Efficiency

CS 5539: Advanced Topics in Natural Language Processing

https://shocheen.github.io/courses/advanced-nlp-fall-2024

Logistics

- Have you formed your course project teams?
    - Project proposal deadline: September 30

Goal for today’s class

How to train/use LLMs with a low compute budget

Part I: Efficient fine tuning techniques (LoRA)

Part II: LLM.int8() – quantizing LLM parameters to take less memory

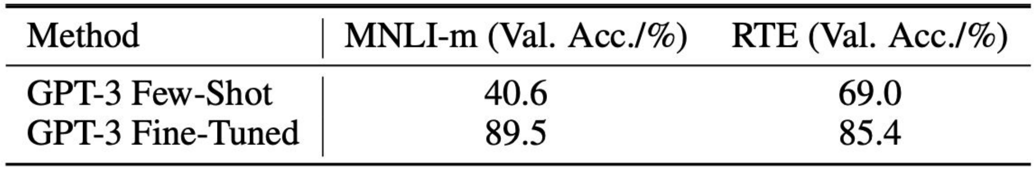
Part I: Parameter Efficient Finetuning

LoRA - Stakeholder

✍: Suchit Gupte

Finetuning yields performance

GPT-3



Method	MNLI-m (Val. Acc./%)	RTE (Val. Acc./%)
GPT-3 Few-Shot	40.6	69.0
GPT-3 Fine-Tuned	89.5	85.4

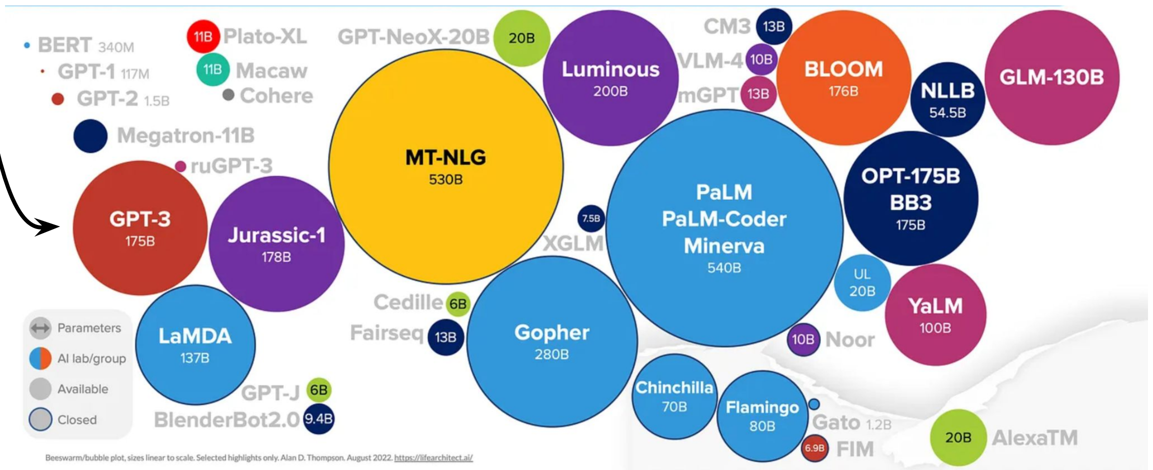
✍️: Suchit Gupte

Finetuning yields performance

GPT-3

Method	MNLI-m (Val. Acc./%)	RTE (Val. Acc./%)
GPT-3 Few-Shot	40.6	69.0
GPT-3 Fine-Tuned	89.5	85.4

175 B Training parameters



✍️: Suchit Gupte

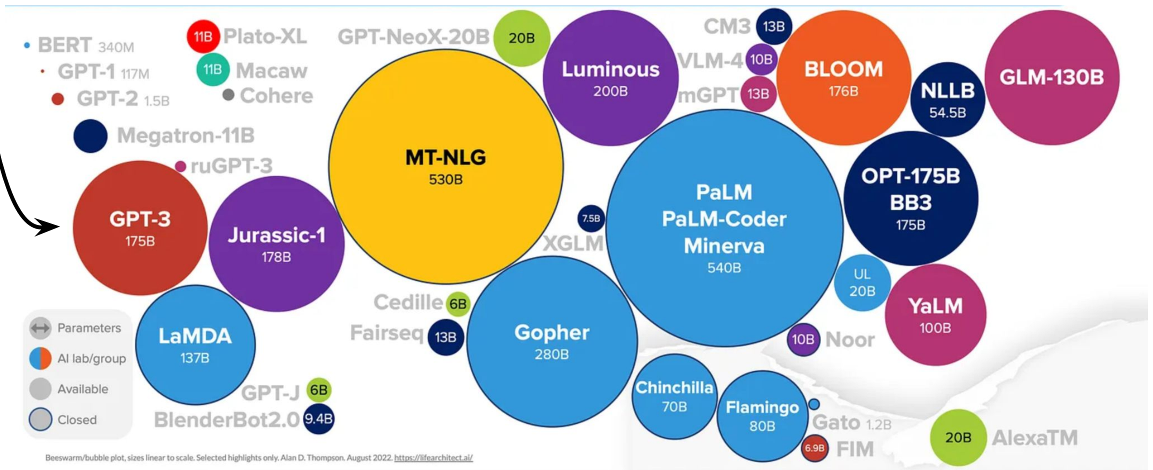
Finetuning yields performance

GPT-3

Method	MNLI-m (Val. Acc./%)	RTE (Val. Acc./%)
GPT-3 Few-Shot	40.6	69.0
GPT-3 Fine-Tuned	89.5	85.4

175 B Training parameters

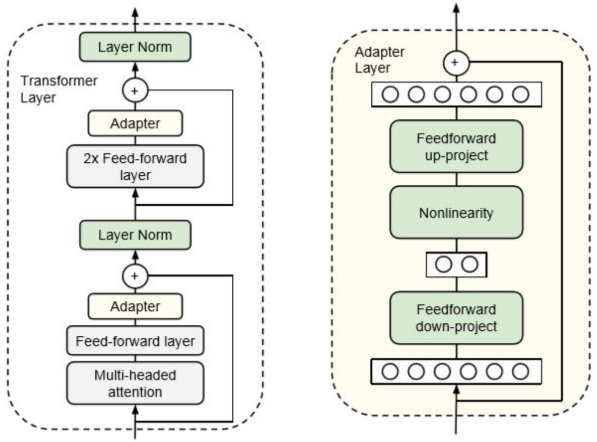
Finetuning billions of parameters - NIGHTMARE!



✍: Suchit Gupte

Existing finetuning techniques

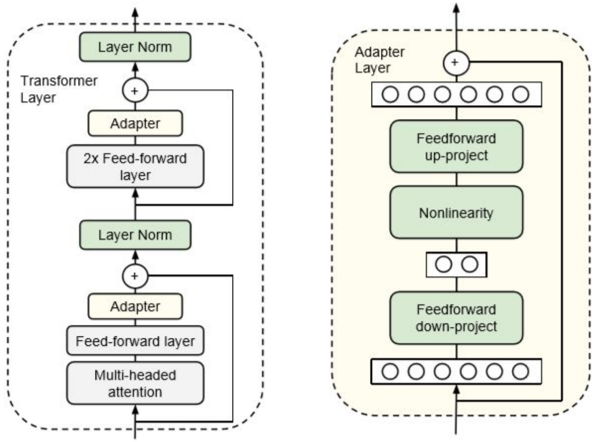
Adapter tuning



✍️: Suchit Gupte

Existing finetuning techniques

Adapter tuning

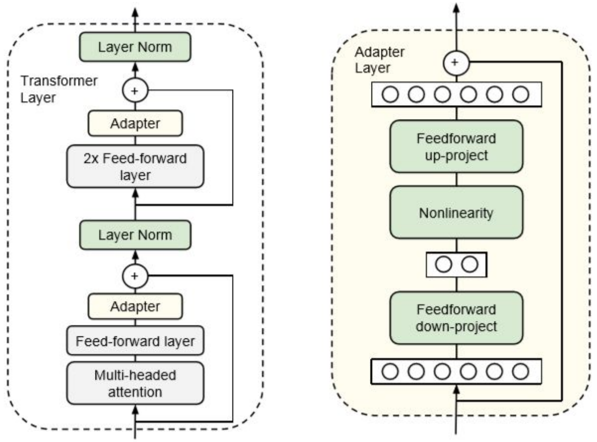


Inference latency

✍: Suchit Gupte

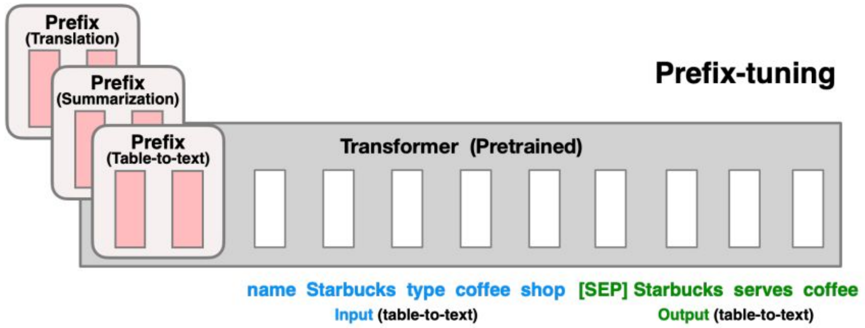
Existing finetuning techniques

Adapter tuning



Inference latency

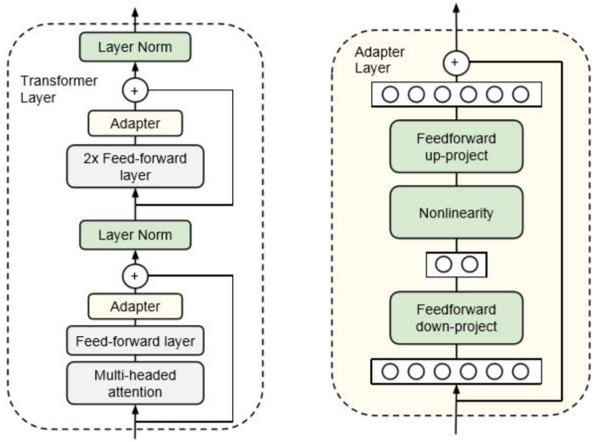
Prefix tuning



✍️: Suchit Gupte

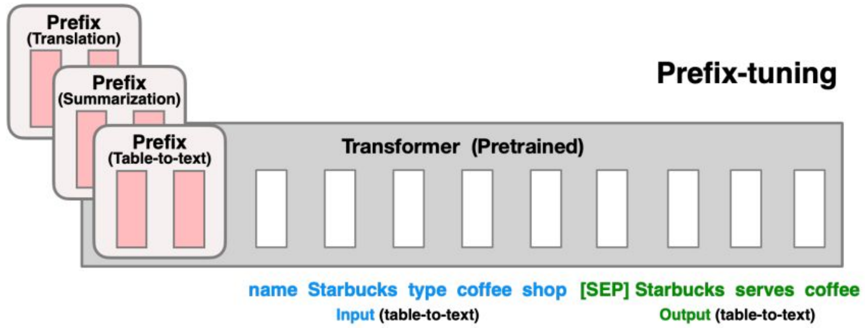
Existing finetuning techniques

Adapter tuning



Inference latency

Prefix tuning



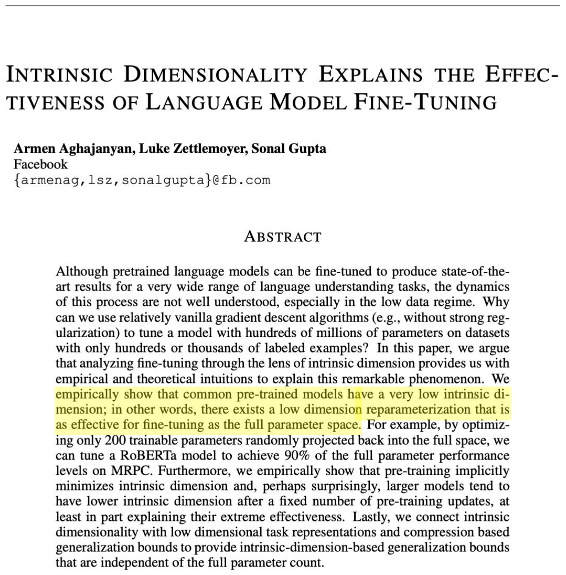
Suboptimal performance

✍️: Suchit Gupte

What is LORA?

A method of fine-tuning large pre-trained models by decomposing weight matrices

Pretrained LMs have a low intrinsic dimension



INTRINSIC DIMENSIONALITY EXPLAINS THE EFFECTIVENESS OF LANGUAGE MODEL FINE-TUNING

Armen Aghajanyan, Luke Zettlemoyer, Sonal Gupta
Facebook
{armenag,lsz,sonalgupta}@fb.com

ABSTRACT

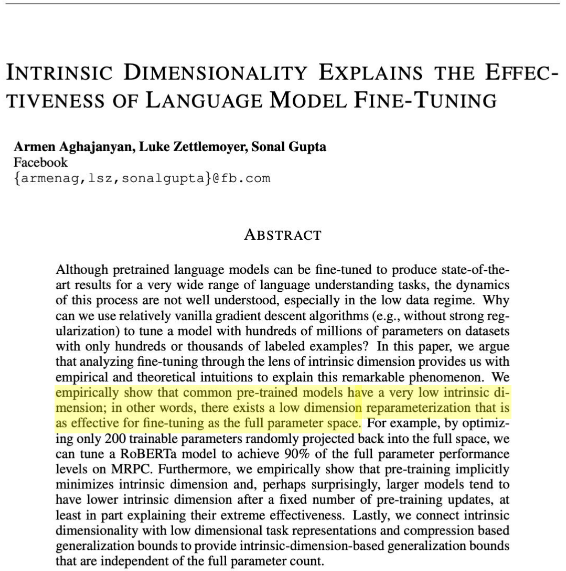
Although pretrained language models can be fine-tuned to produce state-of-the-art results for a very wide range of language understanding tasks, the dynamics of this process are not well understood, especially in the low data regime. Why can we use relatively vanilla gradient descent algorithms (e.g., without strong regularization) to tune a model with hundreds of millions of parameters on datasets with only hundreds or thousands of labeled examples? In this paper, we argue that analyzing fine-tuning through the lens of intrinsic dimension provides us with empirical and theoretical intuitions to explain this remarkable phenomenon. We empirically show that common pre-trained models have a very low intrinsic dimension; in other words, there exists a low dimension reparameterization that is as effective for fine-tuning as the full parameter space. For example, by optimizing only 200 trainable parameters randomly projected back into the full space, we can tune a RoBERTa model to achieve 90% of the full parameter performance levels on MRPC. Furthermore, we empirically show that pre-training implicitly minimizes intrinsic dimension and, perhaps surprisingly, larger models tend to have lower intrinsic dimension after a fixed number of pre-training updates, at least in part explaining their extreme effectiveness. Lastly, we connect intrinsic dimensionality with low dimensional task representations and compression based generalization bounds to provide intrinsic-dimension-based generalization bounds that are independent of the full parameter count.

✍️: Suchit Gupte

What is LORA?

A method of fine-tuning large pre-trained models by decomposing weight matrices

Pretrained LMs have a low intrinsic dimension



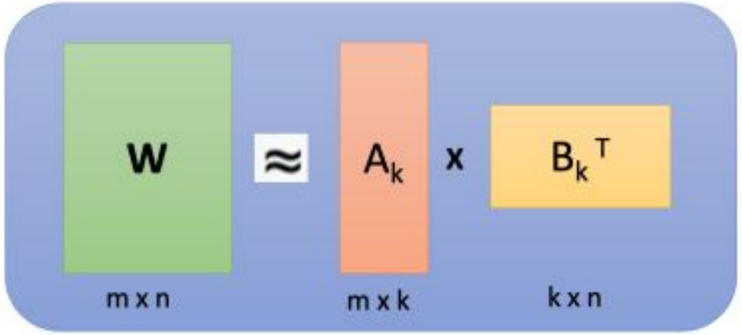
INTRINSIC DIMENSIONALITY EXPLAINS THE EFFECTIVENESS OF LANGUAGE MODEL FINE-TUNING

Armen Aghajanyan, Luke Zettlemoyer, Sonal Gupta
Facebook
`{armenag,lsz,sonalgupta}@fb.com`

ABSTRACT

Although pretrained language models can be fine-tuned to produce state-of-the-art results for a very wide range of language understanding tasks, the dynamics of this process are not well understood, especially in the low data regime. Why can we use relatively vanilla gradient descent algorithms (e.g., without strong regularization) to tune a model with hundreds of millions of parameters on datasets with only hundreds or thousands of labeled examples? In this paper, we argue that analyzing fine-tuning through the lens of intrinsic dimension provides us with empirical and theoretical intuitions to explain this remarkable phenomenon. We empirically show that common pre-trained models have a very low intrinsic dimension; in other words, there exists a low dimension reparameterization that is as effective for fine-tuning as the full parameter space. For example, by optimizing only 200 trainable parameters randomly projected back into the full space, we can tune a RoBERTa model to achieve 90% of the full parameter performance levels on MRPC. Furthermore, we empirically show that pre-training implicitly minimizes intrinsic dimension and, perhaps surprisingly, larger models tend to have lower intrinsic dimension after a fixed number of pre-training updates, at least in part explaining their extreme effectiveness. Lastly, we connect intrinsic dimensionality with low dimensional task representations and compression based generalization bounds to provide intrinsic-dimension-based generalization bounds that are independent of the full parameter count.

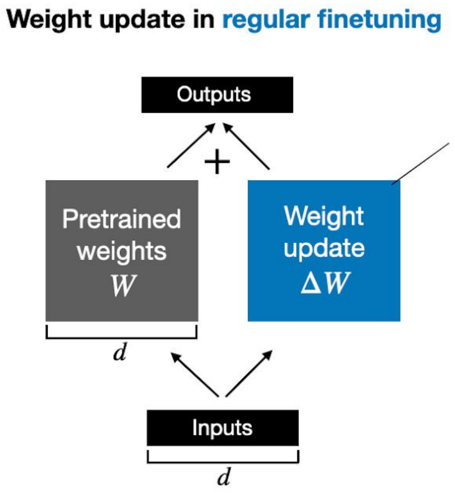
Low rank decomposition



✍️: Suchit Gupte

What is LORA?

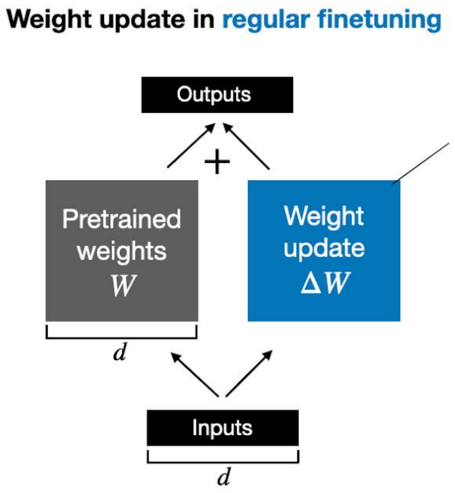
A method of fine-tuning large pre-trained models by decomposing weight matrices



✍️: Suchit Gupte

What is LORA?

A method of fine-tuning large pre-trained models by decomposing weight matrices



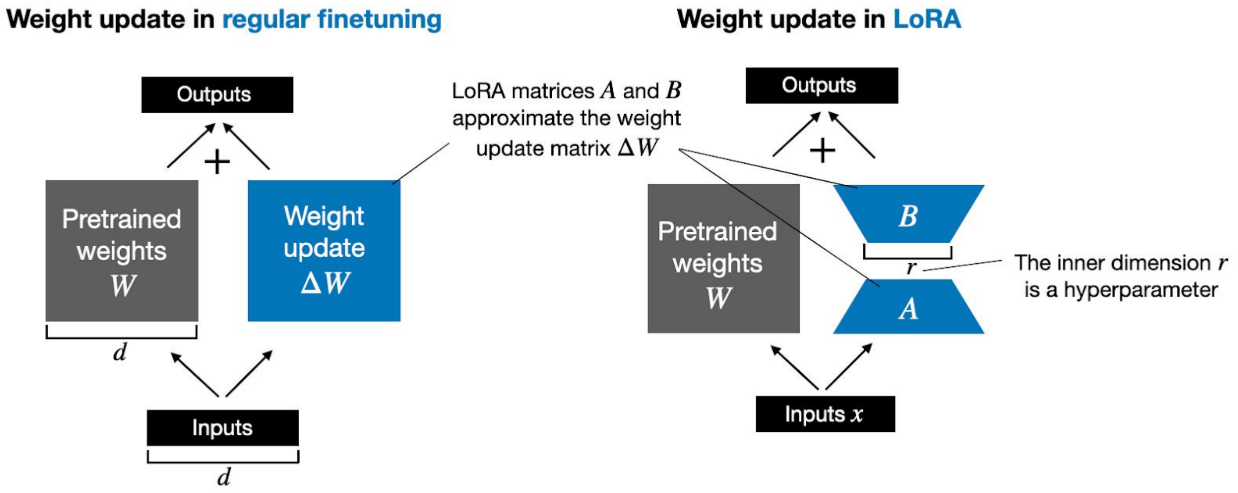
Assumption:
If pretrained weights are low rank, weight update also must be low rank

✍️: Suchit Gupte

What is LORA?

A method of fine-tuning large pre-trained models by decomposing weight matrices

Assumption: If pretrained weights are low rank, weight update also must be low rank

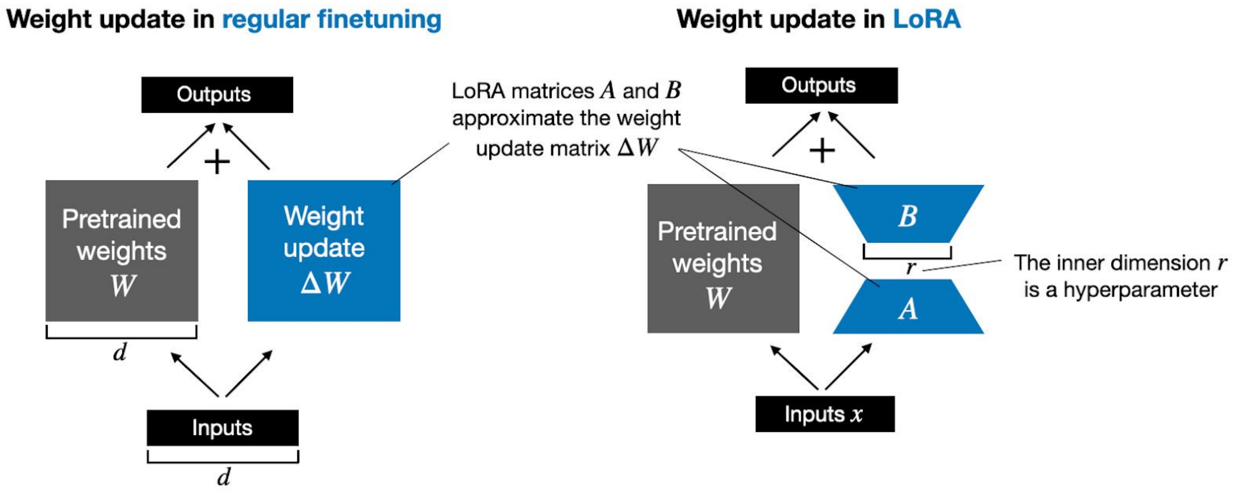

Weight update in regular finetuning
Outputs
+
Pretrained weights W
Weight update ΔW
d
Inputs
d
LoRA matrices A and B approximate the weight update matrix ΔW
Weight update in LoRA
Outputs
+
Pretrained weights W
B
r
A
The inner dimension r is a hyperparameter
Inputs x

✍: Suchit Gupte

What is LORA?

A method of fine-tuning large pre-trained models by decomposing weight matrices

Assumption: If pretrained weights are low rank, weight update also must be low rank



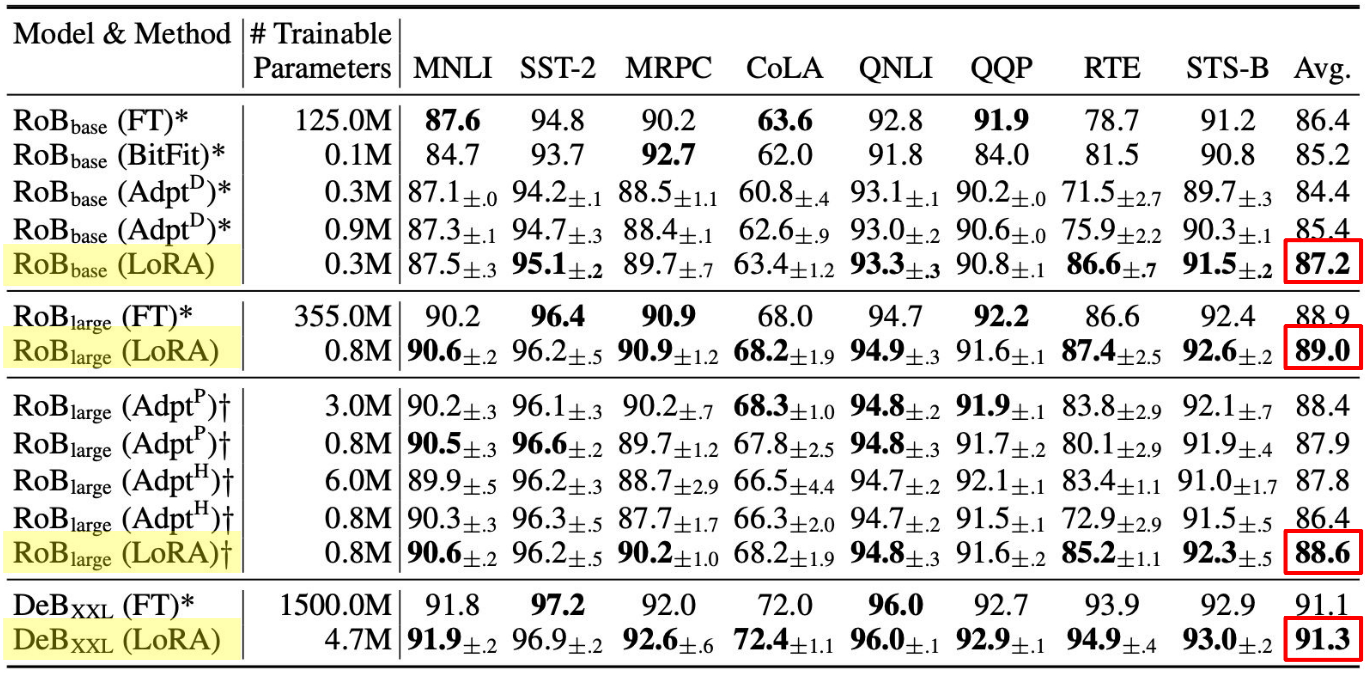
Weight update in regular finetuning
Outputs
Pretrained weights W
Weight update ΔW
d
Inputs
d
LoRA matrices A and B approximate the weight update matrix ΔW
Weight update in LoRA
Outputs
Pretrained weights W
B
r
A
The inner dimension r is a hyperparameter
Inputs x

Modified forward pass

$$\boxed{h = W_0 x + \Delta W x = W_0 x + BAx}$$

✍: Suchit Gupte

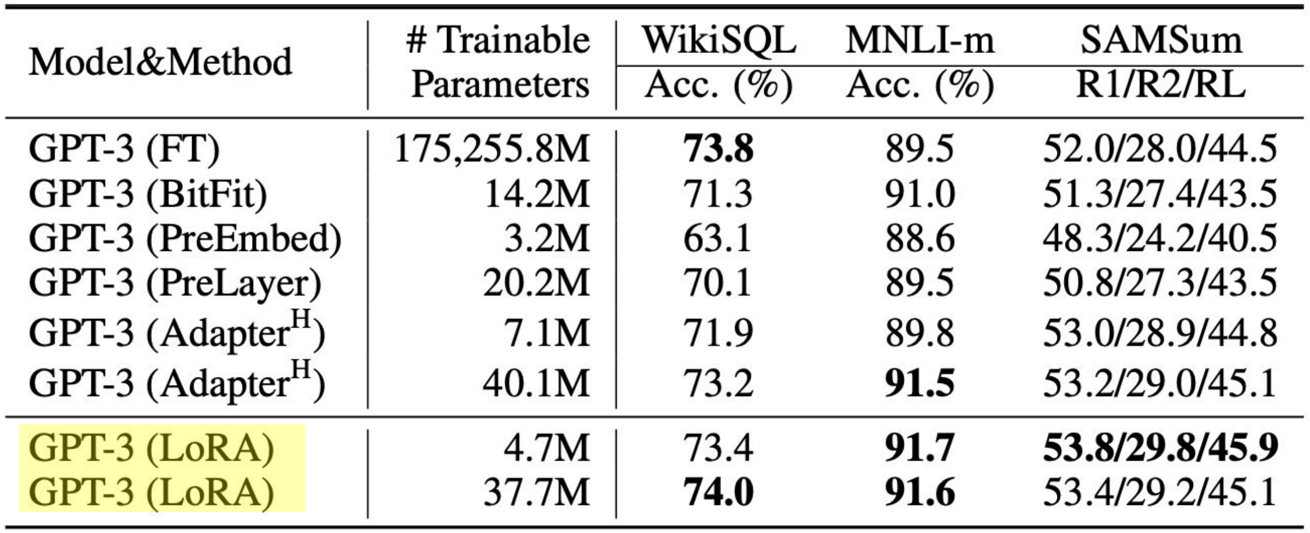
Performance on GLUE benchmark



Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB$_{\text{base}}$ (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB$_{\text{base}}$ (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB$_{\text{base}}$ (Adpt$^{\text{D}}$)*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB$_{\text{base}}$ (Adpt$^{\text{D}}$)*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB$_{\text{base}}$ (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	$\boxed{\mathbf{87.2}}$
RoB$_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
RoB$_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	$\boxed{\mathbf{89.0}}$
RoB$_{\text{large}}$ (Adpt$^{\text{P}}$)†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
RoB$_{\text{large}}$ (Adpt$^{\text{P}}$)†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
RoB$_{\text{large}}$ (Adpt$^{\text{H}}$)†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
RoB$_{\text{large}}$ (Adpt$^{\text{H}}$)†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
RoB$_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	$\boxed{\mathbf{88.6}}$
DeB$_{\text{XXL}}$ (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB$_{\text{XXL}}$ (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	$\boxed{\mathbf{91.3}}$

✍️: Suchit Gupte

Performance on GPT-3



Model&Method	# Trainable Parameters	WikiSQL	MNLI-m	SAMSum
		Acc. (%)	Acc. (%)	R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^H)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^H)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

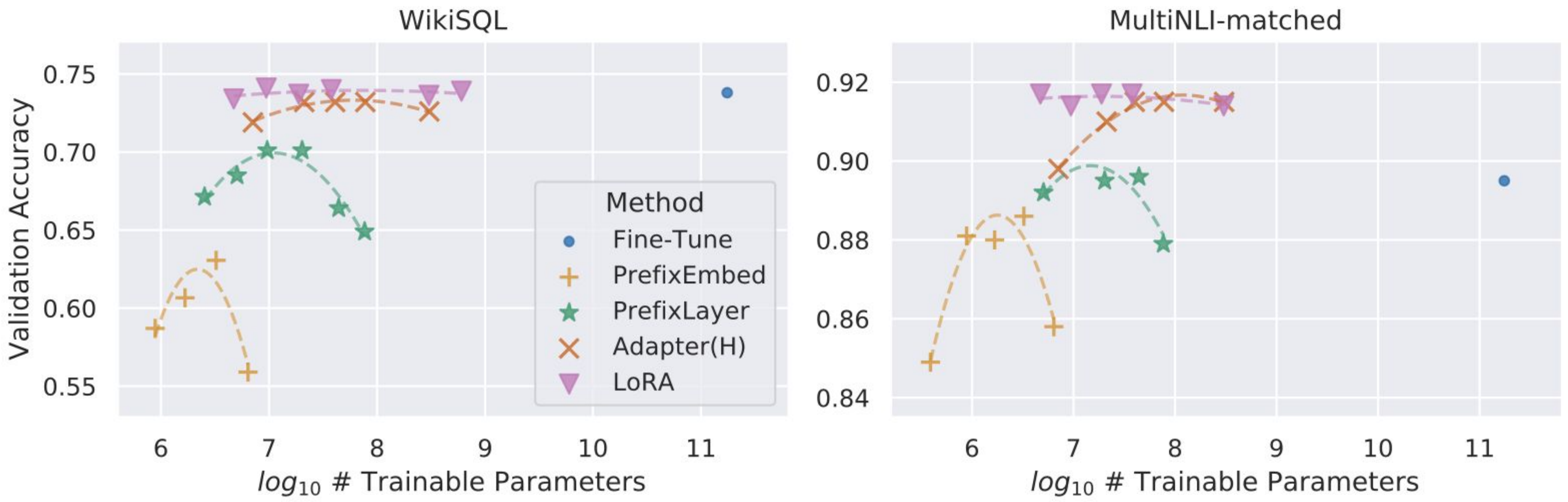
# Trainable parameters

0.0025% 🔻

0.02% 🔻

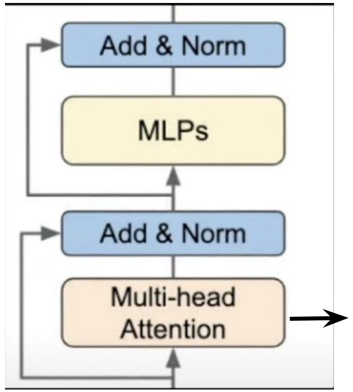
✍️: Suchit Gupte

Scalability - Performance vs # Parameters



✍: Suchit Gupte

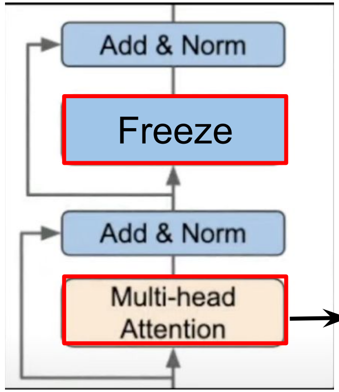
Applying LORA to transformers



$\rightarrow$ Weights - Query, Key, Value and Output matrices

✍️: Suchit Gupte

Applying LORA to transformers

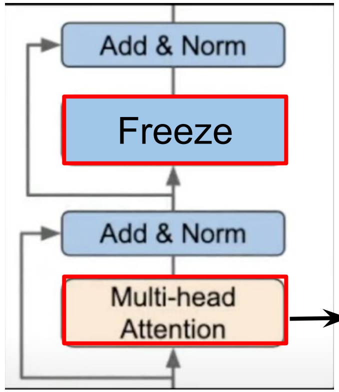


Weights - Query, Key, Value and Output matrices

Apply LORA

✍️: Suchit Gupte

Applying LORA to transformers



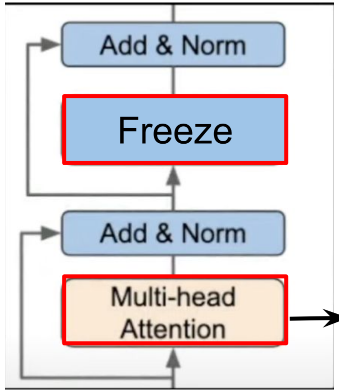
Weights - Query, Key, Value and Output matrices

Apply LORA → **WHAT IS THE OPTIMAL RANK r FOR LORA?**

✍: Suchit Gupte

Applying LORA to transformers

	Weight Type	$r=1$	$r=2$	$r=4$	$r=8$	$r=64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4



Weights - Query, Key, Value and Output matrices

Apply LORA → **WHAT IS THE OPTIMAL RANK r FOR LORA?**

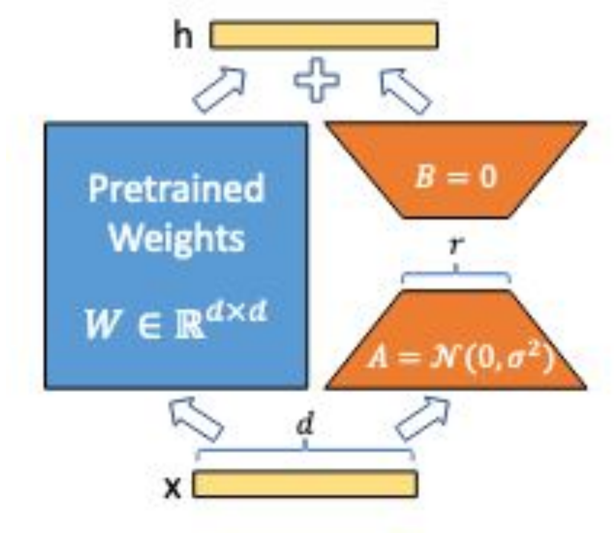
✍: Suchit Gupte

LoRA - Reviewer

Alex Felderean

Summary

- **Lo**w-**R**ank **A**daptation (LoRA) for reducing parameters during fine-tuning
- Demonstrates:
  - parameter reduction by 10,000x
  - GPU memory requirement by 3x
- Compared to other common methods:
  - no added inference latency (vs adapters)
  - on-par or better practical performance
  - higher training throughput


h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Strengths and Weaknesses

Originality:	+ Parallel, *NOT* sequential modules used throughout fine-tuning
	+ Reduces time significantly by cutting out downstream work
	- Lacks in-depth limitations of method in practicality
Quality:	+ Graphics and explanations clear, good line of reasoning
Clarity:	+ Sections well organized and labeled to follow logic
	- Could use more visuals, especially for building context
Significance:	+ Very powerful for resource-limited environments (say, only 1 GPU)
	~ Pros and cons shift depending on chosen models & resources
	- Full fine-tuning is still preferred, esp. for larger datasets

Questions

- Despite its generalization, where do we see LoRA tuning *not* converging roughly to training the original model? How does that limit the performance of LoRA?
- How have developments using the LoRA method paced with improvements of other fine-tuning methods? What limitations have been addressed with that method that affect LoRA's usefulness?
- The paper roughly tested GPT-3's 170B parameters. With models suspected to break the 1T parameter mark, does LoRA's basis of generalization still hold the same impact in performance?

Ratings

Soundness: 3/4 (relatively well supported with evidence)

Presentation: 4/4 (very easy to follow and understand)

Contribution: 4/4 (grown into a popular method, impactful to AI field)

Overall: 8/10 (Strong Accept)

Confidence: 4/5

LoRA - Archaeologist

By Yifei Li, 09/16/2024

Background

- **Why** parameter-efficient fine-tuning (PEFT)?

Background

- **Why** parameter-efficient fine-tuning (PEFT)?
    - **Fine-tuning** on a **better pre-trained model** > directly train a **task-specific model**

Background

- **Why** parameter-efficient fine-tuning (PEFT)?
  - **Fine-tuning** on a **better pre-trained model** > directly train a **task-specific model**
  - Models are getting **bigger**! Training/inference/deployment…
    - T5 (330M, 770M, 3B, 11B)
    - GPT-2 (345M, 774M, 1.5B) – GPT-3 (175B) – ChatGPT, GPT-4, GPT-4o, o1 (?)

Background

- **Why** parameter-efficient fine-tuning (PEFT)?
  - **Fine-tuning** on a **better pre-trained model** > directly train a **task-specific model**
  - Models are getting **bigger**! Training/inference/deployment…
    - T5 (330M, 770M, 3B, 11B)
    - GPT-2 (330M, 774M, 1.5B) – GPT-3 (175B) – ChatGPT, GPT-4, GPT-4o, o1 (?)

- **How** to *parameter-efficiently* do fine-tuning?
  - The **existence** of Low-Rank Structures in Deep Learning
  - Techniques for parameter-efficient fine-tuning

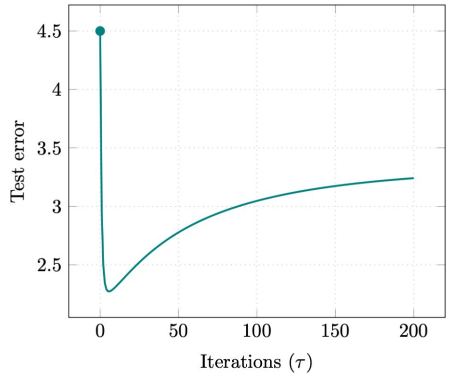
Low-Rank Structures in Deep Learning

- Many machine learning problems have certain intrinsic low-rank structure
  - https://arxiv.org/pdf/1906.05392
  - Why N(parameters) >> N(examples), but still generalizable?
  - Low-rank information space vs. nuisance space

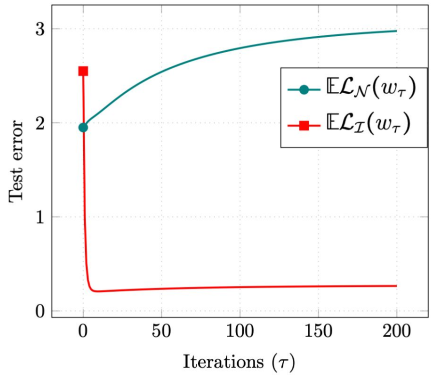
Oymak, Samet, et al. "Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian." *arXiv preprint arXiv:1906.05392* (2019).

Low-Rank Structures in Deep Learning

- SVD on Jacobian matrix: low-rank information space vs. nuisance space



(a) Total test error

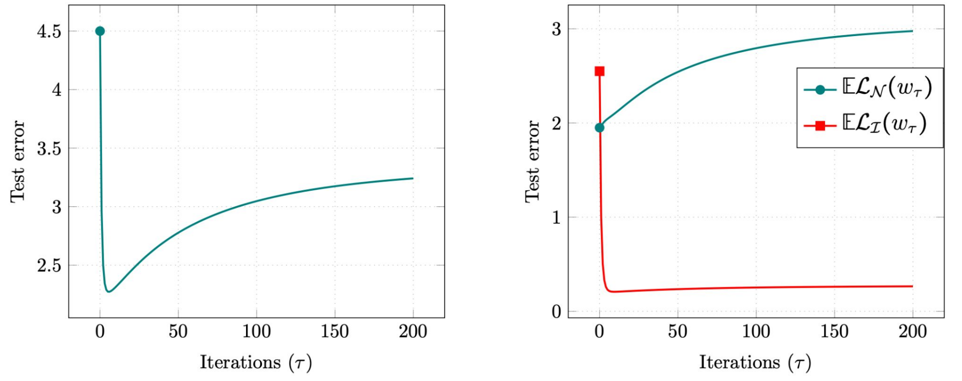


(b) Test error along information and nuisance spaces

Oymak, Samet, et al. "Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian." *arXiv preprint arXiv:1906.05392* (2019).

Low-Rank Structures in Deep Learning

- SVD on Jacobian matrix: low-rank information space vs. nuisance space
  - **Few but large** eigenvalues vs. **Many but small** eigenvalues

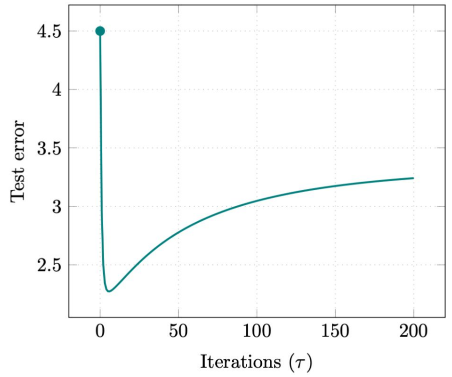


(a) Total test error (b) Test error along information and nuisance spaces

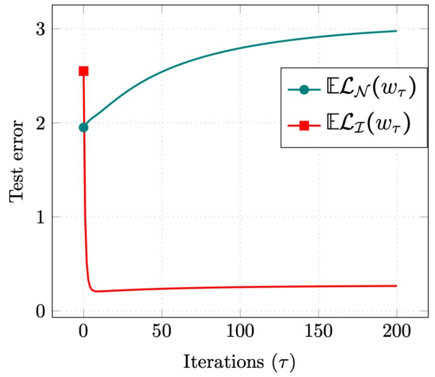
Oymak, Samet, et al. "Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian." *arXiv preprint arXiv:1906.05392* (2019).

Low-Rank Structures in Deep Learning

- SVD on Jacobian matrix: low-rank information space vs. nuisance space
  - **Few but large** eigenvalues vs. **Many but small** eigenvalues
  - Model **converges and generalizes fast** in information space



(a) Total test error

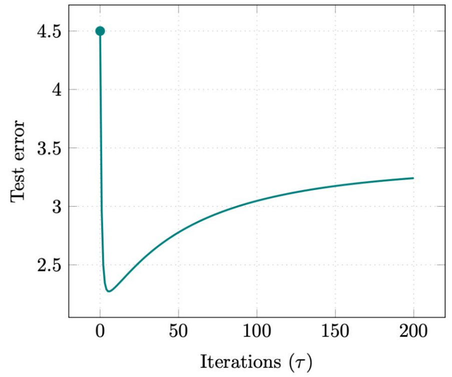


(b) Test error along information and nuisance spaces

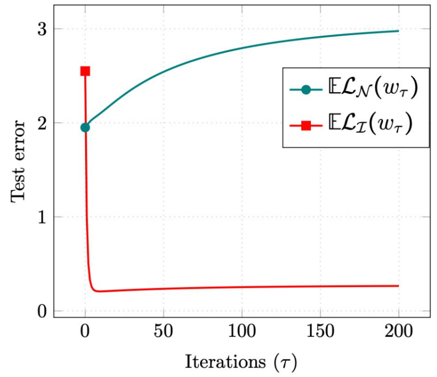
Oymak, Samet, et al. "Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian." *arXiv preprint arXiv:1906.05392* (2019).

Low-Rank Structures in Deep Learning

- SVD on Jacobian matrix: low-rank information space vs. nuisance space
  - **Few but large** eigenvalues vs. **Many but small** eigenvalues
  - Model converges and generalizes fast in information space
  - Model **converges slowly** in nuisance (noisy) space and **affects generalization**



(a) Total test error

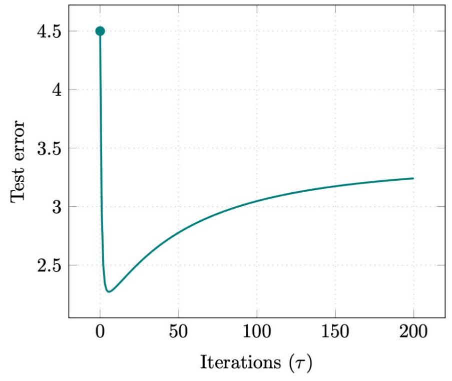


(b) Test error along information and nuisance spaces

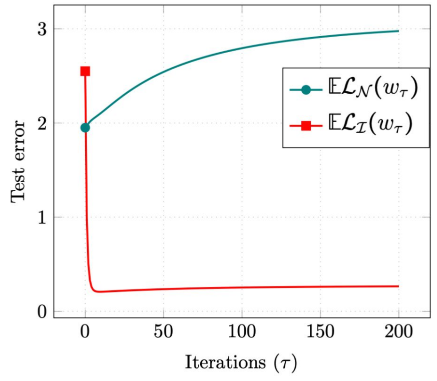
Oymak, Samet, et al. "Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian." *arXiv preprint arXiv:1906.05392* (2019).

Low-Rank Structures in Deep Learning

- SVD on Jacobian matrix: low-rank information space vs. nuisance space
  - **Few but large** eigenvalues vs. **Many but small** eigenvalues
  - Model converges and generalizes fast in information space
  - Model converges slowly in nuisance (noisy) space and affects generalization
  - That’s why **early-stopping** may work



(a) Total test error



(b) Test error along information and nuisance spaces

Oymak, Samet, et al. "Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian." *arXiv preprint arXiv:1906.05392* (2019).

Low-Rank Structures in Deep Learning

- Explicitly integrates low-rank matrix factorization during training
  - SVD, PCA, top-k eigenvalues

Low-Rank Structures in Deep Learning

- Explicitly integrates low-rank matrix factorization during training
  - SVD, PCA, top-k eigenvalues
  - Directly use factorized matrices to fit NN layers
    - Sainath et al., 2013; Povey et al., 2018 … (see “related works” in the paper)

Low-Rank Structures in Deep Learning

- Explicitly integrates low-rank matrix factorization during training
  - SVD, PCA, top-k eigenvalues
  - Directly use factorized matrices to fit NN layers
    - Sainath et al., 2013; Povey et al., 2018 … (see “related works” in the paper)

- Why not incorporate this into the fine-tuning stage?

Low-Rank Structures in Deep Learning

- Explicitly integrates low-rank matrix factorization during training
  - SVD, PCA, top-k eigenvalues
  - Directly use factorized matrices to fit NN layers
    - Sainath et al., 2013; Povey et al., 2018 … (see “related works” in the paper)

- Why not incorporate this into the fine-tuning stage?
  - A **low-rank update** to a **frozen model** for **adaptation to downstream tasks**

PEFT Techniques

- Where do the efficient parameters come from?

PEFT Techniques

- Where do the efficient parameters come from?
  - Additional inner modules?
    - Adapter-tuning: https://arxiv.org/pdf/1902.00751

PEFT Techniques

- Where do the efficient parameters come from?
  - Additional inner modules?
    - Adapter-tuning: https://arxiv.org/pdf/1902.00751
  - Embedding tokens?
    - Prefix-tuning: https://arxiv.org/pdf/2101.00190

PEFT Techniques

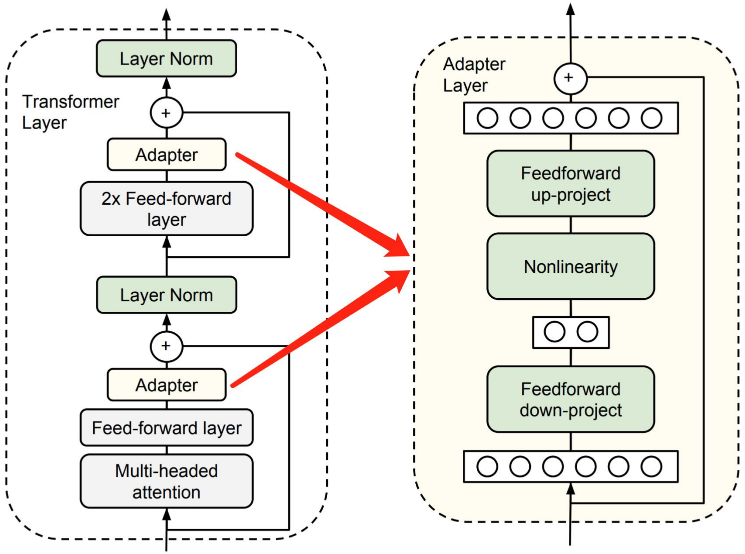
- Where do the efficient parameters come from?
    - Additional inner modules?
        - Adapter-tuning: https://arxiv.org/pdf/1902.00751
    - Embedding tokens?
        - Prefix-tuning: https://arxiv.org/pdf/2101.00190
    - Sub-networks?
        - Child-tuning: https://arxiv.org/pdf/2109.05687

PEFT Techniques

- Where do the efficient parameters come from?
  - Additional inner modules?
    - Adapter-tuning: https://arxiv.org/pdf/1902.00751
  - Embedding tokens?
    - Prefix-tuning: https://arxiv.org/pdf/2101.00190
  - Sub-networks?
    - Child-tuning: https://arxiv.org/pdf/2109.05687
  - Additional input/output layers?
    - Input-tuning: https://arxiv.org/pdf/2203.03131

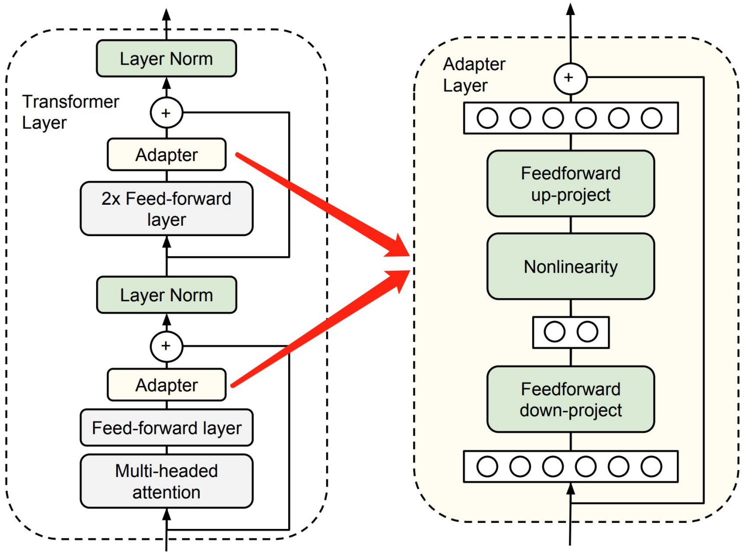
PEFT Techniques (1): Adapters

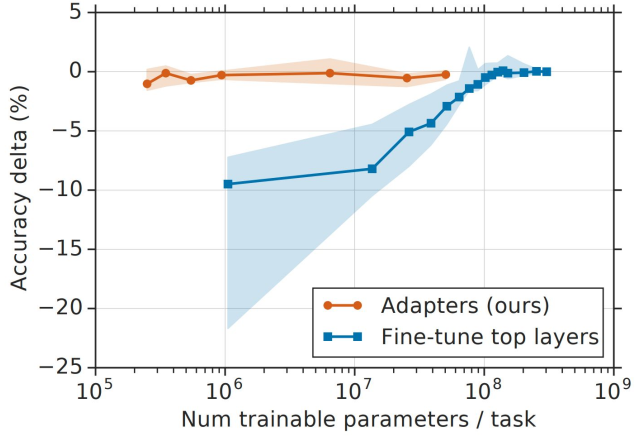
- Inserting **adapters** between NN layers (i.e. submodules **between** layers)
  - Rebuffi et al., 2017; Houlsby et al., 2019; Lin et al., 2020



PEFT Techniques (1): Adapters

- Inserting **adapters** between NN layers (i.e. submodules **between** layers)
  - Rebuffi et al., 2017; Houlsby et al., 2019; Lin et al., 2020
  - Better than fine-tuning only top layers!

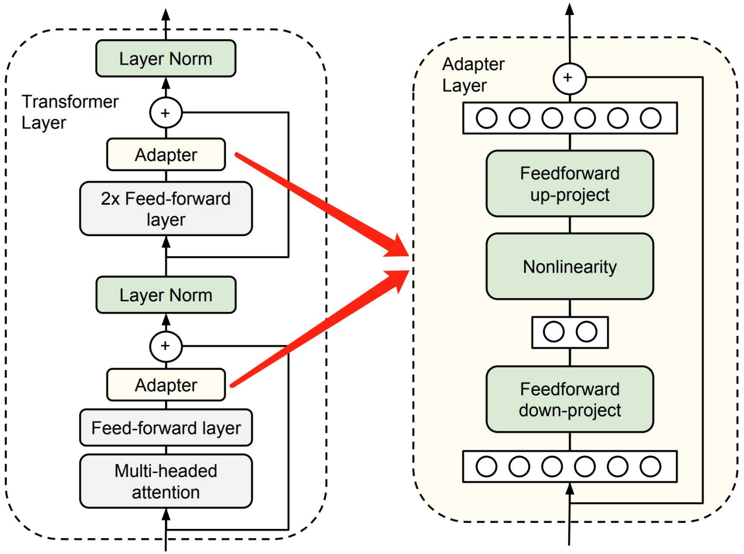


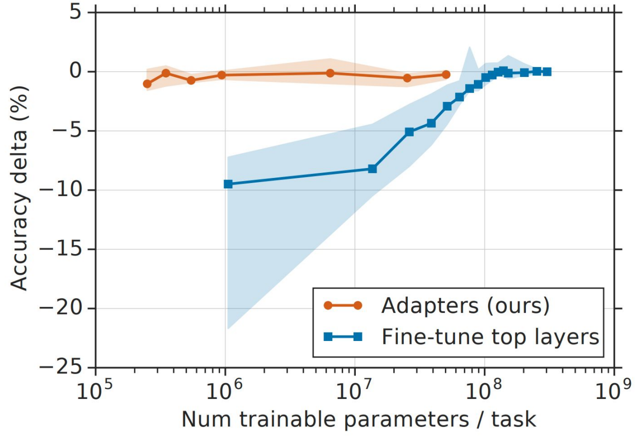


PEFT Techniques (1): Adapters

- Inserting **adapters** between NN layers (i.e. submodules **between** layers)
  - Rebuffi et al., 2017; Houlsby et al., 2019; Lin et al., 2020
  - Better than fine-tuning only top layers!

Extend **model length** and increase **inference latency**





PEFT Techniques (2): **P**-Tuning (**P** for **P**refix or **P**rompt)

- Prefix-tuning (Stanford in Jan. 2021)
  - https://arxiv.org/abs/2101.00190

- P-tuning (Tsinghua in Mar. 2021)
  - https://arxiv.org/pdf/2103.10385

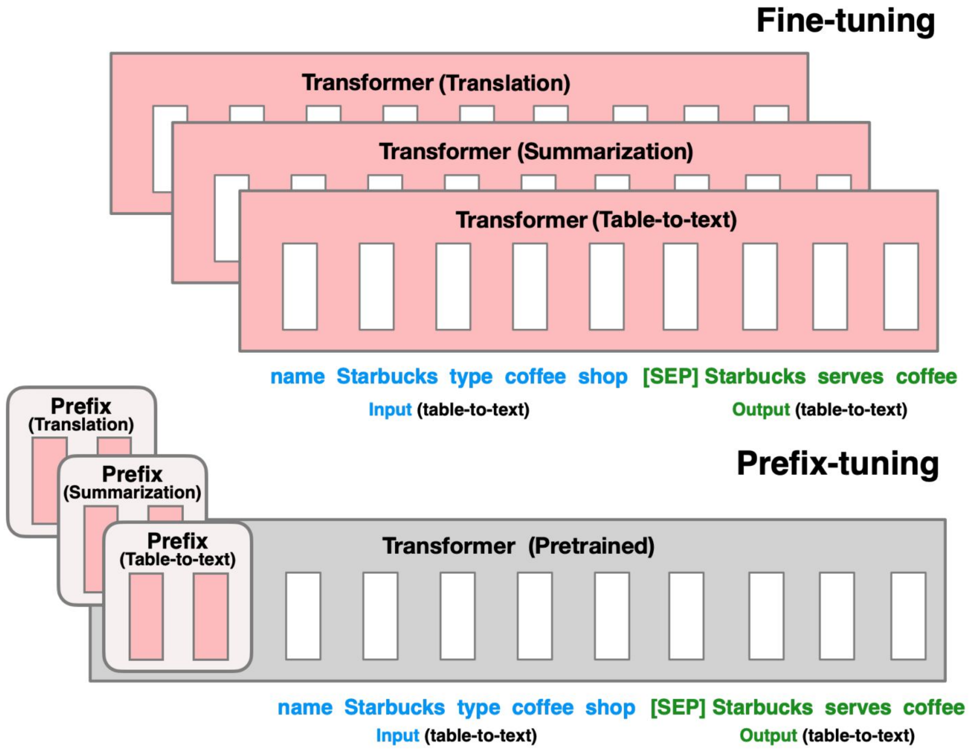
- Prompt-tuning (Google in Apr. 2021)
  - https://arxiv.org/pdf/2104.08691

PEFT Techniques (2): **P**-Tuning (**P** for **P**refix or **P**rompt)

- Prefix-tuning (Stanford in Jan. 2021)
  - https://arxiv.org/abs/2101.00190
- P-tuning (Tsinghua in Mar. 2021)
  - https://arxiv.org/pdf/2103.10385
- Prompt-tuning (Google in Apr. 2021)
  - https://arxiv.org/pdf/2104.08691

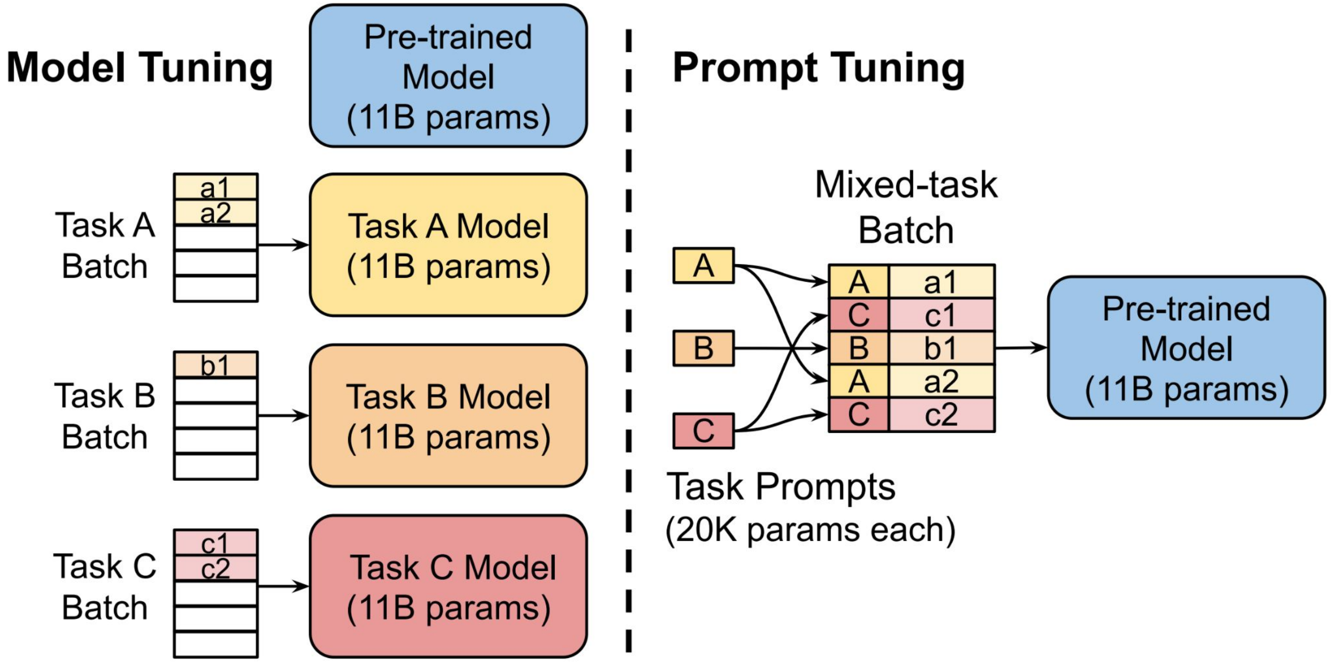
Optimizing
continuous prompt tokens

Prefix-tuning


Fine-tuning
Transformer (Translation)
Transformer (Summarization)
Transformer (Table-to-text)
name Starbucks type coffee shop [SEP] Starbucks serves coffee
Input (table-to-text)
Output (table-to-text)
Prefix (Translation)
Prefix (Summarization)
Prefix (Table-to-text)
Prefix-tuning
Transformer (Pretrained)
name Starbucks type coffee shop [SEP] Starbucks serves coffee
Input (table-to-text)
Output (table-to-text)

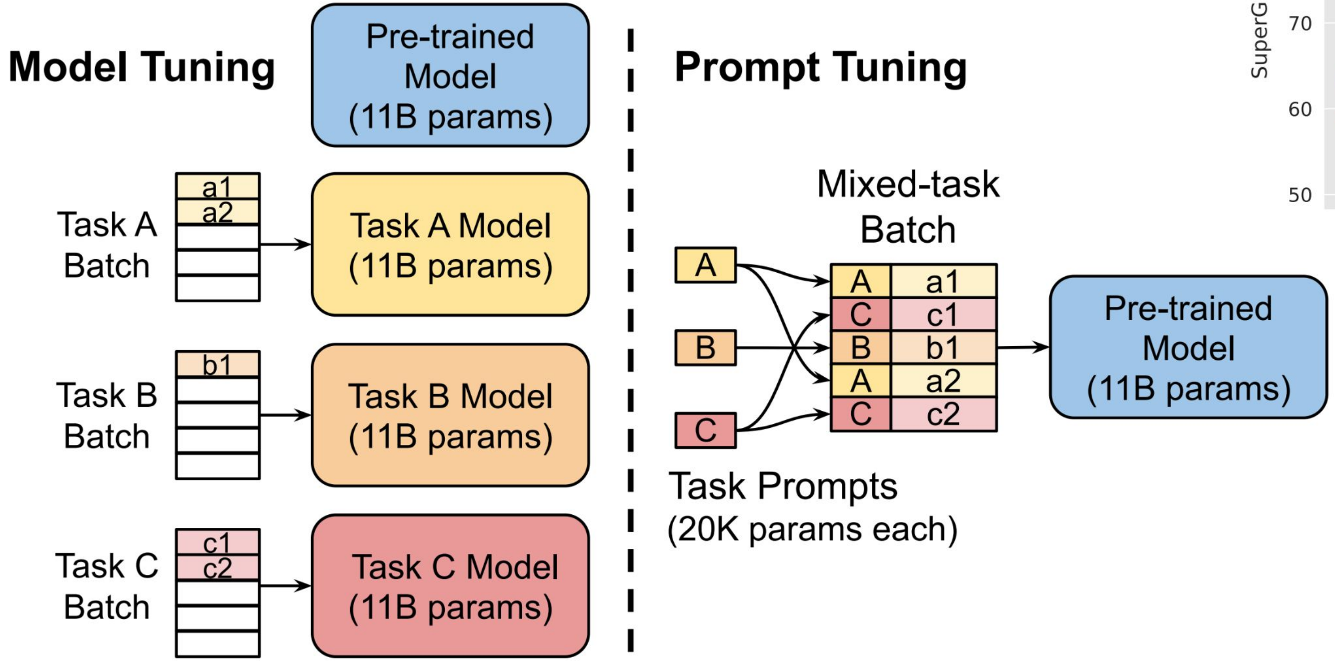
Li, Xiang Lisa, and Percy Liang. "Prefix-tuning: Optimizing continuous prompts for generation." *arXiv preprint arXiv:2101.00190* (2021).

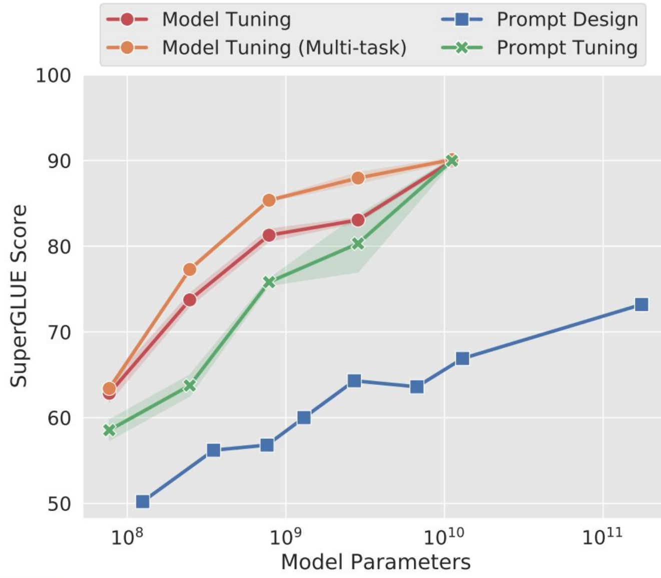
Prompt-tuning


Model Tuning
Pre-trained Model (11B params)
Task A Batch
a1
a2
Task A Model (11B params)
Task B Batch
b1
Task B Model (11B params)
Task C Batch
c1
c2
Task C Model (11B params)
Prompt Tuning
Mixed-task Batch
A
B
C
A a1
C c1
B b1
A a2
C c2
Pre-trained Model (11B params)
Task Prompts (20K params each)

Lester, Brian, Rami Al-Rfou, and Noah Constant. "The power of scale for parameter-efficient prompt tuning." *arXiv preprint arXiv:2104.08691* (2021).

Prompt-tuning

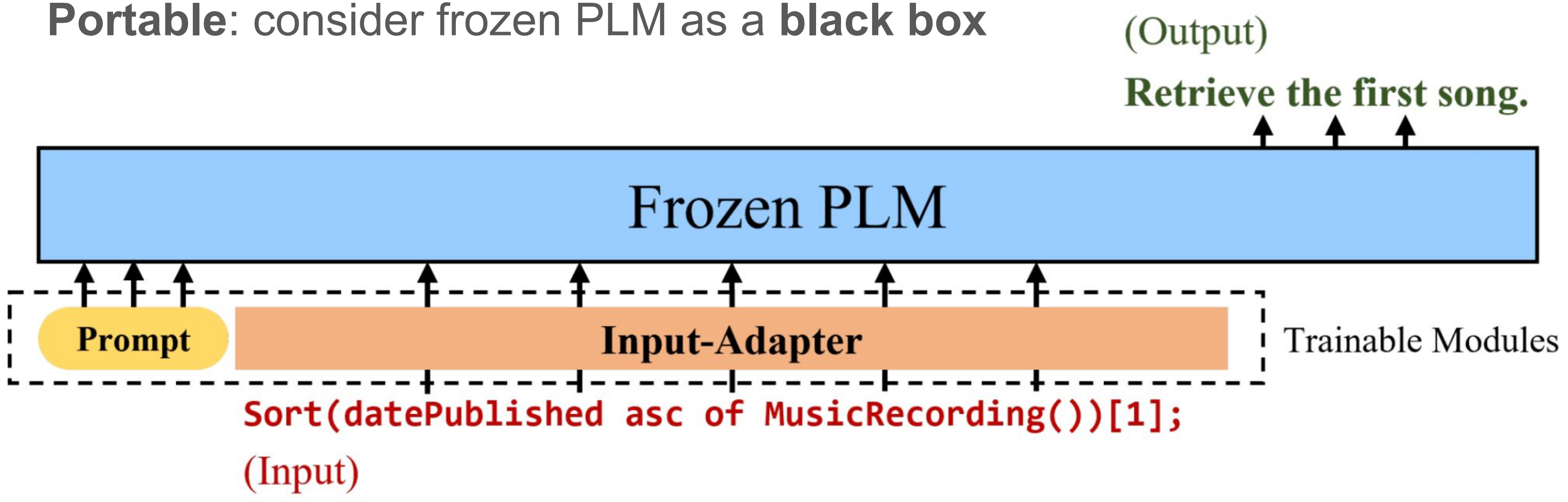

Model Tuning
Pre-trained Model (11B params)
Task A Batch
a1
a2
Task A Model (11B params)
Task B Batch
b1
Task B Model (11B params)
Task C Batch
c1
c2
Task C Model (11B params)
Prompt Tuning
Mixed-task Batch
A
B
C
A a1
C c1
B b1
A a2
C c2
Pre-trained Model (11B params)
Task Prompts (20K params each)



Lester, Brian, Rami Al-Rfou, and Noah Constant. "The power of scale for parameter-efficient prompt tuning." *arXiv preprint arXiv:2104.08691* (2021).

Input-Tuning

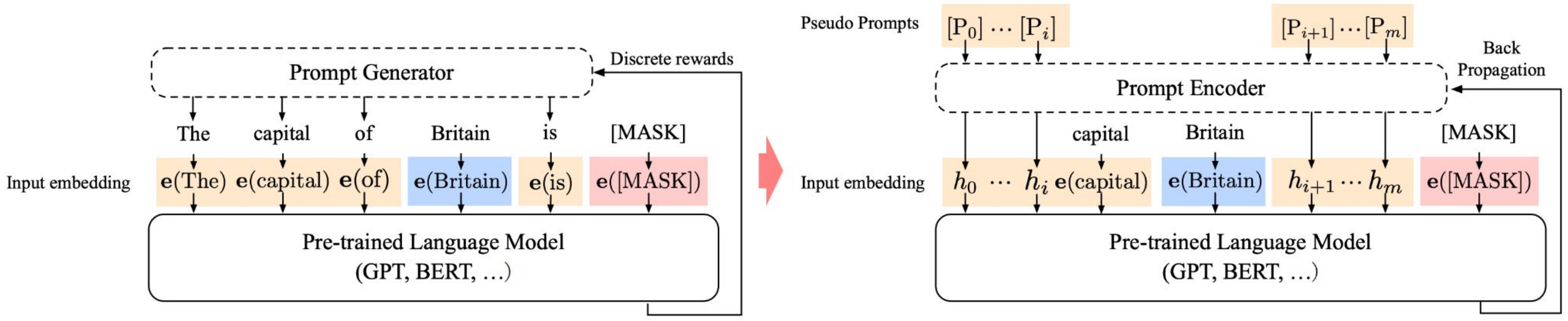
- Adding a **Input-Adapter** after inputs
- **Portable**: consider frozen PLM as a **black box**



An, Shengnan, et al. "Input-tuning: Adapting unfamiliar inputs to frozen pretrained models." *arXiv preprint arXiv:2203.03131* (2022).

P-tuning

- **Discrete** prompt words → **Continuous** trainable embedding tokens



Prompt Generator
Discrete rewards
The
capital
of
Britain
is
[MASK]
Input embedding
$\mathbf{e}(\text{The})$ $\mathbf{e}(\text{capital})$ $\mathbf{e}(\text{of})$ $\mathbf{e}(\text{Britain})$ $\mathbf{e}(\text{is})$ $\mathbf{e}([\text{MASK}])$
Pre-trained Language Model (GPT, BERT, ...)
Pseudo Prompts
$[\text{P}_0] \cdots [\text{P}_i]$
$[\text{P}_{i+1}] \cdots [\text{P}_m]$
Back Propagation
Prompt Encoder
capital
Britain
[MASK]
Input embedding
$h_0 \cdots h_i$ $\mathbf{e}(\text{capital})$ $\mathbf{e}(\text{Britain})$ $h_{i+1} \cdots h_m$ $\mathbf{e}([\text{MASK}])$
Pre-trained Language Model (GPT, BERT, ...)

(a) Discrete Prompt Search

(b) P-tuning

PEFT Techniques (2): **P**-Tuning (**P** for **P**refix or **P**rompt)

- Prefix-tuning (Stanford in Jan. 2021)
  - https://arxiv.org/abs/2101.00190
- P-tuning (Tsinghua in Mar. 2021)
  - https://arxiv.org/pdf/2103.10385
- Prompt-tuning (Google in Apr. 2021)
  - https://arxiv.org/pdf/2104.08691

Optimizing
continuous prompt tokens

Reduce the model's
usable sequence length

Summary

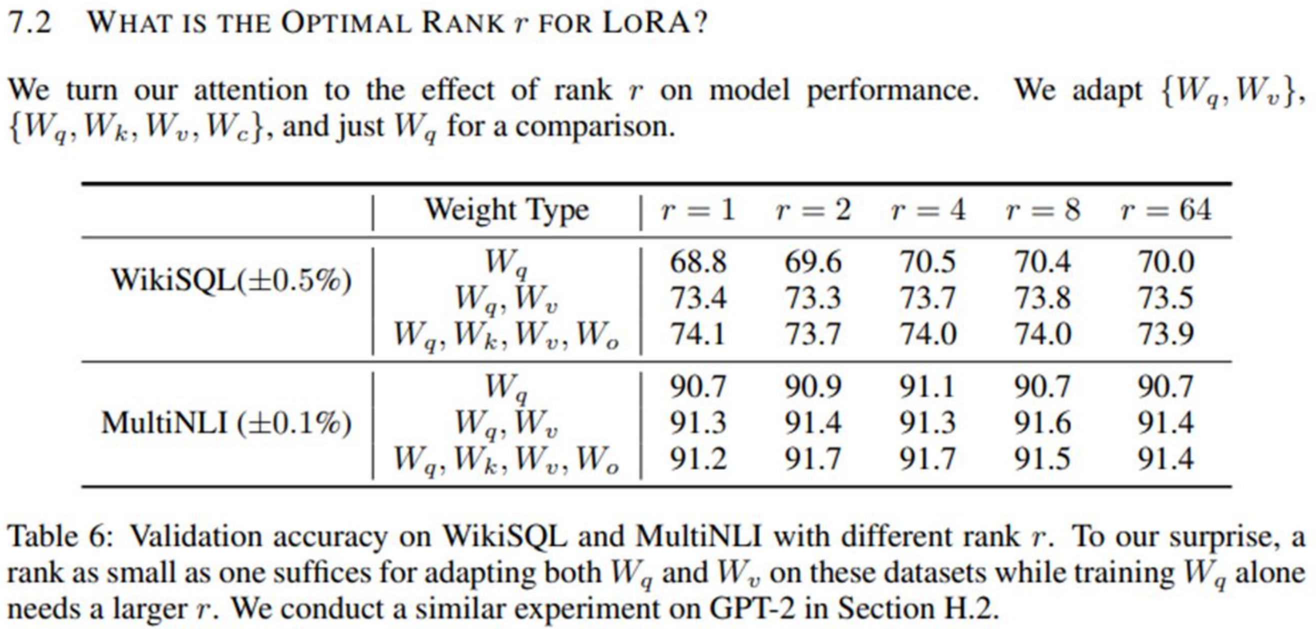
- Low-Rank Structures in Deep Learning
  - Why they exist and work

Summary

- Low-Rank Structures in Deep Learning
  - Why they exist and work

- Techniques for parameter-efficient fine-tuning
  - Source of tunable parameters: Adapters, Sub-networks, Continuous input embeddings, etc..
  - Pros/Cons: inference latency / usable sequence length

Follow up ideas



7.2 What is the Optimal Rank r for LoRA?

We turn our attention to the effect of rank r on model performance. We adapt $\{W_q, W_v\}$, $\{W_q, W_k, W_v, W_c\}$, and just W_q for a comparison.

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Table 6: Validation accuracy on WikiSQL and MultiNLI with different rank r. To our surprise, a rank as small as one suffices for adapting both W_q and W_v on these datasets while training W_q alone needs a larger r. We conduct a similar experiment on GPT-2 in Section H.2.

ΔW = A X B

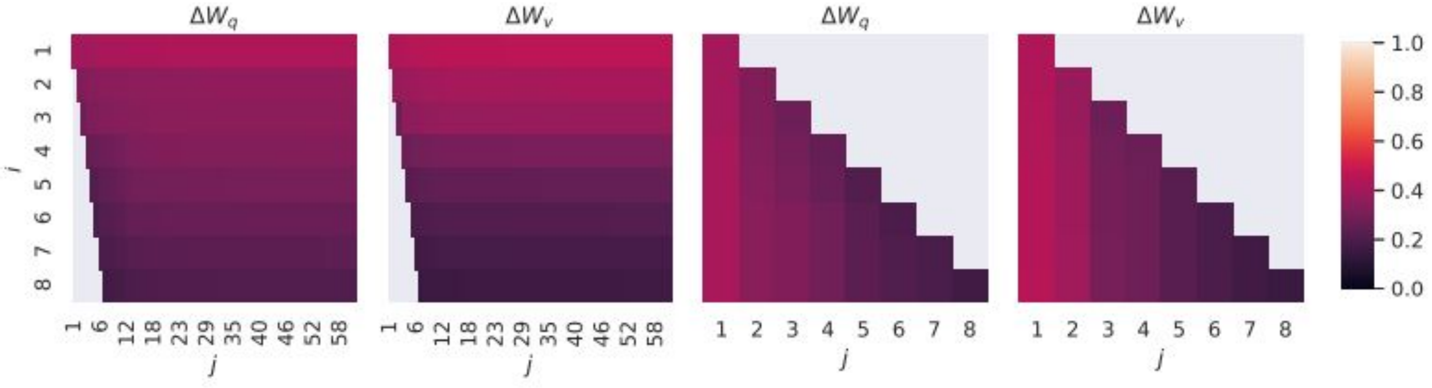
size: A = d X r, B = r X d

How to get optimal r?

Why performed well in downstream missions?

Although the ranks are different, their first few singular vector directions overlap considerably.

They share a dimension and their normalized similarity is greater than 0.5.This means that although the rank is lowthe fitness matrix still performs well in downstream tasks.



$r << d$

“We argue that increasing r does not cover a more meaningful subspace, which suggests that a low-rank adaptation matrix is sufficient.”

We can calculate ΔW in every iterate progress

If ΔW begins to stabilize then reduce r.

Thanks!

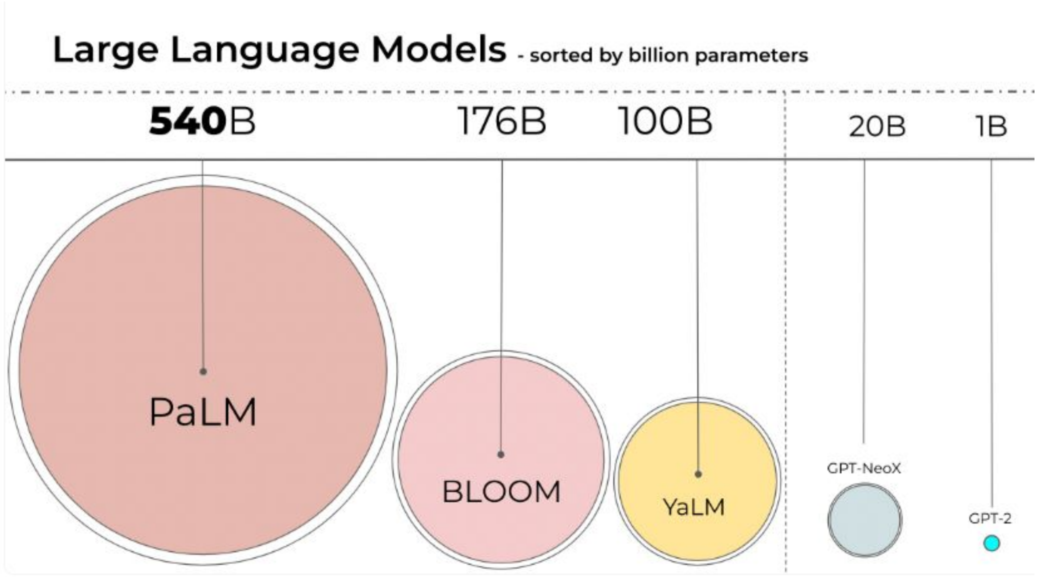
Part II: Quantization

LLM.int8()

8-bit Matrix Multiplication for Transformers at Scale

Stakeholder -- Zeyi Liao

Efficiency status quo



1. Accessible
   a. Handy Repo: lmdeploy(https://github.com/InternLM/lmdeploy)

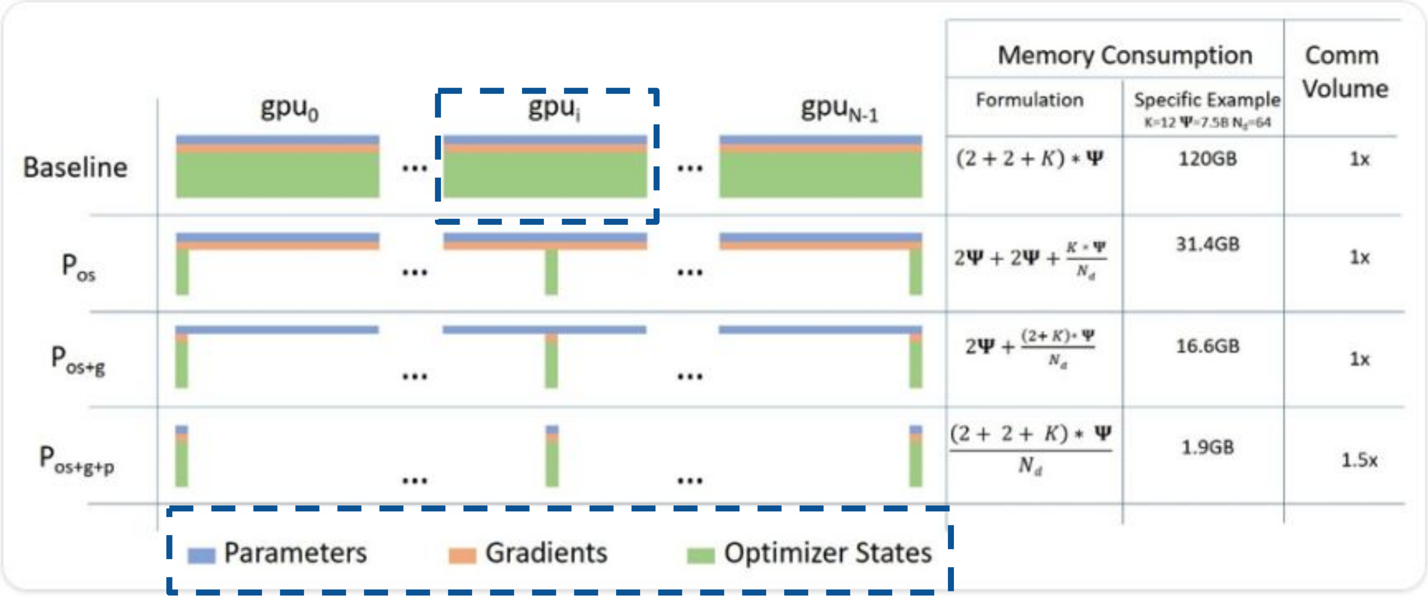
   b. Various Design: Grouped Attention mechanism, Flashed Attention.

2. Time Cost
   a. Speculative Decoding(https://x.com/BeidiChen/status/1826300342985711719).

   Feature: High throughput, low latency etc..

3. Inference-scaling paradigm.

Where does the computation come from?



	Memory Consumption		Comm Volume
	Formulation	Specific Example $K=12$ $\Psi=7.5B$ $N_d=64$	
Baseline	$(2+2+K)*\Psi$	120GB	1x
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

Training:

🟦 Parameters 🟧 Gradients 🟩 Optimizer States

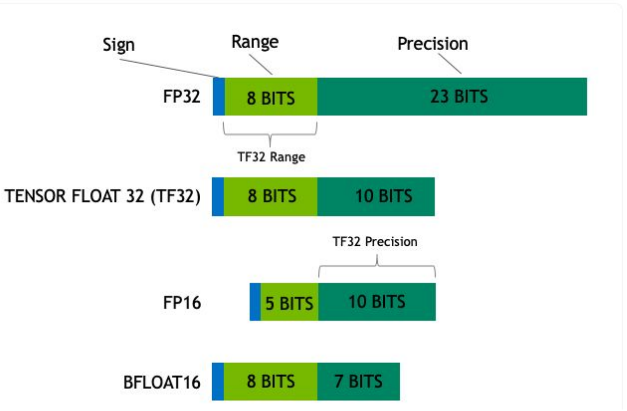
Inference:

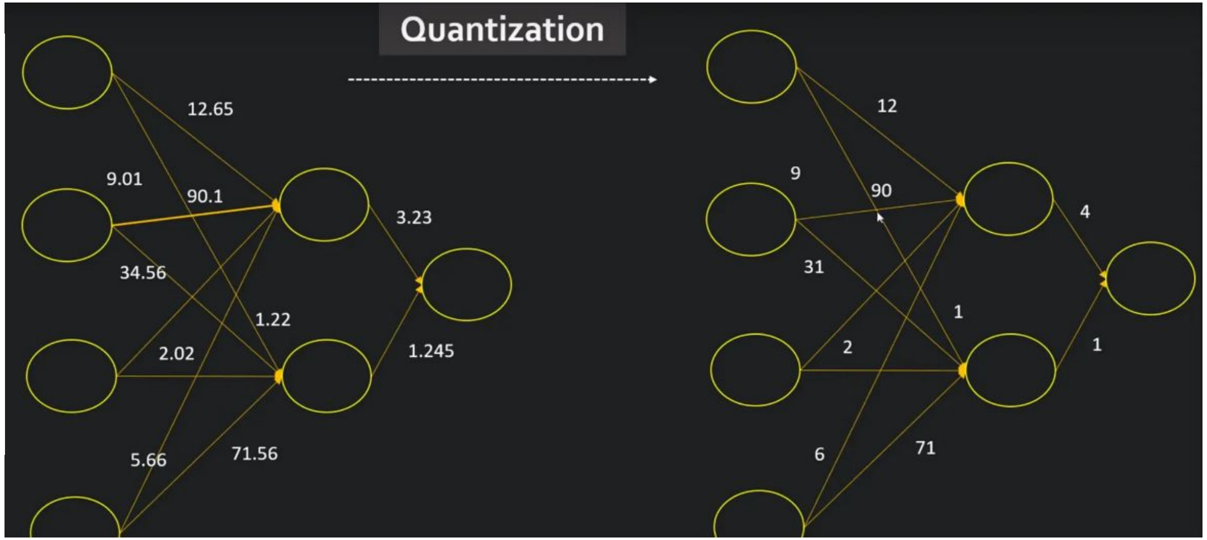
🟦 Parameters 🟩 **Activation (less than computation from parameters)**

Qinghua Liu, https://arxiv.org/abs/2207.11912. Nice survey, check it out!

Let’s dive deeper into the quantization

High Level illustration

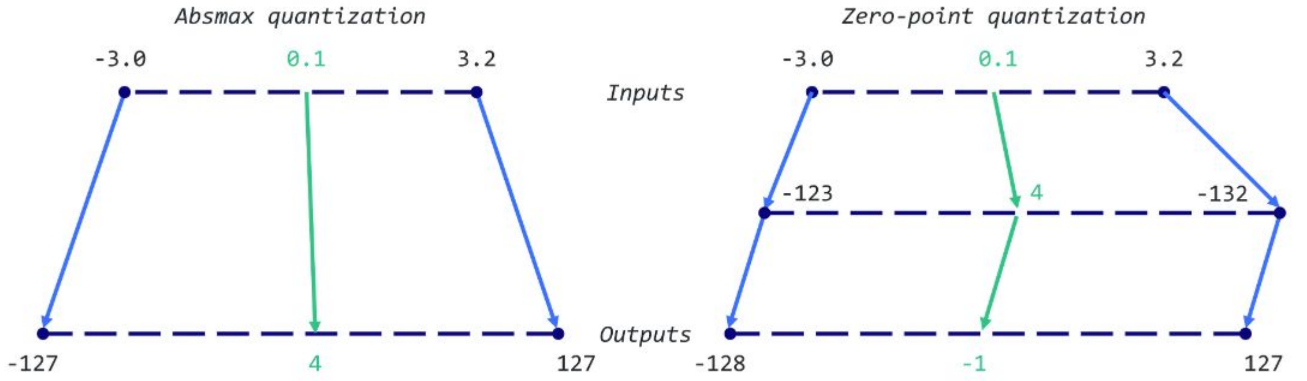




Models in bf16 only need twice less memory compared to using fp32.

Yet, you can not do quantization randomly, you should do it reasonably. Otherwise, model will lose its capability as model per se is just matrix containing numerous numbers.

Absmax / Zero-point quantization



symmetry

asymmetry

$$X_{\text{quant}} = \text{round}\left(\boxed{\frac{127}{\max|\mathbf{X}|}} \cdot \mathbf{X}\right)$$

$$\mathbf{X}_{\text{dequant}} = \frac{\max|\mathbf{X}|}{127} \cdot \mathbf{X}_{\text{quant}}$$

$$\text{scale} = \frac{255}{\max(\mathbf{X}) - \min(\mathbf{X})}$$

$$\text{zeropoint} = -\text{round}(\text{scale} \cdot \min(\mathbf{X})) - 128$$

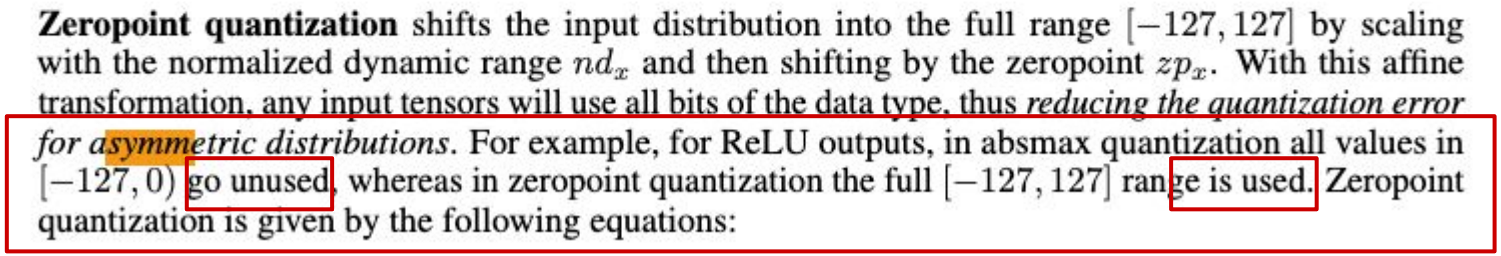
$$\mathbf{X}_{\text{quant}} = \text{round}\left(\boxed{\text{scale}} \cdot \mathbf{X} + \boxed{\text{zeropoint}}\right)$$

$$\mathbf{X}_{\text{dequant}} = \frac{\mathbf{X}_{\text{quant}} - \text{zeropoint}}{\text{scale}}$$

Somehow like **normalization?**

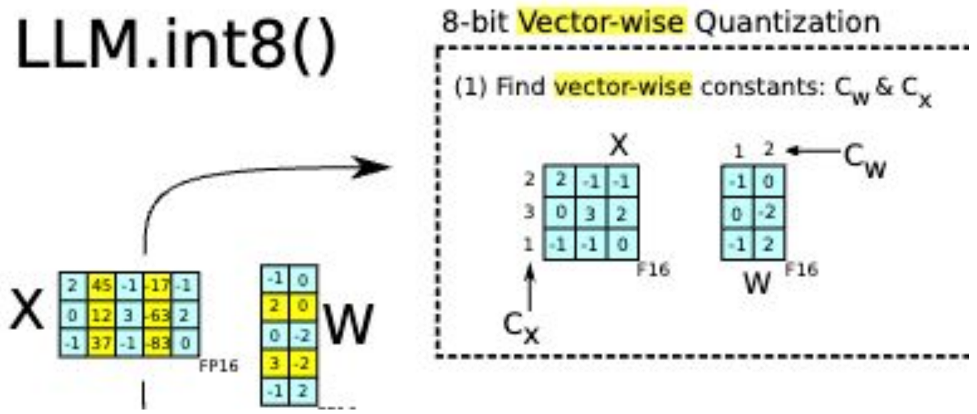
https://medium.com/@sachinsoni600517/introduction-to-model-quantization-4effc7a17000

Absmax / Zero-point quantization



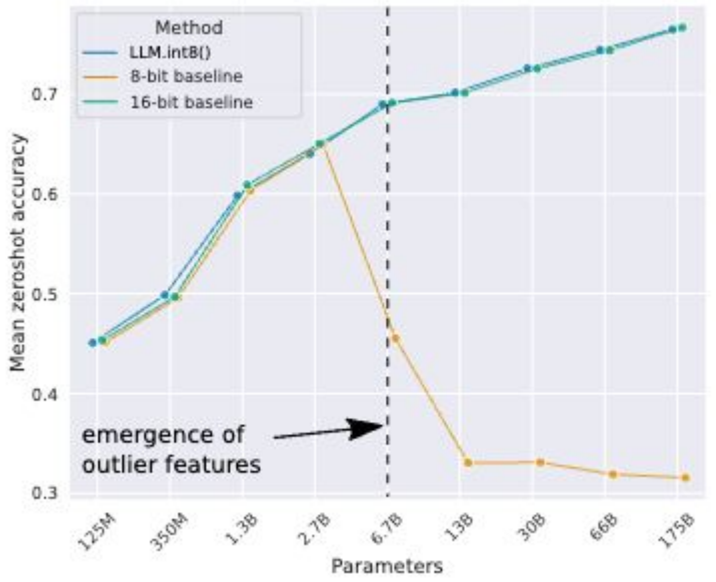
Zeropoint quantization shifts the input distribution into the full range $[-127, 127]$ by scaling with the normalized dynamic range nd_x and then shifting by the zeropoint zp_x. With this affine transformation, any input tensors will use all bits of the data type, thus *reducing the quantization error for asymmetric distributions*. For example, for ReLU outputs, in absmax quantization all values in $[-127, 0)$ go unused, whereas in zeropoint quantization the full $[-127, 127]$ range is used. Zeropoint quantization is given by the following equations:

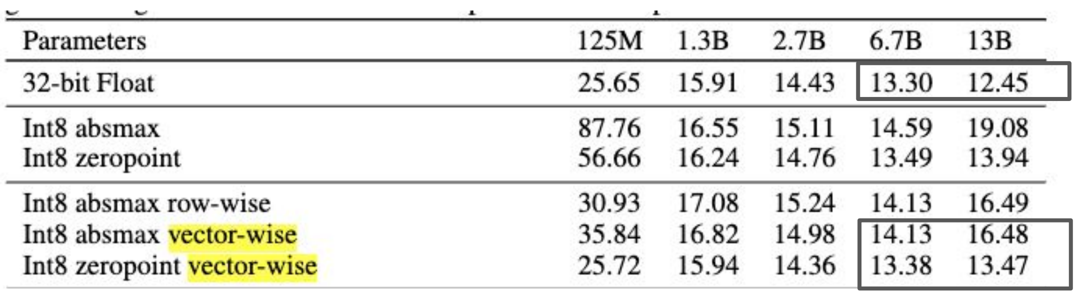
Absmax / Zero-point + **Vector-wise** quantization


LLM.int8()
8-bit Vector-wise Quantization
(1) Find vector-wise constants: C_w & C_x
X
W
FP16
F16
C_w
C_x

1. View matrix multiplication as a sequence of independent inner products.
2. Compute scaling factor for each **row** of **X** and each **column** of **W**.
   a. Robust to outlier.

Performance of quantization





Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vector-wise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vector-wise	25.72	15.94	14.36	13.38	13.47

Takes:
Vector-wise quantization is helpful but insufficient.

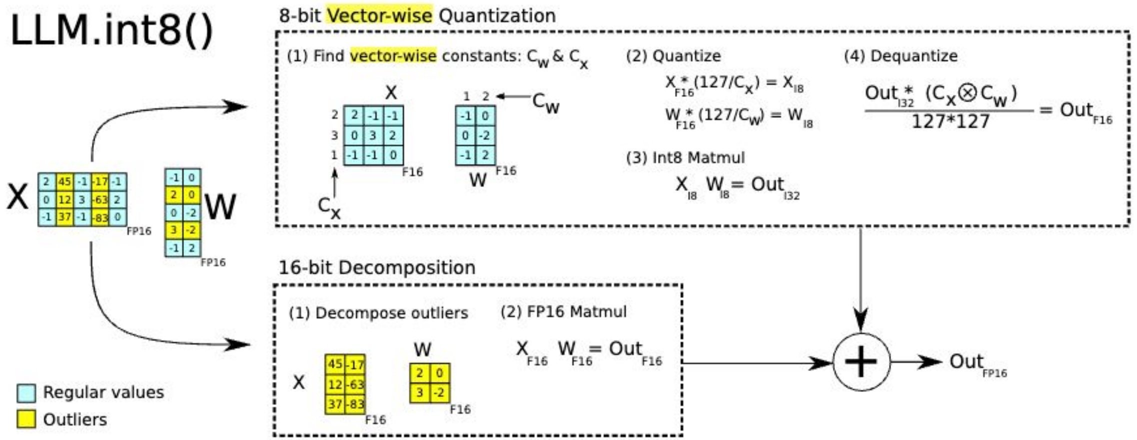
Why fail? Outlier features!

They **empirically** find and define outlier features.

They **empirically** find that certain calculation need more precision beyond Int8.

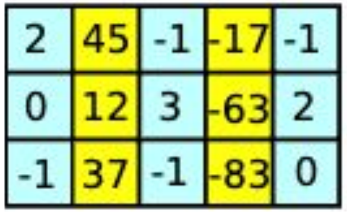
They **empirically** show that performance no more degrade!

Luckily, they find that outlier features is sparse and systematic in practice, thus LLM.int8() only need minor additional overhead compared to pure Int8 quantization



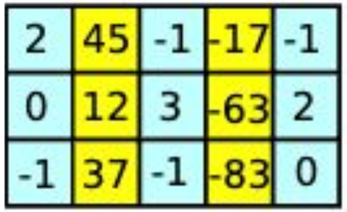
How Define Outlier features

Layer 0



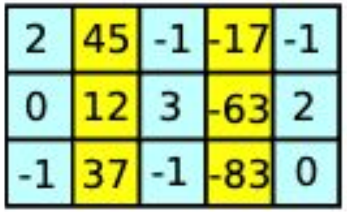
2	45	-1	-17	-1
0	12	3	-63	2
-1	37	-1	-83	0

Layer 1



2	45	-1	-17	-1
0	12	3	-63	2
-1	37	-1	-83	0

Layer 2



2	45	-1	-17	-1
0	12	3	-63	2
-1	37	-1	-83	0

NOTE:
BTW, the feature here just means the certain dimension, not eigenvector after SVD.

Find the dimension

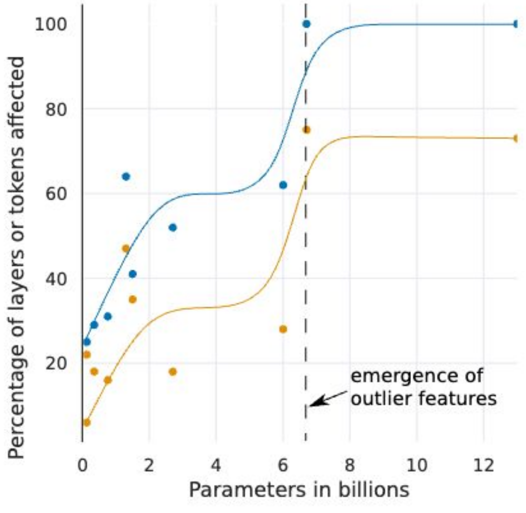
a. Find dimensions containing value > 6.
b. Same feature appear in at least 25% **layers** within transformer.
c. Same feature appear in at least 6% of all **sequence**.

Again, the defining the threshold is sort of heuristic, but it works, then it is what it is.

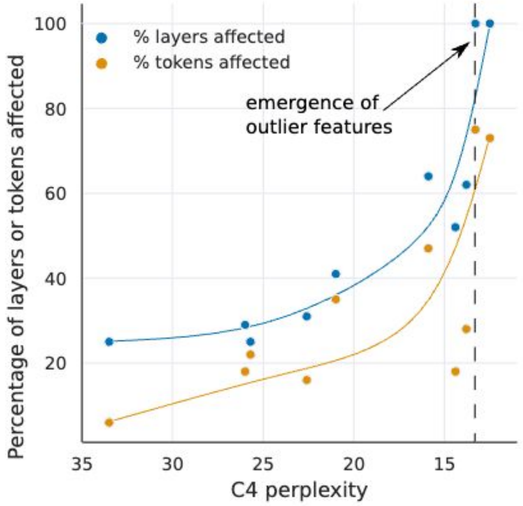
Prove the existence of outlier features by some experiments

Rigorous setting!

4 models from OpenAI, 5 models from MetaAI, 1 from EleutherAI, 2 inference framework: Fairseq and huggingface.



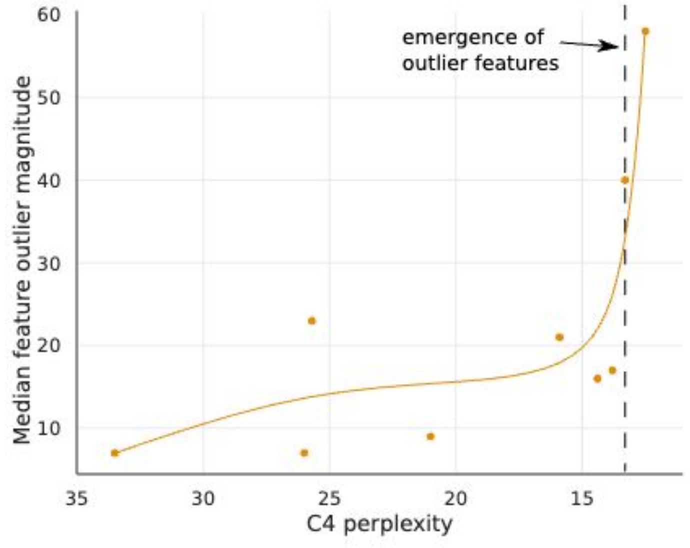
(a)



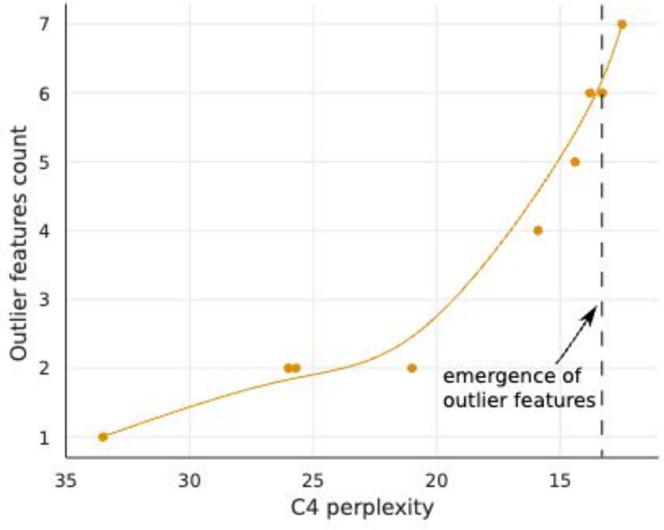
(b)

1. Yeah, larger models have more outlier features.
2. Maybe not because of the mere model size, but the perplexity (somehow correlated with model size) is the determining factor.

Prove the existence of outlier features by some experiments

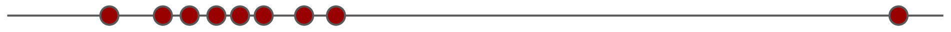


(a)

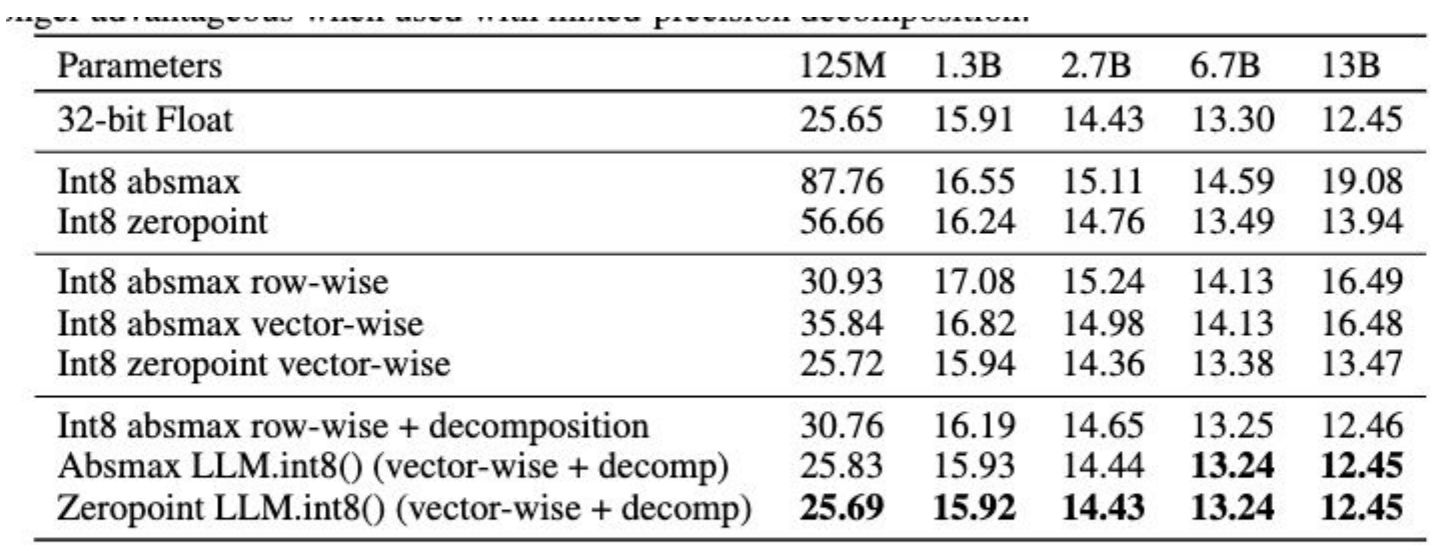


(b)

Median feature is too **large**!, So that quantization doesn't work. Recall that scale is related the **extreme** values.



Performance of Int8.LLM



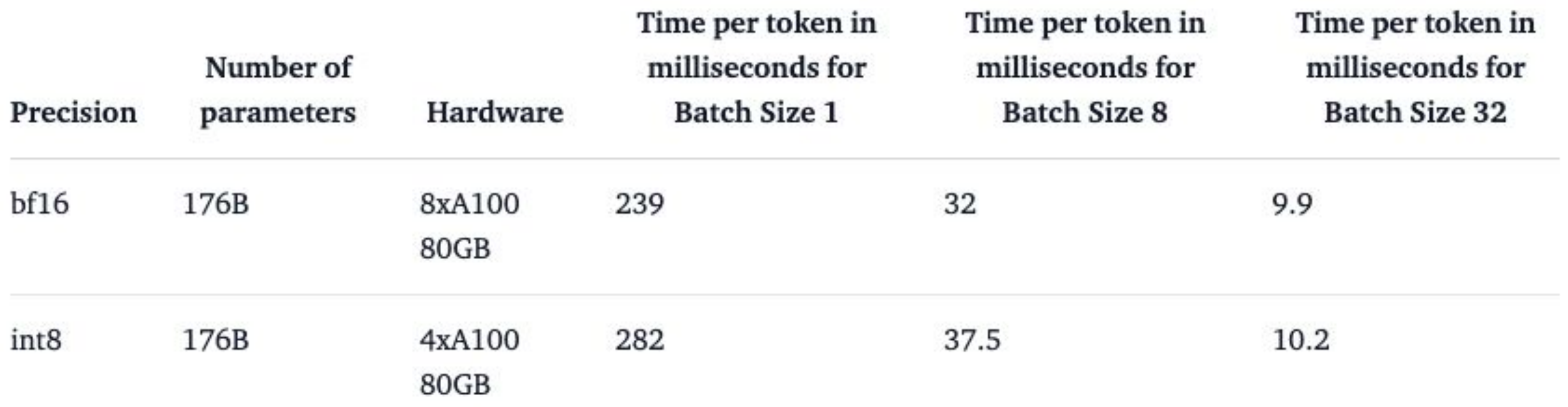
nger advantageous when used with mixed precision decomposition.

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vector-wise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vector-wise	25.72	15.94	14.36	13.38	13.47
Int8 absmax row-wise + decomposition	30.76	16.19	14.65	13.25	12.46
Absmax LLM.int8() (vector-wise + decomp)	25.83	15.93	14.44	**13.24**	**12.45**
Zeropoint LLM.int8() (vector-wise + decomp)	**25.69**	**15.92**	**14.43**	**13.24**	**12.45**

Takes:

1. Zeropoint is better than Absmax in this LLM context, due to its attribute of being asymmetry.
2. When model size reachs to a certain point, even zeropoint with vector-wise quantization can not handle the extreme magnitude well.
3. Using separate precision makes the performance great again.

Drawback? Yes, additional overhead



Precision	Number of parameters	Hardware	Time per token in milliseconds for Batch Size 1	Time per token in milliseconds for Batch Size 8	Time per token in milliseconds for Batch Size 32
bf16	176B	8xA100 80GB	239	32	9.9
int8	176B	4xA100 80GB	282	37.5	10.2

Acceptable but not ideal, esp. for production level use.

https://huggingface.co/blog/hf-bitsandbytes-integration

LLM.int8() - Review

By Mona Gandhi - 09/16/24

Summary

- **Objective**: Reduce the memory for inference while retaining full precision.
- **Method**:
  - Developed procedure for Int8 matrix multiplication.
  - Vector-wise Quantization.
  - Mixed-precision Decomposition Scheme – for outliers.
- Show that by using LLM.int8(), they can perform inference in LLMs with up to 175B params w/o performance degradation.

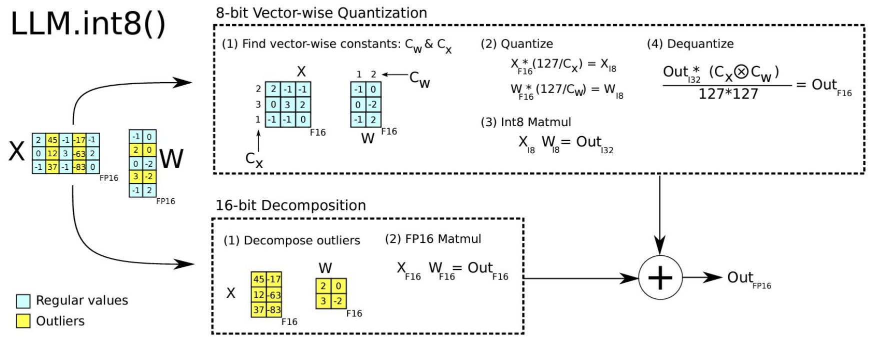

LLM.int8()
8-bit Vector-wise Quantization
(1) Find vector-wise constants: C_w & C_x
(2) Quantize
$X_{F16} * (127/C_x) = X_{I8}$
$W_{F16} * (127/C_w) = W_{I8}$
(3) Int8 Matmul
$X_{I8} W_{I8} = Out_{I32}$
(4) Dequantize
$\frac{Out_{I32} * (C_x \otimes C_w)}{127*127} = Out_{F16}$
16-bit Decomposition
(1) Decompose outliers
(2) FP16 Matmul
$X_{F16} W_{F16} = Out_{F16}$
Out_{FP16}
Regular values
Outliers

Figure 2: Schematic of LLM.int8(). Given 16-bit floating-point inputs $\mathbf{X}_{f16}$ and weights $\mathbf{W}_{f16}$, the features and weights are decomposed into sub-matrices of large magnitude features and other values. The outlier feature matrices are multiplied in 16-bit. All other values are multiplied in 8-bit. We perform 8-bit vector-wise multiplication by scaling by row and column-wise absolute maximum of $\mathbf{C}_x$ and $\mathbf{C}_w$ and then quantizing the outputs to Int8. The Int32 matrix multiplication outputs $\mathbf{Out}_{i32}$ are dequantization by the outer product of the normalization constants $\mathbf{C}_x \otimes \mathbf{C}_w$. Finally, both outlier and regular outputs are accumulated in 16-bit floating point outputs.

🔎: Mona Gandhi

Reviewer Comments

- Making huge models available to use with fewer resources.
- Addresses outlier issues, performs experiments to show the importance of their decomposition method.

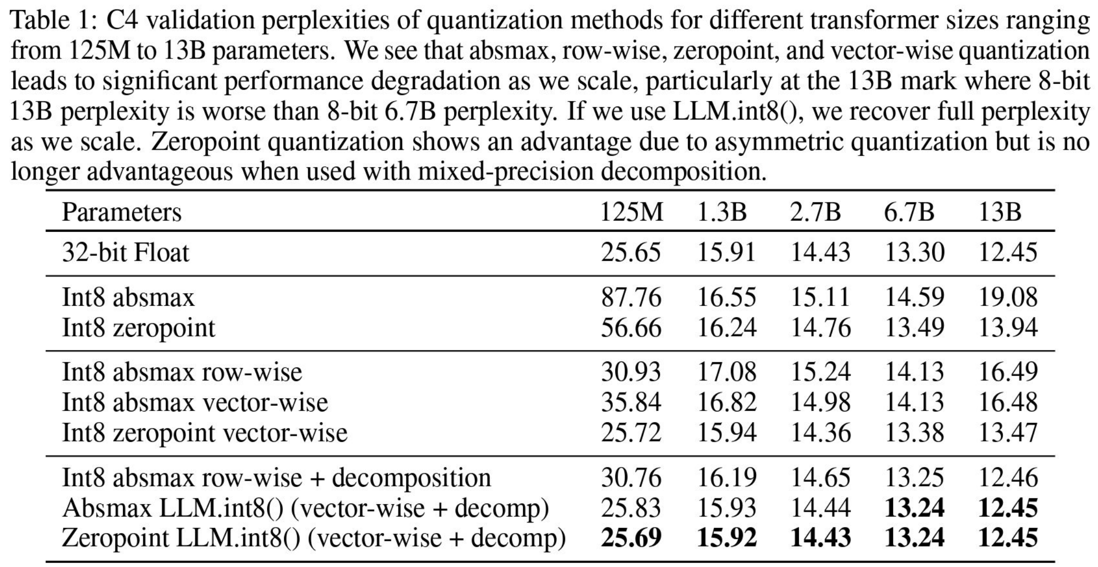


Table 1: C4 validation perplexities of quantization methods for different transformer sizes ranging from 125M to 13B parameters. We see that absmax, row-wise, zeropoint, and vector-wise quantization leads to significant performance degradation as we scale, particularly at the 13B mark where 8-bit 13B perplexity is worse than 8-bit 6.7B perplexity. If we use LLM.int8(), we recover full perplexity as we scale. Zeropoint quantization shows an advantage due to asymmetric quantization but is no longer advantageous when used with mixed-precision decomposition.

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vector-wise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vector-wise	25.72	15.94	14.36	13.38	13.47
Int8 absmax row-wise + decomposition	30.76	16.19	14.65	13.25	12.46
Absmax LLM.int8() (vector-wise + decomp)	25.83	15.93	14.44	**13.24**	**12.45**
Zeropoint LLM.int8() (vector-wise + decomp)	**25.69**	**15.92**	**14.43**	**13.24**	**12.45**

🔎: Mona Gandhi

Reviewer Comments

- Why is the threshold set to 6?
- With Mixed-Precision Decomposition, zero point ~ absolute maximum, however vector-wise still has an edge!

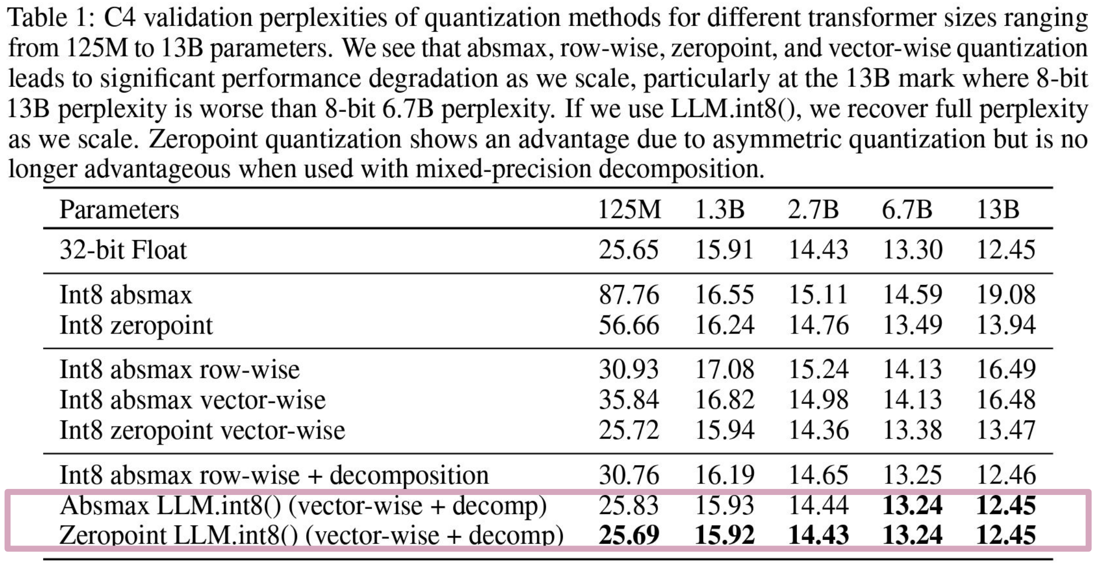


Table 1: C4 validation perplexities of quantization methods for different transformer sizes ranging from 125M to 13B parameters. We see that absmax, row-wise, zeropoint, and vector-wise quantization leads to significant performance degradation as we scale, particularly at the 13B mark where 8-bit 13B perplexity is worse than 8-bit 6.7B perplexity. If we use LLM.int8(), we recover full perplexity as we scale. Zeropoint quantization shows an advantage due to asymmetric quantization but is no longer advantageous when used with mixed-precision decomposition.

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vector-wise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vector-wise	25.72	15.94	14.36	13.38	13.47
Int8 absmax row-wise + decomposition	30.76	16.19	14.65	13.25	12.46
Absmax LLM.int8() (vector-wise + decomp)	25.83	15.93	14.44	**13.24**	**12.45**
Zeropoint LLM.int8() (vector-wise + decomp)	**25.69**	**15.92**	**14.43**	**13.24**	**12.45**

🔎: Mona Gandhi

Reviewer Comments

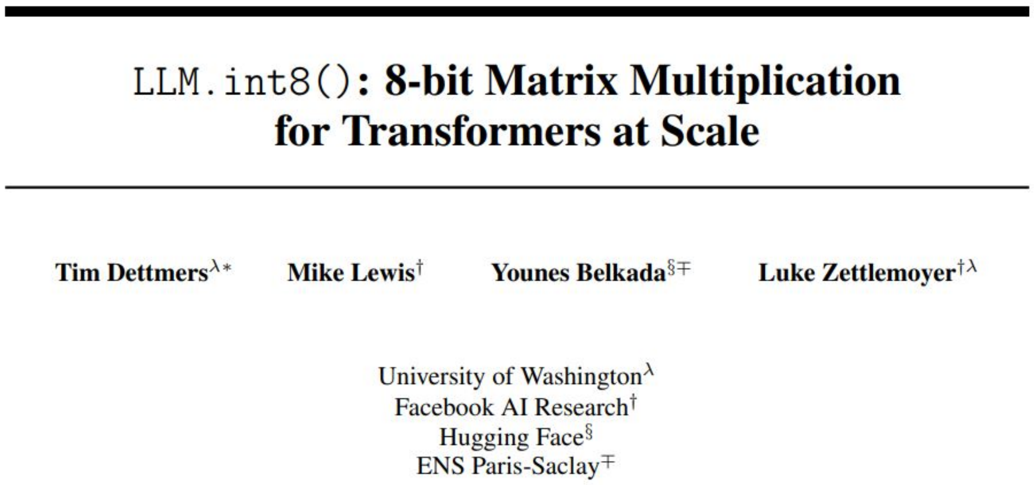
- Addressed in limitations: tried only on int8(), what happens at larger scale is unknown, does not focus on training and fine tuning.
- Why perplexity? And not other metrics for evaluation?
- Maybe try reducing the size of the weight matrix, ignoring insignificant weights – pruning?

Reviewer Comments

- Has very significant broader impact!
- Will certainly be useful for academic institutions specially.
- Would be useful for having LLMs on smaller mobile devices, accessible to all easily.

🔎: Mona Gandhi

Archaeologist



LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

Tim Dettmers[λ*] Mike Lewis[†] Younes Belkada[§∓] Luke Zettlemoyer[†λ]

University of Washington[λ]
Facebook AI Research[†]
Hugging Face[§]
ENS Paris-Saclay[∓]

How was LLM.int8() inspired by previous work?

Hanane Moussa

Archaeologist

LLM.int8()

Dettmers et al. develop LLM.int8(), a two step quantization procedure:

- vector wise-**quantization** for most features using 8-bit matrix multiplication (99.9%)
- mixed-precision decomposition for the **emergent outliers** using 16-bit (0.1%)

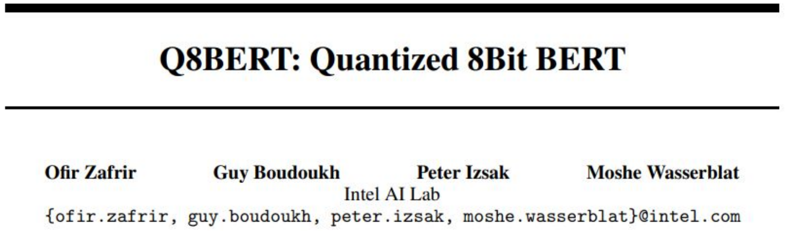
Allows inference in LLMs with up to 175B parameters without any performance degradation.

Archaeologist

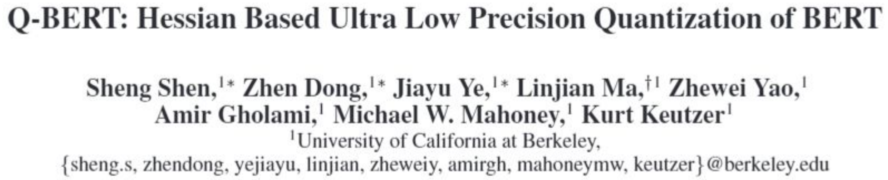
Quantization of BERT models

- Q8BERT (2019) quantizes the model's weights and activations from 32-bit floating point to 8-bit integer.
- 4x reduction in model size.
- Some performance loss.

- Q-BERT (2020) quantizes model weights to 2-bit precision.
- Uses a mixed-precision approach using a Hessian Matrix to determine which parts of the model are more sensitive to quantization
- Some accuracy loss.


Q8BERT: Quantized 8Bit BERT

Ofir Zafrir **Guy Boudoukh** **Peter Izsak** **Moshe Wasserblat**
Intel AI Lab
{ofir.zafrir, guy.boudoukh, peter.izsak, moshe.wasserblat}@intel.com

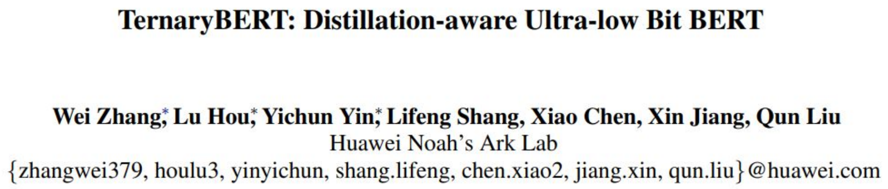

Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT

Sheng Shen,[1*] Zhen Dong,[1*] Jiayu Ye,[1*] Linjian Ma,[†1] Zhewei Yao,[1]
Amir Gholami,[1] Michael W. Mahoney,[1] Kurt Keutzer[1]
[1]University of California at Berkeley,
{sheng.s, zhendong, yejiayu, linjian, zheweiy, amirgh, mahoneymw, keutzer}@berkeley.edu

Quantization of BERT models

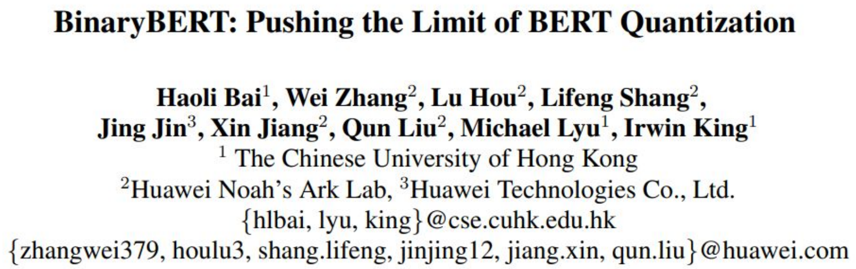
- TernaryBERT (2020) quantizes model weights to ternary values (-1, 0, 1).
- Model performance is improved using knowledge distillation from full-precision BERT model

- BinaryBERT (2021) further quantizes the model to binary values (1, -1)
- Also uses a knowledge-distillation approach.



TernaryBERT: Distillation-aware Ultra-low Bit BERT

Wei Zhang*, Lu Hou*, Yichun Yin*, Lifeng Shang, Xiao Chen, Xin Jiang, Qun Liu
Huawei Noah's Ark Lab
{zhangwei379, houlu3, yinyichun, shang.lifeng, chen.xiao2, jiang.xin, qun.liu}@huawei.com



BinaryBERT: Pushing the Limit of BERT Quantization

Haoli Bai[1], Wei Zhang[2], Lu Hou[2], Lifeng Shang[2], Jing Jin[3], Xin Jiang[2], Qun Liu[2], Michael Lyu[1], Irwin King[1]
[1] The Chinese University of Hong Kong
[2]Huawei Noah's Ark Lab, [3]Huawei Technologies Co., Ltd.
{hlbai, lyu, king}@cse.cuhk.edu.hk
{zhangwei379, houlu3, shang.lifeng, jinjing12, jiang.xin, qun.liu}@huawei.com

Archaeologist

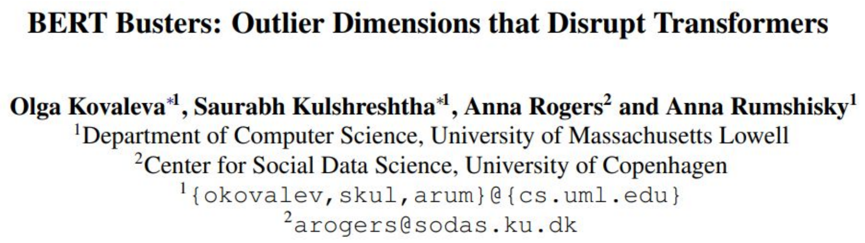
Limitations

- Q8BERT, Q-BERT, TernaryBERT, and BinaryBERT all require quantization aware training
- Only work for models with less than 1-Billion parameters (up to 340M for BERT Large)

Archaeologist

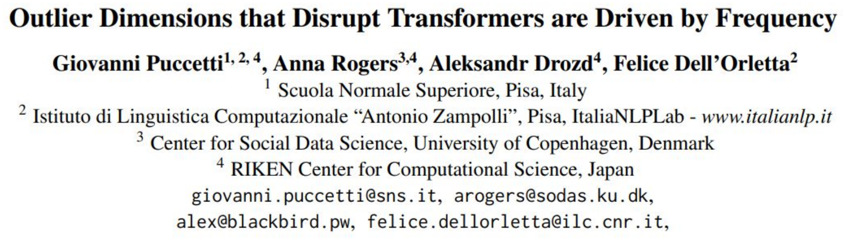
Outlier features

- Pretrained transformer models are not entirely robust to pruning; they are fragile to the removal of a very small number of features in the layer outputs
- “Disabling only 48 out of 110M parameters in BERT-base drops its performance by nearly 30% on MNLI” - Puccetti et al.



BERT Busters: Outlier Dimensions that Disrupt Transformers

Olga Kovaleva[*1], **Saurabh Kulshreshtha**[*1], **Anna Rogers**[2] **and Anna Rumshisky**[1]
[1]Department of Computer Science, University of Massachusetts Lowell
[2]Center for Social Data Science, University of Copenhagen
[1]{okovalev,skul,arum}@{cs.uml.edu}
[2]arogers@sodas.ku.dk



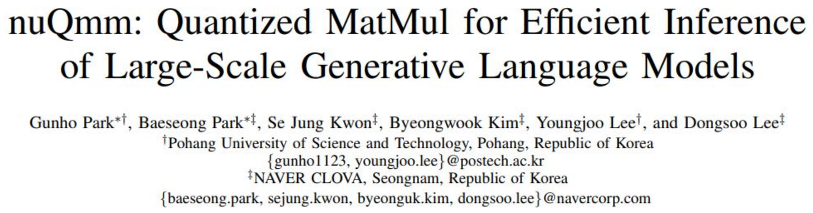
Outlier Dimensions that Disrupt Transformers are Driven by Frequency

Giovanni Puccetti[1, 2, 4], **Anna Rogers**[3,4], **Aleksandr Drozd**[4], **Felice Dell’Orletta**[2]
[1] Scuola Normale Superiore, Pisa, Italy
[2] Istituto di Linguistica Computazionale “Antonio Zampolli”, Pisa, ItaliaNLPLab - *www.italianlp.it*
[3] Center for Social Data Science, University of Copenhagen, Denmark
[4] RIKEN Center for Computational Science, Japan
giovanni.puccetti@sns.it, arogers@sodas.ku.dk,
alex@blackbird.pw, felice.dellorletta@ilc.cnr.it,

Archaeologist

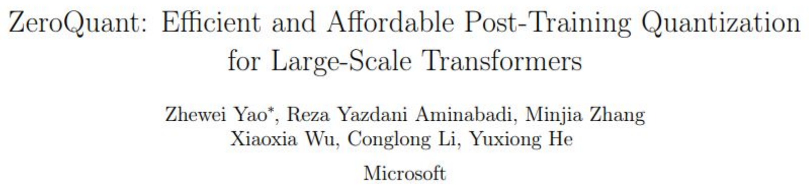
Parallel work: nuQmm and ZeroQuant

- Both methods use group-wise quantization which offers greater granularity / precision
- Require custom CUDA kernels
- Only on models of 2.7B and 20B parameters respectively
- Focus on accelerating inference and reducing memory footprint



nuQmm: Quantized MatMul for Efficient Inference of Large-Scale Generative Language Models

Gunho Park*†, Baeseong Park*‡, Se Jung Kwon‡, Byeongwook Kim‡, Youngjoo Lee†, and Dongsoo Lee‡
†Pohang University of Science and Technology, Pohang, Republic of Korea
{gunho1123, youngjoo.lee}@postech.ac.kr
‡NAVER CLOVA, Seongnam, Republic of Korea
{baeseong.park, sejung.kwon, byeonguk.kim, dongsoo.lee}@navercorp.com



ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers

Zhewei Yao*, Reza Yazdani Aminabadi, Minjia Zhang
Xiaoxia Wu, Conglong Li, Yuxiong He
Microsoft

Advanced NLP

LLM.int8()-visionary

Junjie Zhang

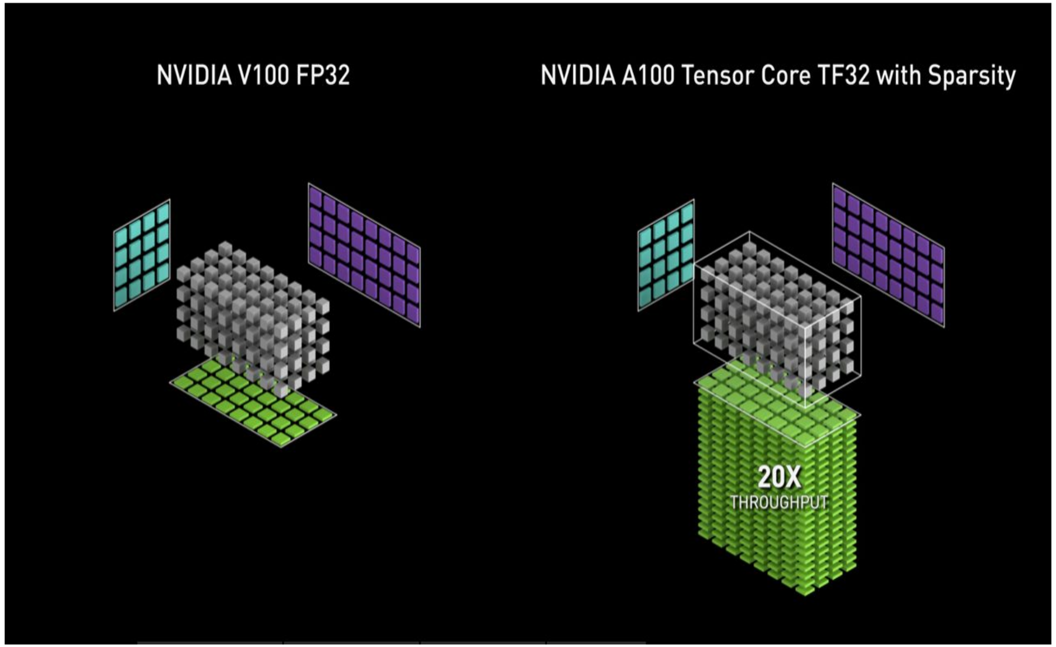
THE OHIO STATE UNIVERSITY

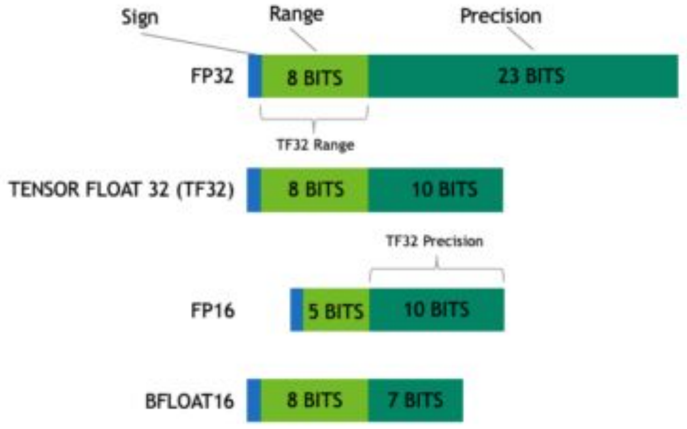
Develop new GPUs

TensorFloat-32 in the A100 GPU Accelerates AI Training, HPC up to 20x

NVIDIA's Ampere architecture with TF32 speeds single-precision work, maintaining accuracy and using no new code.

May 14, 2020 by Paresh Kharya





TF32 strikes a balance that delivers performance with range and accuracy.

Source:https://blogs.nvidia.com/blog/tensorfloat-32-precision
-format/

Develop new math for matrix decomposition and composition

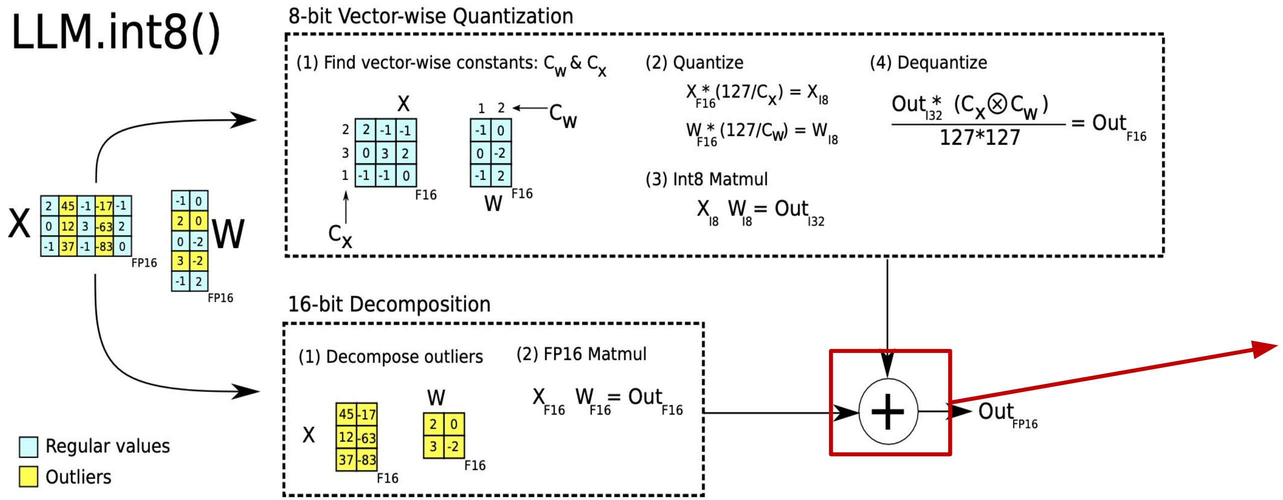

LLM.int8()
8-bit Vector-wise Quantization
(1) Find vector-wise constants: C_w & C_x
(2) Quantize
$X_{F16} * (127/C_x) = X_{I8}$
$W_{F16} * (127/C_w) = W_{I8}$
(3) Int8 Matmul
$X_{I8} W_{I8} = Out_{I32}$
(4) Dequantize
$\frac{Out_{I32} * (C_x \otimes C_w)}{127*127} = Out_{F16}$
16-bit Decomposition
(1) Decompose outliers
(2) FP16 Matmul
$X_{F16} W_{F16} = Out_{F16}$
Out_{FP16}
Regular values
Outliers

Figure 2: Schematic of LLM.int8(). Given 16-bit floating-point inputs $\mathbf{X}_{f16}$ and weights $\mathbf{W}_{f16}$, the features and weights are decomposed into sub-matrices of large magnitude features and other values. The outlier feature matrices are multiplied in 16-bit. All other values are multiplied in 8-bit. We perform 8-bit vector-wise multiplication by scaling by row and column-wise absolute maximum of $\mathbf{C}_x$ and $\mathbf{C}_w$ and then quantizing the outputs to Int8. The Int32 matrix multiplication outputs $\mathbf{Out}_{i32}$ are dequantization by the outer product of the normalization constants $\mathbf{C}_x \otimes \mathbf{C}_w$. Finally, both outlier and regular outputs are accumulated in 16-bit floating point outputs.

The extra addition operations will increase the inference time.

Source: LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

Apply to IoT devices

Source: Google Images

Model Interpretability

“

“ These outliers are highly systematic: at the 6.7B scale, 150,000 outliers occur per sequence, but they are concentrated in only 6 feature dimensions across the entire transformer. ”

Some others

- int.4() ?
- Use the same quantization method during training