Fine-Tuning

Monday, 26 June 2023 10:57 PM

PEFT:
https://ljvmiranda921.github.io/notebook/2023/05/01/peft/

Fine Tuning not always:
https://www.tidepool.so/2023/08/17/why-you-probably-dont-need-to-fine-tune-an-llm/

Embedding finetuning:
https://github.com/run-llama/finetune-embedding?trk=feed-detail_main-feed-card_feed-a

Hands-On:
https://colab.research.google.com/drive/1BiQiw31DT7-cDp1-0ySXvvhzqomTdI-o?usp=shari
https://www.philschmid.de/fine-tune-flan-t5-peft
Comparing LLMs with LangChain

https://magazine.sebastianraschka.com/p/finetuning-large-language-models

https://docs.google.com/document/d/1h-GTjNDDKPKU_Rsd0t1lXCAnHltaXTAzQ8K2HRhQf9

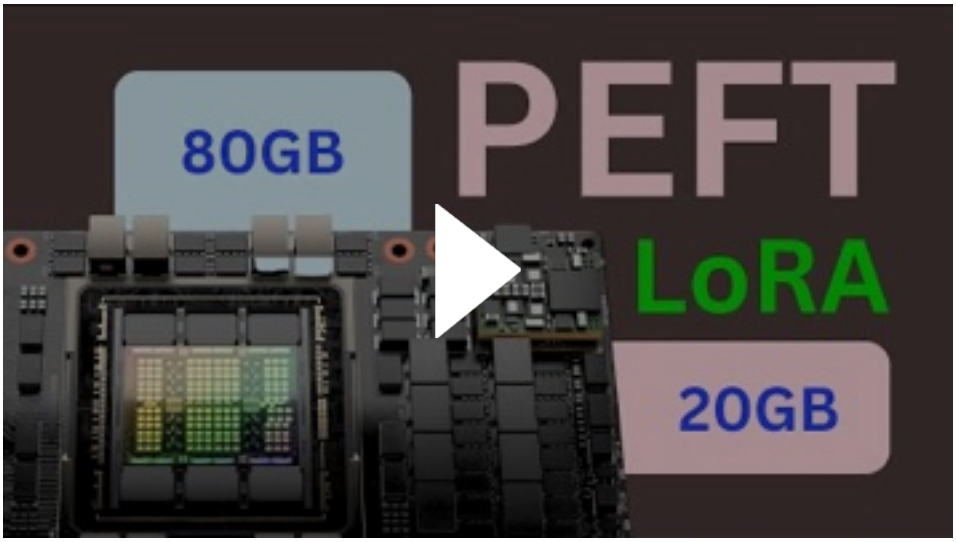
How to finetune your own Alpaca 7B

rticle-content

ng#scrollTo=C2EgqEPDQ8v6

U/edit?pli=1

LoRA / qLoRA:
[PEFT LoRA Explained in Detail - Fine-Tune your LLM on your local GPU]



PEFT:
Parameter-Efficient Fine-Tuning of Billion-Scale Models on Low-Resource Hardware

LoRA:
Low-**R**ank **A**daptation
(a random projection to a smaller subspace)

Adapter Transformers:
adapter-transformers extend 😊transformers

HuggingFace **Library**
freezing most parameters of a pre-trained LLM

freezes pre-trained model weights and injects trainable **rank decomposition matrices** into each layer of Transformer

AdapterHub: A Framework for Adapting Transformers

💊 LoRA is an algorithm that helps finetune large language models quickly. So how does the algorithm work? And how does it make the training efficient?

Here's my understanding of it.

First of all, LoRA means low-rank adaptation of large language models.

Language models like GPT-3 use a Transformer architecture which includes layers with attention and feed-forward networks. LoRA focuses on the latter: the feed-forward networks.

Let's consider just one layer of a Transformer model. The feed-forward network (FFN) can be represented as:

FFN(x) = W2 * ReLU(W1*x + b1) + b2

Here, x is the input, W1 and W2 are weight matrices, b1 and b2 are biases, and ReLU is the activation function.

The core idea of LoRA is to modify this FFN to have a new feed-forward network (FFN') that looks like this:

FFN_modified(x) = (W2 + U2V2) * ReLU((W1 + U1V1)*x + b1) + b2

U1, U2, V1, and V2 are matrices that will be learned during adaptation.

These matrices have lower ranks than the original weights matrices W1 and W2.

This low-rank structure means that the number of parameters we need to learn during adaptation is relatively small, keeping the adaptation process efficient.

For example, if U has a shape (d,r) and V has a shape (r, d), where d is the original dimension and r is the rank of the adaptation, then the number of parameters in the low-rank matrix is 2dr.

This number of parameters can be much smaller than d^2, the number of parameters in the original matrix W if it was to be fine-tuned.

So this is where the efficiency comes from!

During the adaptation process, we keep the original weights (W1, W2) and biases (b1, b2) fixed, and only learn the new parameters (U1, U2, V1, V2) using gradient descent on the specific task we're interested in.

Quantization:
https://huggingface.co/blog/hf-bitsandbytes-integration

Quantization:
https://huggingface.co/blog/hf-bitsandbytes-integration

https://lightning.ai/pages/community/tutorial/pytorch-memory-vit-llm/

https://lightning.ai/pages/community/tutorial/accelerating-large-language-models-with-mixed-precision-techniques/

Matrix Factorization:
https://medium.com/data-science-in-your-pocket/matrix-factorization-algorithms-explained-with-an-example-99426a42ce71

https://huggingface.co/blog/merve/quantization
https://www.ml6.eu/blogpost/low-rank-adaptation-a-technical-deep-dive

LoRA Maths:
PEFT LoRA Explained in Detail - Fine-Tune your LLM on your local GPU
https://github.com/Lightning-AI/lit-llama/blob/main/howto/download_weights.md
https://huggingface.co/blog/merve/quantization

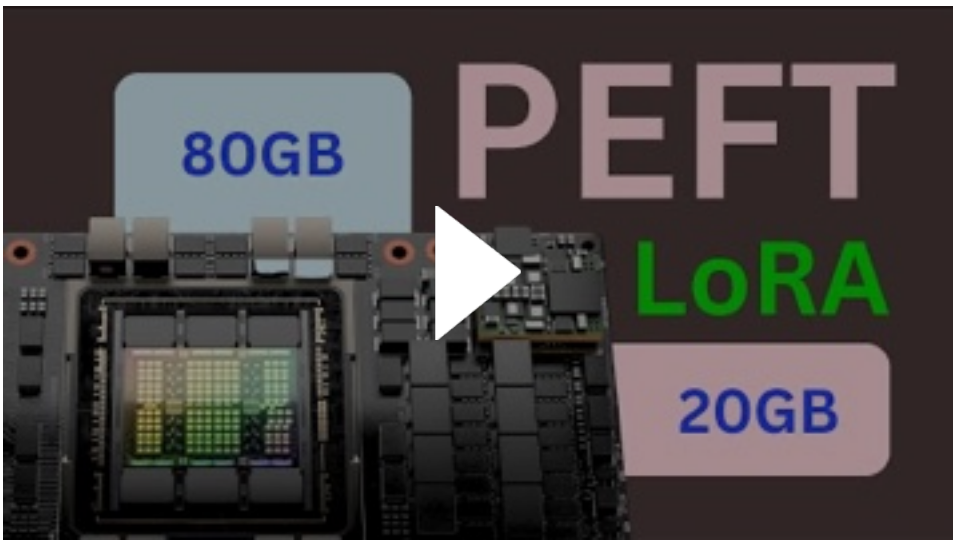
https://gitlostmurali.com/machine-learning/data-science/lora-qlora

Tuning and Testing:
https://www.sematic.dev/blog/tuning-and-testing-llama-2-flan-t5-and-gpt-j-with-lora-sematic-and-gradio?
utm_medium=web&utm_source=linkedin&utm_campaign=fine_tune_llms

PEFT:
https://lightning.ai/pages/community/article/understanding-llama-adapters/

LoRA:
https://www.linkedin.com/feed/
https://gitlostmurali.com/machine-learning/data-science/lora-qlora

http://www.aritrasen.com/generative-ai-llms-lora-fine-tuning-1-4/



QLORA stands for Quantization and Low-Rank Adapters

In this method, the original pre-trained weights of the model are quantized to 4-bit and kept fixed during fine-tuning. Then, a small number of trainable parameters in the form of low-rank adapters are introduced during fine-tuning. These adapters are trained to adapt the pre-trained model to the specific task it is being fine-tuned for, in 32-bit floating-point format.

When it comes to computations (like forward and backward passes during training, or inference), the 4-bit quantized weights are dequantized back to 32-bit floating-point numbers.

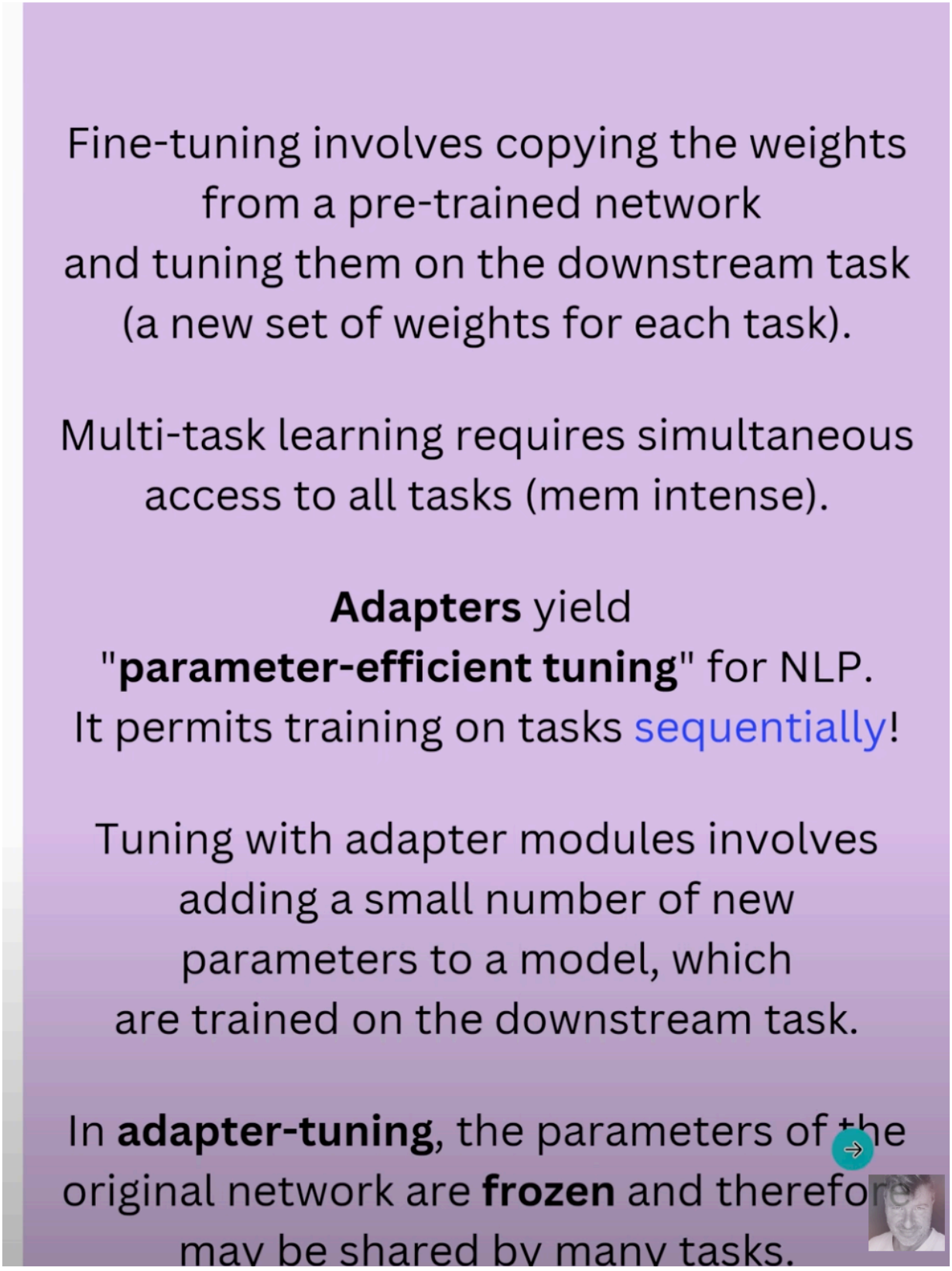
After the fine-tuning process, the model consists of the original weights in 4-bit form, and the additional low-rank adapters in their higher precision format.

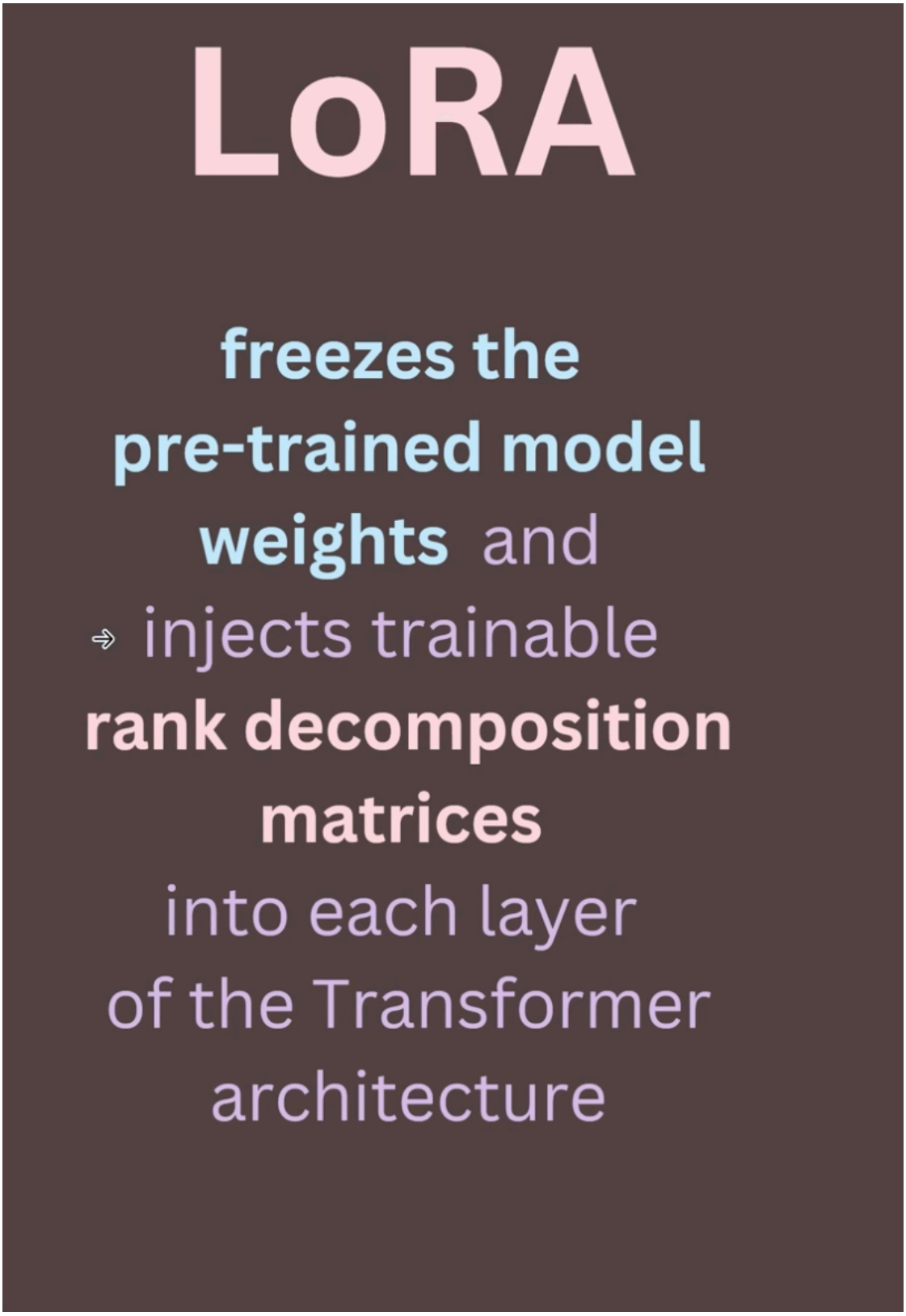
The additional low-rank adapters in the QLORA method are in a higher precision format, typically 32-bit floating-point (bflaot16), for a few reasons:

Higher precision allows the model to capture more subtle patterns in the data. This is particularly important for the low-rank adapters, as they are responsible for adapting the pre-trained model to the specific task it is being fine-tuned for.

Training neural networks involves a lot of incremental updates to the weights. Weights in a higher precision format ensures that updates are accurately captured.

While quantizing all weights can save memory, the computational efficiency might not always improve. GPUs are optimized for 32-bit (bfloat16) operations. Computations w/ 32-bit floating-point can be faster than with lower precision.



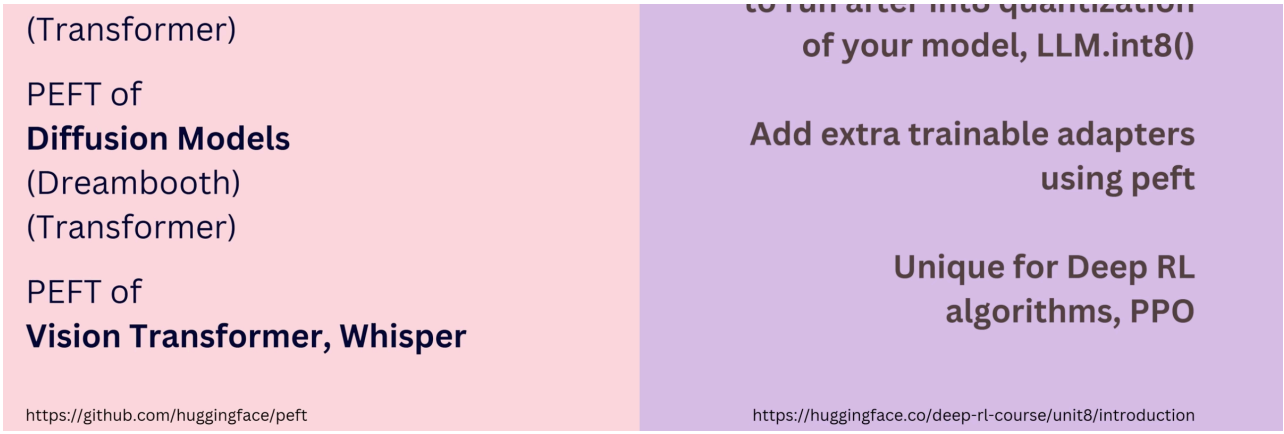


PEFT

Parameter Efficient
Fine-Tuning (PEFT)
of **LLM**

LoRA is just one method for PEFT

PEFT is ideal to run after int8 quantization



What is the relation between bitsandbytes and gptq?

GPTQ uses Integer quantization + an optimization procedure that relies on an input mini-batch to perform the quantization. Once the quantization is completed, the weights can be stored and reused.
Bitsandbytes can perform integer quantization but also supports many other formats. However, bitsandbytes does not perform an optimization procedure that involves an input mini-batch to perform quantization. That is why it can be used directly for any model.

However, it is less precise than GPTQ since information of the input data helps with quantization.

Reopen this issue if things are still unclear or you have more questions regarding the difference between GPTQ and bitsandbytes.