EMBEDDED MACHINE LEARNING

HIGHRR SUMMER SCHOOL 2023

Holger Fröning
holger.froening@ziti.uni-heidelberg.de
Computing Systems Group, Institute of Computer Engineering
Heidelberg University

ML APPLICATIONS

Augmented Reality

Robotics

Near-Sensor Processing

Speech Recognition

Data set containing N input-target pairs: $\mathscr{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$

MODERN ML

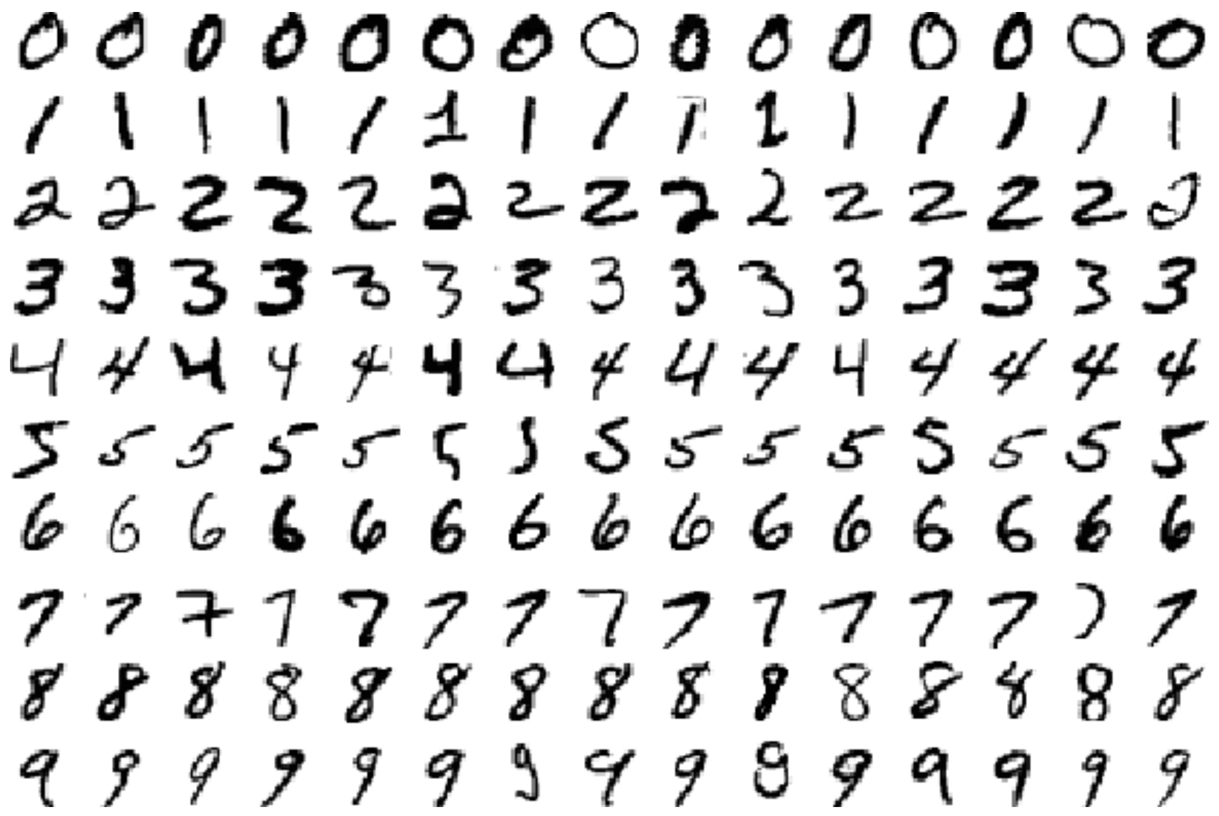
Training: ~ $O(10^{18})$ OPs
Inference: ~ $O(10^{9})$ OPs

Image & video: classification, object localization & detection

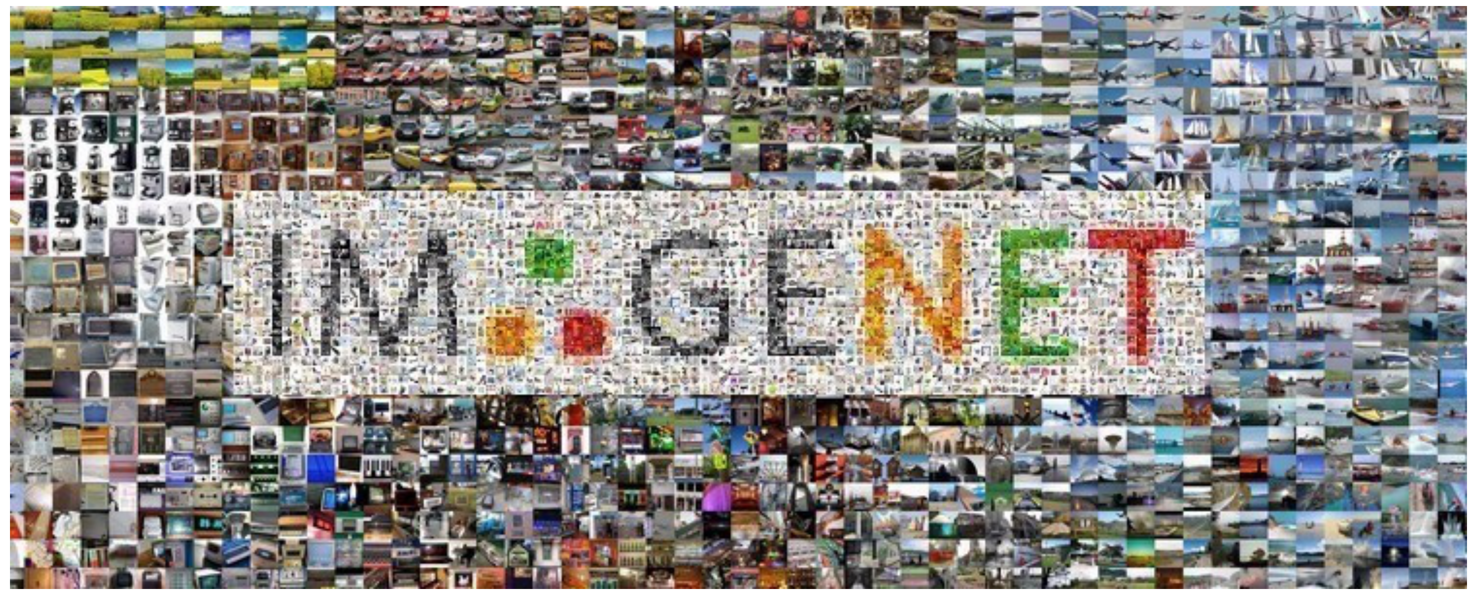
Speech and language: speech recognition, natural language processing

Medical: imaging, genetics of diseases

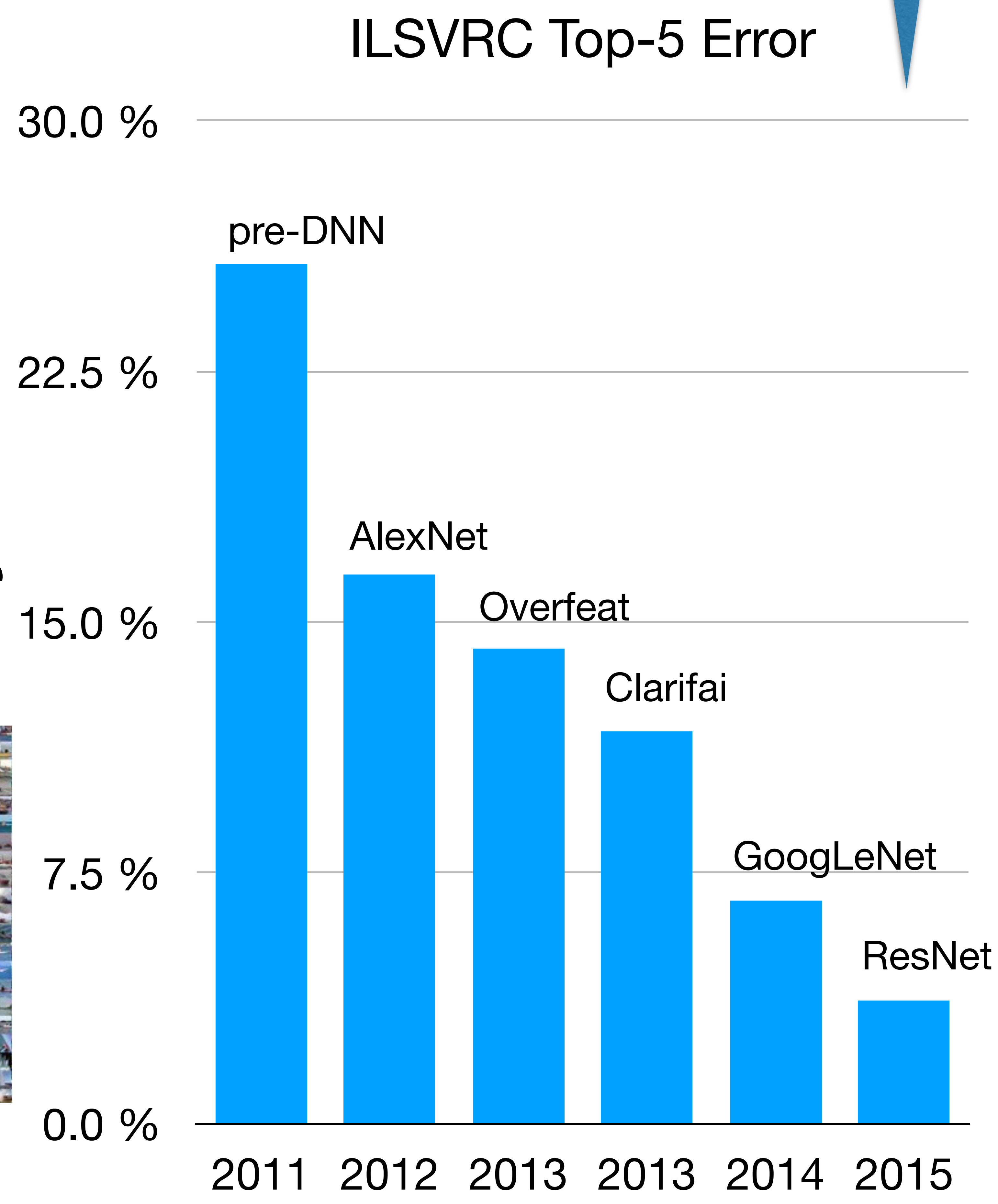
Various: game play, robotics



MNIST handwritten database



IMAGENET: 1000 classes



Artificial Neural Networks (ANNs) deliver state-of-the-art accuracy for many AI tasks
… at the cost of extremely high computational complexity

(PUBLIC) DATASET OVERVIEW

	Image size	Classes	Dataset size	SOTA error
MNIST	28x28x1	10	60,000 + 10,000	0.21% [1]
SVHN	Variable (32x32x3)	10	73,257 + 26,032 (+ 531,131)	1.69% [2]
CIFAR-10	32x32x3	10	50,000 + 10,000	96.53% [3] (accuracy)
CIFAR-100	32x32x3	100	50,000 + 10,000	75.72% [4] (accuracy)
ILSVRC2015	224x224x3	1000	14M	4.49% (TOP-5)/ 19.38% (TOP-1) [5]

Trains on my wimpy laptop in ~10min

Trains in ~10min, if you had 2k GPUs (ResNet-50, M40s)

[1] *Wan, L., Zeiler, M. D., Zhang, S., LeCun, Y., and Fergus, R. (2013). Regularization of neural networks using dropconnect. ICML*

[2] *Lee, C., Gallagher, P. W., and Tu, Z. (2015). Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. CoRR, abs/1509.08985.*

[3] *Graham, B. (2014). Fractional max-pooling. CoRR, abs/1412.6071.*

[4] *Clevert, D., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). CoRR, abs/1511.07289.*

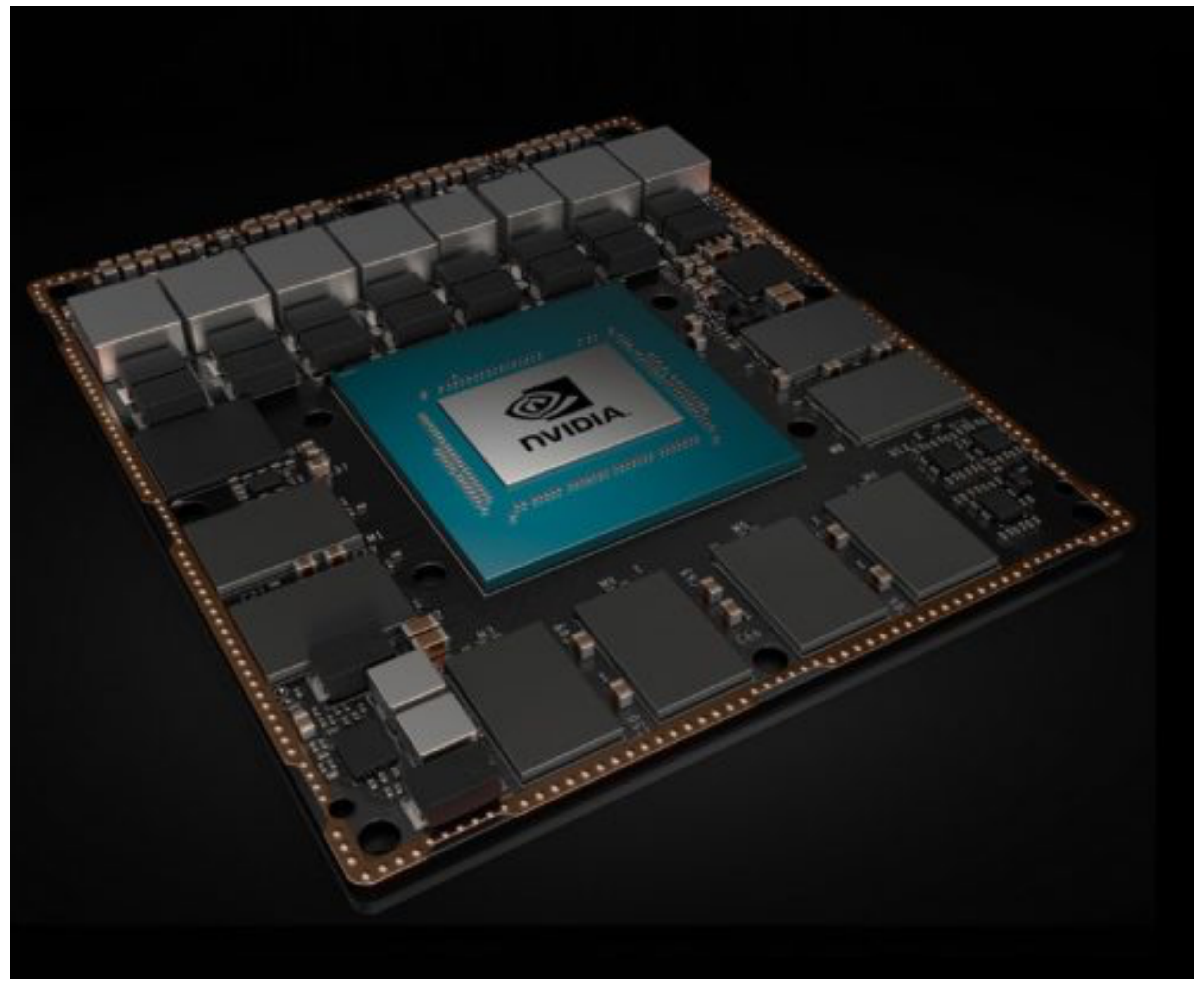
[5] *He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. CoRR, abs/1512.03385.*

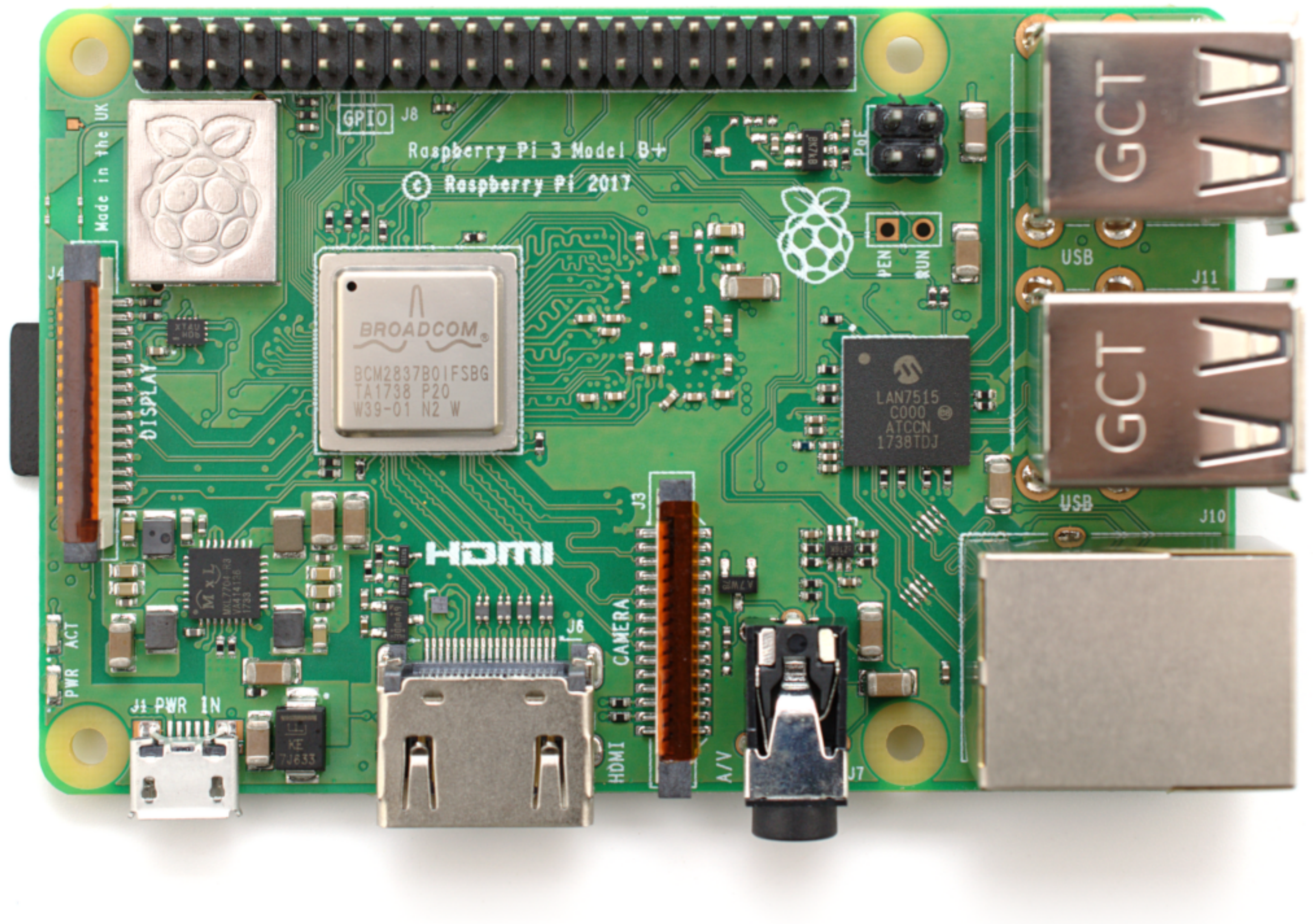
ANN TRENDS

	LeNet 5	AlexNet	Overfeat fast	VGG-16	GoogLeNet v1	ResNet 50	ResNet 152
Top-5 error [%]	n/a	16.4	14.2	7.4	6.7	5.3	4.5
# CONV layers	2	5	5	13	57	53	155
Weights	2.6k	2.3M	16M	14.7M	6.0M	23.5M	58M
MACs	283k	666M	2.67G	15.3G	1.43G	3.86G	11.3G
# FC layers	2	3	3	3	1	1	1
Weights	58k	58.6M	130M	124M	1M	2M	2M
MACs	58k	58.6M	130M	124M	1M	2M	2M
Total weights	60k	61M	146M	138M	7M	25.5M	60M
Total MACs	341k	724M	2.8G	15.5G	1.43G	3.9G	11.3G

FORWARD PATH ONLY. ADDITIONAL LAYERS (POOLING, BATCH NORMALIZATION, ...) AND ACTIVATION FUNCTION NOT INCLUDED.

EXTREME MISMATCH BETWEEN ANN COMPLEXITY AND MOBILE PROCESSOR CAPABILITY





	NVIDIA Xavier	XILINX Zynq Ultrascale+ ZU19EG	Raspberry Pi 3 B+
Wattage	30W	~10W	6W
Peak GFLOP/s	1,300 (325 images/s[1])	difficult	5.6 (1.4 images/s[1])
Total memory	16GB	2GB	1GB
In-core memory	2.9MB (2.8%[2])	4.3MB (4.2%[2])	2.3MB (2.3%[2])

[1] based on theoretical peak GFLOP/s performance, [2] weights only, both for ResNet-50/ImageNet

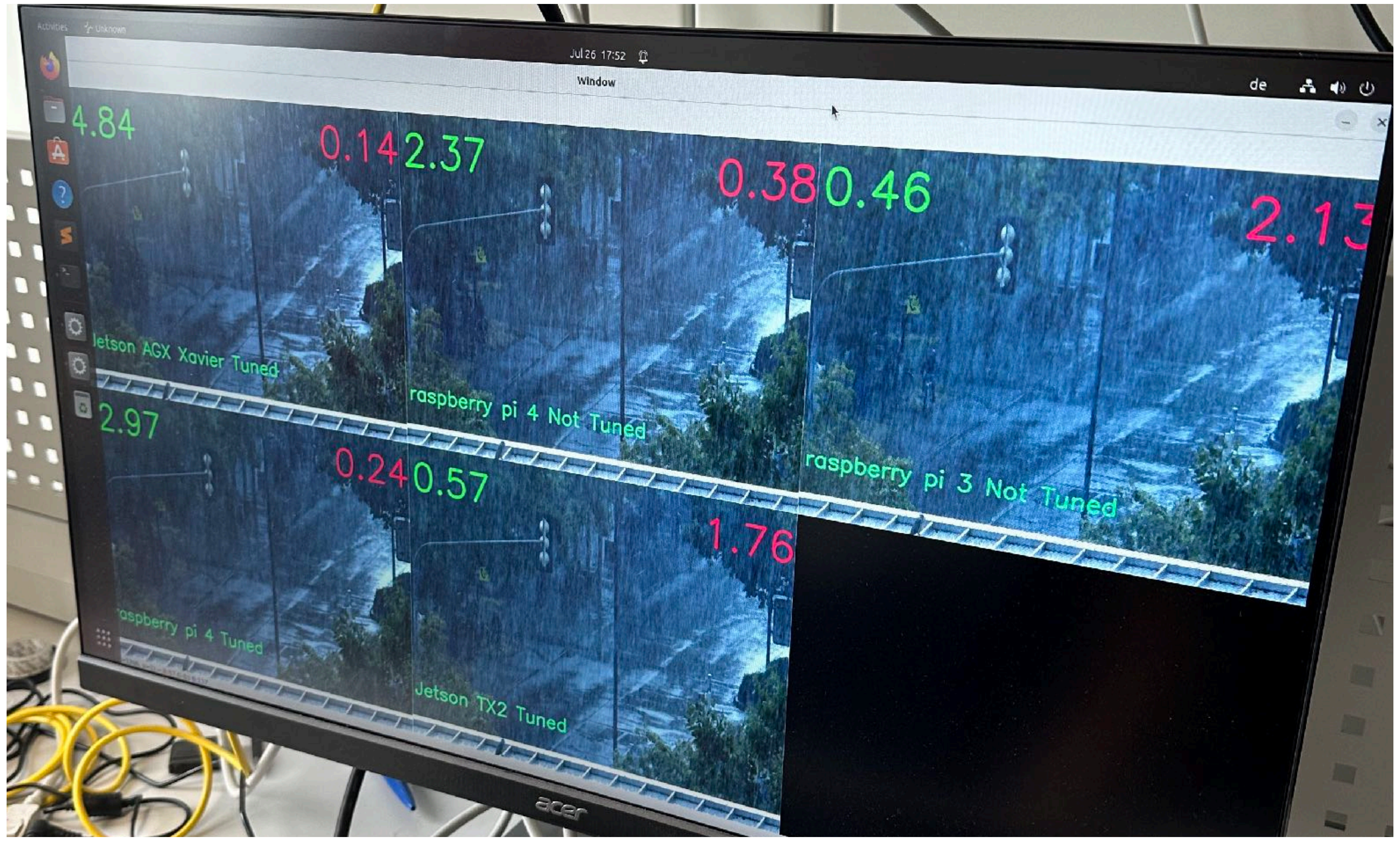
EMBEDDED MACHINE LEARNING

Embedded like

- Embedded systems as resource-constrained devices
- (For reasonable ML tasks, basically any computing system is resource-constrained)
- Embedded in the real world and exposed to uncertainties

What will we learn?

- Some DNN basics - language, notations, (few) intuitions
- DNN compression methods



LINEAR AND POLYNOMIAL REGRESSION

Learning, generalization, model selection, regularization, overfitting

With material from Andrew Ng (CS229 lecture notes) and Christopher Bishop (Pattern Recognition and Machine Learning)

SUPERVISED LEARNING

Given such housing data, how can we learn to predict other house prices?

"Unseen data"

Notation

Input features $x^{(i)}$

Target variable (or output variable or label) $t^{(i)}$

Training sample (or observation) $(x^{(i)}, t^{(i)})$

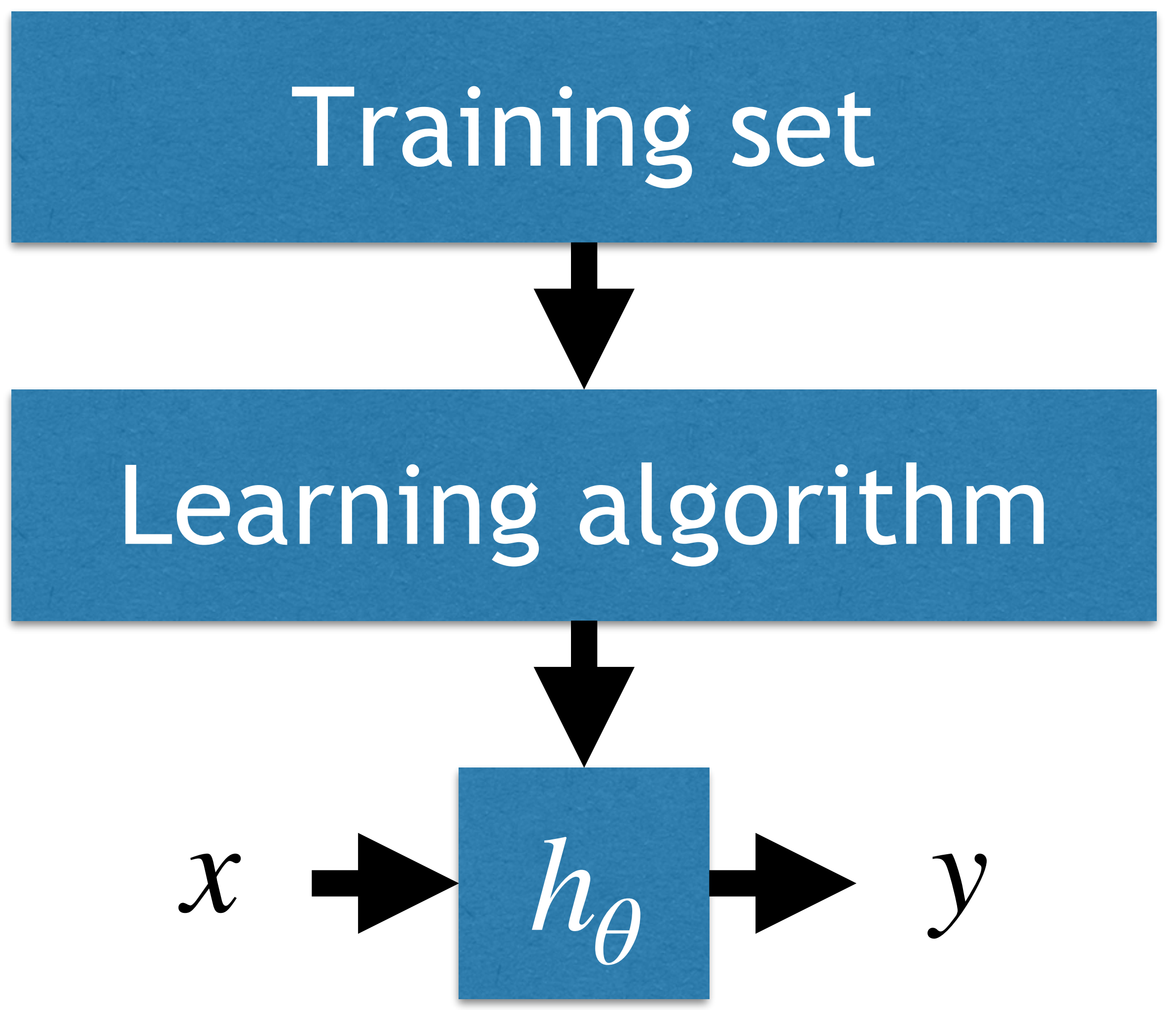
Training set: set of all training samples (size N)

Supervised learning problem: find good prediction function $y = h_\theta(x)$

θ (theta) are the parameters (weights) of the model

Classification (discrete) vs. regression (continuous) problem

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮



LINEAR REGRESSION

$$\mathbf{x} = (x_1, x_2)^T = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$$

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

Supervised learning: choose function h

$$y = h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Simplification given D model parameters:

$$h_\theta(\mathbf{x}) = h(\mathbf{x}) = \sum_{d=1}^{D} \theta_d x_d = \theta^T \mathbf{x}$$ (model intercept θ_0 by $x_0 = 1$)

Learning: make $h(x)$ close to t for the N training samples we have

Cost (or error or loss) function “how close is that”: $J(\theta) = \frac{1}{2} \sum_{n=1}^{N} \left(h_\theta(x^{(n)}) - t^{(n)} \right)^2$

Least-squares method to find the optimal parameters by minimizing this sum of squared residuals

GRADIENT DESCENT

Choose θ such that $J(\theta)$ is minimal

Start with initial guess of θ, repeatedly perform gradient descent:

$\theta_d := \theta_d - \alpha \frac{\partial}{\partial \theta_d} J(\theta)$, simultaneously for all $d = 1,\ldots,D$ and learning rate α

$$\frac{\partial}{\partial \theta_d} J(\theta) = \frac{\partial}{\partial \theta_d} \frac{1}{2} \sum_{n=1}^{N} (h_\theta(x) - t)^2 = \frac{2}{2} \sum_{n=1}^{N} (h_\theta(x) - t) \cdot \frac{\partial}{\partial \theta_d} \Big(\big(\sum_{i=1}^{D} \theta_i x_i \big) - t \Big) = \sum_{n=1}^{N} (h_\theta(x) - t) x_d$$

Hint: remember chain rule of calculus - for $f(x) = u\big(v(x)\big), f'(x) = u'\big(v(x)\big)v'(x)$

=> Update rule: $\theta_d := \theta_d + \alpha \sum_n \big(t^{(n)} - h_\theta(x^{(n)}) \big) x_d^{(n)}$

Magnitude of update is proportional to error term

Which set of the training samples (elements n) to consider for one update?

BATCH GRADIENT DESCENT

Only one global optima as J is a convex quadratic function

Batch gradient descent: $\forall d \in D$

$$\theta_d := \theta_d + \alpha \sum_{n=1}^{N} \left(t^{(n)} - h_\theta(x^{(n)})\right) x_d^{(n)}$$

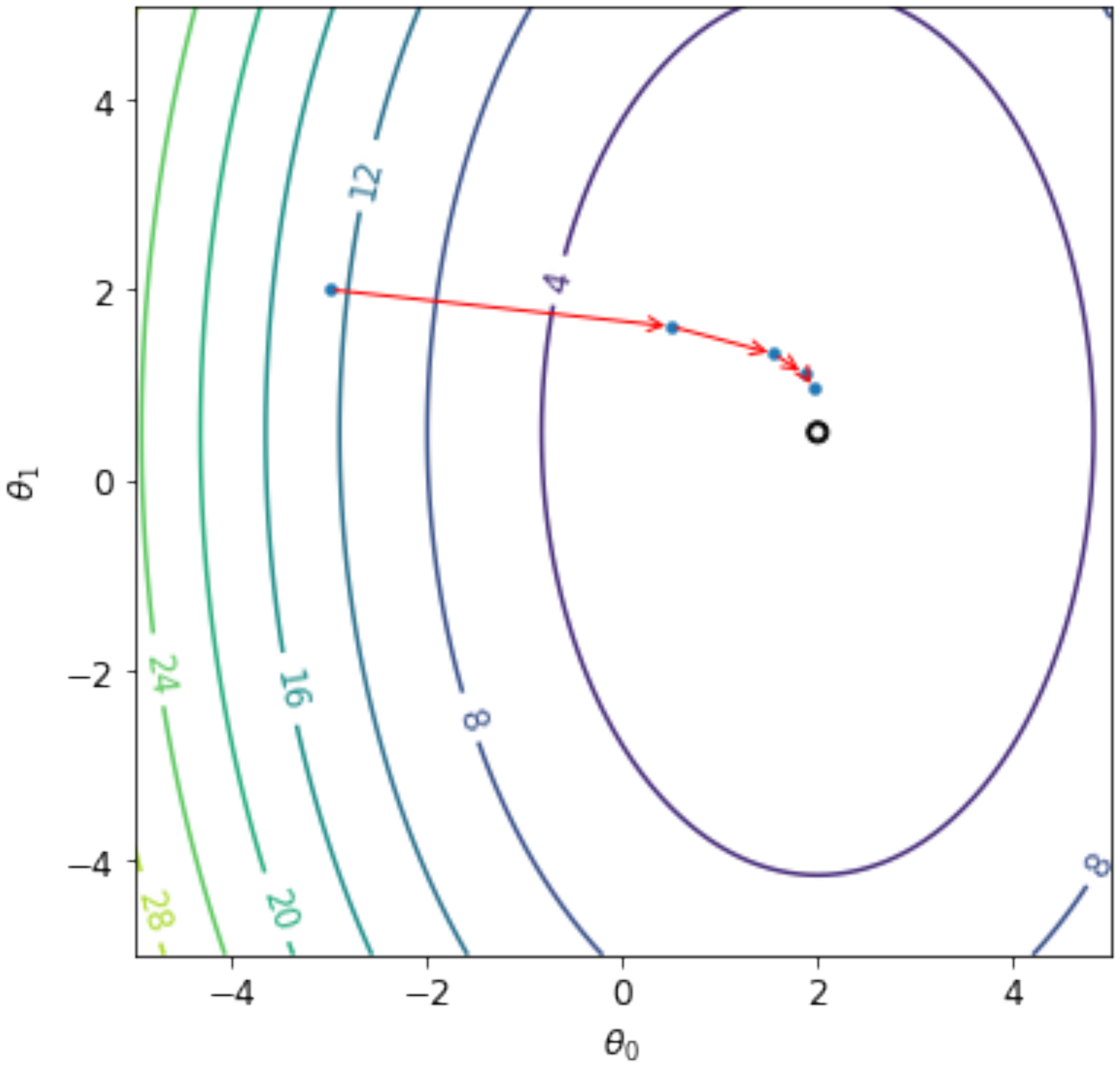
Repeat until convergence

Looks at every training sample ($\forall n \in N$) on every step

Number of steps depend on convergence

Guaranteed to be optimal, but expensive

Cost function



STOCHASTIC (INCREMENTAL) GRADIENT DESCENT

Scanning the complete data set for every step can be costly

Stochastic gradient descent is based on randomly selecting training samples to perform gradient descent

for all n in N:

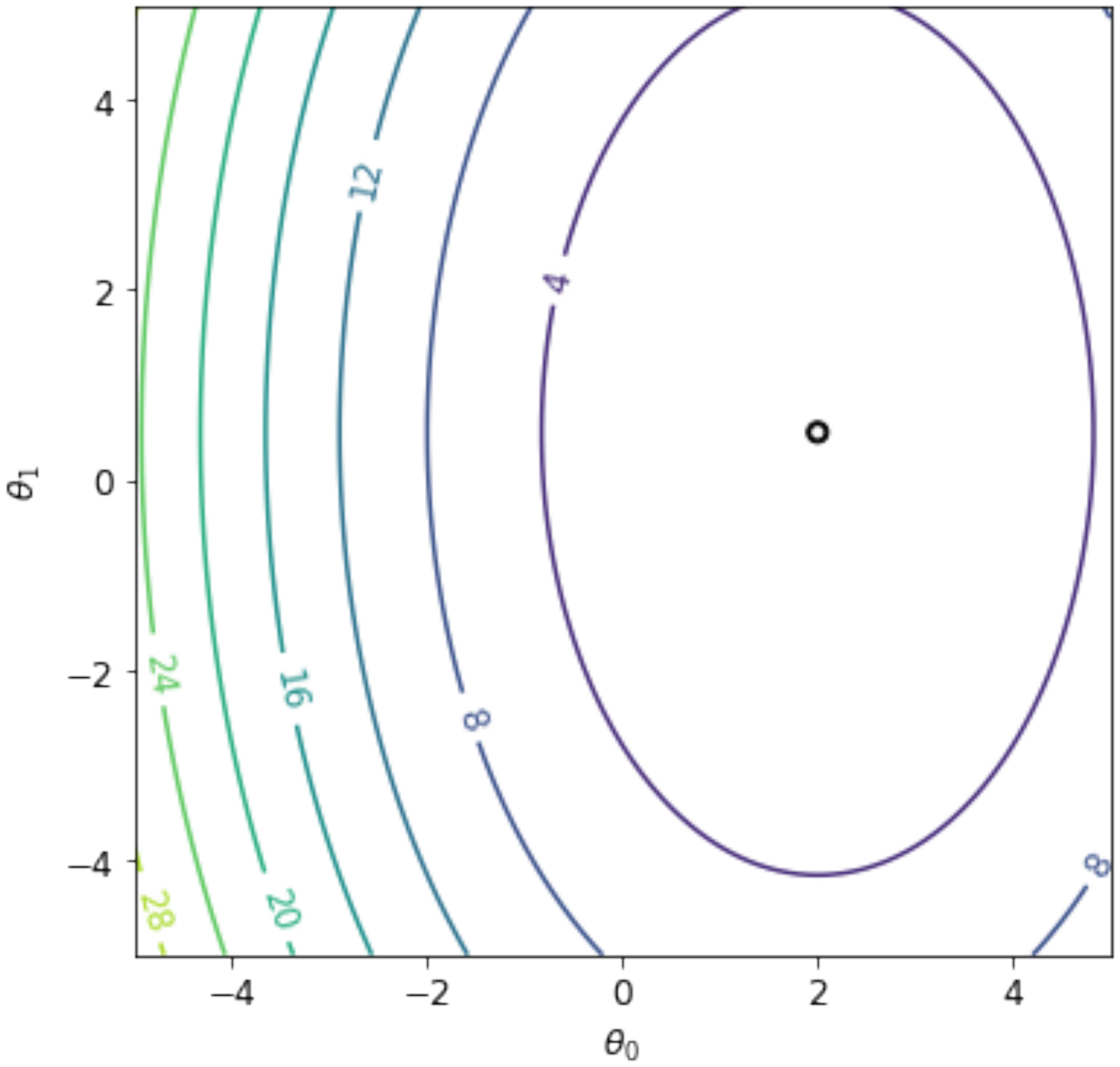
$$\theta_d := \theta_d + \alpha\left(t^{(n)} - h_\theta(x^{(n)})\right)x_d^{(n)}; \forall d \in D$$

Repeat until convergence

Makes progress for each training sample

Mini-batch Stochastic Gradient Descent considers a subset of the training set for each update (so-called mini-batch)

Cost function



POLYNOMIAL CURVE FITTING

Training set: N observations of $\mathbf{x} = (x_1, \ldots, x_N)^T$ and $\mathbf{t} = (t_1, \ldots, t_N)^T$

- Ground truth: $t = sin(2\pi x)$, but (Gaussian) noise present
- Many data sets have an underlying regularity, but observations are corrupted by random noise

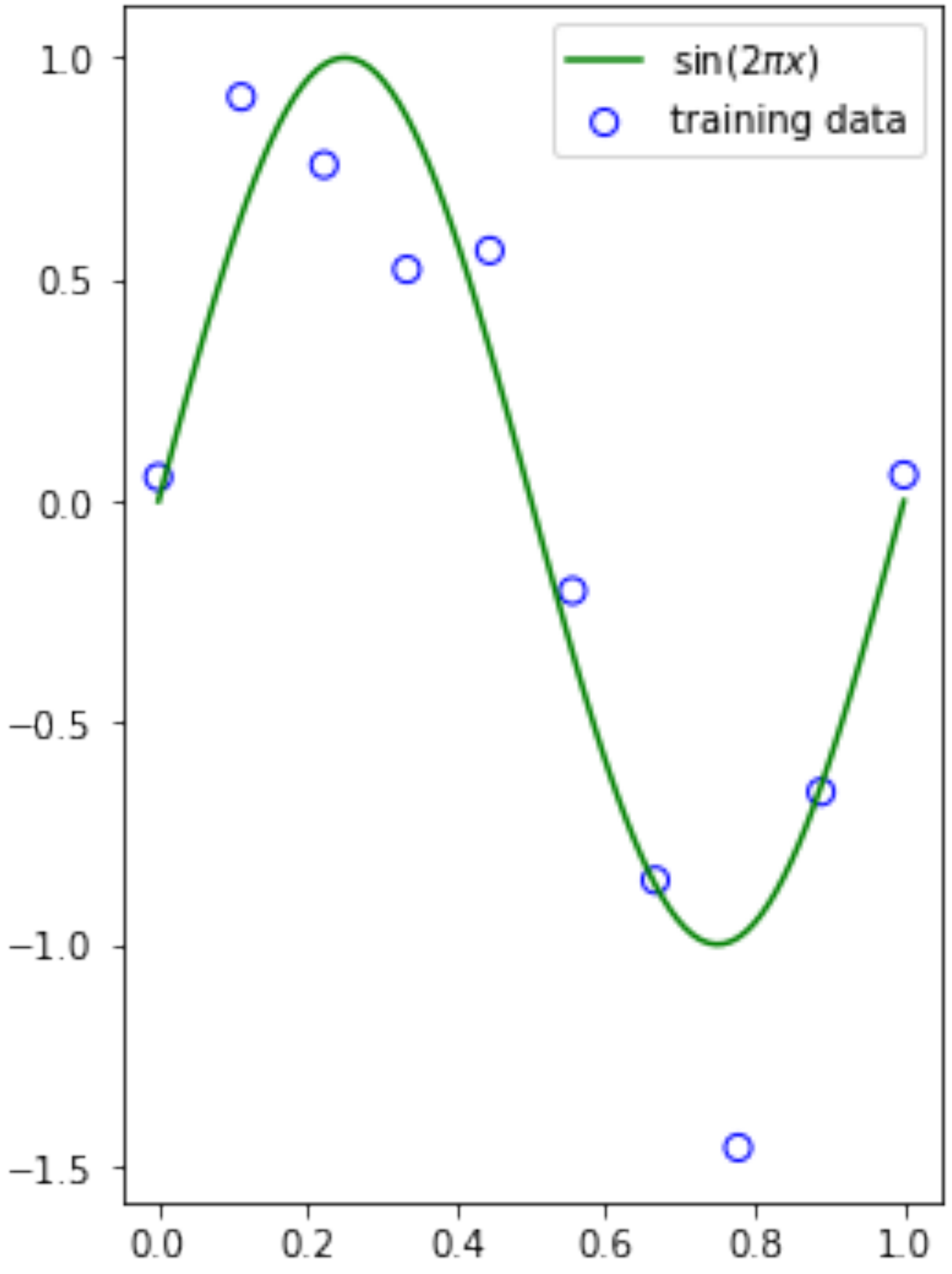
Objective: make good predictions $\hat{y}$ of new values $\hat{x}$

- *Generalize* from a finite data set

Model: polynomial function of order of M

$$h(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{m=0}^{M} w_m x^m$$

- Although $h(x, \mathbf{w})$ is a nonlinear function of x, it is a linear function of the coefficients $\mathbf{w}$ => linear model



FITTING

Determine the coefficients $\mathbf{w}$ by fitting to N training samples

Minimize error function $E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h(x_n, \mathbf{w}) - t_n\right)^2$

Again: quadratic function of coefficients $\mathbf{w}$

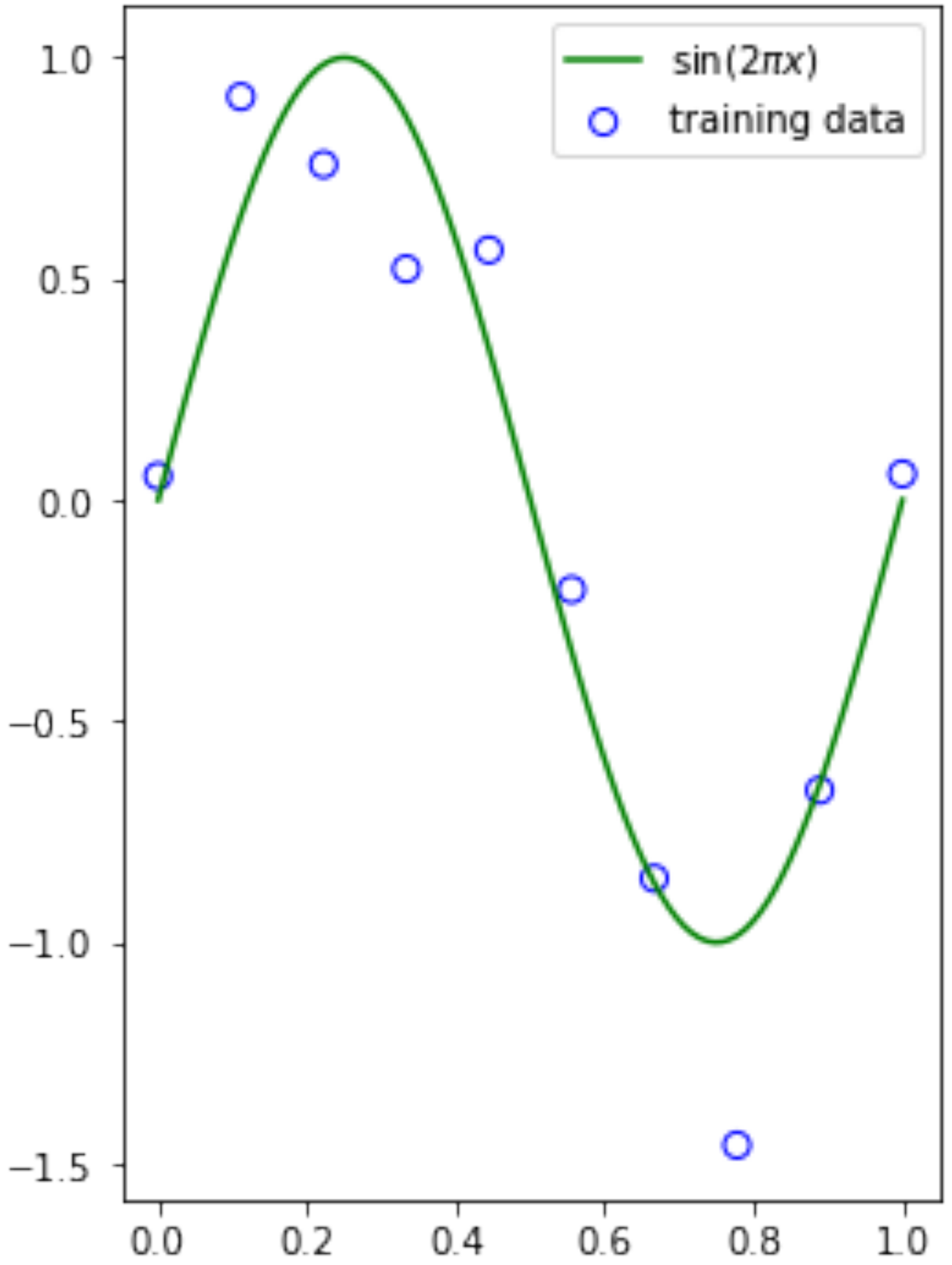
=> partial derivates (with respect to the coefficients) are linear in the elements of $\mathbf{w}$

=> unique solution $\mathbf{w}^*$

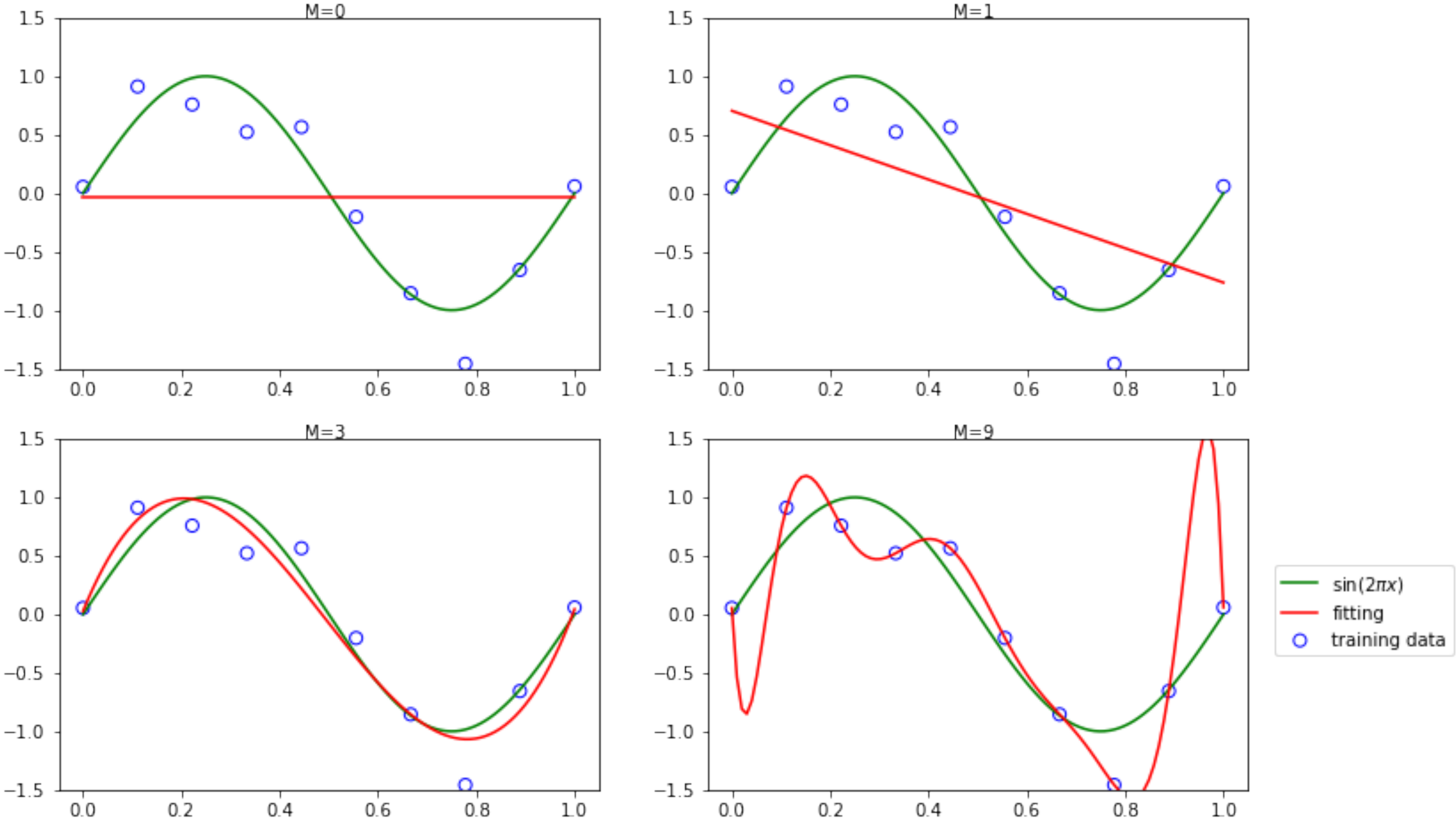
But what about order M?

$$h(x, \mathbf{w}) = \sum_{m=0}^{M} w_m x^m$$

=> model selection



MODEL SELECTION



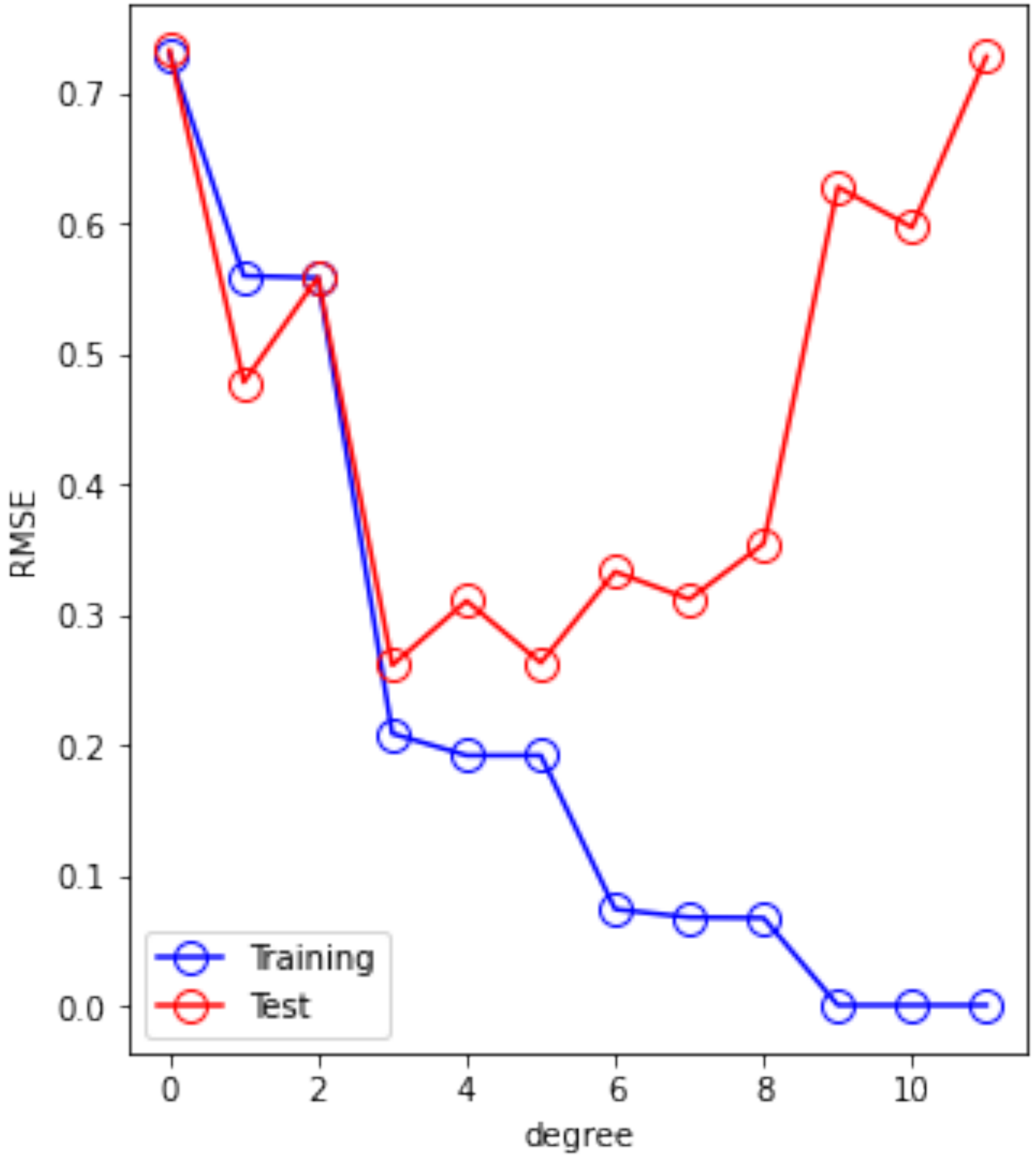
GENERALIZATION AND OVERFITTING

Good generalization: making accurate predictions for new (unseen) data

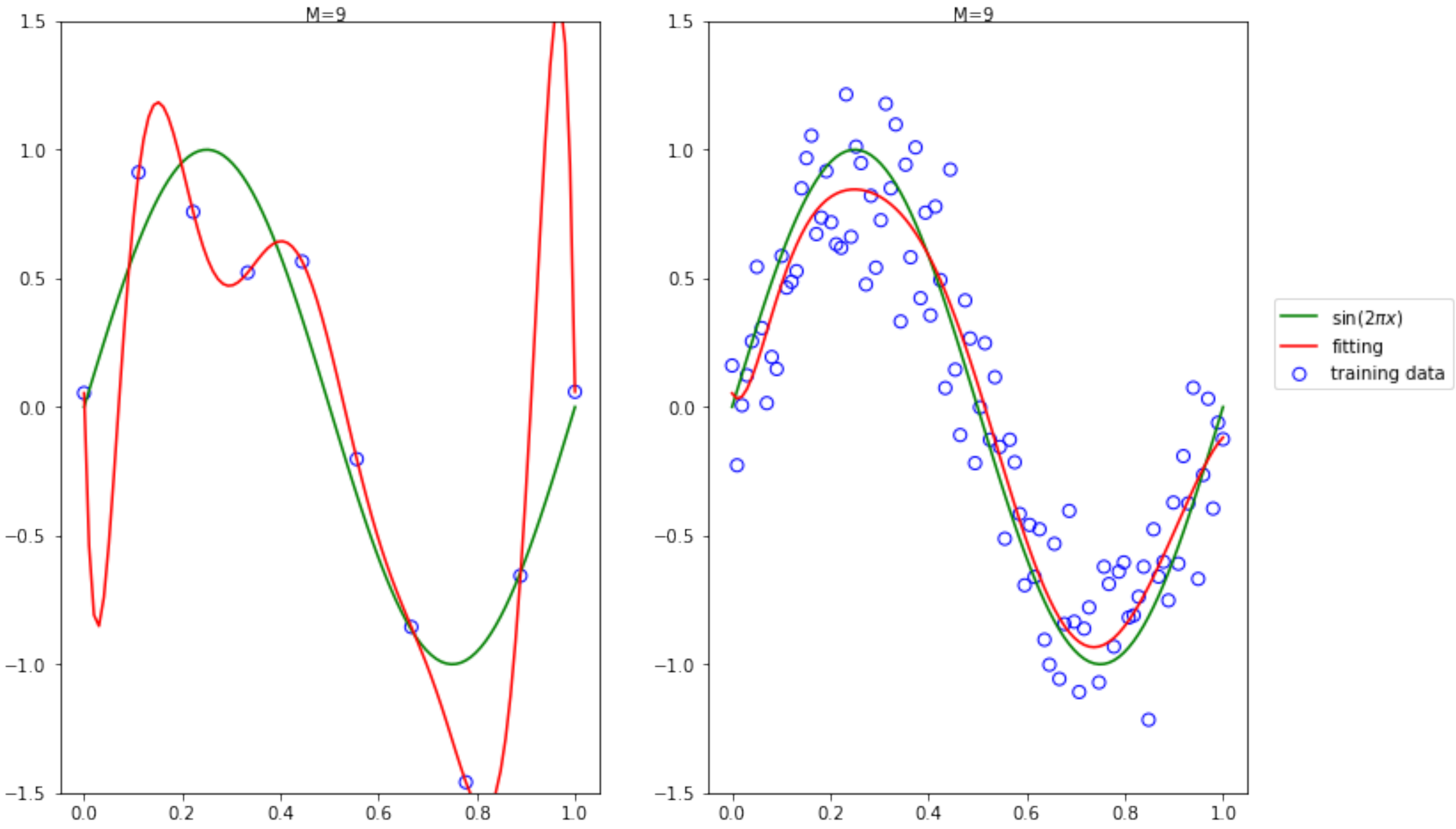
- Test set: here generated like the training set
- Usually: split data set into training set and test set, don't show test set during training time

Identify overfitting

- Training error: $E(\mathbf{w}^*)$ for the training set
- Test error: $E(\mathbf{w}^*)$ for the test set



MODEL SELECTION DEPENDS ON DATA SET SIZE



N=10 N=100

REGULARIZATION

Regularization can control overfitting by adding a penalty term to the error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(h(x_n, \mathbf{w}) - t_n\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|$$

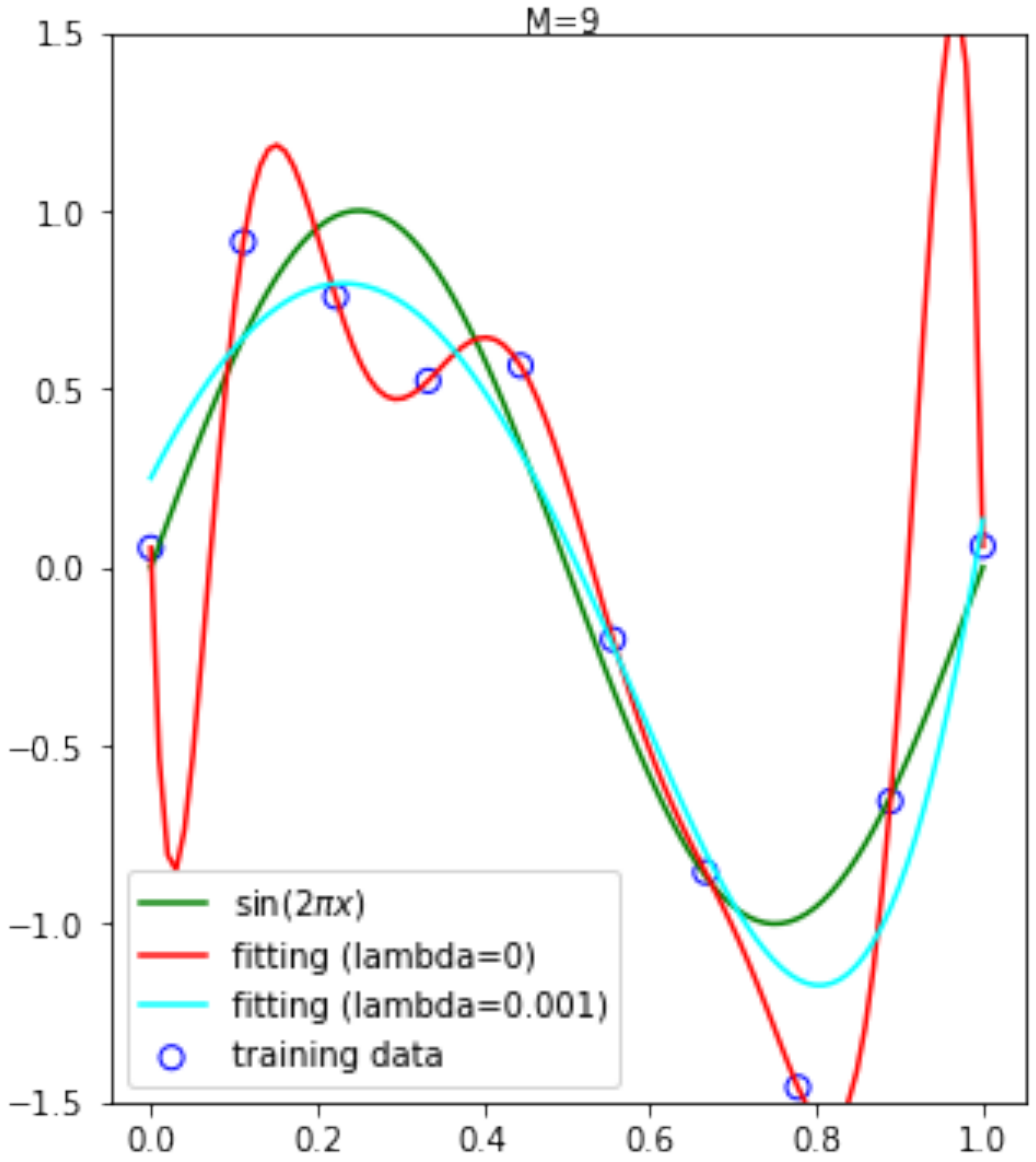
where $\|\mathbf{w}\| = \mathbf{w}^T\mathbf{w}$

λ governs the relative importance of the regularization term

Such shrinkage methods reduce the value of the coefficients

Quadratic regularizer: *ridge regression* or *weight decay* or *L2 regularization*

Validation set to optimize either M or λ



REGULARIZATION

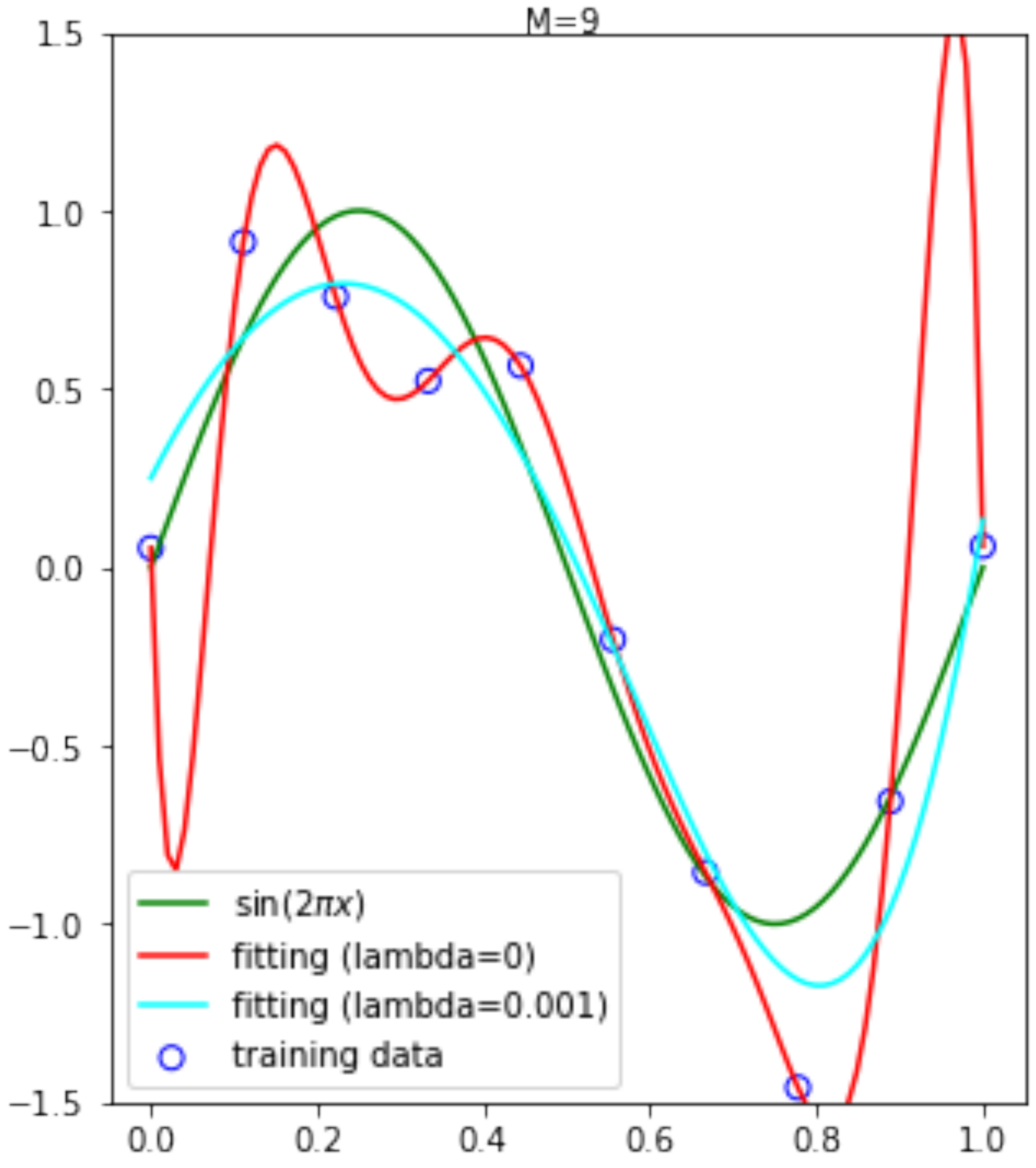
Regularization refers to a set of different methods that lower the complexity of a neural network model during training to prevent overfitting

- Many regularization approaches are based on limiting the capacity of models
  - Neural networks, linear regression, polynomial regression, etc.
  - A form of regression that shrinks (constrains, regularizes) the coefficient estimates (weights, not biases) towards zero
  - Prevents the learning of complex models to avoid the risk of overfitting
  - Penalize the flexibility of a model
- Trading increased bias for reduced variance
  - Profitable trade: reducing variance significantly while not overly increasing the bias
- Regularizer examples: shrinkage methods (capacity reduction), early stopping, dropout, weight initialization techniques, and batch normalization

MODEL PARAMETER ANALYSIS

M Reg.	1 no	2 no	4 no	9 no	9 L2
Weights	-3.6E-02	7.8E-01	1.1E-02	-3.2E+01	1.8E-01
		-1.6E+00	9.3E+00	5.5E+02	5.3E+00
			-2.7E+01	-2.7E+03	-1.0E+01
			1.7E+01	4.8E+03	-4.3E+00
				2.0E+03	1.8E+00
				-1.9E+04	4.5E+00
				2.8E+04	4.4E+00
				-1.8E+04	2.4E+00
				4.2E+03	-6.1E-01
					-4.2E+00

Overfit (often?) correlates with large weights

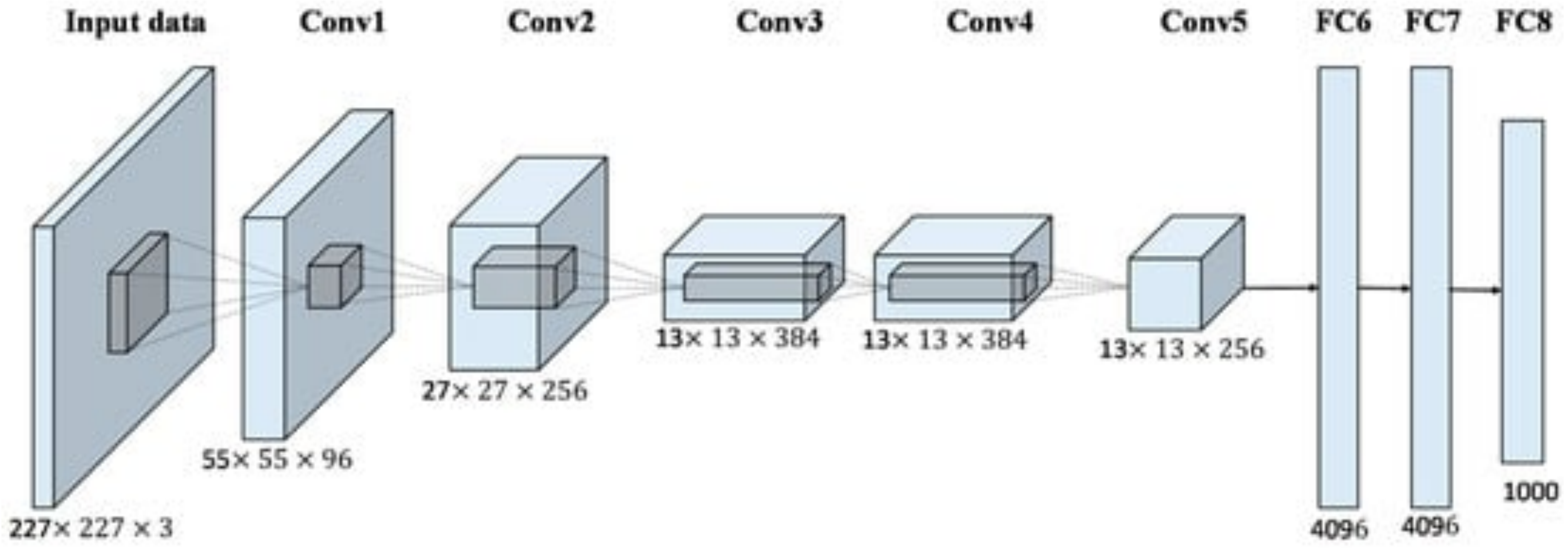


AN INTUITION

Assume x_1 and x_2 are equal

Assume either one slightly changes

ARTIFICIAL NEURAL NETWORKS



ARTIFICIAL NEURAL NETWORKS (ANNS)

Kind of inspired by biology

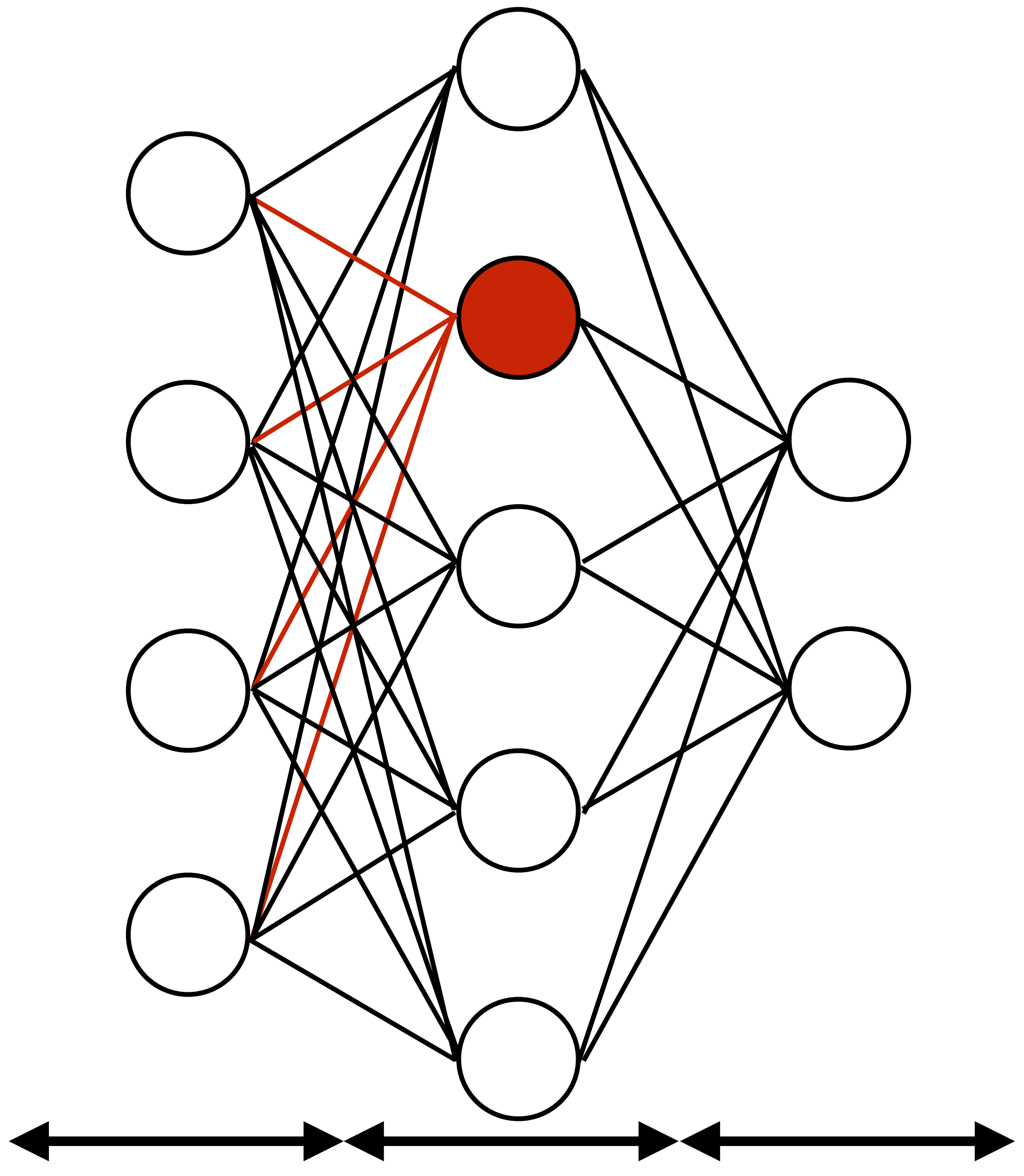
- Term “biologically inspired” is often a complaint
- != spiking neural networks
- c.f. “non-differentiable”

More complex problems require more complex models

- Informal term of “model capacity”
- Curse of dimensionality: one pixel = one dimension

“Universal approximation theorems imply that neural networks can represent a wide variety of interesting functions when given appropriate weights”

Deep neural networks (DNNs) = increasing number of hidden layers



MULTI-LAYER PERCEPTRON (MLP)

E.g.: MNIST: 28x28 images in 10 classes => MLP with 28x28 inputs ($\mathbf{x}$) & 10 outputs ($\mathbf{y}$)

For neuron k of a given layer

$$y_k = f\Big(\sum_j (w_{k,j} \cdot x_j) + b_k \Big)$$

f: non-linear function (sigmoid, reLU, …)

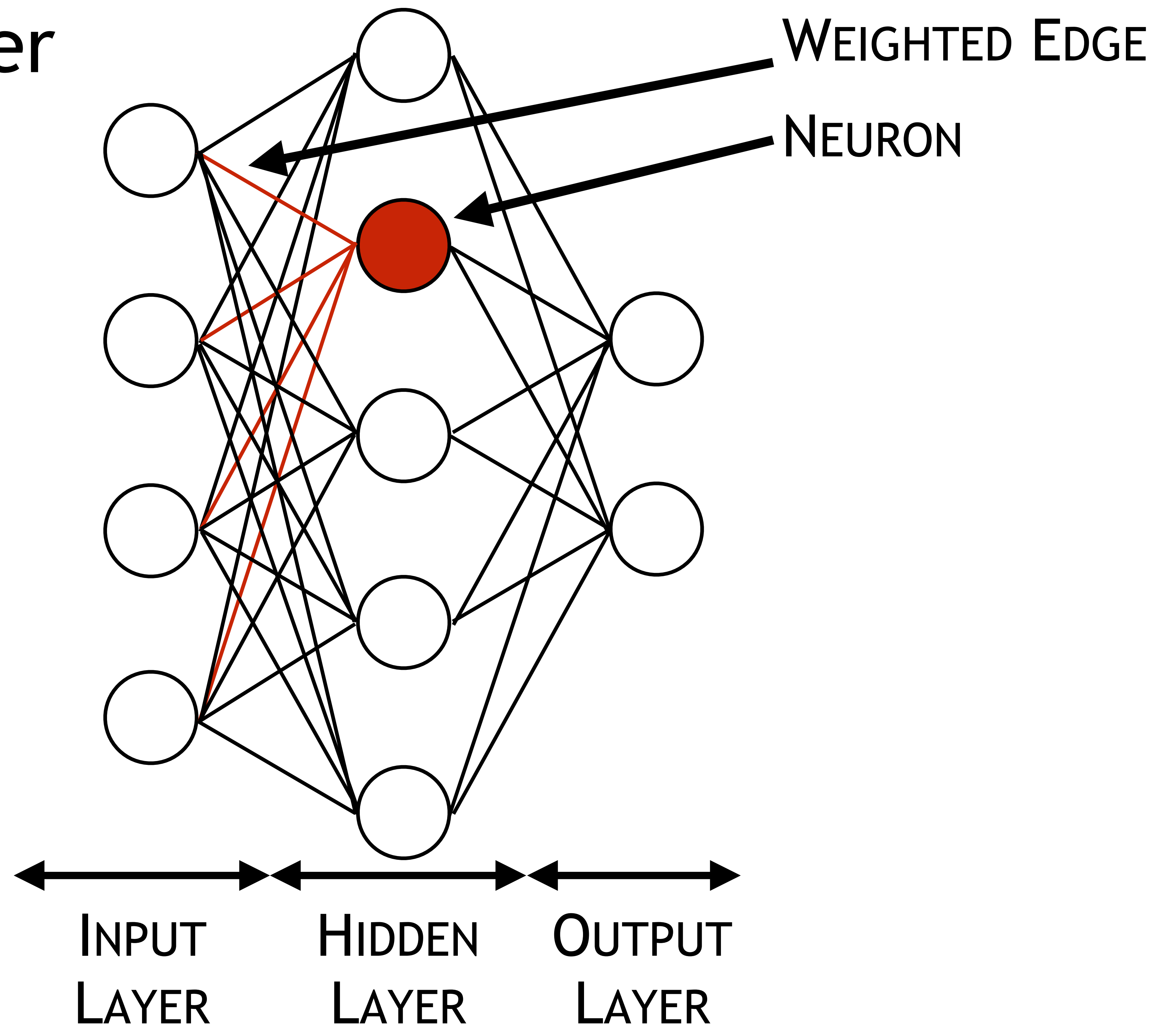
$\mathbf{W}$: weight matrix

$\mathbf{x}$: activation vector

$\mathbf{b}$: bias vector (hidden)

Vector notation for layer l

$$\mathbf{x}_l = f(\mathbf{W}_l \cdot \mathbf{x}_{l-1})$$



VECTOR AND MATRIX NOTATION

Matrix $\mathbf{W}$, composed of elements $w_{k,j}$

Matrix = bold uppercase

Matrix element $w_{k,j}$ has row k, column j

Vector $\mathbf{x}$, composed of elements x_i

Vector = bold lowercase

Vectors are vertical, use $\mathbf{x}^T$ for horizontal vectors

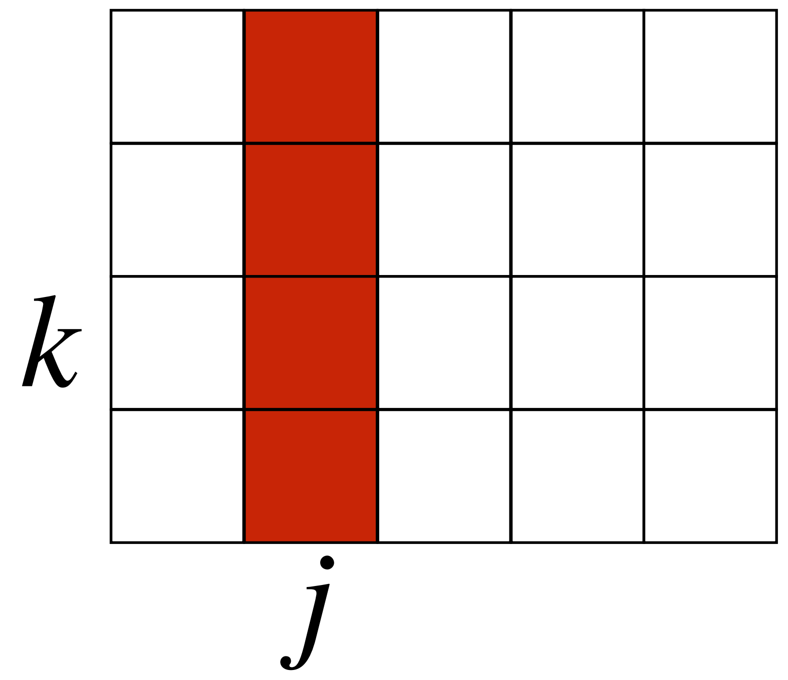
Matrix-vector multiplication

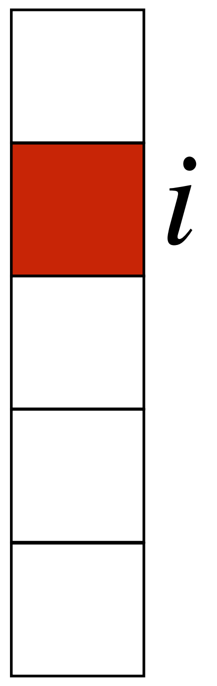
Length of the vector equals the number of columns of the matrix

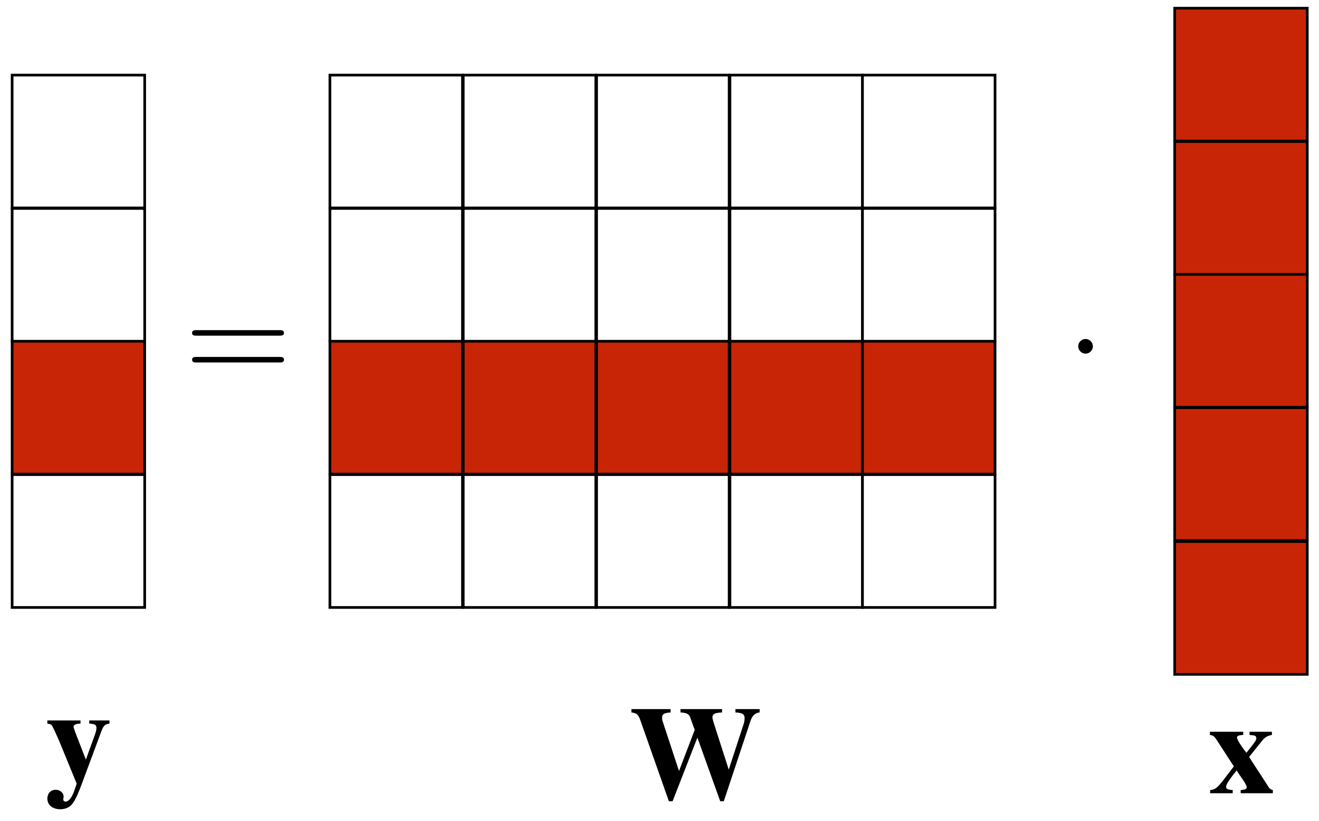
$y_k = \sum_j (w_{k,j} \cdot x_j)$, resp. $\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$

Vector-vector multiplication (dot product)

$a = \sum_j (b_j \cdot c_j)$, resp. $a = \mathbf{b} \cdot \mathbf{c}^T = \mathbf{c} \cdot \mathbf{b}^T$







MULTI-LAYER PERCEPTRON (MLP)

E.g.: MNIST: 28x28 images in 10 classes => MLP with 28x28 inputs ($\mathbf{x}$) & 10 outputs ($\mathbf{y}$)

For neuron k of a given layer

$$y_k = f\Big(\sum_j (w_{k,j} \cdot x_j) + b_k\Big)$$

f: non-linear function (sigmoid, reLU, …)

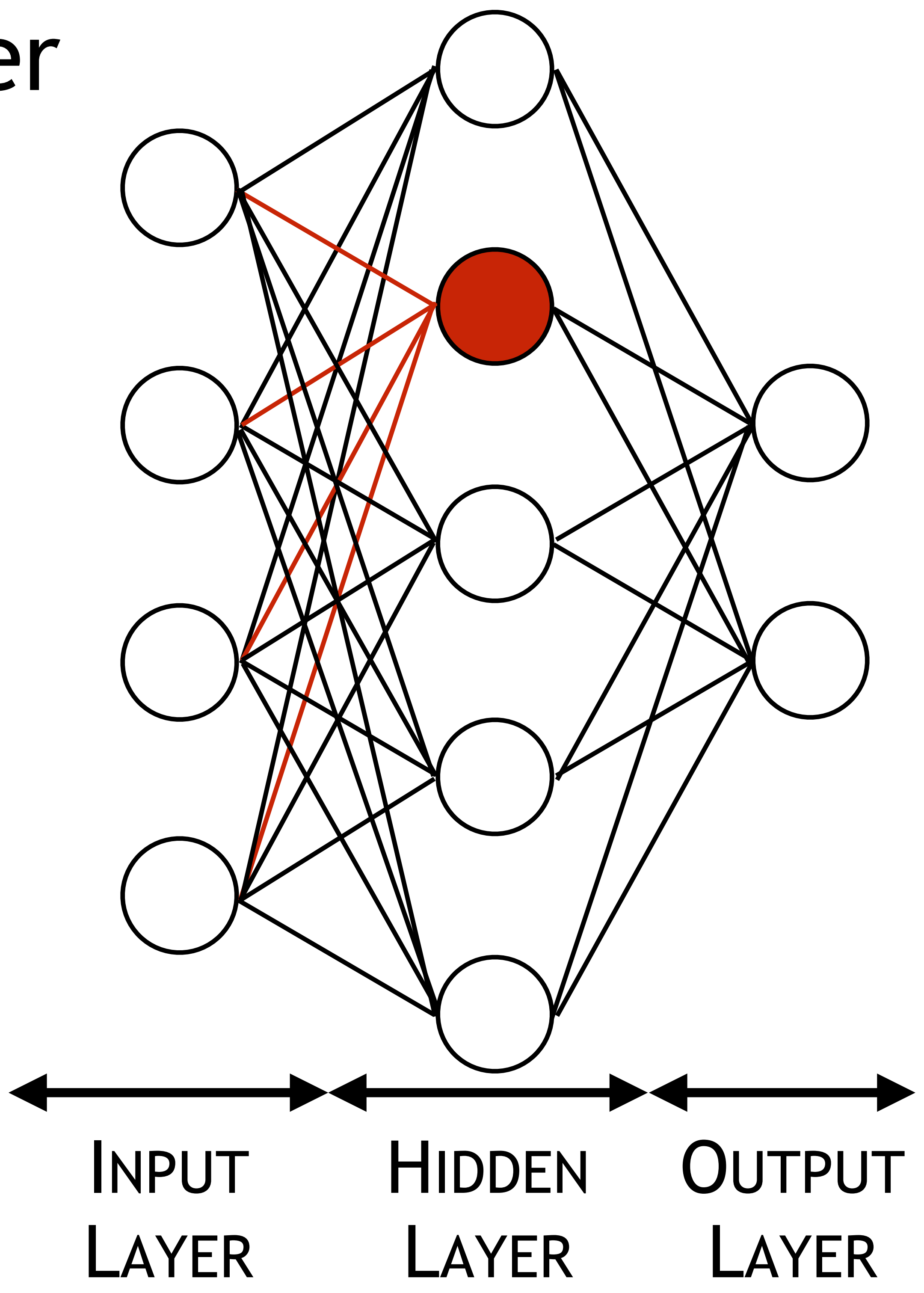
$\mathbf{W}$: weight matrix

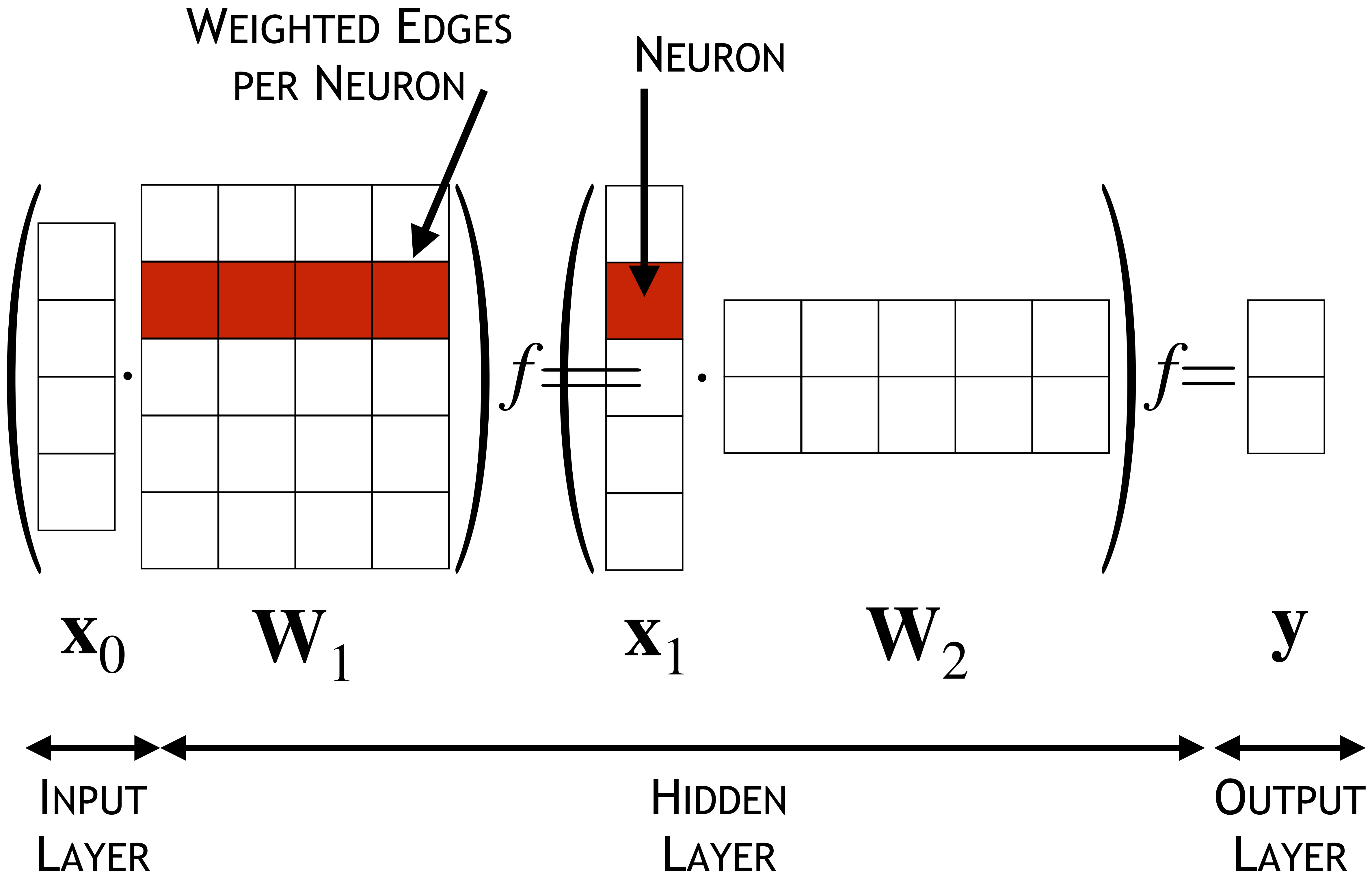
$\mathbf{x}$: activation vector

$\mathbf{b}$: bias vector (hidden)

Vector notation for layer l

$$\mathbf{x}_l = f(\mathbf{W}_l \cdot \mathbf{x}_{l-1})$$





FORWARD PROP ON ONE SLIDE

(Deep) Neural Networks: L stacked processing units, where each unit computes an activation function

$x_l = f(\mathbf{W}_l \oplus \mathbf{x}_{l-1} + \mathbf{b}_l)$, for nonlinear activation function $f(\cdot)$, linear operation $\oplus$, and weight matrix $\mathbf{W}$, input activations $\mathbf{x}$, and bias $\mathbf{b}$ of layer l

Bias vector $\mathbf{b}$ is usually encoded in the weight matrix $\mathbf{W}$ by introducing another activation element which is fixed to (e.g., $x_0 = 1$)

Then a complete MLP with L layers is

$$\mathbf{y}(\mathbf{W}, \mathbf{x}_0) = \mathbf{x}_L = f(\mathbf{W}_L \oplus f(\mathbf{W}_{L-1} \oplus f(\ldots \oplus f(\mathbf{W}_1 \oplus \mathbf{x}_0))\ldots)$$

Reminder: *“Universal approximation theorems imply that neural networks can represent a wide variety of interesting functions when given appropriate weights”*

EXAMPLE NONLINEARITIES

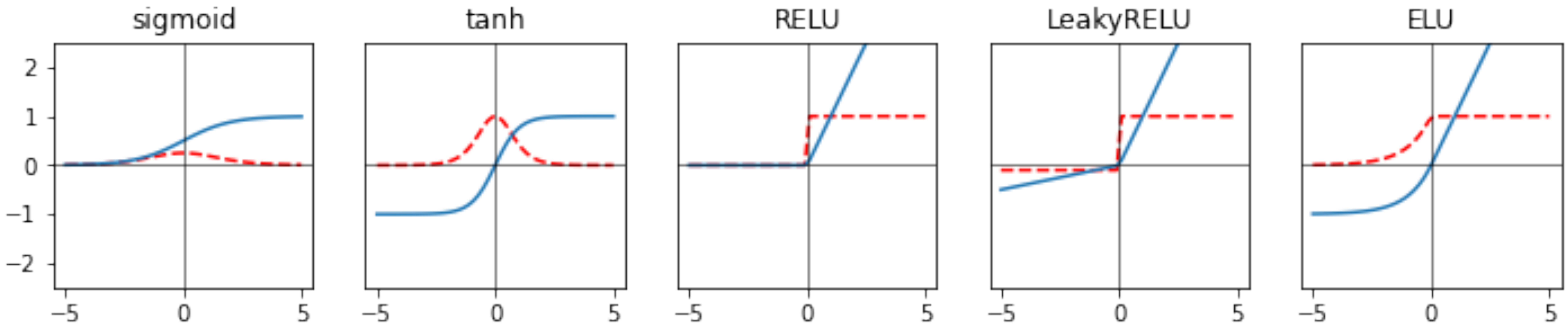
sigmoid: $f(x) = \frac{1}{1+e^{-x}}$ => output in range $[0,1]$

tanh: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ => output in range $[-1,1]$

ReLU: $f(x) =$ max$(x,0)$ => no negative output

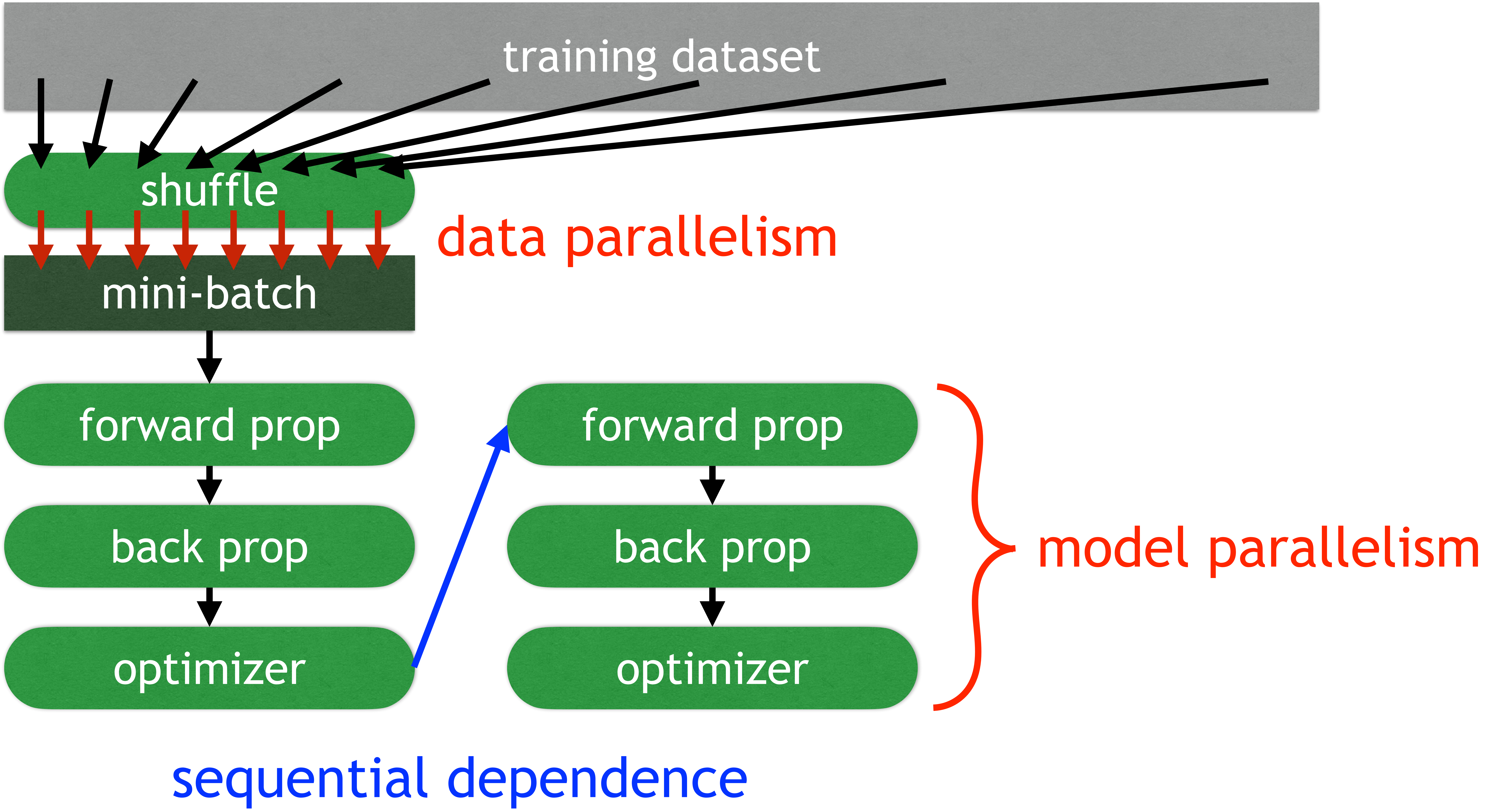
LeakyReLU: $f(x) = \begin{cases} x; x \geq 0 \\ \alpha x; x < 0 \end{cases}$ => no clamping to zero for negative inputs

ELU: $f(x) = \begin{cases} x; x \geq 0 \\ e^x - 1; x < 0 \end{cases}$ => smoother gradient



Basically any non-linear function can be used

TRAINING OF DEEP NEURAL NETWORKS



BACK PROP ON ONE SLIDE

Data set containing N input-target pairs: $\mathscr{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$

Training ANNs: adjust randomly initialized weights $\mathbf{W}$ to solve a given task by minimizing a loss function $\mathscr{L}$ using gradient-based optimization

$$\mathscr{L}(\mathbf{W}; \mathscr{D}) = \sum_{n=1}^{N} l(y(\mathbf{W}, \mathbf{x}_n), t_n) + \lambda r(\mathbf{W});$$

based on a data term l that penalizes wrong prediction (error function); and

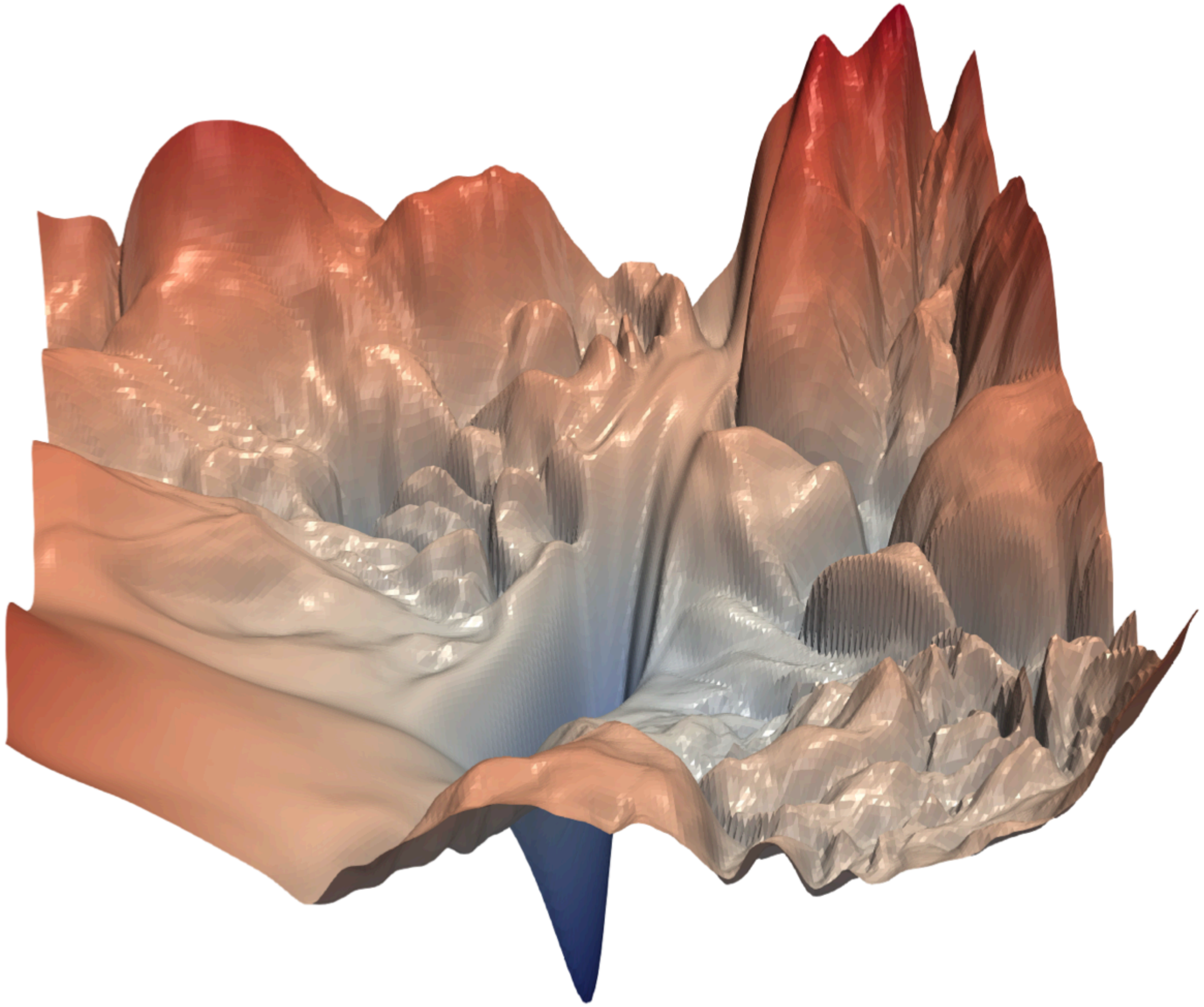
for a regularizer $r(\mathbf{W})$ such as ℓ^1–norm or ℓ^2–norm and a trade-off hyperparameter λ

Backpropagation: compute gradient for input-target pair and minimize the loss function by iteratively calculating

$\mathbf{W} := \mathbf{W} - \eta \nabla_{\mathbf{W}} \mathscr{L}(\mathbf{W}; \mathscr{D})$, for $\nabla_{\mathbf{x}} = \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n} \right)$ and learning rate η

Key operations: chain rule of calculus, partial derivative and all-reduce

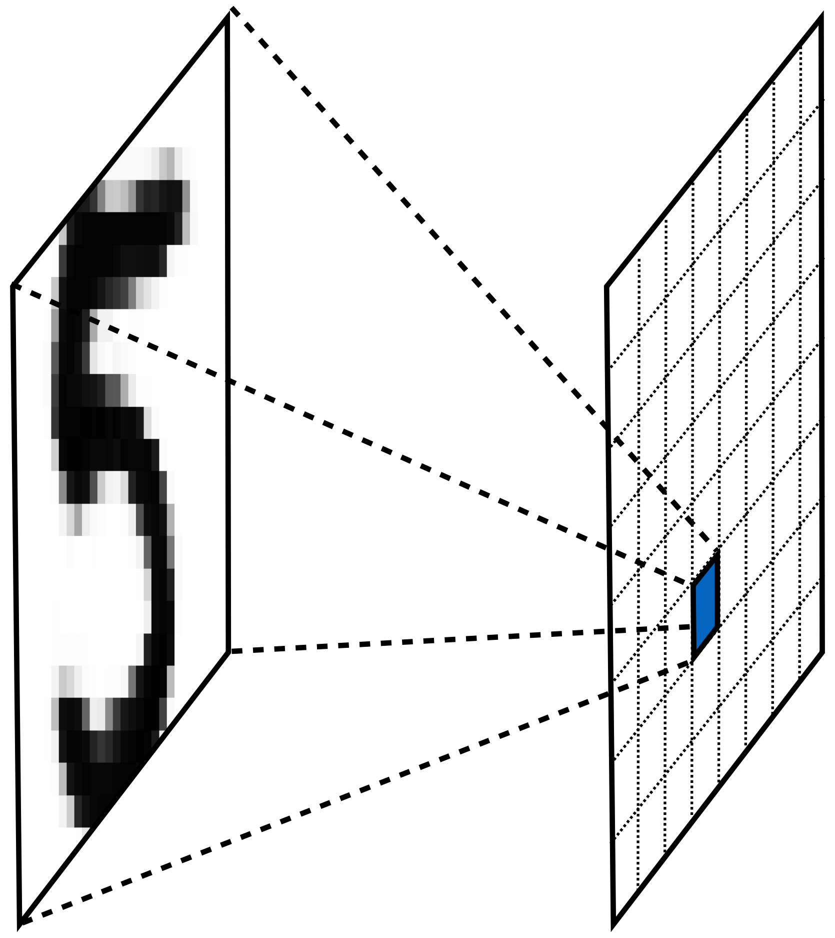
EXAMPLE LOSS LANDSCAPES IN MODERN ANNS



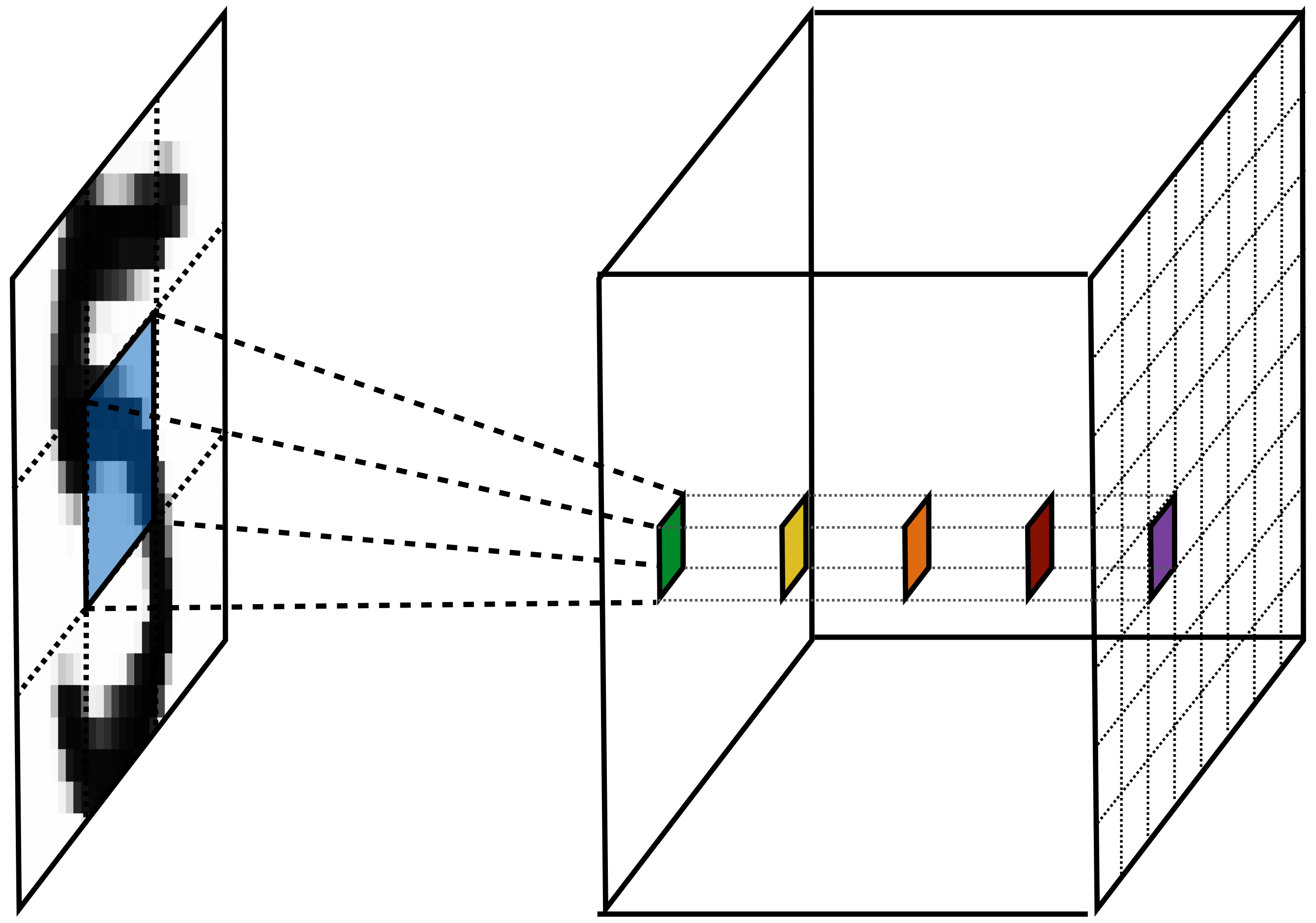
Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. 2018. Visualizing the loss landscape of neural nets. In 32nd International Conference on Neural Information Processing Systems (NIPS'18)

CONVOLUTIONAL LAYERS

CONVOLUTIONAL LAYERS



Fully-connected layer



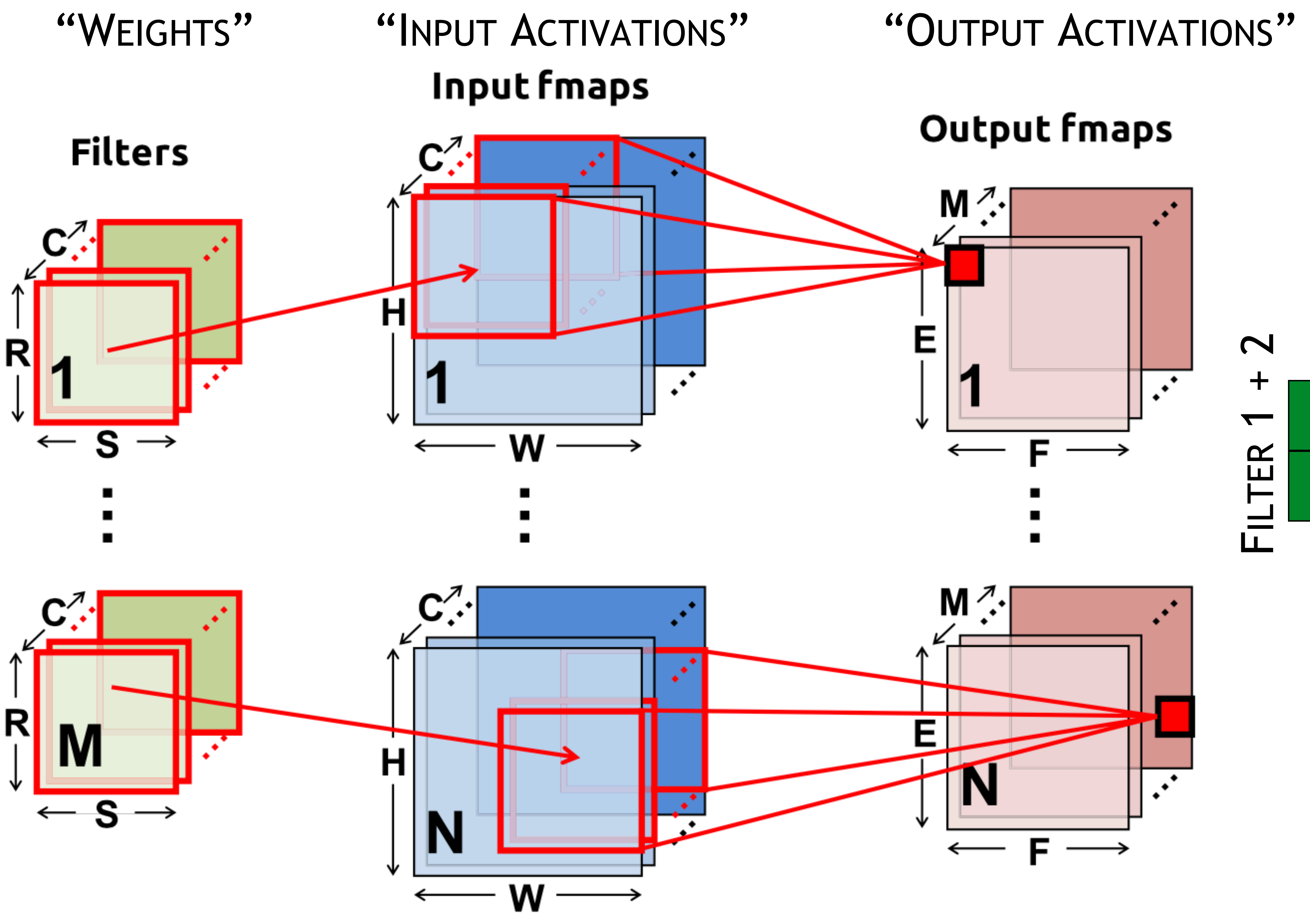
Convolutional layer

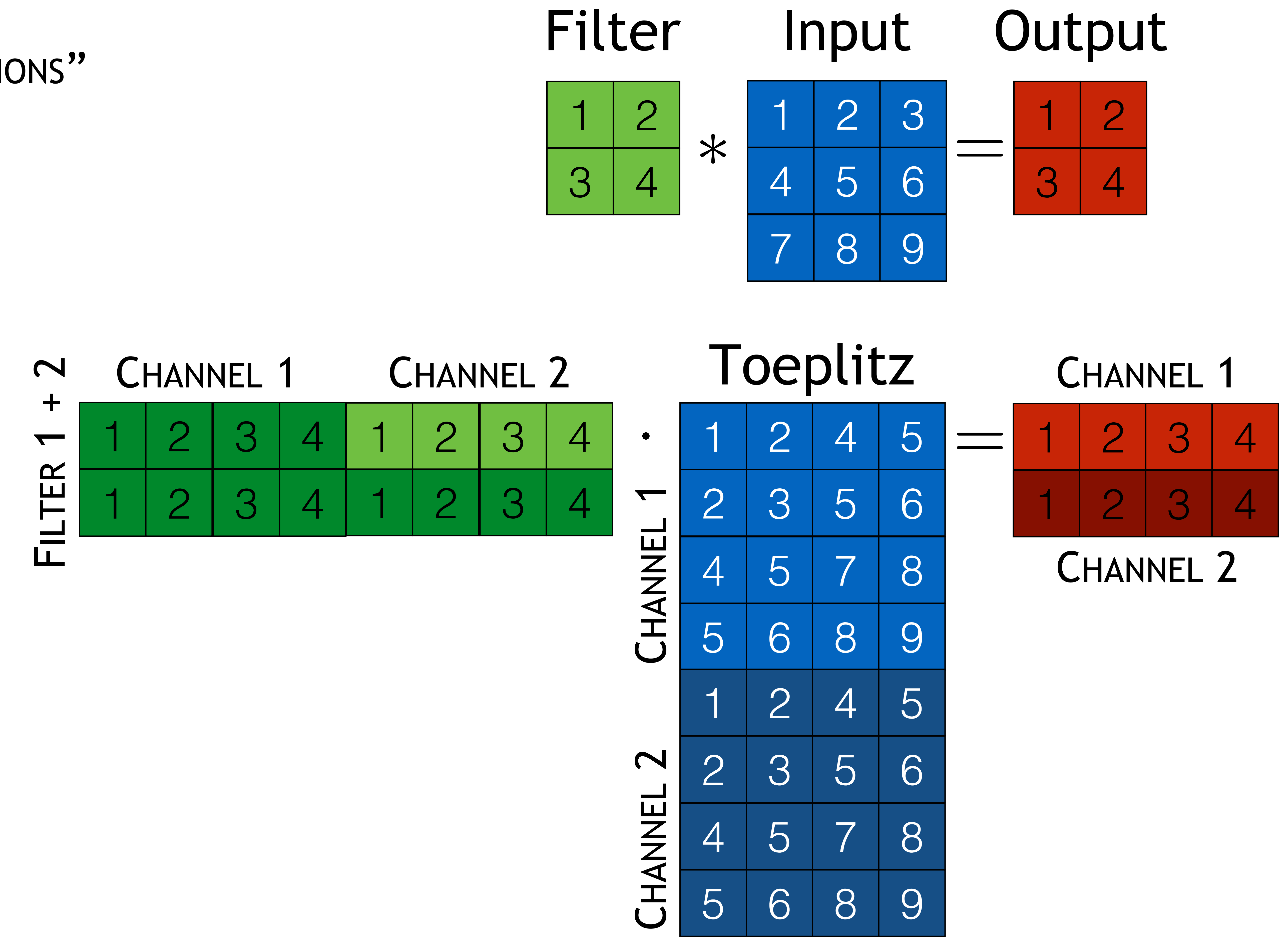
Receptive field: spatially local correlation (patches)

Shared weights: as each filter is applied to all patches of the input

3D layers: “depth” of one layer is the number of filters (kernels) learned

CONVOLUTION OPERATION

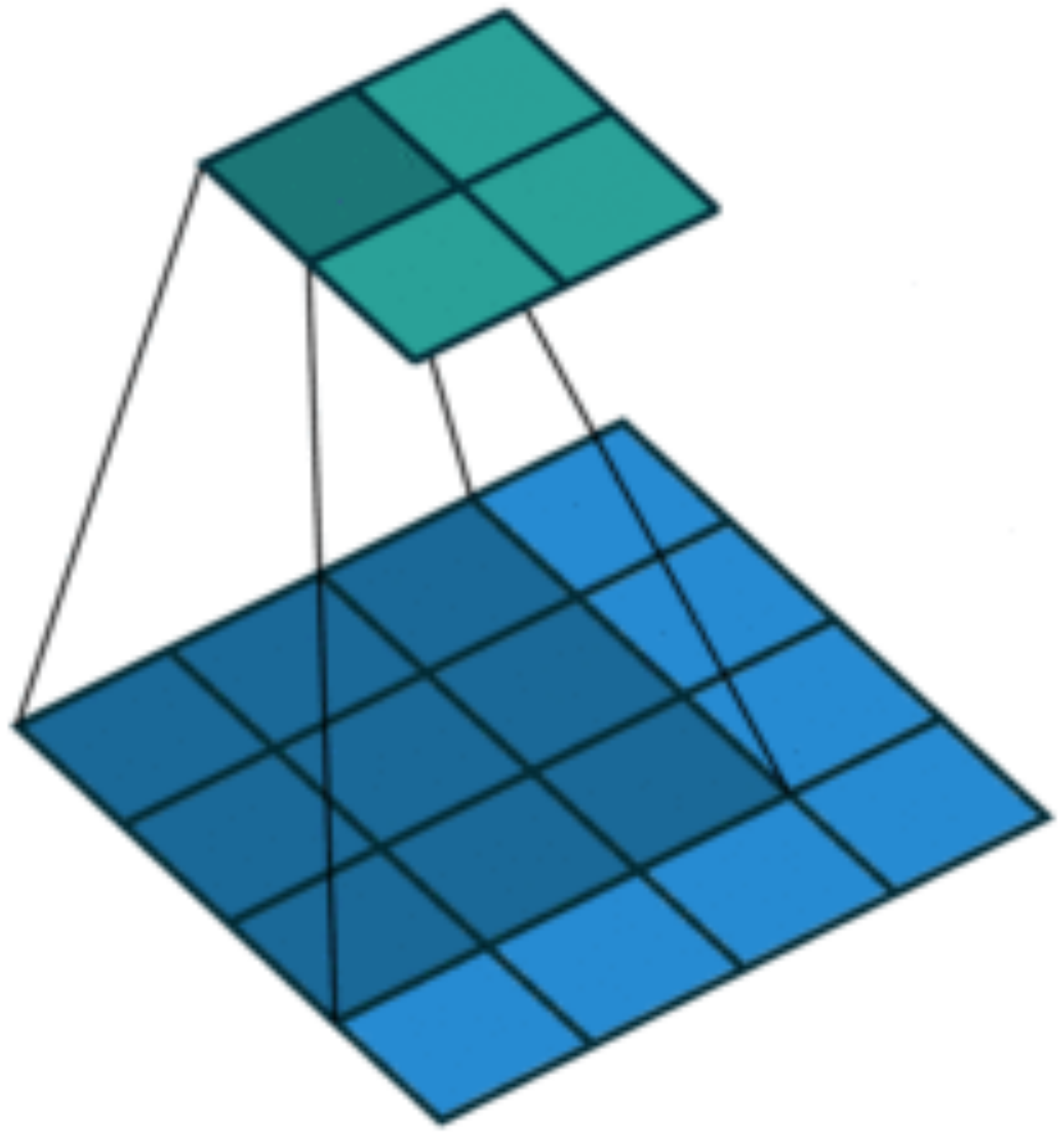




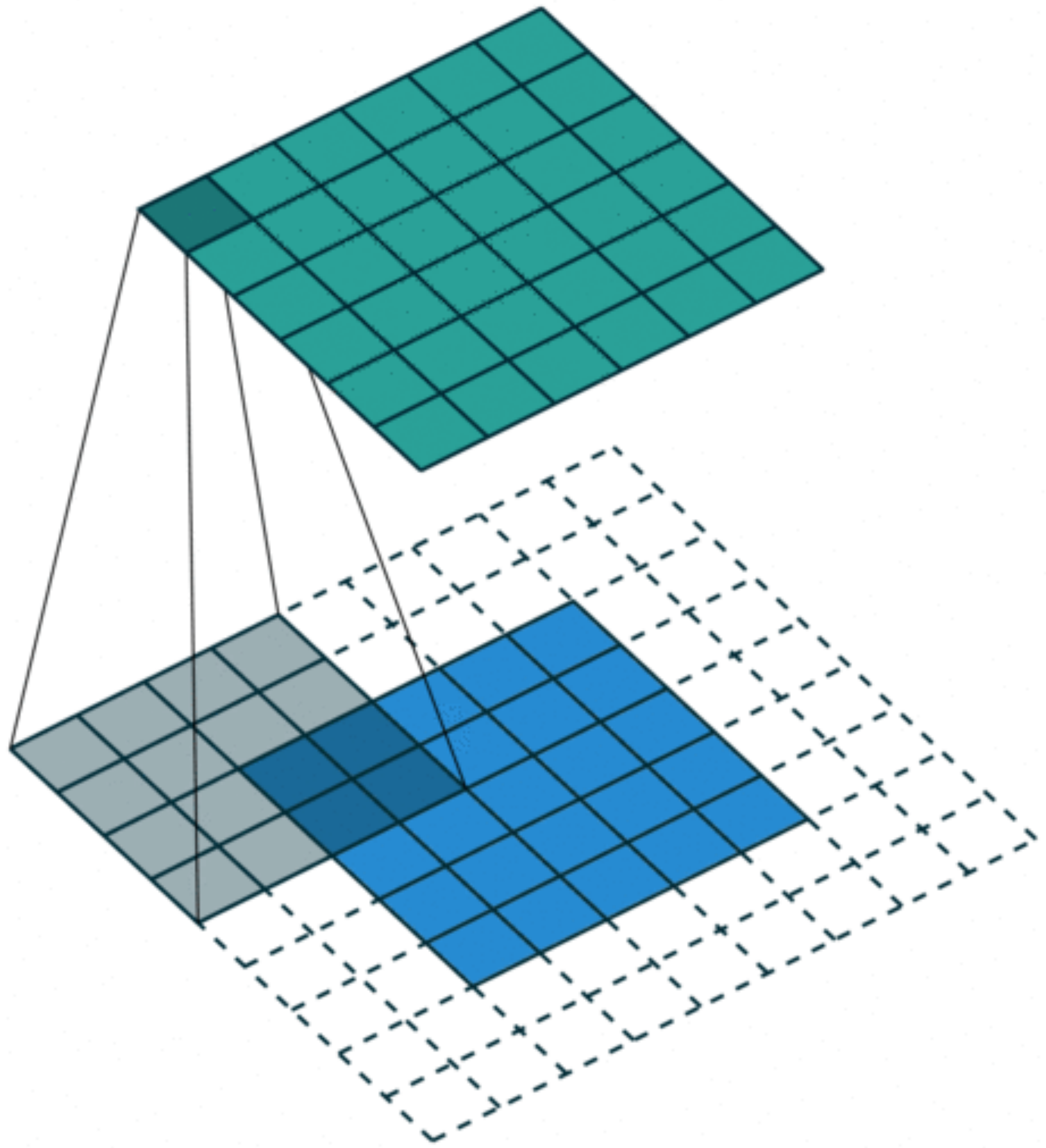
Convolutions increase data reuse, but are usually still mapped to matrix operations

V. Sze, T.-J. Yang, Y.-H. Chen, J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, 2017.

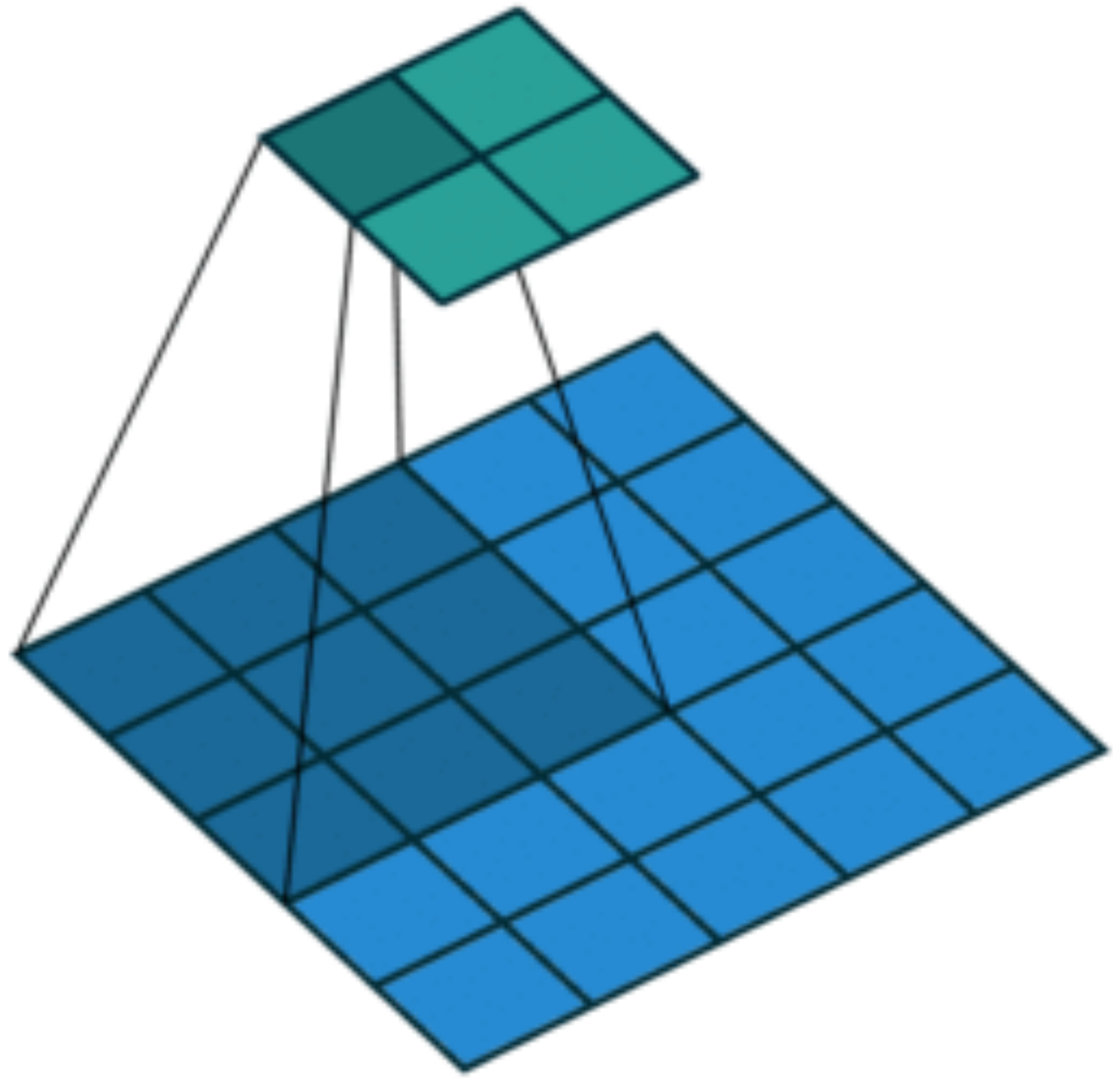
CONVOLUTION EXAMPLES



No padding,
No strides

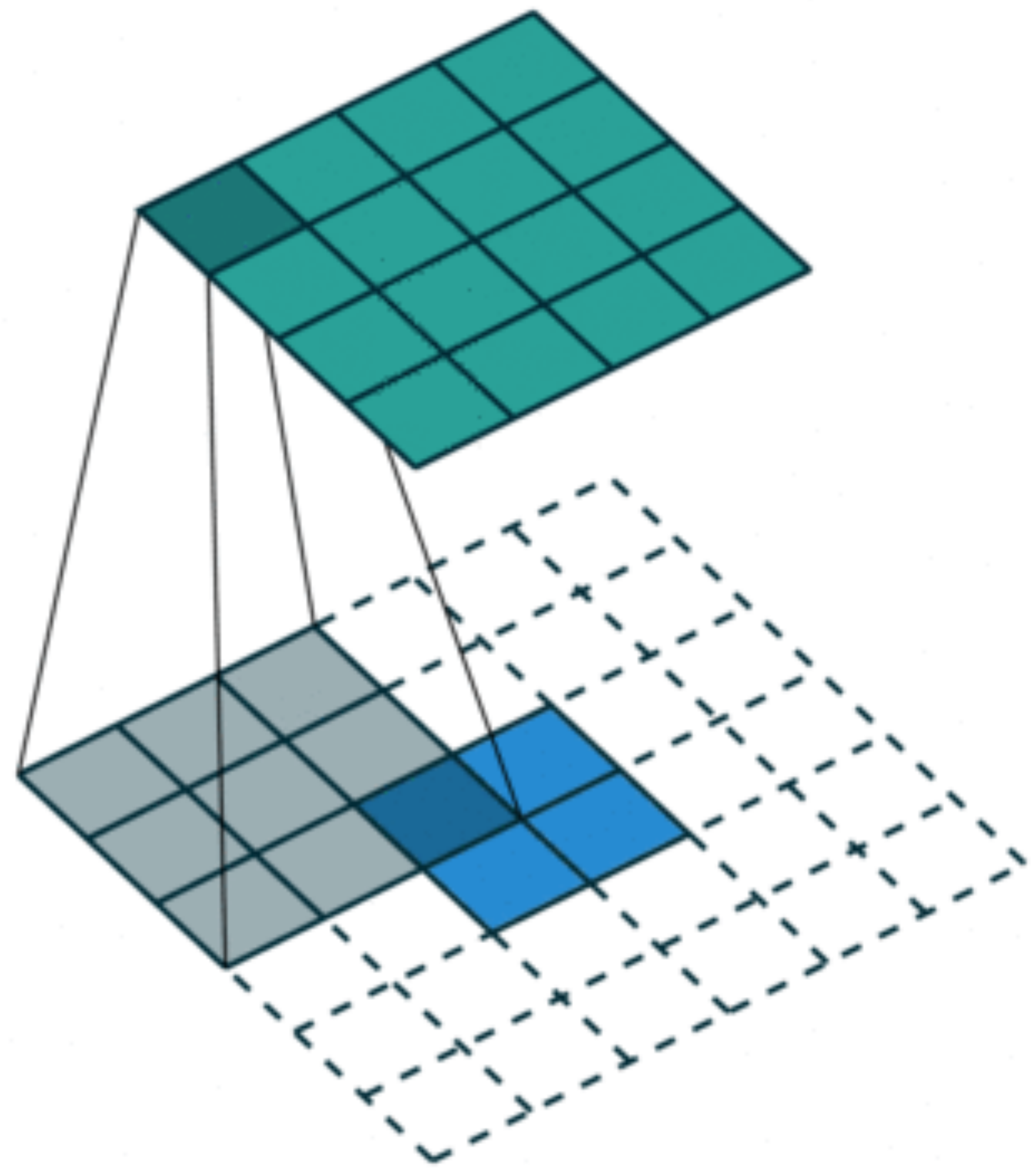


Padding,
No strides

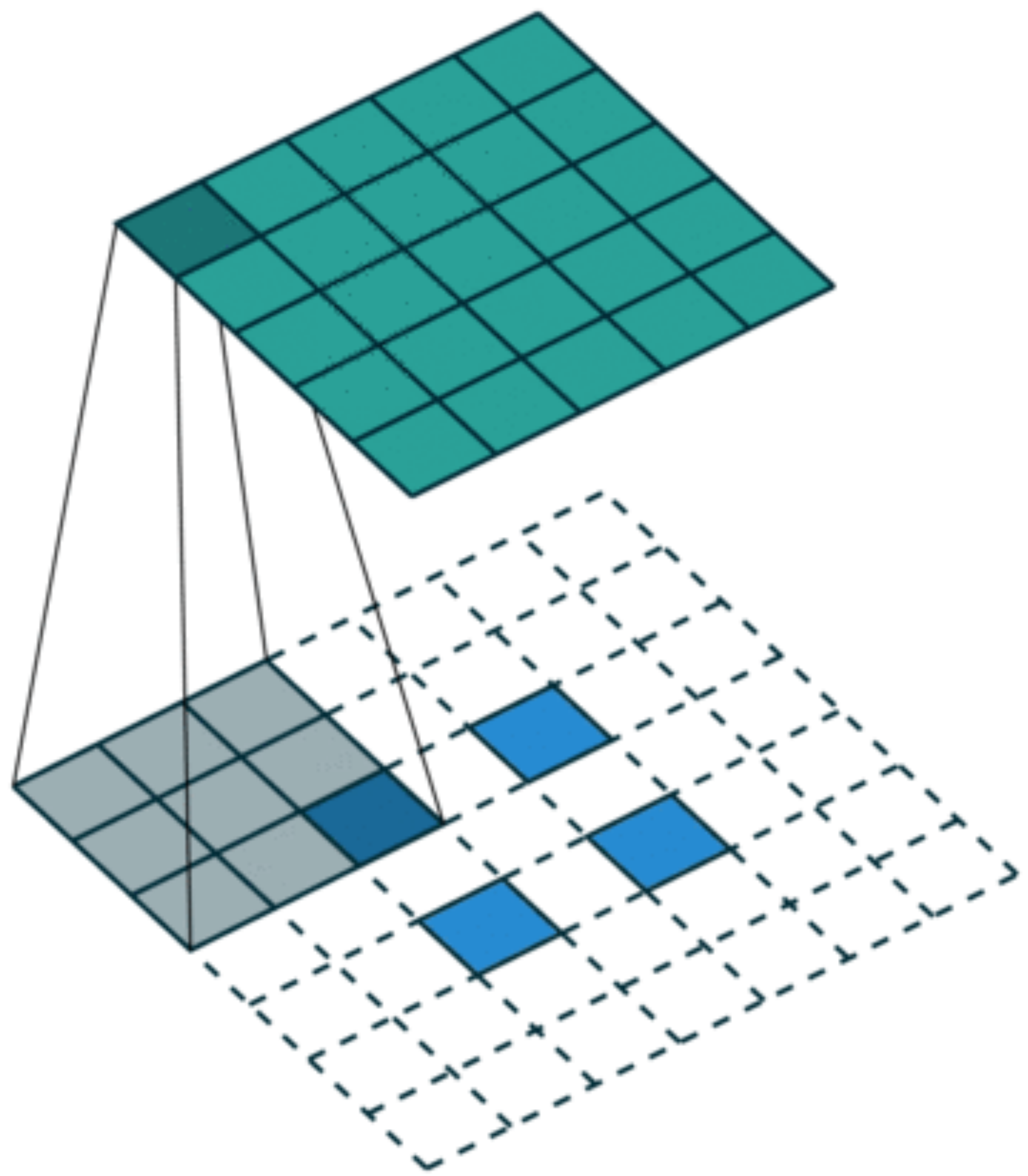


No padding,
Strides

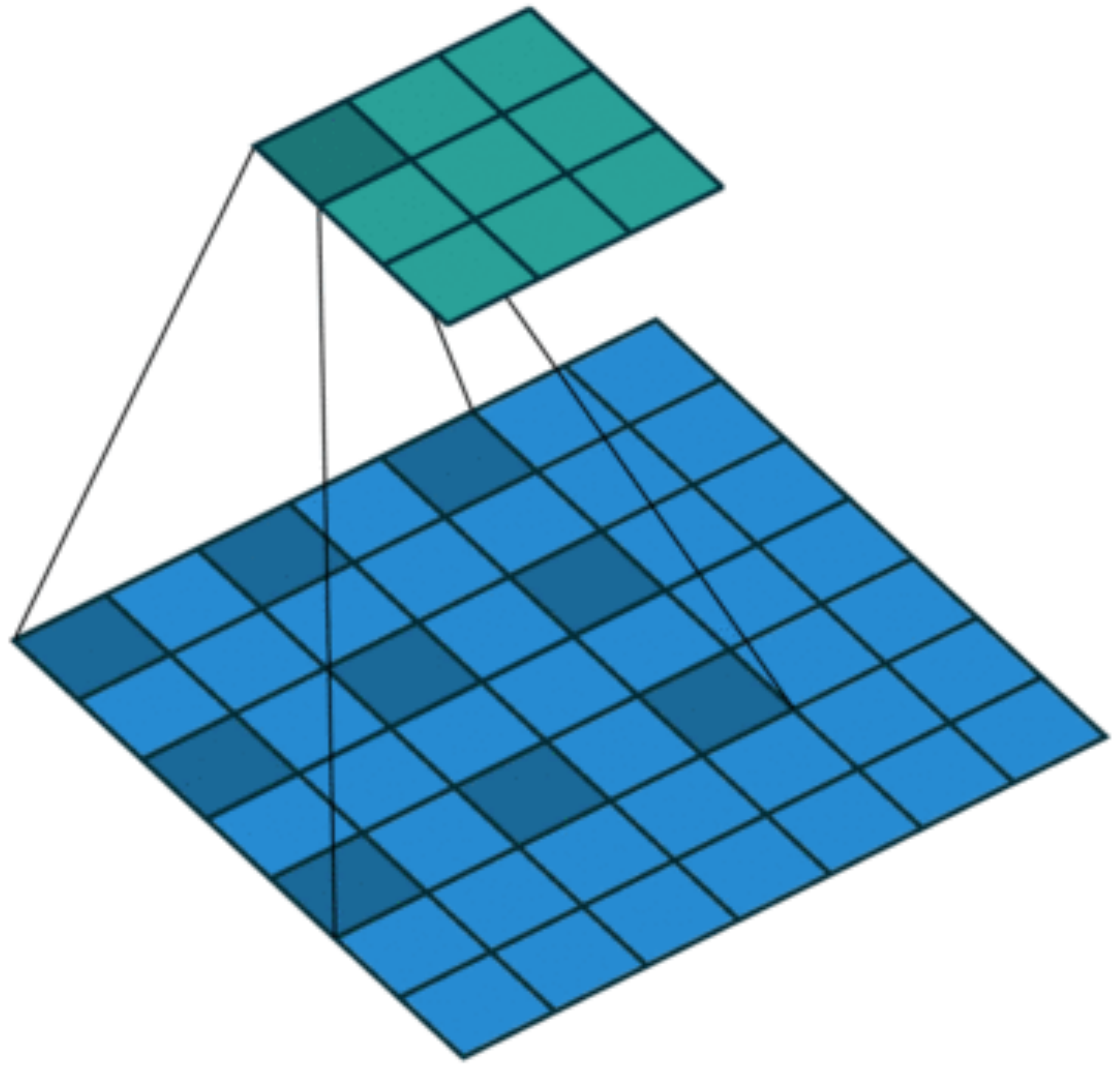
CONVOLUTION



Transposed,
No padding,
No strides



Transposed,
No padding,
Strides



Dilated,
No padding,
No strides

CONVOLUTION

$$\mathbf{O}[z][u][x][y] = \sum_{k=0}^{C-1}\sum_{i=0}^{S-1}\sum_{j=0}^{R-1} \mathbf{I}[z][k][Ux+i][Uy+j]\cdot\mathbf{W}[u][k][i][j] + \mathbf{B}[u]$$

ofmap $\mathbf{O}$, ifmap $\mathbf{I}$, filters (weights) $\mathbf{W}$, and biases $\mathbf{B}$

ofmap = output filter map (output activations)

ifmap = input filter map (input activations)

$$E = (H - R + U)/U$$

$$F = (W - S + U)/U$$

N	Batch size (3D fmaps)	0 <= z <= N
M	number of 3D filters / number of ofmaps	0 <= u <= M
C	number of ifmap/filter channels	
H / W	ifmap plane height/width	
R / S	filter plane height/width	
E / F	ofmap plane height/width	0 <= x <= F, 0 <= y <= E
U	stride	

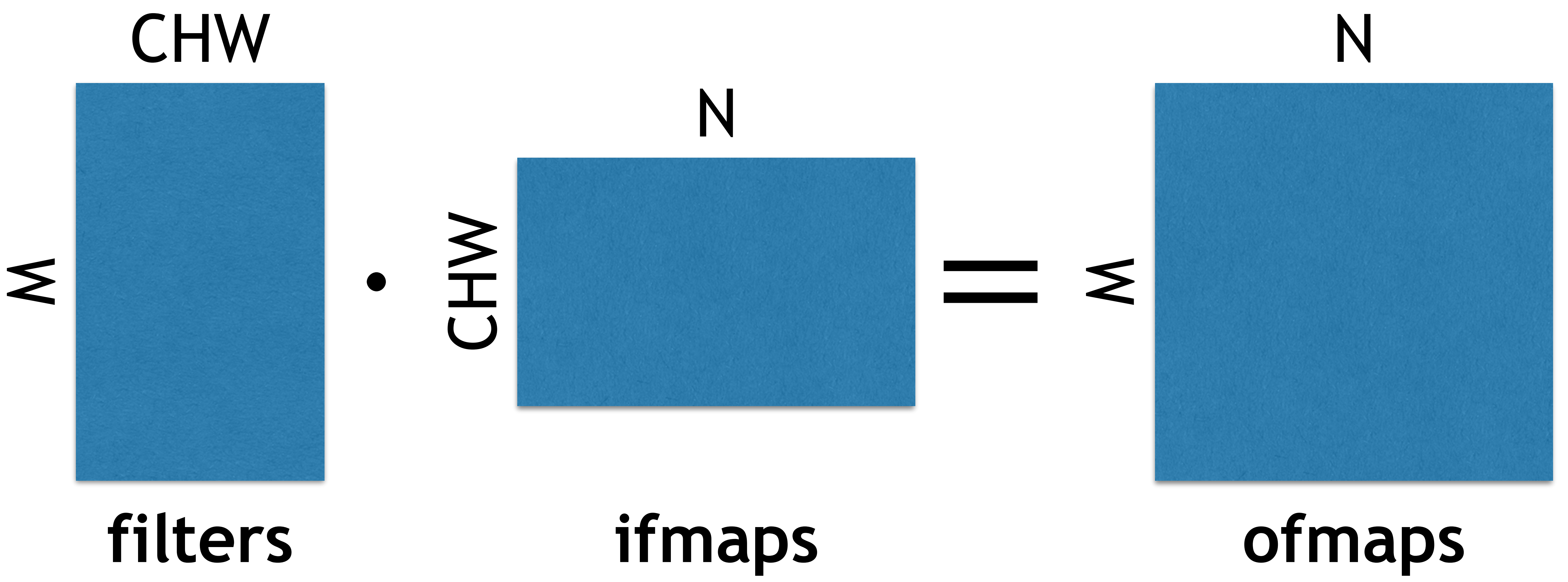
V. Sze, T.-J. Yang, Y.-H. Chen, J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, 2017.

FC AS SIMPLIFIED CONV LAYER

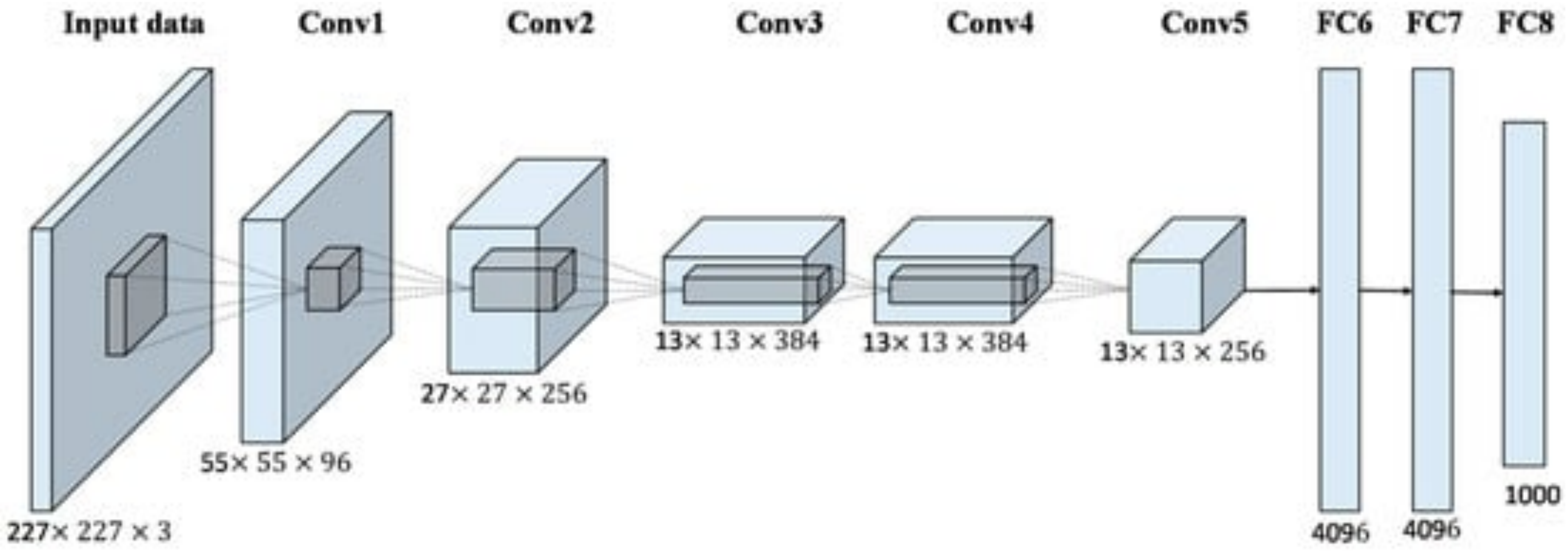
$$\mathbf{O}[z][u][x][y] = \sum_{k=0}^{C-1} \sum_{i=0}^{S-1} \sum_{j=0}^{R-1} \mathbf{I}[z][k][Ux+i][Uy+j] \cdot \mathbf{W}[u][k][i][j] + \mathbf{B}[u]$$

with $H = R$, $W = S$, $E = F = 1$ and $U = 1$

(read: filter size = input size, output per “filter” is a single element, no stride)



EXAMPLE MODEL ARCHITECTURE



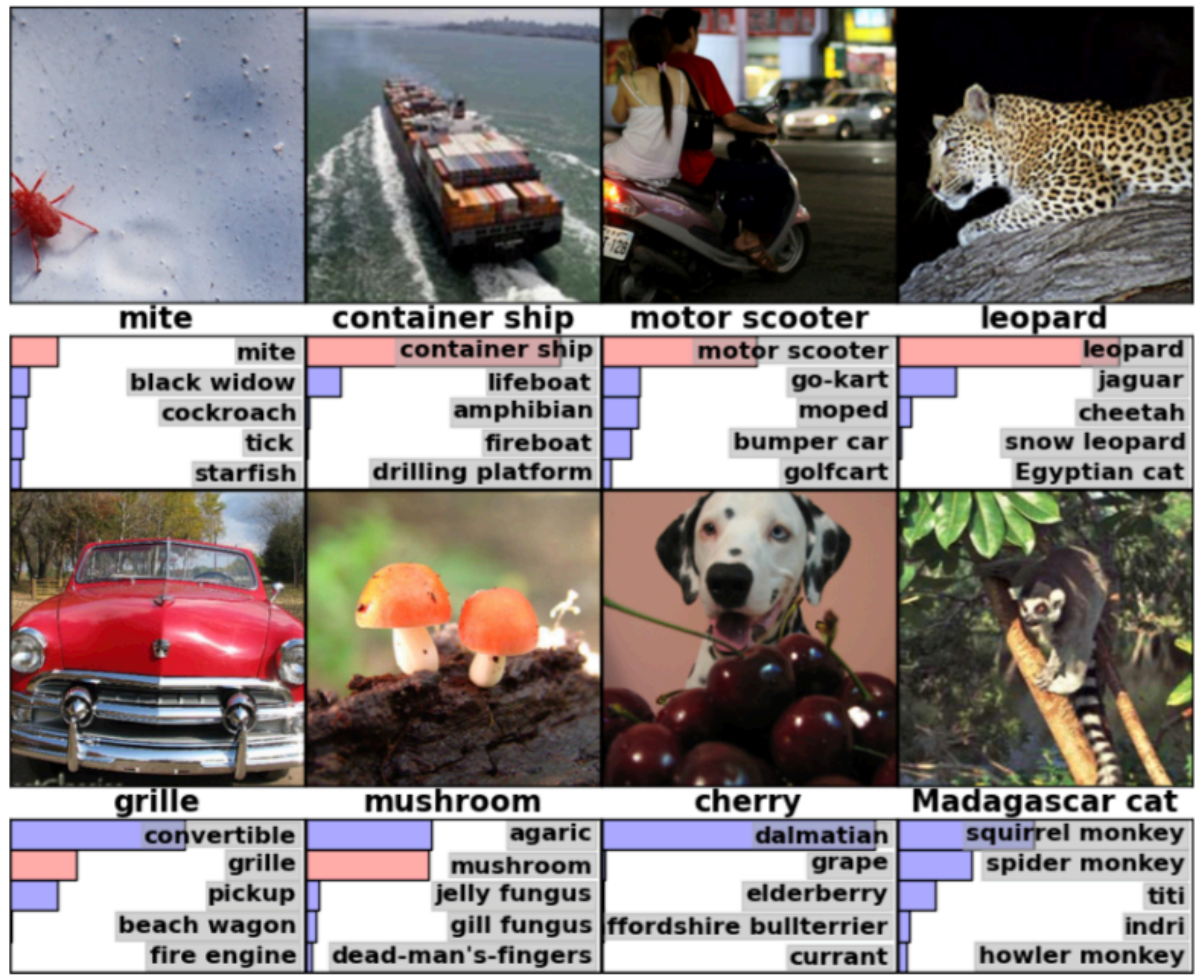
AlexNet: Alex Krizhevsky et al., "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012.

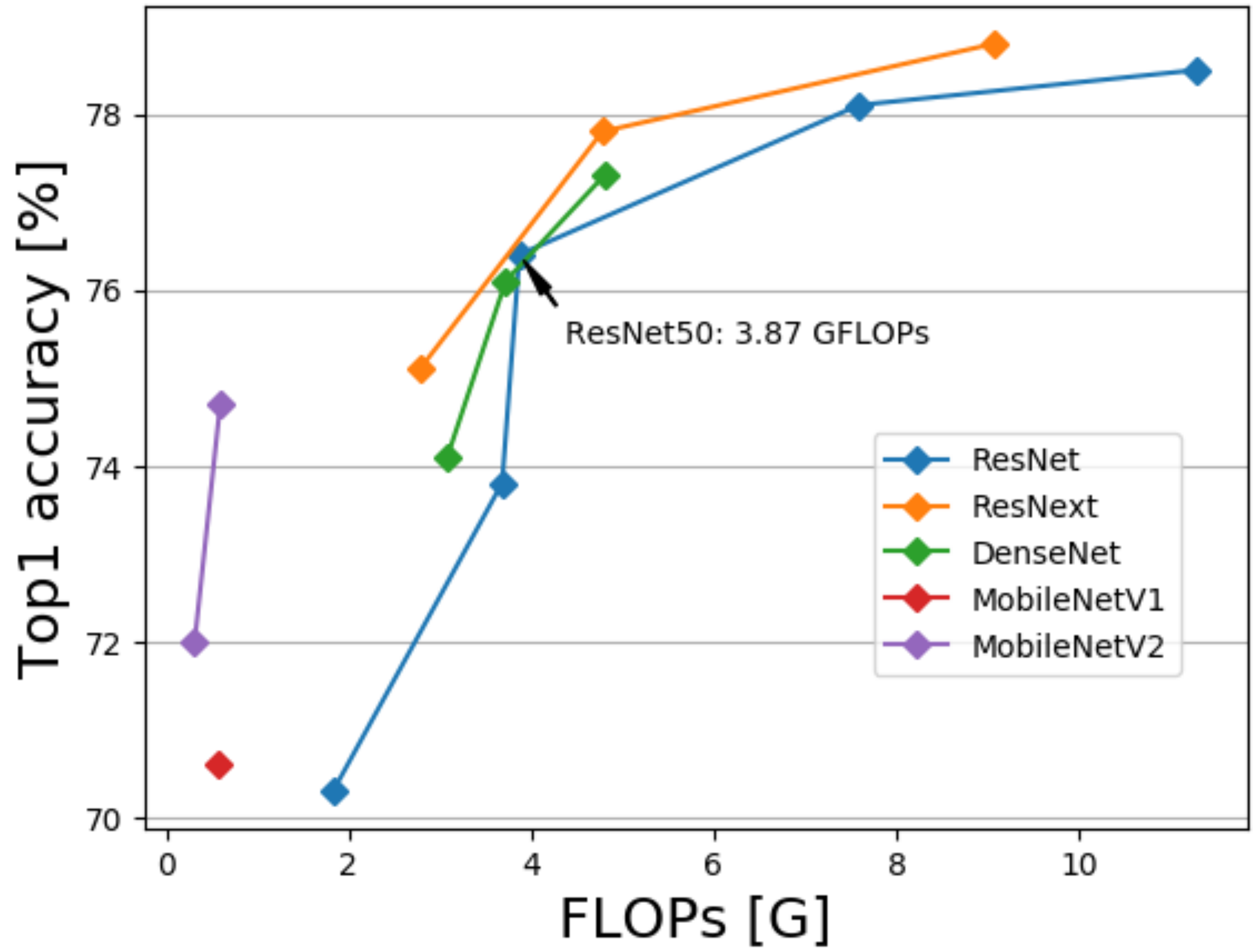
EFFICIENCY METRICS

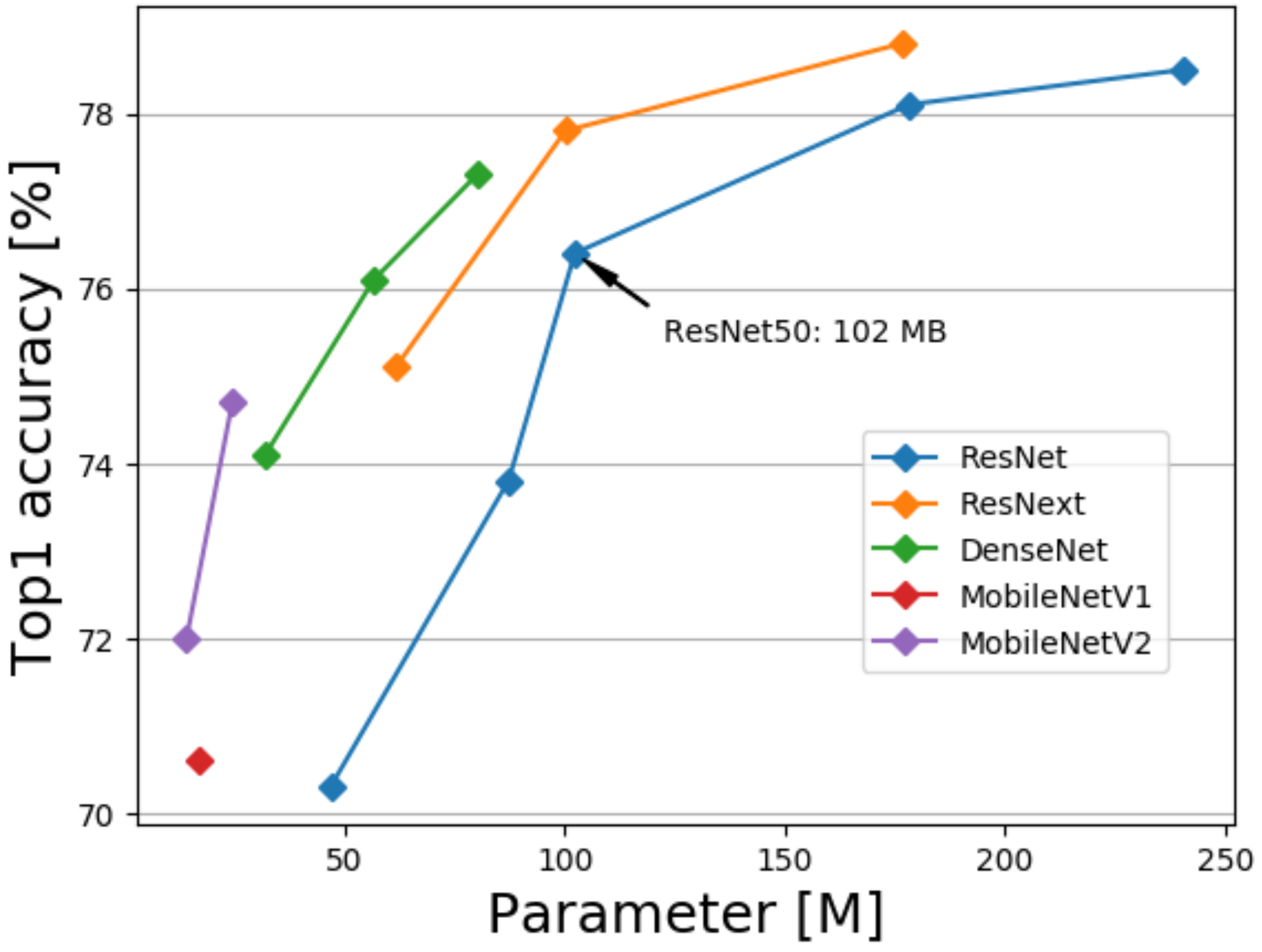
	MACs	Parameters (weight state)	Units (activation state)
FC	$MAC_f = WHCO$	$W_f = WHCO$	$U_f = O$
Convolution	$MAC_c = (EF \cdot RSC) \cdot M$	$W_c = RSCM$	$U_c = EFM$
Grouped convolution	$MAC_{cg} = \frac{MAC_c}{g}$	$W_{cg} = \frac{W_c}{g}$	$U_{cg} = EFM$
Depthwise separable convolution	$MAC_{cds} = EF \cdot (RSC + CM)$	$W_{cds} = RSC + CM$	$U_{cds} = EFC + EFM$

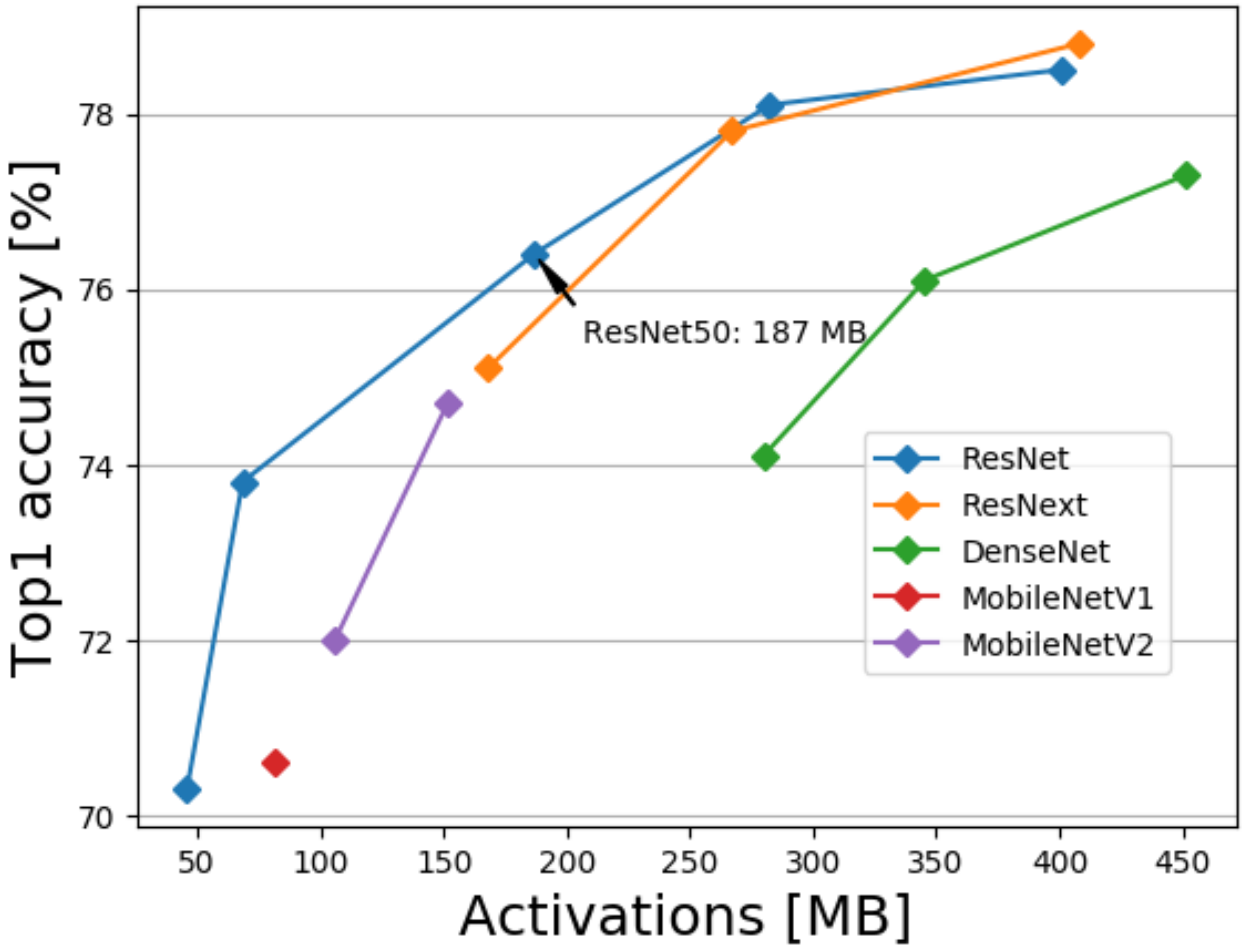
QUANTIZATION AS UNSAFE OPTIMIZATION

DNN REQUIREMENTS









DNNs are extremely compute and memory intensive

- Example ImageNet task with 224x224 pixels
- Accuracy scales with computations and memory
- ResNet50: 76% accuracy at 3.9 GFLOPs, 102MB parameter and 187MB activation

Objective: reduce computations and memory while maintaining prediction quality

DNNS SIMPLICITY WALL

Simplicity wall: DNNs spend most of their time in matrix multiplications

- Predictability, static loop-trip counts, little control overhead

Safe optimizations: use without restraints, no implication towards model's/workload's accuracy

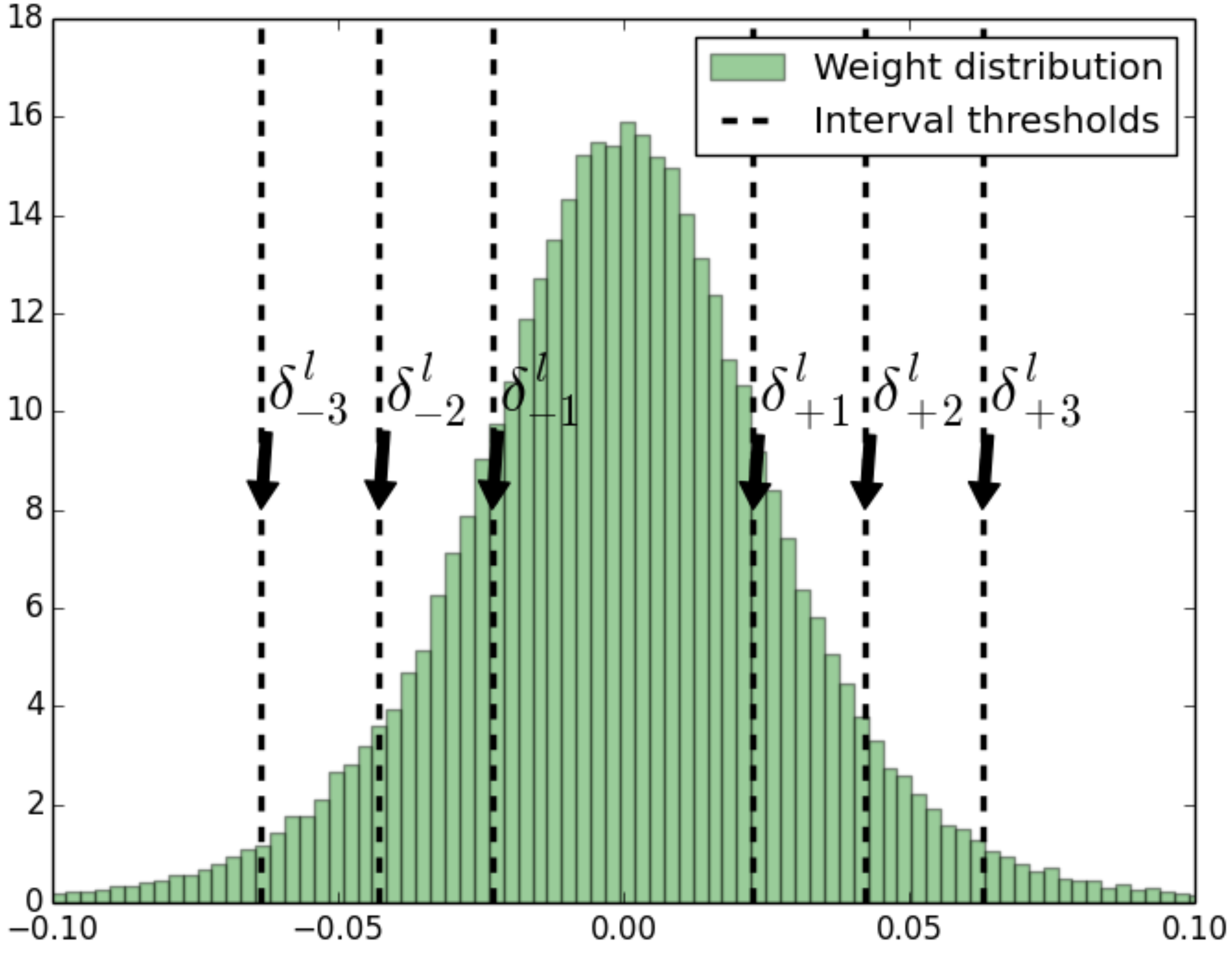
- Shorter communication paths
- Data reuse to minimize data volume being transferred
- => Dedicated architectures

Unsafe optimizations: potential implications towards model's/ workload's accuracy

- Reduce number of operations & model size: compression, pruning
- Reduce precision of operations and operands: quantization (fixed point, binarization)

Brandon Reagen; Robert Adolf; Paul Whatmough; Gu-Yeon Wei; David Brooks; Margaret Martonosi, "Deep Learning for Computer Architects", Morgan & Claypool, 2017, doi:10.2200/S00783ED1V01Y201706CAC041

QUANTIZED NEURAL NETWORKS



N-Ary quantization [1]

	Uniform	Non-Uniform	Bits
Binary	{-1,+1}	$\{W^p, W^n\}$	1
Ternary	{-1, 0, +1}	$\{W^p, 0, W^n\}$	2
Quaternary-	Na	$\{W^p, 0, W^{n,0}, W^{n,1}\}$	2
Quaternary+	Na	$\{W^{p,0}, W^{p,1}, 0, W^n\}$	2

[1] Schindler, G., Roth W., Pernkopf, F., Fröning, H.: N-Ary Quantization for CNN Model Compression and Inference Acceleration.

UNIFORM QUANTIZATION

Quantizer Q: piece-wise constant function

- Input values in given quantization interval mapped to corresponding quantization level
- Apply to activations/weights(/gradients)

$$Q(x) = q_l, \text{ if } x \in (t_l, t_{l+1}]$$

quantization level l (L total)

quantization intervals

Uniform quantization if all levels are equidistant

- $q_{i+1} - q_i = \Delta, \forall i,$ where Δ is a constant quantization step
- Limited model capacity
- Easy to store & compute ($log_2(L)$ bits without the quantization levels)
- Easy to compute if A & W quantized identically
- Keep activation function in mind when quantizing

$$Q(x) = \begin{cases} +1 : x \geq 0 \\ -1 : x < 0 \end{cases}$$

Example for binary quantization (sign function)

EXAMPLE (UNIFORM) QUANTIZATION USING K BITS

Real number

$$a_i \in [0,1]$$

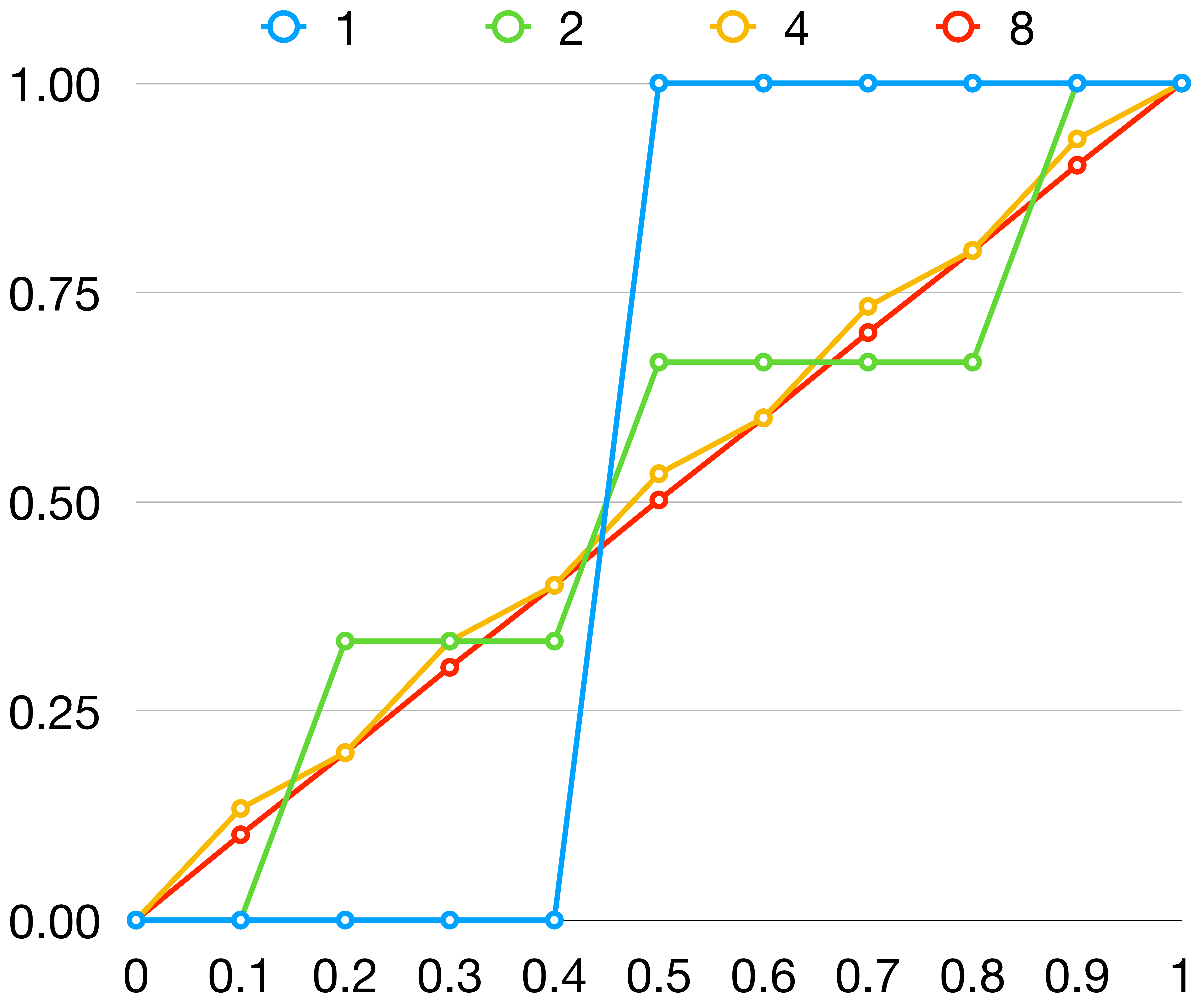
k-bit fixed-point integer

$$a_i^q \in [0,1]$$

Quantizer [1]

$$a_i^q = \frac{1}{2^k - 1} \cdot \text{round}\left((2^k - 1)a_i\right)$$

Assuming e.g. 10 possible input values (x-axis), one can reason about quantization error



[1] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR, abs/1606.06160, 2016. URL http://arxiv.org/abs/1606.06160.

NON-UNIFORM QUANTIZATION

$$Q(x) = q_l, \text{ if } x \in (t_l, t_{l+1}]$$

quantization level l (L total)

quantization intervals

Non-uniform quantization improves model capacity

Storage: $log_2(L)$ bits plus the levels

Computation: requires quantization level q_l

Trainable quantization levels (scaling factors) to adapt to weights/activations

$$w_l^i = \begin{cases} W_l^p : w_l > \Delta_l \\ 0 : |w_l| \leq \Delta_l \\ -W_l^n : w_l < -\Delta_l \end{cases}$$

$$\Delta_l = t \cdot max(|w|); t \in [0,1]$$

RELATED WORK QUANTIZATION

Own work

SW quantization concepts

	Weights	Activations
BNN	{-1,+1}	{-1,+1}
XNOR	{-S,+S}	{-1,+1}
DoReFa	{-S,+S}	{0,+1}
TWN	{-S,0,+S}	float32
TTQ	{-Sn,0,+Sp}	float32
HWGQ	XNOR	2bit

AlexNet/ImageNet accuracy (%) for state-of-the-art quantization

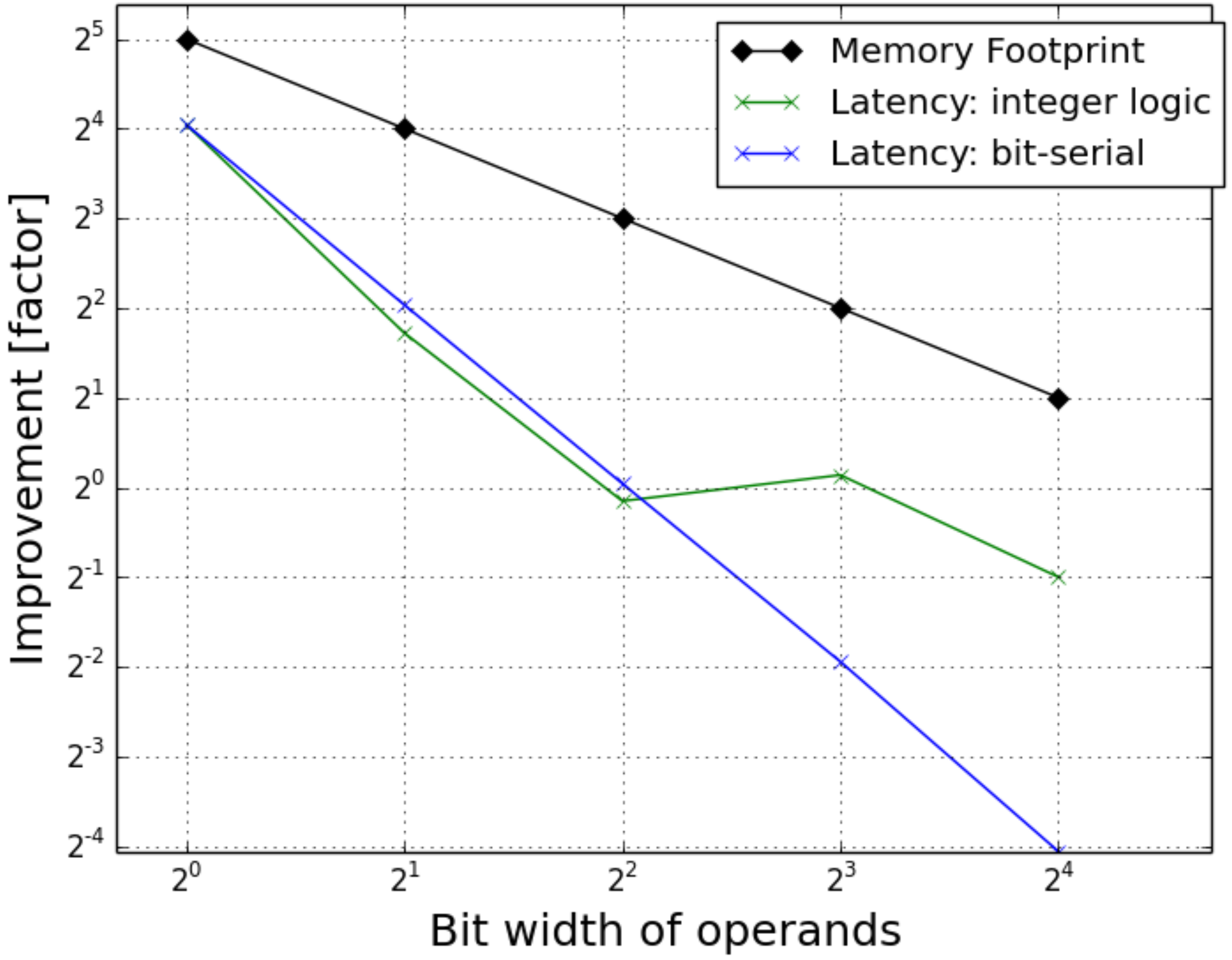
A-W	Deep Cmpr.	BNN	XNOR	DoReFa	TWN	TTQ	HWGQ	Deep Chip
32-32	80.3	80.2	80.2	80.3	80.3	80.3	81.5	
32-8/	80.3							
32-2					76.8	79.7		
8-2								79.0
32-1				76.3				
2-1							76.3	
1-1		50.4	69.2	69.3				

Key observation: DNNs contain plenty of redundancy
Nonuniform quantization outperforms uniform quantization

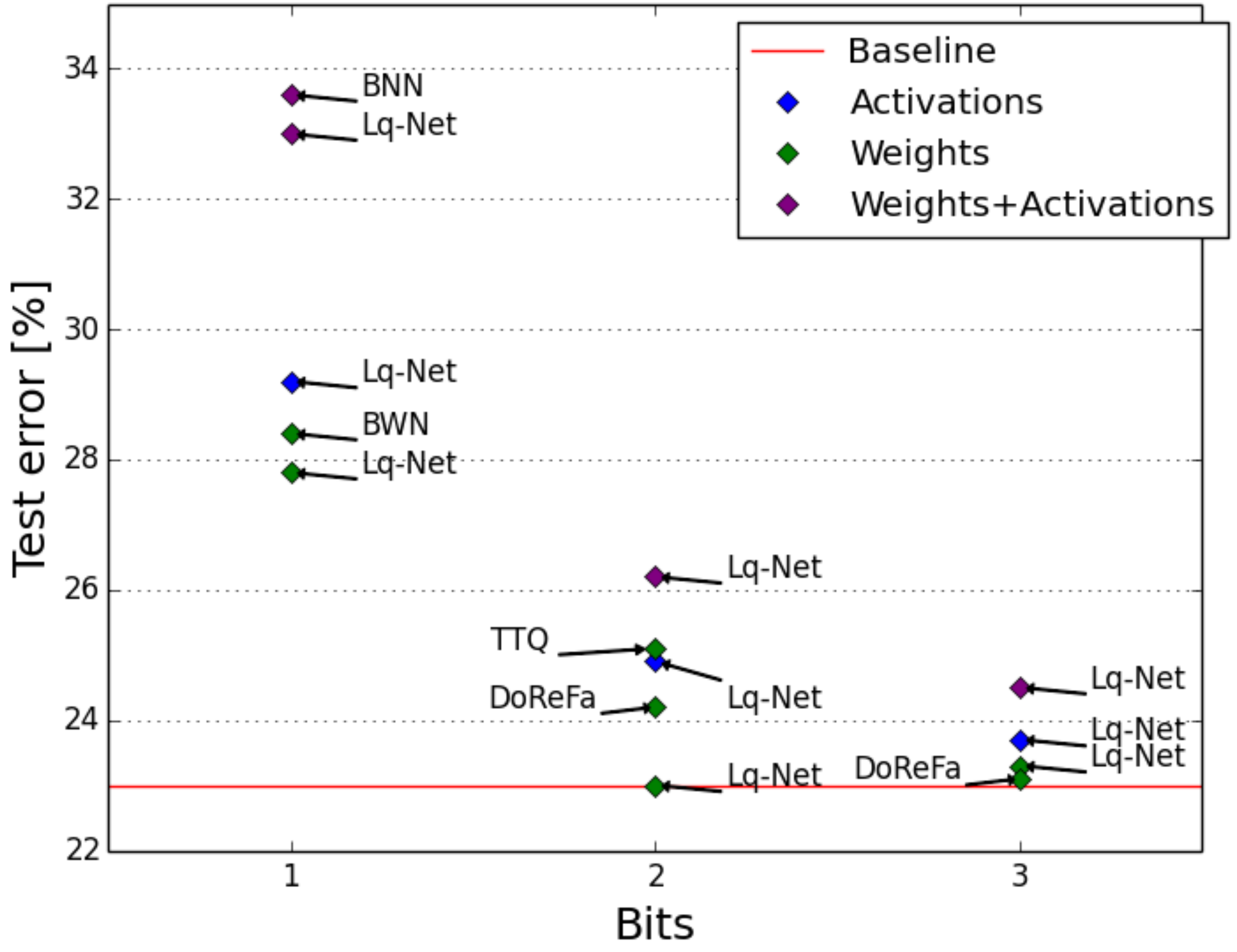
Wolfgang Roth, Günther Schindler, Bernhard Klein, Robert Peharz, Sebastian Tschiatschek, Holger Fröning, Franz Pernkopf, Zoubin Ghahramani, Resource-Efficient Neural Networks for Embedded Systems. ArXiv:2001.03048 [stat.ML], Dec. 2022. http://arxiv.org/abs/2001.03048

QUANTIZED NEURAL NETWORKS

Improvement of QNNs [1]



Degradation of QNNs [2]



[1] Schindler, G., Mücke, M., Fröning, H. Linking Application Description with Efficient SIMD Code Generation for Low-Precision Signed-Integer GEMM, 10th Workshop on UnConventional High Performance Computing 2017 (UCHPC 2017), in conjunction with EuroPAR 2017.

[2] Roth, W., Schindler, G., Zöhrer, M., Pfeifenberger, L., Peharz, R., Tschiatschek, S., Fröning, H., Pernkopf, F., Ghahramani, Z. Resource-Efficient Neural Networks for Embedded Systems. https://arxiv.org/abs/2001.03048

DEEPCHIP’S REDUCE-AND-SCALE

Quantization (and pruning) for mobile ARM processors

DEEPCHIP: MODEL COMPRESSION FOR DEEP LEARNING ON RESOURCE-CONSTRAINED DEVICES (2016-)

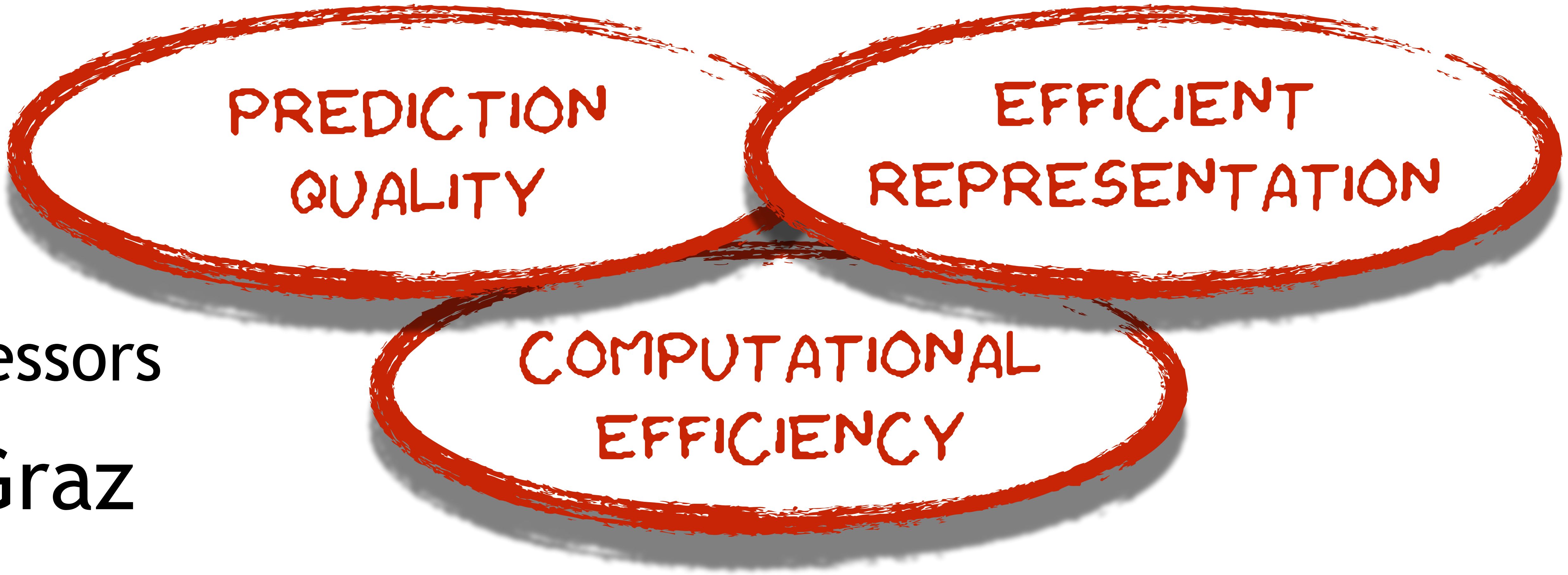
DL-based speech & image processing for resource-constrained devices

- Trading among precision, model size and accuracy
- Preferred: no accuracy loss compared to state-of-the-art

Reduced precision (quantization), sparsity and asynchrony

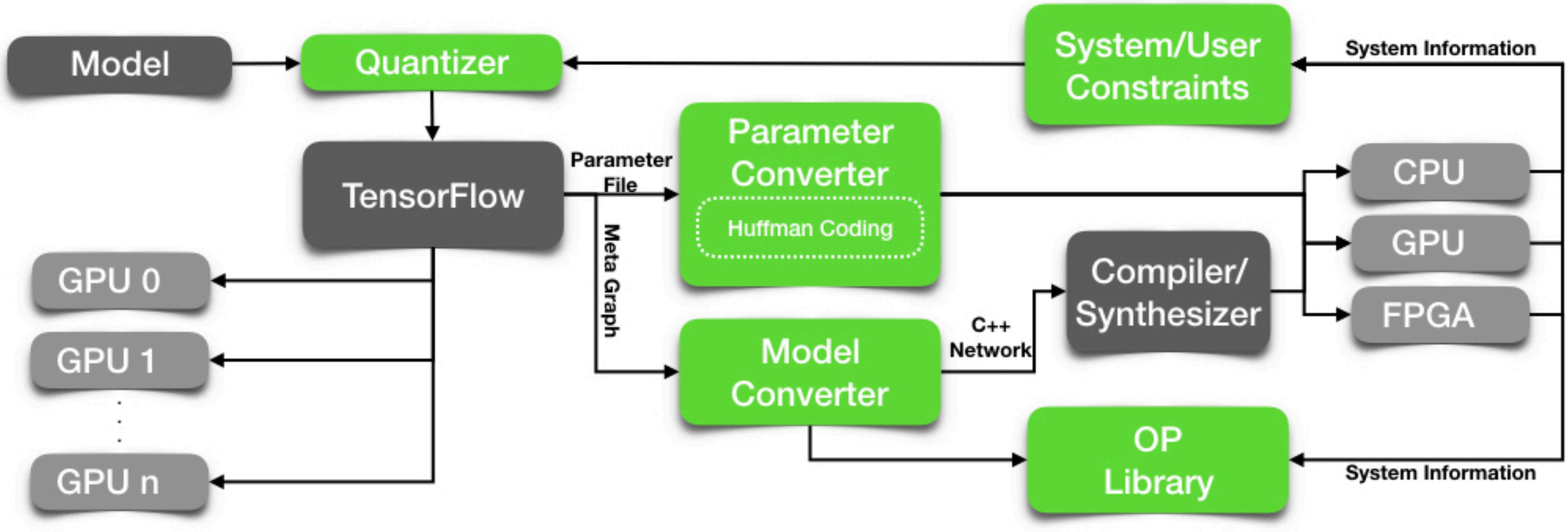
1. Inference architecture suitable for various embedded processors
2. New neural networks concepts with particular low requirements
3. Software inference architecture based on quantization and pruning
4. Exploring applicability to various processors

Collaboration with SPSC group @ TU Graz



http://www.deepchip.org

DEEPCHIP: SW ARCHITECTURE FOR QUANTIZATION



Günther Schindler, Matthias Zöhrer, Franz Pernkopf, and Holger Fröning, Towards Efficient Forward Propagation on Resource-Constrained Systems, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2018).

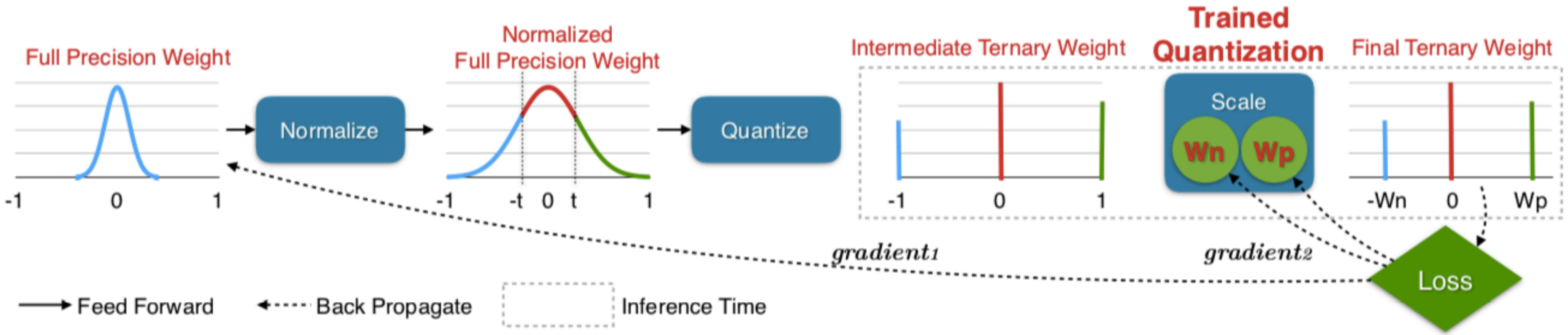
TRAINED TERNARY QUANTIZATION

Train full-precision weights & train scale factors for ternary weights (hyperparameter t)

1. Normalization: weights in range $[-1, +1]$
2. Quantization by thresholding: $\{-t, 0, +t\}$
3. Learning ternary assignments: gradient to full-precision weights
4. Learning ternary values: gradient to scaling factors

$$w_l^i = \begin{cases} W_l^p : w_l > \Delta_l \\ 0 : |w_l| \leq \Delta_l \\ -W_l^n : w_l < -\Delta_l \end{cases}$$

$$\Delta_l = t \cdot max(|w|); t \in [0,1]$$



Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. CoRR, abs/1612.01064, 2016. URL http://arxiv.org/abs/1612.01064

REDUCE-AND-SCALE QUANTIZATION

Weight quantization to ternary values according to TTQ

- Scale factors $\{W_p, W_n\}$: independent + asymmetric, trained using SGD
- Hyperparameter t => trading among accuracy and space

$$w_l^i = \begin{cases} W_l^p : w_l > \Delta_l \\ 0 : |w_l| \le \Delta_l \\ -W_l^n : w_l < -\Delta_l \end{cases}$$

$$\Delta_l = t \cdot \max(|w|); t \in [0,1]$$

Bounding activations, quantization to fixed point (flexible bit-width k, DoReFa)

- Bounded ReLU => $0 \le a_i \le 1$
- cf. TTQ using floating point

$$a_i = \begin{cases} 0 : \tilde{a}_i \le 0 \\ \tilde{a}_i : 0 < \tilde{a}_i < 1 \\ 1 : \tilde{a}_i \ge 1 \end{cases}$$

$$a_i^q = \frac{1}{2^k - 1} \mathrm{round}\left((2^k - 1)a_i\right)$$

PARAMETER CONVERTER

- Space-efficient data structures
  - Intermediate matrices I_l^p and I_l^n
  - Indices vector i_l based on W_l^T
- Run-length encoding of weight matrix
  - Non-zero values + signs
  - Only sign and distance vector stored
  - => Reduced cardinality
- Compression using Huffman coding (not shown)

$$\mathbf{W}_l^T = \begin{pmatrix} 0 & W_l^p & W_l^p & 0 & W_l^n \\ W_l^n & 0 & 0 & W_l^p & 0 \\ W_l^p & W_l^n & 0 & W_l^n & W_l^n \end{pmatrix}$$

$$\mathbf{I}_l^p = \begin{pmatrix} 1 & 2 & - \\ 3 & - & - \\ 0 & - & - \end{pmatrix} \qquad \mathbf{I}_l^n = \begin{pmatrix} 4 & - & - \\ 0 & - & - \\ 1 & 3 & 5 \end{pmatrix}$$

Signs $\mathbf{s}_l = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$

Indices $\mathbf{i}_l = \begin{pmatrix} 1 & 2 & 4 & 5 & 8 & 10 & 11 & 13 & 14 \end{pmatrix}$

Distance $\mathbf{d}_l = \begin{pmatrix} 1 & 1 & 2 & 1 & 3 & 2 & 1 & 2 & 1 \end{pmatrix}$

OPERATOR LIBRARY - REDUCE & SCALE

Saving complexity

1. Reduced precision
2. Sparsity
3. Only partial sums and two multiplications

More general: one multiplication per quantization level (n-ary)

$$c = \sum_{i=1}^{N} w_i \cdot a_i, \quad w_i, a_i \in \mathbb{R} \quad \forall i$$

$$\Downarrow$$

$$c = W_l^p \cdot \sum_{i \in \mathbf{i}_l^p} a_i + W_l^n \cdot \sum_{i \in \mathbf{i}_l^n} a_i, \quad where$$

$$\mathbf{i}_l^p = \{i | b_i = W_l^p\} \quad \text{and} \quad \mathbf{i}_l^n = \{i | b_i = W_l^n\}$$

MULT vs. ADD

Instruction	Cycles [ARM] (normalized)	Energy (pJ) [Horowitz]
float32 FMA	8.0	4.6
int16 FMA	3.0	1.6
int16 ADD	1.5	0.05

AlexNet/ImageNet

	Baseline	BNN	INT8	DeepChip
Top-5 Accuracy [%]	78.3	56.4	-[1]	79.0
Sparsity [%]	0.0	0.0	0.0	63.0
Inference Rate [FPS]	4	22	7	8
Memory [MB]	244	24	61	25

[1] *Authors claim no change in accuracy*

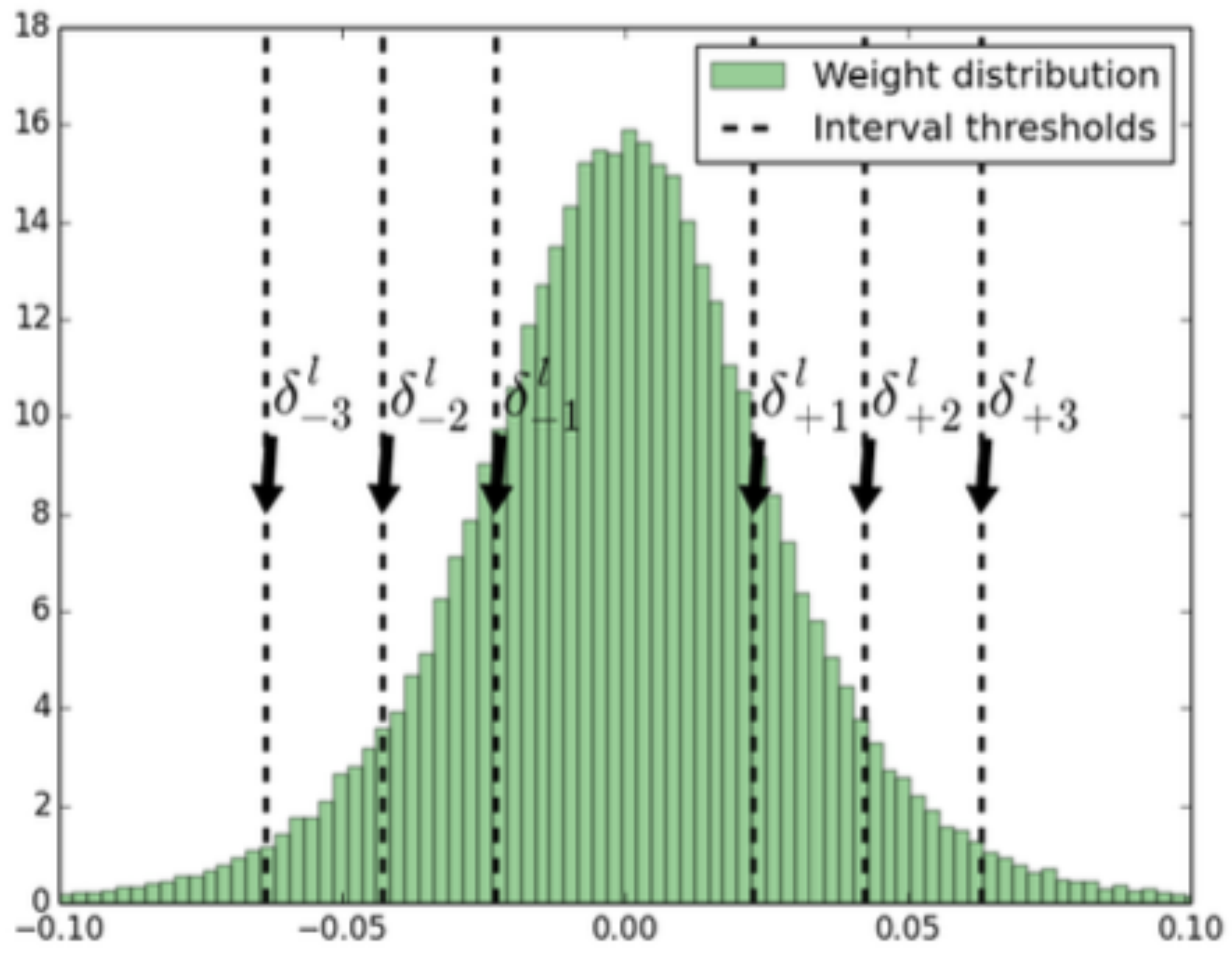
N-ARY QUANTIZATION (NAQ)

Up to now: all good for ConvNet+SVHN, AlexNet+ImageNet, ResNet-44+CIFAR-10

- I.e., complex model + simple data, or simple model + complex data
- But: quantization depends on complexity(data) & complexity(model)

Non-uniform n-ary weight representations

- Multiple scale factors, cost-effective nested-means clustering



ResNet-18/ImageNet

	Weights [bit]	Activations [bit]	Training	Top-5 [%]
LQNet	2	2	2.3x	85.9
RaS - ternary	2	2	1.2x	86.7
LQNet	3	32	1.7x	88.8
RaS - quinary	3	32	2.0x	89.0

PRUNING

Basics and structured pruning

MODEL COMPRESSION

Quantization

Data type

Number format, representation, bit width?

Homogeneous or heterogeneous

Layer, filter, neuron, weight

Efficiency depends on HW

Pruning

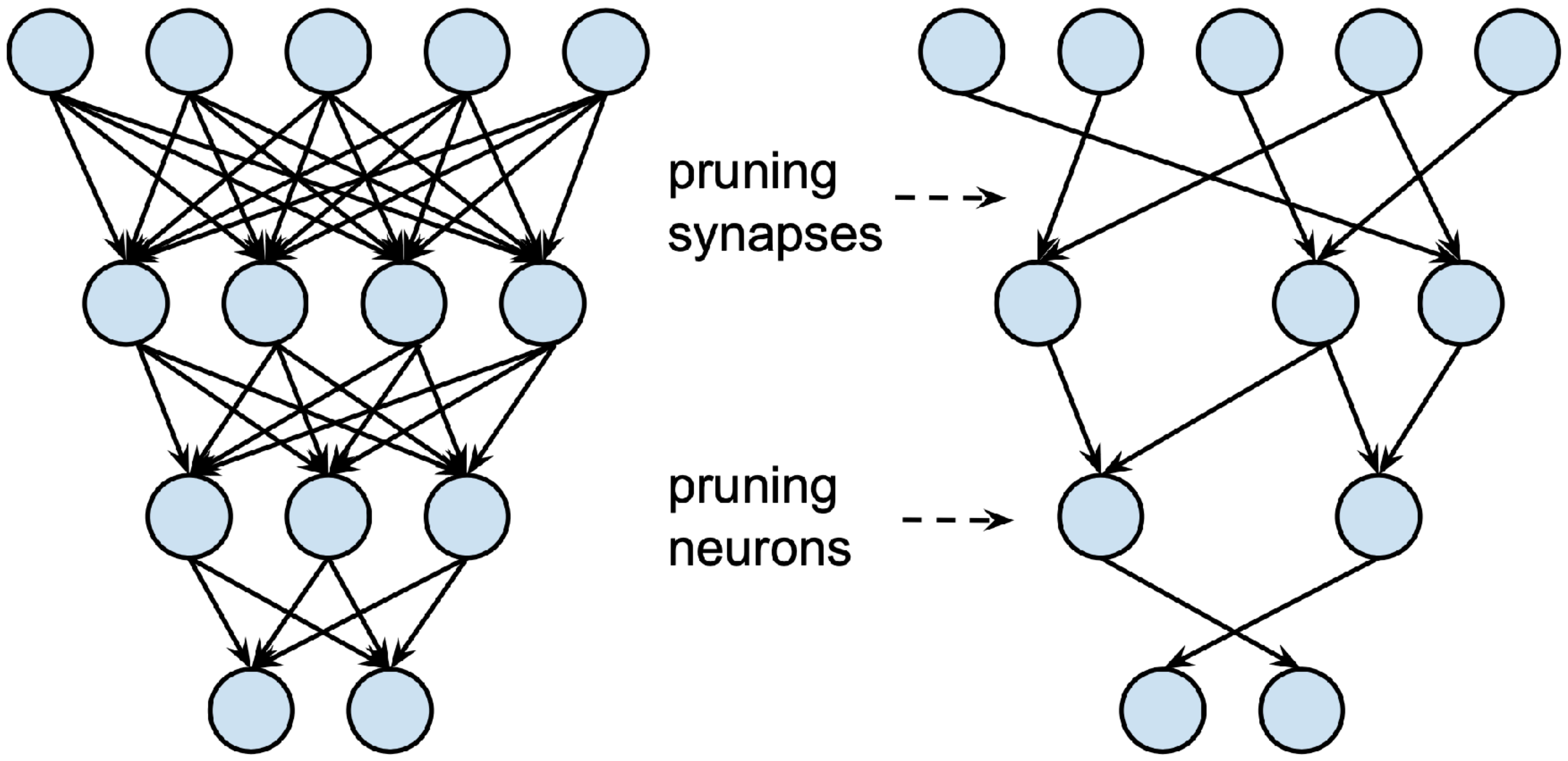
Unstructured vs. Structured

Magnitude based, magnitude+x, regularization?

Homogeneous or heterogeneous

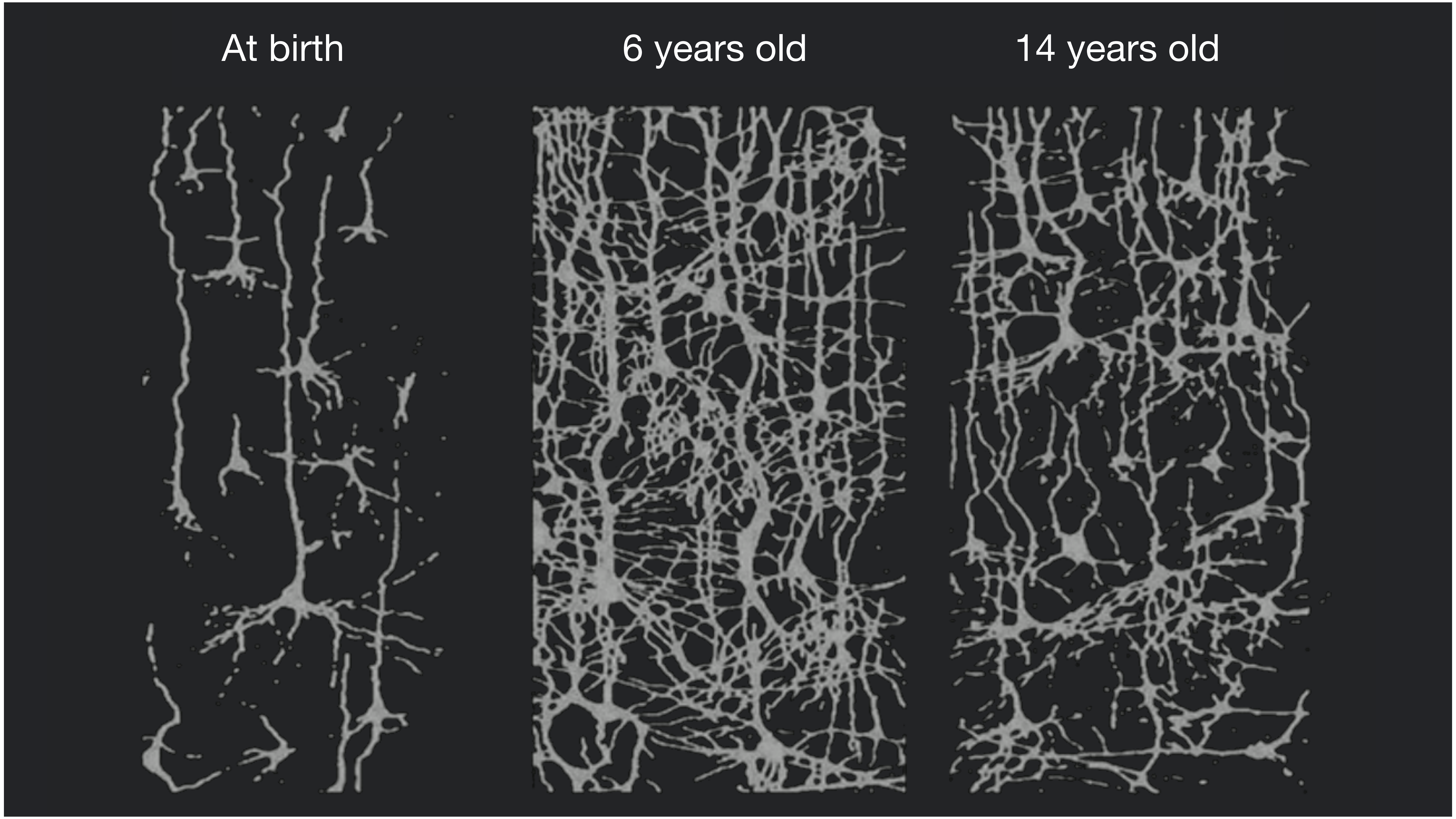
Layer, filter, neuron, weight

Efficiency depends on HW



Song Han, Jeff Pool, John Tran, William J. Dally, Learning both Weights and Connections for Efficient Neural Networks, NIPS 2015, https://arxiv.org/abs/1506.02626

EVOLUTION OF HUMAN BRAIN DURING LIFE



Source: Rethinking the Brain: New Insights into Early Development

UNSTRUCTURED MAGNITUDE-BASED PRUNING

Many parameters, so pruning methods have to be computationally cheap

- Early work considers second- and first-order Taylor expansions on the Hessian of the loss function, which is not cheap at all

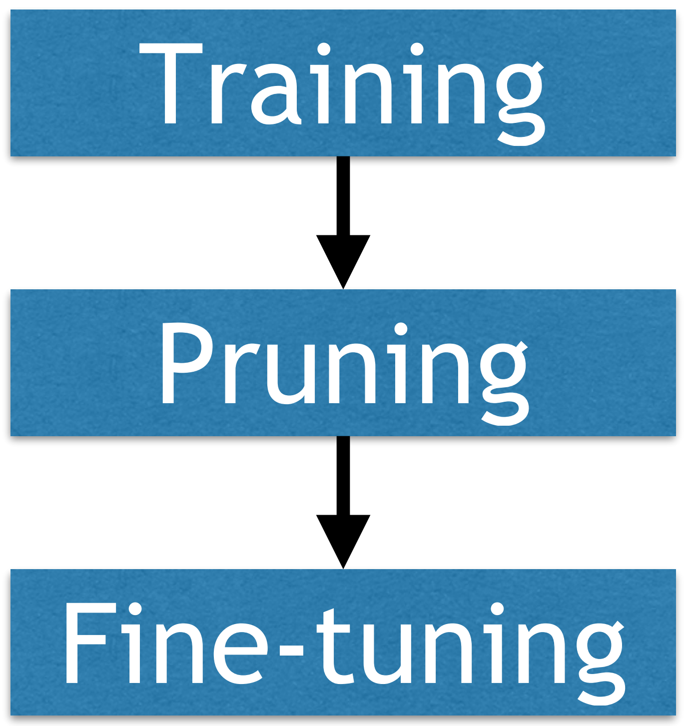
1. <u>Pruning granularity</u>: fine-grained pruning (individual weights) is most accurate

- Possibly difficult to exploit sparsity on massively-parallel processors

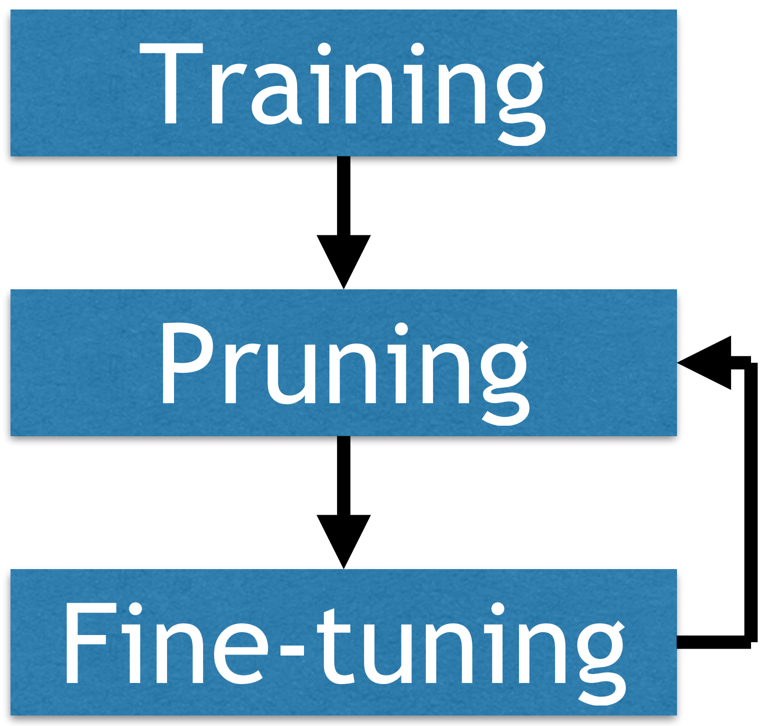
2. <u>Pruning procedure</u>: when to remove weights

- Neurons can also be removed if all associated weights are pruned

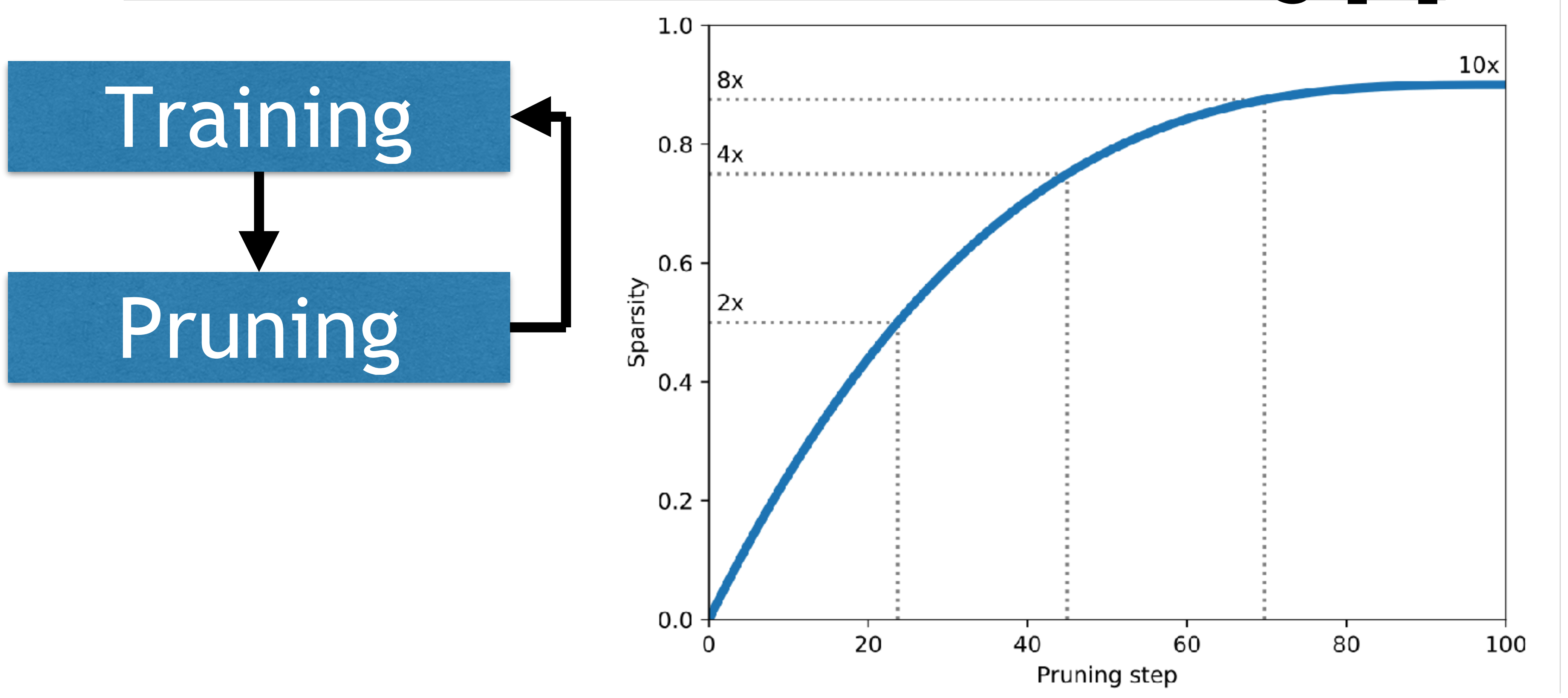
<u>2a. One-shot pruning</u>



<u>2b. Iterative pruning</u>



<u>2c. Automated Gradual Pruning [1]</u>



[1] Michael Zhu, Suyog Gupta, To prune, or not to prune: exploring the efficacy of pruning for model compression, https://arxiv.org/abs/1710.01878

UNSTRUCTURED MAGNITUDE-BASED PRUNING

3. Pruning criteria: which connections to remove, i.e., which weights to set to zero

3a. Weight fraction pruning

Remove smallest weights among all weights, e.g. based on a certain percentage

Sparsity in percent is known a-priori

3b. Weight magnitude pruning

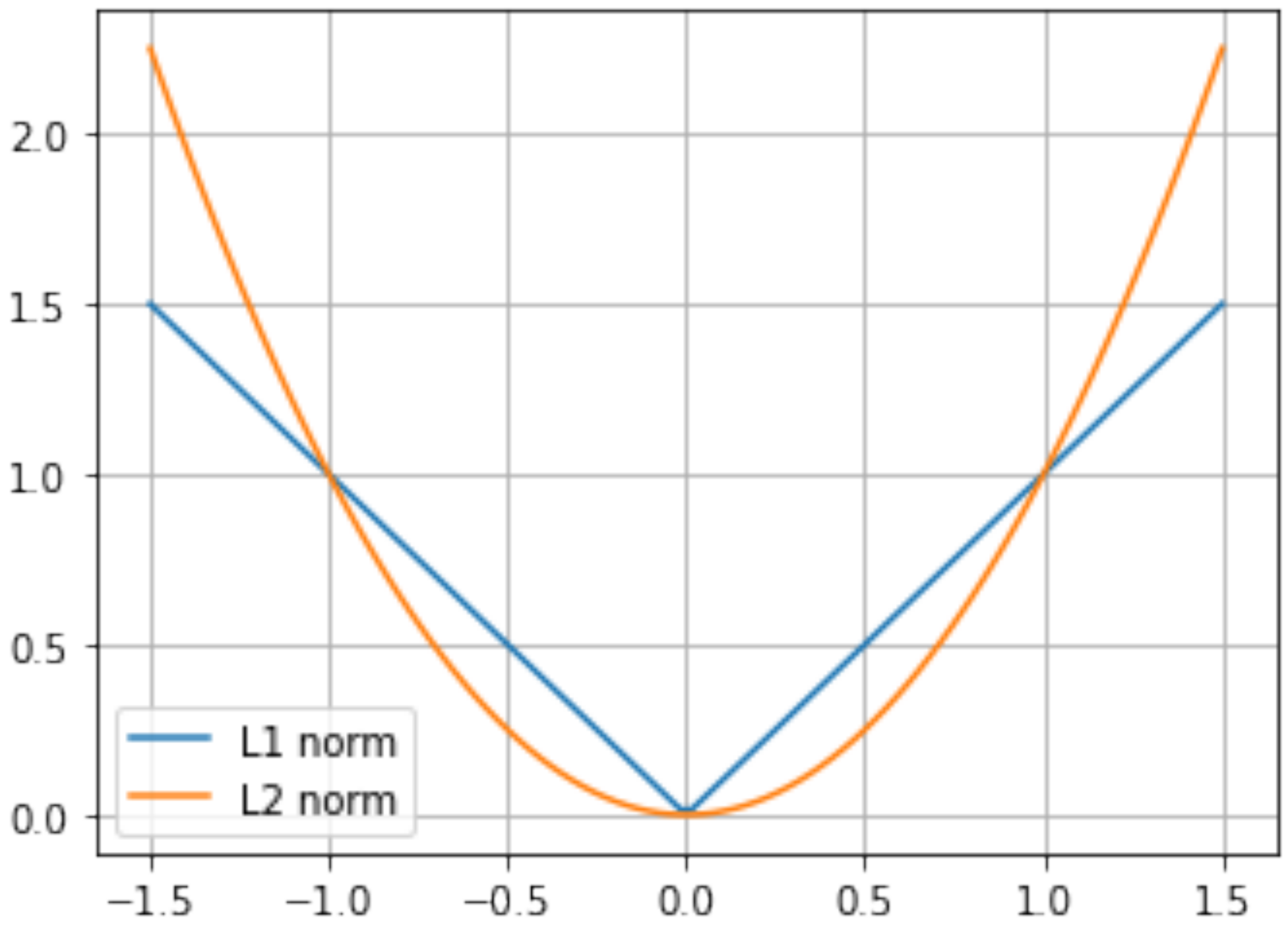
Remove weights below a certain threshold: $|x_i| \leq t$

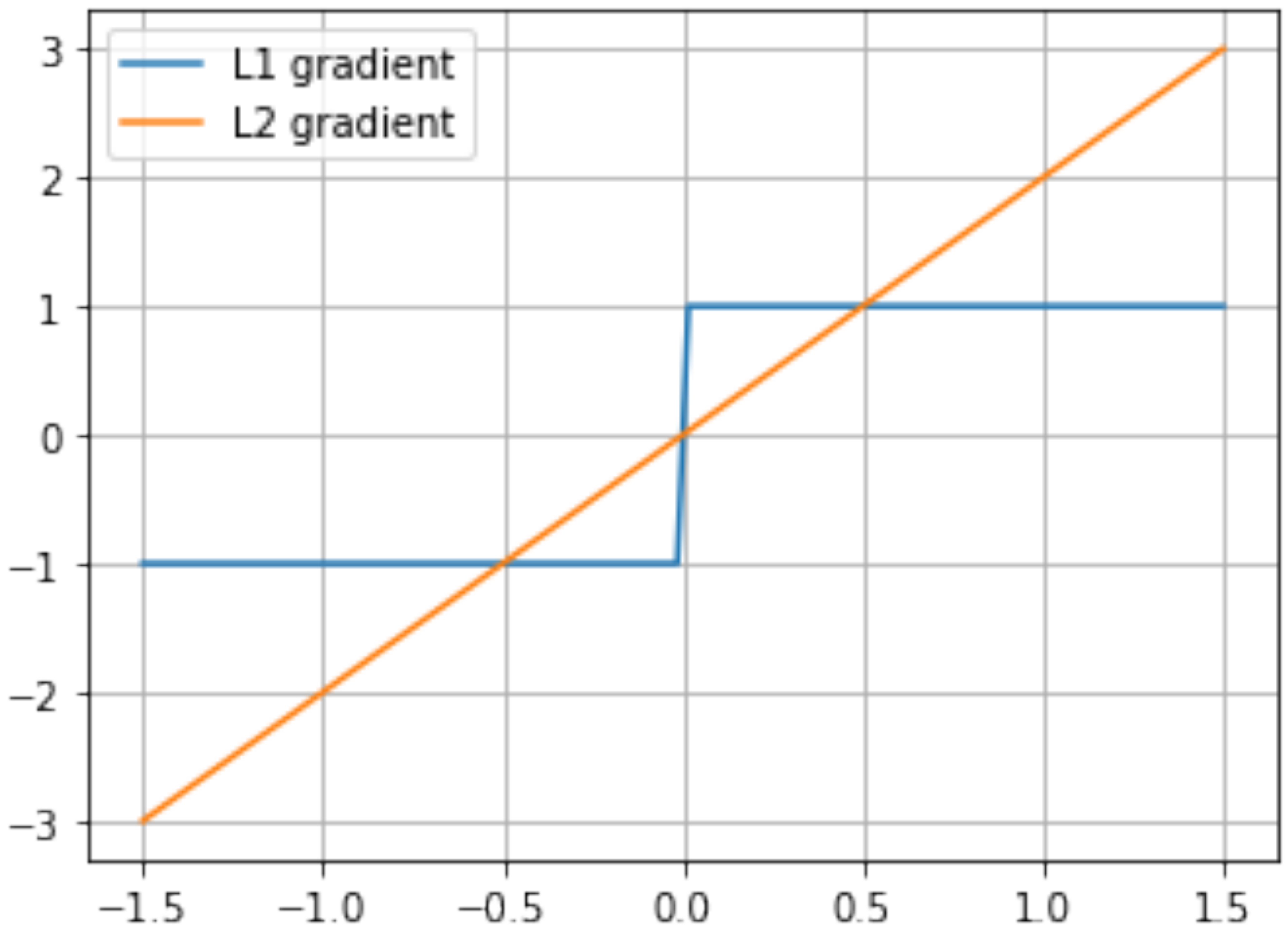
Sparsity in percent is not known a-priori

3c. Gradient magnitude pruning

Multiple weights by their gradient before thresholding: $|x_i \cdot g_i| \leq t$

RECAP: L1 VS L2 NORM FOR LOSS FUNCTION

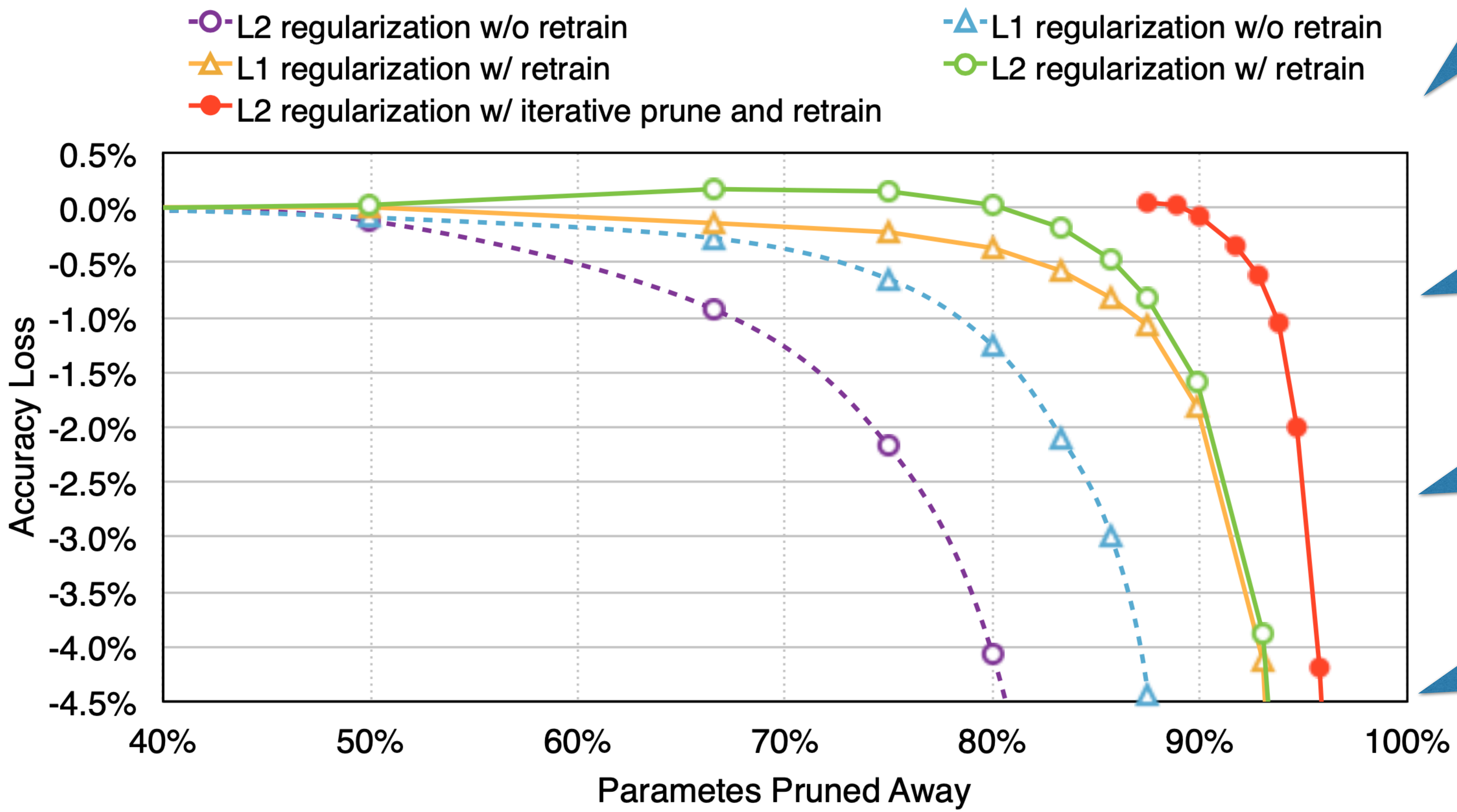




$$\mathscr{R}_{L1}(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_j |w_j|$$

$$\mathscr{R}_{L2}(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2 = \frac{1}{2}\sum_j w_j^2$$

VALUE OF PRUNING



Top-5 accuracy for AlexNet/ImageNET

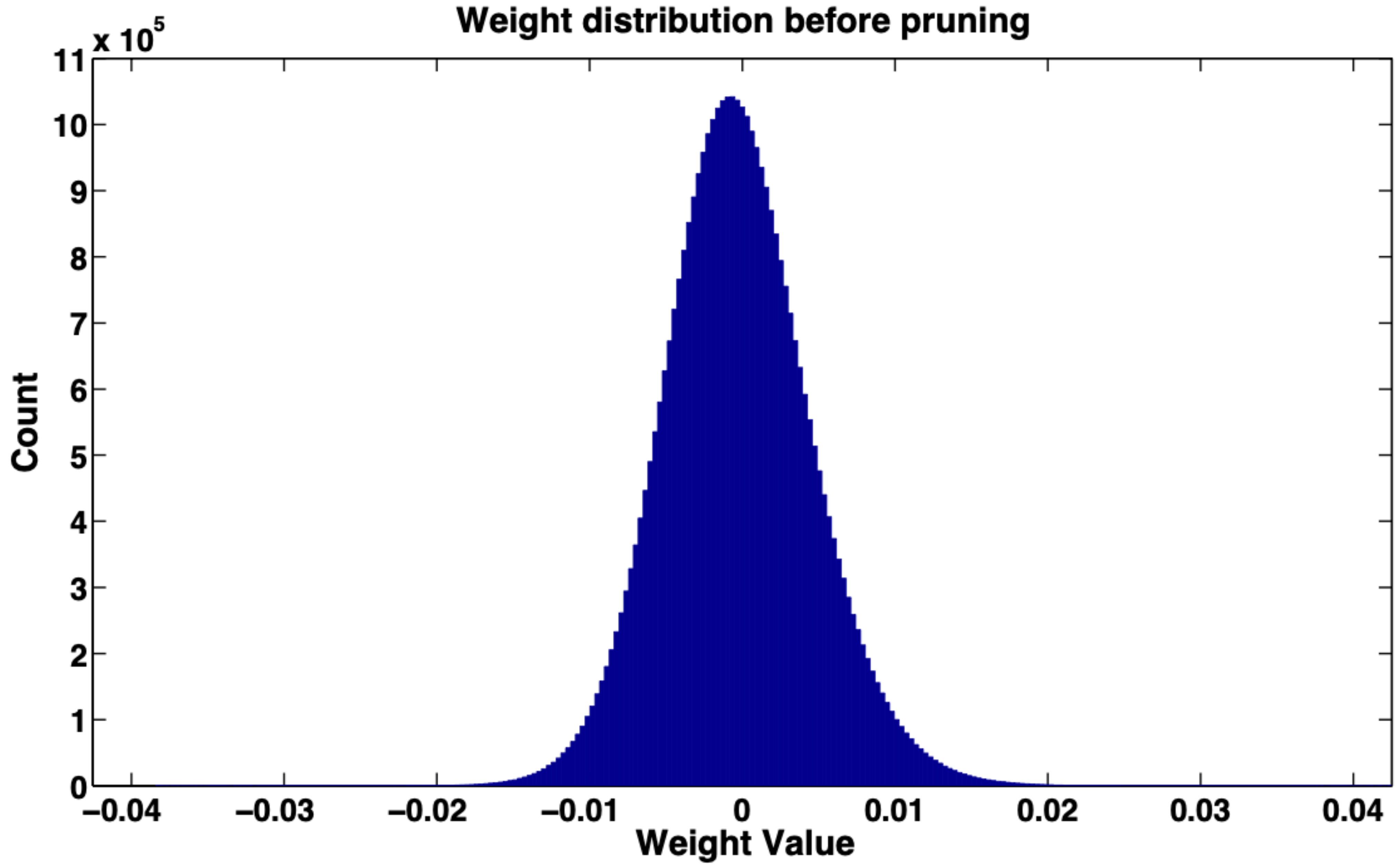
Without retraining, L1 regularization is the best option

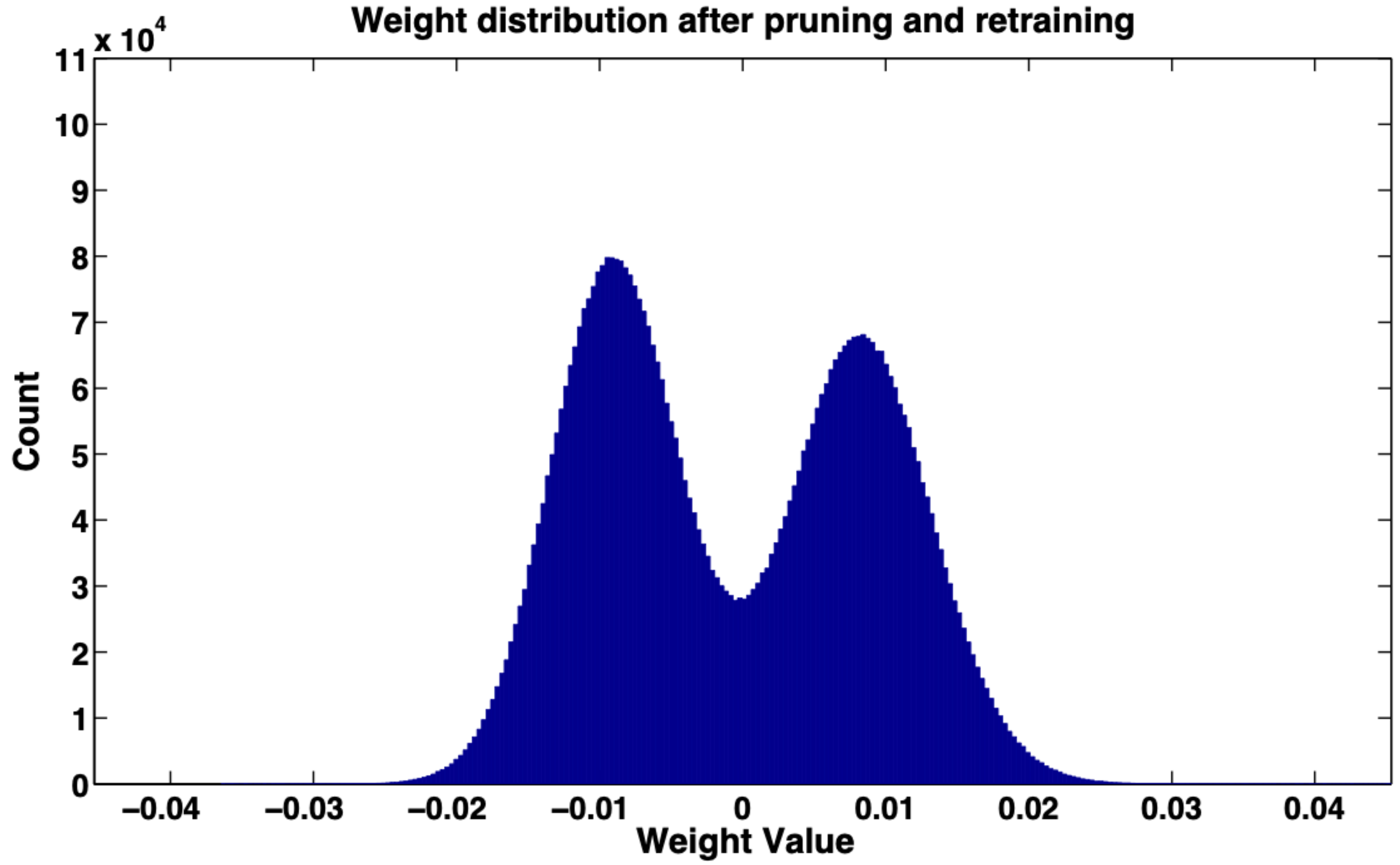
With retraining, L2 regularization is the best option

Retraining is mandatory to recover from accuracy loss

Song Han, Jeff Pool, John Tran, William J. Dally, Learning both Weights and Connections for Efficient Neural Networks, NIPS 2015, https://arxiv.org/abs/1506.02626

RETRAINING CHANGES WEIGHT DISTRIBUTIONS





Song Han, Jeff Pool, John Tran, William J. Dally, Learning both Weights and Connections for Efficient Neural Networks, NIPS 2015, https://arxiv.org/abs/1506.02626

PRUNING GRANULARITY

Fine-grained pruning is most accurate

- Possibly difficult to exploit sparsity on massively-parallel processors

Coarse-grained pruning is fastest/most effective on processors

- Massive parallelization requires structure in the computation (see performance bugs for GPUs such as memory coalescing, branch divergence, vectorization for CPUs)
- Overhead on the example of compressed sparse row (CSR) coding

Directly addressable in dense format

$$\mathbf{D} \in \mathbb{R}^{M \times N} \begin{pmatrix} 0 & 5 & 3 & 0 \\ 6 & 1 & 0 & 4 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 4 \end{pmatrix}$$

Space complexity: 16 vs 21 elements

Indirect addressing in CSR format

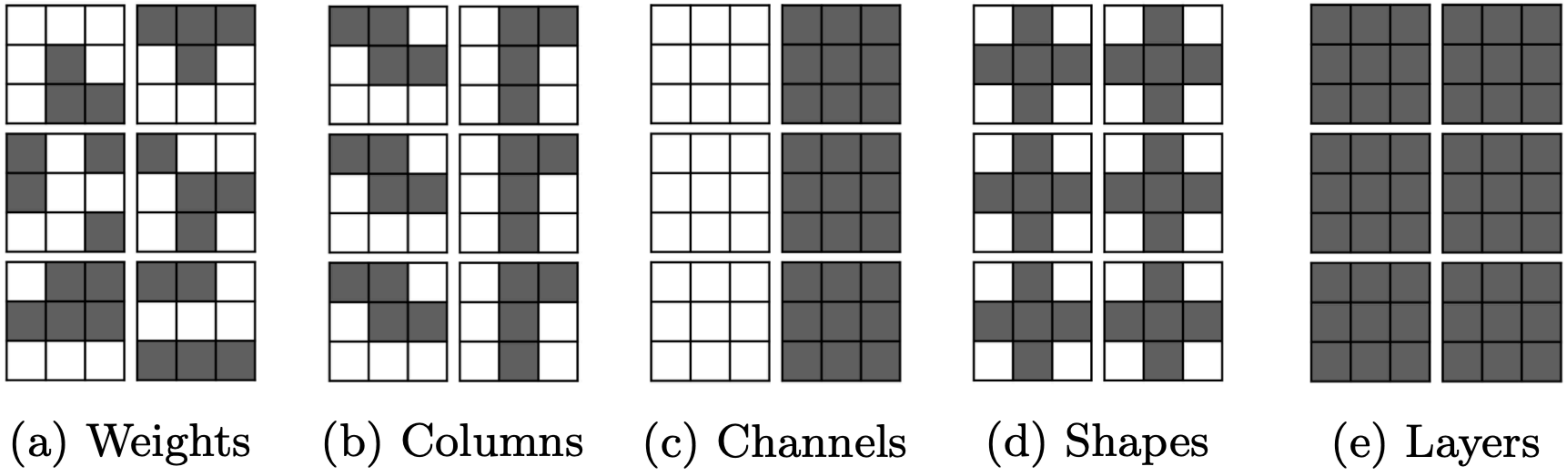
Row pointer $\mathbf{r} \in \mathbb{R}^{M+1} = (0 \quad 2 \quad 5 \quad 5 \quad 8)$

Column index $\mathbf{i} \in \mathbb{R}^{I} = (1 \quad 2 \quad 0 \quad 1 \quad 3 \quad 0 \quad 2 \quad 3)$

Data array $\mathbf{d} \in \mathbb{R}^{D} = (5 \quad 3 \quad 6 \quad 1 \quad 4 \quad 2 \quad 1 \quad 4)$

I and D are data-dependent

STRUCTURED PRUNING



Consider $\mathbf{z} = g(\mathbf{W} \oplus \mathbf{x})$

- For nonlinearity $g()$, weight tensor $\mathbf{W}$, input activation vector $\mathbf{x}$, linear operation $\oplus$ (e.g., convolution, fully-connected)

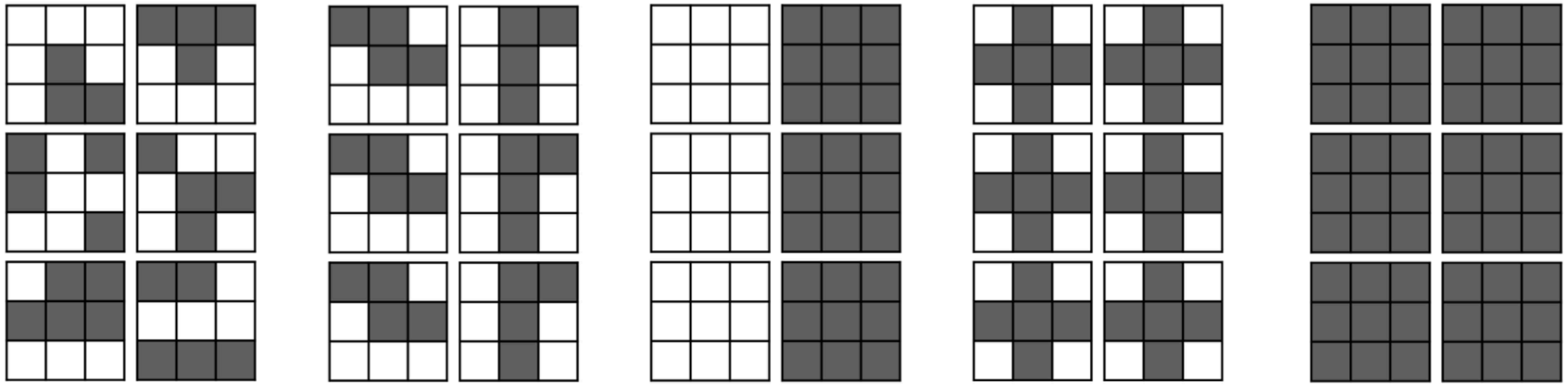
Divide tensor $\mathbf{W}$ into sub-tensors $\{\mathbf{w}_i\}$

- So that each $\mathbf{w}_i = (w_{i,j})_{j=1}^{m}$ constitutes the m weights of structure i
- Structures can be of arbitrary shape

Desired: learnable structured sparsity in $\mathbf{W}$ => parametrization of sub-tensors

- Make parametrization part of back propagation

PARAMETRIZED STRUCTURED PRUNING (PSP)



(a) Weights (b) Columns (c) Channels (d) Shapes (e) Layers

During forward propagation, substitute sub-tensors $\mathbf{w}_i$ with structure-sparse subtensor $\mathbf{q}_i = \mathbf{w}_i \cdot \alpha_i$

Gradient of structure parameter α_i is calculated using chain rule, thus descends towards the predominant direction of the weights

Pruning for L2 regularization based on thresholding function $\alpha_i(v_i) = \begin{cases} 0 : |v_i| < \epsilon \\ v_i : |v_i| \geq \epsilon \end{cases}$ for tunable threshold ϵ

As $v_i()$ is not differentiable, use STE instead: $\partial E / \partial v_i = \partial E / \partial a_i$

Backprop updates dense parameters v_i, so improperly pruned structures can reappear during training

Forward path uses sparse parameters $\alpha_i()$ instead

Günther Schindler, et al., Parameterized Structured Pruning for Deep Neural Networks, 6th International Conference on Machine Learning, Optimization, and Data Science (LOD 2020). Best paper finalist.

PARAMETRIZED STRUCTURED PRUNING (PSP)

Thus, gradient of α_i is calculated following the chain rule

- Trained together with weights using gradient descent based on loss J, but regularized and pruned independently

Update rule #1:

$$\Delta\alpha_i(t+1) := \mu\Delta\alpha_i(t) - \eta\frac{\partial J}{\partial\alpha_i(t)} - \lambda\eta \cdot \alpha_i(t)$$

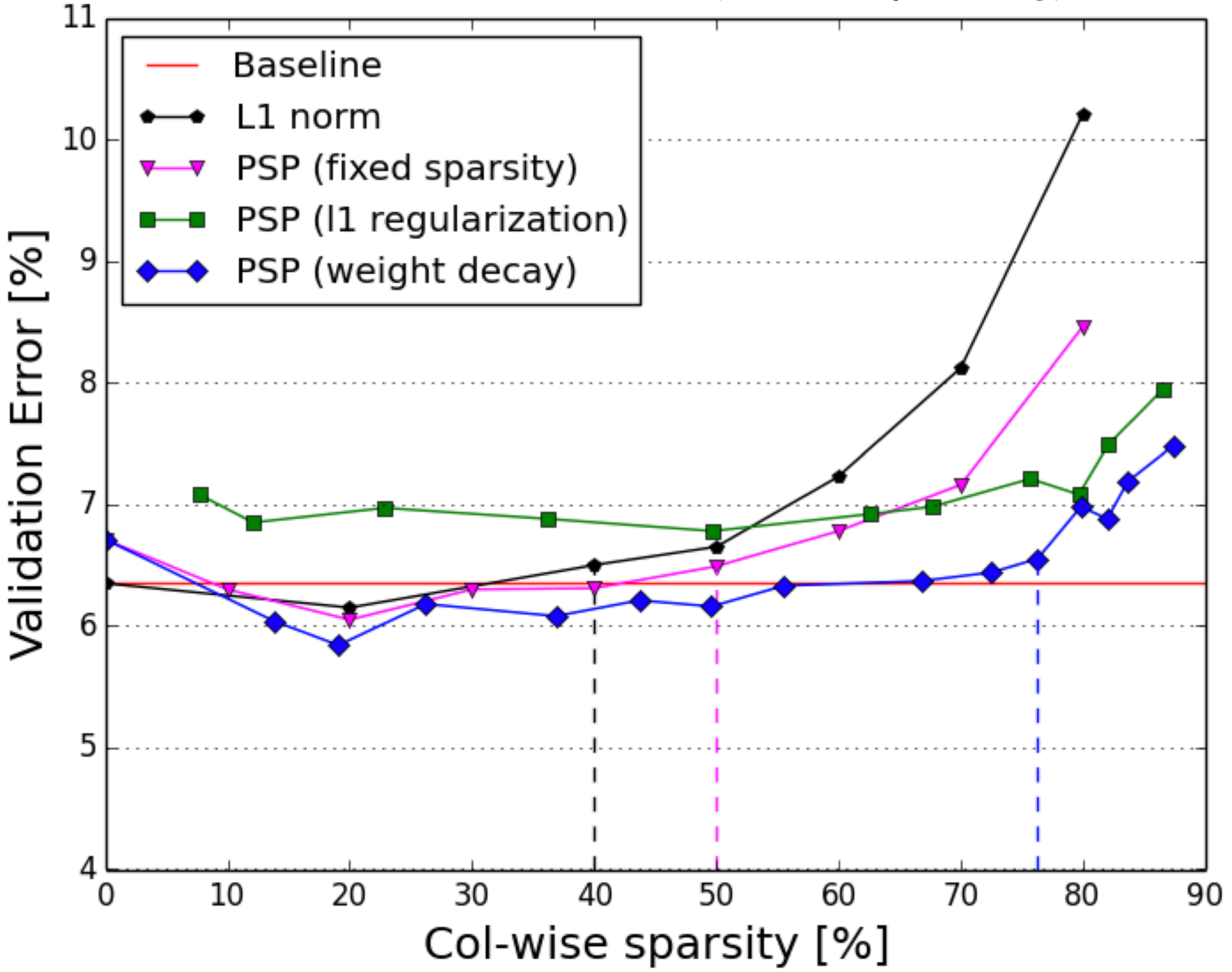
Update rule #2:

$$\Delta\alpha_i(t+1) := \mu\Delta\alpha_i(t) - \eta\frac{\partial J}{\partial\alpha_i(t)} - \lambda\eta \cdot \text{sign}(\alpha_i(t))$$

Surprisingly, option #1 performs better than option #2

- Different learning dynamics, seen in weight distributions
- L2 produces unimodal, bimodal and trimodal distributions with clear distinctions, while L1 lacks those distinctions

ResNet-56/CIFAR10 (column pruning)



Günther Schindler, et al., Parameterized Structured Pruning for Deep Neural Networks, 6th International Conference on Machine Learning, Optimization, and Data Science (LOD 2020). Best paper finalist.

MORE READING

Jonathan Frankle and Michael Carbin, ‘The lottery ticket hypothesis: Finding sparse, trainable neural networks’, in ICLR2018

- Hypothesis: inside a large network, only a sub-network together with its initialization makes the training effective (combination == “winning ticket”)
- Then: training the winning ticket in isolation is equal to the large network
- Example for unstructured pruning

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell, ‘Rethinking the value of network pruning’, ICLR2019, https://openreview.net/forum?id=rJlnB3C5Ym

- Contradicts the lottery ticket hypothesis
- Main differences: structured pruning, model architectures, rather large learning rate, data set complexity (from MNIST/CIFAR-10 to ImageNET)

WRAPPING UP

HARDWARE LOTTERY HYPOTHESIS

"Tooling [...] has played a disproportionately large role in deciding which ideas succeed and which fail"

HW determines which ideas succeed

- ANNs == matrix-matrix ops == excellent performance of GPUs
- Most ML researchers ignore hardware

Recent trends

- Convolutions and transformers (attention heads, based on softmax)
- GPT-3: 175B parameters (800GB of state); Alphafold-2: 23TB of training data

What if another processor was existing, e.g. excelling in processing large graphs?

- Probabilistic graphical models, sum-product networks, graph neural networks, etc.?

PROCESSOR SPECIALIZATION IS CONSIDERED HARMFUL FOR INNOVATION

ADDITIONAL READING

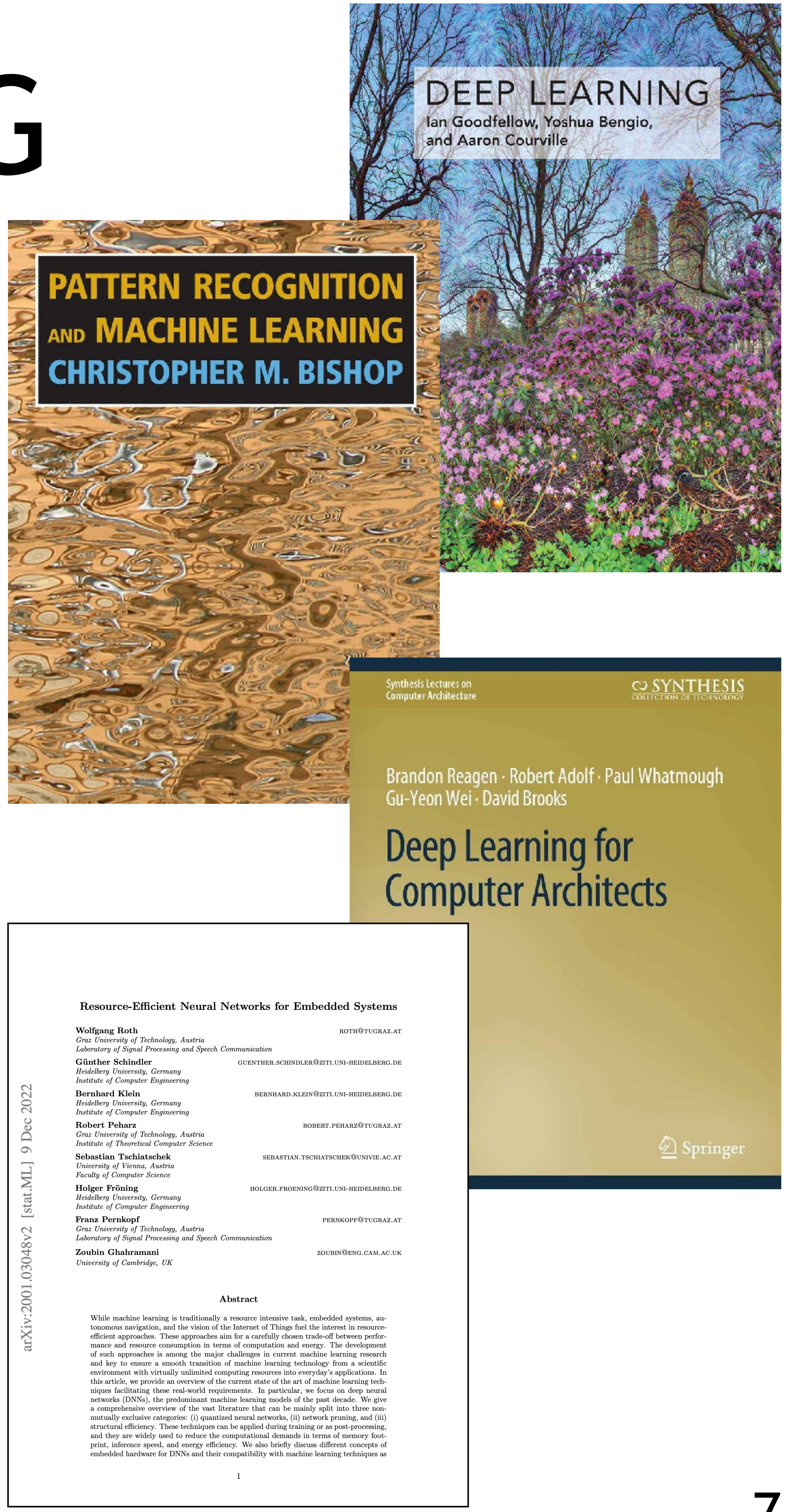
Recommended textbooks

- Goodfellow et al. - Deep Learning (https://www.deeplearningbook.org)
- Bishop - Pattern Recognition and Machine Learning (https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf)
- Reagan et al. - Deep Learning for Computer Architects (https://doi.org/10.2200/S00783ED1V01Y201706CAC041)

Wolfgang Roth, Günther Schindler, Bernhard Klein, Robert Peharz, Sebastian Tschiatschek, Holger Fröning, Franz Pernkopf, Zoubin Ghahramani, Resource-Efficient Neural Networks for Embedded Systems. ArXiv:2001.03048 [stat.ML], Dec. 2022. http://arxiv.org/abs/2001.03048

More information sources

- medium.com
- openreview.net
- paperswithcode.com



SUMMARY

Artificial NNs are universal function approximators

- Deep (many layers), thin (few parameters per layer), multi-branch (Inception, ResNet, DenseNet)
- Pervasively used, important for society

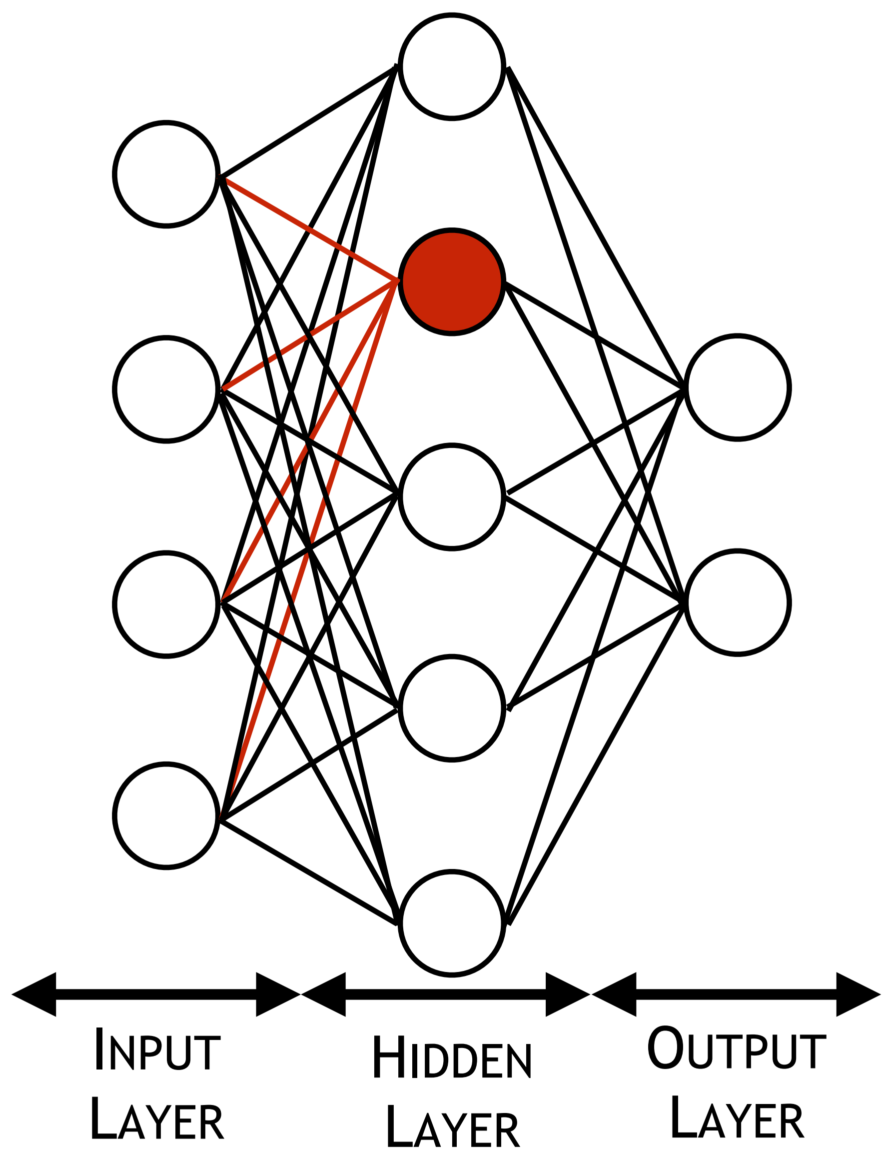
Playground for safe and unsafe optimizations

- Simplicity wall - plenty of structure and regularity (applies at least for most models as of today)
- Quantization and pruning as main methods for model compression, further include network architecture search and knowledge distillation
- Native support in PyTorch for (basic) pruning & quantization

Main pitfalls

- Model compression should always include re-training
- Accuracy is often only repeatable within a +/-1% interval
- Everything depends on model & data & HW

Open questions: uncertainty, truthworthiness, interpretability, democratization, continuous learning, ….



$$w_l^i = \begin{cases} W_l^p : w_l > \Delta_l \\ 0 : |w_l| \leq \Delta_l \\ -W_l^n : w_l < -\Delta_l \end{cases}$$

http://www.deepchip.org

REFERENCES I - QUANTIZATION

Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR, abs/1606.06160, 2016. URL http://arxiv.org/abs/1606.06160

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. CoRR, abs/1603.05279, 2016. URL http://arxiv.org/abs/1603.05279

Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. CoRR, abs/1602.02830, 2016. URL http://arxiv. org/abs/1602.02830

Fengfu Li and Bin Liu. Ternary weight networks. CoRR, abs/1605.04711, 2016. URL http: //arxiv.org/abs/1605.04711

Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. CoRR, abs/1612.01064, 2016. URL http:// arxiv.org/abs/1612.01064

Z. Cai, X. He, J. Sun, and N. Vasconcelos, “Deep learning with low precision by half-wave Gaussian quantization,” in Proc. CVPR, 2017.

V. Sze, T.-J. Yang, Y.-H. Chen, J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, 2017.

Brandon Reagen; Robert Adolf; Paul Whatmough; Gu-Yeon Wei; David Brooks; Margaret Martonosi, "Deep Learning for Computer Architects," in Deep Learning for Computer Architects , 1, Morgan & Claypool, 2017, doi:10.2200/S00783ED1V01Y201706CAC041

Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in Proc. ISCA, 2016, pp. 367–379.

Günther Schindler, Matthias Zöhrer, Franz Pernkopf, and Holger Fröning, Towards Efficient Forward Propagation on Resource-Constrained Systems, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2018).

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In European Conference on Computer Vision (ECCV), 2018.

REFERENCES II - ML PROCESSORS

Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in Proc. ISCA, 2016, pp. 367–379.

V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, “A 240 G-ops/s mobile coprocessor for deep neural networks,” in *CVPR Workshop*, 2014, pp. 682–687.

M. Sankaradas, et al., “A massively parallel coprocessor for convolutional neural networks,” in Proc. ASAP, 2009, pp. 53–60.

V. Sriram, D. Cox, K. H. Tsoi, and W. Luk, “Towards an embedded biologically- inspired machine vision processor,” in Proc. FPT, 2010, pp. 273–278.

S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, “A dynamically configurable coprocessor for convolutional neural networks,” in Proc. ISCA, 2010, pp. 247–257.

S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, “A 1.93 TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big- data applications,” in IEEE ISSCC Dig. Tech. Papers, 2015, pp. 1–3.

L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, “Origami: A convolutional network accelerator,” in Proc. GLVLSI, 2015, pp. 199–204.

Z. Du, et al., “ShiDianNao: Shifting vision processing closer to the sensor,” in Proc. ISCA, 2015, pp. 92–104.

S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in Proc. ICML, 2015, pp. 1737–1746.

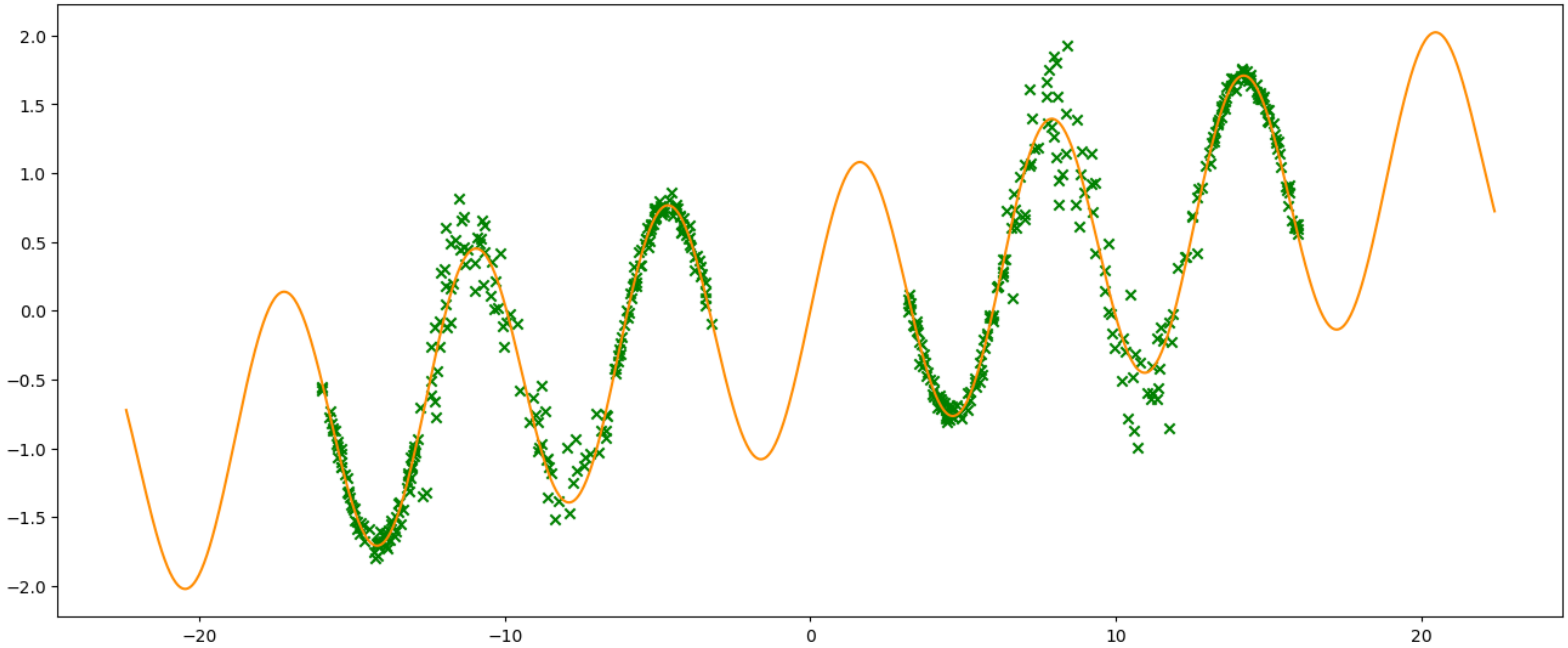
M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, “Memory-centric accelerator design for convolutional neural networks,” in Proc. ICCD, 2013, pp. 13–19.

C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” in Proc. FPGA, 2015, pp. 161–170.

T. Chen, et al., “DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” in Proc. ASPLOS, 2014, pp. 269–284.

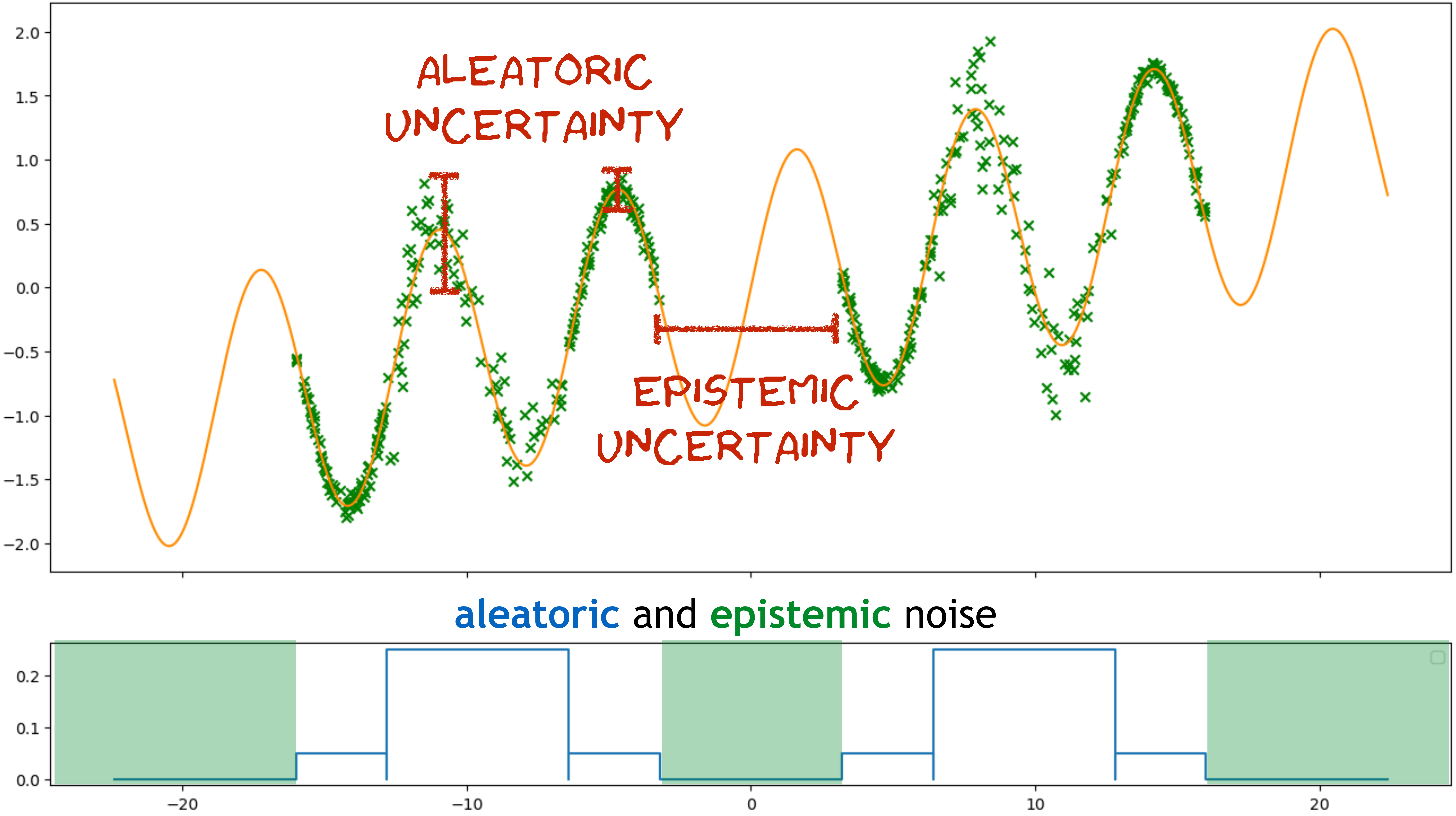
Y. Chen, et al., “DaDianNao: A machine- learning supercomputer,” in Proc. MICRO, 2014, pp. 609–622.

DISCUSSION



2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−20
−10
0
10
20

NEED TO ADDRESS UNCERTAINTY



Aleatoric uncertainty

- Inherent of the process
- Does not reduce with more data

Epistemic uncertainty

- Modeling uncertainty
- Decrease with more data or better models

PERFORMANCE SCALING

$$Perf(\frac{ops}{s}) = \frac{Instructions}{cycle} \cdot frequency$$

CLASSICAL DENNARD SCALING

$\propto PipelineCount \cdot PipelineDepth$

scales with feature size

$$Perf(\frac{ops}{s}) = Power(W) \cdot Efficiency(\frac{ops}{Joule})$$

POST DENNARD SCALING

fixed

REGIME I

operator cost

+

Specialization —> heterogeneity and asymmetry

GPU COMPUTING

PROGRAMMING COMPLEXITY

NUMA EFFECTS & LATENCY HIDING

REGIME II

data movement cost

SPATIAL ARCHITECTURES

FPGA, TPU, TENSORCORE

3 operands x 64bit/operand

LOCALITY

$$Energy = \#bits \cdot dist[mm] \cdot energy_{per\ bit, per\ mm}[\frac{J}{mm}]$$

Partly by Bill Dally, Sudha Yalamanchili (UCAA Workshop, 2012)

DISCUSSION: LEARN TO LOVE THE PICOJOULE

Integer		pJ
Add		
	8 bit	0.03
	32 bit	0.1
Mult		
	8 bit	0.2
	32 bit	3.1

FP		pJ
FAdd		
	16 bit	0.4
	32 bit	0.9
FMult		
	16 bit	1.1
	32 bit	3.7

Memory		pJ
Cache		(64 bit)
	8kB	10
	32kB	20
	1MB	100
	DRAM	1300 - 2600

~10x

15x

Computations: reducing precision, number format, ADD instead of MULT

ADD scales with n, MULT with n^2 (n=bit width)

Memory: exploit locality

NEED FOR REDUCED PRECISION, AVOID MEMORY ACCESSES

table data: M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC). doi: 10.1109/ISSCC.2014.675732

BOPS: BIT OPERATIONS

Wanted: abstract metric to compare different model compression techniques

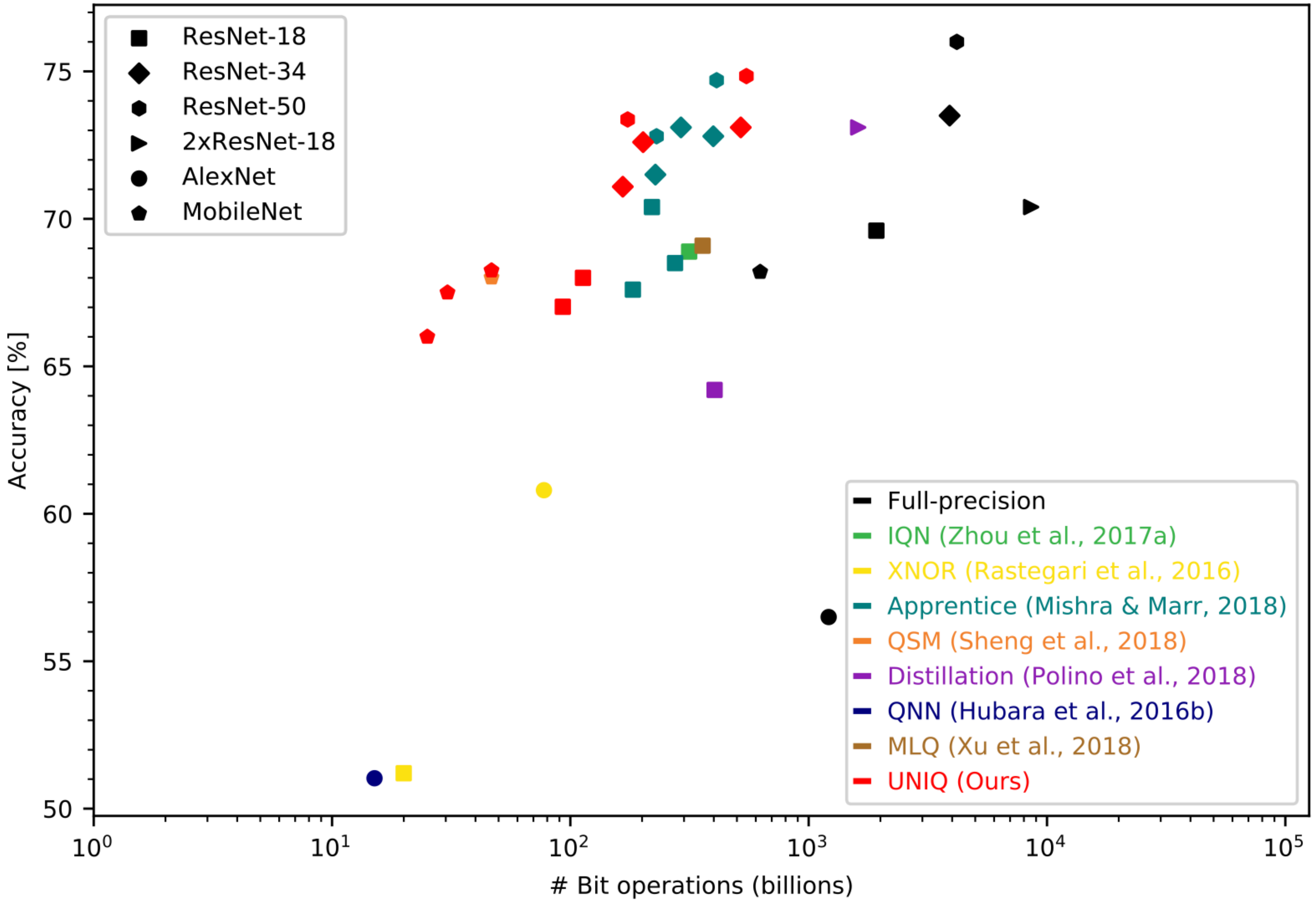
- MACs not appropriate for custom data types

For a convolutional layer with

- b_w bit weights, b_a bit activations, n input channels, m output channels, $k \times k$ filters
- Maximum output value is then about $2^{b_a+b_w}nk^2$
- Accumulator: $b_o = b_a + b_w + log_2(nk^2)$

BOPSconv $\approx mnk^2(b_a b_w + b_o)$

- Disclaimer: only for fixed point, floating point requires additional extensions



C. Baskin, N. Liss, E. Schwartz, E. Zheltonozhskii, R. Giryes, A. M. Bronstein, and A. Mendelson. UNIQ: Uniform Noise Injection for Non-Uniform Quantization of Neural Networks. ACM Trans. Comput. Syst., 2019, https://arxiv.org/abs/1804.10969

ANALOG (ELECTRONIC) COMPUTATIONS

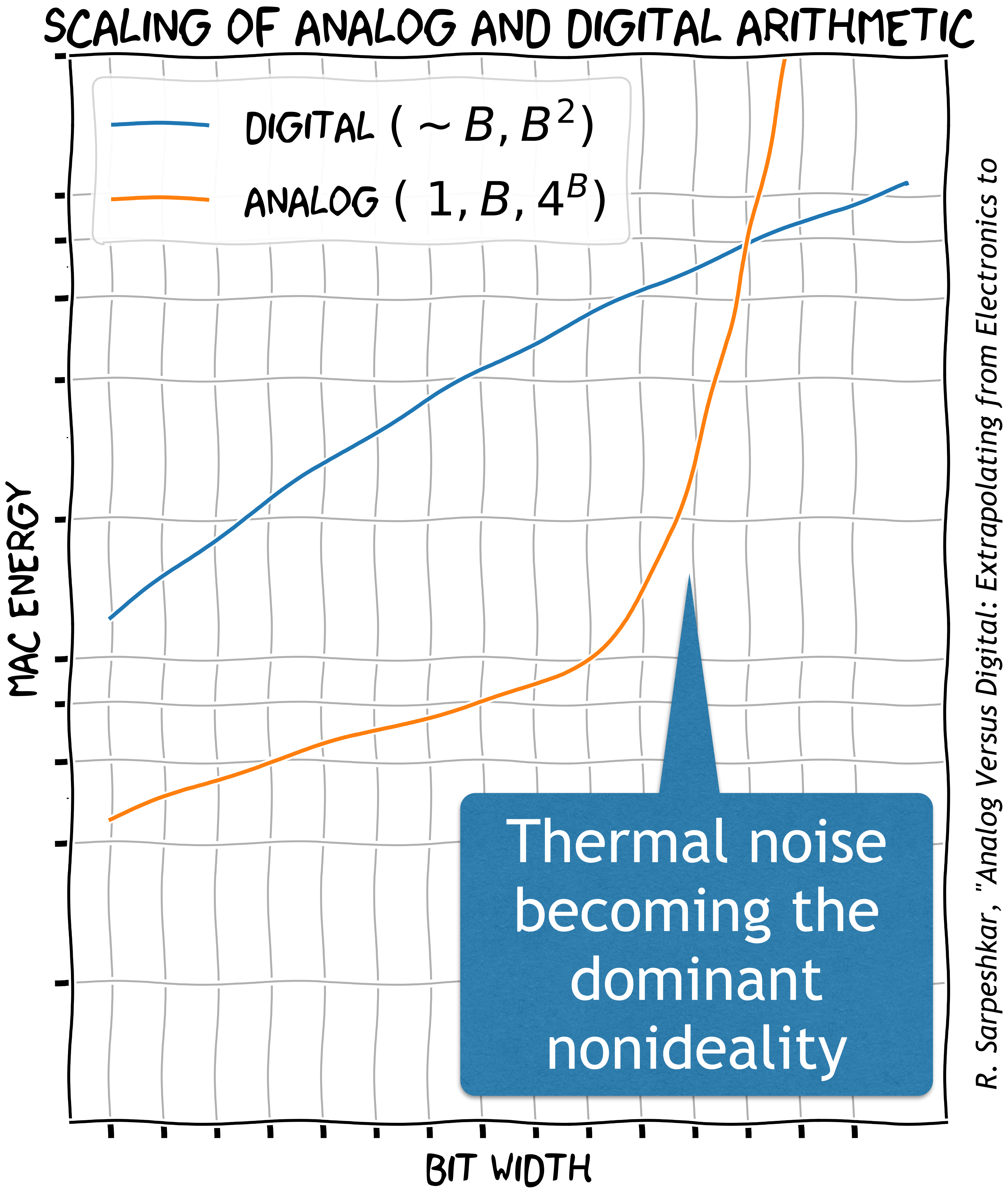
Energy efficiency

- Computations very efficient if thermal noise is non-dominant
- Data movements extremely cheap as often as simple as flow of electrons (current)

Noise

- Accumulation of noise
- Additive & multiplicative noise

Possibly even better for analog optical computing



R. Sarpeshkar, "Analog Versus Digital: Extrapolating from Electronics to Neurobiology," in Neural Computation, vol. 10, no. 7, pp. 1601-1638, 1 Oct. 1998, https://ieeexplore.ieee.org/document/6790538

	Digital	Analog
Compute with	discrete values of physical variables	continuous quantities of physical variables
Primitives	Boolean logic (easily automated, amount of computation per transistor low)	Physics of computing devices (transistors, capacitors, resistors), Kirchhoff's current and voltage laws
Wire	1bit of information per time unit	Possibly many bits per time unit
Computation resilience	Computation is not offset prone, insensitive to mismatches (physical device parameters), single bit error with catastrophic failures	Computation is offset prone, sensitive to physical device parameters, graceful degradation wrt errors
Noise	due to round-off error	due to thermal fluctuations
Signal restoration	to {0,1} after each stage	custom, but frequently mandatory
Noise accumulation	No, thus complex systems easy to build	Accumulates with cascading stages