The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Spring 2021

ADVANCED TOPICS IN COMPUTER VISION

Atlas Wang
Assistant Professor, The University of Texas at Austin

Visual Informatics Group@UT Austin
https://vita-group.github.io/

Deep Learning on the Edge

- Deploying CNNs on resource-constrained platforms/at the edge
- Two Scenarios: **Inference** (pre-trained model), and **Training** (online adaptation)

Real-Time Machine Learning (RTML)

PROGRAM SOLICITATION
NSF 19-566

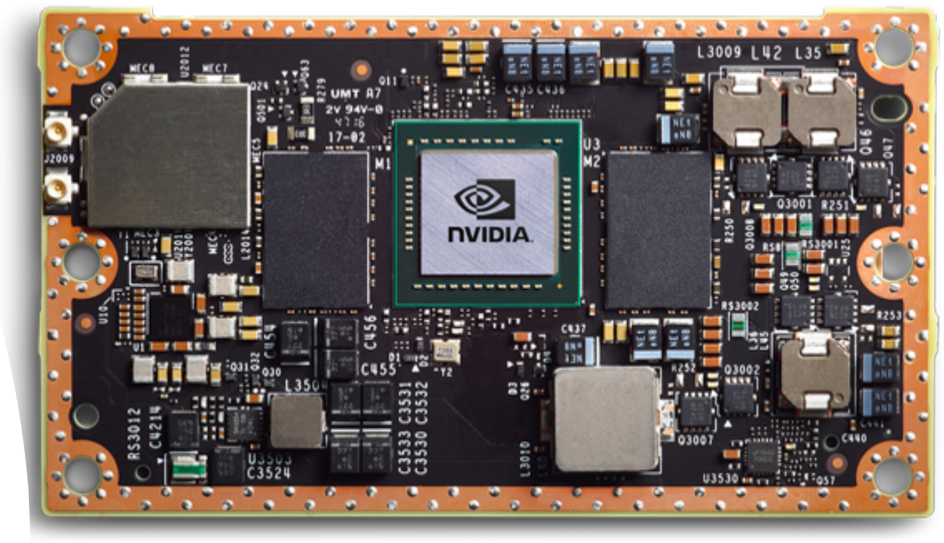
National Science Foundation

Directorate for Computer and Information Science and Engineering
Division of Computing and Communication Foundations

Directorate for Engineering
Division of Electrical, Communications and Cyber Systems

RTML Program goal: *"for next-generation **co-design** of RTML algorithms and hardware, with the principal focus on developing novel hardware architectures and learning algorithms in which **all stages of training** (including incremental training, hyperparameter estimation, and deployment) can be performed in real time."*

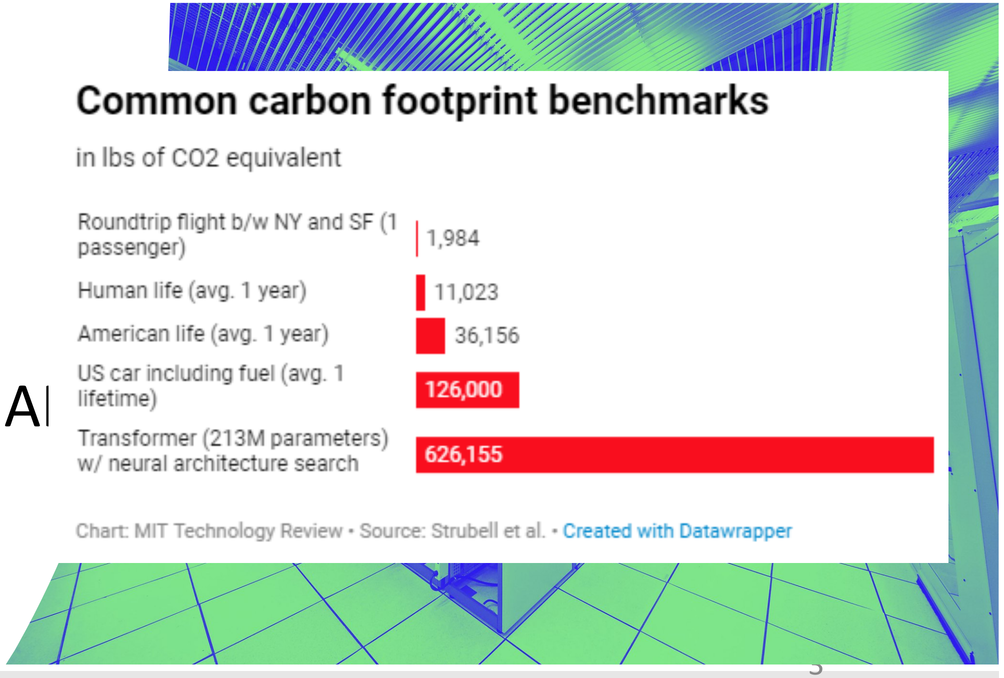
Deep Learning on the Edge



- Three Top Concerns:
  - **Storage and Memory**
  - **Speed or Latency**
  - **Energy Efficiency**

- The three goals all pursue “light weight”
- … but they are often **not aligned***
- … so need to **consider all** in implementation
- … and for both **Inference** and **Training**

- Broad economic viability requires energy efficient AI
- Energy efficiency of a brain is **100x better** than current SOTA hardware!



* Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks, IEEE ISSCC 2016

Model Compression

- Training Phase:
  - The easiest way to extract a lot of knowledge from the training data is to learn many different models in parallel.
  - 3B: Big Data, Big Model, Big Ensemble
  - Imagenet: 1.2 million pictures in 1,000 categories.
  - AlexNet: ~ 240Mb, VGG16: ~550Mb
- Testing Phase:
  - Want small and specialist models.
  - Minimize the amount of computation and the memory footprint.
  - Real time prediction
  - Even able to run on mobile devices.

Two Main Streams

- **“Transfer”:** How to transfer knowledge from big general model (teacher) to small specialist models (student)?
  - Example: “Distilling the Knowledge in a Neural Network”, G. Hinton et. al., 2015
- **“Compress”:** How to reduce the size of the same model, during or after training, without losing much accuracy.
  - Example: “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”, S. Han et. al., 2016
- **Comparison:** Knowledge Transfer provides a way to train a <u>new small model</u> inheriting from big general models, while Deep Compression Directly does the surgery on big models, using a pipeline: pruning, quantization & Huffman coding.

Knowledge Transfer/“Distillation”: Main Idea

- Introduce “Soft targets” as one way to transfer the knowledge from big models.
  - Classifiers built from a softmax function have a great deal more information contained in them than just a classifier;
  - The correlations in the softmax outputs are very informative.

- Hard Target: the ground truth label (one-hot vector)
- Soft Target: T is “temperature”, z is logit

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

- More information in soft targets

cow	dog	cat	car	
0	1	0	0	original hard targets

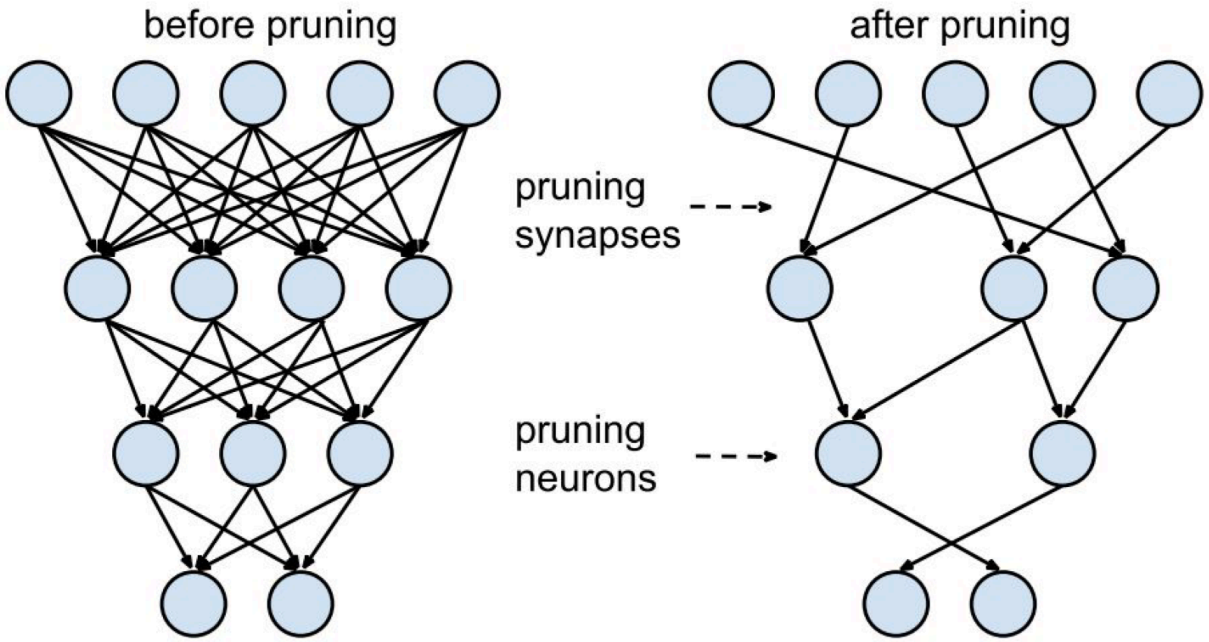
cow	dog	cat	car	
.05	.3	.2	.005	softened output of ensemble

Hinton’s Observation: If we can extract the knowledge from the data using very big models or ensembles of models, it is quite easy to distill most of it into a much smaller model for deployment.

More follow-up observations: teachers can be weak, or even the same as student …

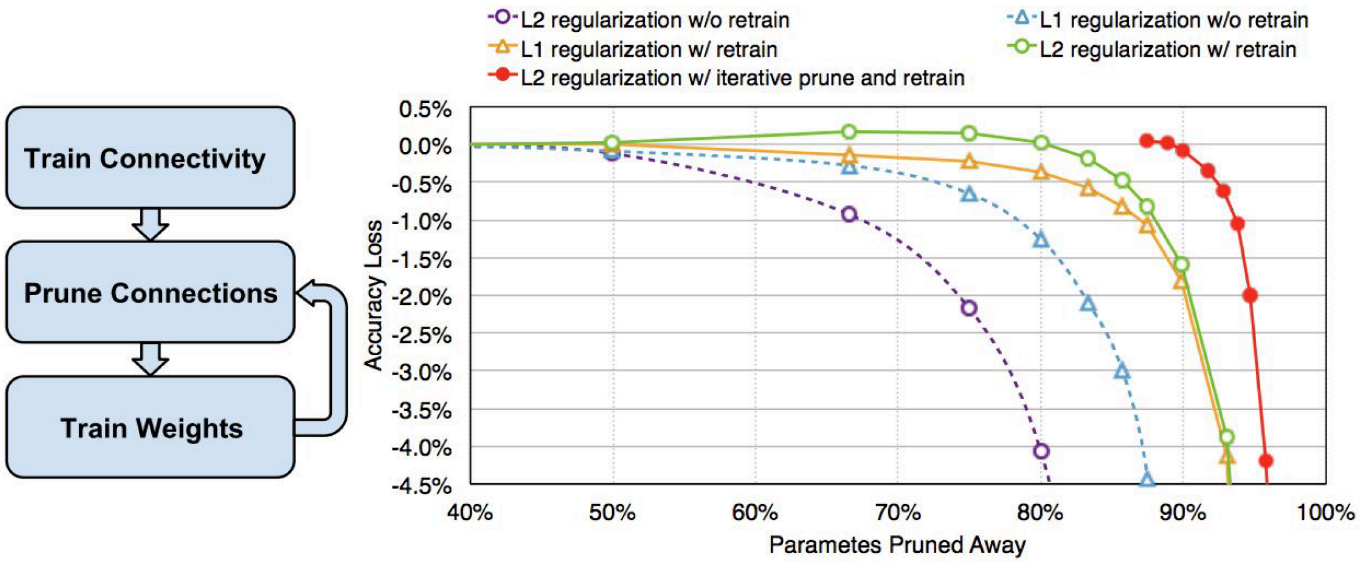
Deep Compression: Main Idea (i)

Pruning



Deep Compression: Main Idea (ii)

Retrain to Recover Accuracy



Network pruning can save 9x to 13x parameters without drop in accuracy

Deep Compression: Main Idea (iii)

Weight Sharing (Trained Quantization)

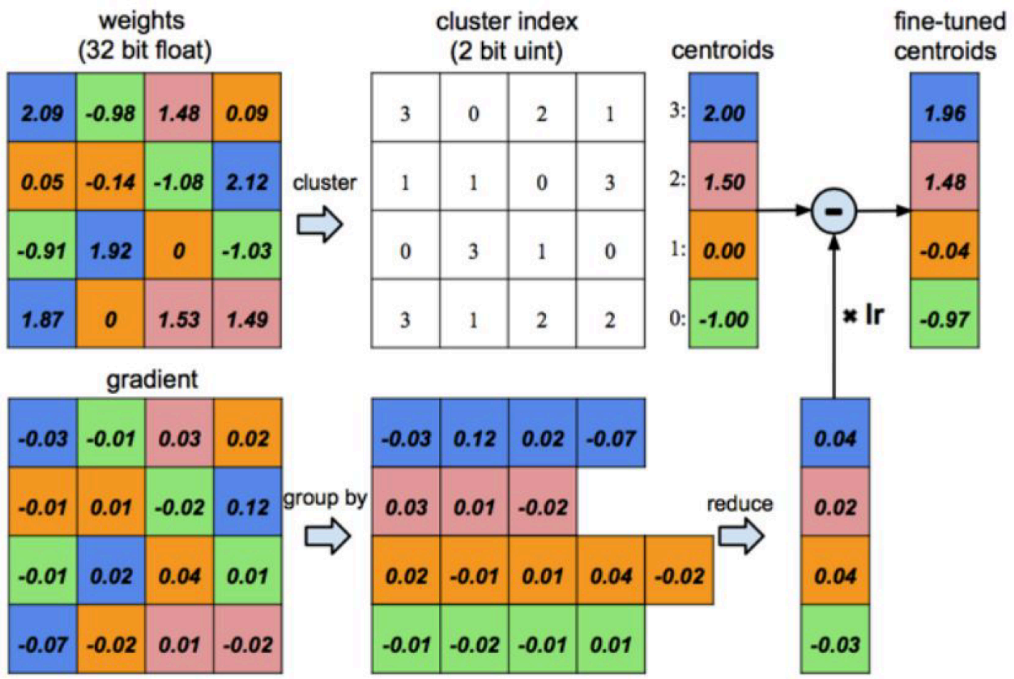
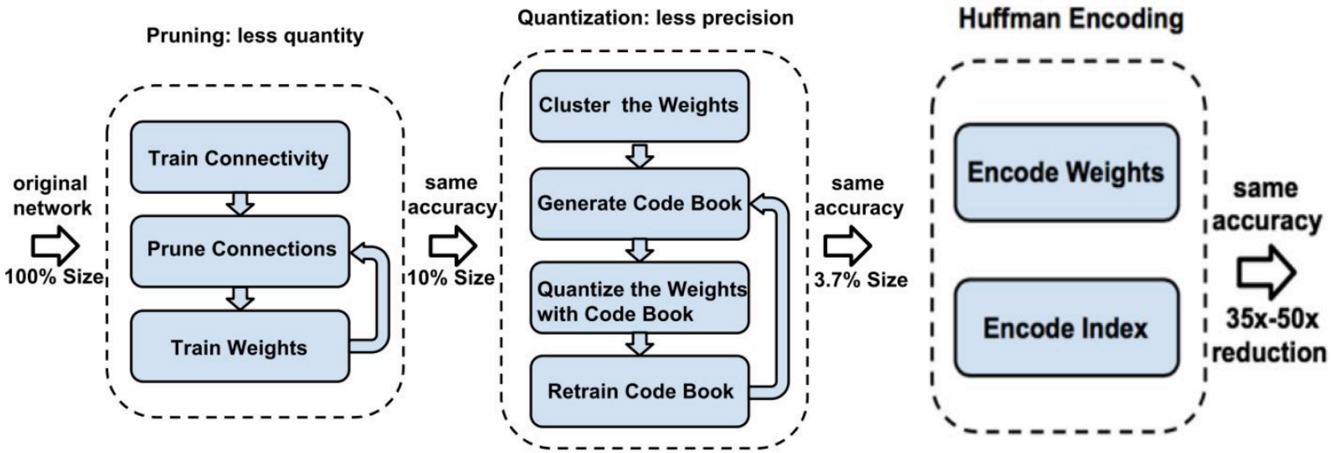


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom)

Deep Compression: Main Idea (iv)

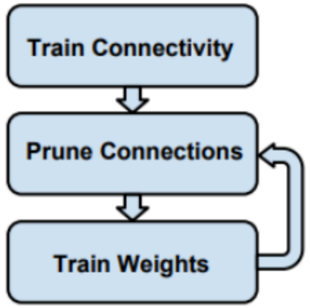
Huffman Coding



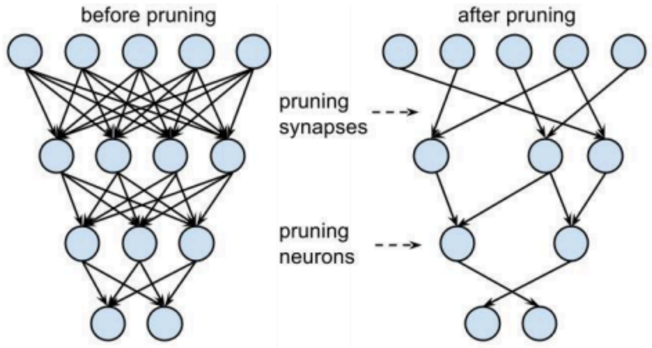
More About Pruning

- Determining **low-saliency parameters**, given a pre-trained network
- Follows the framework proposed by LeCun et al. (1990):

1. **Train** a deep model until convergence
2. **Delete** “unimportant” connections w.r.t. a certain criteria
3. **Re-train** the network
4. **Iterate** to step 2, or stop

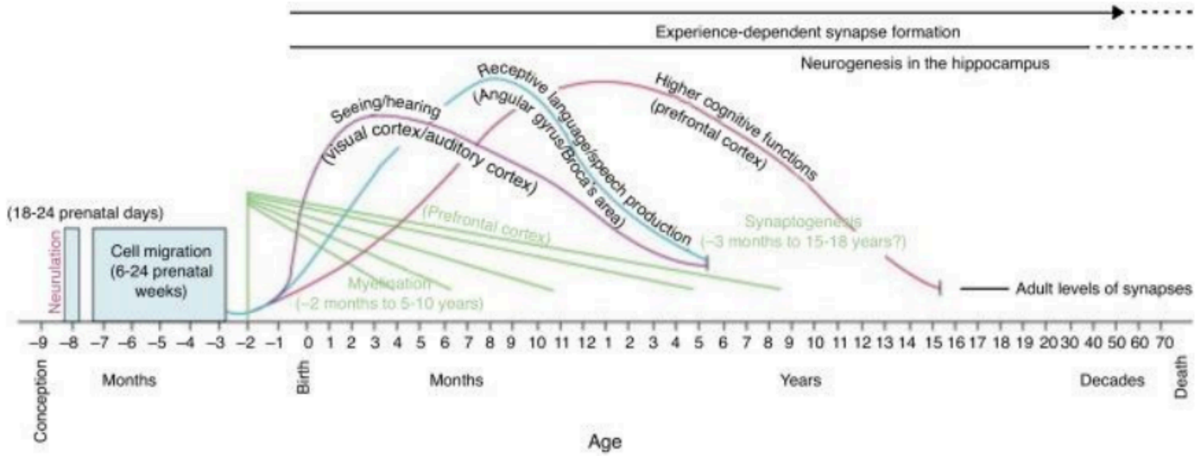


- Defining which connection is unimportant can vary
  - Weight magnitudes (L^2, L^1, …)
  - Mean activation [Molchanov et al., 2016]
  - Avg. % of Zeros (APoZ) [Hu et al., 2016]
  - Low entropy activation [Luo et al., 2017]
  - …



Human Brain Prunes too!

- Human brains are also using pruning schemes as well
- **Synaptic pruning** removes redundant synapses in the brain during lifetime



Optimal Brain Damage (OBD)

- Network pruning **perturbs weights $\mathbf{W}$** by **zeroing** some of them
- How the loss L would be changed when $\mathbf{W}$ is perturbed?

- **OBD** approximates L by the 2nd order Taylor series:

$$\delta L \simeq \underbrace{\sum_i \frac{\partial L}{\partial w_i}\delta w_i}_{\text{1st order}} + \underbrace{\frac{1}{2}\sum_i \frac{\partial^2 L}{\partial {w_i}^2}\delta w_i^2 + \frac{1}{2}\sum_{i,j} \frac{\partial^2 L}{\partial w_i \partial w_j}\delta w_i \delta w_j}_{\text{2nd order}} + O(||\delta \mathbf{W}||^3)$$

- **Problem:** Computing $H = \left(\frac{\partial L}{\partial w_i \partial w_j}\right)_{i,j}$ is usually intractable
  - Requires $O(n^2)$ on # weights
  - Neural networks usually have enormous number of weights
    - e.g. AlexNet: **60M** parameters $\Rightarrow H$ consists $\approx \mathbf{3.6 \times 10^{15}}$ elements

Optimal Brain Damage (OBD)

- **Problem:** Computing $H = \left(\frac{\partial L}{\partial w_i \partial w_j}\right)_{i,j}$ is usually intractable
- Two additional assumptions for tractability

1. **Diagonal** approximation: $H = \frac{\partial^2 L}{\partial w_i \partial w_j} = 0 \quad \text{if} \quad i \neq j$
2. **Extremal** assumption: $\frac{\partial L}{\partial w_i} = 0 \quad \forall i$
   - **W** would be in a local minima if it's pre-trained

- Now we get: $\delta L \simeq \frac{1}{2} \sum_i \frac{\partial^2 L}{\partial {w_i}^2} \delta w_i^2 + O(||\delta \mathbf{W}||^3)$
  - It only needs $\mathrm{diag}(H) := \left(\frac{\partial^2 L}{\partial w_i^2}\right)_i$
- $\mathbf{diag}(\boldsymbol{H})$ can be computed in $O(n)$, allowing a backprop-like algorithm
  - For details, see [LeCun et al., 1987]

Optimal Brain Damage (OBD)

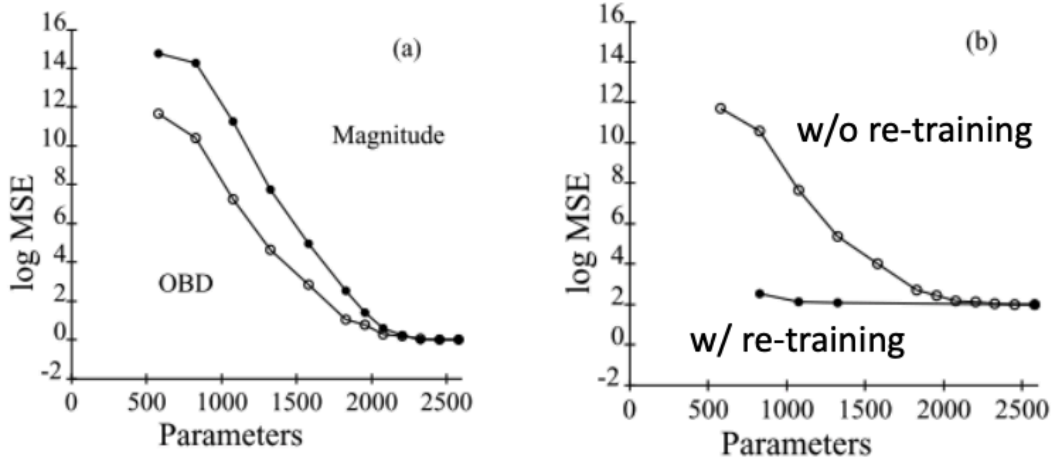
- How the loss L would be changed when $\mathbf{W}$ is perturbed?

$$L(\delta \mathbf{W}) \simeq \frac{1}{2} \sum_i \frac{\partial^2 L}{\partial {w_i}^2} \delta w_i^2 =: \sum_i \frac{1}{2} h_{ii} \delta w_i^2$$

- The **saliency** for each weight ⇒ $s_i := \frac{1}{2} h_{ii} |w_i|^2$

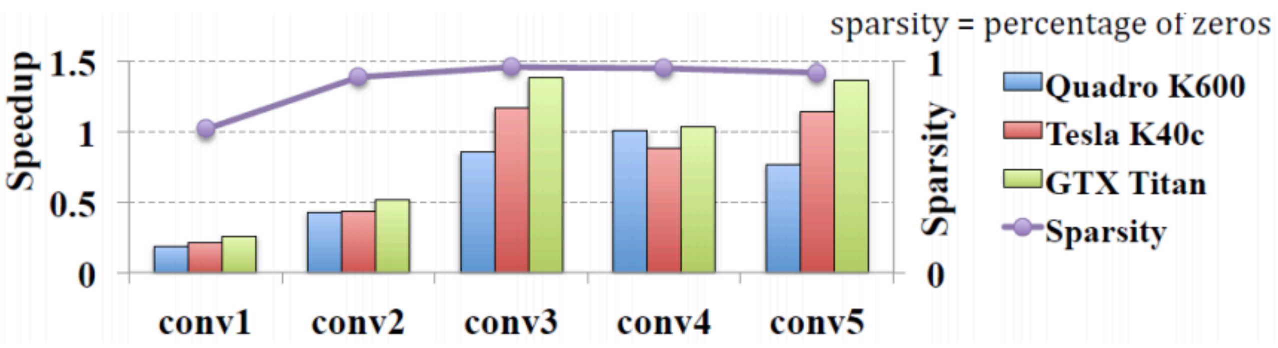
$$\boxed{s_i := |w_i|}$$

- OBD shows robustness on pruning compared to magnitude-based deletion
- After re-training, the original test accuracy is recovered



Structured Sparsity

- “Un-structured” weight-level pruning may not engage a **practical speed-up**
  - Despite of extremely high sparsity, actual speed-ups in GPU is limited



Speed-up ratio of weight-level pruning

5× speedup after concatenation of nonzero rows and columns

Structured sparsity

- **Structured sparsity** can be induced by adding group-lasso regularization

$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}) + \lambda \sum_{l=1}^{L} R_g(\mathbf{W}^{(l)}), \;\; R_g(\mathbf{w}) = \sum_{g=1}^{G} \|\mathbf{w}^{(g)}\|_2$$

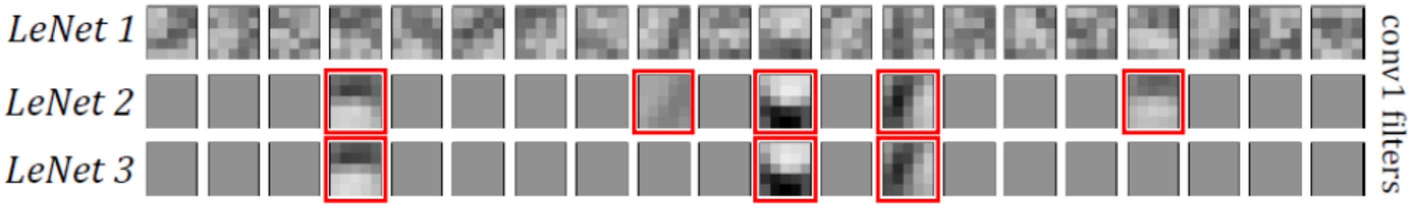
- **Filter-wise** and **channel-wise:**

$$R_g(\mathbf{W}^{(l)}) = \sum_{n_l=1}^{\overset{\text{\# filters}}{N_l}} \|\mathbf{W}^{(l)}_{n_l,:,:,:}\|_2 + \sum_{c_l=1}^{\overset{\text{\# channels}}{C_l}} \|\mathbf{W}^{(l)}_{:,c_l,:,:}\|_2$$

Table 1: Results after penalizing unimportant filters and channels in *LeNet*

LeNet #	Error	Filter # §	Channel # §	FLOP §	Speedup §
1 (*baseline*)	0.9%	20—50	1—20	100%—100%	1.00×—1.00×
2	0.8%	5—19	1—4	25%—7.6%	1.64×—5.23×
3	1.0%	3—12	1—3	15%—3.6%	1.99×—7.44×

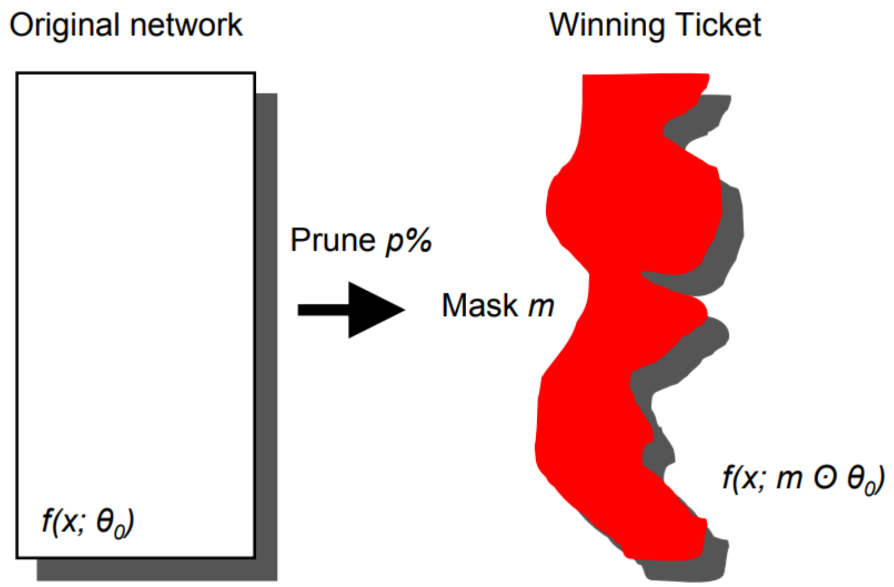
§In the order of *conv1*—*conv2*



Fewer but smoother feature extractors

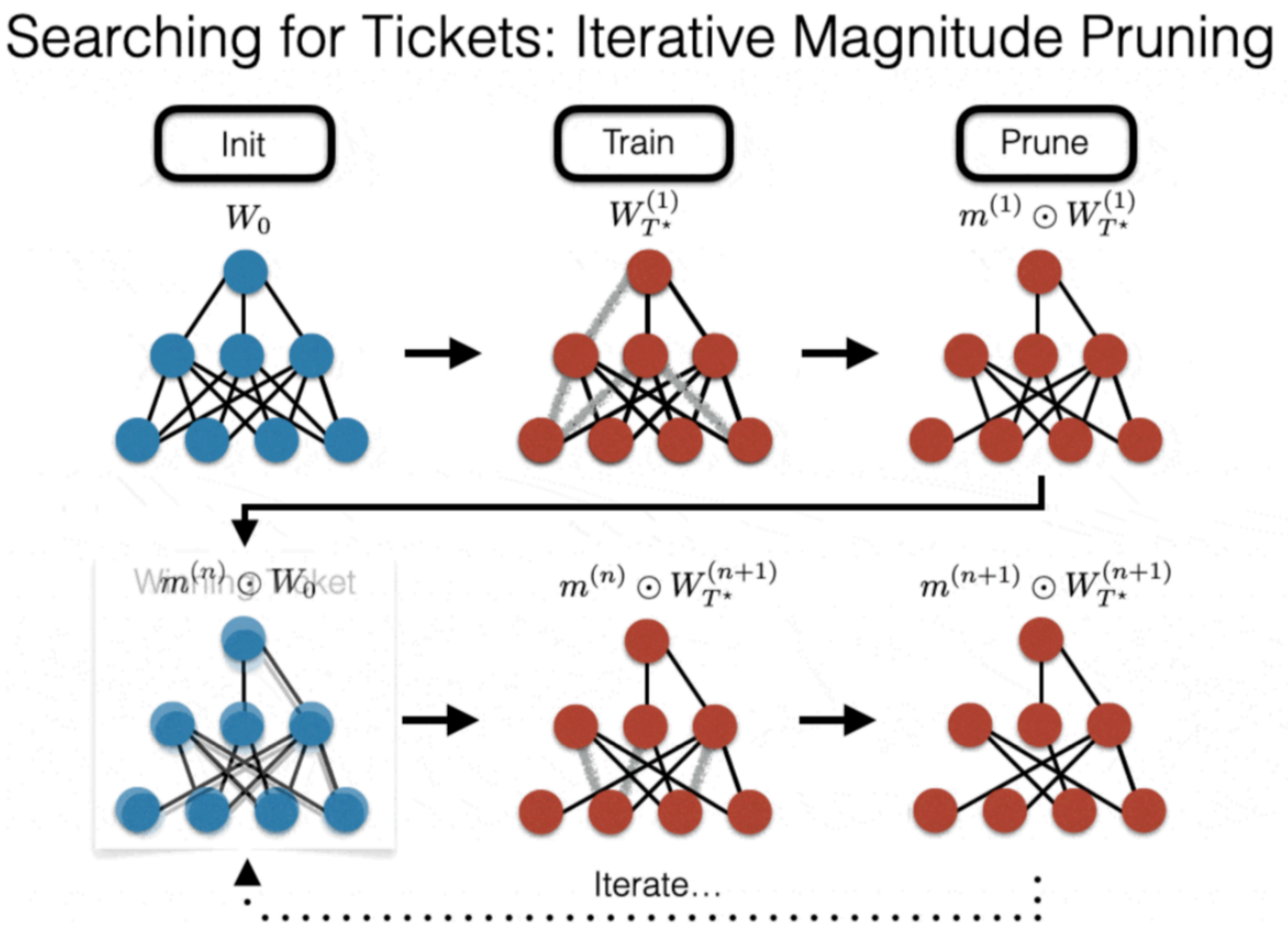
Lottery Ticket Hypothesis

The Lottery Ticket Hypothesis. *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.*



- Winning Ticket gives
  - Better or same results
  - Shorter or same training time
  - Notably fewer parameters
  - Is trainable from the beginning

Lottery Ticket Hypothesis



Lottery Ticket Hypothesis on Big Pre-Trained Models


MIT News
ON CAMPUS AND AROUND THE WORLD
SUBSCRIBE

Shrinking massive neural networks used to model language

A new approach could lower computing costs and increase accessibility to state-of-the-art natural language processing.

Daniel Ackerman | MIT News Office
December 1, 2020

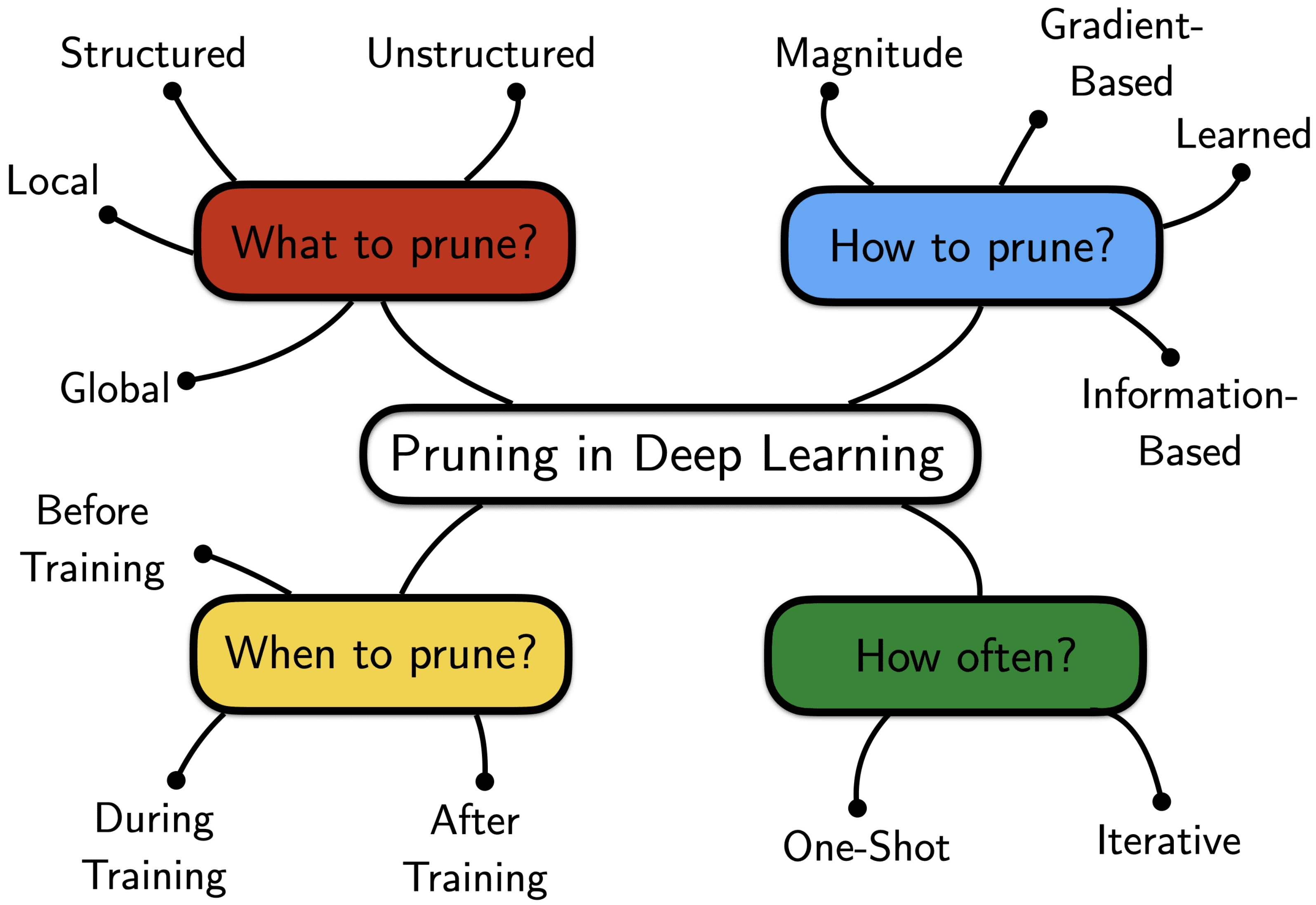
The Lottery Ticket Hypothesis for Pre-trained BERT Networks

Tianlong Chen[1], Jonathan Frankle[2], Shiyu Chang[3], Sijia Liu[3], Yang Zhang[3], Zhangyang Wang[1], Michael Carbin[2]
[1]University of Texas at Austin, [2]MIT CSAIL, [3]MIT-IBM Watson AI Lab, IBM Research
{tianlong.chen,atlaswang}@utexas.edu,{jfrankle,mcarbin}@csail.mit.edu,
{shiyu.chang,sijia.liu,yang.zhang2}@ibm.com

Subnetworks on the Source Tasks (Sparsity %)	MNLI	QQP	STS-B	WNLI	QNLI	MRPC	RTE	SST-2	CoLA	SQuAD v1.1	MLM	
MNLI (70%)	82.56	89.20	84.77	47.89	87.34	72.14	60.75	90.83	11.19	82.90	57.52	2
QQP (70%)	80.87	89.95	84.18	52.11	87.27	72.30	60.17	88.80	16.50	81.57	57.64	1
STS-B (70%)	80.05	88.26	87.34	56.34	86.17	72.71	57.40	87.92	4.31	80.74	57.59	1
WNLI (70%)	79.70	87.52	67.13	53.87	84.96	69.90	55.23	87.27	0.00	80.31	57.75	1
QNLI (70%)	80.80	88.75	83.16	54.93	88.89	71.73	58.96	89.56	3.65	82.44	57.47	3
MRPC (70%)	79.98	87.88	81.25	56.34	85.66	75.57	54.87	88.15	7.48	79.89	57.74	1
RTE (70%)	80.18	88.18	79.50	55.87	86.49	71.57	58.37	88.15	1.55	80.77	57.78	1
SST-2 (70%)	80.15	88.44	77.61	53.99	85.77	70.67	56.92	89.99	7.52	81.05	57.76	2
CoLA (70%)	80.06	88.29	77.48	54.93	86.30	70.83	55.60	88.57	38.89	81.01	57.81	1
SQuAD v1.1 (70%)	80.90	88.90	84.09	53.99	89.40	72.06	59.93	90.18	8.03	86.37	57.47	3
(IMP) MLM (70%)	82.59	90.03	87.43	55.05	89.44	81.58	59.81	91.86	47.15	86.54	63.16	6
Pruning θ_0 (70%)	82.46	89.62	85.28	53.52	89.13	72.55	58.84	91.06	32.21	85.33	57.09	4

Transfer Tasks

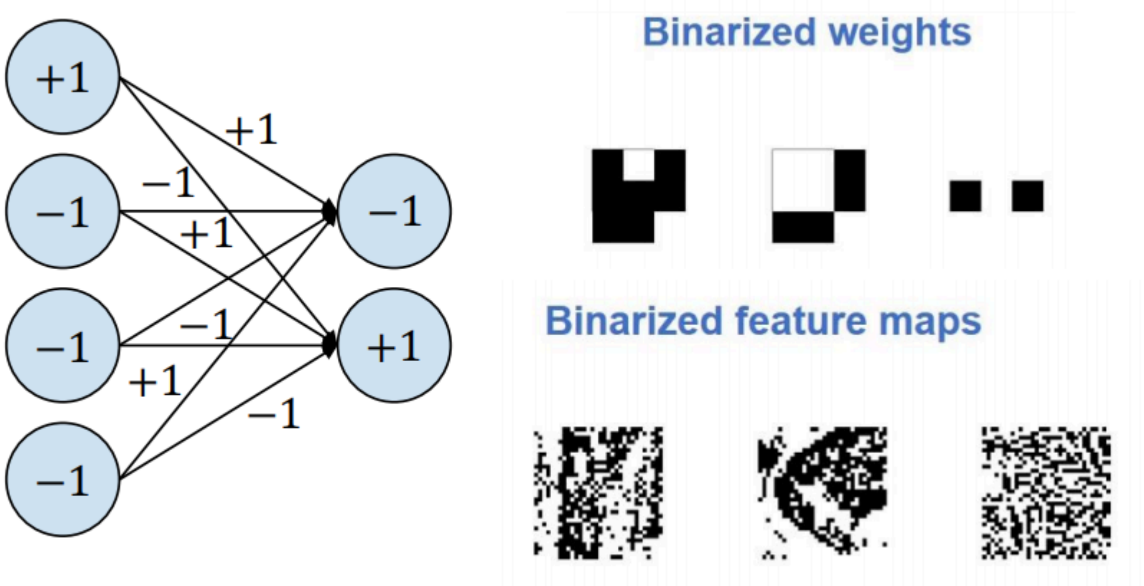
Summary of Pruning



More About Quantization

- Neural networks can be even **binarized** (**+1** or **-1**)
  - DNNs trained to use binary weights and binary activations
- Expensive **32-bit MAC** (**M**ultiply-**AC**cumulate) ⇒ Cheap **1-bit XNOR-Count**
  - "MAC == XNOR-Count": when the weights and activations are ± 1

1s in bits



Binary Neural Networks

- **Idea**: Training real-valued nets (W_r) treating binarization (W_b) as noise
  - Training W_r is done by **stochastic gradient descent**

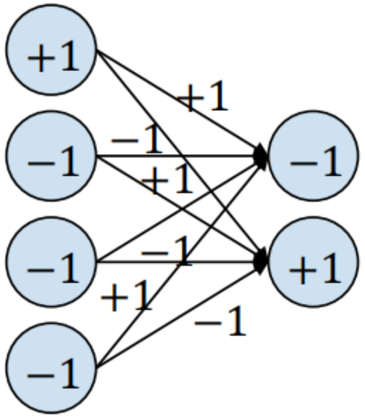
- **Binarization** ($W_r \rightarrow W_b$) occurs for each forward propagation
  - On each of **weights**: $W_b = \mathrm{sign}(W_r)$
  - ... also on each **activation**: $a_b = \mathrm{sign}(a_r)$

- Gradients for W_r is estimated from $\frac{\partial L}{\partial W_b}$ [Bengio et al., 2013]
  - "Straight-through estimator": Ignore the binarization during backward!

$$\frac{\partial L}{\partial W_r} = \frac{\partial L}{\partial W_b} \mathbf{1}_{|W_r| \leq 1}$$

$$\frac{\partial L}{\partial a_r} = \frac{\partial L}{\partial a_b} \mathbf{1}_{|a_r| \leq 1}$$

  - Cancelling gradients for better performance
    - When the value is too large

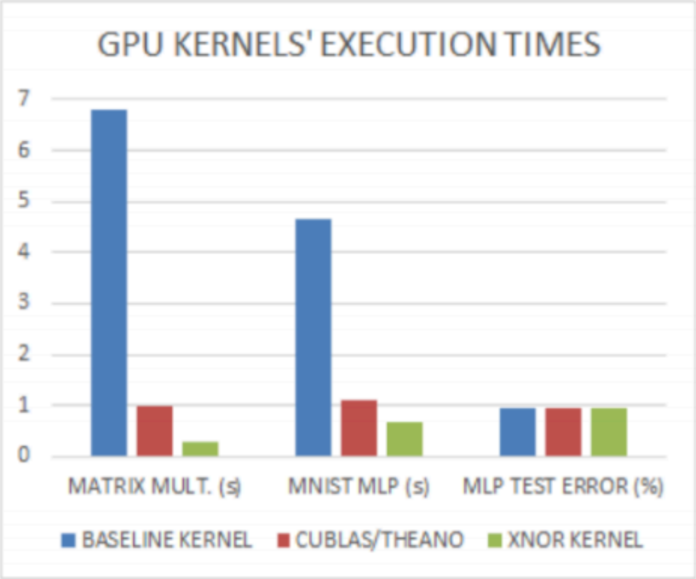


Binary Neural Networks

- BNN yields **32x less memory** compared to the baseline 32-bit DNNs
  - … also expected to reduce energy consumption drastically

- **23x faster** on kernel execution times
  - BNN allows us to use XNOR kernels
  - **3.4x** faster than cuBLAS

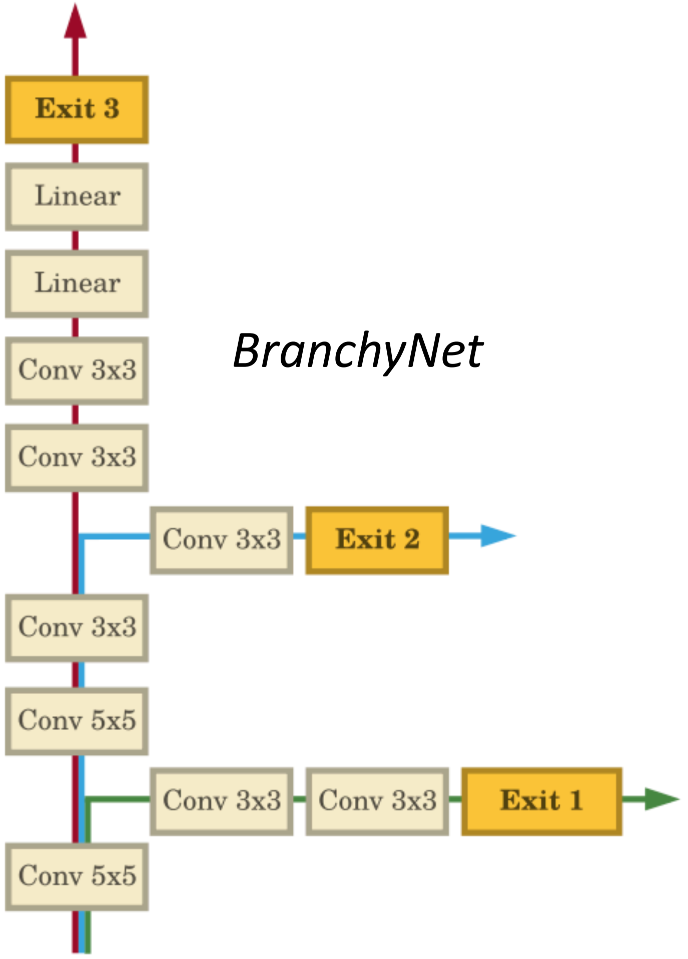
Operation	MUL	ADD
8bit Integer	0.2pJ	0.03pJ
32bit Integer	3.1pJ	0.1pJ
16bit Floating Point	1.1pJ	0.4pJ
32tbit Floating Point	3.7pJ	0.9pJ



- **BNN** achieves comparable error rates over existing DNNs

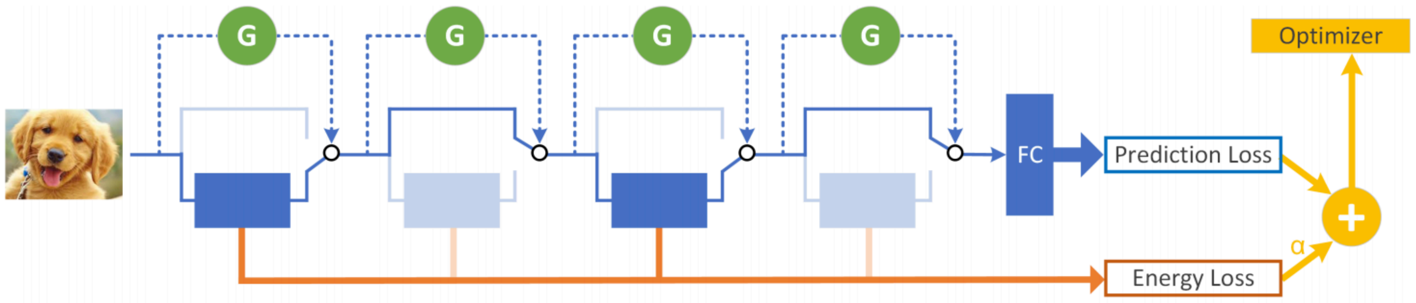
Dynamic Inference

- Only execute a fraction of the network per needed
- Can enable both “input-dependent” and “resource-dependent” forms



BranchyNet

SkipNet



Real-World Efficient ML: Way to Go

- Jointly utilizing several compression means
  - Also can choose efficient “by-design” models (MobileNets, or even non-deep models, etc.)
- **Data processing** is often a key concern, maybe more important
- **Hardware co-design** is another key concern
- Resource constraints & user demands often **change over time**

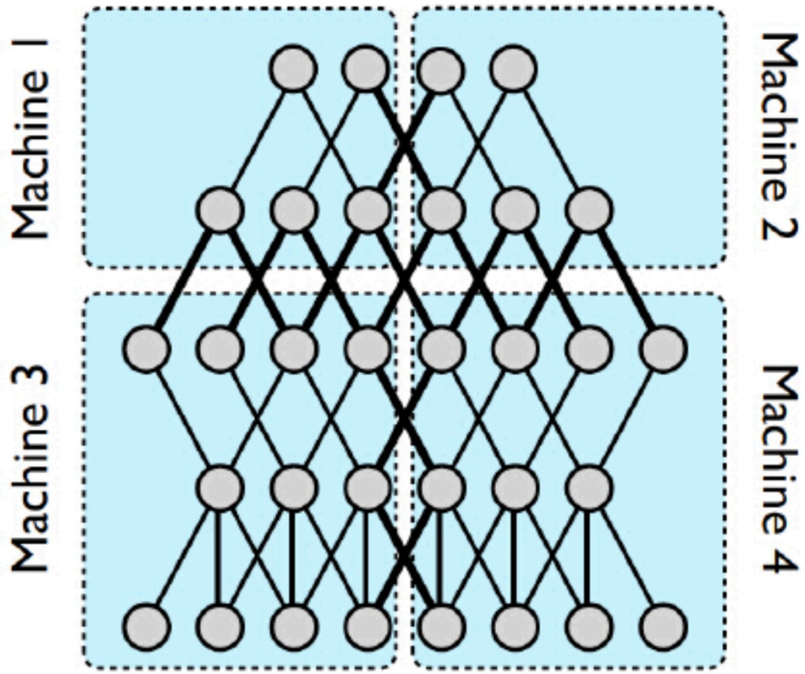
Demo: Energy-Efficient UAV-Based Text Spotting System

- **Task:** accurate detecting signs and recognizing texts in the video, captured by an unmanned aerial vehicle (UAV), with minimal energy cost as possible (**Hardware:** Raspberry Pi 3B+)
- Our solution won 2nd prize in the high-visibility IEEE CVPR **2020 Low-Power Computer Vision (LPCV) Challenge,** among 11 university & company teams that submitted 84 independent solutions.

SPONSORS

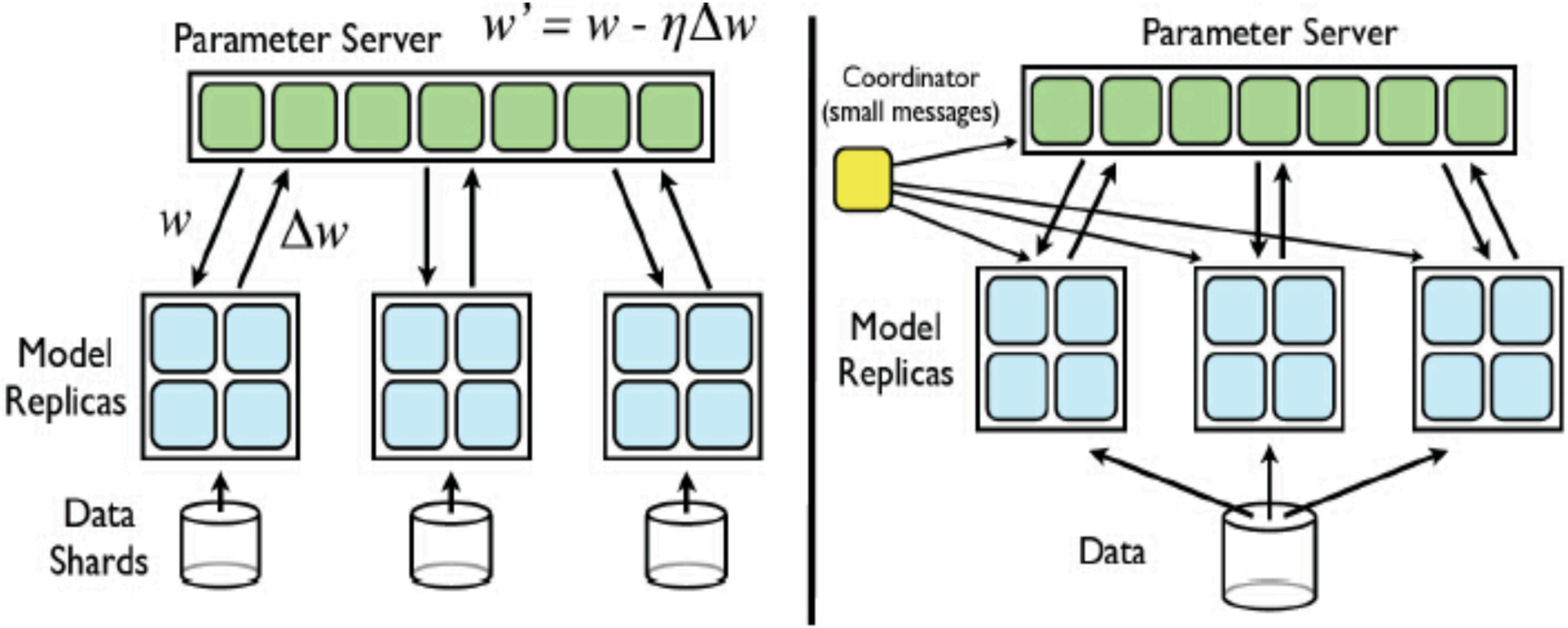
2020 Low-Power Computer Vision Challenge

Model Parallelism



- Deep net is stored and processed on multiple cores (multi-thread) or machines (message passing)
- Performance benefit depends on connectivity structure vs. computational demand

Data Parallelism



Asynchronous SGD

Distributed L-BFGS

From Inference to Training: Lessons and Challenges

- **Training v.s. Inference:** one-pass feedforward v.s. iterative forward + backward

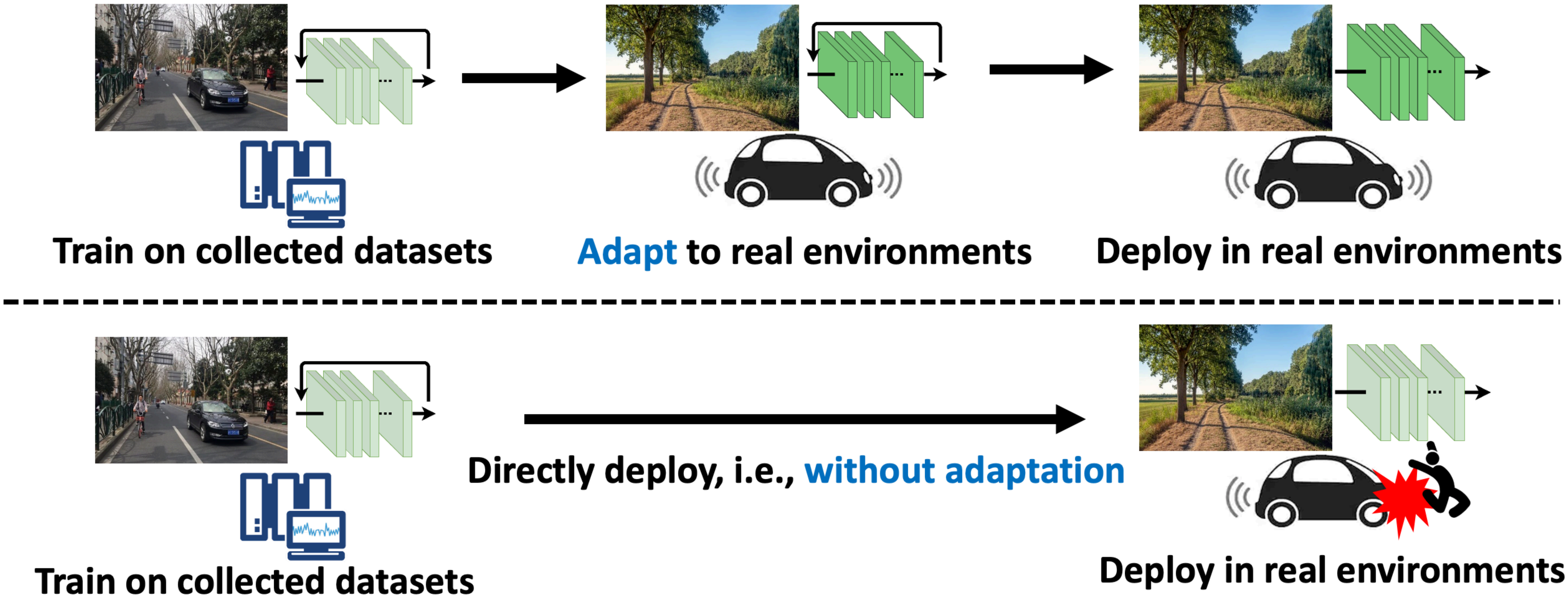
- **Lessons that we learned from Inference:**
  - Model parameters are not born equally, and many redundancies do exist
  - *Know your specific goal:* saving memory, latency and energy are often not aligned
  - To achieve energy goal, realistic energy models and/or hardware measurements are very helpful
  - Consider a more "end-to-end" effort beyond just the model itself (data, hardware, architecture…)

- **New Challenges posed for Training:**
  - Saving per-sample (mini-batch) complexity (both feed-forward and backward)
  - The empirical convergence (how many iterations needed) matters more than per-MB complexity
  - Data access/movement bottlenecks are (even more) crucial

Energy-Efficient Training: Prevailing Demands

- Shifting model training from the cloud to the edge
  - Facilitating personalization; saving bandwith/communication energy; protecting privacy
- Deep learning has a terrible carbon footprint
  - "**Training a single AI model can emit as much carbon as five cars in their lifetimes**", MIT Tech Review

On-Device Training (Adaptation) is on Growing Demand



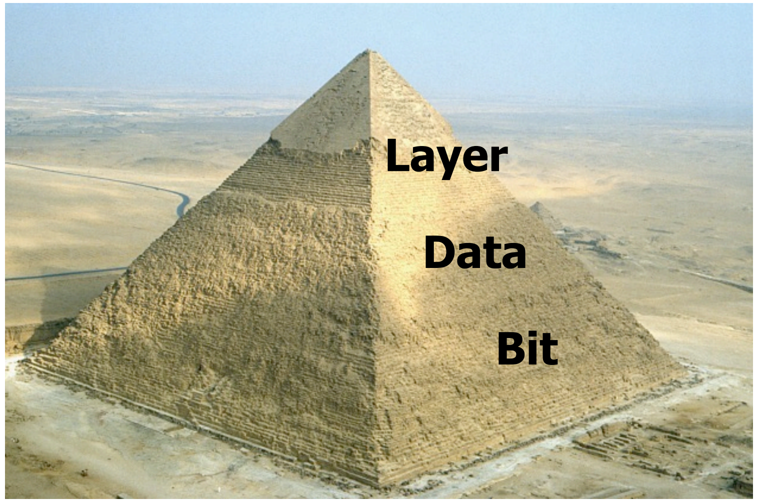
Problem Setting

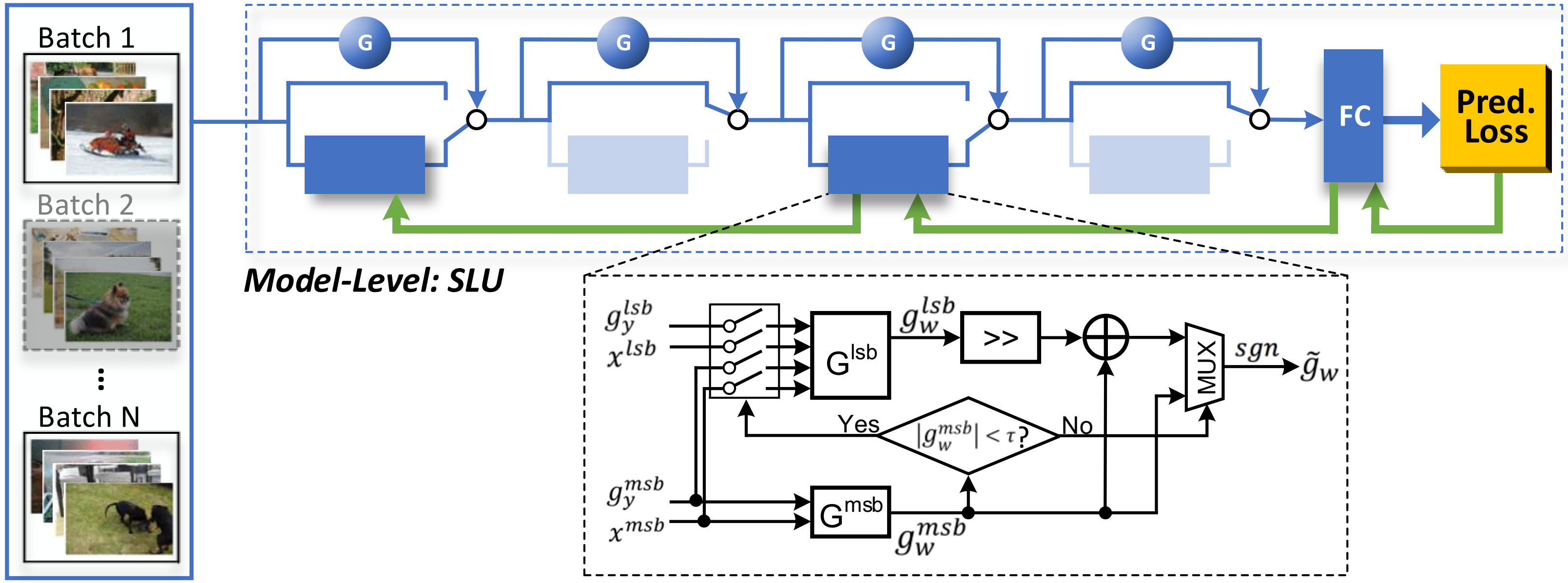
- We consider the most basic CNN training, assuming both the model structure and the dataset to be pre-given, training from scratch
  - Trim down the total energy cost for in-situ, resource-constrained training.
  - not usually the realistic IoT case, but address it as a starting point

- Many existing works are on accelerated CNN training
  - … they mostly focus on reducing the total training time in resource-rich settings, such as by distributed training in **large-scale** GPU clusters

E2-Train: Energy-Efficient CNN Training [NeurIPS'2019]

Motivation:





Model-Level: SLU

Data-Level: SMD

Bit-level: PSG

"Three-Pronged" Approach:

- **Data-Level:** stochastic mini-batch dropping
- **Layer-Level:** selective layer update
- **Bit-Level:** predictive sign gradient descent

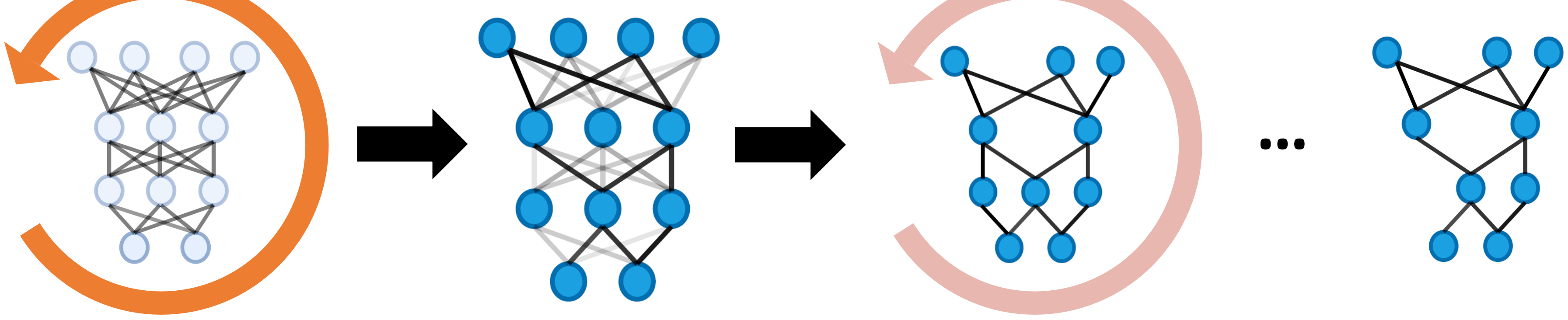
Datasets	Models	Accuracy (vs. Original One)	Energy Savings
CIFAR-10	MobileNetV2	92.06% (vs. 92.47%)	88%
	ResNet-110	93.01% (vs. 93.57%)	83%
CIFAR-100	MobileNetV2	71.61% (vs. 71.91%)	88%
	ResNet-110	71.63% (vs. 71.60%)	84%

Energy savings is quantified based on **FPGA implementation**

EB-Train: Training via Early-Bird Lottery Ticket [ICLR'2020]

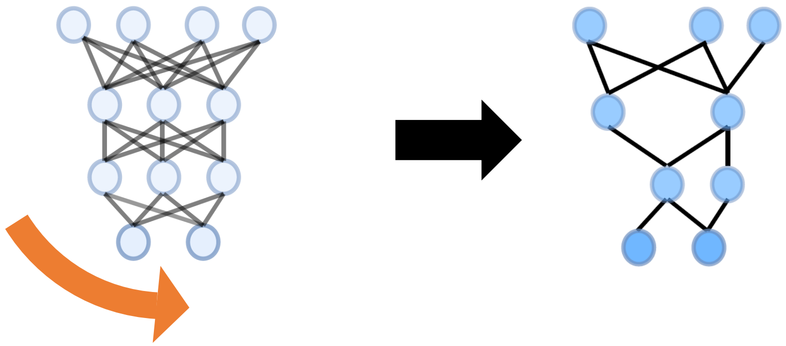
- **Progressive Pruning and Training** (e.g., [J. Frankle, ICLR 2019])

Trained Model



100% training

- **Early-Bird Train (Proposed)**



10% - 20% training

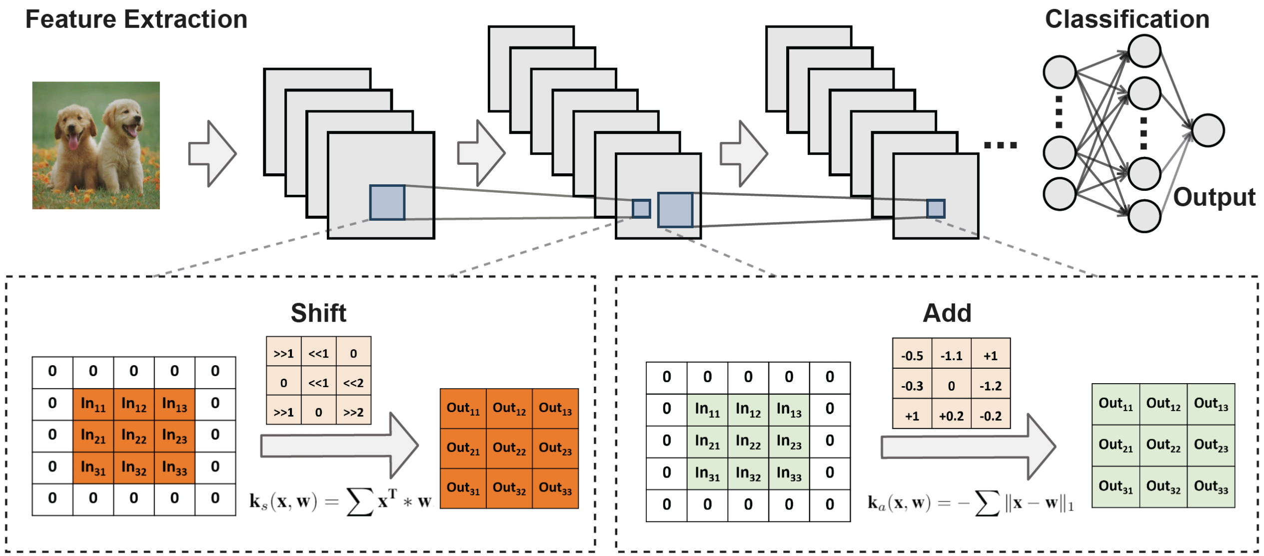
For the first time:

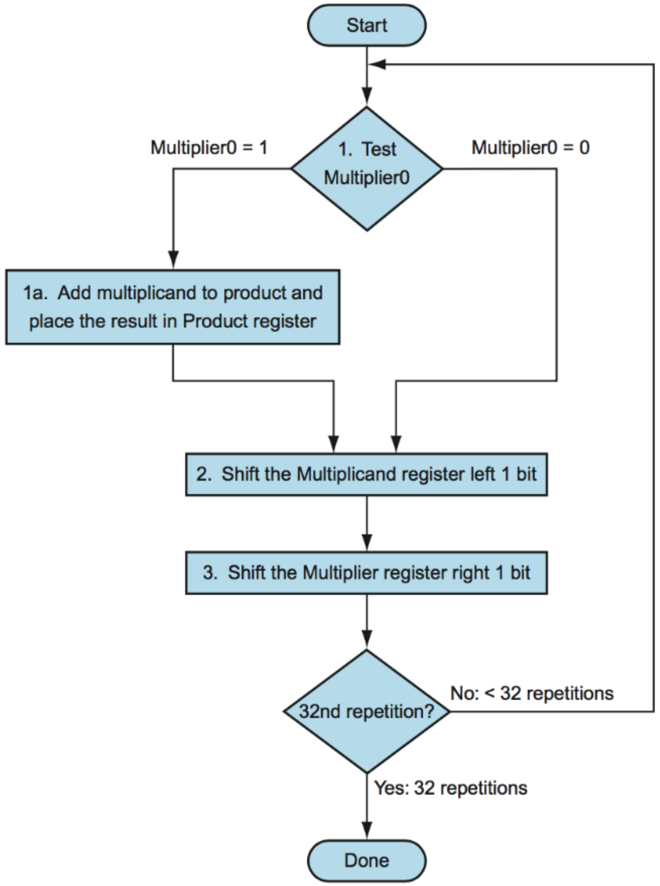
1. We **discover the existence** of Early-Bird (EB) Tickets
2. We **propose a detector** of low cost to detect EB Tickets
3. We leverage the existence of EB Tickets to **develop an efficient training scheme**

➢ **5.8× - 10.7× reduced training energy** with a comparable or even better accuracy over the most competitive baseline

ShiftAddNet: A Hardware-Inspired Deep Network [NeurIPS'2020]

- Multiplication dominates the computation workloads of deep networks
  - How multiplication is efficiently implemented in hardware accelerators?
  - Any multiplication = a left/right **bit shift**, and an **addition** of the residual



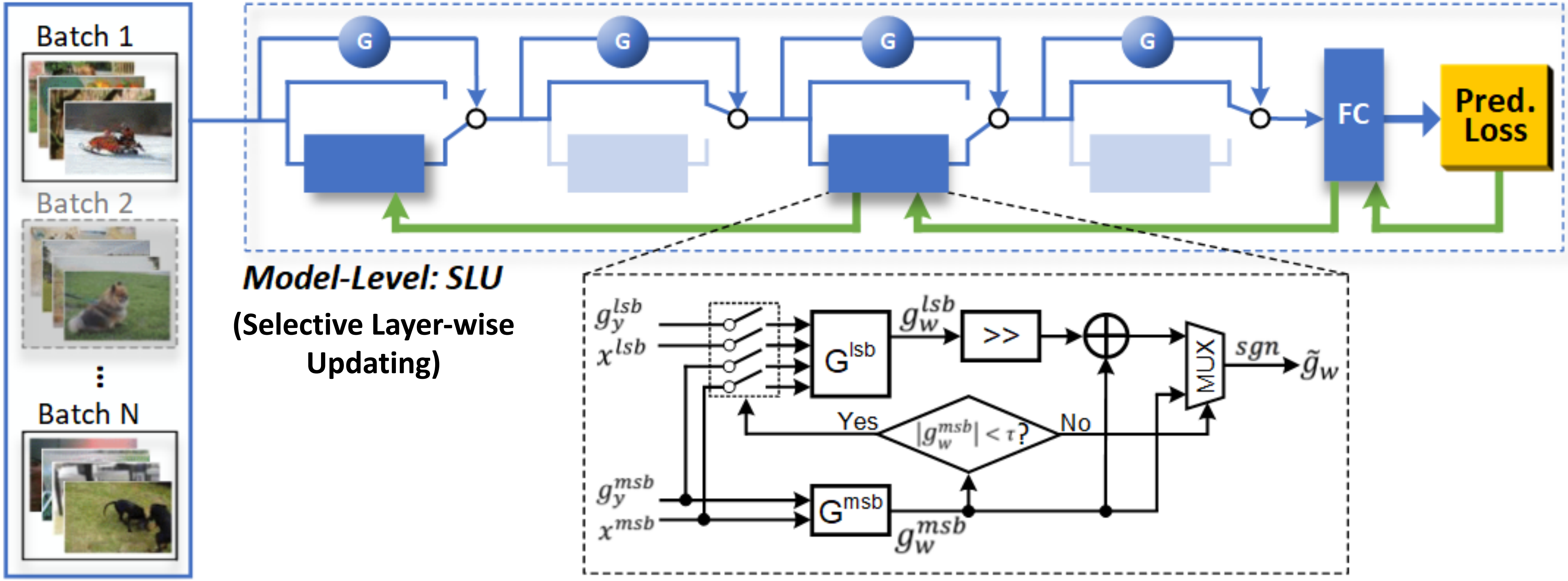


Insight: We explicitly re-build a new multiplication-free deep network, where each multiplication layer is re-parameterized into two learnable layers: element-wise bit-shift layer, and additive layer

Performance: ~ same accuracy + up to **↓80%** energy cost. on CIFAR-10/100 and several IoT datasets (inference + training)

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

E2-Train: Energy-Efficient Training Framework (NeurIPS’19)



Data-Level: SMD

(Stochastic mini-batch dropping)

Algorithm-Level: PSG

(Predictive Sign Gradients)

Data-Level: Stochastic mini-batch dropping (SMD)

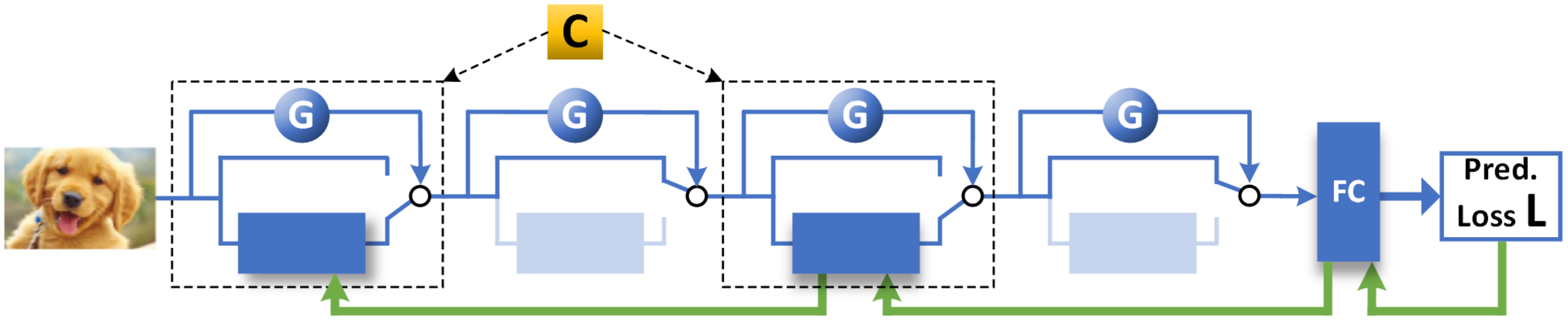
“Frustratingly easy” strategy: randomly skipping mini-batches with 0.5 prob. throughout training

- It sounds ridiculous, but it works!!!
  - We fine-tuned the learning rates, decay, etc., for the original training protocol, but were unable to outperform SMD
- We aim to present some proof from non-asymptotical SGD… **stay tuned!**

Example: ResNet-110 on CIFAR-100

	Energy Savings	Accuracy (top-1)	Accuracy (top-5)
Original	N/A	71.60%	91.50%
SMD	48.43%	70.40%	92.58%

Model-Level: Input-dependent selective layer update (SLU)



- For each minibatch, we select a different subset of CNN layers to be updated, in an input-adaptive way
- **Implementation:** extend the idea of dynamic inference to training , both feed-forward and backward
- Routing by a series of RNN gates: they cost less than **0.04%** FLOPs than the typical base models
- As a side effect, SLU will naturally yield CNNs with dynamic inference capability
- The practice of SLU seems to align with several recent theories on CNN training
    - *"not all layers are created equal"*, and *"lottery ticket"*, etc.

Algorithm-Level: Predictive sign gradient descent (PSG)

- Low-precision implementation is a very effective knob for achieving energy efficient CNNs
- Training with extremely low-precision (binary) gradients, e.g., SignSGD, is shown to be feasible
  - However, they require the computation of full-precision gradients before taking signs -> not energy saving!

- We predict the sign of gradients using low-cost bit-level predictors, therefore completely bypassing the costly full-gradient computation.

$$\tilde{g}_w[i] = \begin{cases} sgn(g_w^{\mathrm{msb}}[i]) & , |g_w^{\mathrm{msb}}[i]| \geq \tau \\ sgn(g_w[i]) & , \mathrm{otherwise} \end{cases}$$

Table 1: Comparing the inference accuracy and energy savings over the baseline of SGD (32-bit floating point) when training with 8-bit fixed point [2], and PSG.

Method	32-bit SGD	8-bit [2]	PSG
Accuracy	93.39%	93.24%	92.95%
Energy savings	NA	38.62%	63.28%

- The prediction failure probability of PSG is upbounded by a term that degrades exponentially with the precision assigned to the predictors

Results: Accuracy versus Energy Trade-off

Training ResNet-74 on CIFAR-10 (baseline acc: top-1 93.57%**)**

FLOPs saving	Energy saving	Accuracy (top-1)
80.27%	83.40%	93.01%
85.20%	87.42%	91.74%
90.13%	91.34%	91.68%

Training ResNet-74 on CIFAR-100 (baseline acc: top-1 71.60%**; top-5** 91.50%**)**

FLOPs saving	Energy saving	Accuracy (top-1)	Accuracy (top-5)
80.27%	81.27%	71.63%	91.72%
85.20%	88.72%	68.61%	89.84%
90.13%	92.90%	67.94%	89.06%

Observation: the proposed training does not slow down the empirical convergence. In fact, it even makes the training loss decrease faster in the early stage.