What is finetuning?

: Taking general purpose models (GPT-3) and specializing them into the specific task

General Purpose		Specialized
GPT-3	→	ChatGPT
GPT-4	→	GitHub Copilot

What does finetuning do for the models?

- Lets you put more data into the pretrained model than what fits into the prompt
- Gets the model to learn the data, rather than just get access to it.
- Steers the model to more consistent outputs
- Reduces hallucinations
- Customizes the model to a specific use case

Prompt Engineering vs Finetuning

	Prompt Engineering	Finetuning
Pros	① No data to get started ② Smaller upfront cost ③ No technical knowledge needed ④ Connect data through RAG (Retrieval Augmented Generation)	① Nearly unlimited data fits ② Learn new information ③ Correct incorrect information ④ Use RAG too
Cons	① Much less data fits ② Forgets data ③ Hallucination ④ Gets incorrect data	① More high-quality data ② Needs some technical knowledge ③ Computing cost

Benefits of finetuning your own LLM

① Performance : 1) Stop hallucination
2) Increase consistency
3) Reduce unwanted info.
4) A smaller (fine-tuned) model can outperform a larger base model.

② Security : 1) Prevents leakage
2) No breaches

③ Cost

④ Reliability

Supervised Fine-tuning

1. Choose fine-tuning task
2. Prepare training dataset
3. Chose a base model
4. Fine tune model via supervised learning
5. Evaluate model performance

3 options for Parameter Training

1) Retrain all parameters

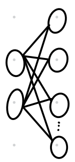
downside: billions of internal model parameters the computational cost for training explodes

2) Transfer learning : Instead of retraining all the parameters, we freeze most of the parameters and only finetune the head (last few layers of the model)

3) Parameter Efficient Fine-tuning (PEFT) : Freeze all of the weights. It augments the model with additional parameters which are trainable.

One of the most popular ways to do PEFT: Low-Rank Adaptation (LoRA)

Pictorial view



$x \to h(x)$

$h(x) = W_0(x)$

$\boxed{W_0}\boxed{x} = h(x)$

$W_0 \in \mathbb{R}^{d \times k}$

$x \in \mathbb{R}^{k \times 1}$

$h(x) = \mathbb{R}^{d \times 1}$

If $d=1000$, $k=1000$, there are 1,000,000 trainable parameters

$\Rightarrow$

LoRA

$h(x) = W_0x + \Delta Wx$, W_0: frozen, $\Delta W = BA$

$= W_0x + BAx$

$\left(\underset{\text{Frozen}}{\boxed{W_0}} + \underbrace{\boxed{B}\boxed{A}}_{\text{Trainable}}\right)\boxed{x} = h(x)$

$W_0, \Delta W = \mathbb{R}^{d \times k}$

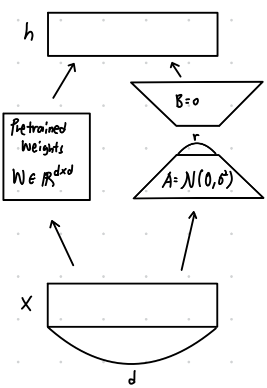
$B \in \mathbb{R}^{d \times r}$ r: intrinsic rank of the model. $r \ll k, d$

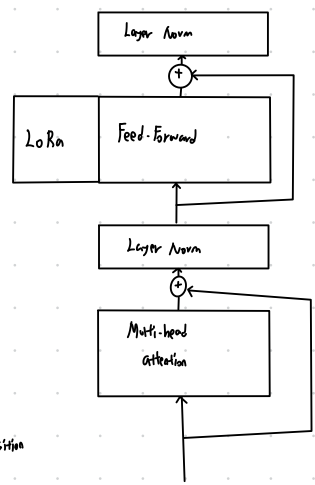
$A \in \mathbb{R}^{r \times k}$

$h(x) = R^{d \times 1}$

let $r=2$, then $(d \times r) + (r \times k)$

$= 4{,}000$ trainable parameters





The hyperparameter we need to choose is the rank r, Since we do not know what the intrinsic rank of the weight matrix is, and if we implicitly remove hopefully linearly dependent columns by the BA decomposition

If rank is too low: It will implicitly delete linearly independent columns too much

too high: We keep too many parameters that are linearly dependent, and waste computation