Machine Learning (CE 40717)

Fall 2024

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

December 24, 2024

Language models

- Definition of a Language Model (LM)
  - A machine learning model designed to predict and generate plausible language by analyzing text patterns.
  - Operates by learning from large amounts of text data to understand linguistic structures, context, and vocabulary.
    - Common example: Autocomplete in text typing

Language models

- Purpose of Language Models
  - Estimate the probability of a word (token) or sequence of words within a longer text sequence.
  - Aim to understand and predict context in sentences.

Language models

Example of Language Model Prediction

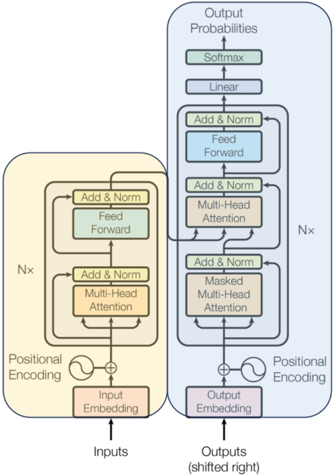
- When I hear rain on my roof, I _______ in my kitchen.
  Potential predictions with probabilities:
  "cook soup" - 9.4%
  "warm up a kettle" - 5.2%
  "cower" - 3.6%
  "nap" - 2.5%
  "relax" - 2.2%

Language models

The LLM Era – Paradigm Shift in Machine Learning

BERT
Oct 2018

Representation



GPT
Jun 2018

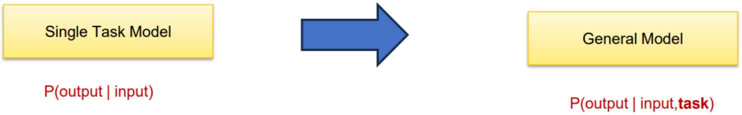
Generation

Generalizing to Unseen Tasks

- LMs can be used for different tasks by pre-training a “base” model and then fine-tuning for the task(s) of interest
- Practical Issues:
  - Too many copies of the model
  - Need for large-scale labeled data for fine-tuning
- Multi-task Training?
  - Data remains a challenge
  - Humans don’t need such large volumes of data to learn – can we do better?
- Train a model that can perform NLP tasks in a zero-shot manner

Task Specifications

- Primary shift comes from modeling assumptions from single-task to general model



- Task descriptions may be provided as text – for example, translate this French text to English

Perplexity

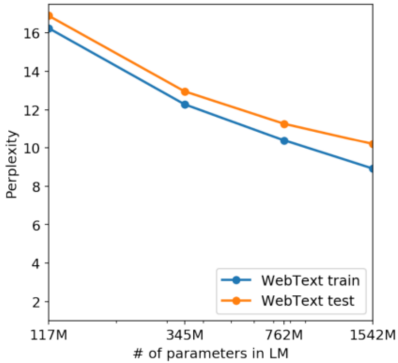
- Perplexity as a measure of uncertainty: Perplexity is the exponentiation of the average negative log-likelihood of a probability distribution, representing the uncertainty of a model in predicting the next item in a sequence.

$$\text{Perplexity}(P) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i)\right)$$

- N is the number of words (or tokens) in the dataset,
- $P(w_i)$ is the probability of the i-th word in the sequence.

Scaling

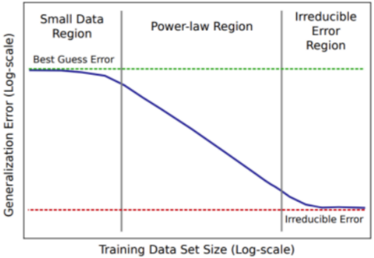
- Scaling improves the perplexity of the LM and improves performance

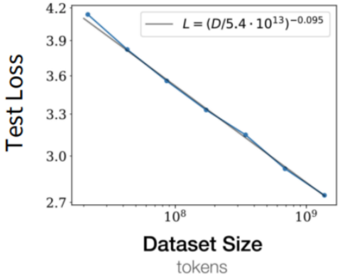


Power-Law Scaling

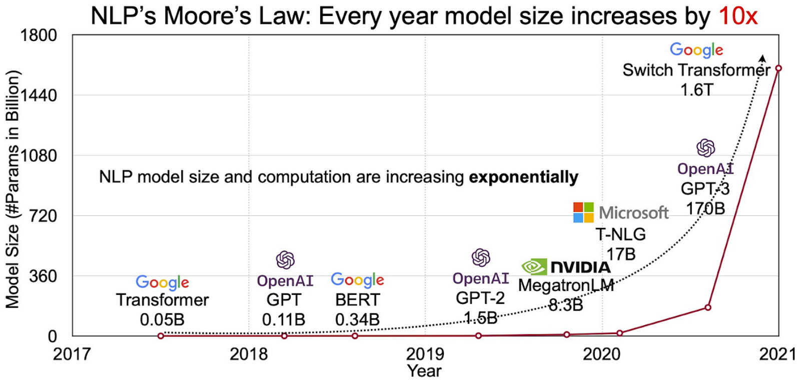
Why is this interesting? Look at data scaling

- Loss and dataset size is linear on a log-log plot
- This is "power-law scaling"





NLP Moore's Law



Large Language models

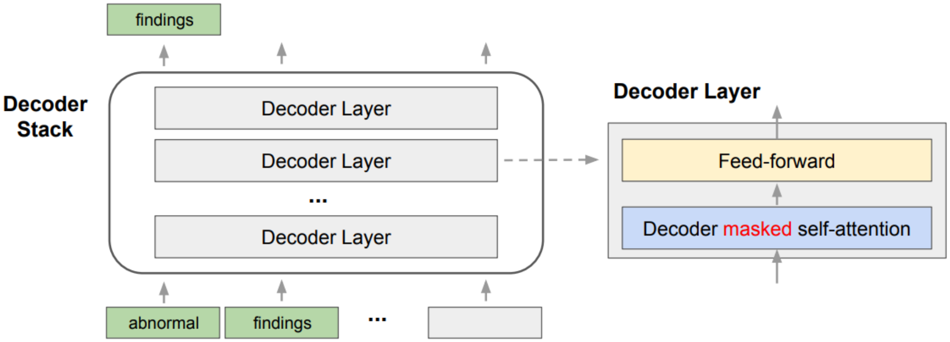
- What are Large Language Models (LLMs)?
    - Large Language Models (LLMs) are advanced statistical models based on neural networks designed to process and generate natural language. Formally, an LLM can be defined as:

$$P(w_1, w_2, \ldots, w_N) = \prod_{i=1}^{N} P(w_i \mid w_1, w_2, \ldots, w_{i-1})$$

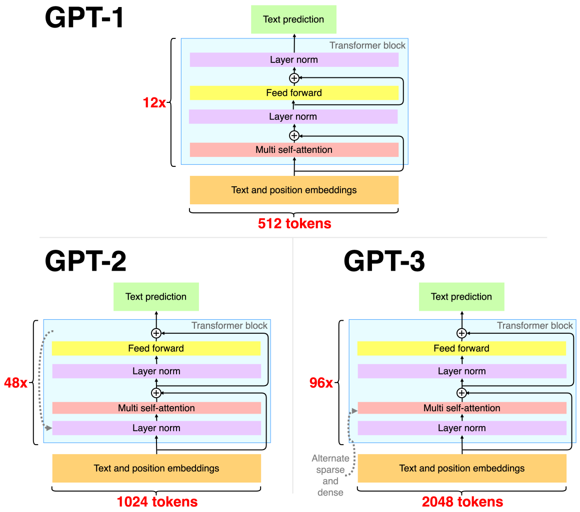
      where:
        - $w_1, w_2, \ldots, w_N$ are words (or tokens) in a sequence,
        - $P(w_i \mid w_1, w_2, \ldots, w_{i-1})$ represents the conditional probability of the i-th word given the previous words.
- Key Characteristics of LLMs
    - **Architecture:** LLMs are typically built using Transformer-based architectures, where attention mechanisms are used to capture dependencies across the input sequence.
    - **Parameterization:** "Large" models have millions to billions of parameters (θ), optimized using a loss function.
    - **Training:** Trained on massive text corpora with unsupervised tasks like next-word prediction or masked token prediction.

Generative Pre-Training (GPT)

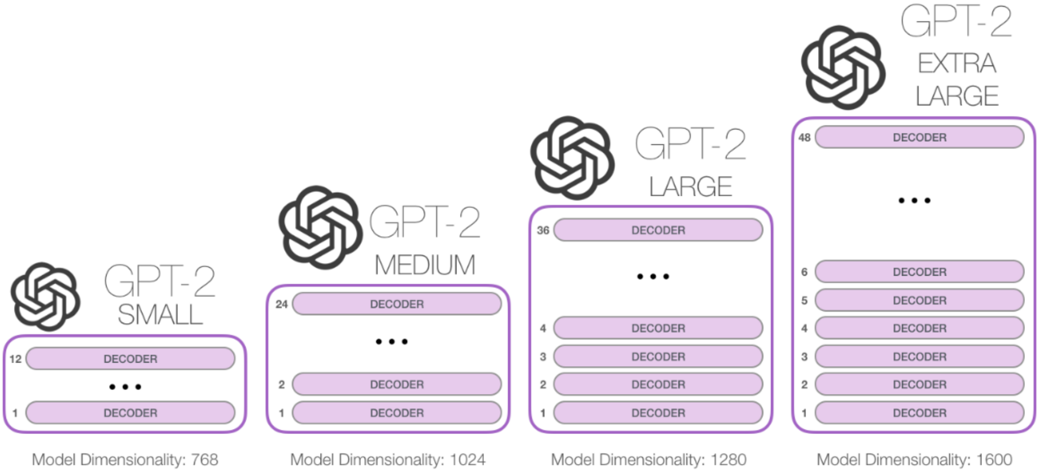
GPT: Based on Transformer decoder layers



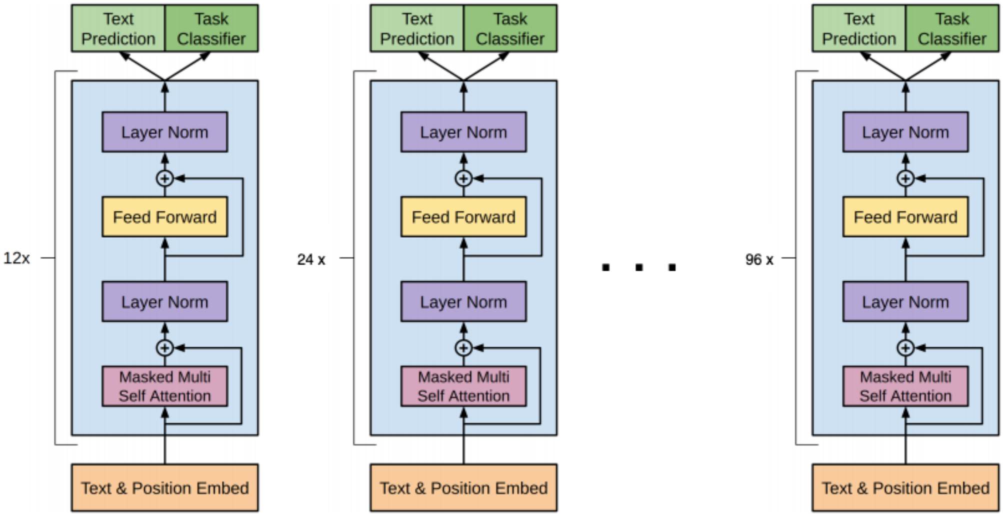
GPTs


GPT-1
Text prediction
Transformer block
Layer norm
Feed forward
Layer norm
Multi self-attention
12x
Text and position embeddings
512 tokens
GPT-2
Text prediction
Transformer block
Feed forward
Layer norm
Multi self-attention
Layer norm
48x
Text and position embeddings
1024 tokens
GPT-3
Text prediction
Transformer block
Feed forward
Layer norm
Multi self-attention
Layer norm
96x
Alternate sparse and dense
Text and position embeddings
2048 tokens

GPT-2


GPT-2
SMALL
12
DECODER
1
DECODER
Model Dimensionality: 768
GPT-2
MEDIUM
24
DECODER
2
DECODER
1
DECODER
Model Dimensionality: 1024
GPT-2
LARGE
36
DECODER
4
DECODER
3
DECODER
2
DECODER
1
DECODER
Model Dimensionality: 1280
GPT-2
EXTRA
LARGE
48
DECODER
6
DECODER
5
DECODER
4
DECODER
3
DECODER
2
DECODER
1
DECODER
Model Dimensionality: 1600

GPT-3

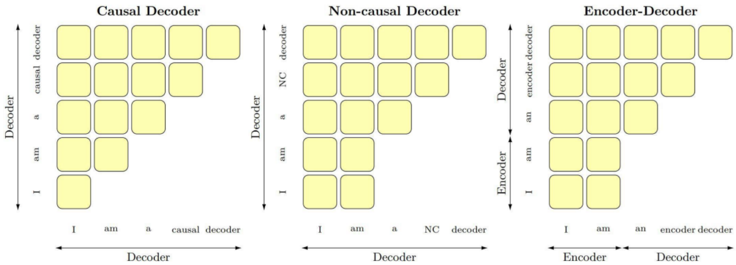

Text Prediction
Task Classifier
Layer Norm
Feed Forward
Layer Norm
Masked Multi Self Attention
Text & Position Embed
12x
24 x
96 x

GPT-3

Emergent Abilities

- Emergent abilities:
  - not present in smaller models but is present in larger models
  - Do LLMs like GPT3 have these ?
- Findings:
  - GPT-3 trained on text can do arithmetic problems like addition and subtraction
  - Different abilities "emerge" at different scales
  - Model scale is not the only contributor to emergence – for 14 BIG-Bench tasks, LaMDA 137B and GPT-3 175B models perform at near-random, but PaLM 62B achieves above-random performanceb
  - Problems LLMs can't solve today may be emergent for future LLMs

Attention patterns



- Causal decoder – each token attends to the previous tokens only.
- In both non-causal decoder and encoder-decoder, attention is allowed to be bidirectional on any conditioning information.
- For the encoder-decoder, that conditioning is fed into the encoder part of the model.

Llama 2 Architecture

- Decoder-only model
- Changes in transformer module:
  - Norm after sublayer -> Norm before sublayer
  - LayerNorm -> RMSNorm for stability
  - Activation: ReLU -> SwiGLU(x)
  - Position Embedding: Absolute/Relative -> RoPE (Rotary PE)
  - Long contexts : Multi-head attention -> Grouped-query attention

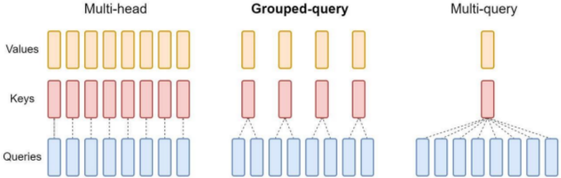
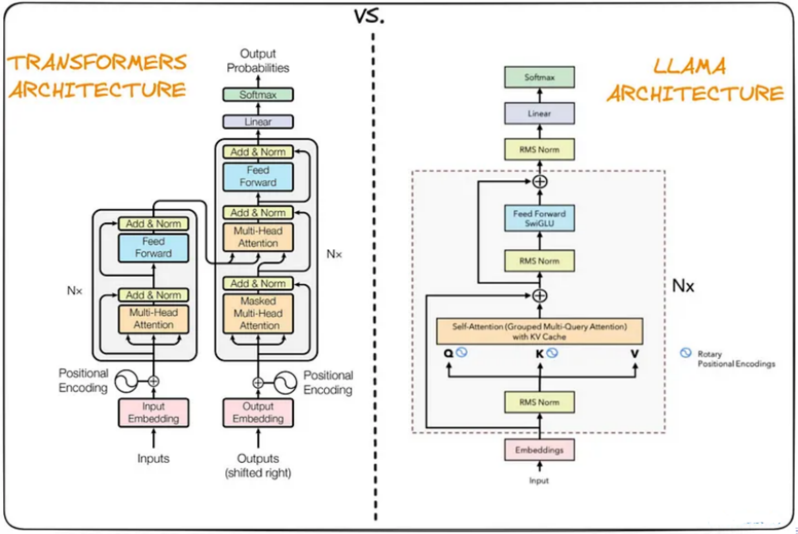


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

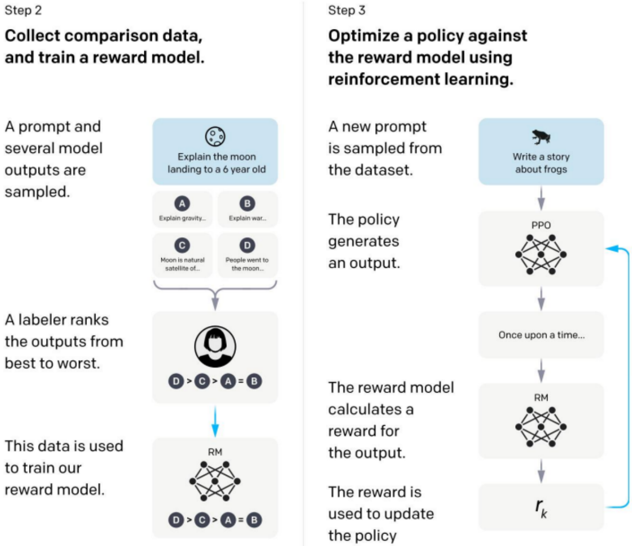
Llama 2 Architecture



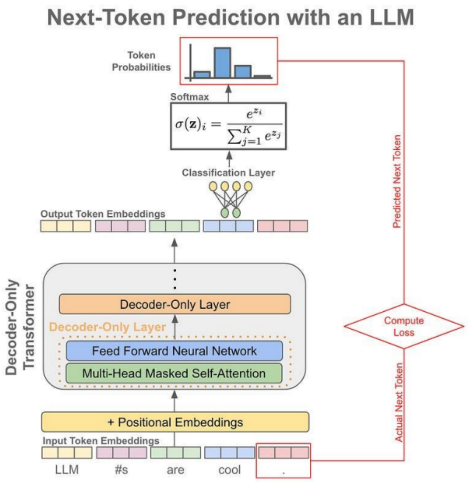
Training of Decoder-only LLMs – Llama 2

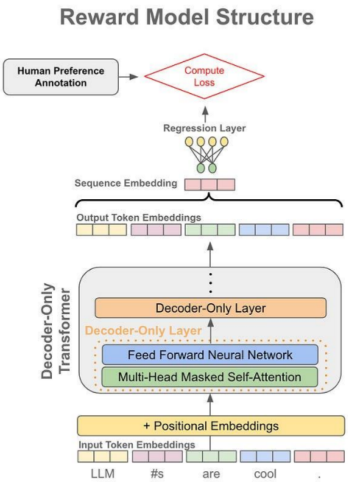
- Auto-regressive Pre-training - Train to predict the next token on very large scale corpora (3 trillion tokens)
- Instruction Fine-tuning/ Supervised Fine-tuning (SFT) - Fine-tune the pretrained model with pairs of (instruction+input,output) with large dataset and then with small high-quality dataset
- Safety / RLHF - Design a reward model based on human feedback and use policy gradient methods with the trained reward model to update LLM parameters so that outputs align with human values

RLHF


Step 2
Collect comparison data, and train a reward model.
A prompt and several model outputs are sampled.
Explain the moon landing to a 6 year old
A
Explain gravity...
B
Explain war...
C
Moon is natural satellite of...
D
People went to the moon...
A labeler ranks the outputs from best to worst.
D > C > A = B
This data is used to train our reward model.
RM
D > C > A = B
Step 3
Optimize a policy against the reward model using reinforcement learning.
A new prompt is sampled from the dataset.
Write a story about frogs
The policy generates an output.
PPO
Once upon a time...
The reward model calculates a reward for the output.
RM
The reward is used to update the policy
r_k

Model Fine-tuning for RLHF





Pre-training and adaptation

- Pre-training
  - Models are initially trained on massive datasets to learn general language patterns, grammar, and common knowledge.
  - The pre-trained model is further trained on a smaller, task-specific dataset to specialize in a particular application (e.g., sentiment analysis, translation).
- Adaptation
  - The pre-trained model is further trained on a smaller, task-specific dataset to specialize in a particular application (e.g., sentiment analysis, translation).
  - Fine-tuning tailors the model's responses and predictions to perform effectively on the target task.

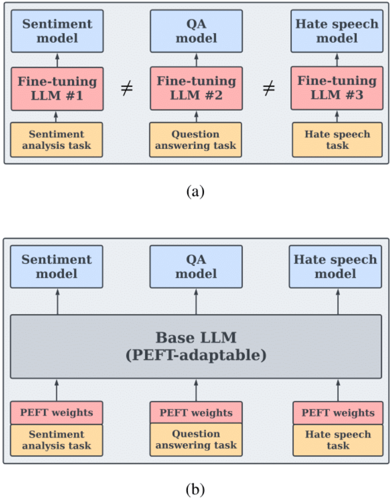
Pre-training and adaptation

Figure 1: Overview of Model Training: Pre-training on large, unlabeled data builds foundational knowledge, while fine-tuning on smaller, labeled datasets adapts the model for specific tasks.

Parameter-Efficient Fine-Tuning (PEFT)

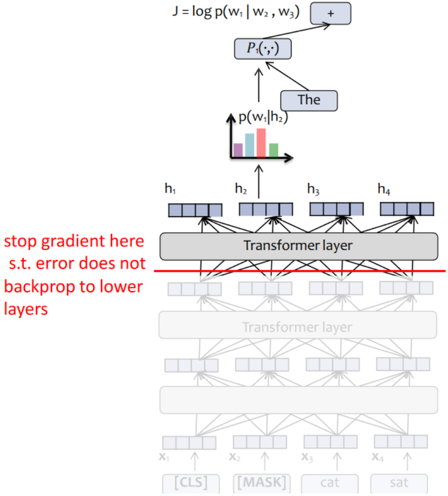
- **Targeted Adaptation:** PEFT involves fine-tuning a small subset of model parameters, allowing the pretrained model to adapt to specific tasks without retraining the entire model.
- **Efficiency Gains:** By preserving the majority of the pretrained model's structure, PEFT significantly reduces training time, memory usage, and computational cost.
- **Scalability:** PEFT enables large language models to be applied effectively to a wide range of tasks, making it feasible to use large models in resource-constrained environments.

- Fine-tuning an LLM for a specific downstream task
  - (a) illustrates vanilla fine-tuning, which requires updating the entire model, resulting in a new model for each task.
  - (b) PEFT instead learns a small subset of model parameters for each task with a fixed base LLM. The same base model can be re-used during inference for different tasks.



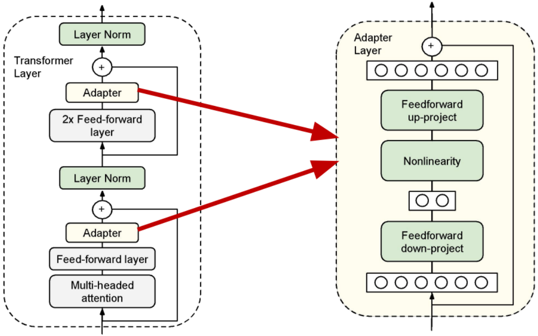
Fine-Tuning the Top Layers Only

- **PEFT Baseline Efficiency:** Freeze parameters except top K layers, reducing computation and memory usage.
- **Flexible Application:** Can be applied to most deep neural networks for efficient fine-tuning.



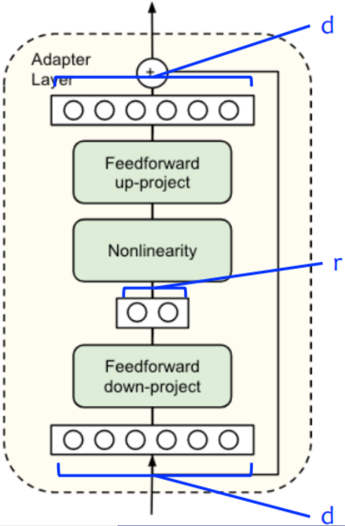
Adapters

- **Adapters:** New modules inserted between layers of a pre-trained model, with the original model weights fixed.
- **Training Efficiency:** Only adapter modules are tuned, initialized to ensure their output resembles that of the original model.



Adapters

- **Adapter Design:** Feedforward layer with one hidden layer and a residual connection.
- **Bottleneck Architecture:** Input and output dimensions are d, with a reduced dimension r in the middle.
- **Initialization:** Near-identity initialization with a skip connection and near-zero weights.



BitFit

- **Bias Fine-Tuning:** Only fine-tune the bias terms and final classification layer, reducing the number of parameters updated (<1% of the model).
- **Implementation:** Use a custom optimizer to fine-tune only the bias parameters by selecting them from the model's named parameters, but this approach may fail with large models.
- Fail when model size is large

- Recall the equations for multi-head attention

$$Q^{m,\ell}(x) = W_q^{m,\ell} x + b_q^{m,\ell}$$

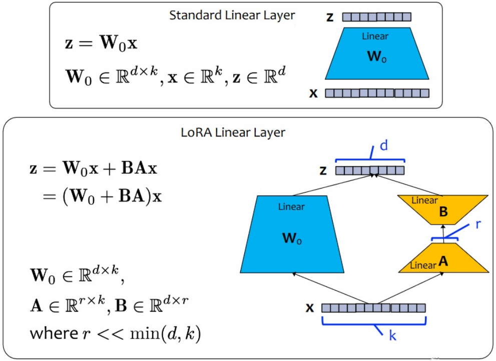
$$K^{m,\ell}(x) = W_k^{m,\ell} x + b_k^{m,\ell}$$

$$V^{m,\ell}(x) = W_v^{m,\ell} x + b_v^{m,\ell}$$

  - ℓ is the layer index
  - m is the attention head index
  - Only the bias terms are updated.

LoRA

- **Key Idea:** Keep the original pretrained parameters $\boldsymbol{W_0}$ fixed and learn an additive modification $\Delta \boldsymbol{W}$ via low-rank decomposition $\Delta W = BA$, where $\boldsymbol{BA}$ has rank r, much smaller than k and d.
- **LoRA Linear Layer:** The modification $\Delta W = BA$ is added to the original linear transformation, where $\boldsymbol{A}$ and $\boldsymbol{B}$ have much smaller dimensions than the original weight matrix $\boldsymbol{W_0}$.



LoRA

Initialization

- We initialize the trainable parameters:

$$A_{ij} \sim \mathcal{N}(0, \sigma^2) \quad \forall i, j$$

- $B = 0$
- Thus, at the start of fine-tuning, the parameters have their pretrained values:

$$\Delta W = BA = 0$$

$$W_0 + BA = W_0$$

LoRA

Hot Swapping Parameters

- $\boldsymbol{W_0}$ and $\boldsymbol{BA}$ have the same dimension, so we can "swap" the LoRA parameters in and out of a Standard Linear Layer.
- To get a Standard Linear Layer with parameters $\boldsymbol{W}$ that includes our LoRA fine-tuning:

$$W \leftarrow W_0 + BA$$

- To remove the LoRA fine-tuning from that Standard Linear Layer:

$$W \leftarrow W - BA = W_0$$

- If we do LoRA training on two tasks such that the parameters $\boldsymbol{B'A'}$ are for task 1 and $\boldsymbol{B''A''}$ are for task 2, then we can swap back and forth between them.

LoRA

- **LoRA Linear Layers:** LoRA linear layers can replace every linear layer in the Transformer model.
- **Original Paper Focus:** The original LoRA paper specifically applies LoRA only to the attention weights (Q, K, V).
- **Mathematical Formulation:**

$$Q = LoRALinear(X; W_q, A_q, B_q)$$

$$K = LoRALinear(X; W_k, A_k, B_k)$$

$$V = LoRALinear(X; W_v, A_v, B_v)$$

QLoRA

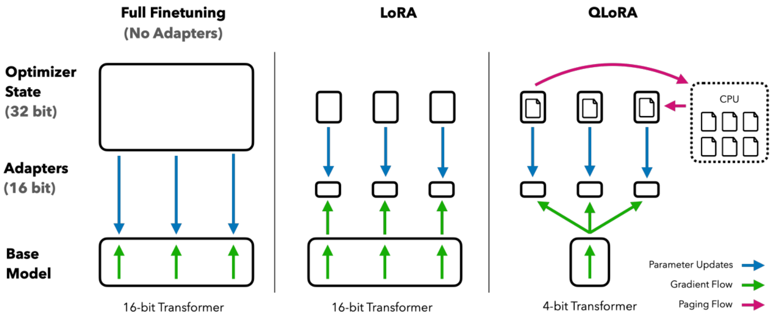
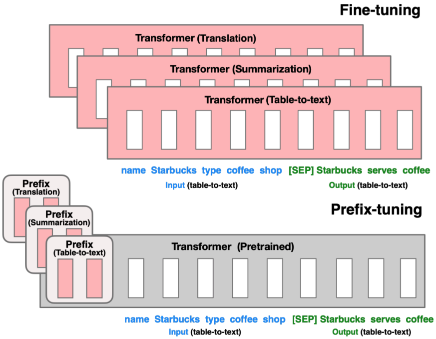


Figure 2: Different finetuning methods and their memory requirements. QLORA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

Prefix Tuning

- Freeze all pretrained parameters and prepend trainable prefix tokens to the input and hidden activations.
- The prefix is processed by the model like real words, allowing each batch element to run a different tuned model during inference.

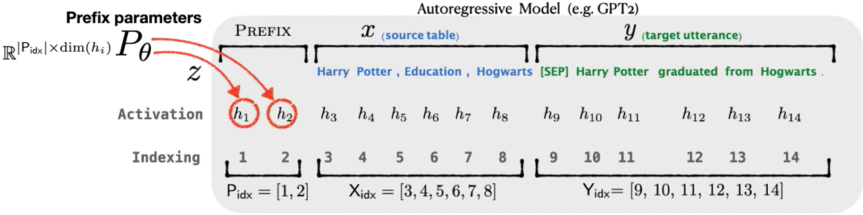


Prefix Tuning

$$h_i = \begin{cases} P_\theta[i,:], & \text{if } i \in \mathsf{P}_{\mathsf{idx}}, \\ \mathrm{LM}_\phi(z_i, h_{<i}), & \text{otherwise.} \end{cases}$$

$$\max_\theta \log p_{\phi,\theta}(y \mid x) = \sum_{i \in \mathsf{Y}_{\mathsf{idx}}} \log p_{\phi,\theta}(z_i \mid h_{<i})$$

freeze LM parameters ϕ
update prefix parameters θ

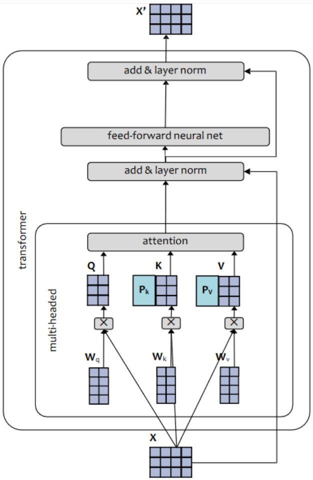


Prefix Tuning

Prefix Tuning with Multi-Head Attention

- The model uses prefix tokens (P_k and P_v) along with the attention matrices (Q, K, V) to guide the attention mechanism in the multi-headed transformer.

- The prefix tokens are processed and combined with the query, key, and value matrices (W_q, W_k, W_v) to form the final attention mechanism in the transformer architecture.



1. Large Language Models
2. Adaptation
3. Parameter-Efficient Fine-Tuning (PEFT)
4. **References**

Contribution

- **These slides were prepared with contributions from:**
  - Amirhossein Akbari

Any Questions?