Compression Methods for Efficient DNN Inference

Ronald Seoh and Hoang Ho
COMPSCI 692S Presentation
14 Oct 2020

Lower the compute costs with smaller models!

Train & Compress with a *large* model

Learning rate rewinding

A *big-little* dual module inference

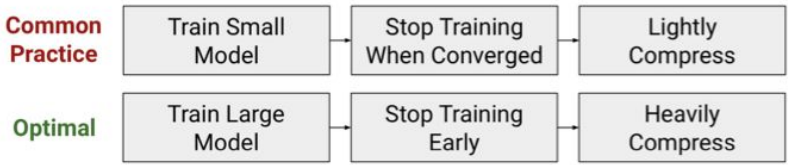
Train Large, Then Compress:

Rethinking Model Size for Efficient Training and Inference of Transformers

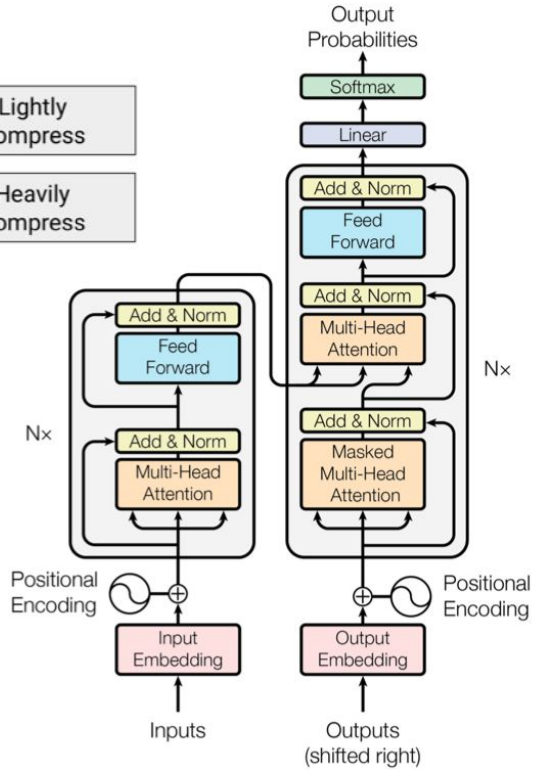
By Zhouhan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, Joseph E. Gonzalez

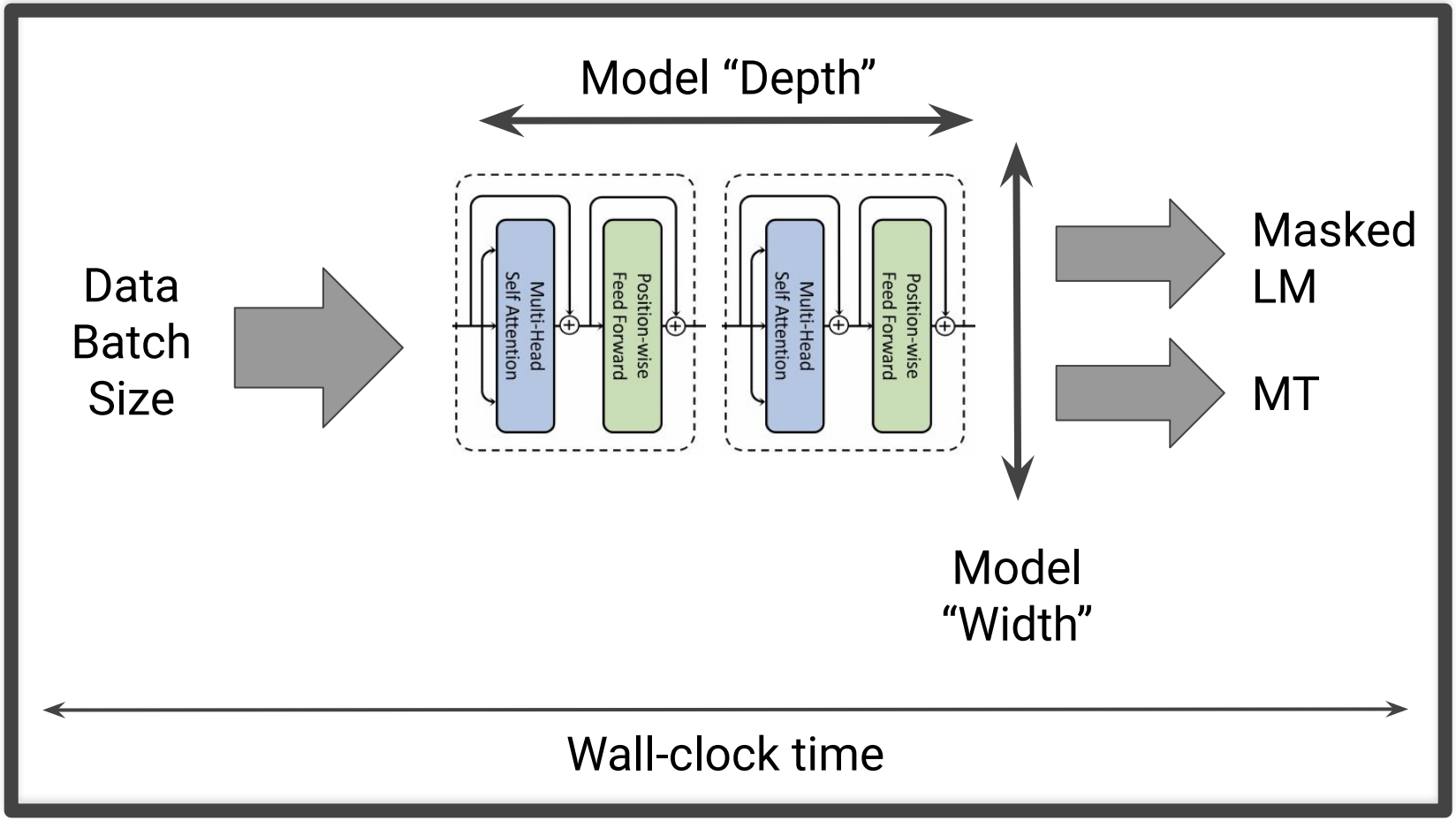
Training Big NLP Models Efficiently

- Training with large text collection is expensive!
- Previous efforts to design smaller architectures
- Do we really need to train the model until convergence?
- **The paper's argument:**
  - **Make models larger**
  - **Train smaller # of steps**
  - **Compress afterwards!!**



Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155



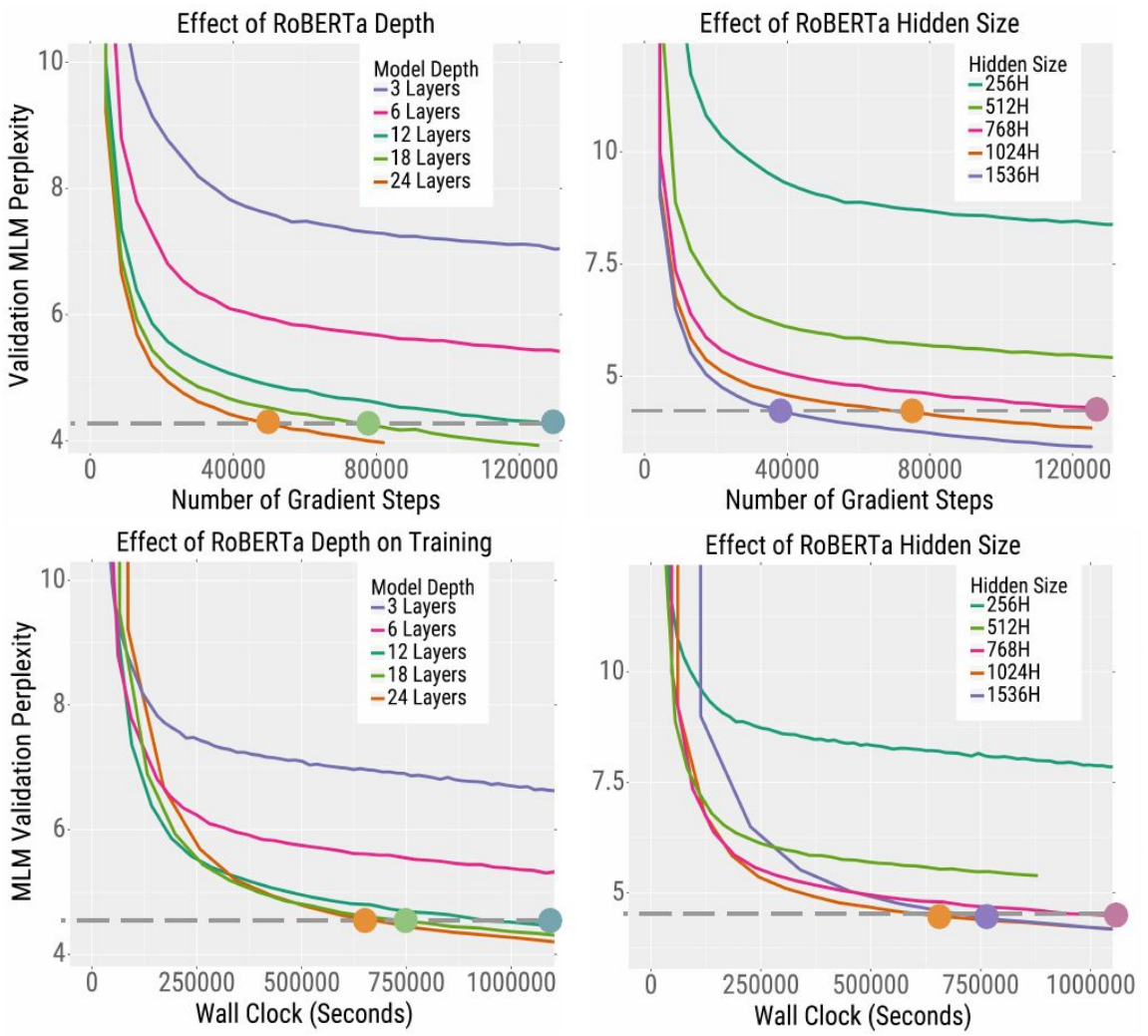

Model “Depth”
Data
Batch
Size
Multi-Head
Self Attention
Position-wise
Feed Forward
Multi-Head
Self Attention
Position-wise
Feed Forward
Masked
LM
MT
Model
“Width”
Wall-clock time
x N
iterations

Setup Details

- Masked Language Model: RoBERTa
  - BookCorpus + Wikipedia Dump: 3.4 billion words (Similar amount to BERT)
  - **Depths**: 3, 6, 12, 18, 24
  - **Hidden Sizes**: 256, 512, 768, 1024, 1536
  - End task fine-tuning: MNLI, SST-2
    - (Chosen because variance in accuracy were lower than others)
- Machine Translation: Standard Transformer
  - WMT14 English-French Dataset (36M Sentences for training)
  - **Depths**: 2, 6, 8
  - **Hidden Sizes**: 128, 256, 512, 768, 1024, 2048
- Primary evaluation metric: wall-clock time
  - Counting gradient steps cannot account for bigger batches/models
  - Some GPUs could process more FP operations in parallel

Larger Models Train Faster

- Increase model width and depth!
- Shape doesn't matter - the total # of parameters is the key determiner of the convergence rate
- Returns diminish as size increases
- **Note**: Smaller models have used larger batch sizes for fairer comparison.



Large LM does ***not*** make fine-tuning difficult

Model	Perplexity	MNLI	SST-2
12-layer, 768H	4.3	84.3	93.0
18-layer, 768H	4.1	85.4	92.6
24-layer, 768H	4.0	85.2	93.1
12-layer, 768H	4.3	84.3	93.0
12-layer, 1024H	3.9	85.5	93.2
12-layer, 1536H	4.3	85.1	93.8

Table 2. We train RoBERTa models of different sizes and stop them at roughly the same pretraining perplexity (the bigger models are trained for less wall-clock time). We then finetune each model on MNLI and SST-2. All models reach comparable accuracies (in fact, the big models often outperform small ones), which shows that larger models are not harder to finetune.

Using larger batch sizes doesn't actually help

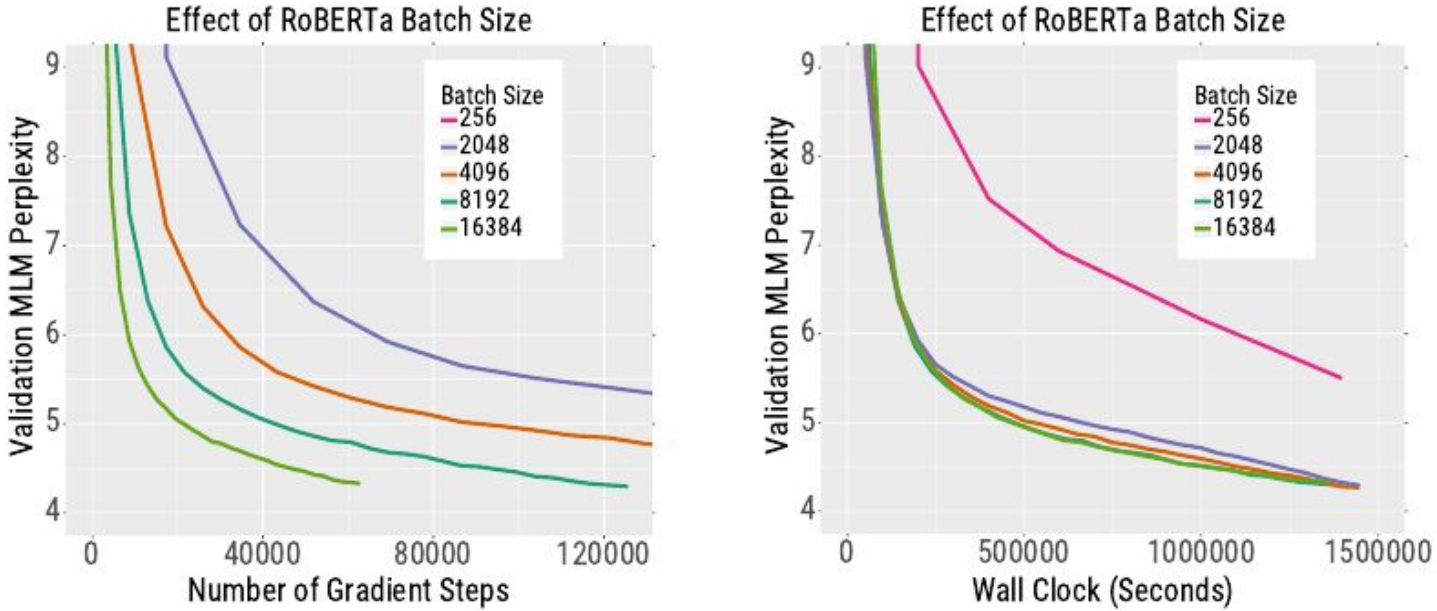
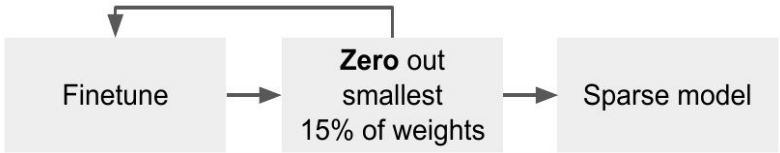


Figure 13. Increasing the batch size and the associated learning rate accelerates convergence in terms of gradient steps. However, increasing the batch size beyond 2048 provides only marginal improvements with respect to wall-clock time. Note that the wall-clock time includes the cost of accumulating gradients on a single machine (see Section 2.2). In other words, beyond a certain point increasing the batch size only provides speedups when additional hardware is available. The 256 batch size result is far to the right in the left plot.

Pruning and Quantization Experiments

- Train models of different sizes for 1,000,000 seconds
- Fine-tune them on MNLI, SST-2
- Apply quantization / pruning
- Report the memory needed to store the model parameters
- Quantization: 4, 6, 8, 32 bits
- Iterative magnitude pruning: Zero out the smallest magnitude parameters
  - Sparsity levels: 15%, 30%, 45%, 60%, 75%, 90%
  - For each epoch, zero out 15%; fine-tune the model on the downstream task until it reaches within 99.5% of its validation accuracy *or* until we reach 1 training epoch



Larger Models Compress Better

- More robust to quantization, pruning, or combination of both

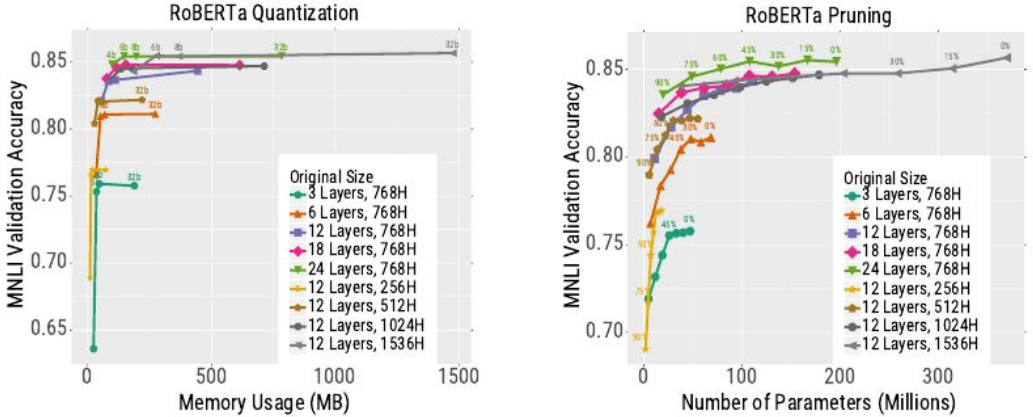


Figure 6. We first pretrain RoBERTa models of different sizes for the *same* total wall-clock time (larger models are trained for fewer steps). We then finetune each model on MNLI and compress them using quantization (left) and pruning (right). For most budgets (x-axis), the highest accuracy models are the ones which are trained large and then heavily compressed. The labels above each point indicate the compression amount (e.g., 4-bit quantization or 45% sparsity); we omit cluttered labels. SST-2 results are shown in Appendix D.

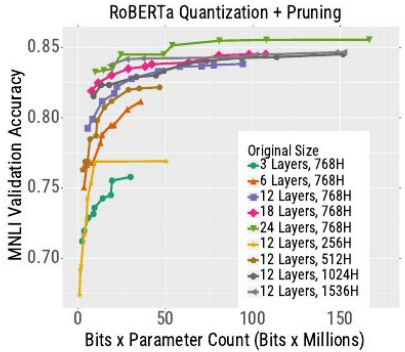


Figure 7. We combine pruning and quantization and find their gains to be complementary. The models which are trained large and then compressed are the best performing for each test-time budget.

It doesn’t matter if the model didn’t converge yet

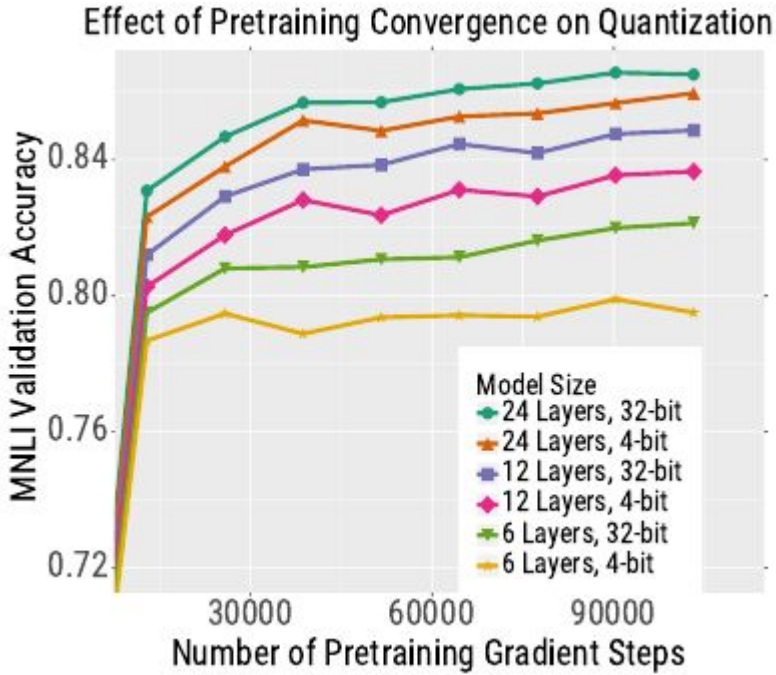


Figure 8. We disentangle whether *model size* or *pretraining convergence* causes the enhanced compressibility of larger models. We finetune RoBERTa models starting from different pretraining checkpoints on MNLI. We then quantize the models to 4-bits. Quantization is hardly affected by convergence—the drop in MNLI accuracy due to quantization is comparable as the pretrained model becomes more converged. Instead, the factor that determines compressibility is model size—the drop in accuracy is very large when compressing smaller models and vice versa.

Why?!?!

- Better sample efficiency
  - Not much work on exploring if and why large models converge faster.
  - Arora et al. (2018): For deep linear NNs, increasing depth can promote movement along directions already taken by the optimizer
  - **Fast minimization:** Empirically, larger Transformer models minimize training error faster
    - Plus, the generalization gap is small for our tasks due to very large training sets
    - Models don't really ever overfit
- Better use of computing power
  - GPU/TPUs can compute a ton of floating points in parallel
  - So let's do big computations few times
    - instead of doing small computations several times
- Smaller compression error
  - The Lottery Ticket Hypothesis (Frankle & Carbin, 2019)

Large models work only if we have large datasets

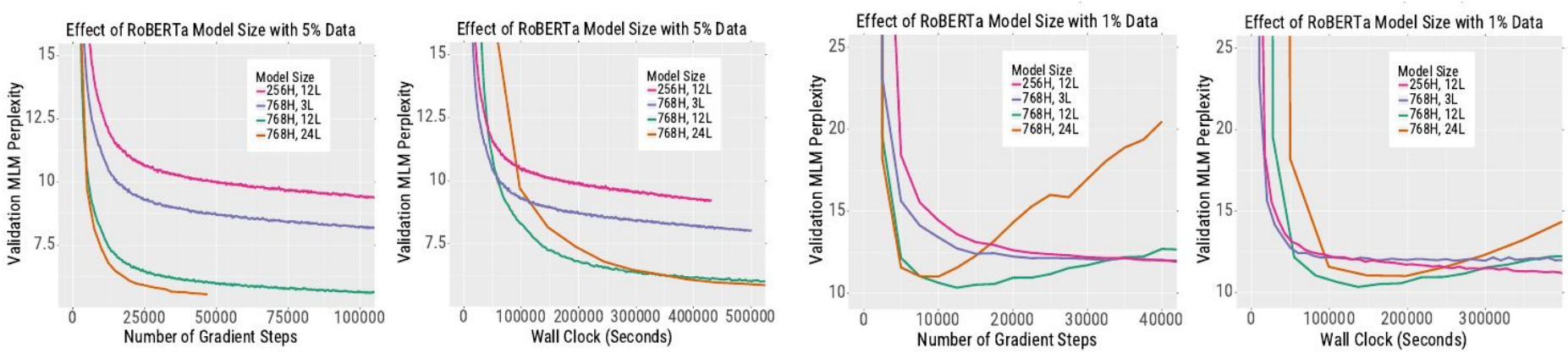


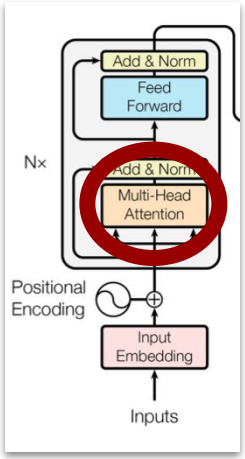
Figure 14. Effect of Smaller Datasets. In our experiments on the full dataset (see main text), the largest models we trained are always faster in terms of wall-clock time. However, when subsampling the data to 5% (top row), the biggest models do not improve on the speed of the smaller models (e.g., compare 24 Layer RoBERTa and 12 Layer RoBERTa). When the data is subsampled to 1% (bottom row), the bigger models are *worse* in terms of perplexity due to overfitting. This illustrates that the optimal model size depends on the dataset size.

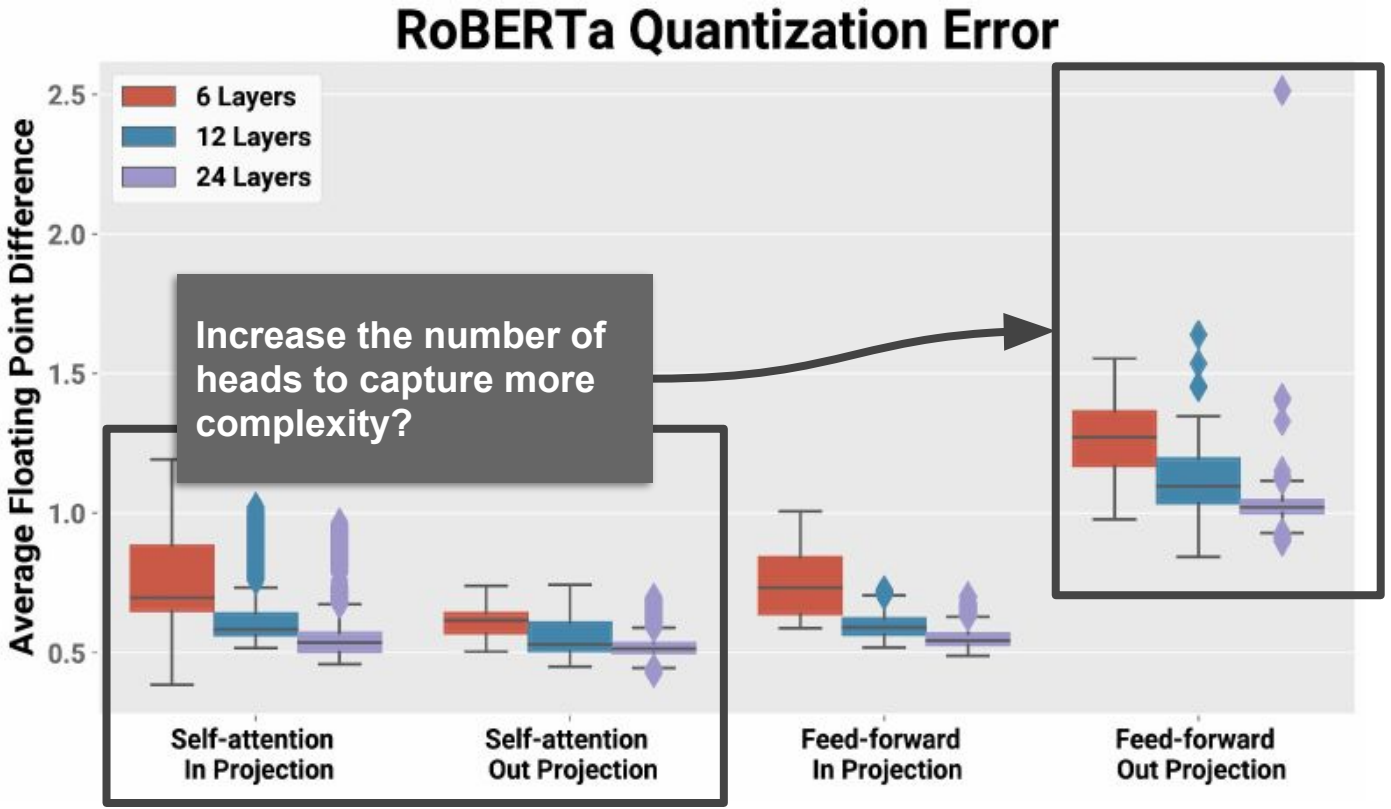
Pros & Cons

- More efficient use of computing **power and time** in large-scale training settings (i.e. big tech companies)
- At the same time, fast and more robust inference performance through quantization / pruning
- Plays well with fine-tuning
- All these benefits with “small” changes!

- If physical memory is **not big and fast enough** to hold the model + data, all these findings are mostly inapplicable.
- Probably not a good idea to increase model size if you have **limited training data.**
- Why not try adding more heads to make the Transformers models “bigger”?

Adding more heads in the Transformers?





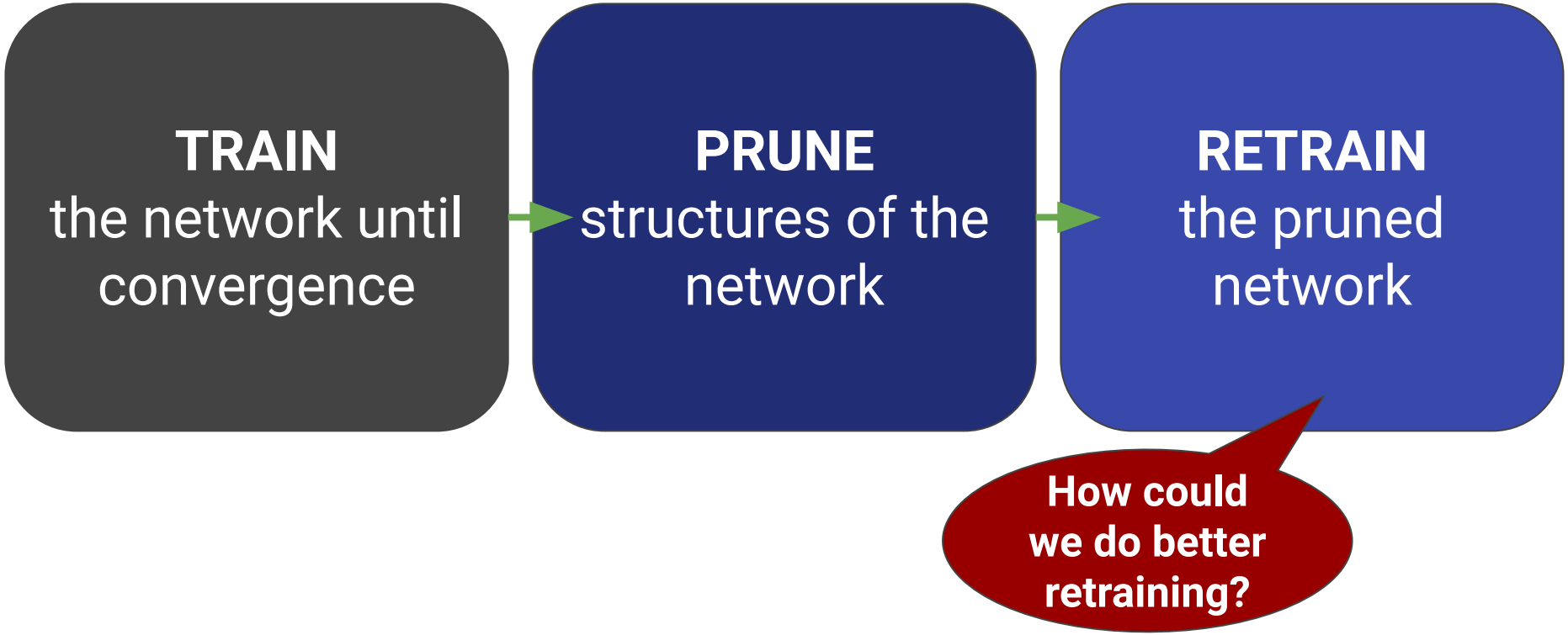
Questions?

Comparing Rewinding and Fine-Tuning

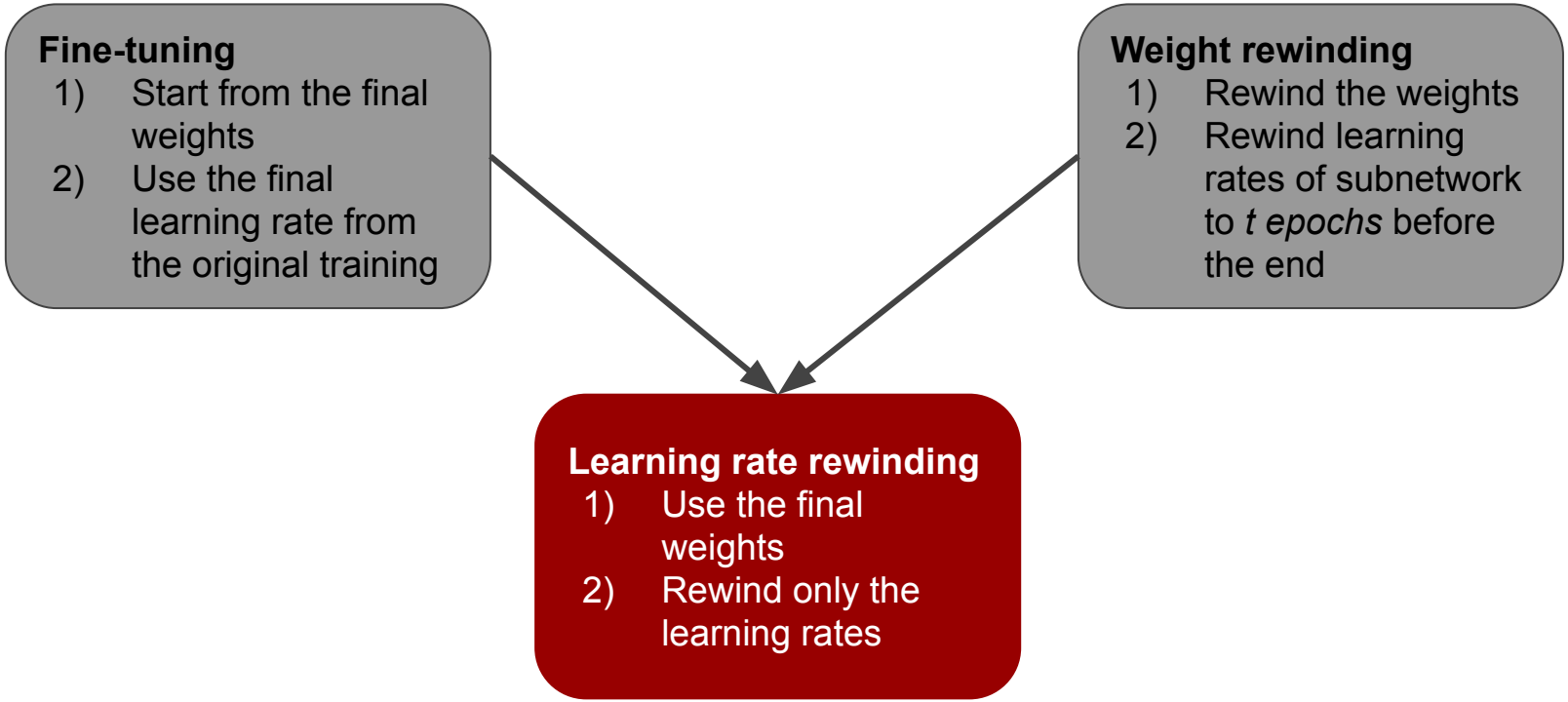
In Neural Network Pruning

By Alex Renda, Jonathan Frankle, Michael Carbin

Retraining-based pruning algorithm

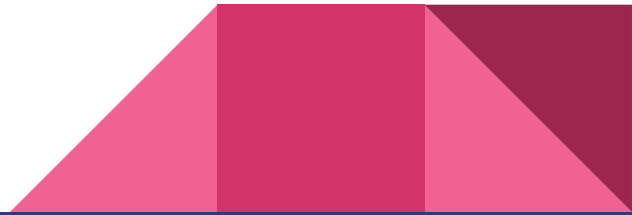


Learning rate rewinding



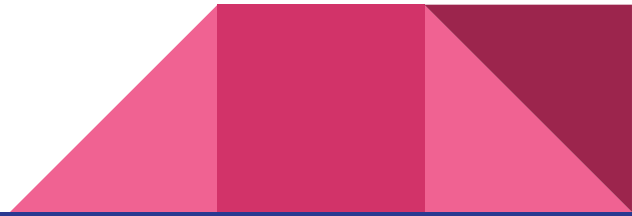
Experiments

1. **TRAIN**: A variety of standard architectures for image classification / machine translation
    a. ResNet (CIFAR-10, ImageNet)
    b. GNMT (WMT16 English-German)
2. **PRUNE**: Unstructured and structured pruning methods
    a. One-Shot Pruning: Prune the network to a target sparsity level all at once
    b. Iterative Pruning: Repeat PRUNE-RETRAIN until we reach target sparsity
        - Prune 20% of weights per iteration
3. **RETRAIN**
    a. Fine-tuning
    b. Weight rewinding
    c. Learning rate rewinding



Evaluation Criteria

- **Accuracy**: Does the pruned net match the accuracy of the unpruned net?
- **Parameter Efficiency**: Parameter count of the network
  - Compression ratio
- **Search Cost**: Computational resources required to find good pruning
  - Retraining time: the total number of additional retraining epochs.



Evaluation: Accuracy vs Parameter Efficiency Tradeoff

One-shot pruning: At higher compression rate, fine-tuning performs worse than weight rewinding. Learning rate rewinding performs better than weight rewinding by a small margin

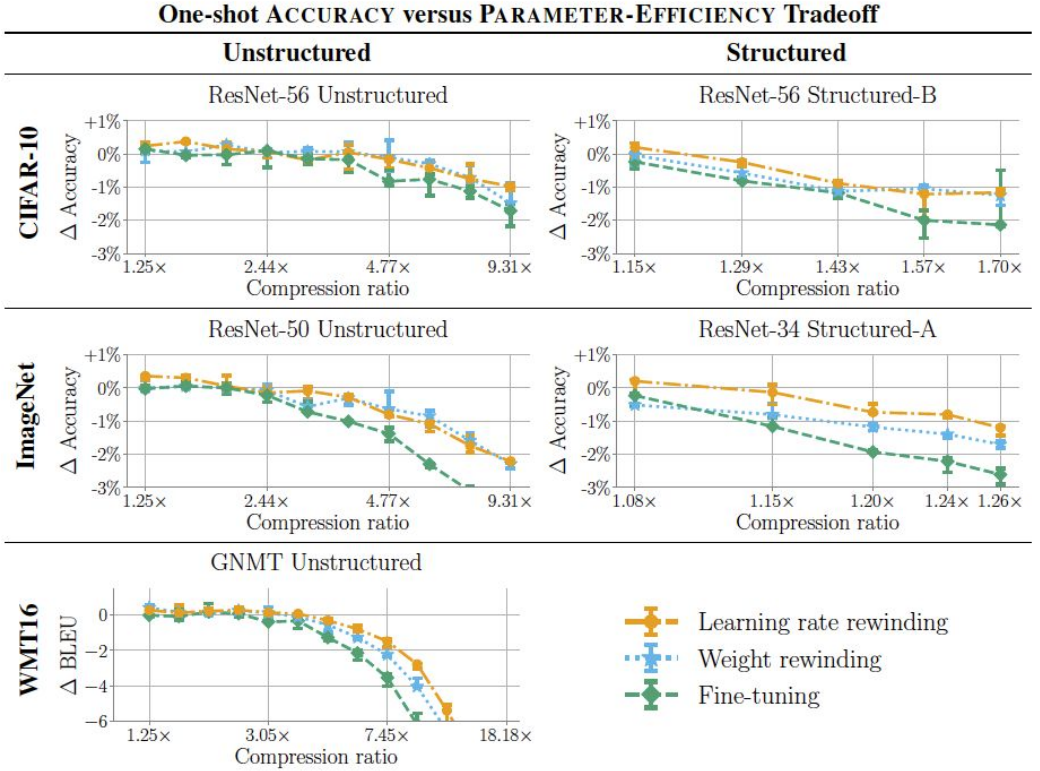


Figure 1: The best achievable accuracy across retraining times by one-shot pruning.

Evaluation: Accuracy vs Parameter Efficiency Tradeoff

Iterative pruning:

- Weight rewinding outperforms fine-tuning
- Learning rate rewinding outperforms weight rewinding.
- Iterative unstructured pruning produces ResNet 50 with 5.96x compression rate with no accuracy drop -> state of the art in compressed network

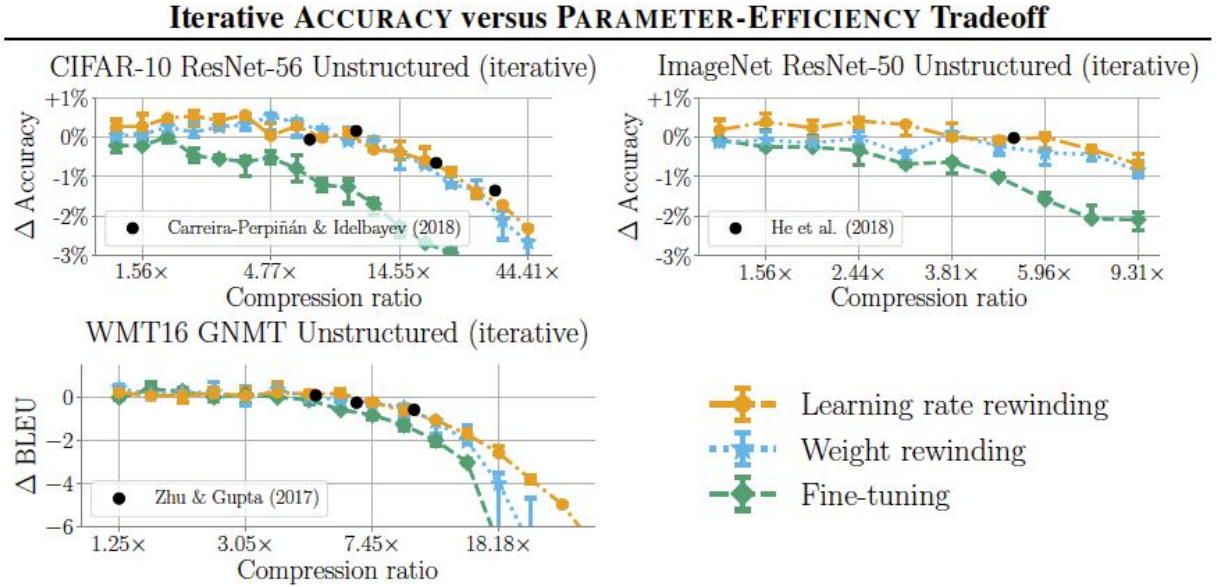


Figure 2: The best achievable accuracy across retraining times by iteratively pruning.

Evaluation: Accuracy vs Search Cost Tradeoff

Unstructured pruning:

- Both rewinding techniques match or outperform fine-tuning for equivalent retraining epochs
- See accuracy drop in rewinding weights and retraining for full original training time

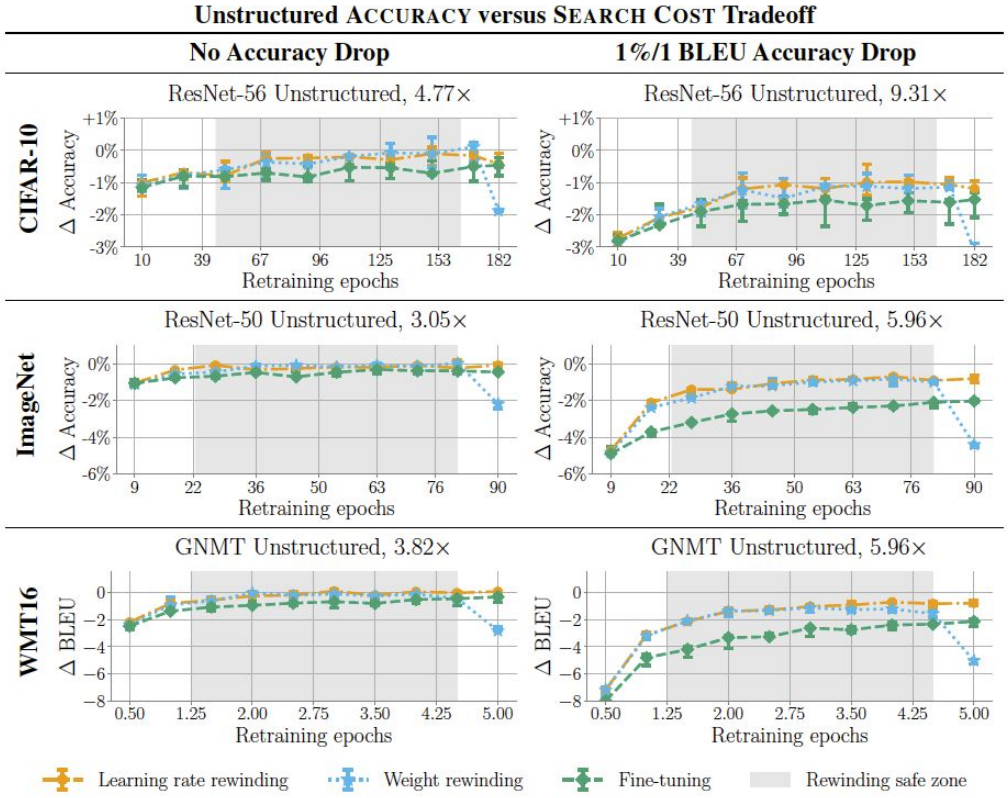


Figure 3: Accuracy curves across networks and compression ratios using unstructured pruning.

Evaluation: Accuracy vs Parameter Efficiency Tradeoff

Structured pruning:

- Similar trend as unstructured
- Except: retraining weight rewinding for full training time doesn't have accuracy drop
- Initialization is less consequential for retraining after structured pruning than after unstructured pruning

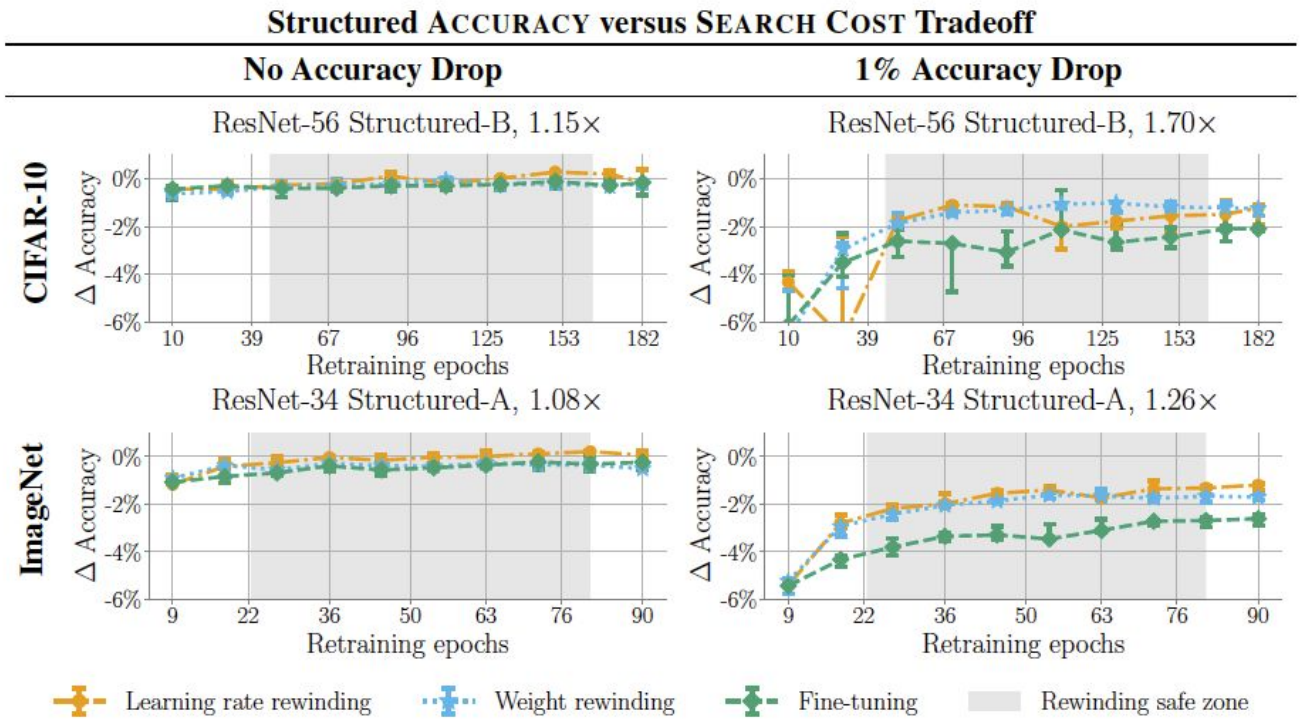


Figure 4: Accuracy curves across networks and compression ratios using structured pruning.

Pruning Algorithm

Algorithm 1 Our pruning algorithm

1. TRAIN to completion.
2. PRUNE the 20% lowest-magnitude weights globally.
3. RETRAIN using learning rate rewinding for the original training time.
4. Repeat steps 2 and 3 iteratively until the desired compression ratio is reached.

- Pruning algorithm requires T(1+k) total training epochs to reach $1/0.8^k$ where T is the original training time and k is the number of pruning iterations.
- Retraining techniques reuse hyperparameters from the original training process

Evaluation: Pruning Algorithm

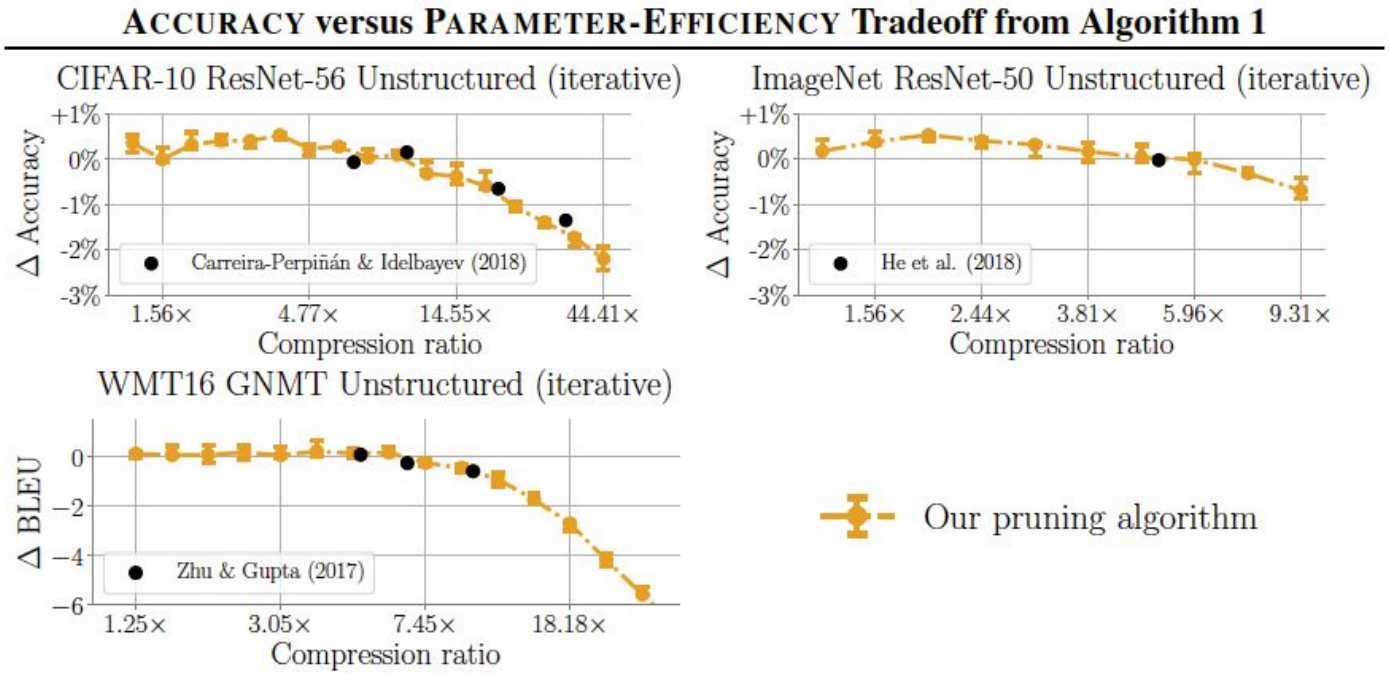


Figure 5: ACCURACY versus PARAMETER-EFFICIENCY tradeoff of our pruning algorithm.

Questions?

Pros & Cons

- Proposed retraining technique, learning weight rewinding, that outperforms other retraining techniques
- Proposed a pruning algorithm that achieve state-of-the-art ACCURACY vs PARAMETER EFFICIENCY

- Iteratively prune 20% lowest magnitude weights globally may not be optimal because different layers may have different distributions
- Whether unstructured pruning or structured pruning have better performance improvement?
- Structured pruning has less than 2.0x compression ratio, wonder if this can be one of the reasons why weight initialization doesn't have much effect in weight rewinding

Boosting Deep Neural Network Efficiency with Dual-Module Inference

By Liu Liu, Lei Deng, Zhaodong Chen, Yuke Wang, Shuangchen Li, Jingwei Zhang, Yihua Yang, Zhenyu Gu, Yufei Ding, Yuan Xie

Motivation: Noise Resilient

Insensitive region: the domain of f that is resilient to the noise δ. An activation function f is resilient to the noise δ when |f(x+δ) - f(x)| < θ, where θ is a threshold

For $tanh$, when $|x| \gg 0$, we have

$$|tanh(x+\delta) - tanh(x)| \approx 2e^{-2|x|}|\delta|.$$

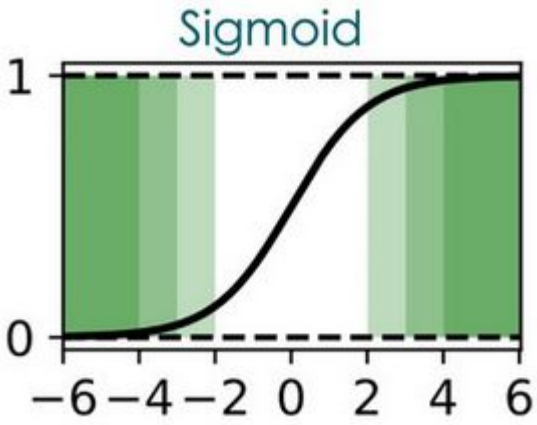
Similarly, for $sigmoid$, when $|x| \gg 0$ we have

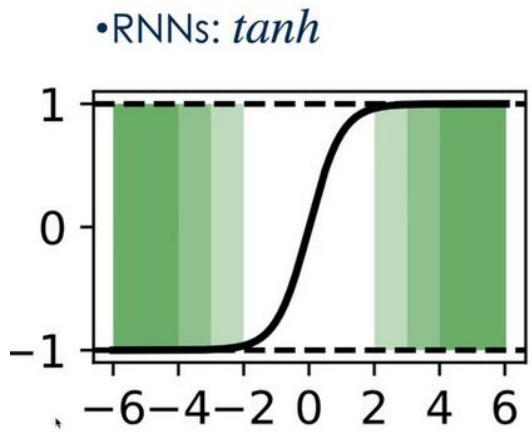
$$|sigmoid(x+\delta) - sigmoid(x)| \approx e^{-|x|}|\delta|.$$

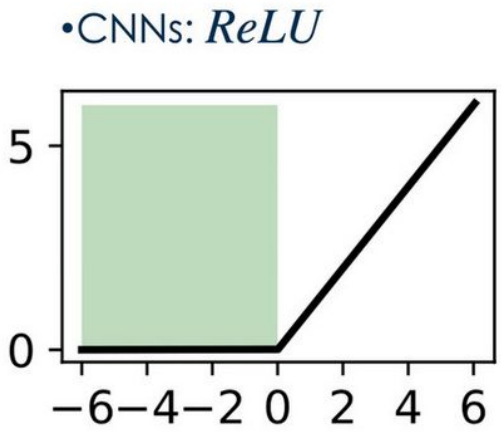
$$|ReLU(x+\delta) - ReLU(x)| \begin{cases} = 0 & x \leq -|\delta| \\ \leq |\delta| & otherwise \end{cases}$$

$$\begin{cases} tanh: & |x| > \frac{1}{2}\ln\frac{2|\delta|}{\theta} \\ sigmoid: & |x| > \ln\frac{|\delta|}{\theta} \\ ReLU: & x < \theta - |\delta| \end{cases}$$

Motivation: Noise Resilient







Aggressive pruning can be applied to insensitive region!

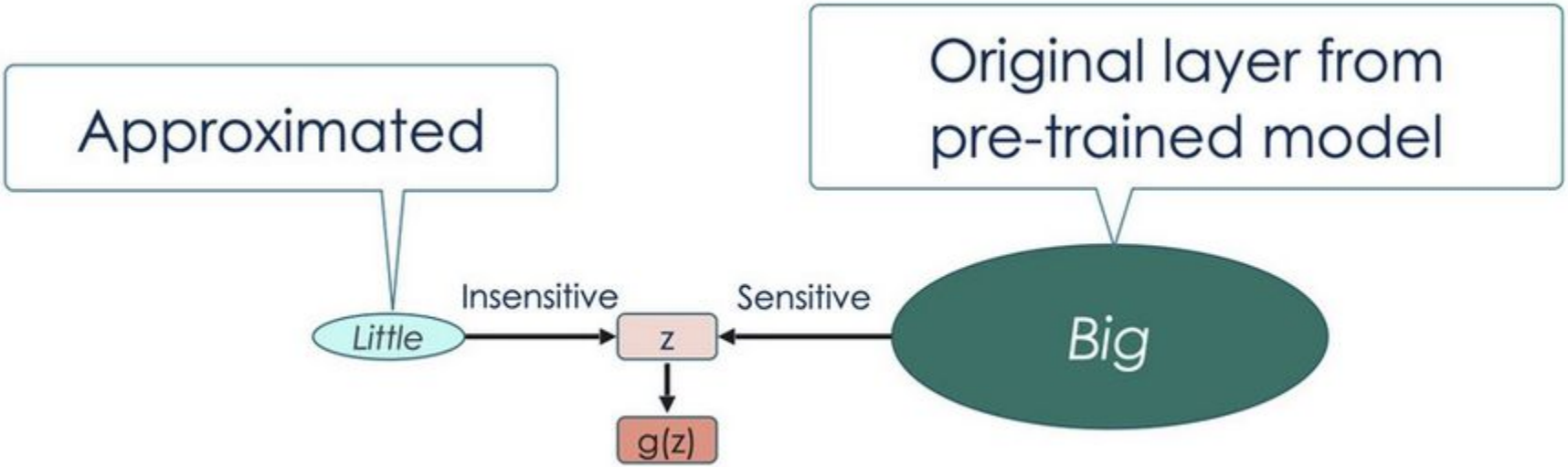
Motivation: Experiments on (In)sensitive regions

Table 1. Comparison of adding Gaussian noises to the sensitive or insensitive region of LSTM gates.

Case	Cosine similarity before gates				Cosine similarity after gates				PPL
	input	forget	cell	output	input	forget	cell	output	
Sensitive	0.953	0.859	0.952	0.932	0.934	0.946	0.882	0.940	85.70
Insensitive	0.944	0.929	0.943	0.947	0.968	0.987	0.969	0.977	81.79

- One LSTM layer with base perplexity of 80.64
  - Adding Gaussian Noise to sensitive and insensitive regions
  - Measure cosine similarity between activations in the original model and noise-introduced model
- Cosine similarity in insensitive region is close to 1

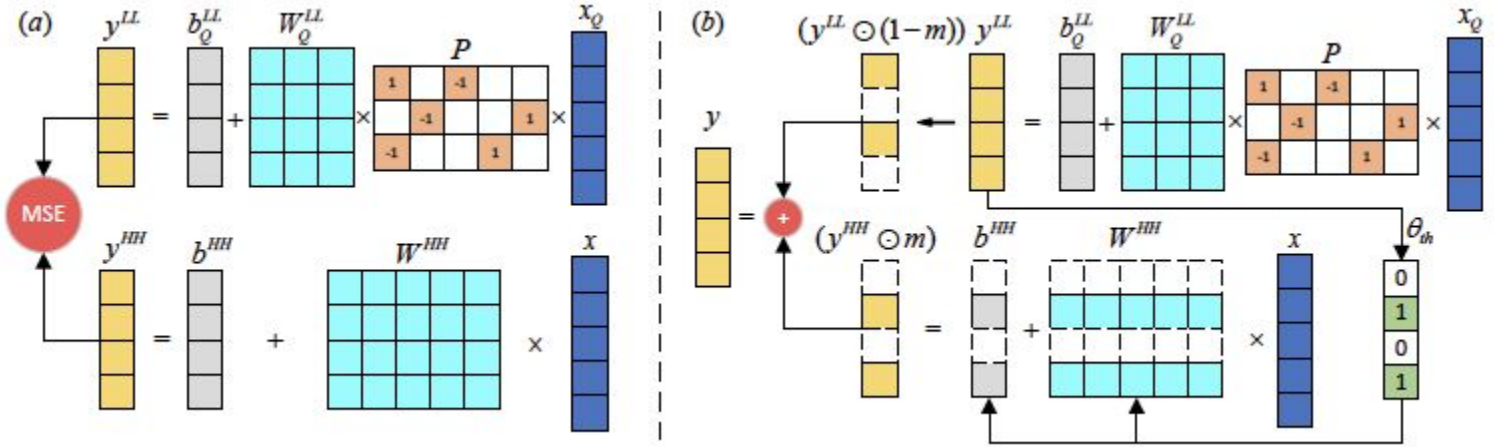
Proposal: DMI



DMI: Design

1. Quantize the input activations and forward through the little module
2. Predict which output neurons belong to the sensitive region based on results from the little module
3. Compute the the output activations of the big module in the predicted sensitive region
4. Combine outputs of the little module in insensitive region with outputs of big module in sensitive region

DMI: Fine-tuning & Inference



Little Module: Dimensionality Reduction

Big Module: $z = \varphi(y^{HH})$; $y^{HH} = W^{HH} x+b$. $W^{HH} \in R^{n \times d}$, input feature $x \in R^{n}$, pre-activated output $y^{HH} \in R^{n}$.

Little Module: $y^{LL} = W^{LL} Px+b$, $W^{LL} \in R^{n \times k}$ $(k < d)$, sparse matrix $P \in R^{k \times d}$.

Approximate $y^{HH} = W^{HH} x+b$ with $y^{LL} = W^{LL} Px+b$

Little Module: Dimensionality Reduction

Hypothesis: Choose k as the smallest number that preserve Euclidean distance in W^{HH}. Using theorem 1.1 in Achlioptas (2003)

$$P_{ij} = \sqrt{\frac{3}{k}} \times \begin{cases} +1 & with\ probability\ 1/6 \\ 0 & with\ probability\ 2/3 \\ -1 & with\ probability\ 1/6 \end{cases}$$

$$k = \frac{4}{\epsilon^2/2 - \epsilon^3/3} log(n).$$

Complexity for $y^{HH} = W^{HH}$ x+b: O(n x d) MACs
Complexity for $y^{LL} = W^{LL}$ Px+b: O(⅓ * k * d + n * k) MACs

ε: *hyperparameter obtained experimentally on validation set*

Training Little Module: Knowledge Distillation

- Train little module via Knowledge distillation: *Big Module* as teacher, *Little Module* as student
- Transfer knowledge as the flow of solution procedure (FSP) matrix between two layer.
- Loss function is defined as:

$$L = ||G_t - G_s||_F^2.$$

In our dual module, we have

$$G_t = x\left(y^{HH}\right)^T, \quad G_s = x\left(y^{LL}\right)^T,$$
$$||G_t - G_s||_F^2 = Tr\left(xx^T\right)||y^{HH} - y^{LL}||_2^2.$$

Training Little Module: Insensitive Region Prediction

- Whether a pre-activation belongs to insensitive region is based on whether y_i^{LL} is in the insensitive region
- Minimize the KL divergence of two distributions p_{HH} and p_{LL}
- Minimize $D_{KL}(p_{HH}|p_{LL})$ is equivalent to minimizing $||y_{HH} - y_{LL}||^2$

Little Module: Threshold for (in)sensitive region

- **Fixed threshold**: Assigned constant value to θ

$$\begin{cases} sigmoid/tanh: & if\ |y_i^{LL}| > \theta_{th},\ m_i = 0;\ else\ m_i = 1 \\ ReLU: & if\ y_i^{LL} < \theta_{th},\ m_i = 0;\ else\ m_i = 1 \end{cases}$$

- **Adaptive threshold:**

$$\theta_{th} = \begin{cases} \frac{1}{2} \ln \frac{2|\delta|}{\theta} & for\ tanh \\ \ln \frac{|\delta|}{\theta} & for\ sigmoid \\ \theta - |\delta| & for\ ReLU \end{cases}$$

θ: global fixed threshold; $|\delta| = 1/n\ ||y^{HH} - y^{LL}||$

Little Module: Threshold for (in)sensitive region

- **Top-K mask**: taking exactly K neurons, K is a hyperparameter tuned on validation set

$$\theta_{th} = \begin{cases} top_K(|\boldsymbol{y}^{LL}|) & for\ tanh/sigmoid \\ top_K(\boldsymbol{y}^{LL}) & for\ ReLU \end{cases}$$

DMI: RNN Evaluation

- Evaluated on a set of representative RNN on CPU-based server platform

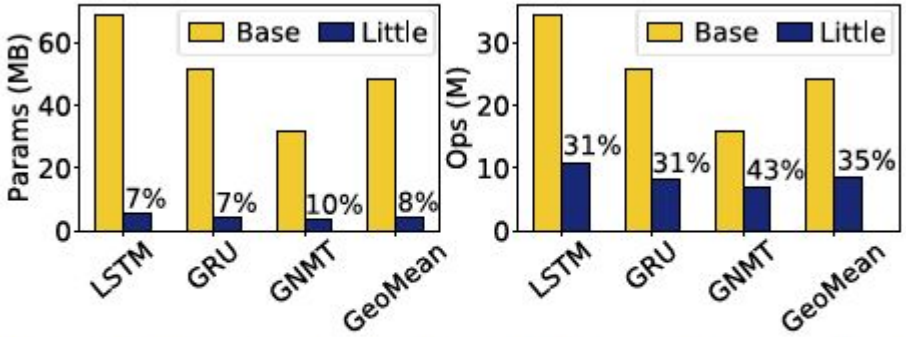


Figure 2. Comparison of the amount of accesses and operations between baseline layers and the *little* module of dual-module enhanced RNN-based models.

DMI: RNN Evaluation

Table 2. RNN quality and execution time (ms). Ln means a LSTM layer with h hidden units; G is short for GRU.

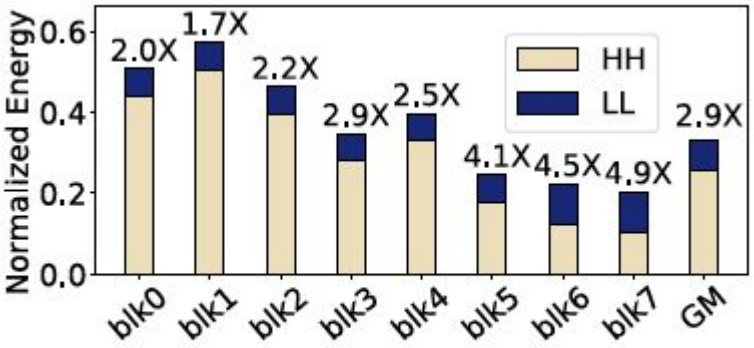
Insensitive	LM, L1500				LM, G1500				GNMT, L1024			
Ratio	PPL	Diff.	Time	Speedup	PPL	Diff.	Time	Speedup	BLEU	Diff.	Time	Speedup
Baseline	80.64	n/a	1.477	1.00x	85.48	n/a	1.182	1.00x	24.32	n/a	0.838	1.00x
10%	80.72	-0.08	1.315	1.12x	85.62	-0.14	1.024	1.15x	24.33	0.01	0.679	1.23x
30%	80.56	0.08	1.095	1.35x	86.01	-0.53	0.869	1.36x	24.18	-0.14	0.541	1.55x
50%	81.36	-0.72	0.885	1.67x	88.73	-3.25	0.726	1.63x	23.73	-0.59	0.480	1.75x
70%	87.48	-6.83	0.641	2.30x	98.09	-12.61	0.545	2.17x	21.92	-2.40	0.360	2.33x
90%	109.37	-28.73	0.380	3.89x	122.75	-37.27	0.350	3.38x	11.77	-12.55	0.243	3.45x

**using wall-clock time as metric*

DMI: CNN Evaluation

Table 3. Comparison of the Top-1 accuracy and FLOPs reduction of our method with prior work on dynamic sparsity. The baseline model is ResNet-18 on ImageNet.

Method	Acc. (%)	Diff. (%)	FLOPs reduction
Dense (*torchvision*)	69.7	n/a	1.00x
LCL (Dong et al., 2017)	66.3	-3.4	1.53x
FBS (Gao et al., 2018)	68.2	-1.5	1.98x
SeerNet (Cao et al., 2019)	69.3	-0.4	1.67x
CGNet (Hua et al., 2019)	68.8	-0.9	1.93x
DMI (Ours)	**69.2**	**-0.5**	**3.02x**



gure 3. Energy efficiency of each residual block in ResNet-18.

DMI: Dimensionality Reduction and Quantization Analysis

Table 4. Sensitivity study of dimension reduction.

Dimension	PPL	Speedup	*little*	*big*
1500 (baseline)	80.64	1.00x	0%	100%
966 ($\epsilon = 0.3$)	80.40	1.37x	22%	44%
417 ($\epsilon = 0.5$)	81.36	1.67x	12%	47%
266 ($\epsilon = 0.7$)	83.51	1.71x	8%	46%

Table 5. Inference quality and parameter size comparison under different levels of quantization on the *little* module

Precision	Base	FP32	INT16	INT8	INT4	INT2
PPL	80.64	81.28	81.18	81.36	81.47	82.43
MSE	n/a	0.408	0.425	0.444	0.451	0.68
Params.	68.7	19.1	9.6	4.8	2.4	1.2

- More aggressive dimensionality reduction can further gain more speed up at the cost of more quality degradation
- Approximation of the little module is compromised by aggressive quantization, which reduce overhead for little module

Questions?

DMI: Pros and Cons

- Combine techniques in dimensionality reduction and knowledge distillation to train Little Module, which can approximate Big Module with less memory access and faster inference time

- Models used in evaluation is small, wonder if using larger model can achieve better insensitive ratio and more robust results?
- Doesn’t offer analysis on how much speed up and reduced memory access is due to the proposed method/quantization?
- Insensitive ratio depends on the input data and the model architecture, but evaluation doesn’t analyze the effect of this? and not sure what training dataset is used?