SNGULAR

LLM Distillation. Fine Tuning

Creating custom LLM model.

Contents:

- What is fine tuning(training-testing)
- Types of fine tuning
  - **Full Fine-Tuning**
  - **LoRA**
  - **QLoRA**
- Dataset preparation
  - **Vectorization**
  - **K-Means(Pairwise distance)**
- Hyperparameters
- Grid Search
- Use case

What is fine tuning about?

Base pretrained optimization function

Let:

- $\mathcal{D}_{\text{pretrain}}$: large dataset used for general pretraining.
- θ: the model parameters (weights) of the LLM.
- The goal is to minimize the loss:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{pretrain}}}[\mathcal{L}(f(x;\theta), y)]$$

where:

- x: input (e.g. tokenized text),
- y: target (e.g. next token in causal language modeling),
- $\mathcal{L}$: loss function (e.g. cross-entropy),
- $f(x;\theta)$: output logits of the model.

Gradient update rule

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta \mathcal{L}(f(x;\theta^{(t)}), y)$$

where:

- η: learning rate,
- ∇_θ: gradient with respect to the model parameters.

Fine tuning optimization function

$$\theta_{\text{fine}} = \arg\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{fine}}}[\mathcal{L}(\text{DistilledLLM}(x;\theta), y)] \quad \text{with} \quad \theta \leftarrow \theta_S^*$$

Learning Process

Train

- This is the **data the model learns from**.
- It **includes both** the input features (**Symptoms**) and the correct output (**Diagnostic**).
- The model uses this data to find **patterns or relationships**.

Test

- This is **new data the model hasn't seen before**.
- It's used to **check how well the model performs** on unseen examples.
- It helps evaluate whether the model **generalizes well**.

Example

Training a model to recognize cats vs. dogs in pictures:

- **Training Set:** 80 photos (40 cats, 40 dogs) → Model learns features (like ears, tails).
- **Test Set:** 20 new photos → We check if the model can still correctly tell cats from dogs.

Full Fine Tuning

Cross-Entropy Loss

$$\mathcal{L}_{\text{CE}} = -\sum_{t=1}^{T} \log P_\theta(y_t \mid x_{<t})$$

Gradient Descent or Adam

Updates all parameters θ

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}(f(x;\theta), y)$$

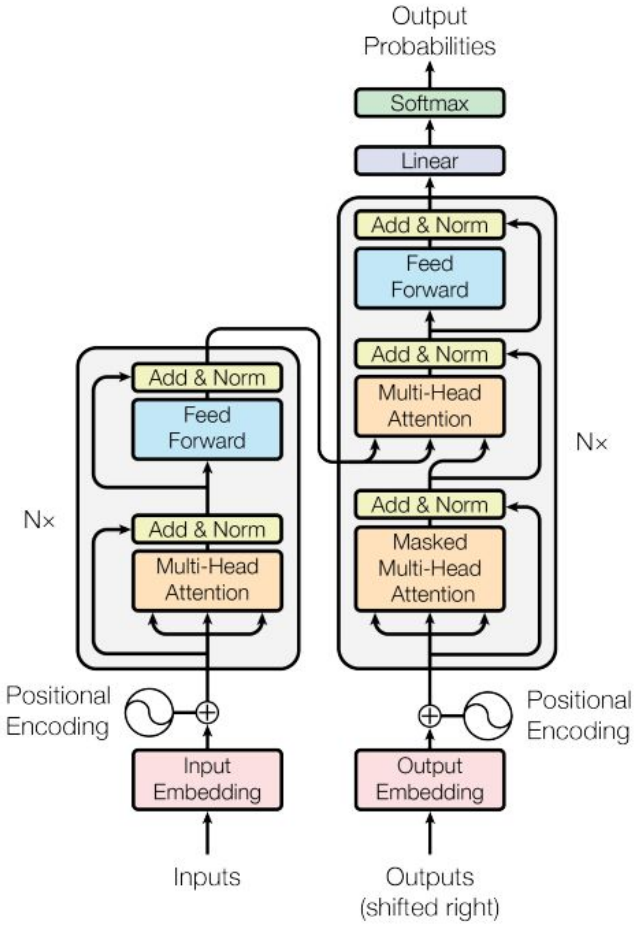
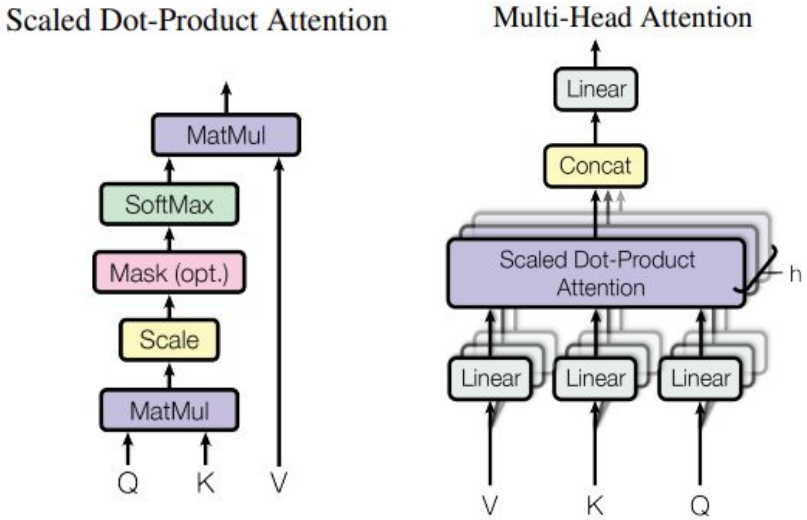


Figure 1: The Transformer - model architecture.

LoRA Fine Tuning

$$W' = W + \Delta W \quad \text{where } \Delta W = BA,\ A \in \mathbb{R}^{r \times k},\ B \in \mathbb{R}^{d \times r}$$

$$h = Wx + BAx$$



SNGULAR

QLoRA Fine Tuning

Instead of using:

$$h = Wx$$

QLoRA uses:

$$h = \underbrace{\tilde{W}x}_{\text{quantized frozen weights}} + \underbrace{\alpha \cdot BAx}_{\text{trainable LoRA adapters}}$$

$$\min_{\phi} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i; \tilde{\theta}, \phi),\ y_i)$$

- $\tilde{\theta}$: quantized model weights (frozen)
- ϕ: LoRA adapters (trainable)
- Only ϕ gets updated during training

- $\tilde{W}$ is stored in **4-bit quantized form** (typically using `NF4` quantization).
- The adapter BA is **full-precision (float32 or bfloat16)**.

Dataset Preparation

Vectorization

1- Input Sentence

"Hello"

2- Character Vocabulary

Assume a vocabulary:

$$\mathcal{V} = \{H, e, l, o\} \quad \Rightarrow \quad \text{Indices: } H = 0,\ e = 1,\ l = 2,\ o = 3$$

3- One-Hot Encoding

Each character is converted into a **one-hot vector** of size 4 (the vocabulary size):

Character	One-hot vector
H	[1, 0, 0, 0]
e	[0, 1, 0, 0]
l	[0, 0, 1, 0]
o	[0, 0, 0, 1]

So the word "Hello" becomes:

$$X = \begin{bmatrix} [1,0,0,0] \\ [0,1,0,0] \\ [0,0,1,0] \\ [0,0,1,0] \\ [0,0,0,1] \end{bmatrix} \quad \in \mathbb{R}^{5\times 4}$$

SNGULAR

Dataset Preparation

Vectorization

4- Embedding Matrix

Let's define a **learned embedding matrix** $E \in \mathbb{R}^{4\times 3}$ to map each character to a 3D vector:

$$E = \begin{bmatrix} \text{H} & \rightarrow & [0.1,\ 0.3,\ 0.5] \\ \text{e} & \rightarrow & [0.2,\ 0.1,\ 0.4] \\ \text{l} & \rightarrow & [0.4,\ 0.4,\ 0.4] \\ \text{o} & \rightarrow & [0.7,\ 0.9,\ 0.2] \end{bmatrix}$$

$$X \cdot E = Z \in \mathbb{R}^{5\times 3}$$

$$Z = \begin{bmatrix} [0.1,\ 0.3,\ 0.5] \\ [0.2,\ 0.1,\ 0.4] \\ [0.4,\ 0.4,\ 0.4] \\ [0.4,\ 0.4,\ 0.4] \\ [0.7,\ 0.9,\ 0.2] \end{bmatrix} \in \mathbb{R}^{5\times 3}$$

Dataset Preparation

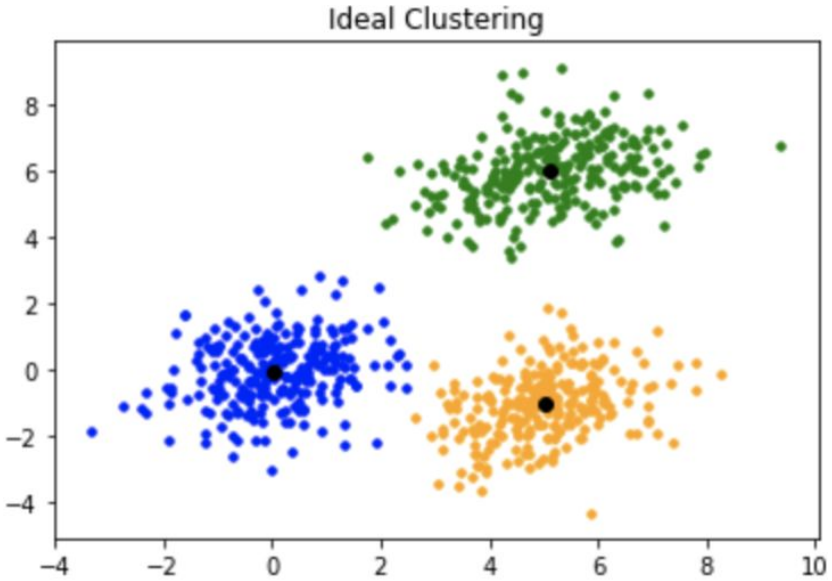
K-Means: Statistically Representative Sample

$$\min_{\{C_1,\ldots,C_K\}} \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

Where:

- C_k: the set of points assigned to cluster k
- μ_k: the **centroid** (mean) of cluster C_k:

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$$



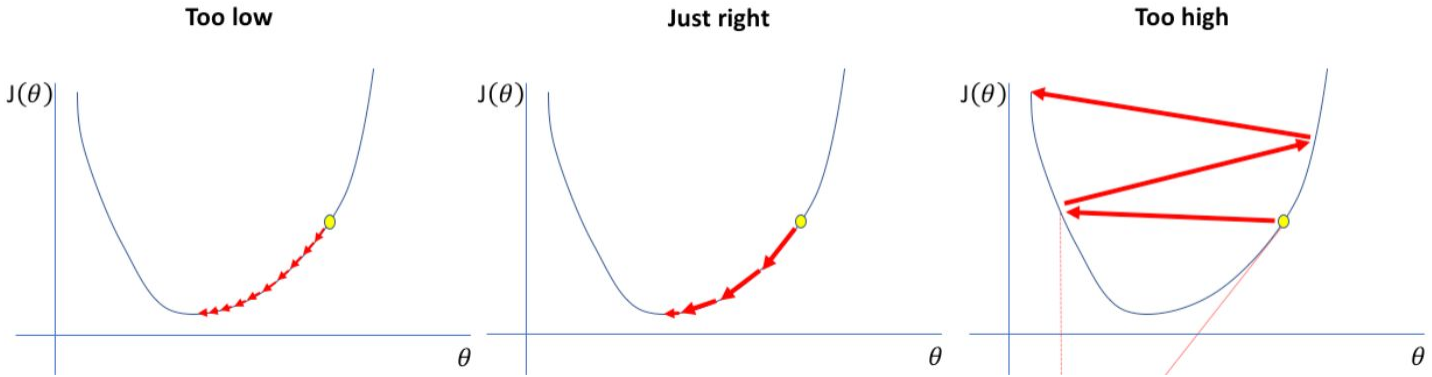
Hyperparameters
Learning Rate

For a model with parameters θ, and a loss function $\mathcal{L}(\theta)$, the gradient update is:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta \mathcal{L}(\theta^{(t)})$$

Where:

- $\theta^{(t)}$: parameters at iteration t
- $\nabla_\theta \mathcal{L}$: gradient of the loss w.r.t. parameters
- η: **learning rate**



Hyperparameters

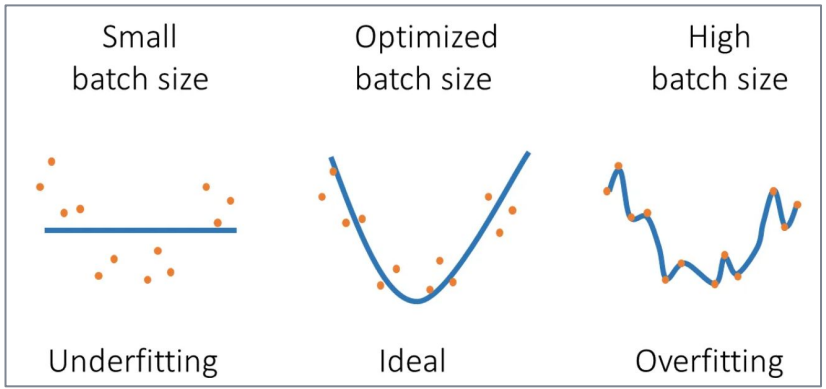
Batch Size

Stochastic Gradient Descent is approximated to the true gradient:

$$\nabla_\theta \mathcal{L}_{\text{true}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \mathcal{L}(\theta; x_i, y_i)$$

With the **mini-batch gradient**:

$$\nabla_\theta \mathcal{L}_{\text{batch}}(\theta) = \frac{1}{B} \sum_{(x_j, y_j) \in \mathcal{B}} \nabla_\theta \mathcal{L}(\theta; x_j, y_j)$$



Hyperparameters

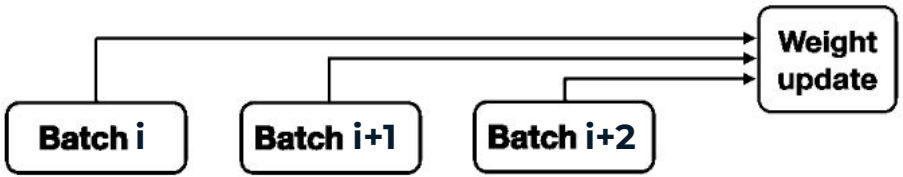
Gradient Accumulation

- b: micro-batch size (e.g., 4)
- g: **gradient accumulation steps**

$$B = b \cdot g$$

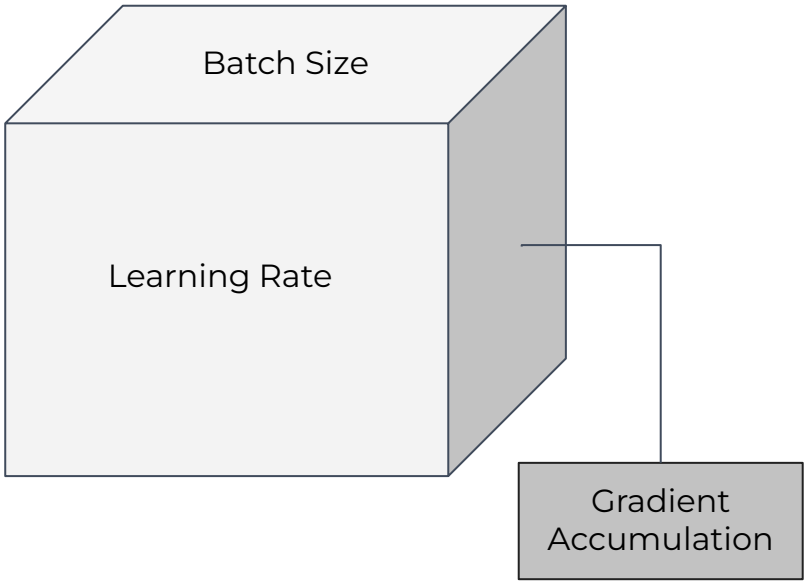
$$\nabla_\theta \mathcal{L}_B = \frac{1}{B} \sum_{i=1}^{B} \nabla_\theta \mathcal{L}(\theta; x_i)$$

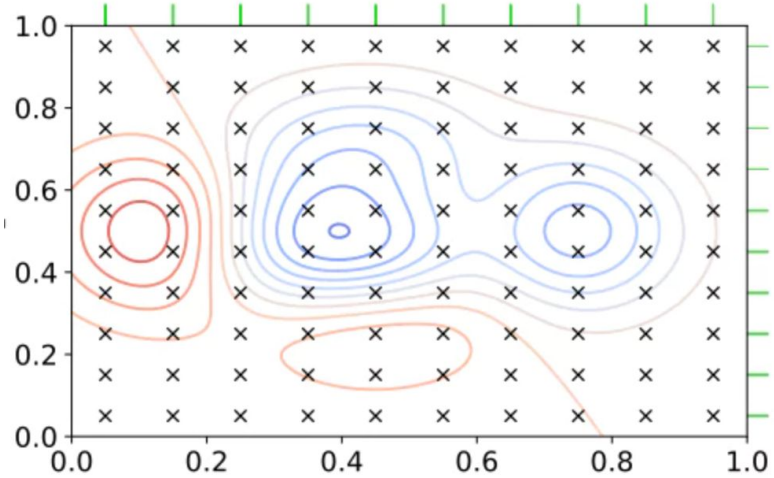
$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_B$$



Hyperparameters

Grid Search





Use Case

01 Find Best Hyperparameters

02 Fine Tune Qwen Distilled Model

03 Push to Hugging Face

Bibliography

- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.
- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2022). Lora: Low-rank adaptation of large language models. *ICLR*, *1*(2), 3.
- Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, *36*, 10088-10115.
- Hamerly, G., & Elkan, C. (2003). Learning the k in k-means. *Advances in neural information processing systems*, *16*.

Next Talk

- Google Cloud Vertex AI's and its role in deployment.
- Use Case:
  - **Service integration with Python SDK Google Cloud Vertex AI.**
  - **Decoder Strategies (Generation Parameters) for LLMs.**

Questions

Thanks for your attention!!!