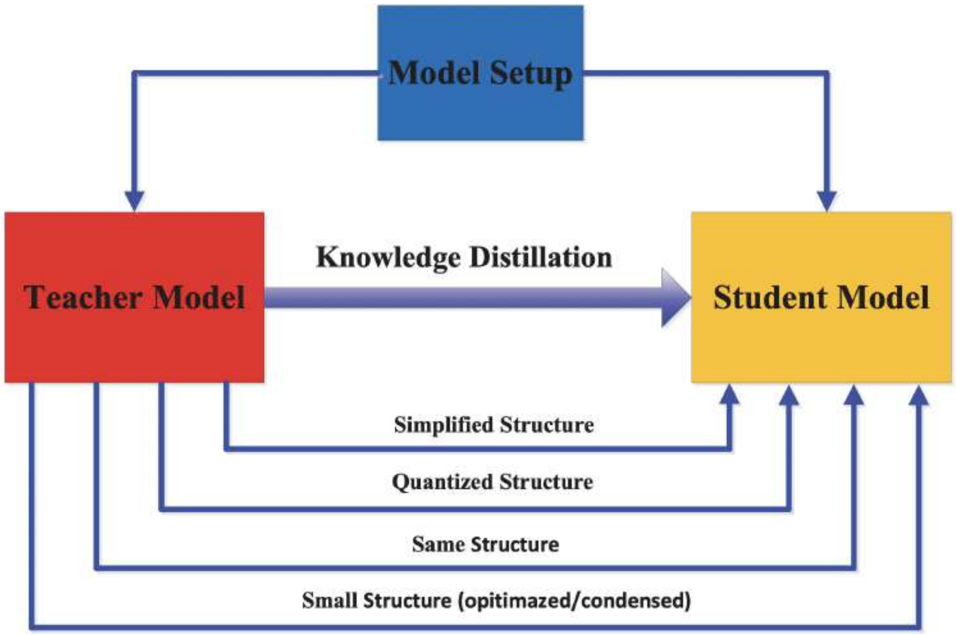
- Distillation as function matching
- Consistent teaching
- Patient teaching
- Good for new data

Knowledge distillation [schemes]



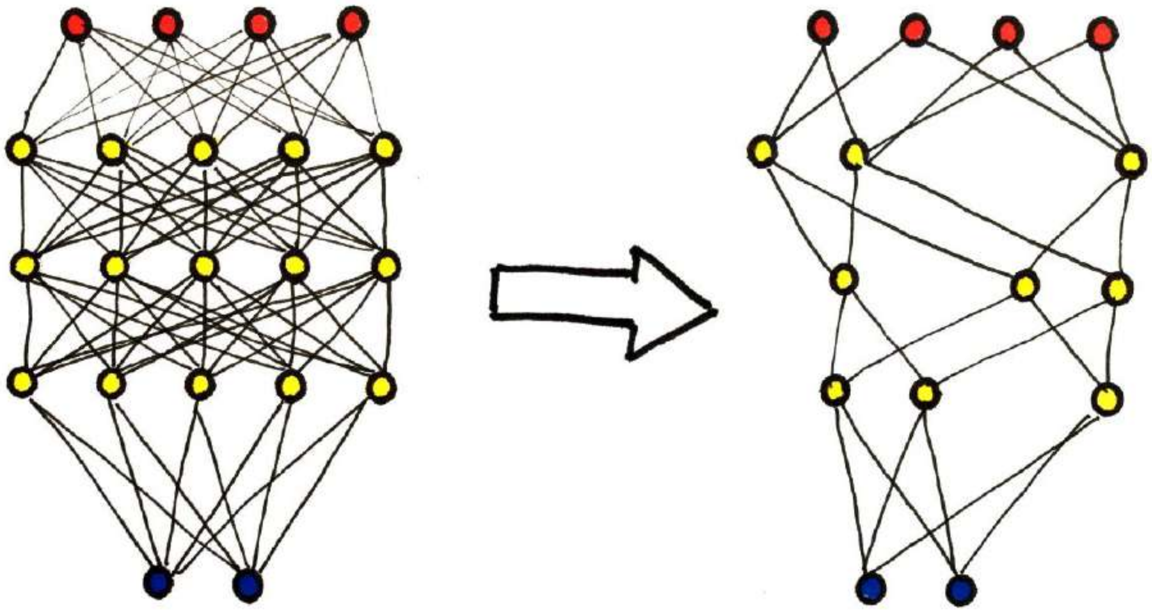
Knowledge distillation [schemes]

- Fewer layers or fewer channels in each layer
- Quantized version
- Efficient basic operations

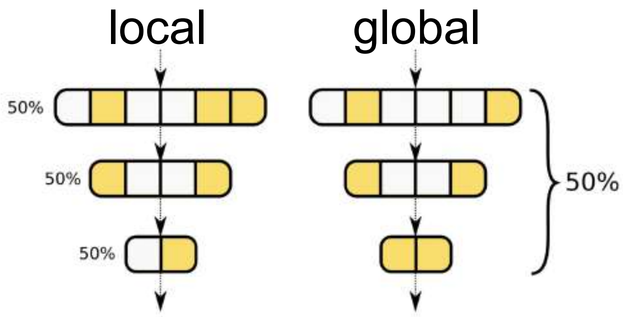


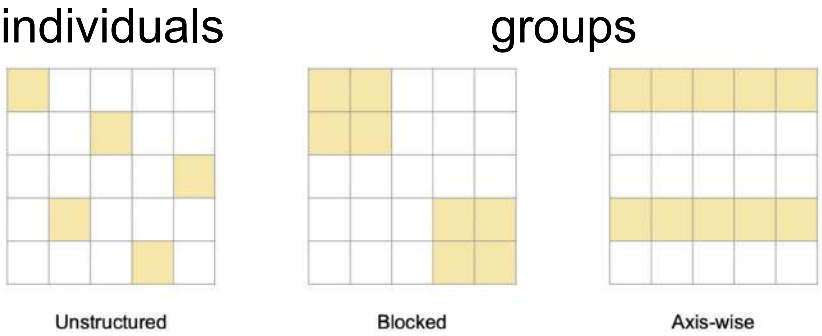
Pruning

Pruning



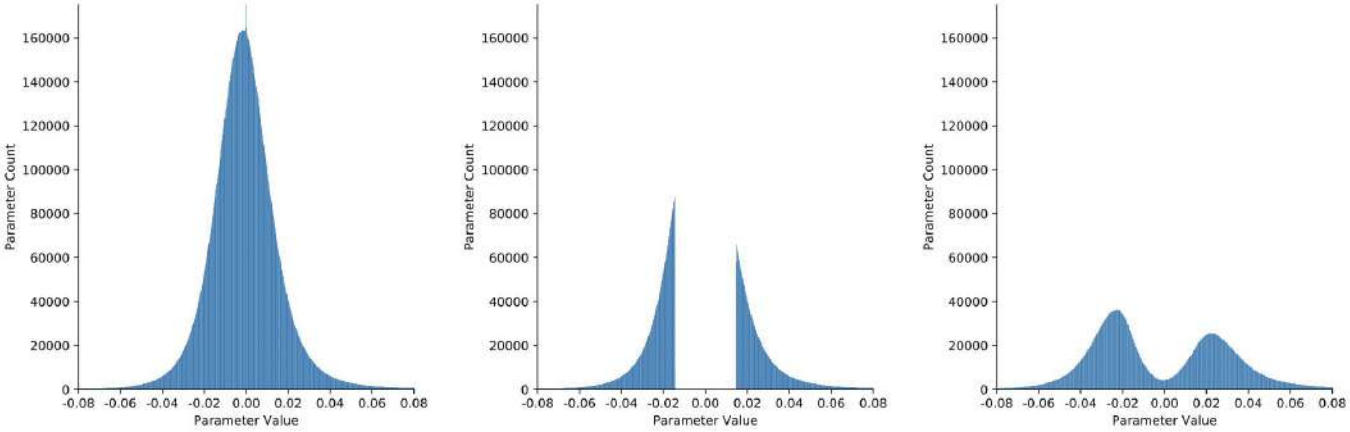
Pruning





$$-\sum_{i=1}^{N}\log p(y_i \mid x_i, W) + \lambda \sum_{\eta\in\mathcal{H},\, w_\eta\in W} \|w_\eta\|_2 \to \min$$

Pruning

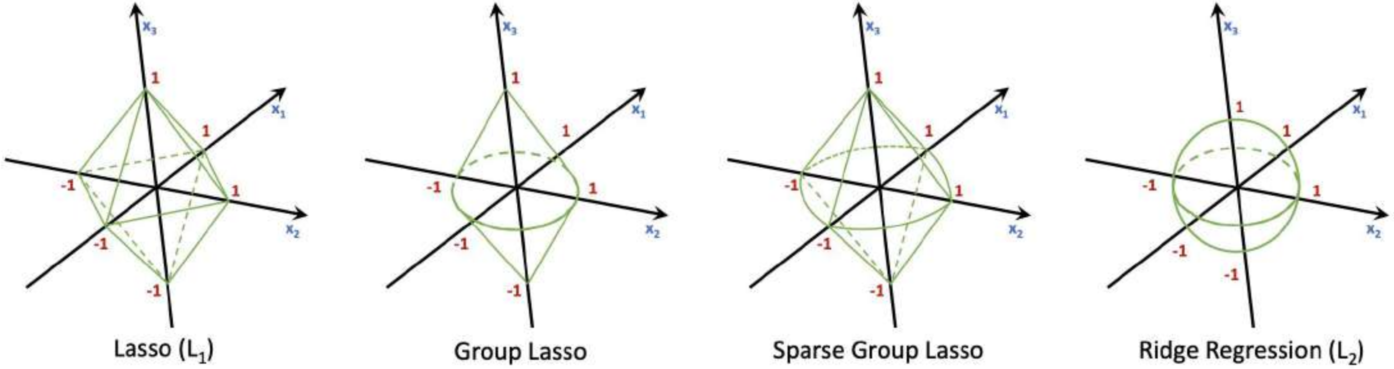


(a) Dense Network (76.0%) (b) 70% Pruned (36.1%) (c) After 3-epoch Retraining (71.4%)

$$\text{threshold}(w_i) = \begin{Bmatrix} w_i, & \text{if } |w_i| > \lambda \\ 0, & \text{if } |w_i| \leq \lambda \end{Bmatrix}$$

$$\text{threshold}_\tau(w_i) = \begin{Bmatrix} w_i, & \text{if } |w_i| \ \text{ in TOP-p} \\ 0, & \textit{else} \end{Bmatrix}$$

Pruning

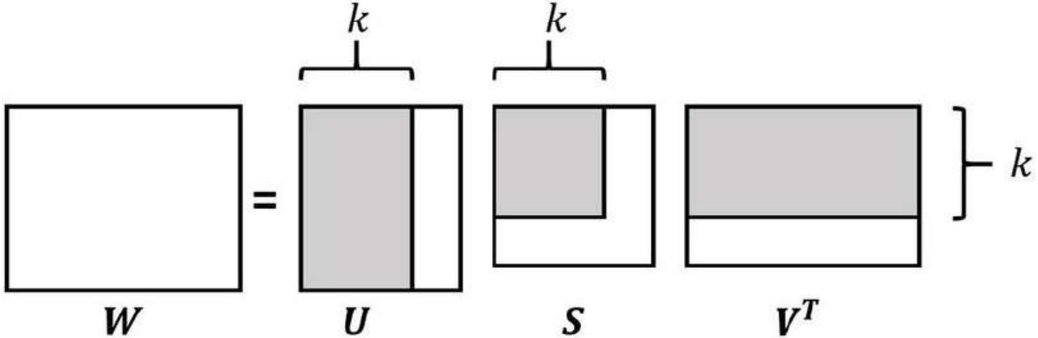


$$\min_{\beta \in \mathbb{R}^p} \left(\left\| \mathbf{y} - \sum_{g=1}^{G} \mathbf{X_g} \beta_g \right\|_2^2 + \lambda_1 \sum_{g=1}^{G} \|\beta_g\|_2 + \lambda_2 \|\beta\|_1 \right)$$

Matrices decompositions

Matrices decompositions

- Linear layer: Y = X W; where X is *(p, n)* and W is *(n, m)*
- SVD: W = U Σ V^T; where U is *(n, n)*, Σ is diagonal *(n, m)*, V is *(m, m)*
- Truncate SVD
- Change order of multiplications
- Acquired complexity change: $p\ n\ m \rightarrow p\ n\ k + \ k\ m\ n$
- $k < p\ n\ m\ /\ (p\ n + m\ n)$

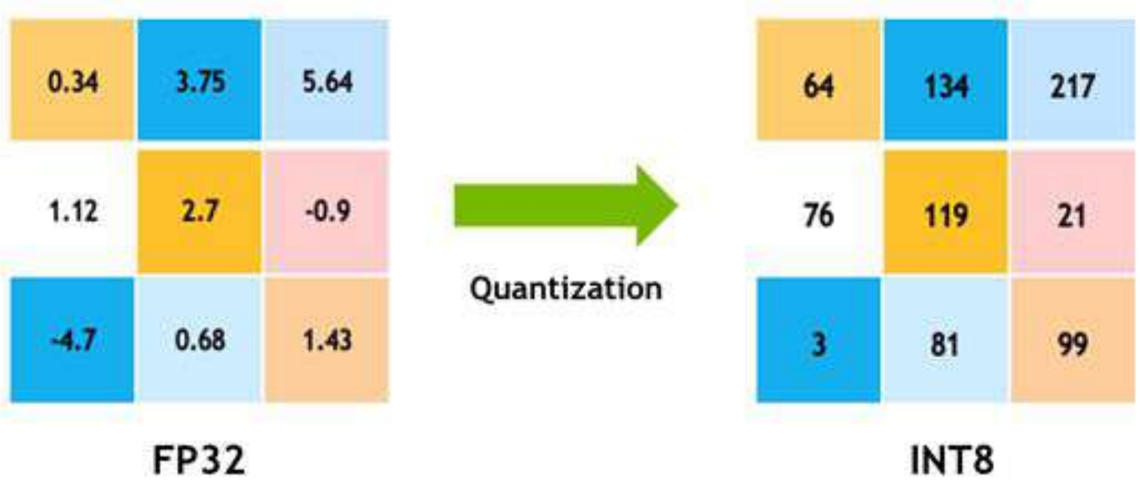


Ways to optimize models

- Efficient architectures
- Number of parameters reduction
  - Knowledge distillation
  - Pruning
  - Matrices decompositions
- Get the most from training
  - Quantization aware training
  - Stochastic weight averaging
- Take the best framework / engine / server solution

Quantization

Quantization



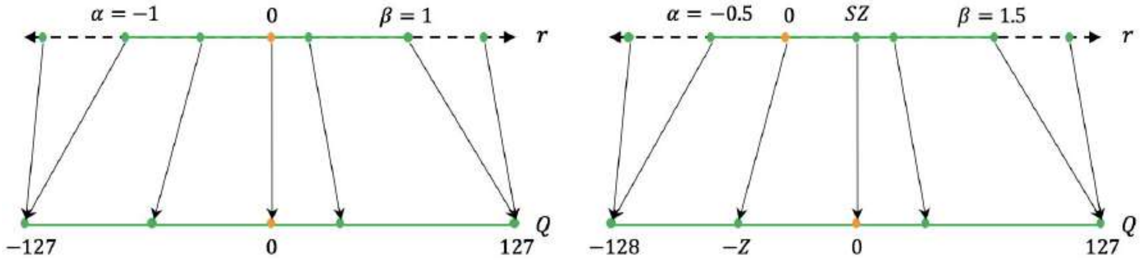
Quantization

Quantize:

$$Q(r) = \mathrm{Int}(r/S) - Z$$

$$S = \frac{\beta - \alpha}{2^b - 1}$$

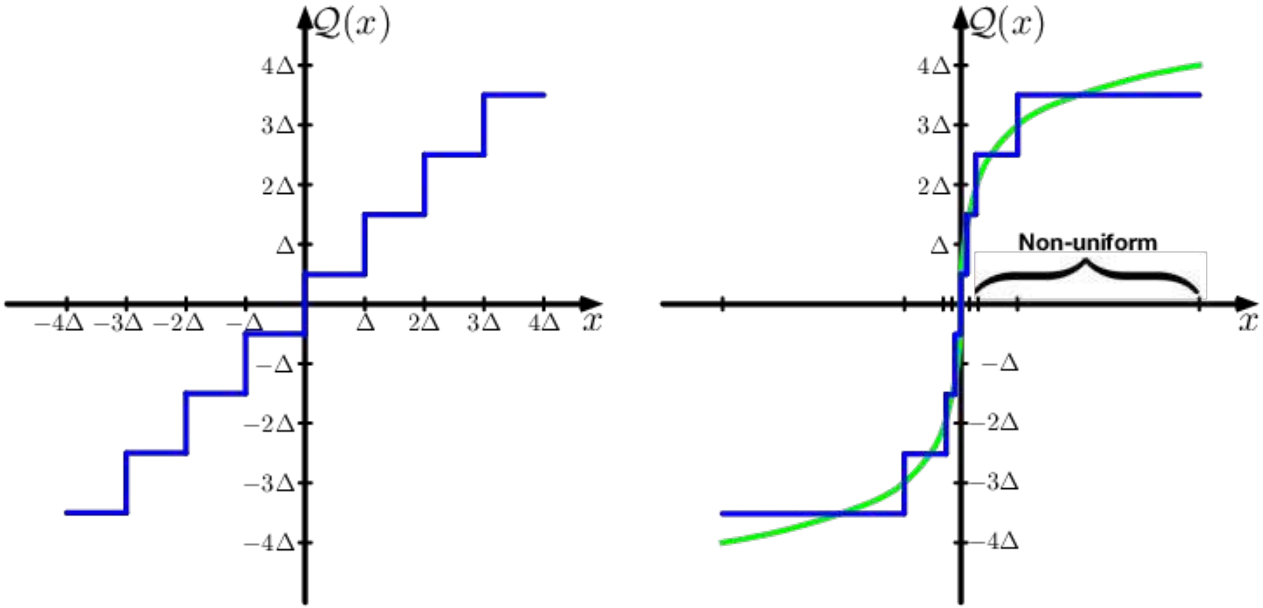
$$Z = -\left(\frac{\alpha}{S} - \alpha_q\right)$$



Dequantize:

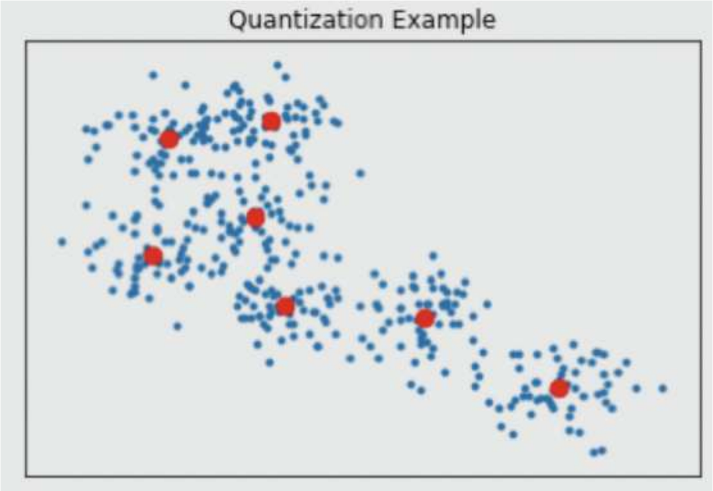
$$\tilde{r} = S(Q(r) + Z)$$

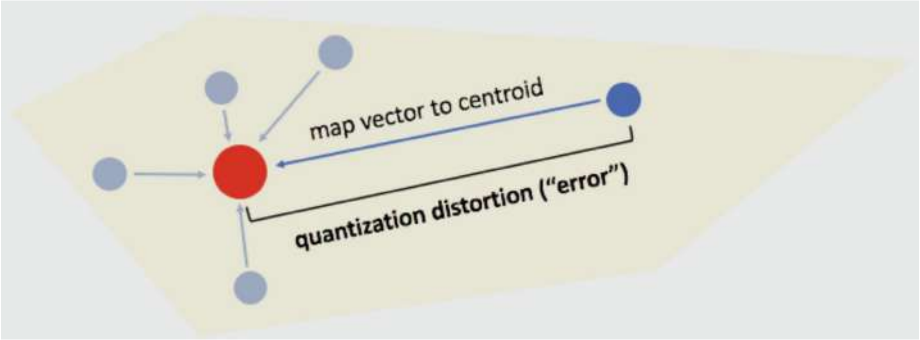
Quantization [uniformity]



(a) (b)

Quantization [clustering]

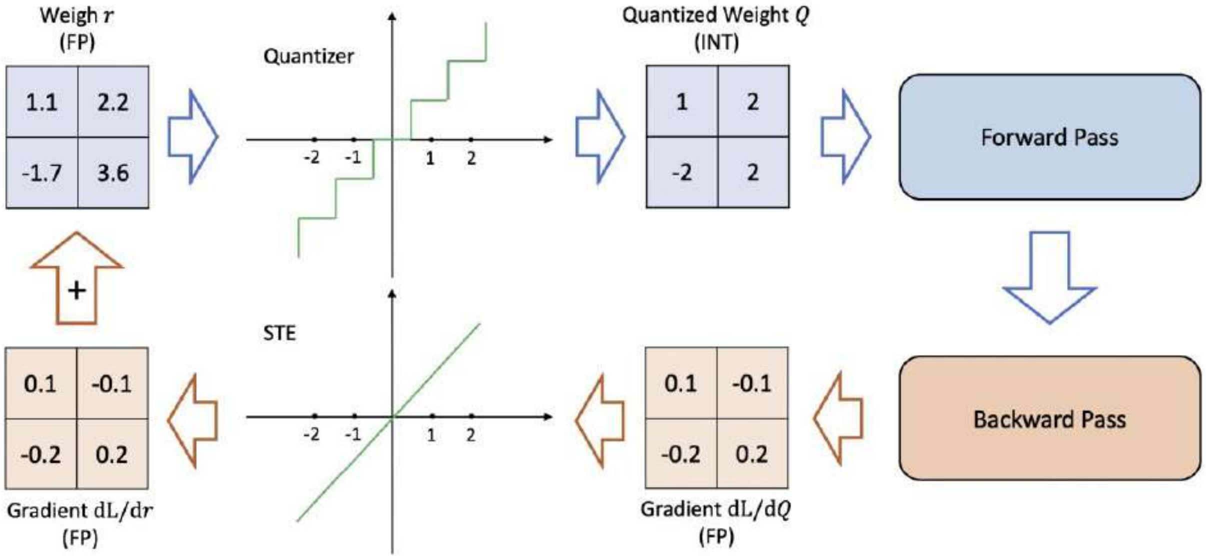




Quantization [static VS dynamic]

- Static quantization
  - Post training procedure
  - Activations are fused to layers if possible
  - Scaling factors are computed on the representative dataset
  - Suitable for CNNs
- Dynamic quantization
  - On the fly during inference
  - Weights are converted to int8, activations are in full precision
  - Scaling factors are computed on the fly in full precision based on activations
  - Suitable for Transformers

Quantization [quantization aware training]

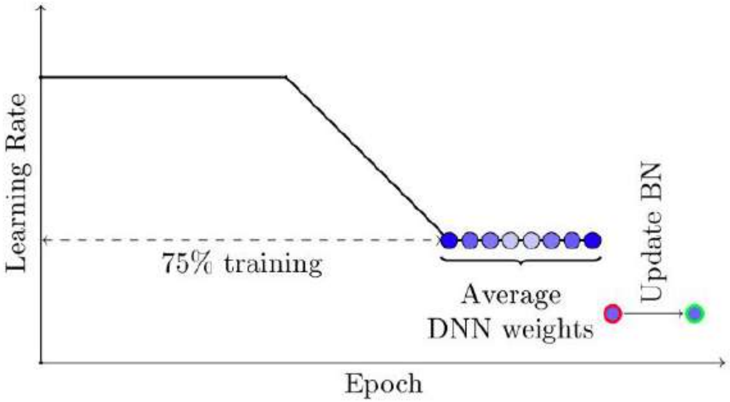


Stochastic weight averaging

Stochastic weight averaging

Get better models to lose less quality while reducing model's size

Simple averaging of the model's weights for the last several epochs may improve convergence



Ways to optimize models

- Efficient architectures
- Number of parameters reduction
  - Knowledge distillation
  - Pruning
  - Matrices decompositions
- Get the most from training
  - Quantization aware training
  - Stochastic weight averaging
- Take the best framework / engine / server solution

Take the best framework / engine / server solution

Use ONNX to fuse layers and quantize model

Use DeepSpeed with ONNX RT and everything else we’ve teached you previously

Takeaways

- Carefully chose architecture for your problem
- Use knowledge distillation
- Quantize model
- Use best practices for training
- Use efficient software