Distributed Optimization for Machine Learning

Lecture 23 - Pre-training and Fine-Tuning LLMs

ECE 5290/7290 & ORIE 5290

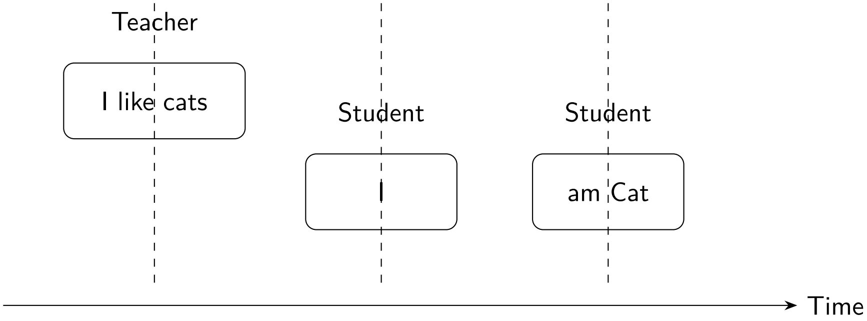
Tianyi Chen

School of Electrical and Computer Engineering
Cornell Tech, Cornell University

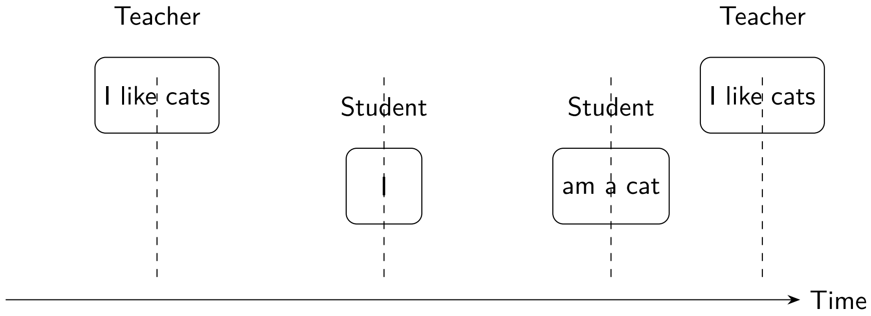
November 24, 2025

Table of Contents

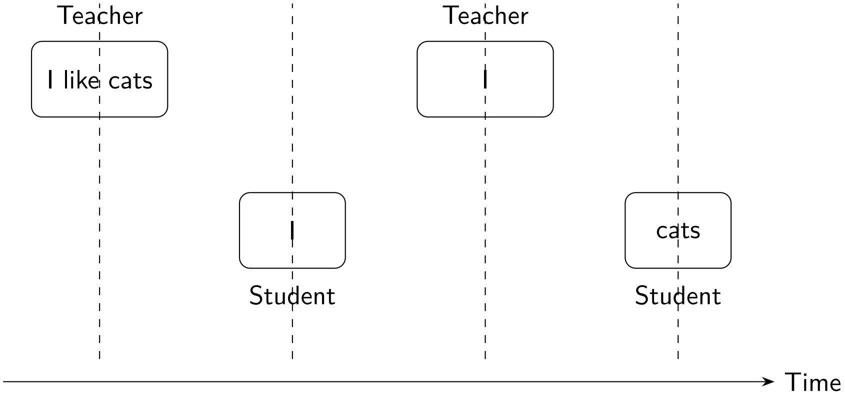
Training without teacher forcing



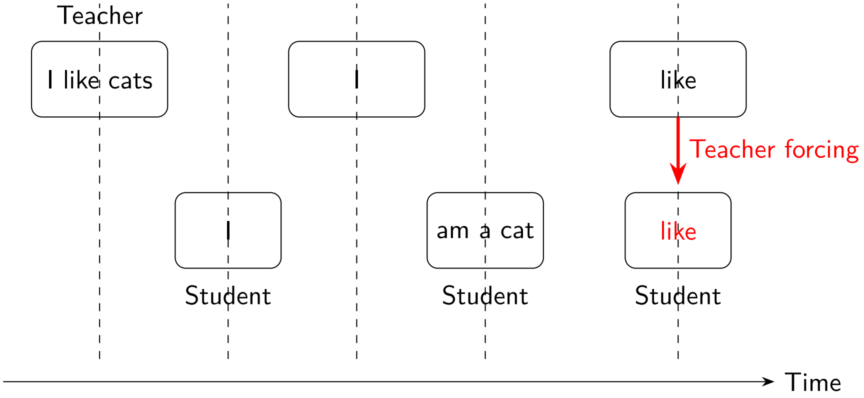
Training without teacher forcing



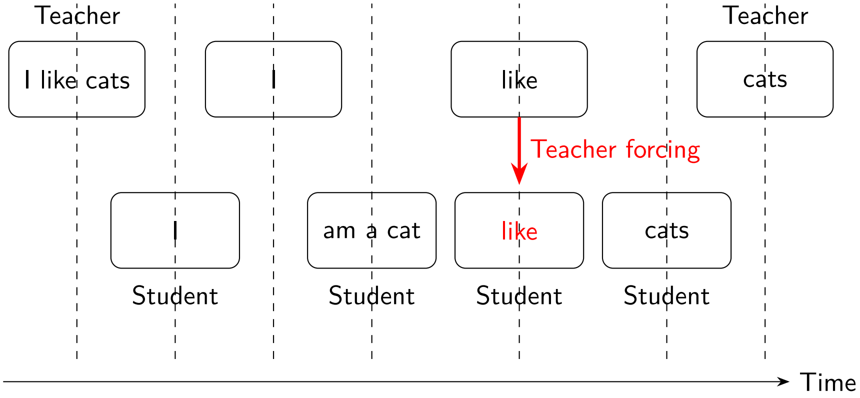
Training with teacher forcing



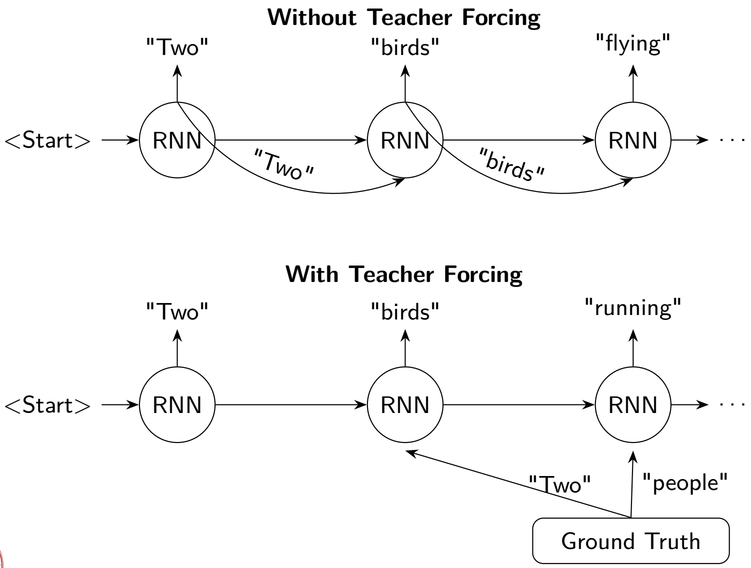
Training with teacher forcing



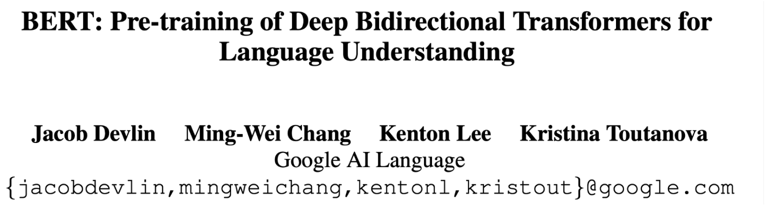
Training with teacher forcing



With and without teacher forcing



BERT: Bidirectional encoder-based transformers



BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
`{jacobdevlin,mingweichang,kentonl,kristout}@google.com`

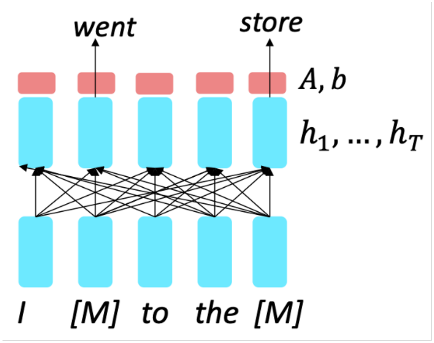
Devlin et al. (2018) proposed the **Masked LM** objective and **released the weights** of a pretrained Transformer encoder - BERT.

BERT: Bidirectional encoder-based transformers

- Encoders get **bidirectional context** - can condition on both left and right.
- How do we pre-train them to build strong representations?

Roughly, BERT pre-trains the encoder of the Transformer.

It replaces a fraction of tokens with a special `[MASK]` symbol and predicts these hidden tokens.



Additional BERT details

Two models were released:

- **BERT-base**: 12 layers, 768-dim hidden, 12 heads, 110M params
- **BERT-large**: 24 layers, 1024-dim hidden, 16 heads, 340M params

Trained on:

- BooksCorpus (800M words)
- English Wikipedia (2,500M words)

Pretraining is expensive.

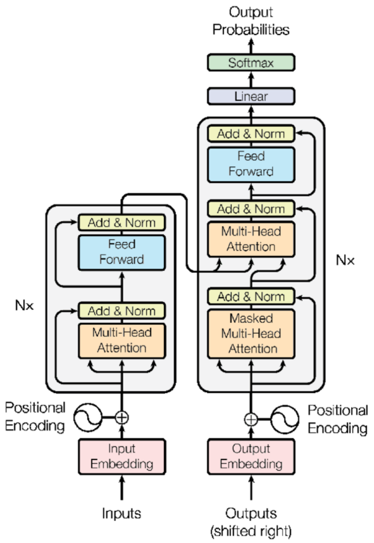
- BERT was trained on 64 TPU chips for 4 days.
- TPUs are specialized tensor acceleration hardware.

Limitations of encoder-based BERT

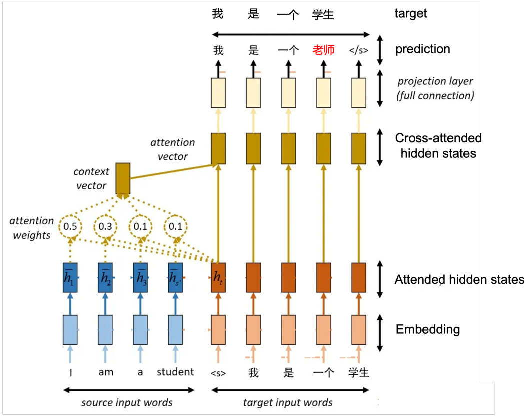
- Pretrained **encoders** such as BERT are excellent for **information extraction** tasks.
- However, BERT is not good for **text generation** because it cannot condition on future tokens (bidirectional masking breaks autoregression).
- For tasks requiring sequence generation, we must use a **pretrained decoder** (e.g., GPT). Encoder-only models do not naturally support left-to-right generation.

Training of transformer models

- Teacher forcing is used when training transformer
- To avoid error accumulation
- To accelerate training via parallel computing endowed by masked attention



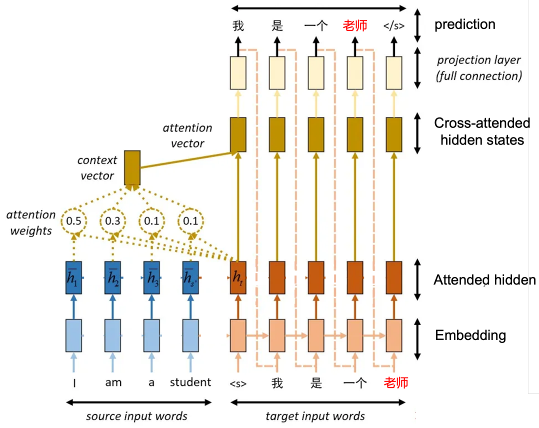
Training transformer with teacher forcing



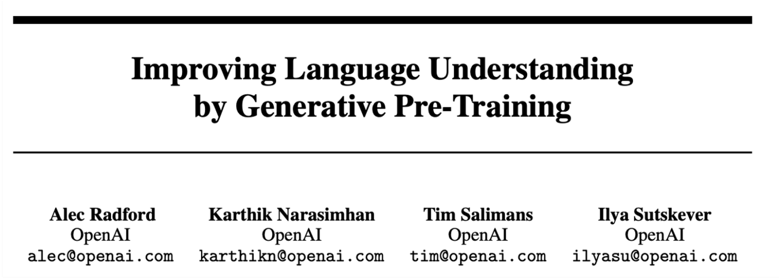
During training, the decoder does **not** use its own previous prediction; instead, it is given the **ground-truth previous token** as the next input.

Inference of transformer models

- During inference, teacher forcing is **not** used, because we do not have the ground truth.
- The model must feed its **own predictions** back as inputs step by step.
- Generation is therefore **sequential** and **cannot be fully parallelized**.



Pretrain decoder-based transformers



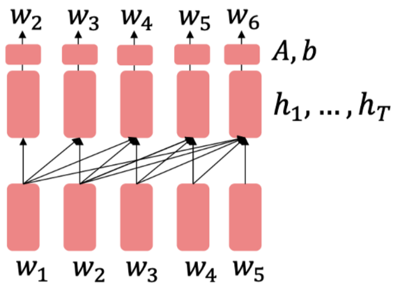
Improving Language Understanding by Generative Pre-Training

Alec Radford OpenAI alec@openai.com
Karthik Narasimhan OpenAI karthikn@openai.com
Tim Salimans OpenAI tim@openai.com
Ilya Sutskever OpenAI ilyasu@openai.com

The Generative Pretrained Transformer (GPT) **Radford et al., 2018** showed the success of pretraining a decoder.

GPT: Generative Pretrained Transformer

- **Decoders** = language models (autoregressive).
- Good for generation; cannot look at future tokens.
- All major pretrained LLMs today (GPT-family) are **decoders**.



Language modeling is much harder than filling in the placeholder, the target of BERT. This explains that why BERT performs better than GPT.

Performance of GPT and BERT

GPT-1 is the first model that significantly improved performance over many tasks. But, BERT surpassed GPT-1 due to bidirectional context.

System	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
Pre-OpenAI SOTA	80.6	80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7
BiLSTM+ELMo+Attn	76.4	76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8
OpenAI GPT	82.1	81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0
BERT$_{\text{BASE}}$	84.6	83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4
BERT$_{\text{LARGE}}$	**86.7**	**85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**

Formalizing language modeling

- **Goal:** Learn the probability distribution of real-world text.
- Let $\mathcal{V}$ be the vocabulary (e.g., all English words).
- A sequence $x = (x_1, x_2, \ldots, x_T)$ consists of tokens $x_t \in \mathcal{V}$.
- By the chain rule of probability, the joint probability of a sequence is:

$$P(x) = \prod_{t=1}^{T} P(x_t \mid x_1, \ldots, x_{t-1})$$

- **Model:** We approximate this conditional probability using a neural network (Transformer) parameterized by weights θ:

$$P_\theta(x_t \mid x_{<t}) \approx P_{\text{data}}(x_t \mid x_{<t})$$

From conditional probability to cross-entropy

- To train the model, maximize the log-probability of the data:

$$\max_{\theta} \quad \sum_{t} \log P_\theta(x_t \mid x_{<t})$$

- **Connecting to the network output f_θ:** Let $f_\theta(x_{<t})$ be the unnormalized scores (**logits**) output by the Transformer. The probability is the **Softmax** of these logits:

$$P_\theta(x_t \mid x_{<t}) = \mathrm{Softmax}\big(f_\theta(x_{<t})\big)_{x_t}$$

- **Defining the Loss ℓ:** Minimizing the negative log-likelihood is equivalent to minimizing the **Cross-Entropy loss** ℓ:

$$\underbrace{\ell\big(f_\theta(x_{<t}), x_t\big)}_{\text{Loss for step } t} := -\log P_\theta(x_t \mid x_{<t})$$

Pre-training GPT as next-token prediction

- Pre-training of GPT solves a **next-token prediction** problem:

$$\min_{\theta} \ \mathbb{E}_{(x_1,\ldots,x_s)\sim\mathcal{D}} \left[\sum_{t=1}^{s} \ell\big(f_\theta(x_{<t}), x_t\big) \right]$$

- f_θ: decoder-only transformer with parameters θ
- ℓ: cross-entropy loss over the vocabulary
- No explicit labels: the context itself defines the training signal.

Optimization viewpoint of GPT pre-training

- We can write the pre-training problem as:

$$\min_{\theta}\ F(\theta) := \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\ell(f_\theta(x), y)\big]$$

  where x is the prefix and y is the next token.

- In practice, we approximate $F(\theta)$ with mini-batches:

$$g_t = \frac{1}{B}\sum_{i=1}^{B} \nabla_\theta \ell(f_\theta(x^{(i)}), y^{(i)})$$

- Parameter update (generic form):

$$\theta_{t+1} = \theta_t - \eta_t\,\mathcal{P}(g_t)$$

  where $\mathcal{P}$ is the optimizer mapping (SGD, AdamW, etc.).

How GPT pre-training differs from supervised learning

- **Self-supervised labels (no human annotation)**
  - In supervised learning: $y =$ human-provided label.
  - In GPT: $y =$ *next token* in the sequence.
  - The context itself provides the supervision.
- **Autoregressive sequence prediction**
  - Standard tasks predict one output per example.
  - GPT predicts every next token:

$$(x_1 \to x_2),\ (x_1, x_2 \to x_3),\ \ldots$$

- **Teacher forcing during training**
  - Training uses the **true previous token**.
  - Inference must rely on the model's own predictions.
- **Decoder-only, causal Transformer architecture**
  - Masked self-attention prevents access to future tokens.
  - Different from encoder-based supervised models (e.g., BERT).

Workhorse for LLM pre-training: AdamW

- **AdamW** (Loshchilov & Hutter, 2018) maintains first and second moments:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$

  and updates (with a small constant $\varepsilon > 0$):

$$\theta_{t+1} = \theta_t - \eta_t \frac{m_t}{\sqrt{v_t} + \varepsilon} - \eta_t \lambda \theta_t.$$

- Benefits for transformers:
  - Handles ill-conditioned layers (e.g., attention, LayerNorm).
  - Adaptive step sizes per parameter.
  - Weight decay (AdamW) decoupled from gradient scaling.
- Empirically more stable than mini-batch SGD for LLM pre-training.

Memory-efficient alternative: Adafactor

- **Problem with AdamW:** Storing the second moment v_t requires memory proportional to the number of parameters ($O(N)$). For 500B+ models, this is expensive.

- **Adafactor** (Shazeer & Stern, 2018) approximates v_t for weight matrices using a **rank-one factorization**:

$$v_t \approx R_t C_t \quad \text{with} \quad R_t \in \mathbb{R}^m, C_t \in \mathbb{R}^n$$

  where R_t stores row-wise and C_t stores column-wise averages.

- For a parameter matrix of size $m \times n$, AdamW stores mn second-moment values, while Adafactor stores only $m + n$ values.

$$\text{Memory reduction } = \frac{mn}{m+n} \approx \min(m, n) \quad \text{(often a 10,000 savings)}$$

Sign-based momentum alternative: Lion

- **Lion** (Chen et al., 2023) was discovered by Google DeepMind.
- It removes the second moment v_t entirely and updates based on the **sign** of the momentum:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ \theta_{t+1} &= \theta_t - \eta_t \,\mathrm{sign}(m_t) \end{aligned}$$

- **Key differences vs. AdamW:**
  - Uses uniform magnitude updates (via sign).
  - Lower memory usage (no v_t buffer).
  - Often converges faster in wall-clock time due to simplicity.

Momentum orthogonalized alternative: Muon

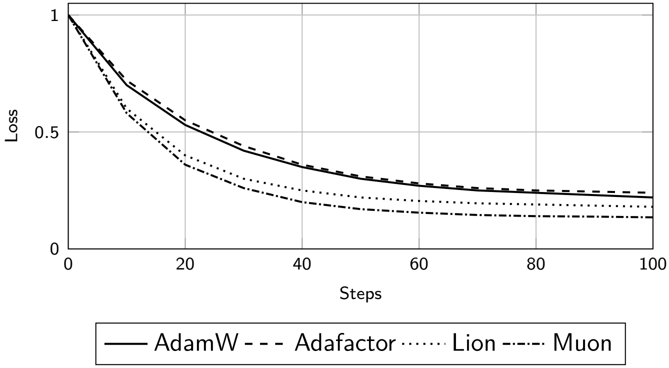
- **Muon** (Jordan et al., 2024) is a new optimizer designed for massive matrix updates (like LLM weights).
- **Key Idea:** Instead of normalizing elements individually (AdamW) or by sign (Lion), Muon **orthogonalizes** the entire momentum matrix

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$\theta_{t+1} = \theta_t - \eta_t \text{NewtonSchulz}(m_t)$$

- **Why Muon works:**
  - It uses $\text{NewtonSchulz}(m_t) \approx m_t(m_t^\top m_t)^{-1/2}$ to project the momentum matrix m_t onto the manifold of orthogonal matrices;
  - Acts like a second-order method (approximating curvature) $H^{-1}g$ without the cost of forming or inverting a Hessian.

Comparison: AdamW vs Adafactor vs Lion vs Muon

- Schematic convergence on a small NN.
- Vertical axis: training loss (lower is better).
- Horizontal axis: optimization steps or wall-clock time.



Stability tricks in LLM optimization

- **Learning rate warm-up**
  - Start with a small η_t, increase over first K steps.
  - Prevents divergence in early training.
- **LR scheduling**
  - Cosine decay after warm-up.
  - Aligns training with fixed compute budget.
- **Gradient clipping**
  - Clip $\|g_t\|$ to a threshold.
  - Protects from rare exploding updates.
- **Mixed precision**
  - Use FP16/BF16 for activations and weights.
  - Requires loss scaling to avoid underflow.

Table of Contents

BERT finetuning

After pretraining, BERT learns powerful token-level representations. We use these as a starting point for downstream tasks.

Example 1: Sentence classification (single-label)

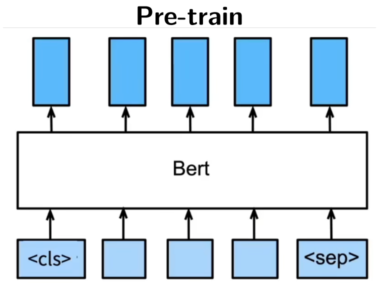
- *Input:* "The movie was absolutely fantastic!"
- *Task:* Predict a single label for the whole sentence
- *Output:* **Positive**

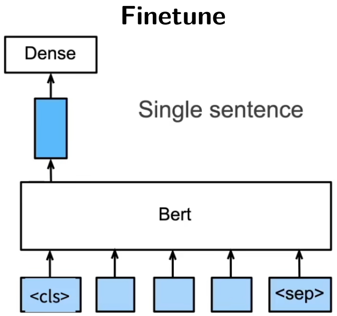
Example 2: Token classification (named entity recognition)

- *Input:* "Barack Obama was born in Hawaii."
- *Task:* Assign a label to each token
- *Token Labels:*

Token	Label
Barack	PERSON
Obama	PERSON
was	O
born	O
in	O
Hawaii	LOCATION

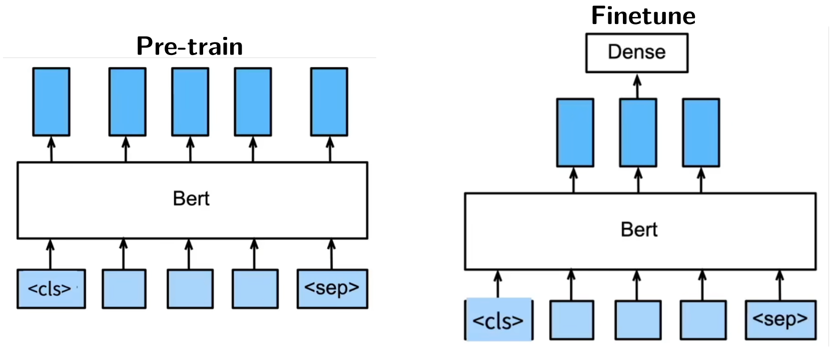
Finetune BERT for sentence classification





Single sentence classification:
use the hidden state of `[CLS]` $\rightarrow$ Dense layer $\rightarrow$ Softmax.

Finetune BERT for entity recognition

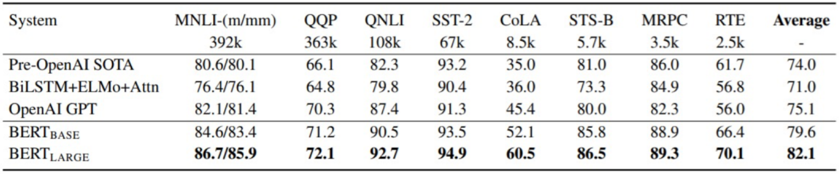


Token-level classification:
each token's hidden state → Dense layer → Tag.

BERT finetune performance

BERT was massively popular and hugely versatile; finetuning BERT led to new SOTA results across many NLP tasks.

- **QQP**: Quora Question Pairs (paraphrase detection)
- **QNLI**: QA-style natural language inference
- **SST-2**: sentiment analysis
- **CoLA**: linguistic acceptability
- **STS-B**: semantic textual similarity
- **MRPC**: Microsoft paraphrase corpus
- **RTE**: natural language inference corpus



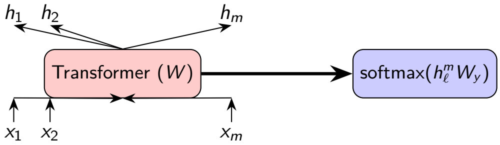
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

GPT finetune: using GPT for downstream tasks

- Pretrained GPT is a **task-agnostic decoder**: it maps a token sequence to a sequence of hidden states.
- To use GPT for a downstream task, we:
  - **Reformat the task** as a single input sequence (e.g., `[Start] Text [Delim] Hypothesis [Extract]`).
  - Feed the whole sequence into the pretrained Transformer.
  - Take the hidden state at a chosen position (often the last or an `[Extract]` token) and pass it to a small **linear head**.
- Different tasks share the same GPT backbone; they only differ in how the sequence is constructed:
  - **Classification**: `[Start] Text [Extract]` $\rightarrow$ label.
  - **Similarity**: two texts encoded and combined before a linear score.
  - **Multiple choice**: one sequence per candidate answer; score each.

Optimization formulation for LLM full finetuning

Given a sample $\xi = (x, y)$ with input tokens $x = \{x_1, \ldots, x_m\}$ and label y, generate the prediction $P(y \mid x; W) = \mathrm{softmax}(h_\ell^m W_y)$.:



- **Objective components:**

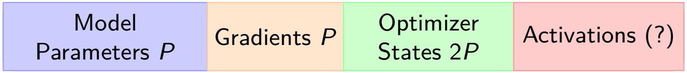
$$\ell_{\mathrm{LM}}(W) = \sum_i \log P(x_i \mid x_{<i}; W), \quad \ell_{\mathrm{Cls}}(W) = \log P(y \mid x; W).$$

- **Full fine-tuning optimization:** Define the total loss as

$$\boxed{\min_W \; \mathbb{E}_{\xi \sim \mathcal{D}}\left[F(W; \xi)\right]} \text{ with } \; F(W; \xi) := -\ell_{\mathrm{Cls}}(W) - \lambda \ell_{\mathrm{LM}}(W)$$

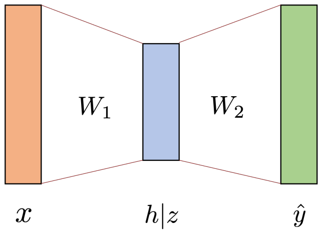
Memory decomposition of training LLMs

$$\textbf{Memory} = \underbrace{\text{Model}}_{\text{weights}} + \underbrace{\text{Gradients}}_{\text{backprop}} + \underbrace{\text{Optimizer States}}_{\text{e.g., Adam m,v}} + \underbrace{\text{Activations}}_{\text{forward/backward pass}}$$



Typical memory usage breakdown in training large models.

Memory cost: One-layer network with batch gradient



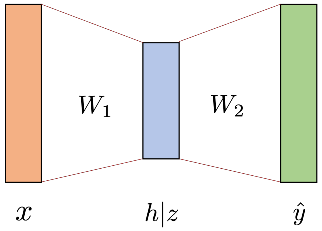
dims:	d	input dimension
	(p, d)	W_1
	p	hidden dimension
	(q, p)	W_2
	q	output dimension

Forward:

$$\begin{aligned} h_b &= W_1 x_b, \\ z_b &= \sigma(h_b), \\ y_b &= W_2 z_b, \\ f &= \frac{1}{B}\sum_{b=1}^{B} \mathcal{L}(y_b) \end{aligned}$$

Store $\{h_b, z_b, y_b\}_{b=1}^{B}$ during forward pass

Memory cost: One-layer network with batch gradient



dims:	d	input dimension
	(p, d)	W_1
	p	hidden dimension
	(q, p)	W_2
	q	output dimension

Backward:

$$\frac{\partial f}{\partial W_2} = \frac{1}{B}\sum_{b=1}^{B} \frac{\partial \mathcal{L}}{\partial y_b} z_b^\top,$$

$$\frac{\partial f}{\partial z_b} = W_2^\top \frac{\partial \mathcal{L}}{\partial y_b},$$

$$\frac{\partial f}{\partial h_b} = \frac{\partial f}{\partial z_b} \odot \sigma'(h_b),$$

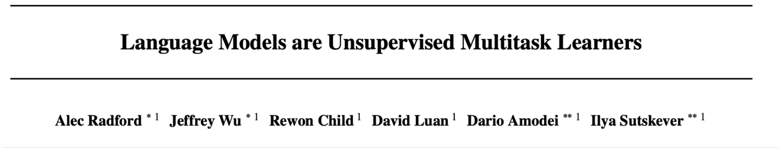
$$\frac{\partial f}{\partial W_1} = \frac{1}{B}\sum_{b=1}^{B} \frac{\partial f}{\partial h_b} x_b^\top$$

Store $\nabla_{W_1} f(W_1)$ and $\nabla_{W_2} f(W_2)$ during backward pass

Table of Contents

The introduction of GPT-2

How to strike back? Increasing the data and enlarging the models!



Language Models are Unsupervised Multitask Learners

Alec Radford [*1] **Jeffrey Wu** [*1] **Rewon Child** [1] **David Luan** [1] **Dario Amodei** [**1] **Ilya Sutskever** [**1]

GPT-2 **Radford et al., 2019** demonstrated that scaling up decoder-only language models leads to strong zero-shot multi-task performance.

The Emergence of GPT-2

GPT increases to 1.5B parameters!

Parameters	**Layers**	$\mathbf{d}_{\text{model}}$
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

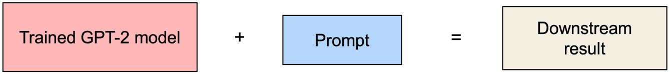
BERT model size

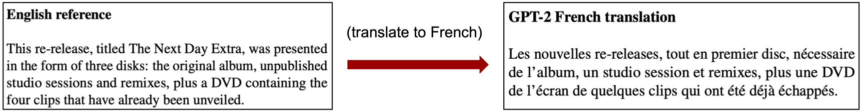
Hyperparameters for the GPT-2 model sizes.

However, GPT-2, even with much larger model sizes, still could not compete with BERT and its variants at the time.

GPT-2 is a zero-shot learner

- GPT-2 requires no downstream dataset and no fine-tuning.
- A well-chosen **prompt** is enough.





English reference

This re-release, titled The Next Day Extra, was presented in the form of three disks: the original album, unpublished studio sessions and remixes, plus a DVD containing the four clips that have already been unveiled.

(translate to French)

GPT-2 French translation

Les nouvelles re-releases, tout en premier disc, nécessaire de l'album, un studio session et remixes, plus une DVD de l'écran de quelques clips qui ont été déjà échappés.

- In contrast, BERT and variants do not exhibit comparable zero-shot generation capability.

GPT-2 is a zero-shot learner

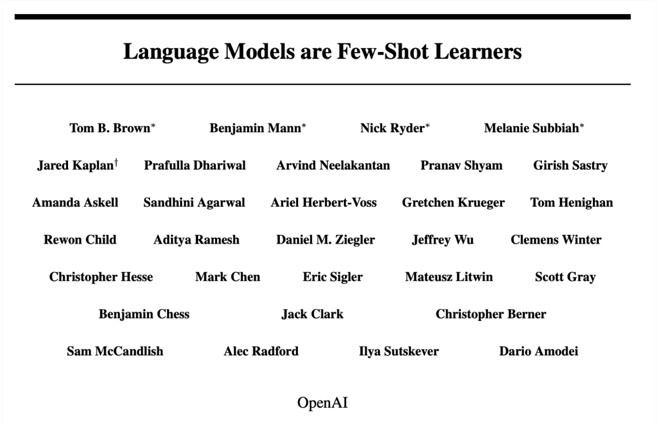
- On some tasks, GPT-2 achieves state-of-the-art (SOTA) results *without* any fine-tuning.

	LAMBADA (PPL)	**LAMBADA** (ACC)	**CBT-CN** (ACC)	**CBT-NE** (ACC)	**WikiText2** (PPL)	**PTB** (PPL)	**enwik8** (BPB)	**text8** (BPC)	**WikiText103** (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3
117M	**35.13**	45.99	87.65	83.4	29.41	65.85	1.16	1.17	37.50
345M	15.60	55.48	92.35	87.1	22.76	47.33	1.01	**1.06**	26.37
762M	10.87	60.12	93.45	88.0	19.93	40.31	**0.97**	1.02	22.05
1542M	**8.63**	63.24	**93.30**	**89.05**	**18.34**	**35.76**	**0.93**	**0.98**	**17.48**

Table: GPT-2 zero-shot performance. PPL = Perplexity (lower is better); ACC = Accuracy (higher is better); BPB = Bits-per-byte (lower is better); BPC = Bits-per-character (lower is better).

- Zero-shot results on many datasets - no downstream training or fine-tuning was performed.

The introduction of GPT-3



Language Models are Few-Shot Learners

Tom B. Brown[*] **Benjamin Mann**[*] **Nick Ryder**[*] **Melanie Subbiah**[*]

Jared Kaplan[†] **Prafulla Dhariwal** **Arvind Neelakantan** **Pranav Shyam** **Girish Sastry**

Amanda Askell **Sandhini Agarwal** **Ariel Herbert-Voss** **Gretchen Krueger** **Tom Henighan**

Rewon Child **Aditya Ramesh** **Daniel M. Ziegler** **Jeffrey Wu** **Clemens Winter**

Christopher Hesse **Mark Chen** **Eric Sigler** **Mateusz Litwin** **Scott Gray**

Benjamin Chess **Jack Clark** **Christopher Berner**

Sam McCandlish **Alec Radford** **Ilya Sutskever** **Dario Amodei**

OpenAI

GPT-3 **Brown et al., 2020** showed that massive scaling of model parameters enables powerful in-context learning.

GPT-3 becomes extremely large

GPT-2 has 1.5B parameters, while GPT-3 reaches 175B parameters.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	**Batch Size**
GPT-3 Small	125M	12	768	12	64	0.5M
GPT-3 Medium	350M	24	1024	16	64	0.5M
GPT-3 Large	760M	24	1536	16	96	0.5M
GPT-3 XL	1.3B	24	2048	24	128	1M
GPT-3 2.7B	2.7B	32	2560	32	80	1M
GPT-3 6.7B	6.7B	32	4096	32	128	2M
GPT-3 13B	13.0B	40	5140	40	128	2M
GPT-3 175B	175.0B	96	12288	96	128	3.2M

GPT-3: From zero-shot to one-shot

GPT-2 mostly supports **zero-shot**, while GPT-3 introduces strong **few-shot learning** abilities.

Zero-shot

The model predicts the answer using only **task instruction**. No gradient updates are performed.

```
Translate English to
French:
cheese =>
```

One-shot

The task description is provided along with **one example**. No gradient updates are performed.

```
Translate English to
French:
sea otter => loutre de
mer
cheese =>
```

GPT-3: Few-shot vs fine-tuning

Few-shot GPT-2 uses zero-shots but GPT-3 uses few-shots.

The model sees a few examples **in the prompt**. No gradient updates.

```
Translate English to
French:
sea otter => loutre de
mer
peppermint => menthe
poivre
plush giraffe =>
girafe peluche
cheese =>
```

Fine-tuning

The model is trained with **gradient updates** over many examples.

```
sea otter => loutre de
mer
```

gradient update

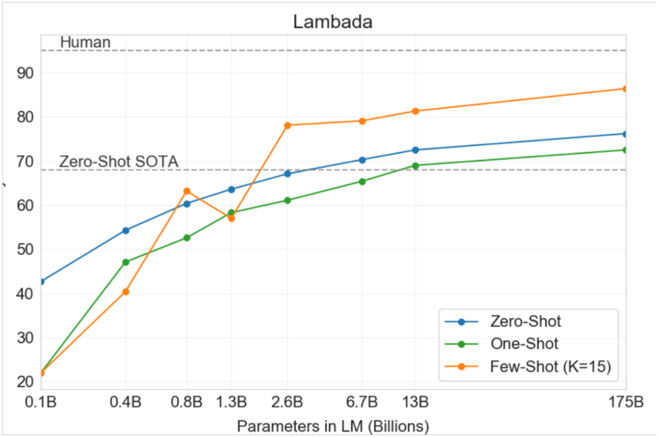
```
peppermint => menthe poivre
```

gradient update

⋮

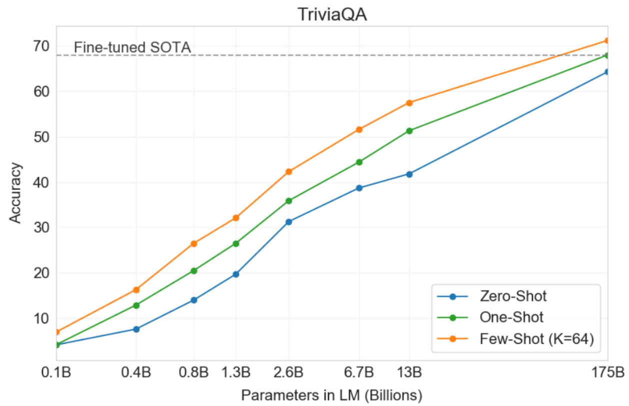
```
plush giraffe => girafe
peluche
```

GPT-3 shows strong performance



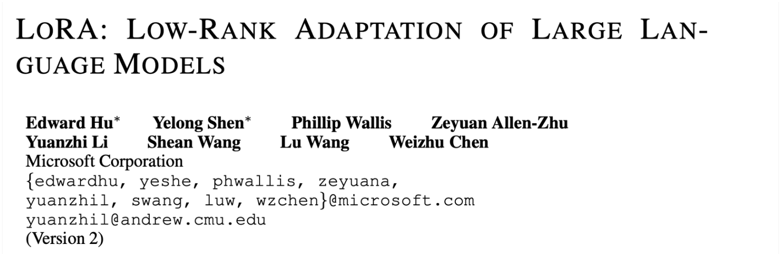
- GPT-3 performance scales smoothly with model size.
- Few-shot prompting yields major improvements over zero-shot.

GPT-3 shows strong performance



- GPT-3 performance scales smoothly with model size.
- For large models (13B+), few-shot nearly matches fine-tuned SOTA.

LoRA: Low-Rank Adaptation of LLMs


LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
Yuanzhi Li **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

LoRA **Hu et al., 2021** introduced the idea that LLMs can be adapted using low-rank update matrices, without modifying the original weights.

Fine-tuning LLM with low-rank adaptation

- **Full parameter fine-tuning:**

$$\min_{W \in \mathbb{R}^{p \times q}} \; \mathbb{E}_{\xi \sim \mathcal{D}}\big[F(W;\, \xi)\big]$$

- **Finetuning with low-rank adaptation:**

$$\min_{A \in \mathbb{R}^{p \times r},\, B \in \mathbb{R}^{r \times q}} \; \mathbb{E}_{\xi \sim \mathcal{D}}\big[F(W + AB;\, \xi)\big]$$

- New knowledge and expertise are injected through the rank r matrices A and B, while keeping the original weight W.

Fine-tuning LLM with low-rank adaptation

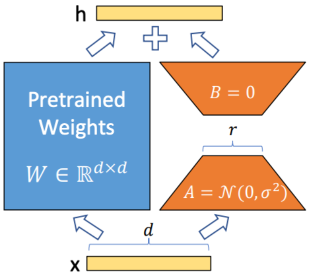
- **Low-rank finetuning objective:**

$$\min_{A\in\mathbb{R}^{p\times r},\, B\in\mathbb{R}^{r\times q}} \mathbb{E}_{\xi\sim\mathcal{D}}\big[F(W+AB;\,\xi)\big]$$

- Only A and B are trained; the pretrained weight W is fixed.
- Effective weight:

$$W' = W + \Delta W = W + AB$$

$$W'x = (W + AB)x = Wx + ABx$$



Pretrained weights $W \in \mathbb{R}^{d\times d}$ plus low-rank adapters $A \in \mathbb{R}^{p\times r}, B \in \mathbb{R}^{r\times q}$.

How much memory can LoRA save?

- Overall memory cost for training large models:

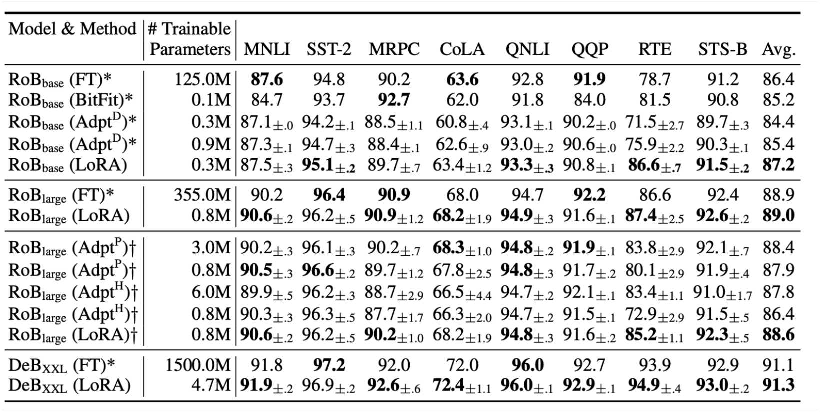
$$\text{Memory} = \text{Models} + \text{Grads} + \text{Optimizers} + \text{Activations}.$$

- **LoRA does not save activation-incurred memory.**
  - It does *not* help when activation memory dominates, e.g., very long-context transformers.

- **LoRA saves model/grad/optimizer memory:**
  - Full fine-tuning: $O(pq)$ parameters for a weight $W \in \mathbb{R}^{p\times q}$.
  - LoRA: $O((p+q)r)$ parameters for $A \in \mathbb{R}^{p\times r}$ and $B \in \mathbb{R}^{r\times q}$ with $r \ll \min\{p, q\}$.

- When rank r is small and activation memory is not the bottleneck, LoRA can give substantial overall memory savings.

Results: LoRA vs. full fine-tuning and other adapters



Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{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_{base} (Adpt^{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_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{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
$\text{RoB}_{\text{large}}$ (Adpt^{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
$\text{RoB}_{\text{large}}$ (Adpt^{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
$\text{RoB}_{\text{large}}$ (Adpt^{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
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

RoBERTa and DeBERTa models on GLUE tasks. LoRA matches or slightly outperforms full fine-tuning while training only a tiny fraction of parameters.

DoRA: Weight decomposition & update rule

The decomposition principle

- DoRA decomposes the pretrained weight $W_0 \in \mathbb{R}^{d \times k}$ into **Magnitude** (m) and **Direction** (V):

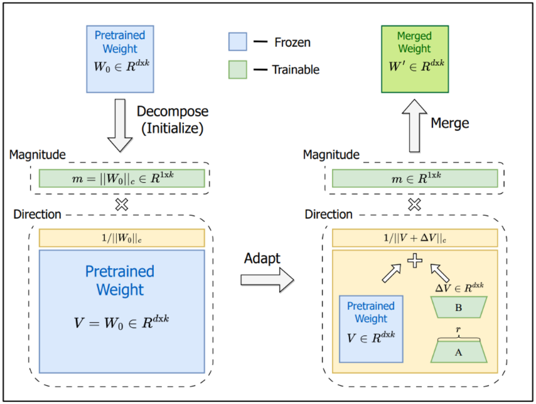
$$W = m \odot \frac{V}{\|V\|_c} \quad \text{(scaling vector } m\text{, normalized matrix } V\text{)}$$

Instead of updating W directly, DoRA trains m and V separately.

- **Magnitude:** Trained directly (Δm).
- **Direction:** Updated via LoRA (BA).

$$W' = \underbrace{(m + \Delta m)}_{\text{Updated Magnitude}} \odot \frac{V_0 + \overbrace{BA}^{\text{Directional Update}}}{\|V_0 + BA\|_c}$$

DoRA trains the magnitude and direction separately



$$W' = m\frac{V + \Delta V}{\|V + \Delta V\|_c} = m\frac{W_0 + BA}{\|W_0 + BA\|_c}$$

DoRA: Analysis of weight updates

To understand the FT behavior, we analyze how the weights evolve in terms of magnitude and direction compared to initialization (W_0).

- **Magnitude variation (ΔM):** The average absolute change in the magnitude vector.

$$\Delta M_{\mathrm{FT}}^{t} = \frac{1}{k} \sum_{n=1}^{k} |m_{\mathrm{FT}}^{n,t} - m_0^n|$$

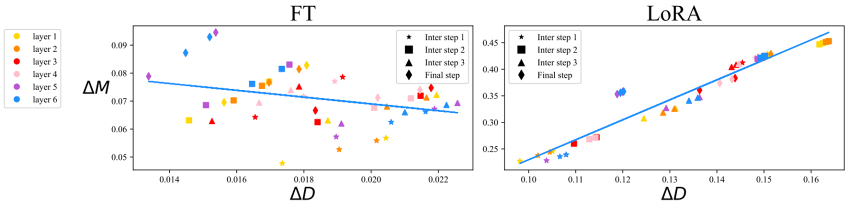
- **Directional variation (ΔD):** The average cosine distance between the updated direction V_{FT} and the original W_0.

$$\Delta D_{\mathrm{FT}}^{t} = \frac{1}{k} \sum_{n=1}^{k} \big(1 - \cos(V_{\mathrm{FT}}^{n,t}, W_0^n)\big)$$

Magnitude and direction variations in LoRA

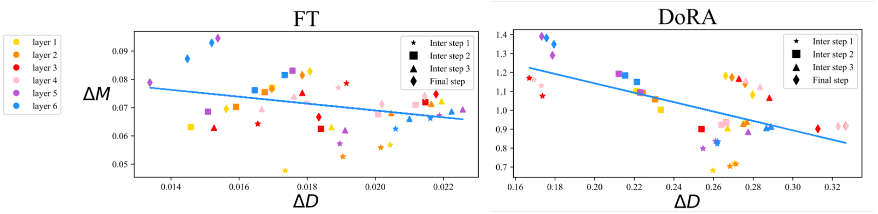
- Similarly, the magnitude and direction variations between W_0 and the LoRA-updated weight W_{LoRA} are:

$$\Delta M^t_{\mathrm{LoRA}} = \frac{1}{k}\sum_{n=1}^{k} |m^{n,t}_{\mathrm{LoRA}} - m^n_0|, \quad \Delta D^t_{\mathrm{LoRA}} = \frac{1}{k}\sum_{n=1}^{k}(1 - \cos(V^{n,t}_{\mathrm{LoRA}}, W^n_0))$$



- This reflects a fundamental difference between LoRA and FT.

Magnitude and direction variations in DoRA



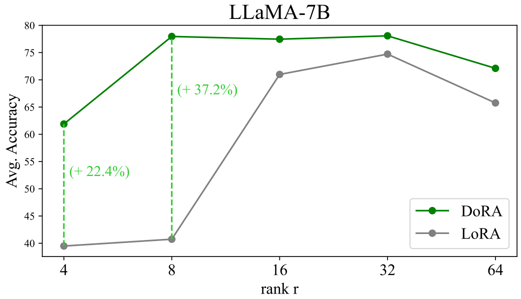
Slight-negative proportionality (FT)

Negative proportionality (DoRA)

- DoRA is more consistent with FT than LoRA.

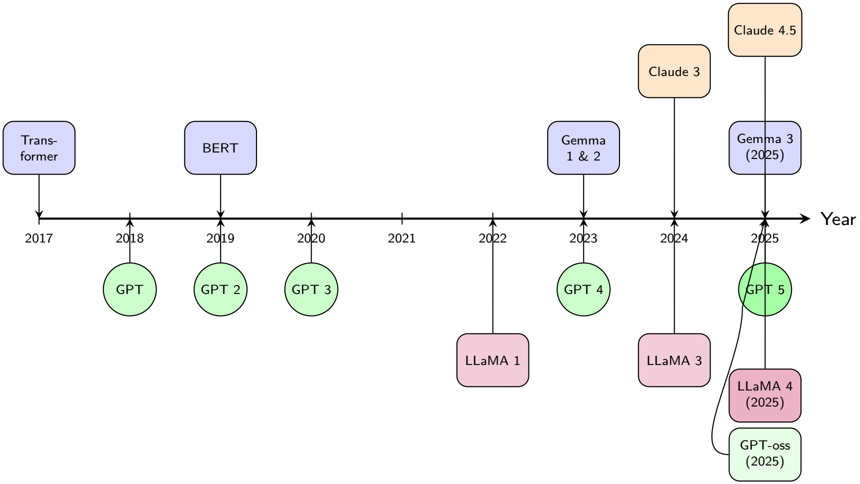
DoRA shows strong performance

Method	# Params(%)	Avg.
FT	100	66.5
LoRA	4.61	66.9
DoRA	4.63	**67.6**



Average accuracy of LoRA and DoRA for varying ranks for LLaMA-7B on the commonsense reasoning tasks.

LLM timeline: Major LLM milestones (2017–2025)



Recap

- What we have talked about today?
  - ⇒ How to formulate the pre-training of LLMs as optimization?
  - ⇒ How to perform full fine-tuning of LLMs? What is the cost?
  - ⇒ How to perform parameter-efficient fine-tuning of LLMs?

Welcome anonymous survey!