📌 Hugging Face

Hugging Face is a platform that provides models, datasets, and libraries to make machine learning (ML) more accessible. What GitHub is to general programming, Hugging Face is to AI & ML.

📌 Fine-Tuning

Fine-tuning is the process of taking a **pre-trained** model and further **training it on a specific dataset** to **adapt it to a new task**.

✒️ Types of Fine-Tuning

- **Full Parameter Fine-Tuning** → All parameters of the model are updated.
- **Parameter-Efficient Fine-Tuning (PEFT)** → Instead of updating the entire model, only a small subset of parameters is trained.

📌 Full parameter fine-tuning

Full parameter fine-tuning refers to the process of updating all the parameters of a pre-trained model during training on a new task. Full fine-tuning modifies every layer of the model.

📌 PEFT

PEFT (Parameter-Efficient Fine-Tuning) is a technique that allows fine-tuning of large pre-trained models efficiently with fewer trainable parameters. Instead of updating all parameters of a model, PEFT focuses on modifying a small subset, making training faster, cheaper, and more memory-efficient.

✒️ Why PEFT

- **Efficient Fine-Tuning** - Requires significantly fewer resources compared to full fine-tuning.
- **Faster Training** - Reduces computation time.
- **Lower Memory Requirements** - Ideal for resource-constrained environments.
- **Better Transfer Learning** - Enables adaptation of large models to multiple tasks efficiently.

📌 PEFT Techniques

📌 LoRA (Low-Rank Adaptation)

Injects small trainable matrices into the model, reducing the number of trainable parameters.

✒️ Matrix decomposition - Matrix decomposition refers to the process of factorizing weight updates in a neural network into two low-rank matrices, thereby reducing the number of trainable parameters.

✒️ Key Idea - Instead of fine-tuning the entire weight matrix W of a neural network, LoRA approximates the weight updates using a low-rank decomposition: $\Delta W = AB$

- A and B are low-rank matrices.
- The original model weights W remain frozen.
- The update is applied as $W' = W + \Delta W$

✒️ Mathematical Breakdown

Full Fine-Tuning (Expensive) - A weight matrix W in a transformer layer has dimensions $d \times k$. Traditional fine-tuning updates all $d \times k$ parameters, which is computationally expensive.

LoRA Decomposition (Efficient) - Instead of updating W, LoRA parameterizes the update ΔW as the product of two low-rank matrices A and B: $\Delta W = AB$

✒️ Example

- Full Fine-Tuning Parameters: $4096 \times 4096 = 16M$
- LoRA Parameters:
  - Matrix A - (4096×8), Matrix B - (8×4096)
  - $(4096 \times 8) + (8 \times 4096) = 65K$
- Reduction: 99.6% fewer trainable parameters!

✒️ Understanding the Expression - The following equation represents how LoRA (Low-Rank Adaptation) modifies a neural network's linear transformation while keeping the original pre-trained weights W frozen.

The transformation applied to the input is $\rightarrow (W + AB)x + b$

- $Wx \rightarrow$ Standard linear transformation using the frozen pre-trained weights.
- $ABx \rightarrow$ LoRA's low-rank adaptation (a learnable modification to W).
- $(W + AB)x \rightarrow$ The effective weight applied to x, which consists of:
    - W (original frozen weights).
    - AB (learned low-rank modification from LoRA).
- $+b \rightarrow$ Bias is added as usual.

📌 Rank in the Matrix

The rank of a matrix A is the number of linearly independent rows or columns in the matrix.

✒️ Rank in LoRA (Low-Rank Adaptation) - In LoRA, the weight update ΔW is represented as a low-rank decomposition: ΔW=AB. **The higher the rank the more precision it will have.**

- A is $d{\times}r$ B is $r{\times}d$, r (rank) is much smaller than d, making LoRA computationally efficient.

Example: Let's take a 6x6 Matrix. So total there will be 36 params.

- **Rank 1** → (6x1) + (1x6) = 12 params.
- **Rank 2** → (6x2) + (2x6) = 24 params.

Effect:

- **Increasing Rank (Higher r)**
    - Retains more expressive power.
    - Improves performance on complex tasks.
    - More trainable parameters → Higher memory usage.
- **Decreasing Rank (Lower r)**
    - Faster training with a lower memory footprint.
    - Less computation is needed.
    - Over-compression may degrade performance on complex tasks.

📌 Alpha (α) / Scaling Factor

The alpha parameter (α) in LoRA acts as a scaling factor that controls the magnitude of the LoRA updates. It is used to re-scale the low-rank adaptation to prevent drastic changes to the original model.

✒️ How Alpha is Applied - Instead of just using AB, LoRA scales it with α / r

- α (alpha) is a scaling hyperparameter.
- r is the rank of the low-rank matrices A and B.

Thus, the modified weight update becomes:

$$W' = W + \frac{\alpha}{r} AB$$

This ensures that the impact of the LoRA update is controlled, preventing excessive deviation from the pre-trained weights. **(α) and Rank are Hyperparameters**.

📌 QLoRA (Quantized LoRA)

QLoRA (Quantized Low-Rank Adaptation) is an advanced parameter-efficient fine-tuning (PEFT) technique that combines LoRA with quantization to further reduce memory usage while fine-tuning large language models (LLMs).

✒️ How QLoRA Works

- 4-bit Quantization of Model Weights - Instead of storing model weights in 32-bit floating-point (FP32) or 16-bit (FP16/BF16) format, QLoRA quantizes them into 4-bit precision.
- Dequantization During Computation - The model decompresses (dequantizes) weights back to a higher precision during forward/backward passes but keeps them stored in 4-bit to save memory.

✒️ Mathematical Representation in QLoRA - QLoRA applies the same LoRA transformation but operates on quantized weights $Wq \rightarrow (Wq + AB)x + b$

- Wq → Quantized pre-trained weight matrix.
- A, B → LoRA's low-rank trainable matrices (FP16/BF16).
- x → Input tensor.
- b → Bias term (trainable).

Since Wq is stored in 4-bit, memory savings are massive, but LoRA fine-tunes the model effectively.

✒️ Use Case

- **Without QLoRA** → A 65B model in FP16 requires > 100GB VRAM. Fine-tuning is impossible on a single consumer GPU.
- **With QLoRA** → The same model fits in 24GB VRAM. Fine-tuning is efficient and feasible.

📌 BERT → Bidirectional Encoder Representations from Transformers

✒️ Encoder → Contextualized Embeddings. This means that each word representation is dynamically adjusted based on the surrounding context.

✒️ Decoder → It uses contextualized embeddings to generate the next values.

- GPT → Built on the Decoder part of the transformer architecture.
- BERT → Built on the Encoder part of the transformer architecture.

Since BERT is built on top of the Encoder part, there is no need for masking.

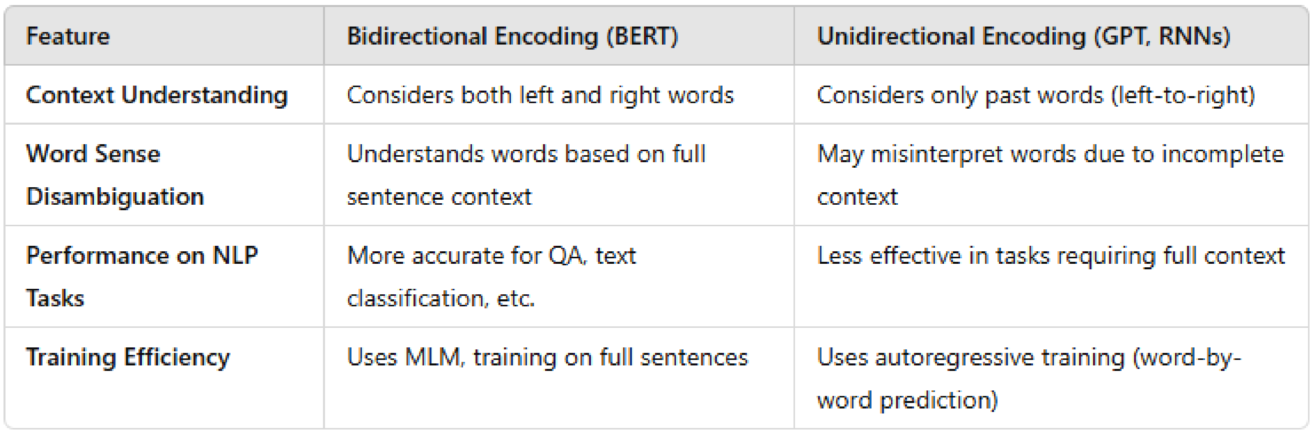
📌 Why Bidirectional Encoder in BERT

BERT uses a bidirectional encoder because it enables the model to understand the context from both left and right directions simultaneously. This is different from traditional language models, which process text in a left-to-right or right-to-left fashion.

✒️ The Need for Bidirectional Understanding

- Traditional NLP models, such as GPT (Generative Pre-trained Transformer), are trained using a unidirectional approach.

- These models can't fully capture the entire sentence structure at once, which is crucial for many NLP tasks.



Feature	Bidirectional Encoding (BERT)	Unidirectional Encoding (GPT, RNNs)
Context Understanding	Considers both left and right words	Considers only past words (left-to-right)
Word Sense Disambiguation	Understands words based on full sentence context	May misinterpret words due to incomplete context
Performance on NLP Tasks	More accurate for QA, text classification, etc.	Less effective in tasks requiring full context
Training Efficiency	Uses MLM, training on full sentences	Uses autoregressive training (word-by-word prediction)

📌 NLU in BERT - BERT is a model that excels in Natural Language Understanding (NLU) tasks.

✒️ Why is BERT Strong in NLU - BERT is designed with deep bidirectional context using self-attention, making it highly effective for understanding the meaning of text.

✒️ NLU Applications of BERT - Question Answering (QA), Named Entity Recognition (NER), Sentiment Analysis, Text Classification, Semantic Search.

📌 Pre-training in BERT

BERT (Bidirectional Encoder Representations from Transformers) is pre-trained using **self-supervised learning** objectives.

📌 Self-Supervised Learning

Self-Supervised Learning (SSL) is a type of machine learning where a model learns from unlabeled data by generating its own labels from the input data itself. It serves as a bridge between unsupervised learning (no labels) and supervised learning (labelled data).

✒️ How Does Self-Supervised Learning Work - SSL uses pretext tasks, which are artificially created learning objectives to help the model understand patterns in the data.

Process:

- Pretraining on a pretext task (self-generated labels).
- Fine-tuning on a downstream task (actual real-world application).

Example:

- Pretext Task: Masked Language Modeling (MLM) → Predict missing words.

Downstream Task: Sentiment Analysis, Question Answering, Named Entity Recognition (NER), etc.

📌 Types of self-supervised learning in BERT

BERT is pre-trained using two **self-supervised learning** objectives.

✒️ MLM (Masked Language Model)

Explanation: BERT is trained to predict the original words based on the surrounding context.

- Let's take as example that 15% of words in the input sequence are randomly selected for replacement.
- Among them:
  - 80% are replaced with [MASK] tokens.
  - 10% are replaced with a random word.
  - 10% remain unchanged.

Example:

- Input - "The cat sat on the [MASK] and slept."
- BERT Prediction - "The cat sat on the mat and slept."

✒️ NSP (Next Sentence Prediction)

Explanation: The Next Sentence Prediction (NSP) task in BERT is a binary classification problem where the model determines whether one sentence logically follows another.

- Given two sentences A and B, BERT is trained to predict whether B follows A or if it is a random sentence. The dataset consists of:
  - 50% "IsNext" pairs (sentence B is the actual next sentence of A).
  - 50% "NotNext" pairs (sentence B is randomly chosen).

Example:

- **IsNext** - Label: IsNext
  - Sentence A: "I went to the store."
  - Sentence B: "I bought some fresh vegetables."

- **NotNext** - Label: NotNext
  - Sentence A: "I went to the store."
  - Sentence B: "The sun is shining brightly today."

📌 Fun Fact

Google Search uses BERT to improve search results by understanding natural language queries more accurately. Since 2019, Google has integrated BERT into its ranking system to enhance Natural Language Understanding (NLU).