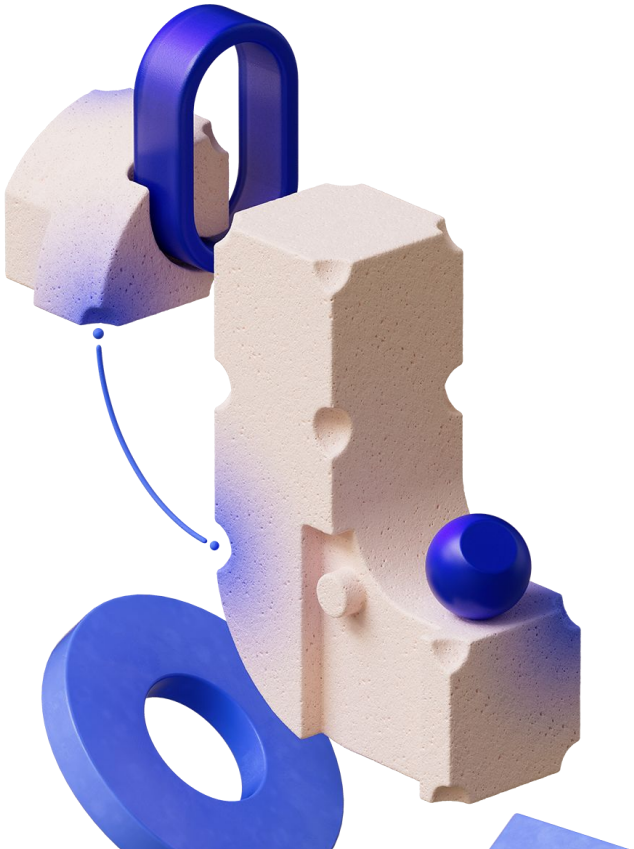
Google DeepMind



Sparsity and Quantization

Utku Evci
April 2025

Why compress?

- **Training**
  - Compute and time is limiting factor.
  - Compression can* accelerate training if accelerated
  - Compression can help reduce communication.
  - Larger batchsize

- **Inference**
  - Server vs On-device
  - Minimize Latency and Increase throughput (while obeying hardware-limits) (i.e. RAM is expensive)
  - Smaller batchsize.
  - Prefill is (more) compute bound.
  - Decoding is memory bound
    - Increase compute intensity (less bytes/params)

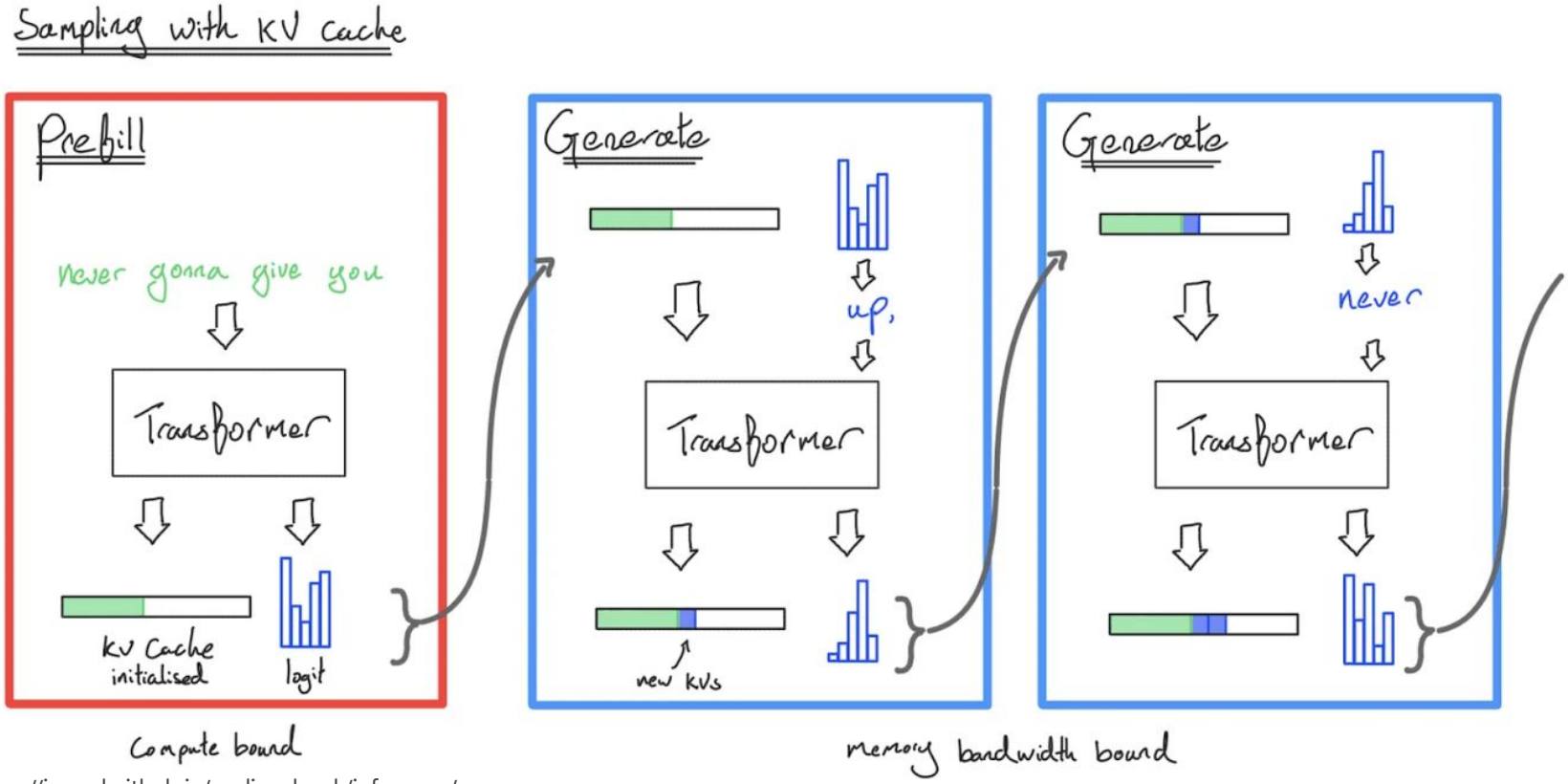
TIME

efficient+parallel implementation

MEMORY

and compute

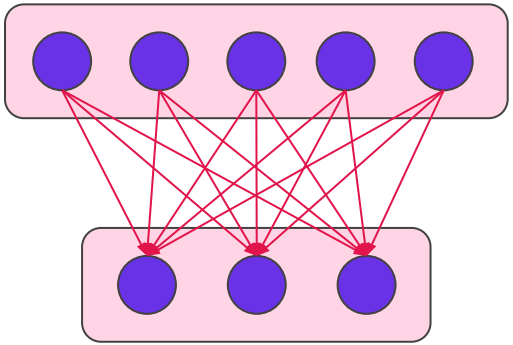
Inference = Prefill + Generate



https://jax-ml.github.io/scaling-book/inference/

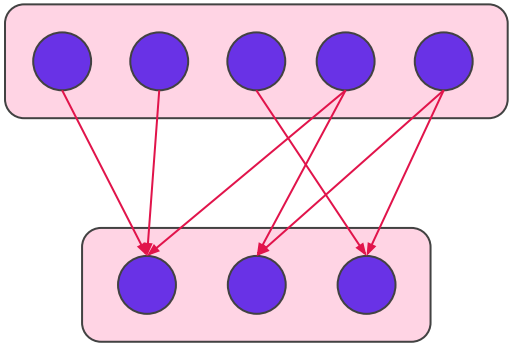
What to compress?

Weights	Activations
Not input dependent	Input dependent
Fixed shape	Not fixed
Constant during inference	Not constant
(offline compression)	(online compression)

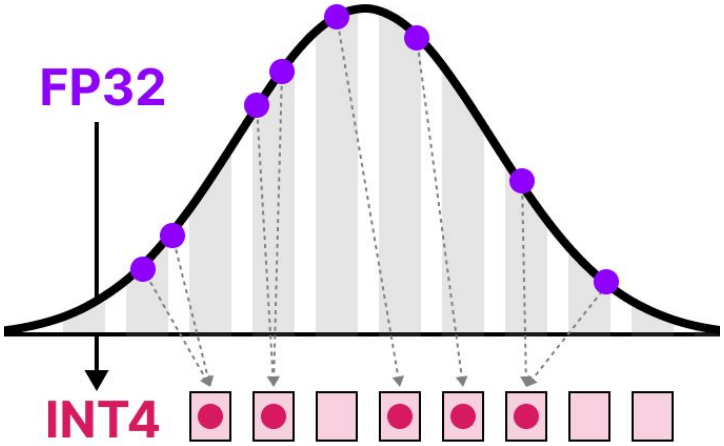


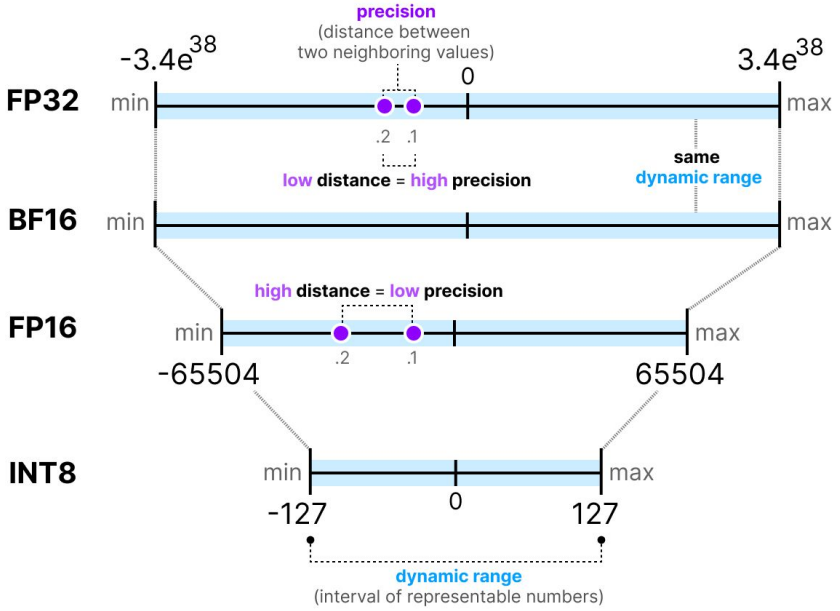
What to compress?

Weights	Activations
Not input dependent	Input dependent
Fixed shape	Not fixed
Constant during inference	Not constant
(offline compression)	(online compression)



01 Quantization





A Visual Guide to Quantization

Demystifying the Compression of Large Language Models

MAARTEN GROOTENDORST

JUL 22, 2024

Weight Quantization

- **Training**
  - We store 3x of weights during training. This can be huge.
  - Going from 16bit->4 reduces memory cost by 4x and enables more sharding options.
  - RAM->chip time is less, however training is often compute bound.

- **Inference**
  - Batch size is low and latency is often memory bound.
  - Thus quantization can provide significant runti improvements.
  - Reduced energy consumption due to less bits transferred.

64-bits $= \frac{64}{8} \times 70B \approx$ **560** GB

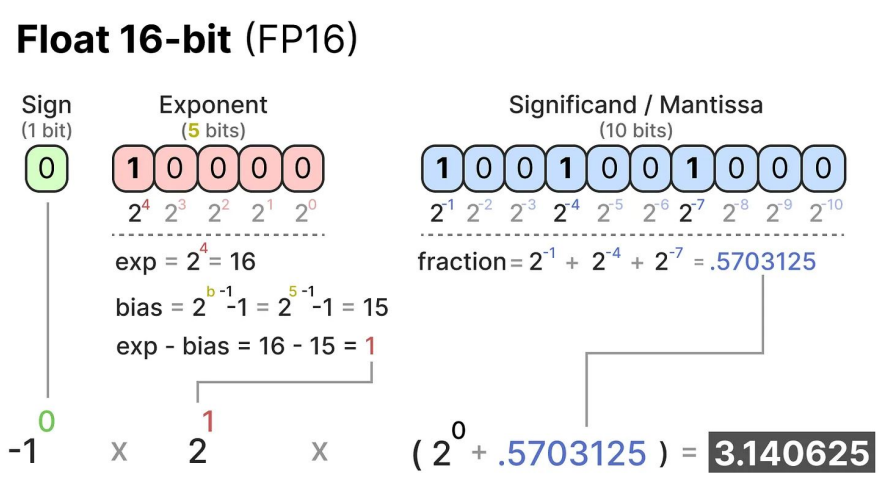
32-bits $= \frac{32}{8} \times 70B \approx$ **280** GB

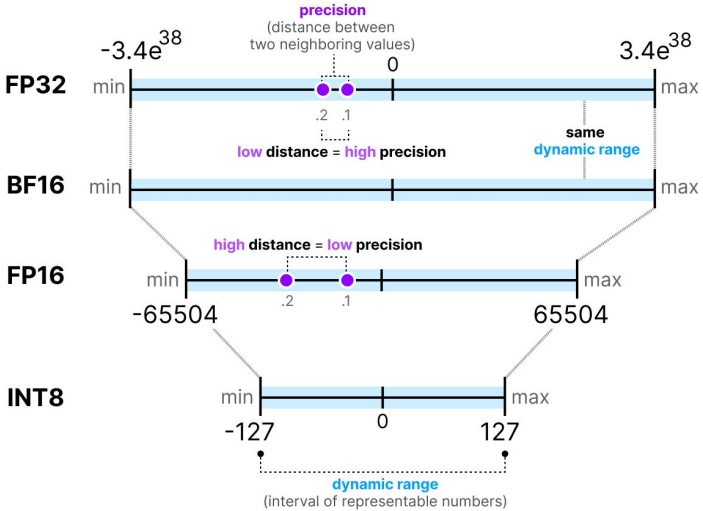
16-bits $= \frac{16}{8} \times 70B \approx$ **140** GB

A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Default Precision is Arbitrary

- "Full" precision = 32, "half" is 16. How about 4?
- How to store "pi"?

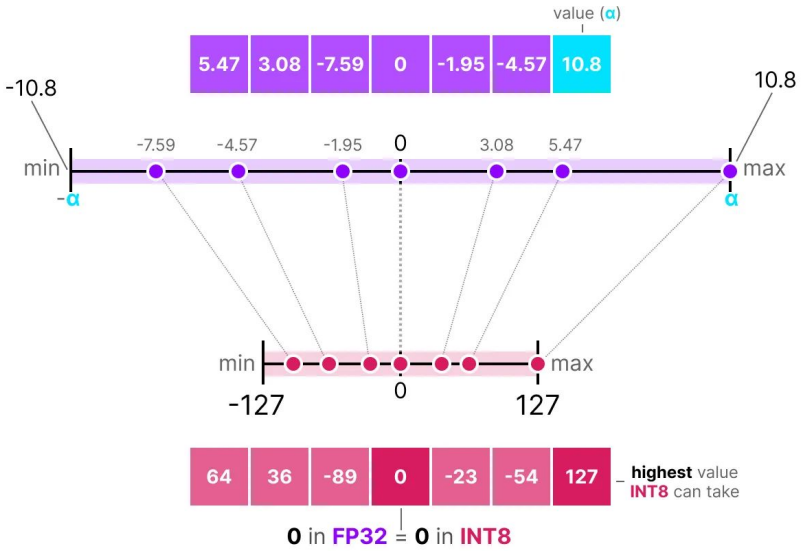




A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Symetric Quantization

- Map largest magnitude float weight to largest possible integer value.
- Keep 0 as 0.
- Equally spaced uniform grid.



A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Symetric Quantization

- Find `scaling factor` (s) that maps largest value (α) to largest integer value
- Use the scale (single float number) to map it back to float.

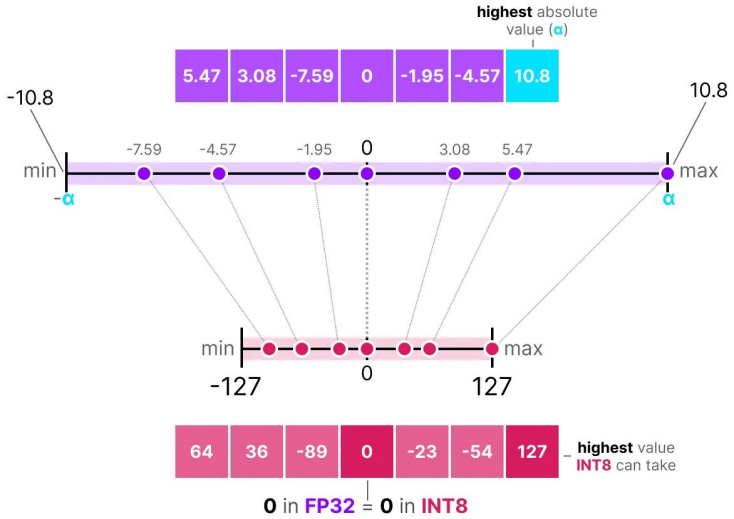
$$s = \frac{2^{b-1}-1}{\alpha} \quad \text{(scale factor)}$$

$$x_{quantized} = \text{round}\left(s \cdot x\right) \quad \text{(quantization)}$$

A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Symetric Quantization

- Map largest magnitude float weight to largest possible integer value.
- Keep 0 as 0.



$$s = \frac{127}{10.8} = 11.76 \quad \text{(scale factor)}$$

$$x_{quantized} = \text{round}\left(11.76 \cdot \blacksquare\right) \quad \text{(quantization)}$$

$$x_{dequantized} = \frac{\blacksquare}{s} \quad \text{(dequantize)}$$

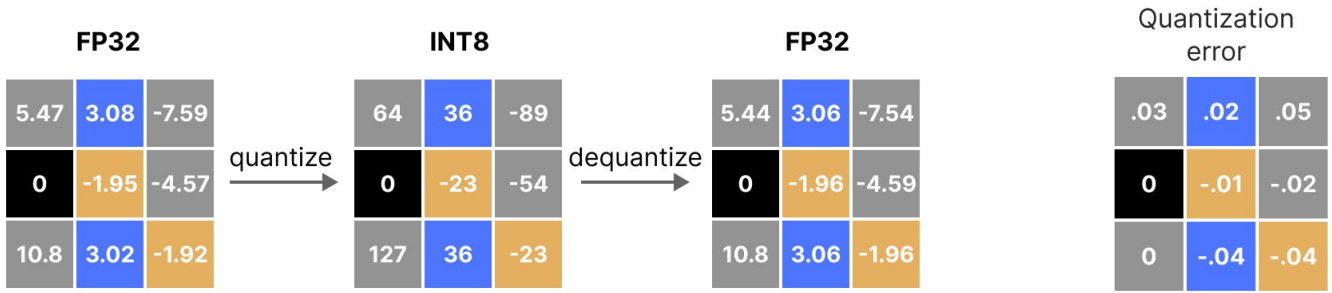
A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Quantization Granularity

- Which weights should share the scale and thus the quantization grid?
  - **Universal** (int4/8 definitions) no-scale.
  - **Per Model?**
  - **Per Tensor**
    - Less scales and more room for hardware/software optimizations.
  - **Per Channel**
    - Each channel has a separate scale.
  - **Per Group**
    - Every N weights share the grid.
    - More difficult to implement/accelerate

Quantization Error

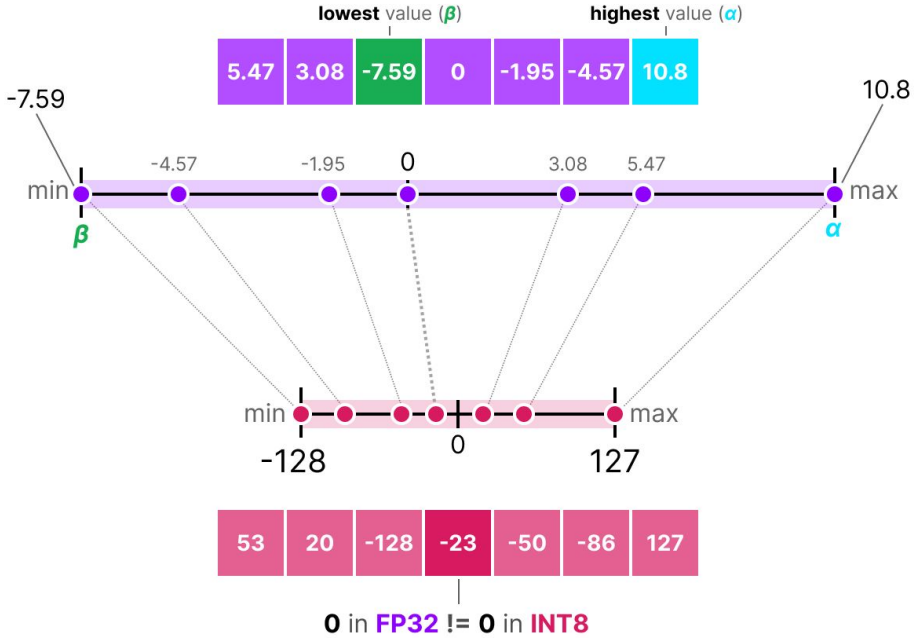
- We can calculate quantization error as following for weights.
  - norm(quant(W)-W)
- How can we minimize this error? This depends greatly on the distribution of the weights.
  - Smaller quantization groups.
  - If not centered: use asymmetric quantization.
  - Clipping



A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Asymmetric Quantization

- What if weights are not symmetric? Let's use zero-point.
- We can always center our distributions. But what if they are not symmetric?



$$S = \frac{128 - -127}{\alpha - \beta} \quad \text{(scale factor)}$$

$$Z = \text{round}\left(-S \cdot \beta\right) - 2^{b-1} \quad \text{(zeropoint)}$$

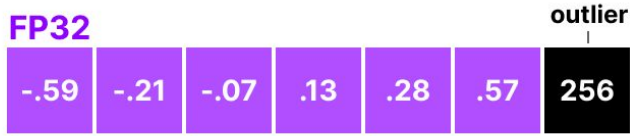
$$X_{quantized} = \text{round}\left(S \cdot X + Z\right) \quad \text{(quantization)}$$

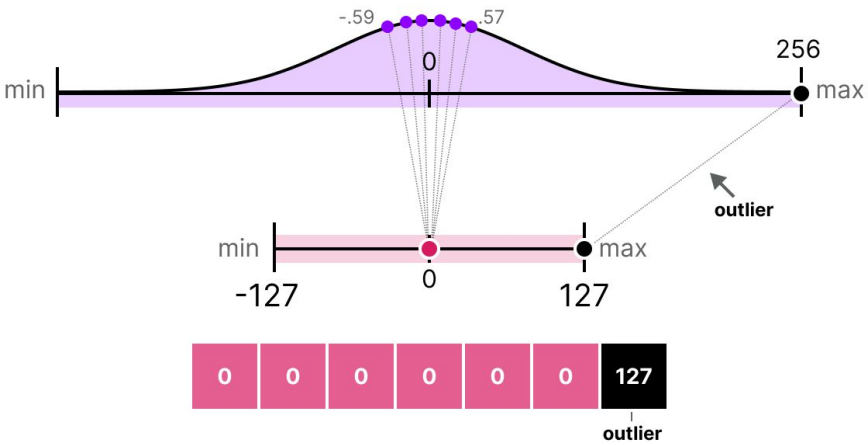
$$X_{dequantized} = \frac{\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare - Z}{S} \quad \text{(dequantize)}$$

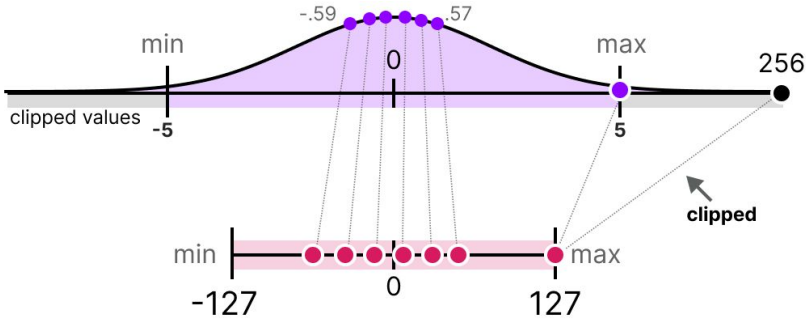
A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Clipping

- What if there are big outliers (and we do have them)?
- Loose precision or clip the large values?
- Minimize error to decide.
  - Error can be reconstruction loss
  - Entropy.



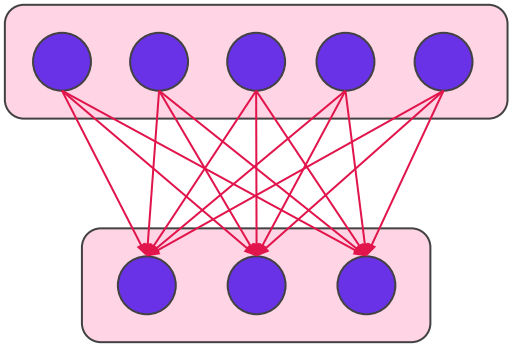


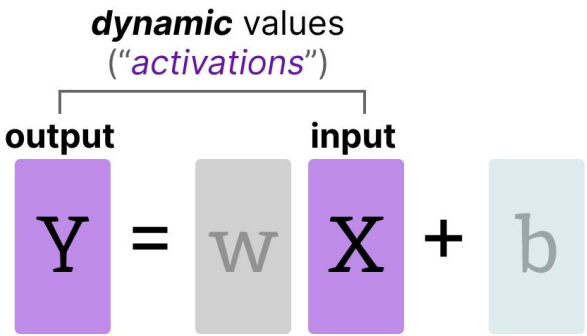


A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Quantizing Activations

- Values are know only during forward pass.
- They also change from token to token.
- Otherwise same as weights.
- Note at the end we care about quality in logits.
  - This can be approximated by looking at the layer output.
- Because of the input dependency:
  - **Static Quant:** Use a validation set to decide the quantization grid.
  - **Dynamic Quant:** Adjust the grid for every token/batch.
- Which activations to quantize? (important ones)
  - KV-cache
  - Attention
  - Large Matmul's.
  - Attention map (if saved).





A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

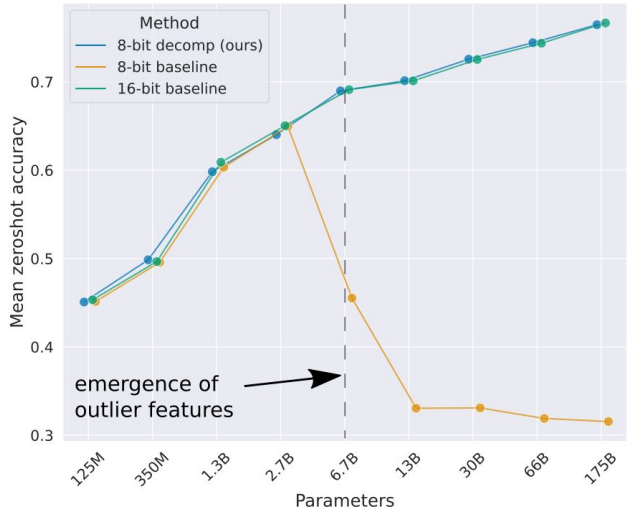
Post Training Quantization (PTQ)

- Absmax Quant
  - Load weights group them to your liking and quantize!
  - Outliers can result on

- How can we improve the baseline?
  - Load weights group them to your liking
  - Outlier removal.
  - Minimize error by updating weights.
  - Clipping

Outlier Removal

- There are only few channels that produce outliers.
- Let's remove them from our quantized MATMULs.
- And only quantize the remaining values.
- Now we have 2 matmuls in different precision.



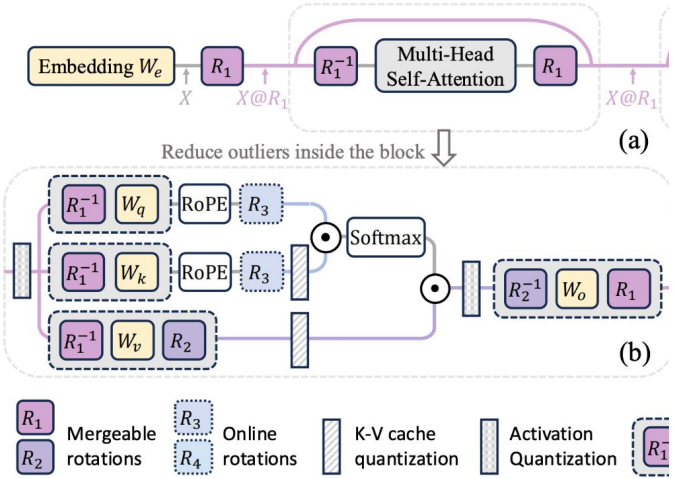
https://timdettmers.com/2022/08/17/llm-int8-and-emergent-features/

Outlier Removal by Rotations

Something fast: Hadamard Transform (nlogn)

SpinQuant: LLM Quantization with Learned Rotations

Zechun Liu* **Changsheng Zhao*** **Igor Fedorov** **Bilge Soran** **Dhruv Choudhary**
Raghuraman Krishnamoorthi **Vikas Chandra** **Yuandong Tian** **Tijmen Blankevoort**
Meta



🥕 QuaRot: Outlier-Free 4-Bit Inference in Rotated LLMs

Saleh Ashkboos
ETH Zurich
saleh.ashkboos@inf.ethz.ch

Amirkeivan Mohtashami
EPFL
amirkeivan.mohtashami@epfl.ch

Maximilian L. Croci
Microsoft Research
mcroci@microsoft.com

Bo Li
ETH Zurich
bolibo@ethz.ch

Pashmina Cameron
Microsoft
pcameron@microsoft.com

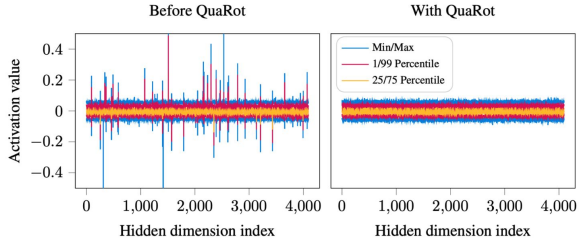
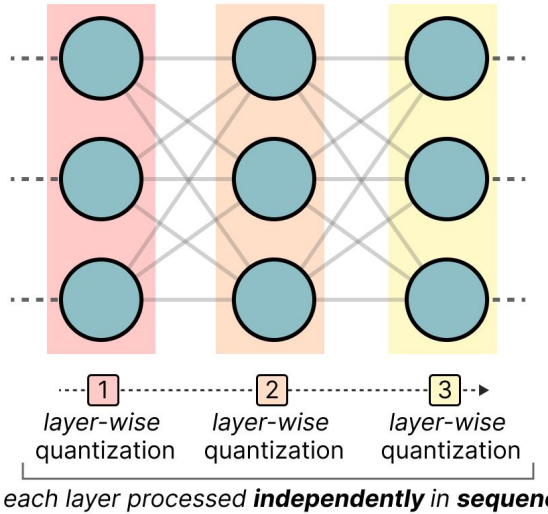


Figure 1: The distributions of activations at the input to the FFN block in LLAMA2-7B model, in the tenth layer. Left: using the default configuration as downloaded from Hugging Face. Right: after processing using QuaRot. The processed distribution has no outliers, leading to superior quantization.

GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers

Elias Frantar* IST Austria
Saleh Ashkboos ETH Zurich
Torsten Hoefler ETH Zurich
Dan Alistarh IST Austria & NeuralMagic

$$\text{argmin}_{\widehat{\mathbf{W}}} ||\mathbf{WX} - \widehat{\mathbf{W}}\mathbf{X}||_2^2.$$



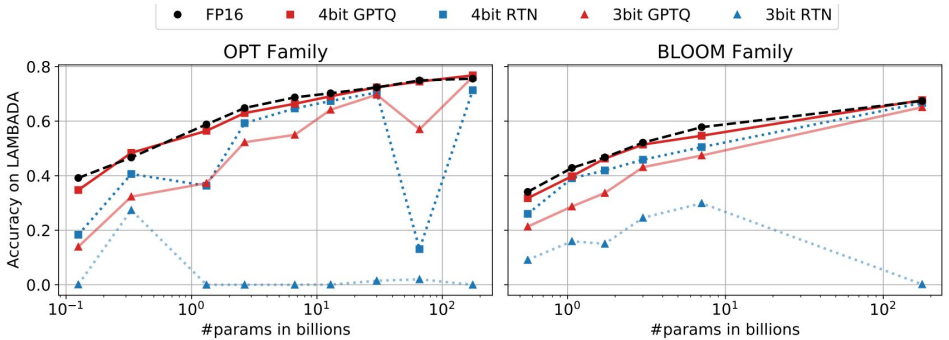


Figure 3: The accuracy of OPT and BLOOM models post-GPTQ, measured on LAMBADA.

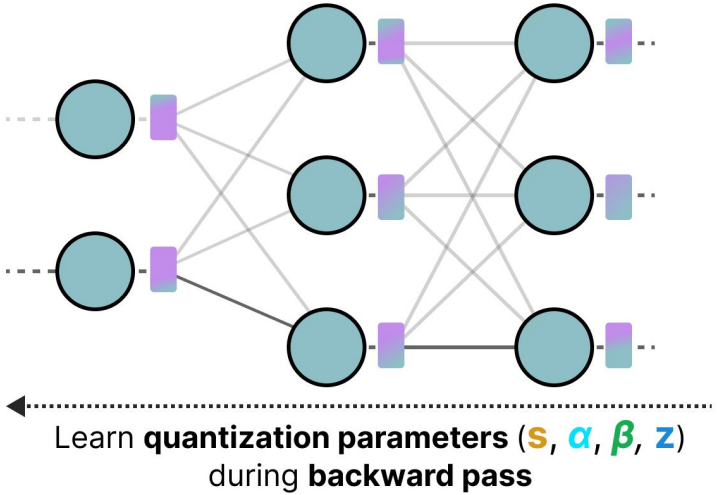
A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

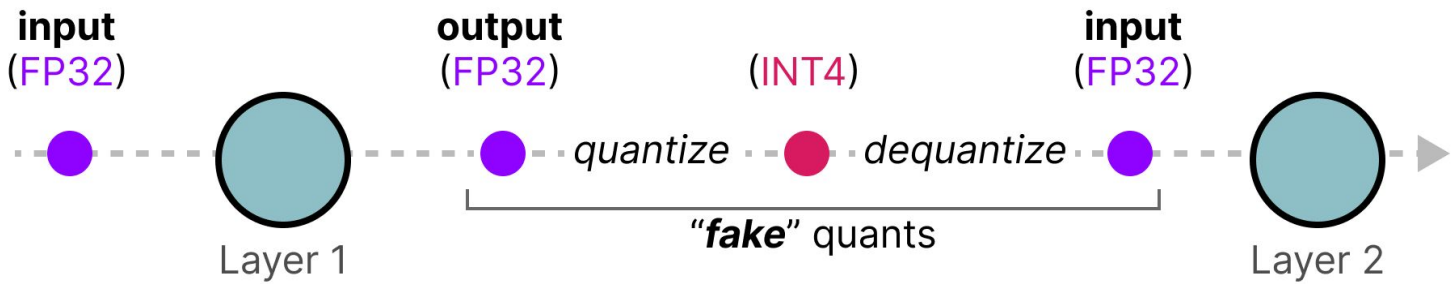
Quantization Aware Training

- Store full-precision weights but quantize just-in-time.
- Weights are kept and updated with full-precision.
- One common trick:

  Straight Through Estimation (STE)

x - stop_grad(x) + stop_grad(f(x))

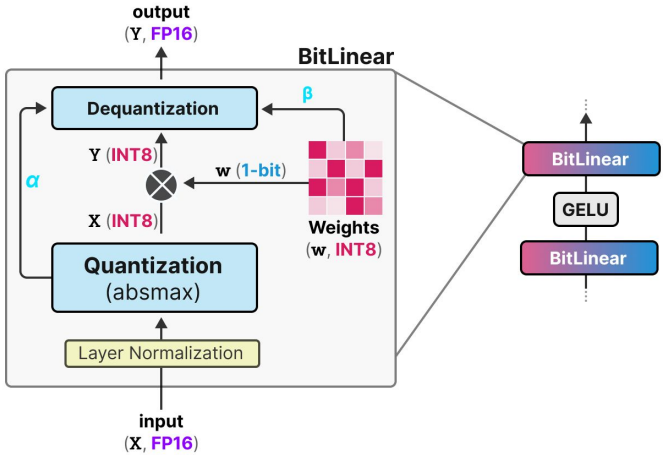




A Visual Guide to Quantization, Maarten Grootendorst, https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

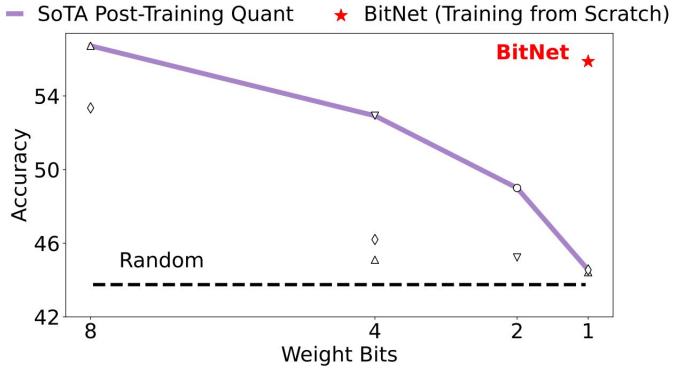
Quantization Aware Training - BitNet

- QAT >> PTQ when compression becomes non iso-quality.
- You can go to the extreme of 1 bit.



BitNet: Scaling 1-bit Transformers for Large Language Models

Hongyu Wang*†‡ Shuming Ma*† Li Dong† Shaohan Huang†
Huaijie Wang§ Lingxiao Ma† Fan Yang† Ruiping Wang‡ Yi Wu§ Furu Wei†◇
† Microsoft Research ‡ University of Chinese Academy of Sciences § Tsinghua University
https://aka.ms/GeneralAI

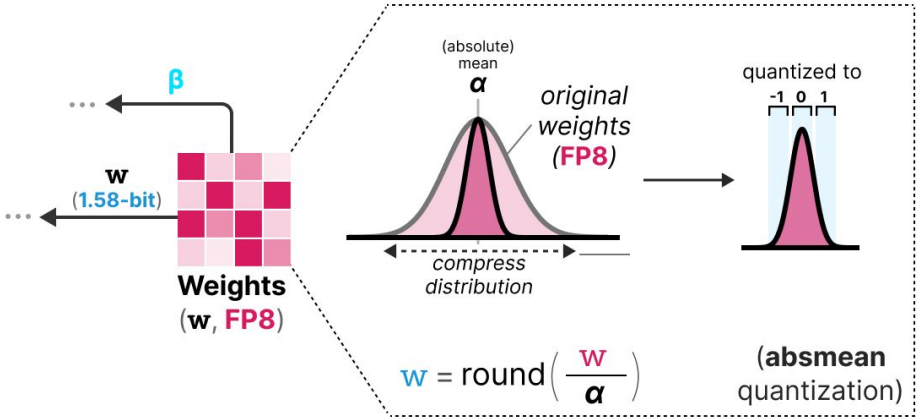


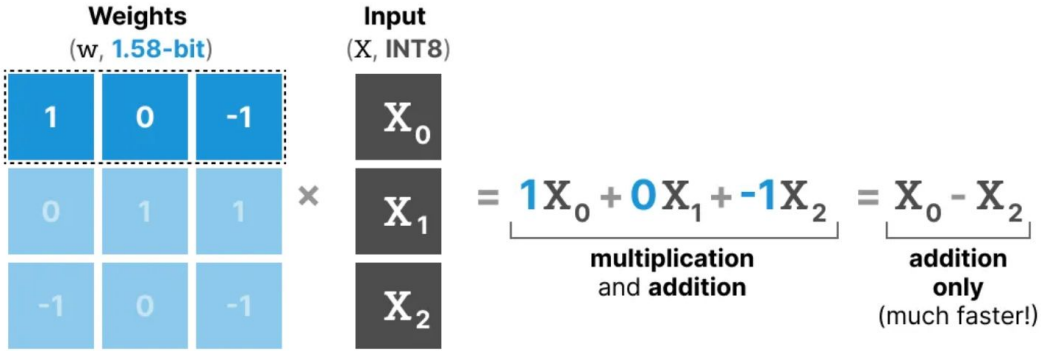
Quantization Aware Training - BitNet + 0

- Quality is way better if you use an additional 0 value.

The Era of 1-bit LLMs:
All Large Language Models are in 1.58 Bits

Shuming Ma* Hongyu Wang* Lingxiao Ma Lei Wang Wenhui Wang
Shaohan Huang Li Dong Ruiping Wang Jilong Xue Furu Wei°
https://aka.ms/GeneralAI





Quantizing Gradients

- What if training is run on multiple devices with slow communication?
  - Gradient all-reduce is a major bottleneck.
- What if doing end-to-end quantization?
  - Requires quantizing gradients.
  - Are activation gradients quantized?
- Key difference is that the direction and variance matters more. So we should adjust our error criteria.

Adaptive Gradient Quantization for Data-Parallel SGD

Fartash Faghri[1,2*] **Iman Tabrizian**[1,2*] **Ilia Markov**[3] **Dan Alistarh**[3,4]

Daniel M. Roy[1,2] **Ali Ramezani-Kebrya**[2]

[1]University of Toronto [2]Vector Institute [3]IST Austria [4]NeuralMagic

faghri@cs.toronto.edu iman.tabrizian@mail.utoronto.ca alir@vectorinstitute.ai

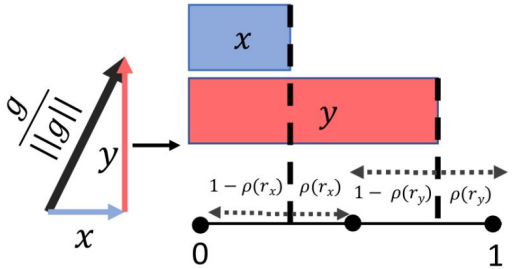
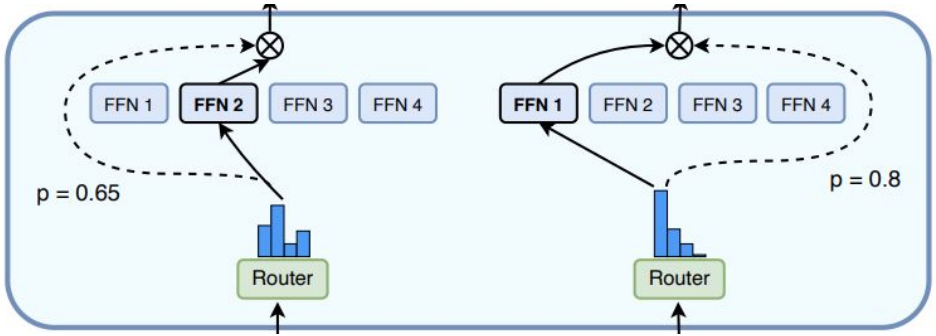


Figure 2: Random quantization of normalized gradient.

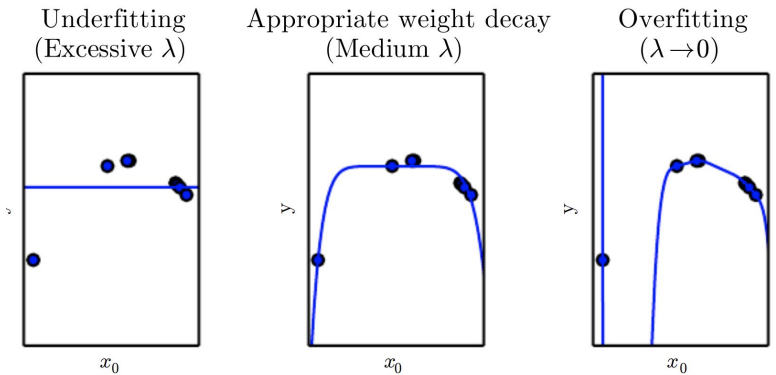
02 Sparsity



Weight Decay and What Happens

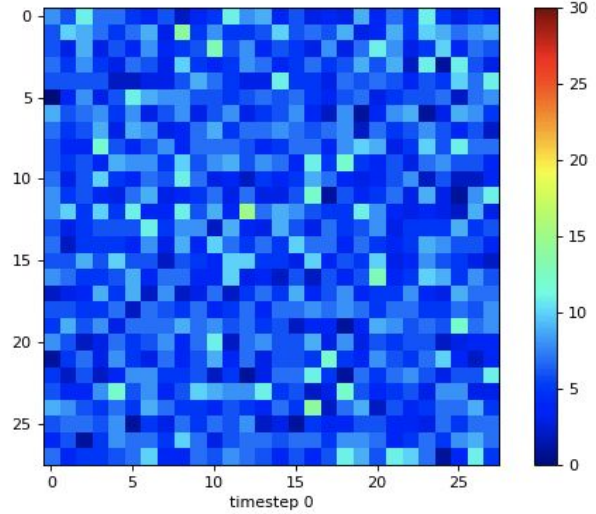
- When data is limited, you often need some kind of regularization.
- Weight decay is a popular method, which pulls weights towards zero.

$$L_{new}(w) = L_{original}(w) + \lambda w^T w$$

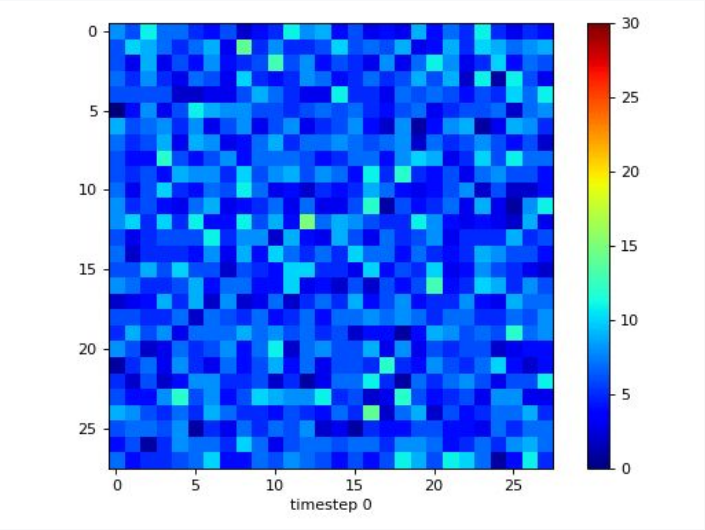


https://paperswithcode.com/method/weight-decay

Evaluation of Connections

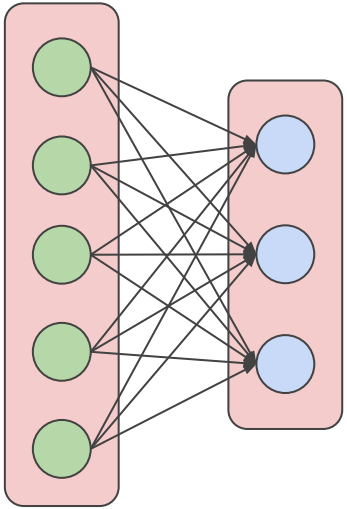


Before training

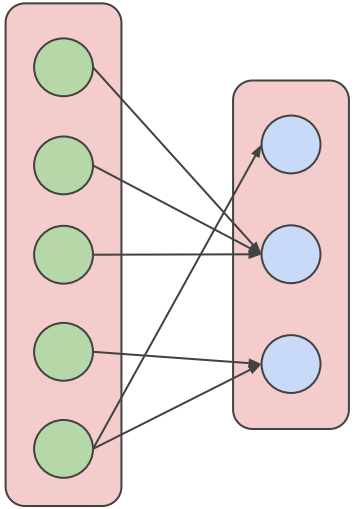


Evaluation of first layer connections during MNIST MLP training.

Sparse Networks



Dense Network
Sparsity: 0

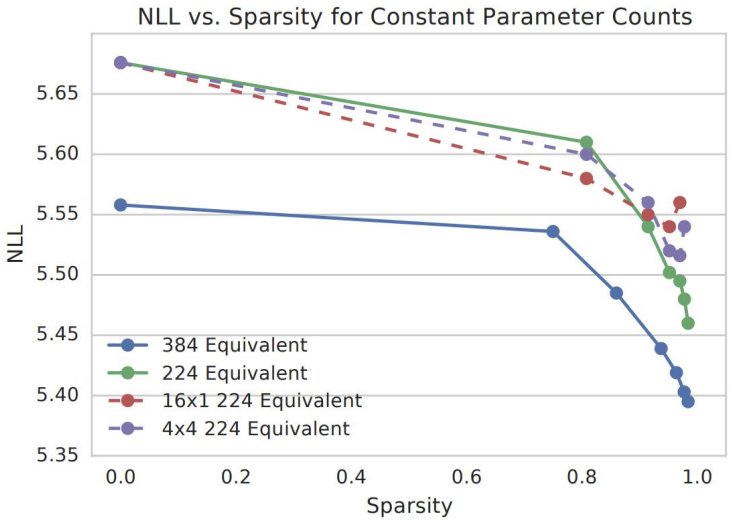


Sparse Network
Sparsity: 60%

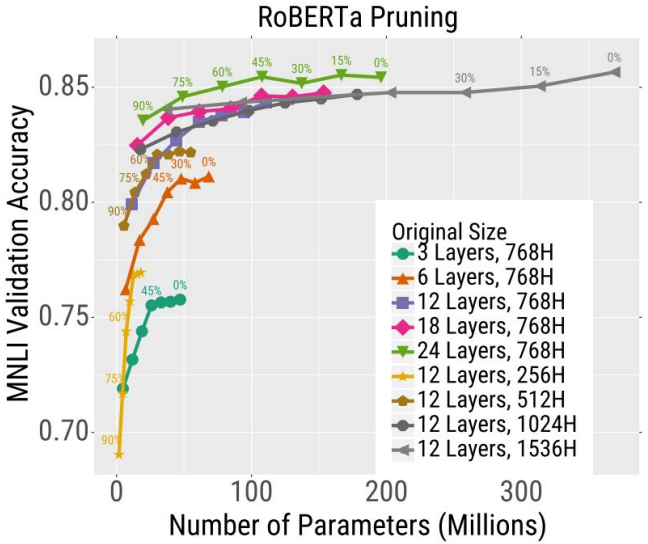
- **On device training/inference:** Reduces FLOPs and network size drastically without harming performance.
- **Reduced memory footprint:** We can fit wider/deeper networks in memory and get better performance from the same hardware.
- **Architecture search:** Causal relationships? Interpretability?

Sparse networks perform better

for the same parameter count.



Efficient Neural Audio Synthesis, Kalchbrenner, N., Elsen, E., Simonyan, K., Noury, S., Casagrande, N., Lockhart, E., Stimberg, F., Oord, A.,Dieleman, S., and Kavukcuoglu, K., 2018



Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers, Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, Joseph E. Gonzalez, 2020

What to Sparsify?

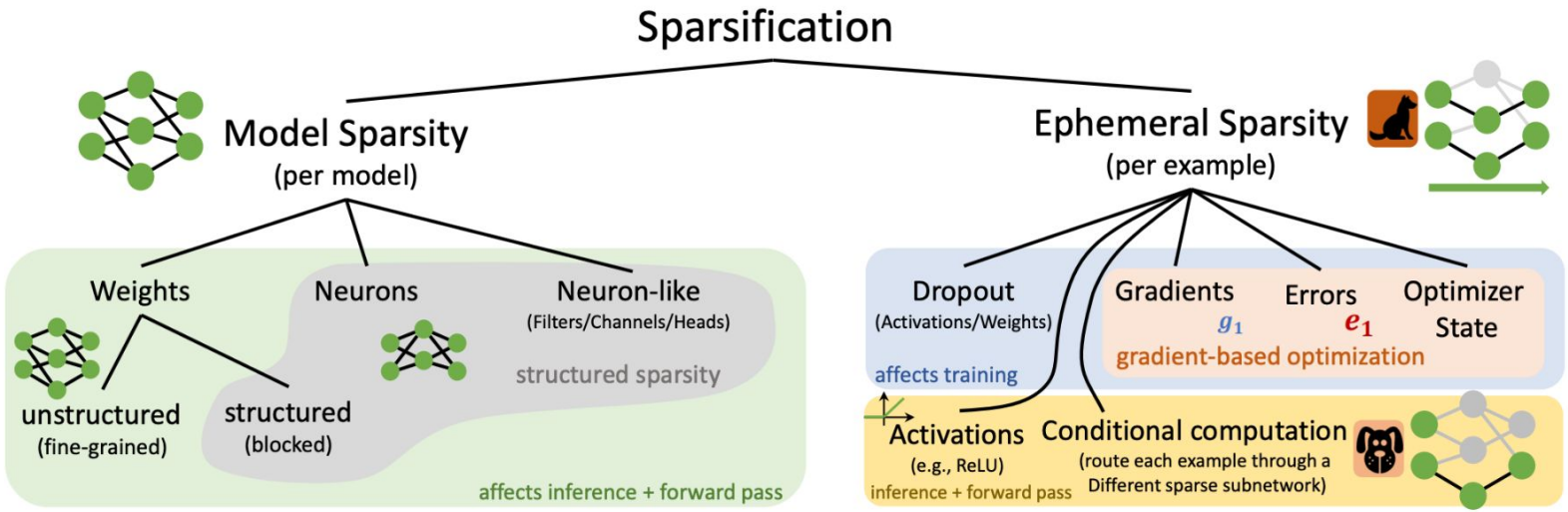
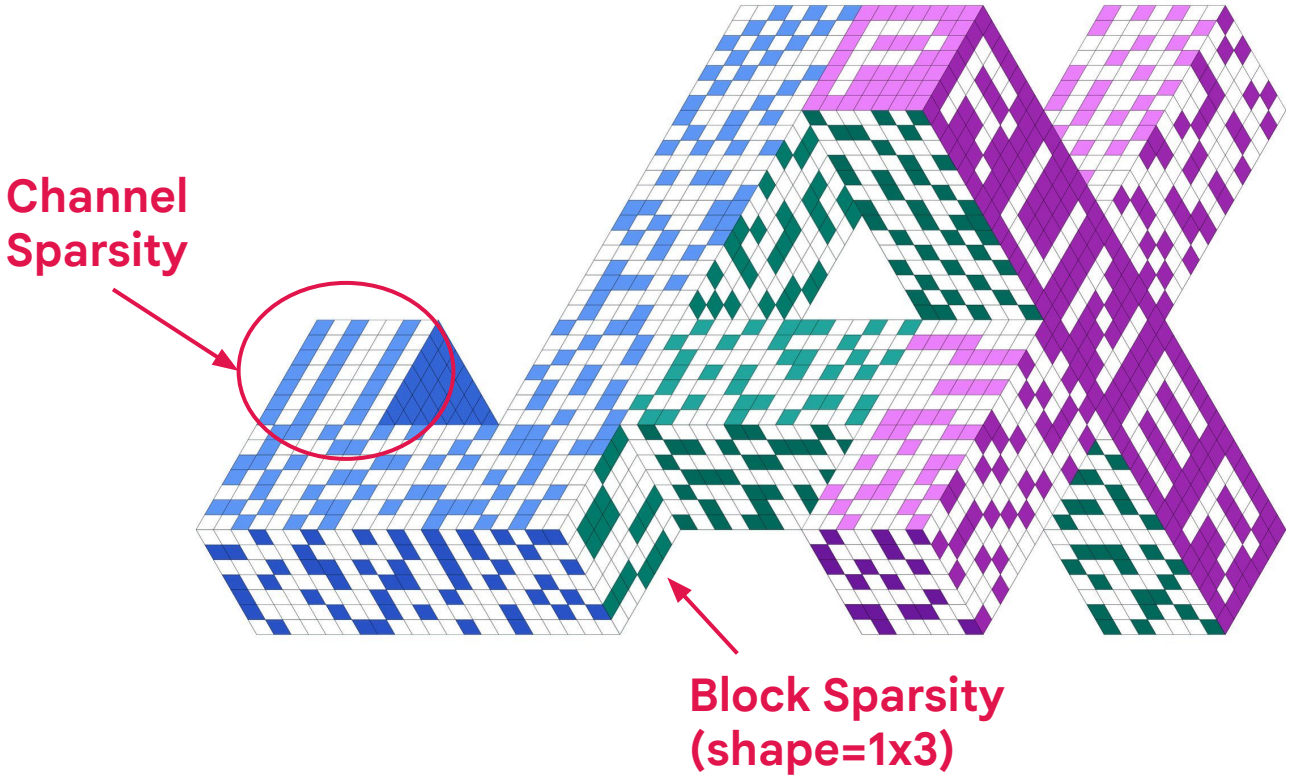


Fig. 6. Overview of DNN elements to sparsify.

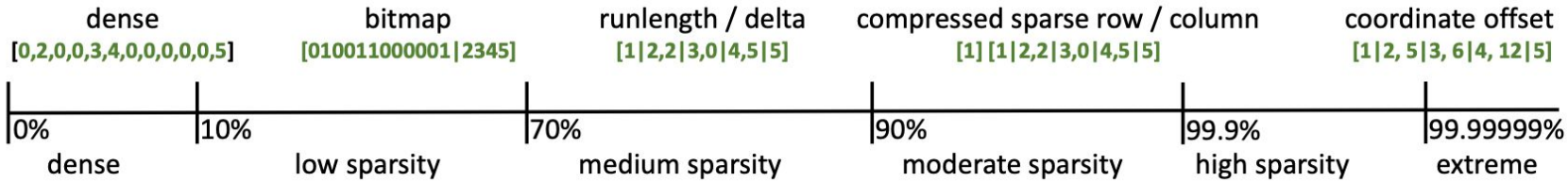
Torsten Hoefler et al., Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks

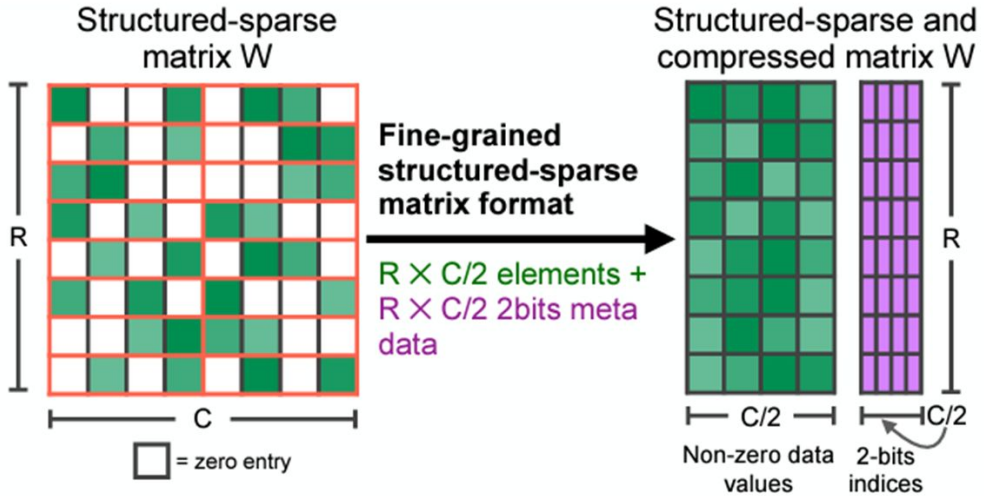
How to Sparsify?



Torsten Hoefler et al., Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks

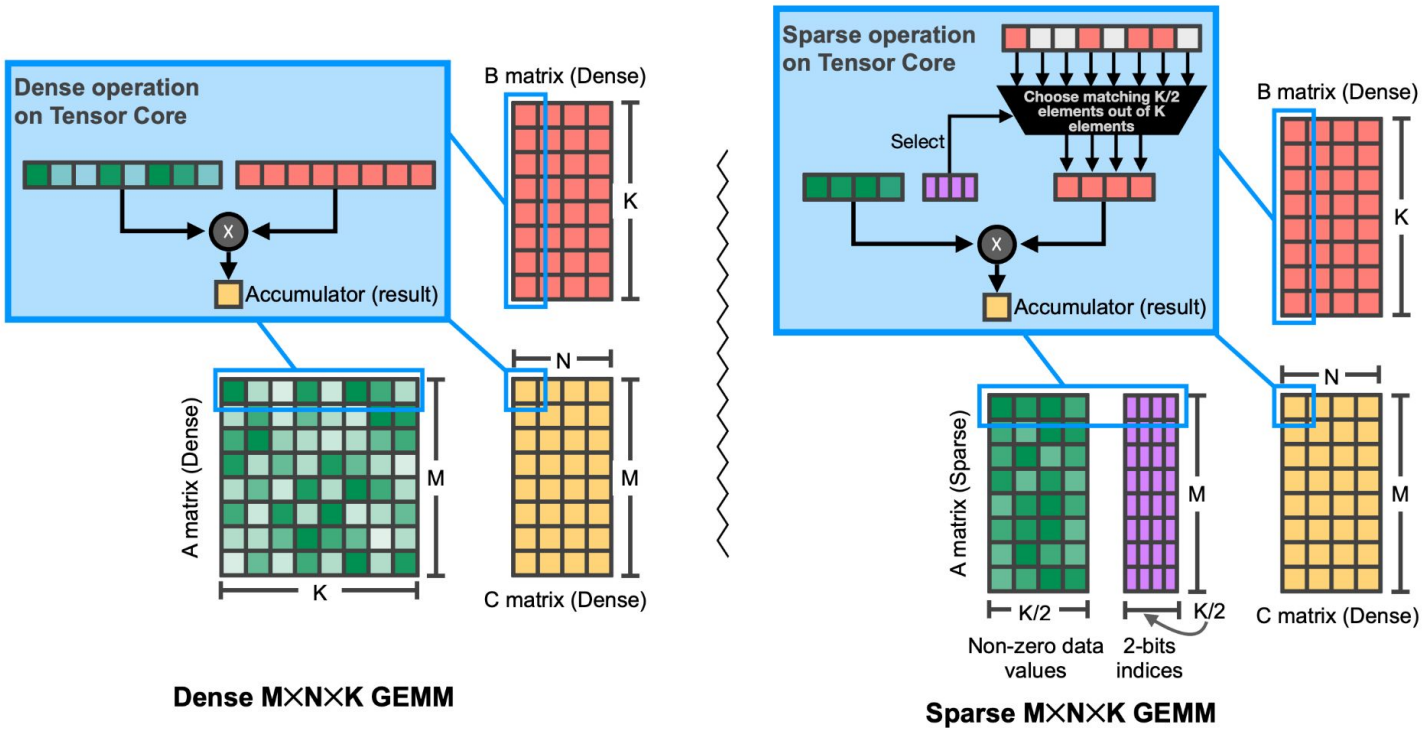
Sparse Storage Formats





Torsten Hoefler et al., Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks
https://developer.nvidia.com/blog/structured-sparsity-in-the-nvidia-ampere-architecture-and-applications-in-search-engines/

2:4 Sparsity Acceleration



Accelerating Sparse Deep Neural Networks, Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, Paulius Micikevicius

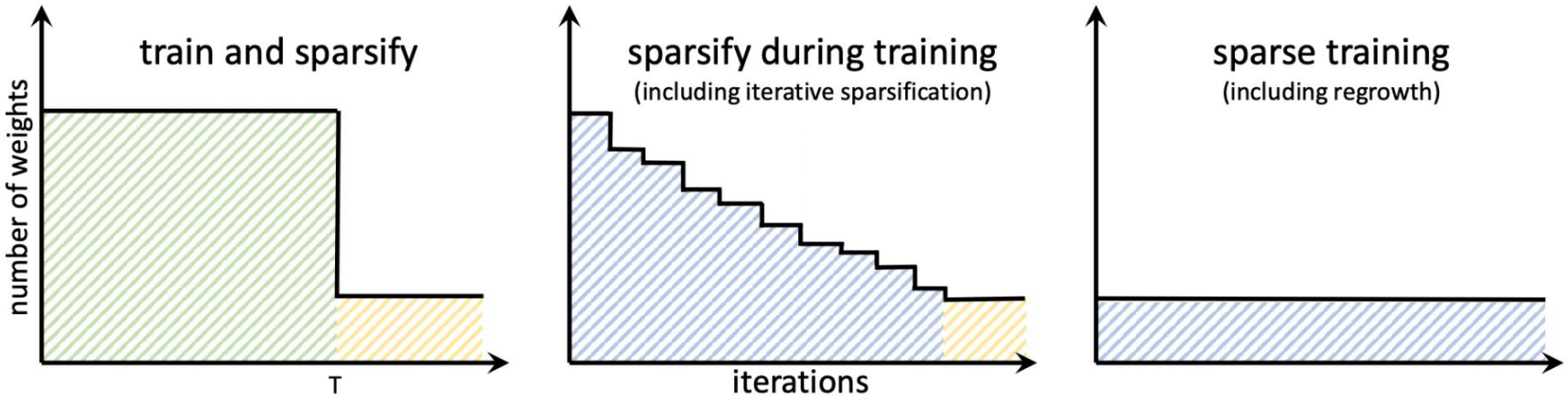
Accelerating sparsity

Is difficult, but possible

- ➔ Block Sparse Kernels: Efficient sparse operations.
- ➔ Efficient WaveRNN: Text-to-Speech
- ➔ Optimizing Speech Recognition for the Edge: Speech Recognition
- ➔ Fast Sparse Convnets: Fast mobile inference for vision models with sparsity (1.3 - 2.4x faster).
- ➔ Nvidia GPUs support **2:4 sparsity** starting A100s.
- ➔ Cerebra's Sparse Llama

When to Sparsify?

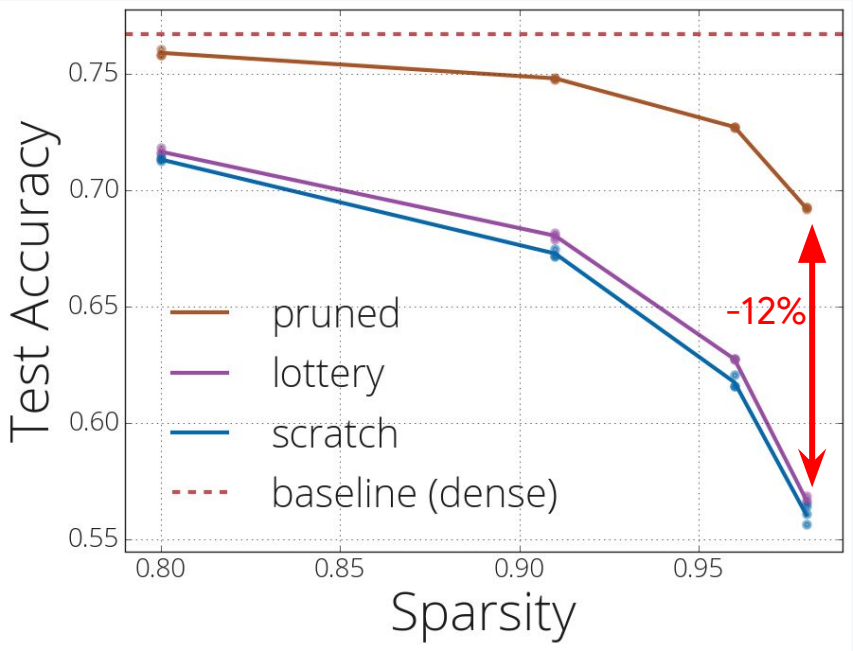
- Similar to quantization sparsity can be applied during or after training.



Torsten Hoefler et al., Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks

How do we find sparse networks?

- **Pruning** method requires dense training: (1) limits the biggest sparse network we can train (2) not efficient.
- **Training from scratch** performs much worse.
- **Lottery*** initialization doesn't help.



Test accuracy of ResNet-50 networks trained on ImageNet-2012 dataset at different sparsity levels*.

*** The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks**
Jonathan Frankle, Michael Carbin, ICLR 2019

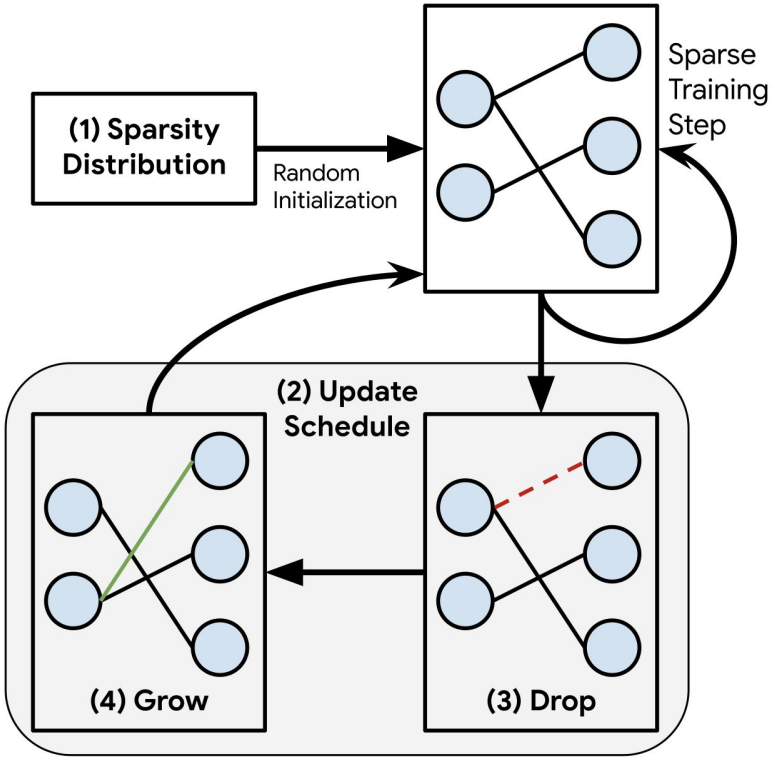
*** The Difficulty of Training Sparse Neural Networks**
Utku Evci, Fabian Pedregosa, Aidan Gomez, Erich Elsen, 2019

Can we train sparse neural networks end-to-end?

(without ever needing the dense parameterization)

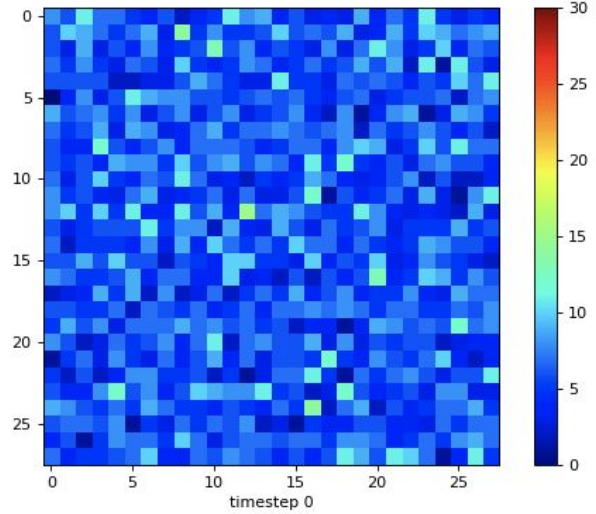
(as good as the dense-to-sparse methods)

Dynamic Sparse Training (DST)

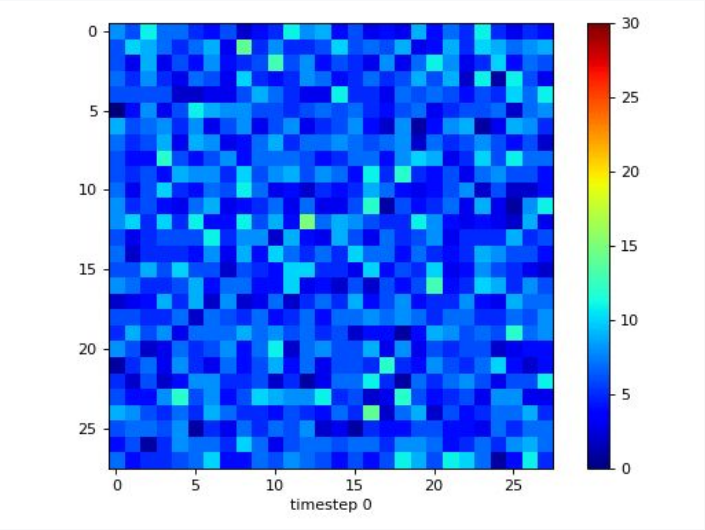


- **Start from a random sparse network.**
- **Train the sparse network.**
- **Every N steps update connectivity:**
  - Drop least magnitude connections
  - Grow new ones using gradient information.

Evaluation of Connections



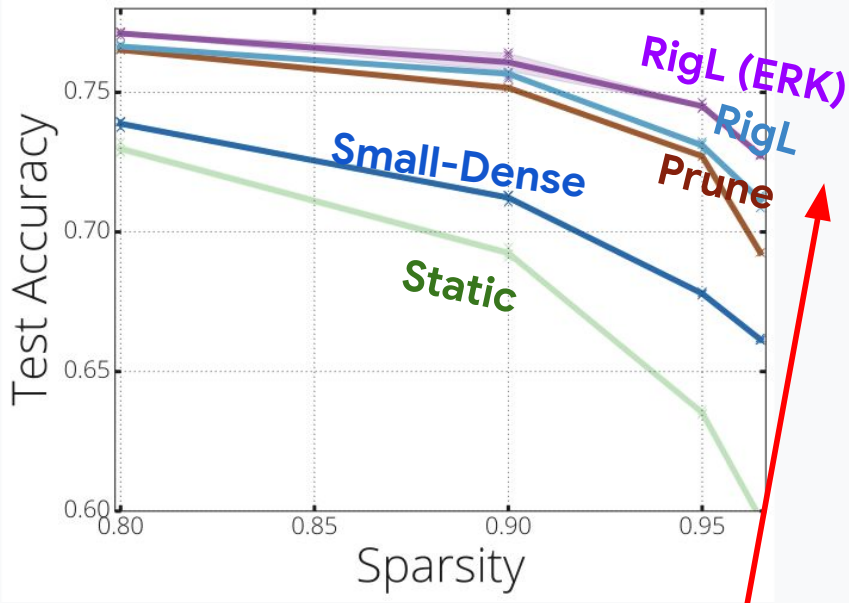
Before training



Evaluation of first layer connections during MNIST MLP training.

Exceeding Pruning Performance

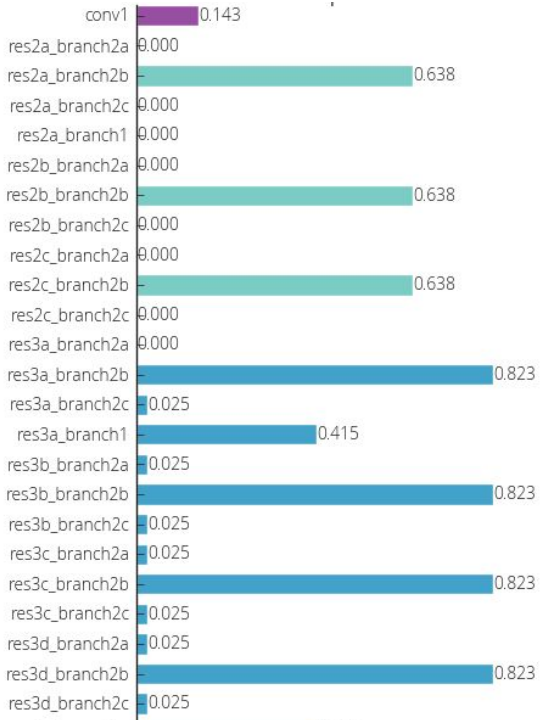
- ➔ RigL
  - ◆ **outperforms** pruning
- ➔ ERK sparsity distribution:
  - ◆ greater performance
  - ◆ more FLOPs.



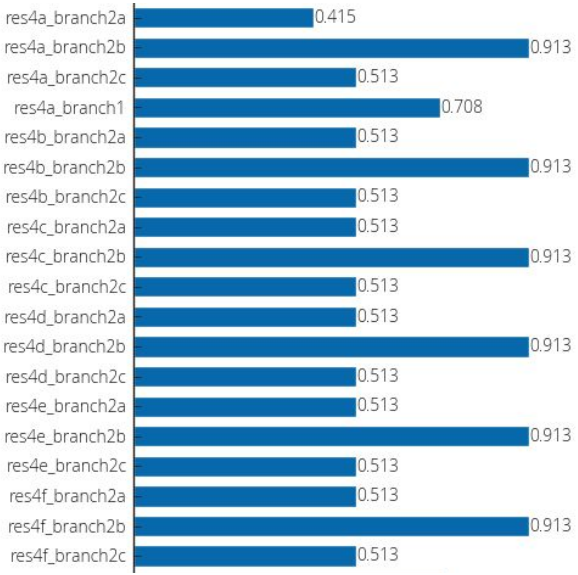
14x less FLOPs and parameters.

Google Research

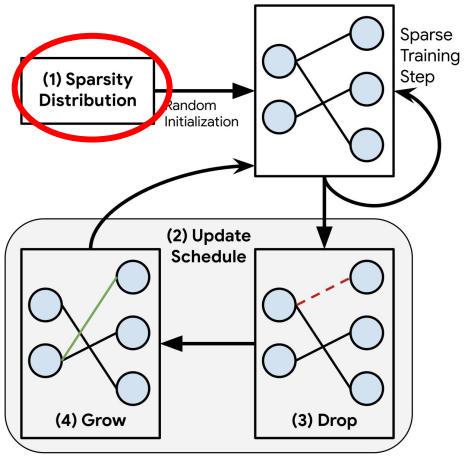
80% ERK on Resnet-50

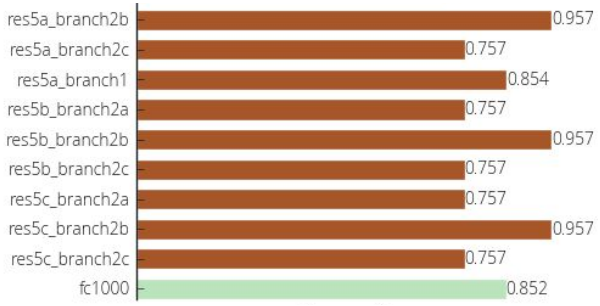


Stage 1-2-3



Stage 4





Stage 5

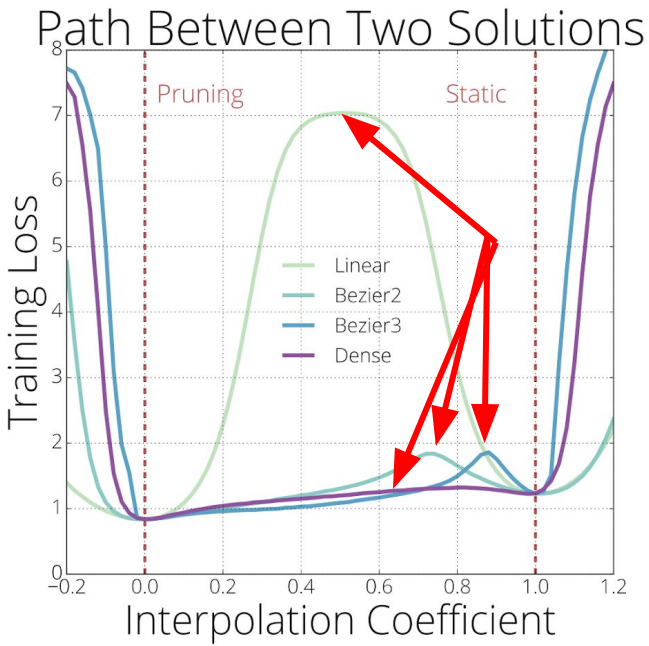
Sparse MobileNets

- ➔ Difficult to prune.
- ➔ Much better results with **RigL.**

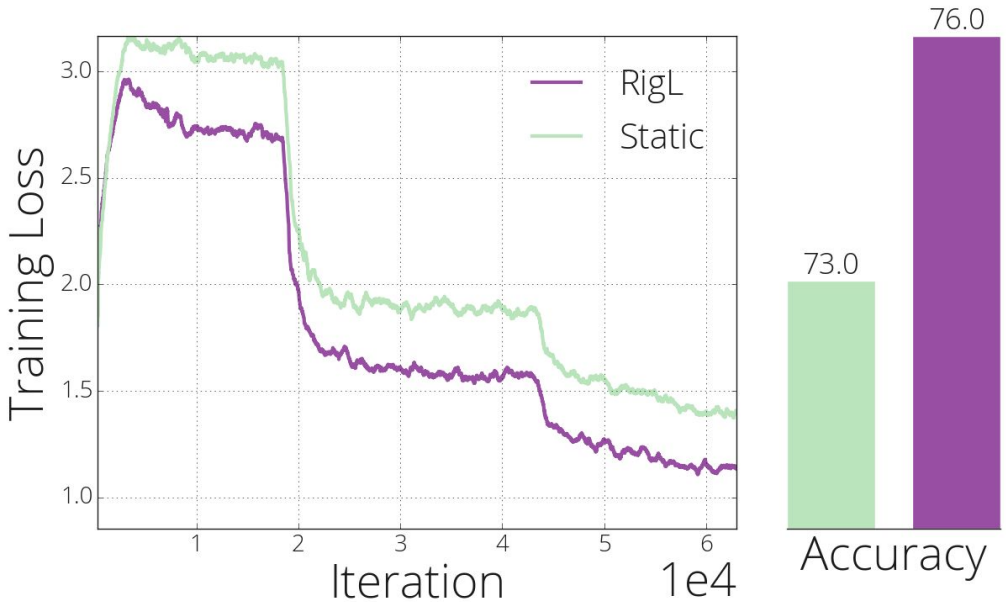
S	Method	Top-1	FLOP-Inf
0.75	Small-Dense$_{5\times}$	66.0±0.11	0.23x
	Pruning (Zhu)	67.7	0.27x
	RigL$_{5\times}$	71.5±0.06	0.27x
	RigL$_{5\times}$ (ERK)	**71.9±0.01**	0.52x
0.90	Small-Dense$_{5\times}$	57.7±0.34	0.09x
	Pruning (Zhu)	61.8	0.12x
	RigL$_{5\times}$	67.0±0.17	0.12x
	RigL$_{5\times}$ (ERK)	**68.1±0.11**	0.27x
	Dense	72.1±0.17	1x (1.1e9)
0.75	Big-Sparse$_{5\times}$	76.4±0.05	0.98x
	Big-Sparse$_{5\times}$ (ERK)	**77.0±0.08**	1.91x

same parameter/flops: **4.3% absolute improvement** in Top-1 Accuracy.

Bad Local Minima and DST



Static training stucks in a suboptimal basin.



RigL helps escaping from it.

More results in our workshop paper
https://arxiv.org/pdf/1906.10732.pdf

Google Research

Top-KAST

- Have N% more then you need during sparse training.

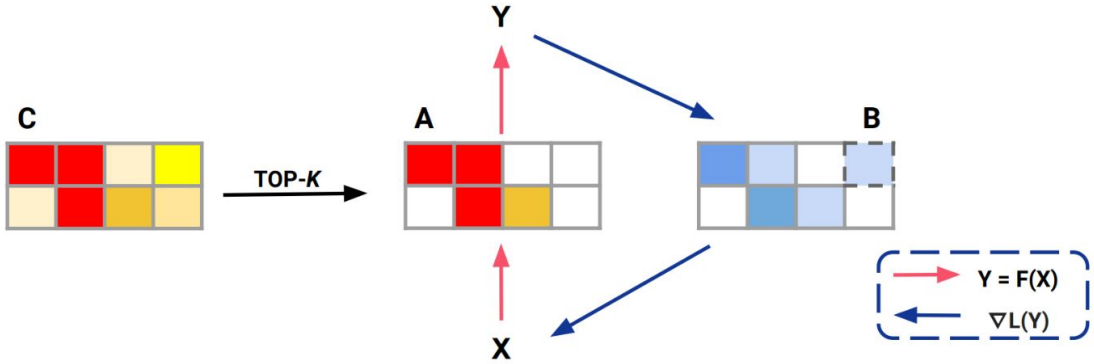
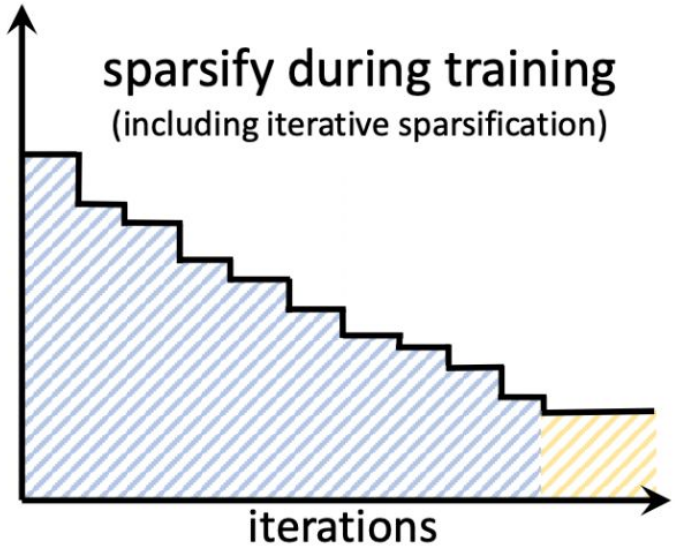


Figure 1: A diagramatic illustration of Top-KAST. While initialised with an effectively random mask, Top-KAST explores different permutations by updating an exploration set of weights and choosing the ones with greatest magnitude.

STE Sparsity Gradual

- Use STE with Gradual sparsity increase (like QAT).

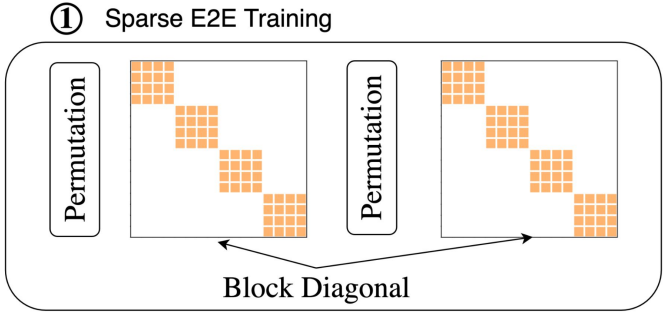


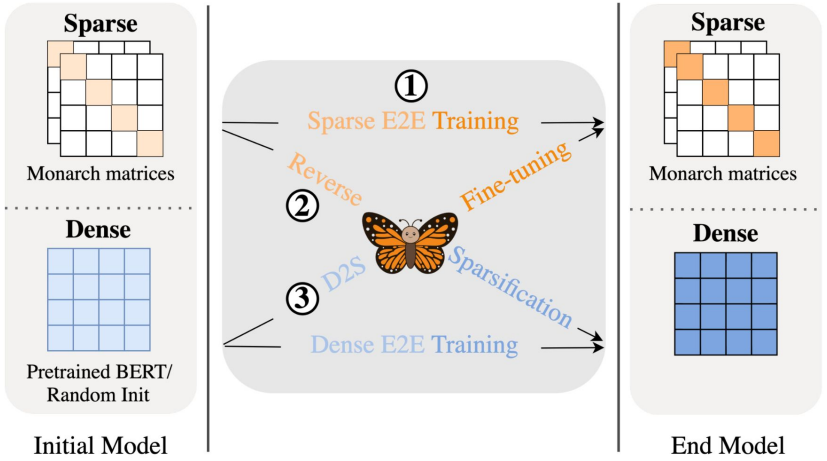
https://pytorch.org/blog/accelerating-neural-network-training/

Sparse->Dense

Monarch: Expressive Structured Matrices for Efficient and Accurate Training

Tri Dao[1], Beidi Chen[1], Nimit Sohoni[1], Arjun Desai[1], Michael Poli[1], Jessica Grogan[2], Alexander Liu[3], Aniruddh Rao[3], Atri Rudra[2], and Christopher Ré[1]

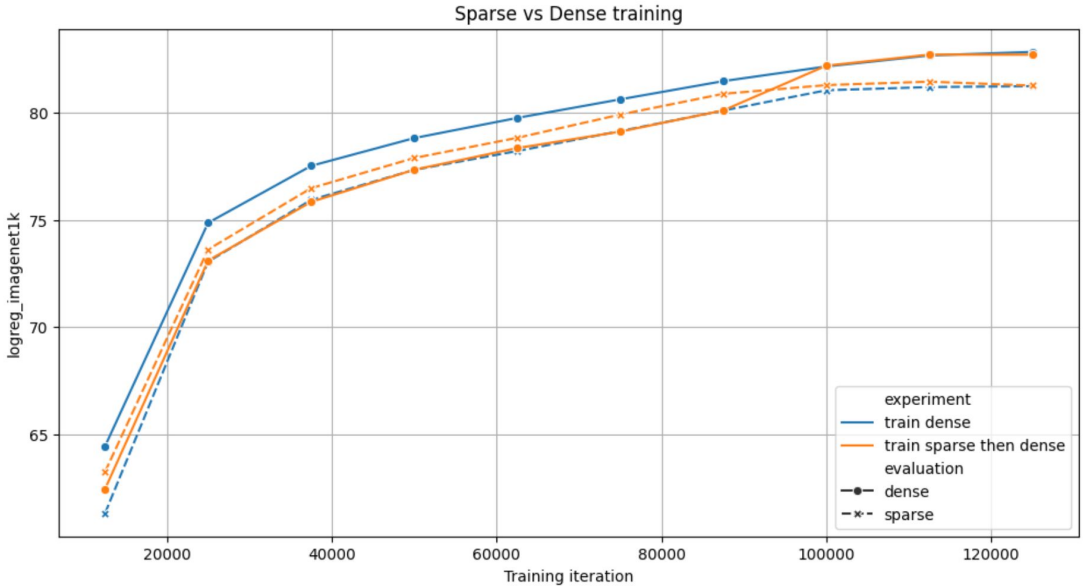




Google Research

Sparse->Dense

- You can train sparse, save compute and finish dense.

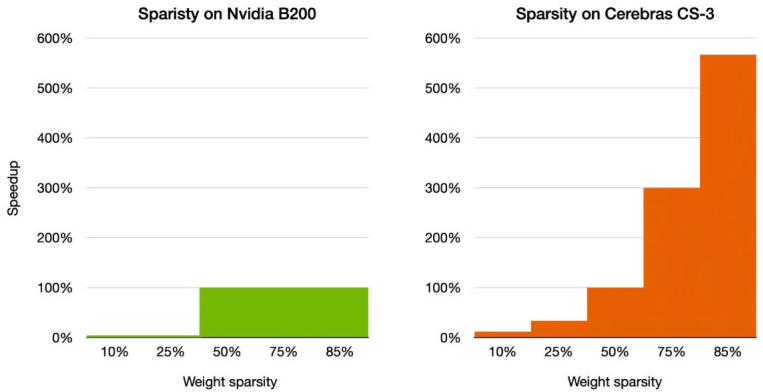


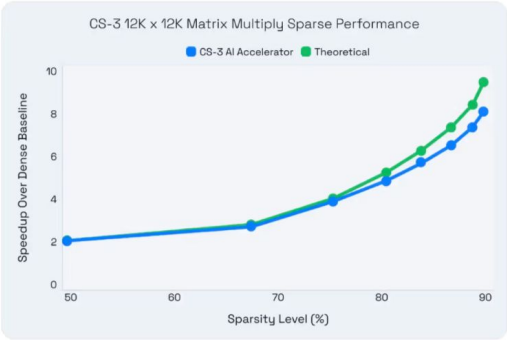
https://pytorch.org/blog/accelerating-neural-network-training/

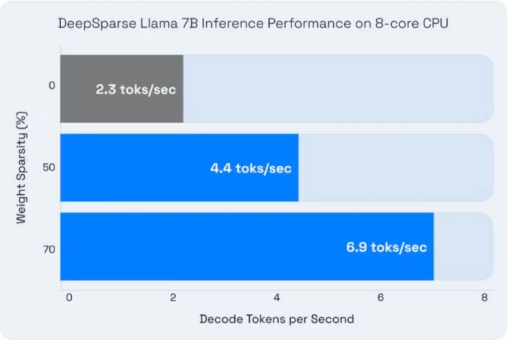
Google Research

Sparse Llama - Cerebras

- Accelerating sparsity is possible.
- It's all about hardware
- And compiler+implementation!





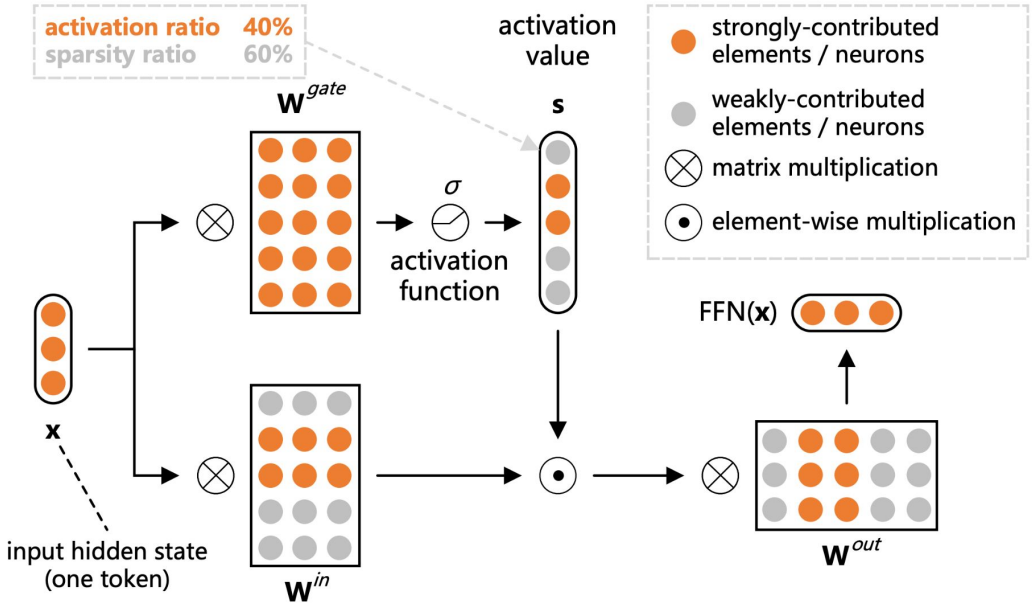


https://www.cerebras.ai/blog/introducing-sparse-llama-70-smaller-3x-faster-full-accuracy

Activation Sparsity

- W_gate activations are generally sparse.
- Taking top-k works very well.

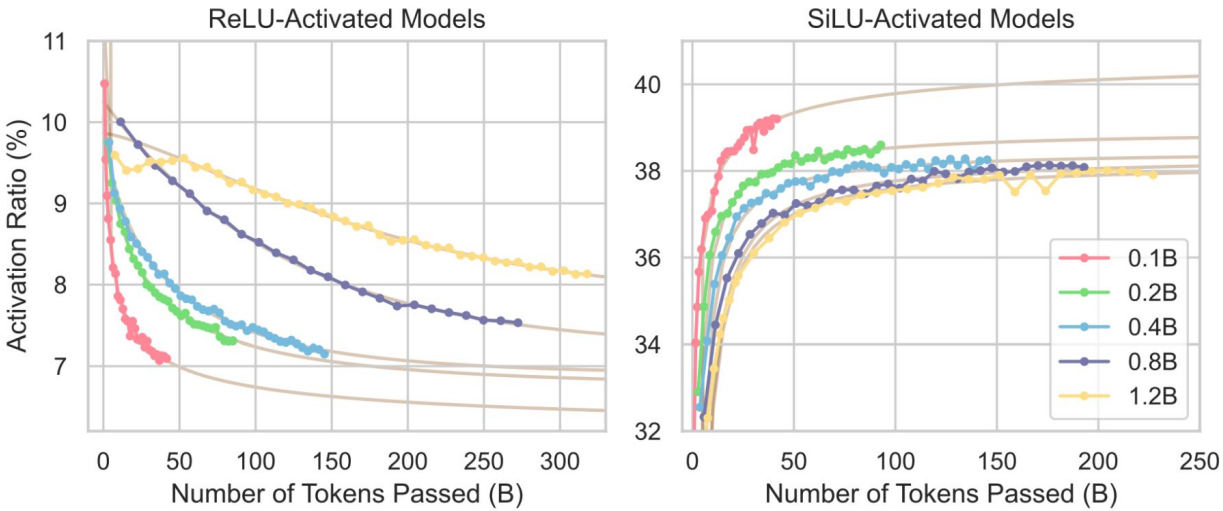
Sparsity is different for each token (activation*)



Sparsing Law: Towards Large Language Models with Greater Activation Sparsity

Activation Sparsity

- Sparsity increase over the course of the training
- It can be also enforced with Top-K



https://pytorch.org/blog/accelerating-neural-network-training/

Activation Sparsity vs MoEs

- MoEs and Activation sparse networks are quite similar
- Activation sparse:
  - Router: gating FFN with argmax
  - Expert: Single neuron experts (many of them)
  - Combine: Sum
- MoEs
  - Larger experts and few of them
  - Router is much cheaper
  - Combine according to the router weight

https://pytorch.org/blog/accelerating-neural-network-training/

Learn more about sparsity

Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks

TORSTEN HOEFLER, ETH Zürich, Switzerland
DAN ALISTARH, IST Austria, Austria
TAL BEN-NUN, ETH Zürich, Switzerland
NIKOLI DRYDEN, ETH Zürich, Switzerland
ALEXANDRA PESTE, IST Austria, Austria

Google Research

Sparse Scaling Laws

- Effect of weight sparsity can be characterized as a multiplier for the model size term of the Chinchilla Scaling law

$$L(S, N, D) = \left(a_S(1-S)^{b_S} + c_S\right) \cdot \left(\frac{1}{N}\right)^{b_N} + \left(\frac{a_D}{D}\right)^{b_D} + c,$$

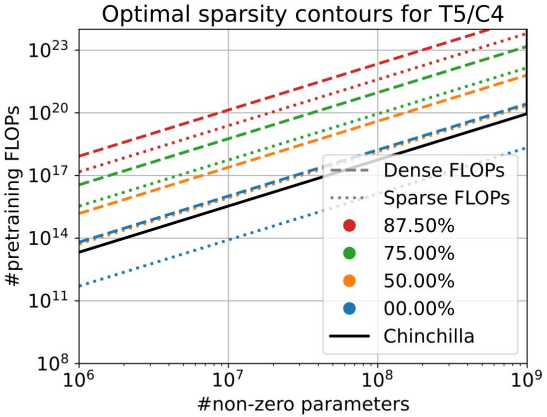
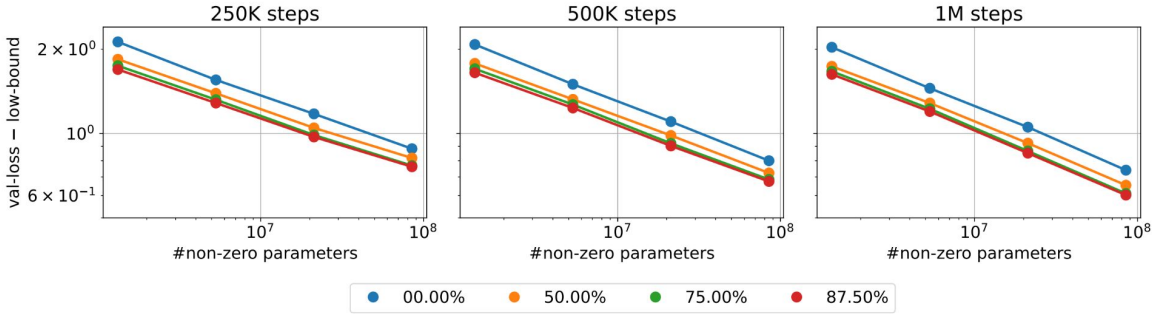


Figure 4: Optimal T5 sparsity contours.

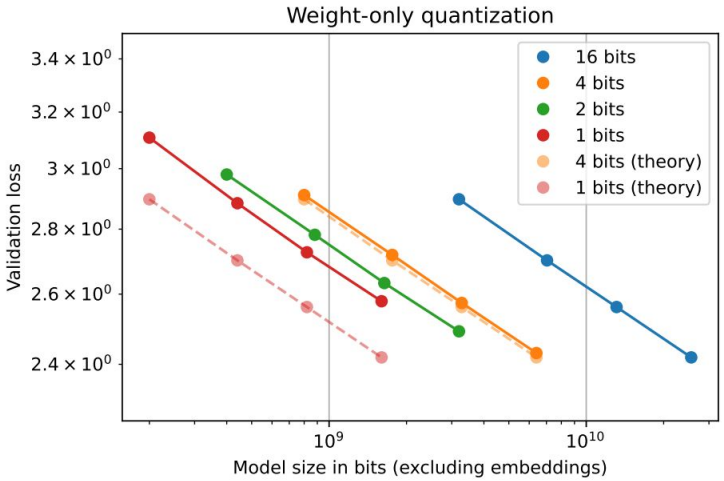


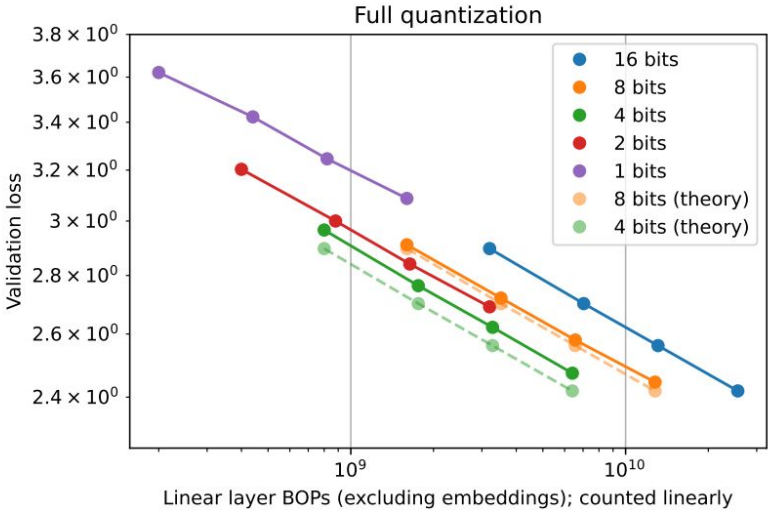
Google Research

Compressed Scaling Laws

- Effect of compression methods can be characterized using a effective compression ratio.
- It shifts the scaling curves

$$L(N, D, C) = \frac{a}{(N \cdot \mathrm{eff}(C))^b} + \frac{c}{D^d} + e, \qquad (1)$$





https://arxiv.org/pdf/2502.16440

Thank you.

Google DeepMind

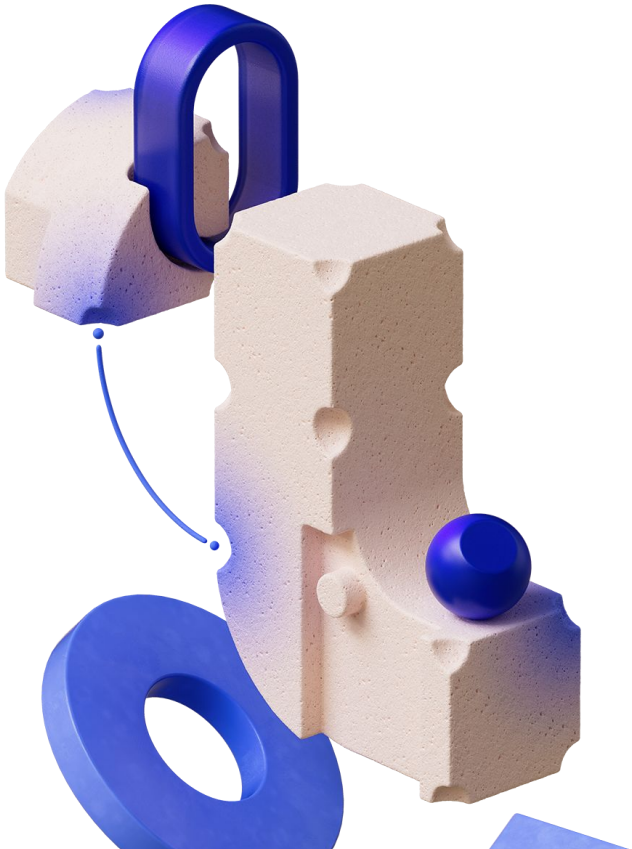
Sparsity/Quant Lab

TODO(add structure and WiP colab)

Todos:

1 - At tensor level, implement n-bit quantization.
2 - At tensor level, implement sparsification with some criterion (weight magnitude below some fixed threshold, smallest k out of n weights get set to 0, with ratio k/n constant across tensors in model?)
3 - At model level, n-bit quantization of trained model
4 - At model level, sparsification of trained model
5 - Plotting code for performance as function of number of bits / sparse fraction
6 - Generic Straight-Through-Estimator implementation for both quantization and sparsification
7 - full finetuning code using jaxpruner
8 - layer-wise error minimization code using jaxpruner (here you just meant training sparse/quantized layers to recover the outputs of the non-perturbed network at the given layer, correct? With a MSE objective?)

Google DeepMind



Parameter Efficient Fine-tuning

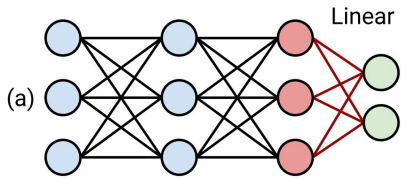
Utku Evci
Research Scientist

Transfer Learning for SOTA

- Train on large *upstream* data set, fine tune on smaller *downstream* data set,
- Unsupervised / supervised pre-training is a popular recipe.
  - **Language** (BERT, GPT-3), **Vision** (CLIP, VIT), **Speech** (wav2vec), **RL**?
- Why not train downstream data set from scratch?
  - Slower convergence
  - Worse generalization

Common Transfer Learning Recipes

- **LINEAR**: Only train a new classification head
  - Cheap to run and store
  - Suboptimal performance



- **FINE-TUNING**: Pretrained feature extractor is tuned together with the head
  - High cost of running and storing for each task.
  - Better performance

Cost of FullFinetuning vs Linear

- Full Fine Tuning
  - Backpropagation requires 2x FLOPs of Forward.
  - You also need to store activations. Can be huge for Transformers!
  - You often also need to store optimization state, which can be 2x of total parameter count.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{i+1}} \frac{\partial \mathbf{a}_{i+1}}{\partial \mathbf{a}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{i+1}} \mathbf{W}^T, \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{a}_i^T \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{i+1}},$$

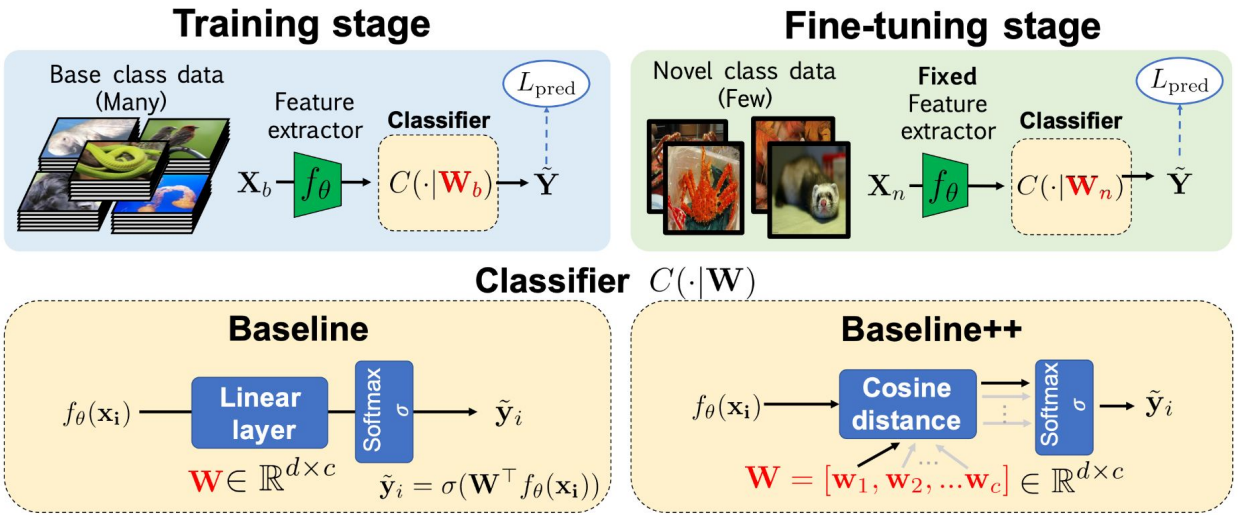
- Linear
  - Backprop is a single matmul.
  - You don't need to store intermediate activations.
  - You can cache data as features (useful when doing epochs).

TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning, Han Cai, Chuang Gan, Ligeng Zhu, Song Han

Google Research

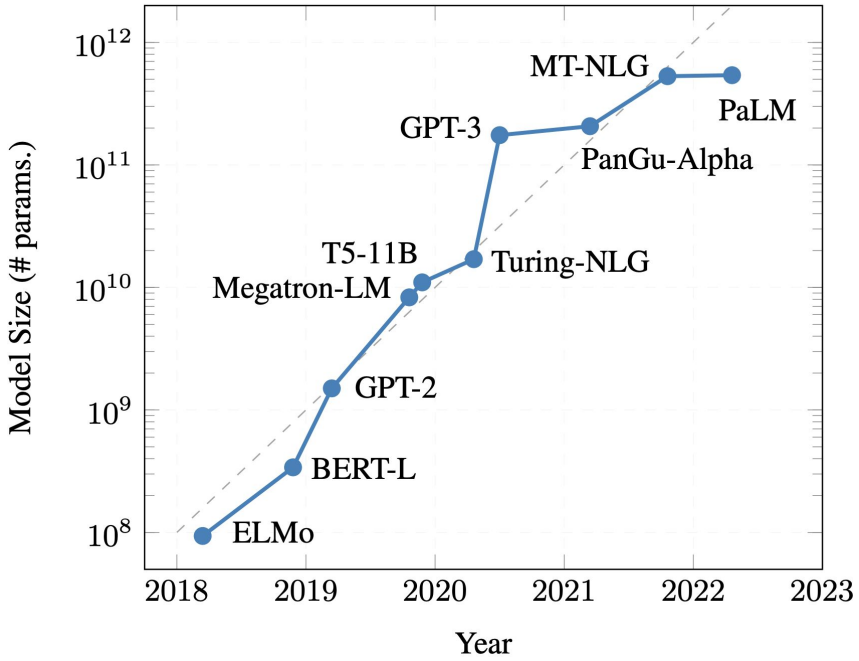
Transfer Learning - Linear Probe

- Train on all data, adapt using few samples.
- #classes might be different. So you need new classifier (last layer).
- You can train a linear projection or Cosine classifier.



https://arxiv.org/pdf/1904.04232

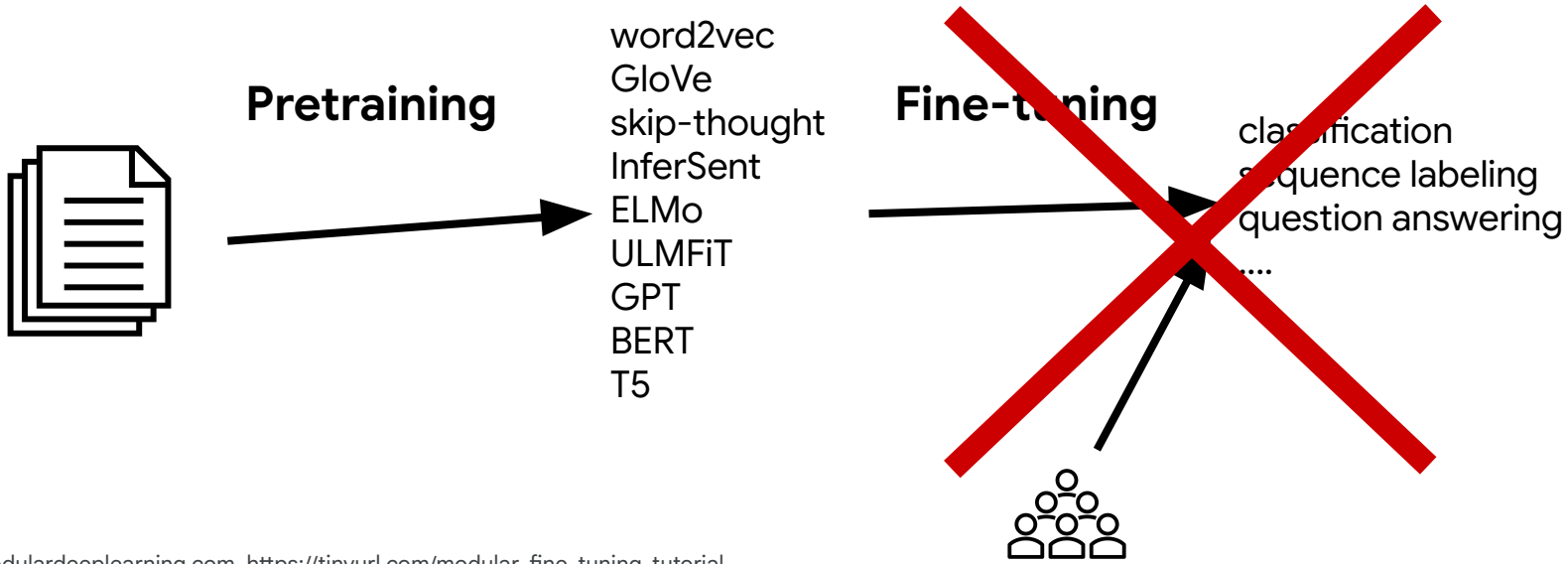
State-of-the-art NLP Models Are Getting Ever Larger



Evolution of the size of large pre-trained models [Treviso et al., 2022]

Transfer Learning in the Era of Large Models

- With increasing model size, fine-tuning becomes increasingly expensive
- The standard transfer learning formula breaks down

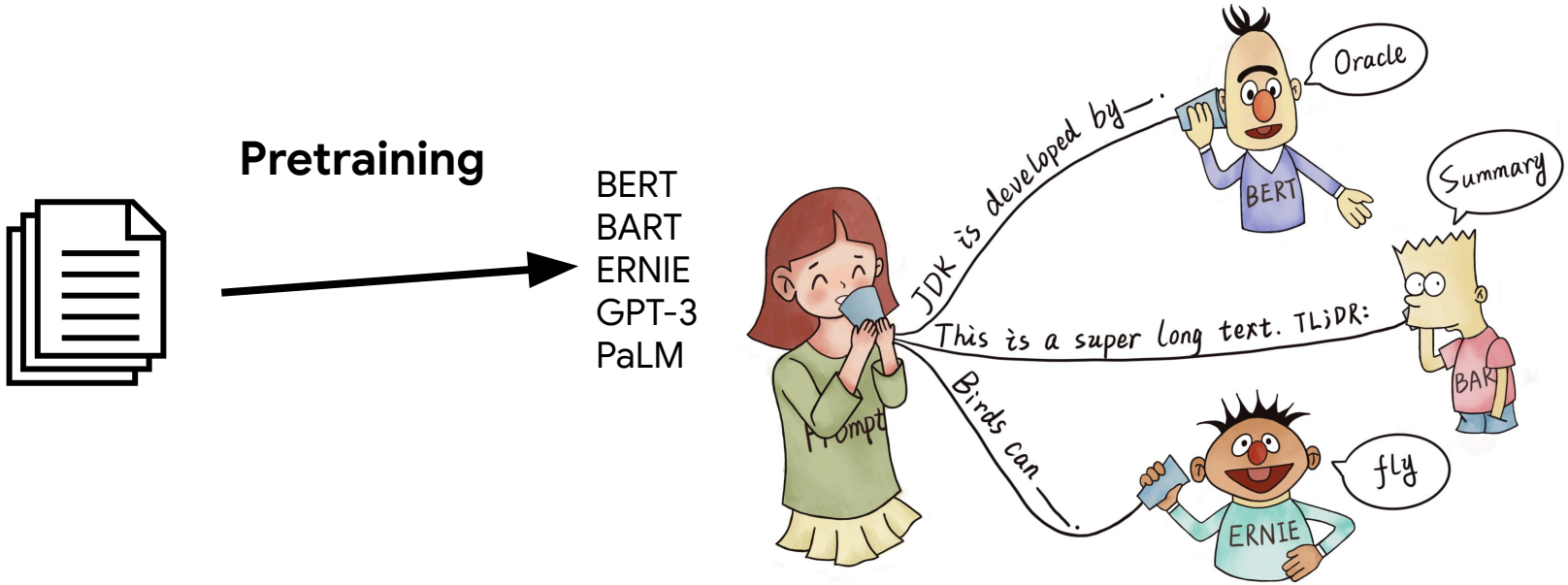


modulardeeplearning.com, https://tinyurl.com/modular-fine-tuning-tutorial

Credit: NAACL 2019 Transfer learning tutorial

Transfer Learning in the Era of Large Models

- In-context learning has mostly replaced fine-tuning for large models



modulardeeplearning.com, https://tinyurl.com/modular-fine-tuning-tutorial

Credit: Liu et al. (2021)

Google Research

Prompt Tuning

- Find best way to feed a dataset/task to LLMs
- Discrete Optimization
- Trial & Error

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1  Translate English to French:      ← task description
2  cheese =>                         ← prompt
```

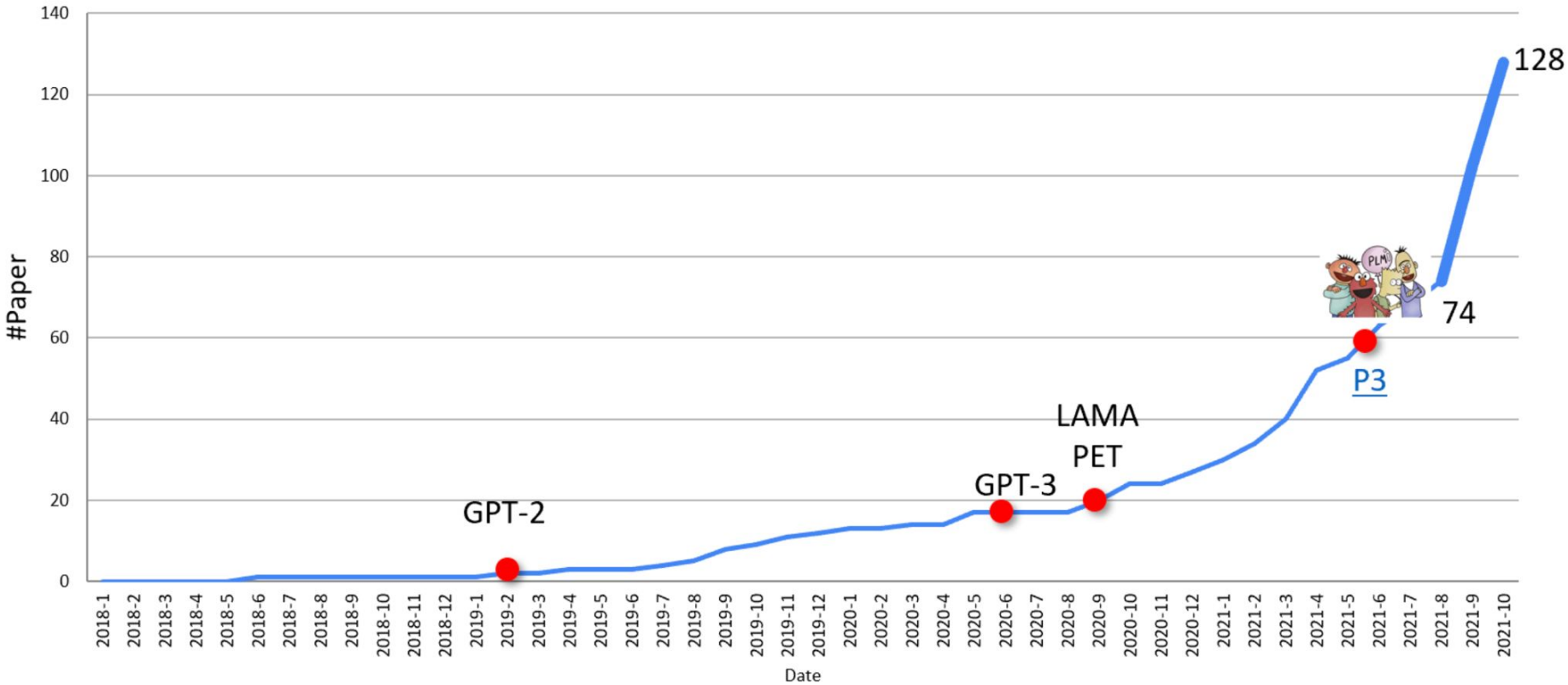
One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:      ← task description
2  sea otter => loutre de mer        ← example
3  cheese =>                         ← prompt
```

https://www.determined.ai/blog/llm-prompting, https://arxiv.org/pdf/2406.06608

Prompt-based Learning has Taken NLP by Storm

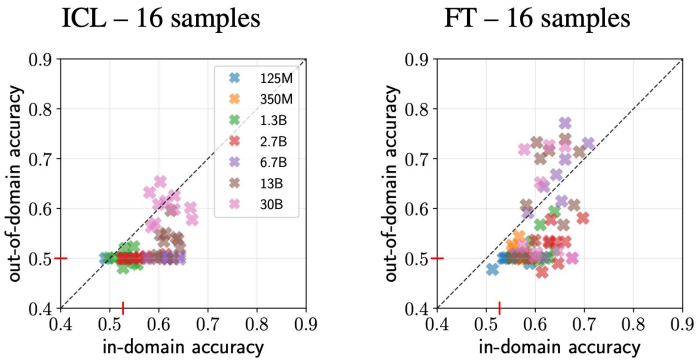


modulardeeplearning.com, https://tinyurl.com/modular-fine-tuning-tutorial

Credit: http://pretrain.nlpedia.ai/

Downsides of Prompt-based Learning

1. **Inefficiency:** The prompt needs to be processed *every time* the model makes a prediction.
2. **Poor performance**: Prompting generally performs worse than fine-tuning [Brown et al., 2020].
3. **Sensitivity** to the wording of the prompt [Webson & Pavlick, 2022], order of examples [Zhao et al., 2021; Lu et al., 2022], etc.
4. **Lack of clarity** regarding what the model learns from the prompt. Even random labels work [Min et al., 2022]!

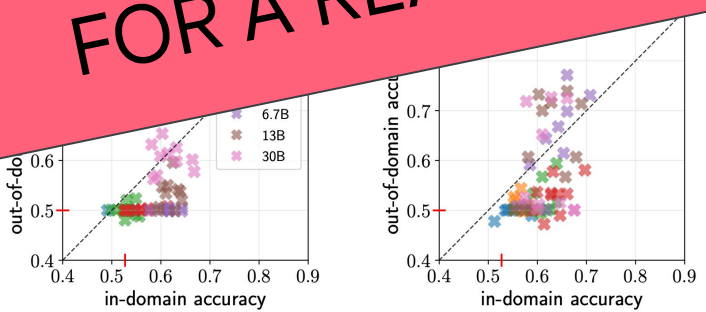


modulardeeplearning.com, https://tinyurl.com/modular-fine-tuning-tutorial
Few-shot Fine-tuning vs. In-context Learning: https://aclanthology.org/2023.findings-acl.779.pdf

Downsides of Prompt-based Learning

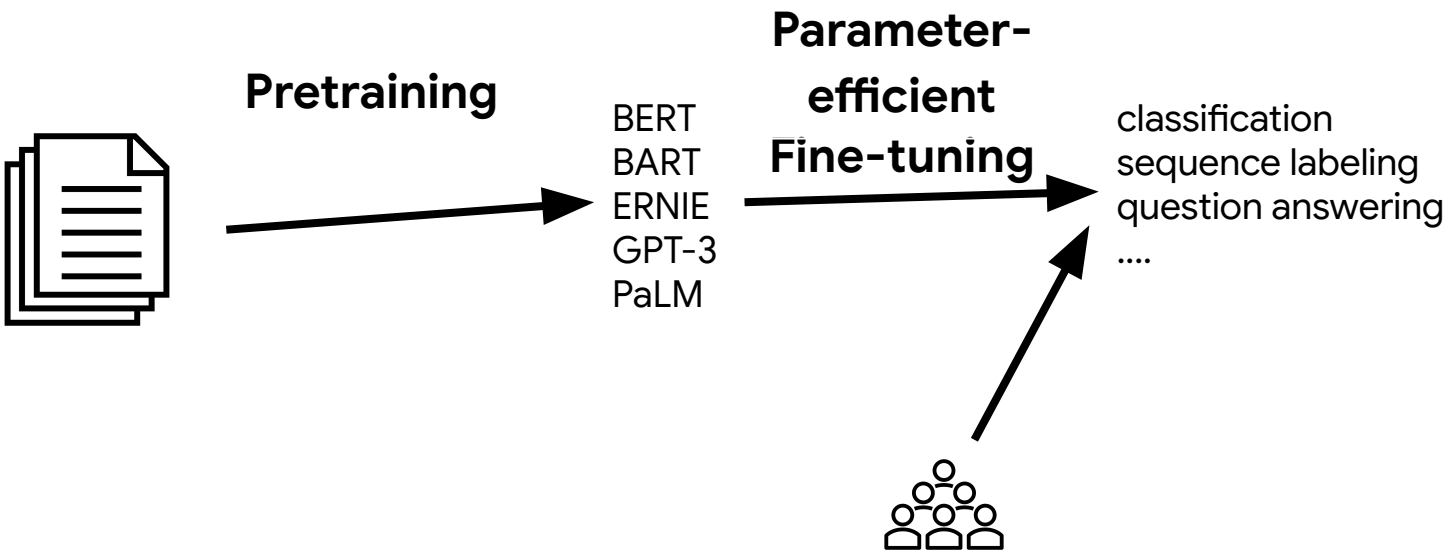
1. **Inefficiency:** The prompt needs to be processed *every time* the mod
2. **Poor performance**: Prompting gener
3. **Sensitivity** t t al., 2022], etc.
4. **Lack of cla**

WE DISCOVERED
GRADIENT DESCENT
FOR A REASON

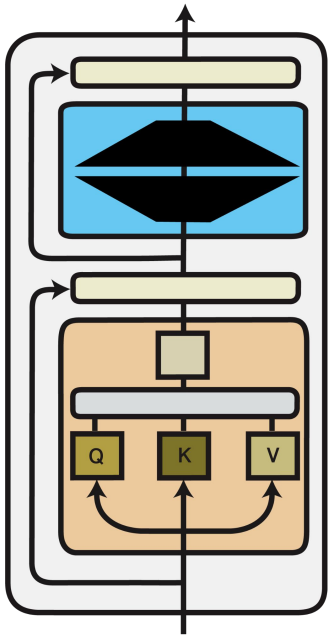


modulardeeplearning.com, https://tinyurl.com/modular-fine-tuning-tutorial
Few-shot Fine-tuning vs. In-context Learning: https://aclanthology.org/2023.findings-acl.779.pdf

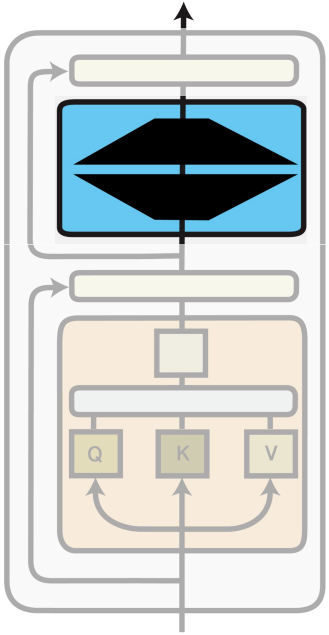
From Fine-tuning to Parameter-efficient Fine-tuning



From Fine-tuning to Parameter-efficient Fine-tuning



Full Fine-tuning
Update **all model parameters**

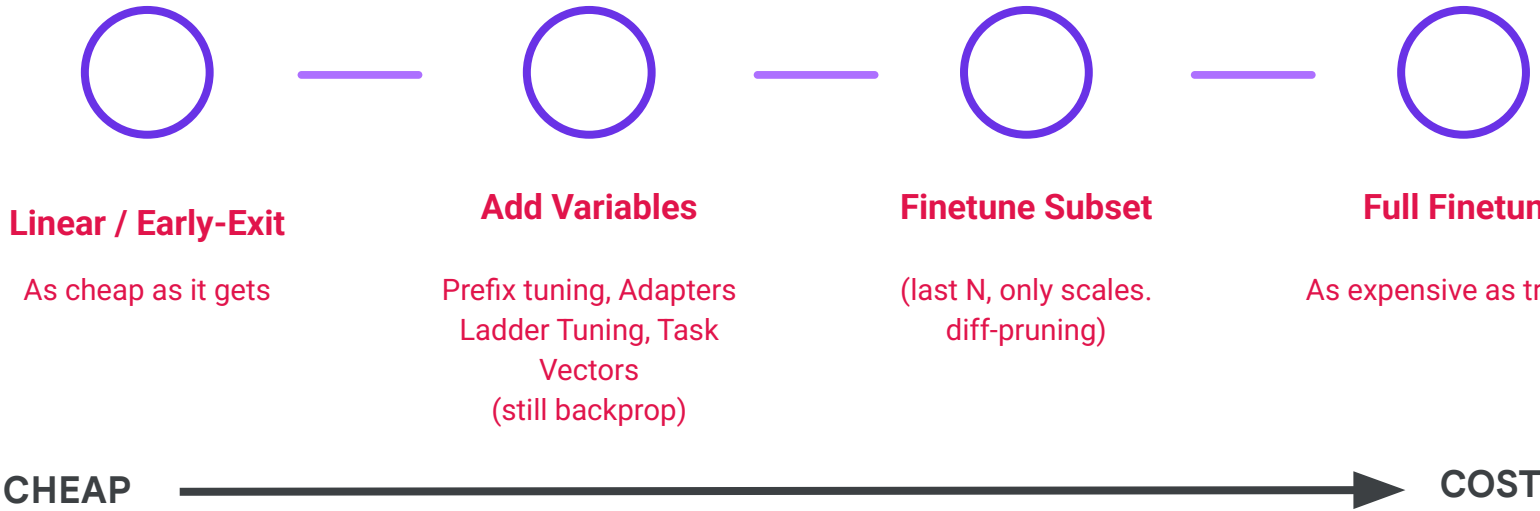


Parameter-efficient Fine-tuning
Update a **small subset** of model parameters

modulardeeplearning.com, https://tinyurl.com/modular-fine-tuning-tutorial

Transfer Learning

- How can we maximize quality while minimize cost*



Transfer Learning

- How can we maximize quality while minimize cost*

Linear / Early-Exit

As cheap as it gets

Add Variables

Prefix tuning, Adapters
Ladder Tuning, LORA
(still backprop)

Finetune Subset

(last N, only scales.
diff-pruning)

Full Finetune

As expensive as training

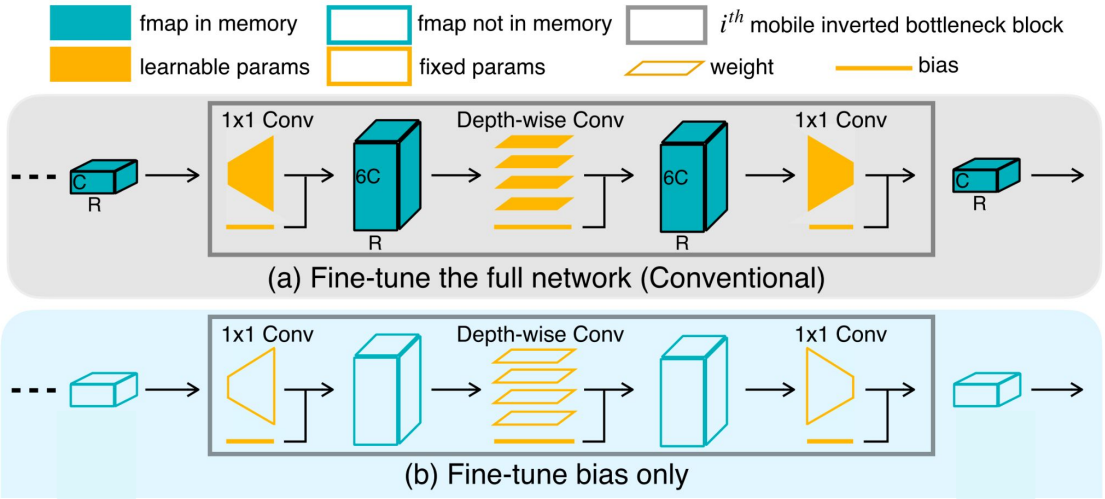
CHEAP → COSTLY

Bias-only Fine-tuning

- A practical choice: updating only biases $\boldsymbol{b}$
- Computing $\nabla \boldsymbol{b}$ does not require storing activations [Cai et al., 2020]!

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{i+1}} \frac{\partial \mathbf{a}_{i+1}}{\partial \mathbf{a}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{i+1}} \mathbf{W}^T, \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{a}_i^T \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{i+1}}, \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{i+1}}.$$

- In NLP, BitFit [Ben-Zaken et al., 2022] implements the same approach
- Query and second MLP layer biases are most important!



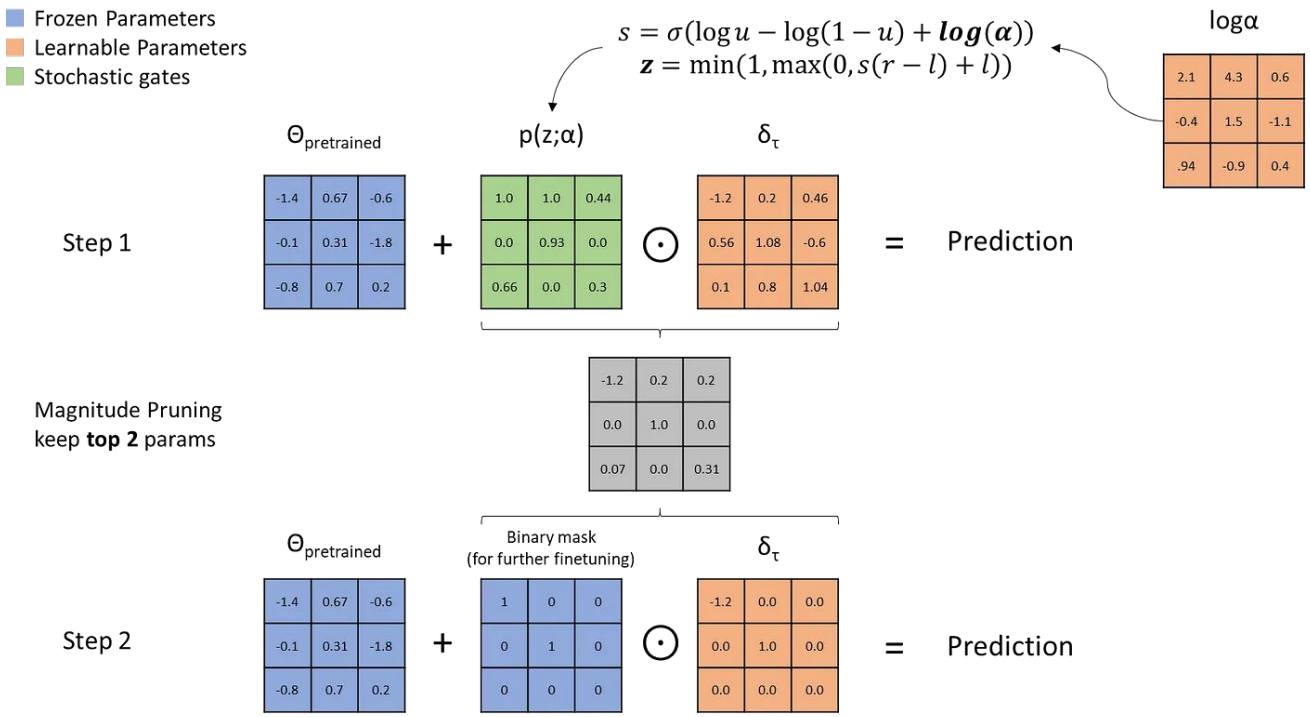
Diff Pruning

- For a model with $f'_\theta = f_{\theta+\phi}$, we can perform pruning only based on the magnitude of the module parameters ϕ rather than the updated parameters $\theta + \phi$
- Diff pruning [Guo et al., 2021] prunes the module parameters via magnitude pruning to make them sparse

$$\min_{\boldsymbol{\delta}_\tau} \; L(\mathcal{D}_\tau, f_\tau, \boldsymbol{\theta} + \boldsymbol{\delta}_\tau) + \lambda R(\boldsymbol{\theta} + \boldsymbol{\delta}_\tau),$$

$$R(\boldsymbol{\theta} + \boldsymbol{\delta}_\tau) = \|\boldsymbol{\delta}_\tau\|_0 = \sum_{i=1}^{d} \mathbb{1}\{\boldsymbol{\delta}_{\tau,i} \neq 0\}.$$

DiffPruning



https://medium.com/@lukas.hauzenberger/an-practical-introduction-to-diff-pruning-for-bert-4278ee4be750

DiffPruning

Diff vector target sparsity	QNLI	SST-2	MNLI_m	MNLI_{mm}	CoLA	MRPC	STS-B	RTE	QQP	Avg
0.10%	92.7	93.3	85.6	85.9	58.0	87.4	86.3	68.6	85.2	82.5
0.25%	93.2	94.2	86.2	86.5	63.3	90.9	88.4	71.5	86.1	84.5
0.50%	93.4	94.2	86.4	86.9	63.5	91.3	89.5	71.5	86.6	84.8
1.00%	93.3	94.2	86.4	87.0	66.3	91.4	89.9	71.1	86.6	85.1
100%	93.5	94.1	86.5	87.1	62.8	91.9	89.8	71.8	87.6	85.0

Table 2: Structured diff pruning results on the validation set with different target sparsity rates.

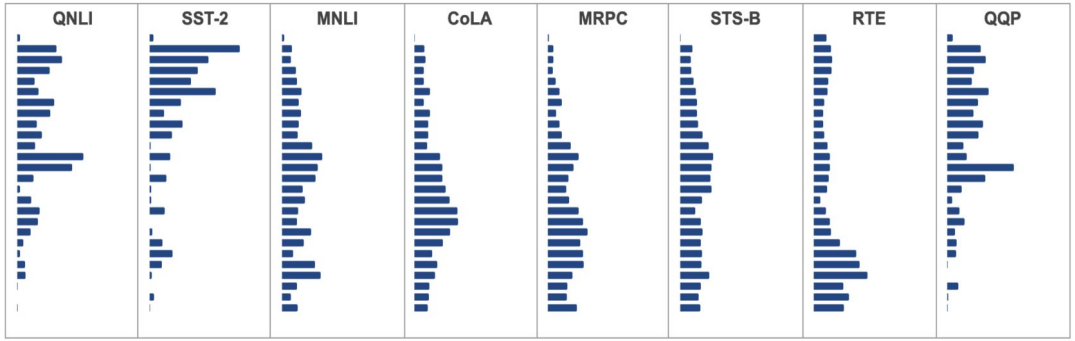
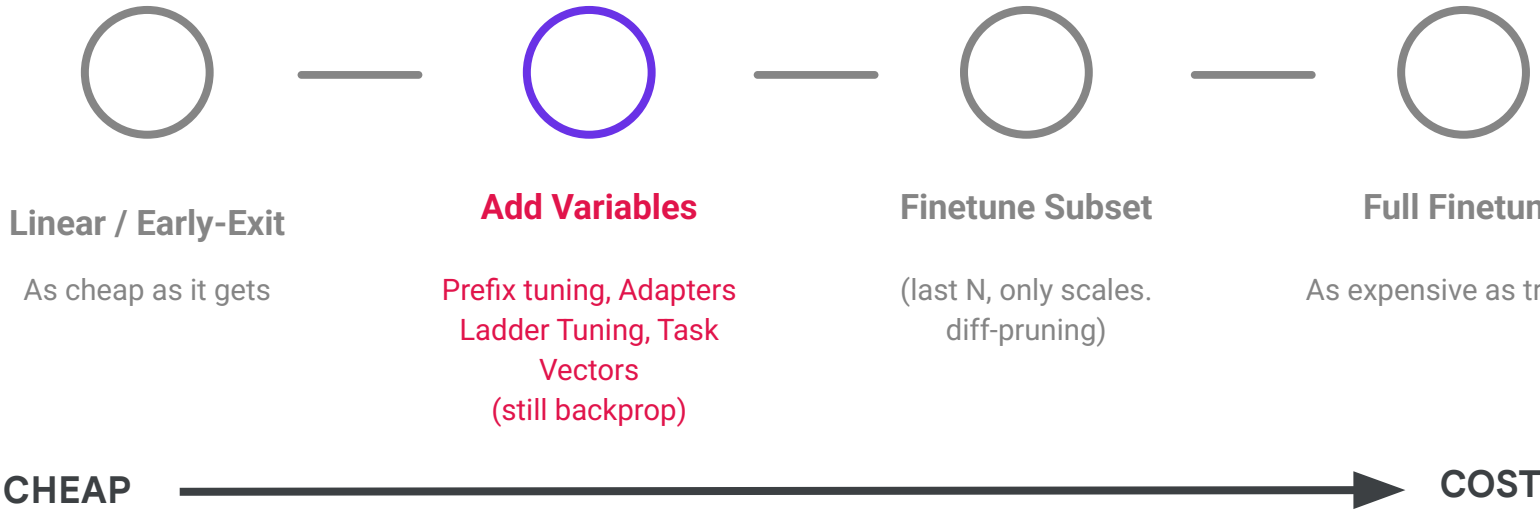


Figure 2: Percentage of modified parameters attributable to each layer for different tasks at 0.5% target sparsity. The layers are ordered from earlier to later (i.e. the embedding layer is shown at the top). The x-axis for each plot goes from 0% to 20%.

https://huggingface.co/docs/peft/en/conceptual_guides/prompting

Transfer Learning

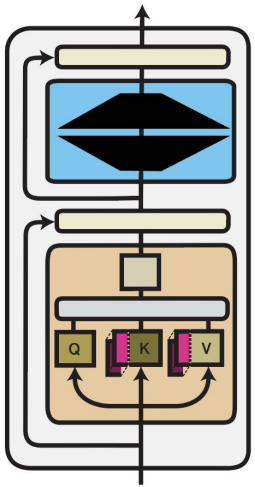
- How can we maximize quality while minimize cost*



Input Composition

- Augment a model's input by augmenting it with a learnable parameter vector ϕ_i:

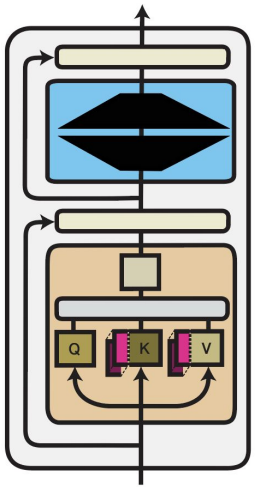
$$f_i'(\boldsymbol{x}) = f_{\theta_i}([\phi_i, \boldsymbol{x}])$$



Input Composition and Prompting

- Standard prompting can be seen as finding a discrete text prompt that—when embedded using the model's embedding layer—yields ϕ_i

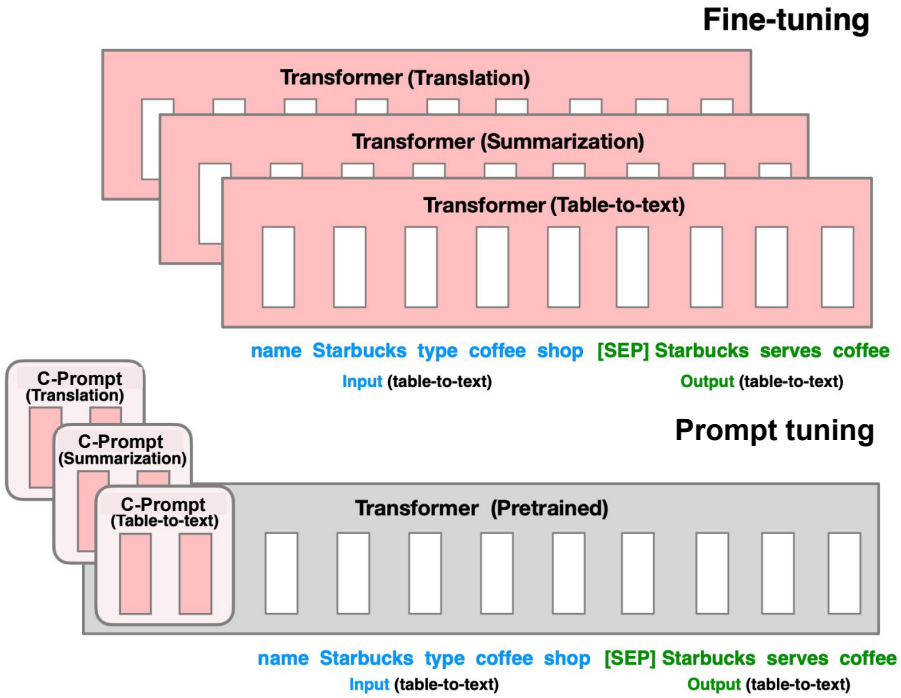
- However, models are sensitive to the formulation of the prompt [Webson & Pavlick, 2022] and to the order of examples [Zhao et al., 2021; Lu et al., 2022]



Prompt Tuning

- Instead, we can directly learn a continuous prompt ϕ that is prepended to the input [Liu et al., 2021; Hambardzumyan et al., 2021; Lester et al., 2021]

- ϕ is typically a matrix consisting of a sequence of continuous prompt embeddings



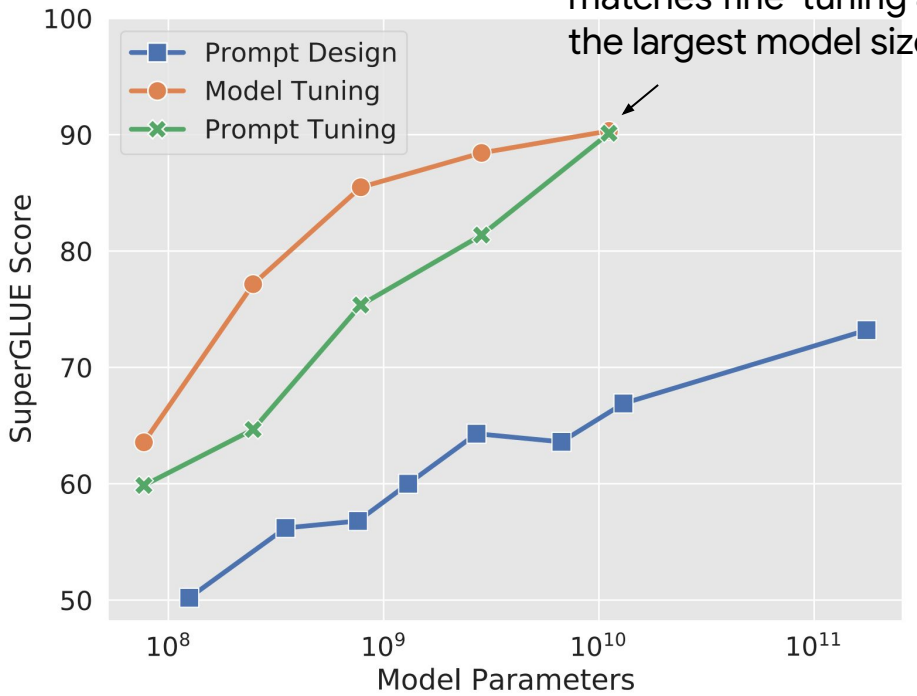
Fine-tuning vs Prompt tuning
(adapted from [Li & Liang, 2021])

Prompt Tuning Only Works Well at Scale

- Only using trainable parameters at the input layer limits capacity for adaptation

→ Prompt tuning performs poorly at smaller model sizes and on harder tasks [Mahabadi et al., 2021; Liu et al., 2022]

Prompt tuning only matches fine-tuning at the largest model size

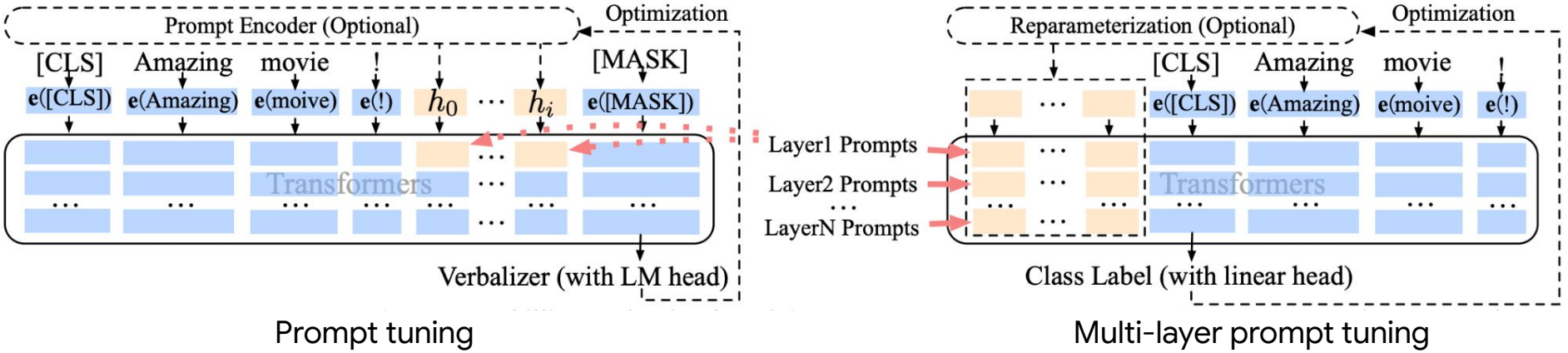


Prompt tuning vs standard fine-tuning and prompt design across T5 models of different sizes [Lester et al., 2021]

Multi-Layer Prompt Tuning

- Instead of learning ϕ_i parameters only at the input layer, we can learn them at *every layer* of the model [Li & Liang, 2021; Liu et al., 2022]

- Continuous prompts in later layers are more important

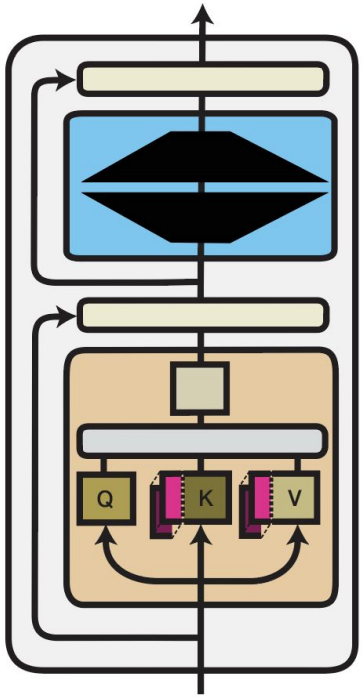


Prompt tuning

Multi-layer prompt tuning

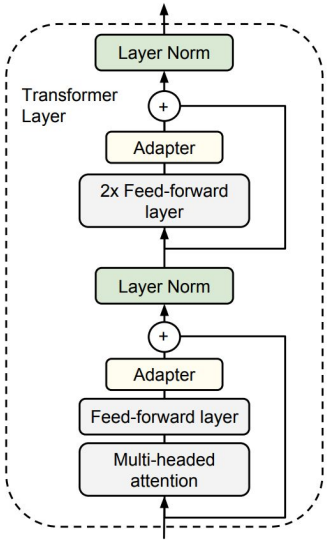
Multi-Layer Prompt Tuning

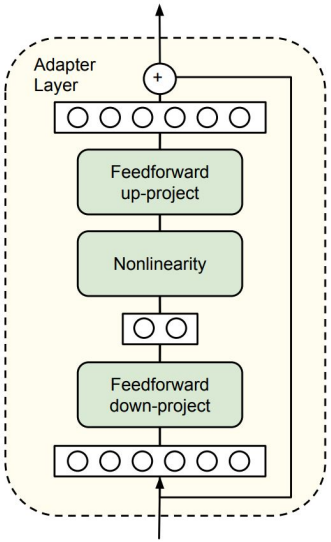
- In practice, continuous prompts ϕ_i are concatenated with the keys and values in the self-attention layer [Li & Liang, 2021]

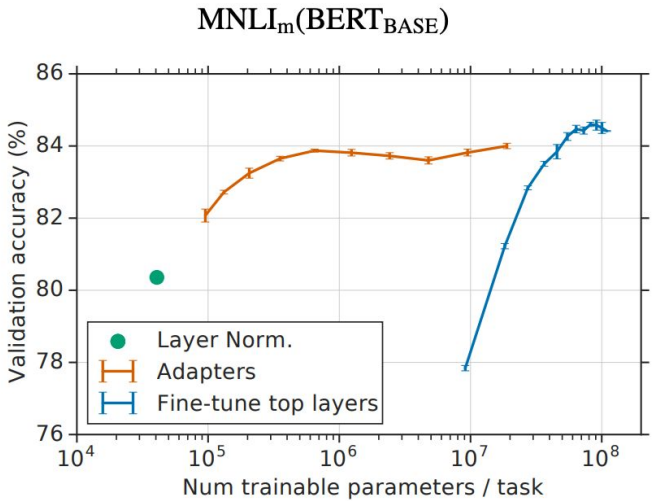


Google Research

Adapters







Dataset	No BERT baseline	BERT$_{BASE}$ Fine-tune	BERT$_{BASE}$ Variable FT	BERT$_{BASE}$ Adapters
Average	72.7	73.7	74.0	73.3
Total number of params	—	17×	9.9×	1.19×
Trained params/task	—	100%	52.9%	1.14%

https://arxiv.org/pdf/1902.00751

Compact Adapter (Compacter)

- Compacter [Mahabadi et al., 2021] reparameterizes the W matrices in the adapter as:

$$W = \sum_{i=1}^{n} A_i \otimes B_i \quad \text{where } A_i \in \mathbb{R}^{n \times n}$$

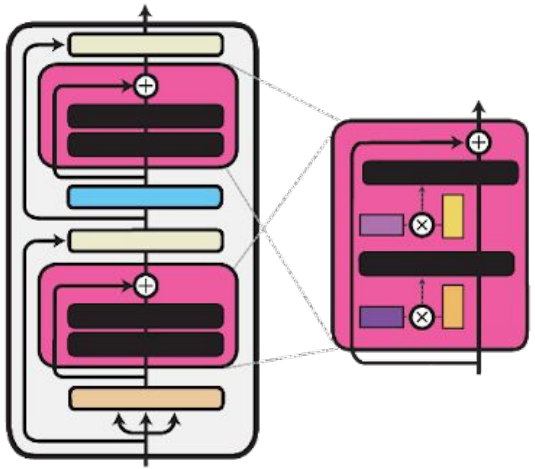
Kronecker product

Shared between all layers

A hyper-parameter (between 4–12)

$$\text{and } B_i = \boldsymbol{s}_i \boldsymbol{t}_i^\top \in \mathbb{R}^{\frac{k}{n} \times \frac{d}{n}}.$$

A low-rank matrix



- Compacter reduces adapter parameters by a factor of 10 and achieves similar or better performance

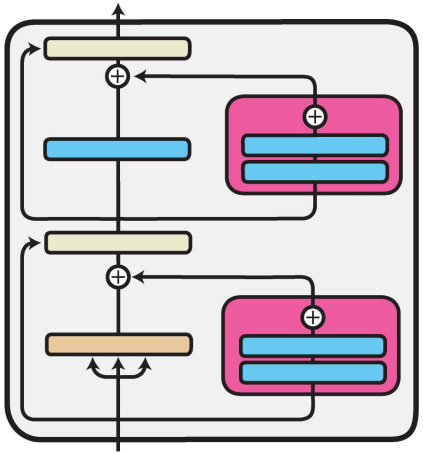
Sequential and Parallel Adapters

- Adapters can be routed sequentially or in parallel
- Sequential adapters are inserted between functions:

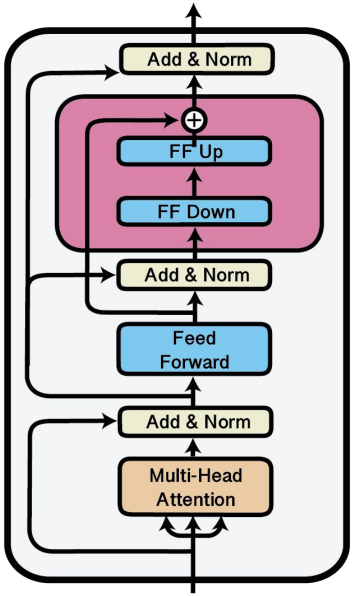
$$f_i'(\boldsymbol{x}) = f_{\phi_i}(f_{\theta_i}(\boldsymbol{x}))$$

- Parallel adapters are applied in parallel:

$$f_i'(\boldsymbol{x}) = f_{\theta_i}(\boldsymbol{x}) + f_{\phi_i}(\boldsymbol{x})$$



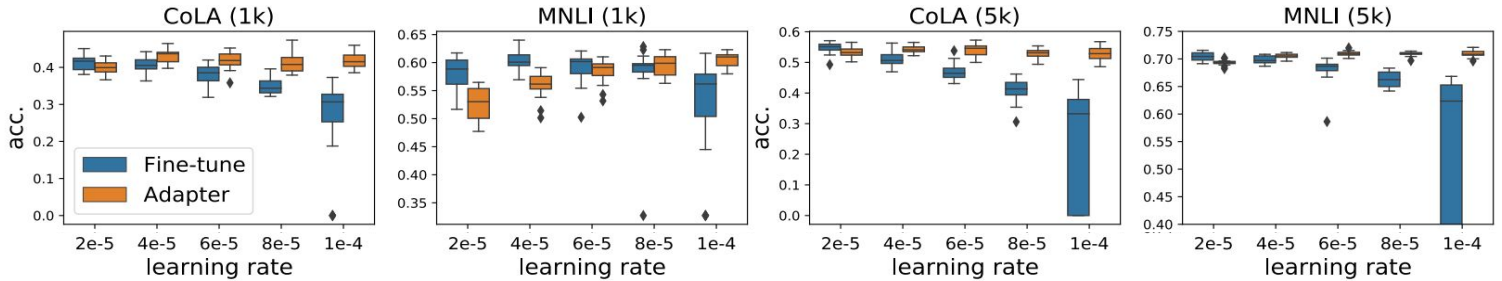
Two parallel adapters [Stickland & Murray, 2019]



A sequential adapter [Houlsby et al., 2019]

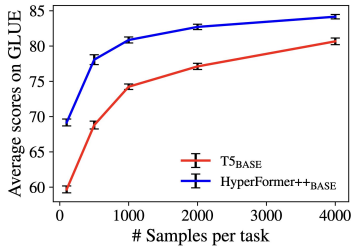
Benefits of Adapters

- Increased robustness [He et al., 2021; Han et al., 2021]



BERT test performance distributions over 20 runs with different learning rates [He et al., 2021]

- Increased sample efficiency



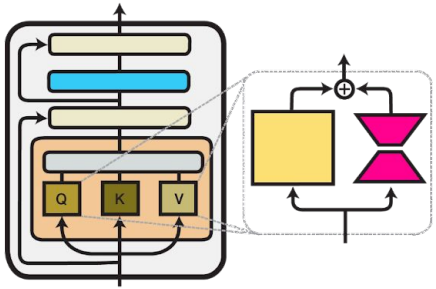
Results on GLUE with different numbers of training samples per task [Mahabadi et al., 2021]

Rescaling

- Instead of learning a function, even rescaling via element-wise multiplication can be powerful:

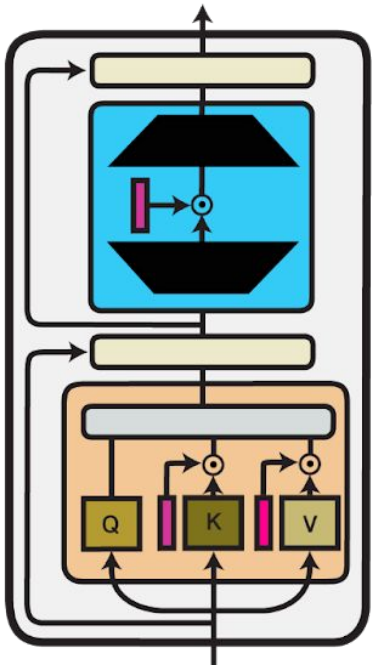
$$f_i'(\boldsymbol{x}) = f_{\theta_i}(\boldsymbol{x}) \circ \phi_i$$

- Commonly applied to normalization parameters, e.g., batch normalization parameters in CV [Bilen et al., 2017], layer normalization in NLP [Houlsby et al., 2019]
- Allows the model to select parameters that are more and less important for a given task
- Compatible with other methods such as LoRA, which includes a tunable scalar parameter



IA^3

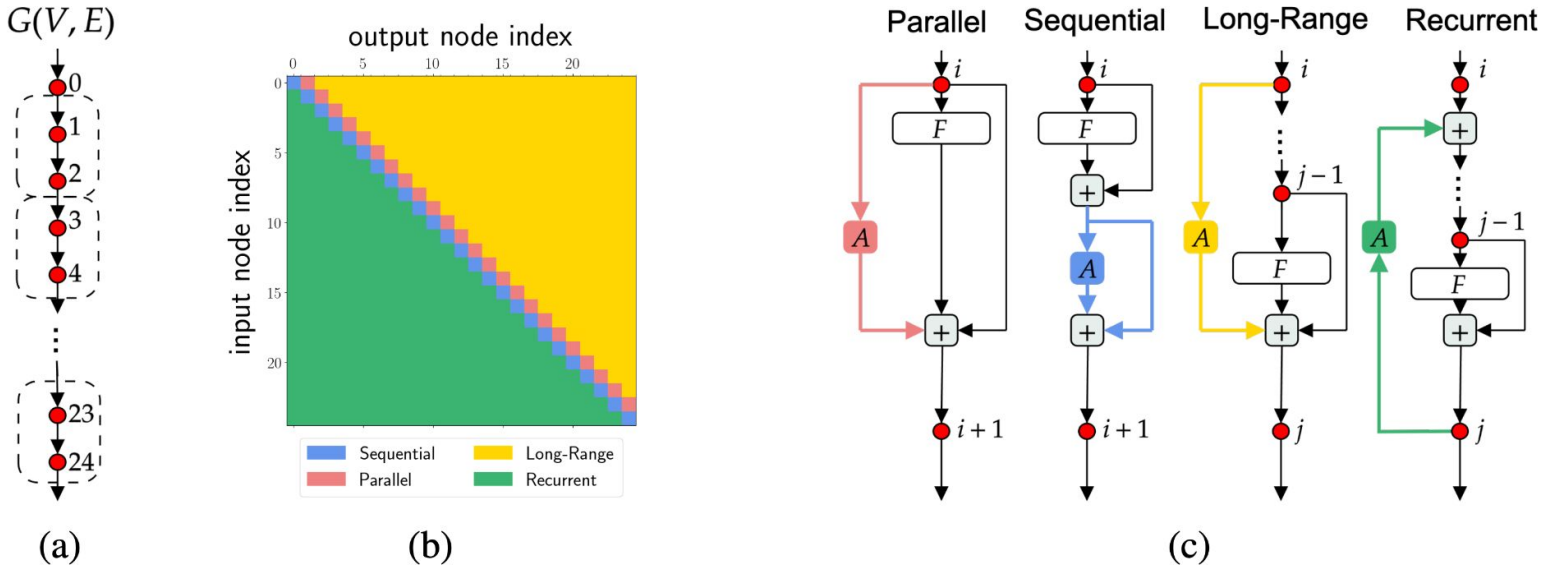
- IA^3 [Liu et al., 2022] multiplies learned vectors with the keys and values in self-attention and the intermediate activations in the feed-forward network of a Transformer



IA^3 [Liu et al., 2022] in the Transformer model

Beyond Sequential Parallel

- You can adapters across multiple layers or even backwards.



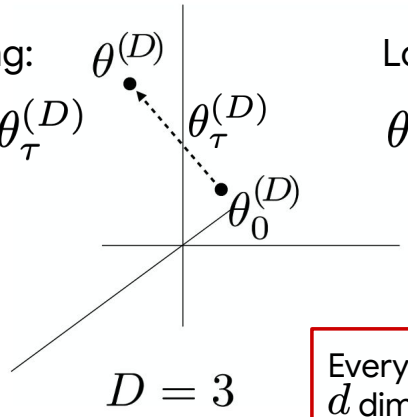
Towards Optimal Adapter Placement for Efficient Transfer Learning // https://arxiv.org/pdf/2410.15858

Low-rank Composition

- Another useful inductive bias: module parameters ϕ should lie in a low-dimensional space
- Li et al. [2018] show that models can be optimized in a low-dimensional, randomly oriented subspace rather than the full parameter space

Standard fine-tuning:

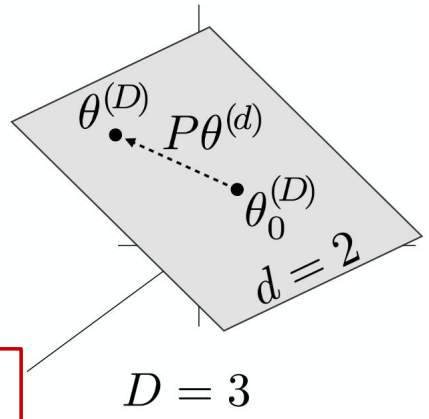
$$\theta^{(D)} = \theta_0^{(D)} + \theta_\tau^{(D)}$$



Low-rank fine-tuning:

$$\theta^{(D)} = \theta_0^{(D)} + P\theta^{(d)}$$

A random $D \times d$ projection matrix



Everything but $\theta^{(d)}$ is fixed. Only d dimensions are optimized.

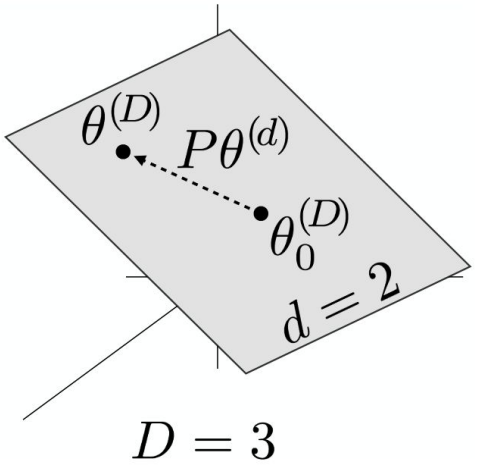
Low-rank Fine-tuning

- In our notation, low-rank fine-tuning takes the form: $f'_\theta = f_{\theta + P\phi}$ where $P \in \mathbb{R}^{D \times d}$.
- With a dense matrix $D \times d$, this scales as $\mathcal{O}(Dd)$ in matrix-vector multiply time and storage space
- For large pre-trained models, we need a more efficient solution

Low-rank fine-tuning:

$$\theta^{(D)} = \theta_0^{(D)} + P\theta^{(d)}$$

A random $D \times d$ projection matrix



Low-rank Fine-tuning via Matrix Factorization

- For the multiplication, we do not need to explicitly store P.

- Instead, we can decompose the matrix.
- A classic choice: Fastfood transform [Le et al., 2013], which requires only $\mathcal{O}(D)$ space and $\mathcal{O}(D \log d)$ time.

- Factorizes P via a sequence of random matrices with special properties:

$$P = HG\Pi HB$$

B: Random diagonal matrix with equal probability ± 1 entries

Π: Random permutation matrix

G: Random diagonal matrix with independent standard normal entries

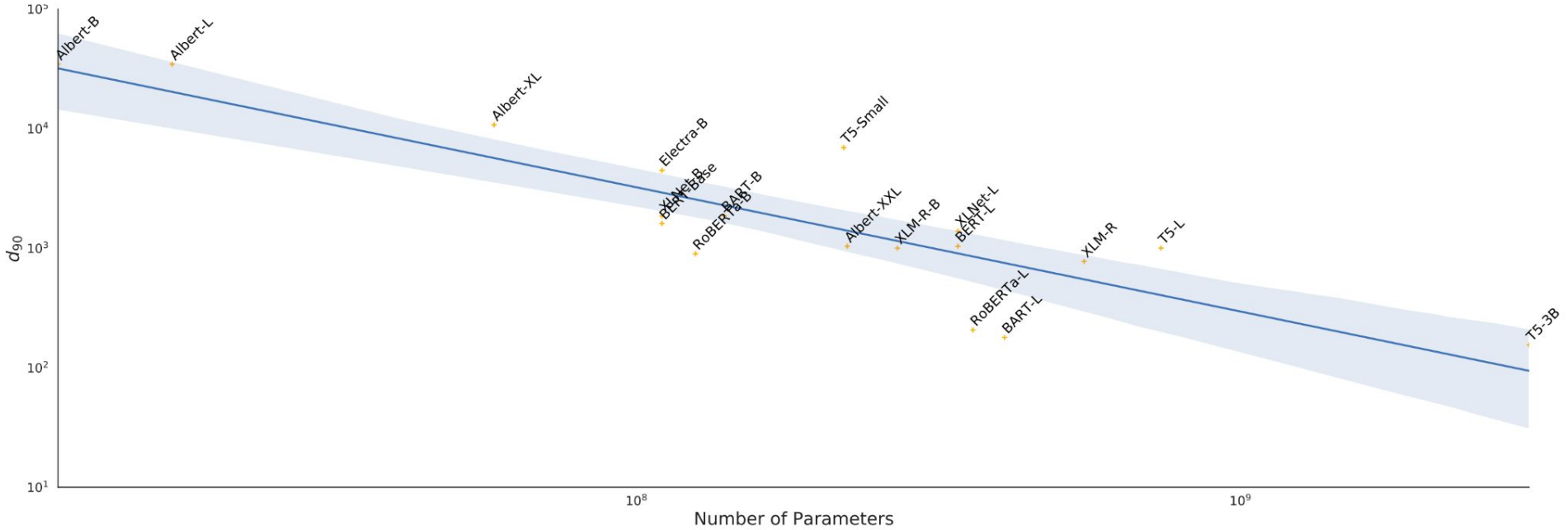
H: Hadamard matrix

Intrinsic Dimensionality

- Li et al. [2018] refer to the minimum d where a model achieves within 90% of the full-parameter model performance, d_{90} as the intrinsic dimensionality of a task
- Aghajanyan et al. [2021] investigate the intrinsic dimensionality of different NLP tasks and pre-trained models

- Observations:
  - Intrinsic dimensionality decreases during pre-training
  - Larger models have lower intrinsic dimensionality

Intrinsic Dimensionality



Intrinsic dimension d_{90} on the MRPC dataset for models of different sizes [Aghajanyan et al., 2021]

Structured Low-rank Methods

- Aghajanyan et al. [2021] also propose a structure-aware version
- Allocate one scalar λ_i per layer to learn layer-wise scaling:

$$\theta_i^{(D)} = \theta_{0,i}^{(D)} + \lambda_i P \theta_i^{(d)}$$

- Improves the intrinsic dimension in general
- However, storing the random matrices still requires a lot of extra space and is slow to train [Mahabadi et al., 2021]

→ We can apply the low-rank constraint only to certain groups

Low-rank Adaptation (LoRA)

- In addition, instead of learning a low-rank factorization via a random matrix P, we can learn the projection matrix directly (if it is small enough)
- LoRA [Hu et al., 2022] learns two low-rank matrices $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$ that are applied to the self-attention weights:

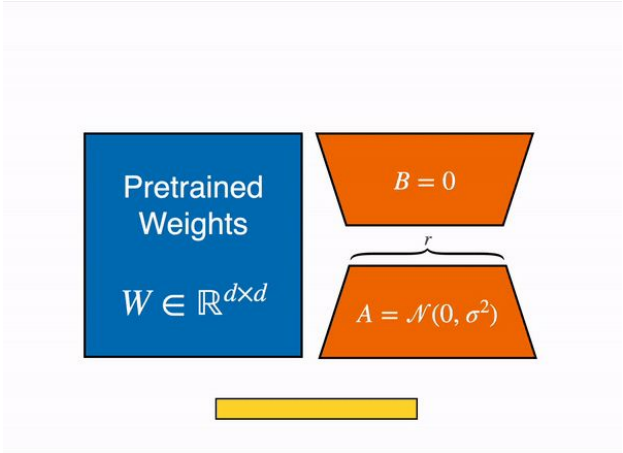
$$h = W_0 x + \Delta W x = W_0 x + BAx$$

Matrix rank | Hidden dimension | Input dimension

- In our notation, this looks like the following:

$$f_i' = f_{\theta_i + \text{vec}(B_i A_i)} \; \forall f_i' \in \mathcal{G}$$

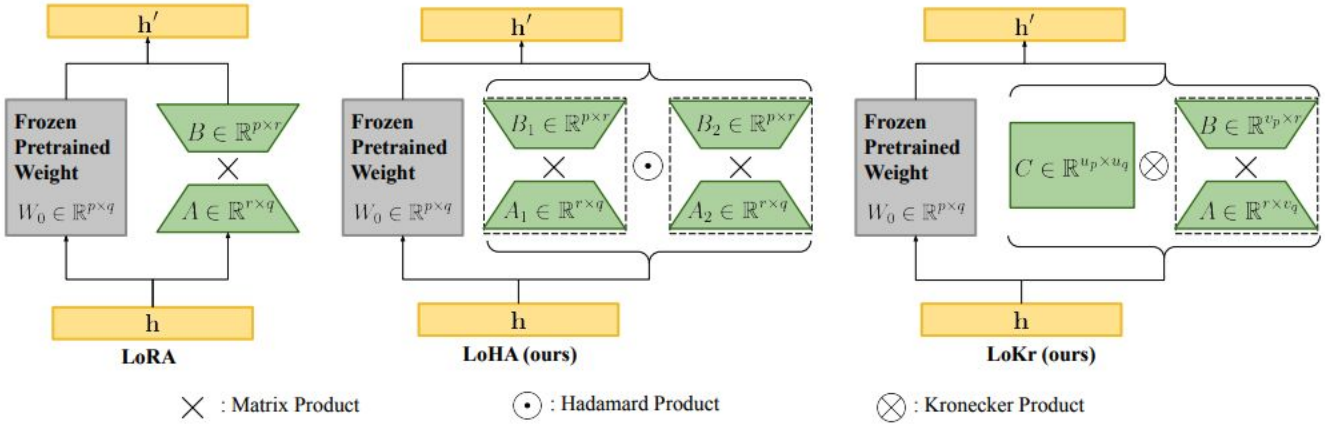
Vectorization



Google Research

LORA

- Simple idea is to approximate the fineuning delta with something cheaper.
- And fold the delta after finetuning into the pretrained weights (no extra compute!)
- So many different variations exist.

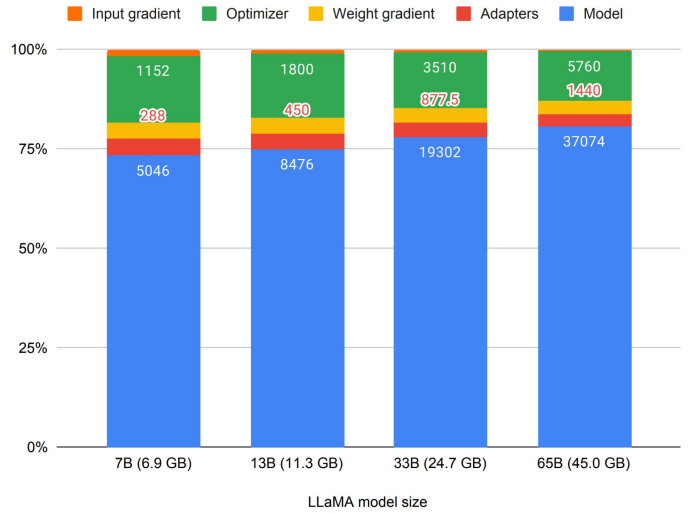


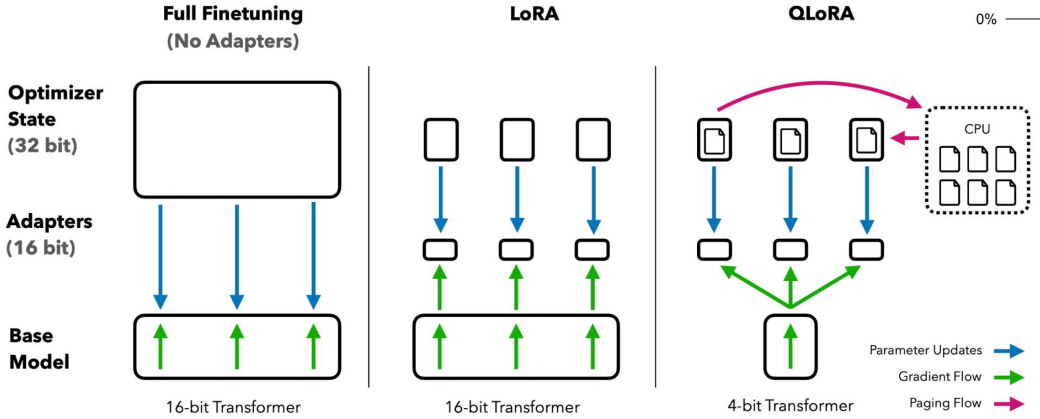
https://huggingface.co/docs/peft/en/conceptual_guides/adapter

Google Research

QLORA

- When you nail PeFT, parameters become the bottleneck.
- Well quantize the params!
- And use LORA to recover quality while finetuning.



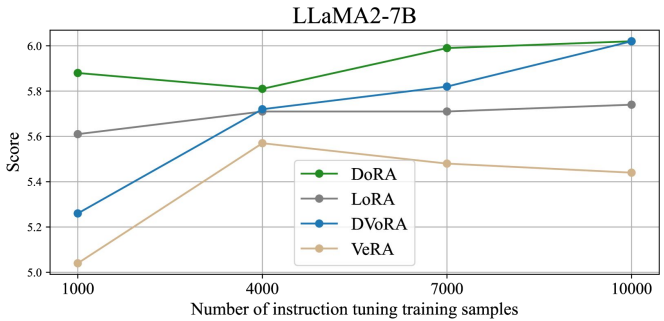


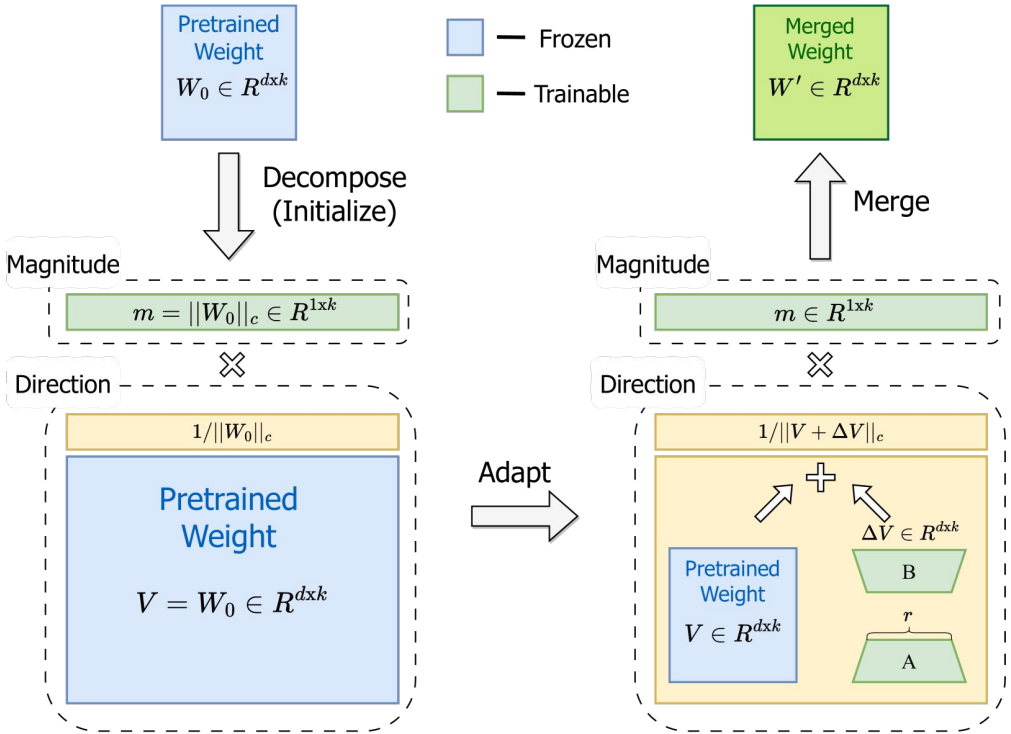
QLORA: Efficient Finetuning of Quantized LLMs https://arxiv.org/pdf/2305.14314

Google Research

DoRA

- Normalize weights each channel to unit norm.
- Train directions and magnitudes separately.

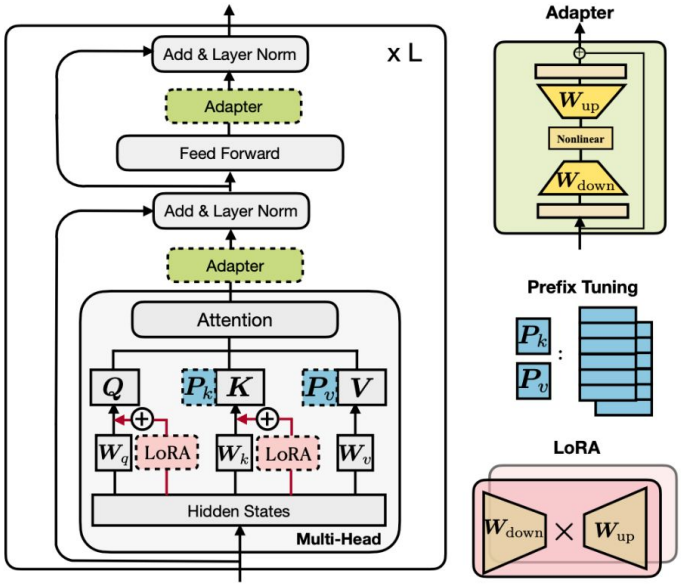


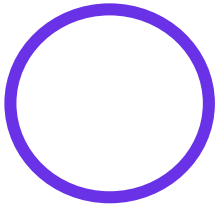


https://arxiv.org/html/2402.09353v6

Google Research

Unified View





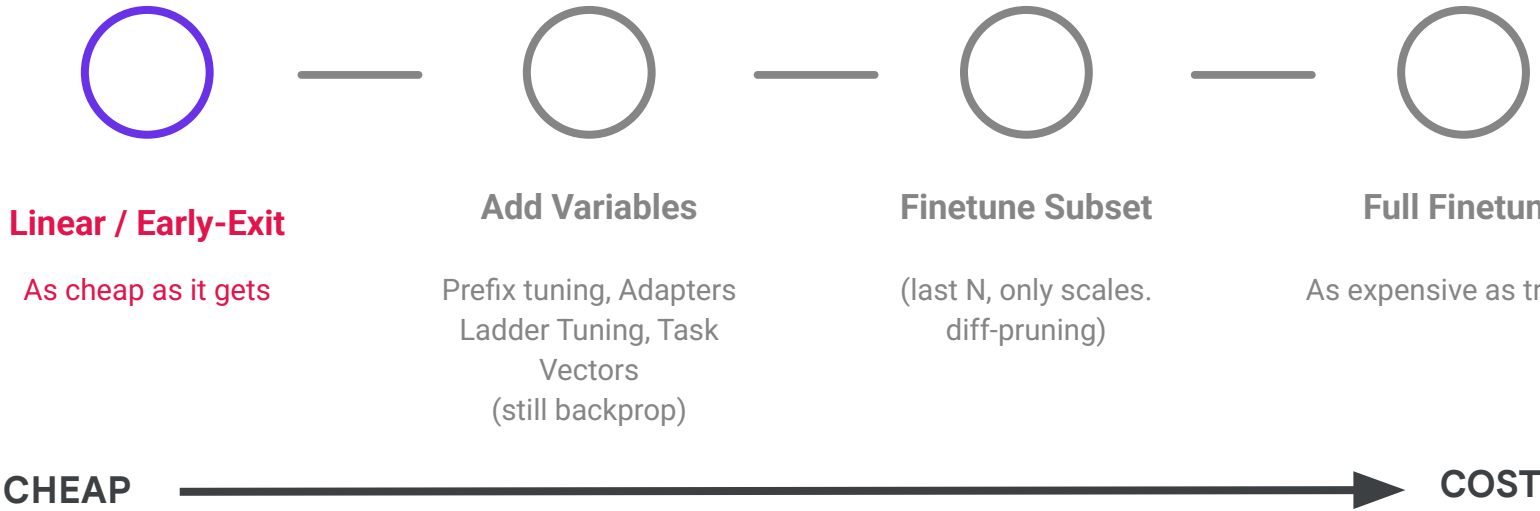
Add Variables

and use
Gradient Descent!

https://arxiv.org/pdf/2110.04366

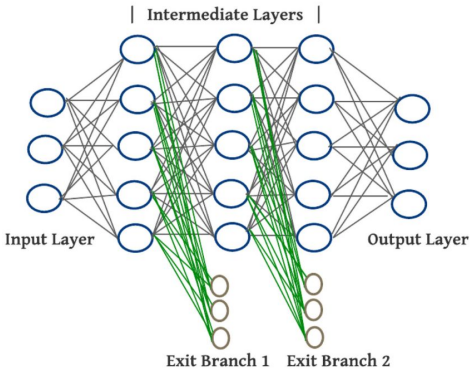
Transfer Learning

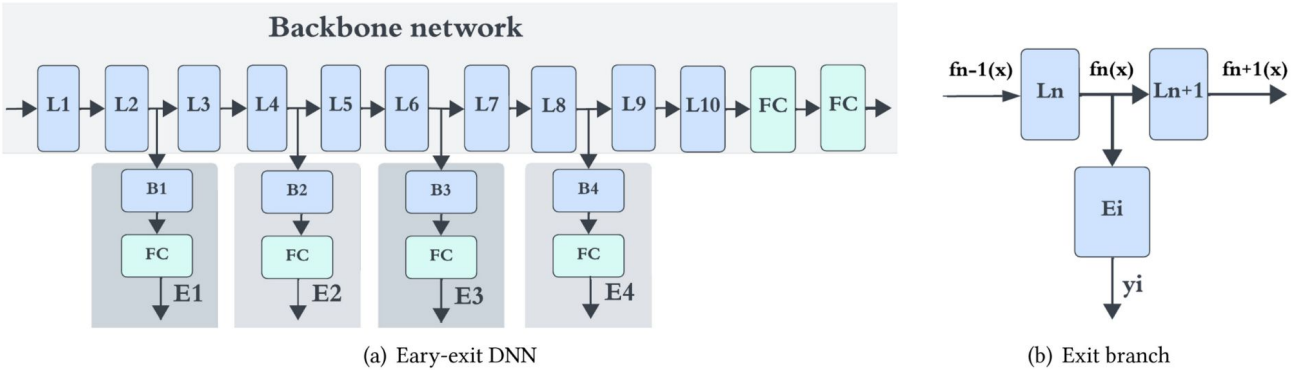
- How can we maximize quality while minimize cost*



Early-Exits

- Can reduce cost of inference and also fine-tuning by truncating the model?
- Similar to Linear Probe and/or tune last-n layers





(a) Eary-exit DNN (b) Exit branch

Fig. 3. (a) Early-exit DNN with four intermediate branches. (b) Illustration of a side branch.

Early-Exit Deep Neural Network - A Comprehensive Survey, https://dl.acm.org/doi/pdf/10.1145/3698767

Head2Toe w/ Group-Lasso

- Given a pretrained NN:

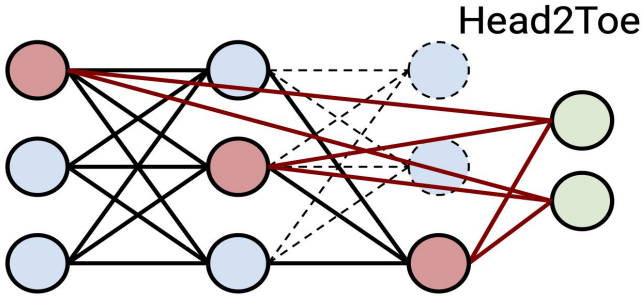
$$\boldsymbol{z}_\ell = \boldsymbol{h}_{\ell-1}\boldsymbol{W}_\ell \quad ; \quad \boldsymbol{h}_\ell = f(\boldsymbol{z}_\ell)$$

$$\boldsymbol{z}'_L = \boldsymbol{h}_{all}\boldsymbol{W}_{all} \quad ; \quad \boldsymbol{h}_{all} = \text{concat}(a_1(\boldsymbol{h}_1), a_2(\boldsymbol{h}_2), ..., a_L(\boldsymbol{h}_L))$$

- We train W_all with group-lasso regularization and select features with highest l2-norm.

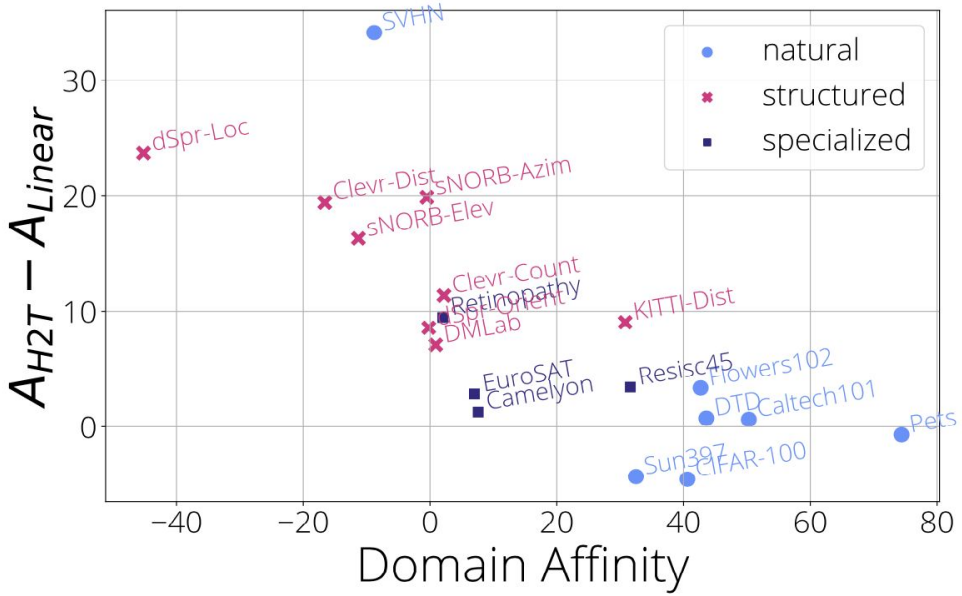
$$|\boldsymbol{W}|_{2,1} = |\boldsymbol{s}|_1 = \sum_i |s_i| \quad ; \quad s_i = \sqrt{\sum_j w_{ij}^2}$$

- After calculating the scores, keep a fraction F of features and train a linear classifier on the selected features.



Google Research

Head2Toe Improves OOD Generalization



Google Research

Results on ResNet-50

- We match/exceed the finetuning results reported in VTAB paper*.

	Natural							Structured				Specialized								
	CIFAR-100	Caltech101	DTD	Flowers102	Pets	SVHN	Sun397	Camelyon	EuroSAT	Resisc45	Retinopathy	Clevr-Count	Clevr-Dist	DMLab	KITTI-Dist	dSpr-Loc	dSpr-Ori	sNORB-Azim	sNORB-Elev	Mean
Linear	49.2	86.4	68.2	84.6	88.8	47.6	35.7	**83.3**	92.4	73.2	74.0	41.3	39.0	37.1	66.7	31.8	30.4	17.9	26.7	56.5
+All-ℓ_2	44.7	87.0	67.8	84.2	86.1	81.1	31.9	82.6	95.0	76.5	74.5	50.0	56.3	38.3	65.5	59.7	44.5	37.5	40.0	63.3
+All-ℓ_1	50.8	88.6	67.4	84.2	87.7	84.2	34.6	80.9	94.9	75.6	**74.7**	49.9	57.0	41.8	72.9	59.0	44.8	37.5	40.8	64.6
+All-$\ell_{2,1}$	49.1	86.7	**68.5**	84.2	88.0	84.4	34.8	81.5	94.9	75.7	74.3	48.3	58.4	42.0	74.4	58.8	45.2	37.8	34.4	64.3
Head2Toe	47.1	88.8	67.6	85.6	87.6	84.1	32.9	82.1	94.3	76.0	74.1	**55.3**	**59.5**	**43.9**	72.3	64.9	**51.1**	**39.6**	**43.1**	**65.8**
Fine-tuning*	**54.6**	**89.8**	65.6	**88.4**	**89.1**	**86.3**	34.5	79.7	**95.3**	**81.0**	72.6	41.8	52.5	42.7	**75.3**	81.0	47.3	32.6	35.8	65.6
Scratch*	11.0	37.7	23.0	40.2	13.3	59.3	3.9	73.5	84.8	41.6	63.1	38.5	54.8	35.8	36.9	**87.9**	37.3	20.9	36.9	42.1

FLOPs (vs FINETUNING)	Size (vs FINETUNING)	Size (vs LINEAR)
0.006295	0.010729	5.674742

Google Research

Thank you.

Google DeepMind

PEFT Lab

TODO(add structure and WiP colab)

LoRA Collab Notebook