GPTQ Quantization on a Llama 2 7B Fine-Tuned Model With HuggingFace

A how-to easy-following guide on quantizing an LLM

Eduardo Muñoz · Follow
Published in Towards AI · 9 min read · Sep 7

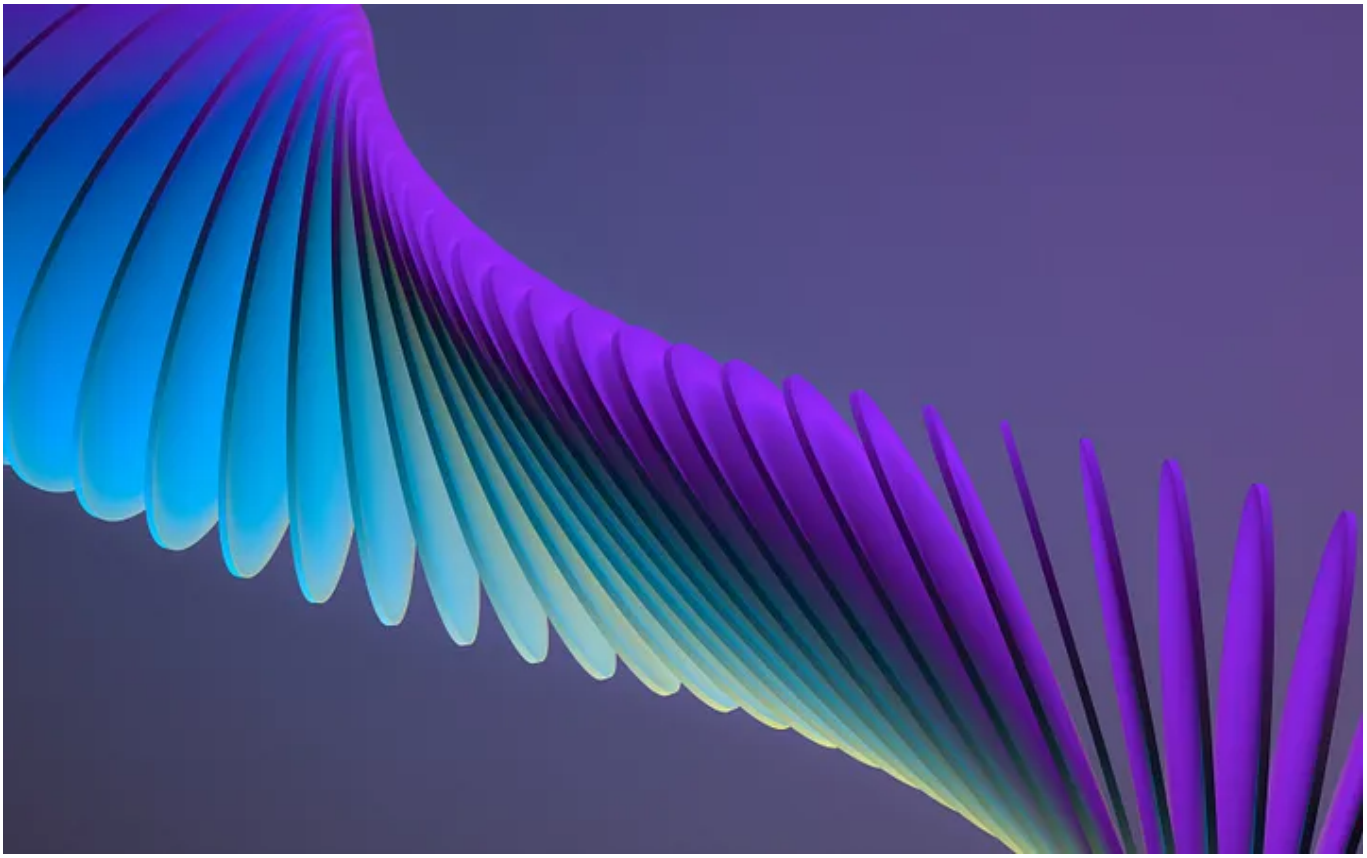


Image by Milad Fakurian on Unsplash

In my previous article, I showed you how to instruction fine-tune the new Llama 2 model, just released by Meta AI, to build a Python code generator in a few lines of code. This time, we will describe how to quantize this model using the GPTQ quantization now that it is integrated with transformers.

Last week, Hugging Face announced the compatibility of its transformers libraries with the AutoGPTQ library, which allows us to quantize a large language model in 2, 3, or 4 bits using the GPTQ methodology.

GPTQ: Post-training quantization on generative models

In a groundbreaking paper [1], researchers unveiled GPTQ, a novel post-training quantization method that has the potential to reshape the world of language model compression. GPTQ is not only efficient enough to be applied to models boasting hundreds of billions of parameters, but it can also achieve remarkable precision by compressing these models to a mere **2, 3, or 4 bits per parameter** without sacrificing significant accuracy.

This cutting-edge technique is showcased by its ability to quantize massive models, such as OPT-175B and BLOOM-176B, in just a matter of a few GPU hours while maintaining minimal perplexity, a stringent measure of accuracy. On the practical front, the researchers have developed an execution harness that enables efficient operation of the compressed models for generative tasks. Remarkably, *they achieved the milestone of running the compressed OPT-175B model on a single NVIDIA A100 GPU*, or with only two more cost-effective NVIDIA A6000 GPUs. Additionally, bespoke GPU kernels optimized for compression result in significant speedups, further enhancing the practicality of these compressed models.

What makes GPTQ stand out is its ability to quantize language models with hundreds of billions of parameters to the 3–4 bits/component range. This is a

remarkable leap, as prior methods struggled to maintain accuracy below 8 bits and typically focused on smaller models. However, the study also **highlights** the complex tradeoffs between perplexity, bit-width, and model size induced by compression. But it comes with **limitations**. GPTQ does not currently offer speedups for actual multiplications due to the lack of hardware support for mixed-precision operands on mainstream architectures. Activation quantization is also not included in the current results but can be addressed through orthogonal techniques.

In sum, GPTQ's ability to compress extremely accurate language models to unprecedented levels marks a significant milestone in the field of machine learning and language modeling. It paves the way for more efficient and accessible applications of these colossal models while pointing toward further research possibilities in the realm of model compression.

Quantization methods in machine learning can be categorized into two distinct approaches, each with its unique advantages:

1. Post-Training Quantization (**PTQ**): In PTQ, pre-trained models are quantized using relatively moderate resources, like a calibration dataset and a few hours of computational time. This method is particularly beneficial for large models, where full retraining or fine-tuning can be cost-prohibitive.
2. Quantization-Aware Training (**QAT**): QAT, on the other hand, involves quantization applied either before model training or during subsequent fine-tuning.

GPTQ's Innovative Approach: GPTQ falls under the PTQ category, making it a compelling choice for massive models. What sets GPTQ apart is its adoption of a mixed int4/fp16 quantization scheme. Here, model weights are quantized as int4, while activations are retained in float16. During inference, weights are dynamically dequantized, and actual computations are performed in float16.

This innovative approach promises memory savings close to 4x for int4 quantization and potential speedups due to the lower bandwidth used for the weights. In practice, GPTQ is a quantization method designed for models that are already fine-tuned and ready for deployment. This method is particularly effective at reducing the precision of model weights to either 4 bits or 3 bits, although it is primarily used for 4-bit quantization.

One key feature of GPTQ is its ability to quantize models without the need to load the entire model into memory. Instead, it quantizes the model module by module, significantly reducing memory requirements during the quantization process. However, it does necessitate a small sample of data for calibration, a step that may take over an hour on a typical consumer GPU.

Image from Pythonfix in Auto-GPTQ package description.

For a detailed explanation about GPTQ, you can read the amazing article by Maxime Labonne, "4-bit Quantization with GPTQ" published by Towards Data Science. It is an article that delves into the more technical aspects for those who wish to understand the details of this process.

And I highly recommend the useful article "GPTQ or bitsandbytes: Which Quantization Method to Use for LLMs — Examples with Llama 2" by Benjamin Marie. There, you can see a comparison between GPTQ and bitsandbytes quantization, pros, and cons, that way you can better understand when it is more convenient to apply each of these techniques.

¿When you should use GPTQ?

The answer to this question will depend on each specific case and on the base model to be used, but an approach that is being applied to numerous models and that is indicated by HuggingFace, and the article I mentioned before, is the following:

- Fine-tune the original LLM model with bitsandbytes in 4-bit, nf4, and QLoRa for efficient fine-tuning.
- Merge the adapter into the original model
- Quantize the resulting model with GPTQ 4-bit

I ran the first two steps in my previous article [3] and now that the AutoGPT library is integrated with the Huggingface ecosystem, we will execute the third step in an extremely simple way.

AutoGPT integrated with Hugging Face transformers

The AutoGPTQ library emerges as a powerful tool for quantizing Transformer models, employing the efficient GPTQ method. Some efforts like GPTQ-for-LLaMa, Exllama, and llama.cpp, focuses on the quantization of the Llama architecture, but AutoGPTQ distinguishes itself by offering seamless support for a diverse array of transformer architectures.

The Hugging Face team has taken a significant step to enhance accessibility to GPTQ, and they have integrated an inclusive Transformers API, simplifying the process of Low-Level Model (LLM) quantization for a wider audience. This integration includes essential optimization options, such as CUDA kernels, catering to common use cases.

For users seeking more advanced quantization options, the Auto-GPTQ library remains a valuable resource, offering capabilities like Triton kernels and fused-attention compatibility, and ensuring versatility and adaptability in the world of transformer model quantization.

> *Our AutoGPTQ integration has many advantages:*
>
> *Quantized models are serializable and can be shared on the Hub.*
>
> *GPTQ drastically reduces the memory requirements to run LLMs, while the inference latency is on par with FP16 inference.*
>
> *AutoGPTQ supports Exllama kernels for a wide range of architectures.*
>
> *The integration comes with native RoCm support for AMD GPUs.*
>
> *Finetuning with PEFT is available.*

Extracted from the Huggingface blog article "Making LLMs lighter with AutoGPTQ and transformers" [5].

Our approach to this task

First, we will load our fine-tuned model Llama 2 7B 4-bit Python coder in a Colab session using a T4 with extra RAM. The model is loaded in 4-bit with bitsandbytes and then we execute about 12 examples to measure the inference time. In order to perform a simple evaluation of the performance at the inference time, we have taken as examples those whose input text was longer than 500 characters and in this way, we will try to better appreciate the impact of quantization during inference.

You can extract the code to load this model in the description of the model in the hugging Face Hub. In my notebook, we will describe how to perform inference on the examples mentioned.

Quantize the model using auto-gptq, 🤗 transformers, and optimum

The GPTQ quantization consumes a lot of GPU VRAM, for that reason we need to execute it in an A100 GPU in Colab. It takes about 45 minutes to quantize the model, less than $1 in Colab. You can find the code in this notebook in my repository.

First, we need to install the libraries as it is recommended in the huggingface tutorial:

```
!pip install -q -U transformers peft accelerate optimum
!pip install auto-gptq --extra-index-url https://huggingface.github.io/autogptq-
# For now, until the next release of AutoGPTQ, we will build the library from so
```

Optimum library, Hugging Face's toolkit for training and inference optimization, provides the integration of AutoGPTQ into Transformers.

The GPTQ algorithm requires calibrating the quantized weights of the model by making inferences on the quantized model. For quantizing a model using auto-gptq, we need to pass a dataset to the quantizer. This can be achieved either by passing a supported default dataset among `['wikitext2','c4','c4-new','ptb','ptb-new']` or a list of strings that will be used as your custom dataset.

Now you just need to load the model using a GPTQ configuration setting the desired parameters, as usual when working with transformers, it is very

easy:

```
from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
import torch

# Set the model to load
hf_model_repo='edumunozsala/llama-2-7b-int4-python-code-20k'
# Load the tokenizer
tokenizer = AutoTokenizer.from_pretrained(hf_model_repo, use_fast=True)
# Set quantization configuration
quantization_config = GPTQConfig(
     bits=4,
     group_size=128,
     dataset="c4",
     desc_act=False,
     tokenizer=tokenizer
)
# Load the model from HF
quant_model = AutoModelForCausalLM.from_pretrained(hf_model_repo,
                quantization_config=quantization_config, device_map='auto')
```

As mentioned, this code takes about 45 minutes to run and consumes a peak of 32 GB of GPU VRAM. *"You will need a GPU to quantize a model. We will put the model in the CPU and move the modules back and forth to the GPU in order to quantize them. If you want to maximize your GPUs usage while using CPU offload, you can set* `device_map = "auto"` *"* [6], hugging Face docs.

Parameters are self-explained, 4-bit quantization, C4 dataset, and the tokenizer to use during quantization. The other two parameters take the default values:

- **group_size:** The group size to use for quantization. Recommended value is 128 and -1 uses per-column quantization

- **desc_act:** Whether to quantize columns in order of decreasing activation size. Setting it to False can significantly speed up inference but the perplexity may become slightly worse. Also known as act-order.

Once you have your model quantized, it is time to upload it to the Huggin Face Hub and share it with the community.

```
quant_model.push_to_hub("edumunozsala/llama-2-7b-int4-GPTQ-python-code-20k")
tokenizer.push_to_hub("edumunozsala/llama-2-7b-int4-GPTQ-python-code-20k")
```

In my experiment using GPTQ, the reduction in model size is striking. **My fine-tuned Llama 2 7B model with 4-bit weighted 13.5 GB on disk, but after quantization, its size was dramatically reduced to just 3.9 GB**, a third of the original size. This feature is very attractive when deploying large language models.

Loading the GPTQ Model from Hugging Face Hub and making some inferences

Probably, all of you know how to do that but just in case you think this could be more “trickier” than with other models, we will show you that it is as usual.

Remember you need to load all the libraries, including optimum, accelerate, and, of course, auto-gptq .

```
!pip install -q -U transformers peft accelerate optimum
```

```
!pip install auto-gptq
```

Then you can upload the tokenizer and the model into your notebook in a **T4 GPU** in Google Colab:

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

model_id = "edumunozsala/llama-2-7b-int4-GPTQ-python-code-20k"

tokenizer = AutoTokenizer.from_pretrained(model_id)

model = AutoModelForCausalLM.from_pretrained(model_id,
                  torch_dtype=torch.float16, device_map="auto")
```

Now we can check our GPU to confirm how much memory we are consuming and, indeed, we can see that **the model occupies 5,053 GB.**

We repeat the performance evaluation we mentioned earlier, making inferences on a bunch of long examples to compare with the original model. Both inference processes were executed on a T4 GPU, **the base model took about 17–19 seconds per inference while the quantized model ran in about 8 to 9 seconds per inference**, a half.

All code and examples are well-explained in his notebook in my repository. Any suggestion or bug fixing is welcome.

References

[1] ICLR 2023 paper "GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers"

[2] "GPTQ or bitsandbytes: Which Quantization Method to Use for LLMs — Examples with Llama 2" by Benjamin Marie.

[3] "Fine-Tuning a Llama 2 7B Model for Python Code Generation" by Eduardo Muñoz

[4] Original fine-tuned model in Huggingface "edumunozsala/llama-2–7b-int4-python-code-20k"

[5] Hugging Face blog article "Making LLMs lighter with AutoGPTQ and transformers"

[6] Hugging Face official documentation about GPTQConfig

[7] "4-bit Quantization with GPTQ" by Maxime Labonne.

Machine Learning | Transformers | Data Science | Python | Hugging Face

Jon Gi... in Towards Data ...

Characteristics of Word Embeddings

✦ · 11 min read · Sep 4, 2021

Jon Gi... in Towards Data ...

The Word2vec Hyperparameters

✦ · 6 min read · Sep 3, 2021

Jon Gi... in

The Word2ve

✦ · 15 min rea

View list

Written by Eduardo Muñoz

406 Followers · Writer for Towards AI

Follow

A Data scientist and Machine Learning practitioner and involved in NLP tasks and advances. Experienced Project Management Lead. Learning every day.

More from Eduardo Muñoz and Towards AI

Eduardo Muñoz in Towards Data Science

Attention is all you need: Discovering the Transformer paper

Nour Islam Mokhtari in Towards AI

How to Extract Key Information from Business Documents using...

Detailed implementation of a Transformer model in Tensorflow

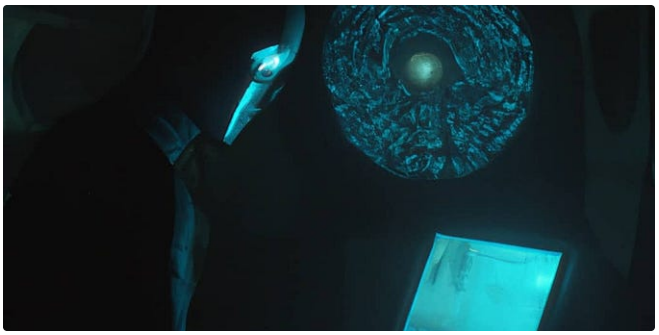
13 min read · Nov 2, 2020

577 8

A quick guide on how to use LayoutLMv3 to streamline business documents,...

✦ · 3 min read · Aug 16

166 1



Pere Martra in Towards AI

Create Your Own Data Analyst Assistant With Langchain Agents

Allow me to share my personal opinion on LLM Agents: They are going to revolutionize...

13 min read · Aug 5

176 2

Eduardo Muñoz in Better Programming

A Guide to the Encoder-Decoder Model and the Attention...

Create and train a neural machine translation model with attention in TF2

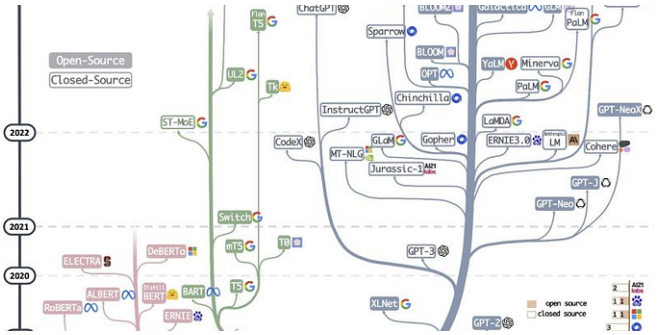
12 min read · Oct 12, 2020

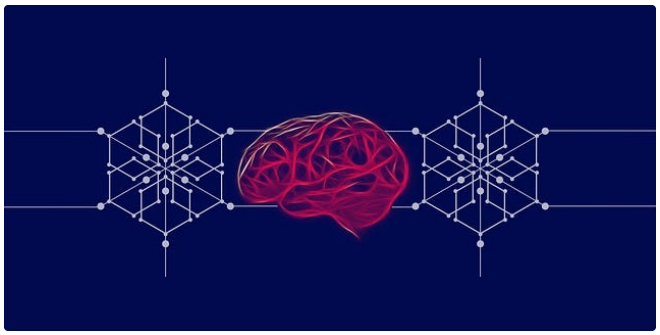
230 5

See all from Eduardo Muñoz

See all from Towards AI

Recommended from Medium





Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14

372

ai geek (wishesh)

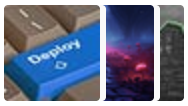
Best Practices for Deploying Large Language Models (LLMs) in...

Large Language Models (LLMs) have revolutionized the field of natural language...

10 min read · Jun 26

100 1

Lists



Predictive Modeling w/ Python

20 stories · 452 saves

Practical Guides to Machine Learning

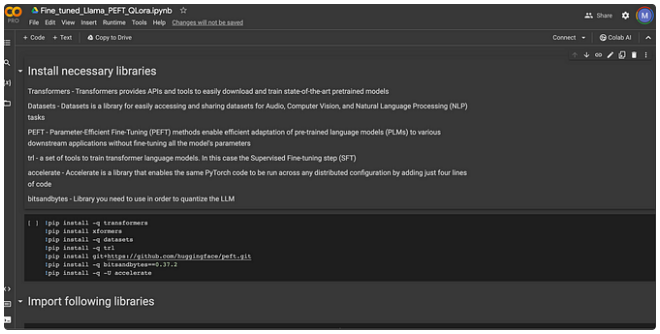
10 stories · 519 saves

Coding & Development

11 stories · 200 saves

New_Reading_List

174 stories · 133 saves



David Shapiro

A Pro's Guide to Finetuning LLMs

Large language models (LLMs) like GPT-3 and Llama have shown immense promise for...

12 min read · Sep 23

283 6

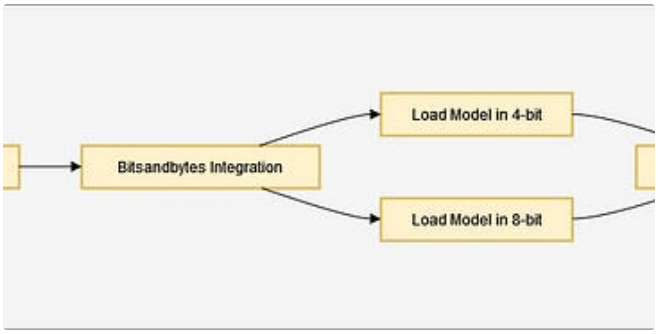
Maya Akim

Complete Guide to LLM Fine Tuning for Beginners

Fine-tuning a model refers to the process of adapting a pre-trained, foundational model...

5 min read · Aug 14

112 1



Rakesh Rajpurohit

Model Quantization with 🤗 Hugging Face Transformers and...

Introduction:

4 min read · Aug 20

31

Han HELOIR, Ph.D. in Artificial Corner

MongoDB and Langchain Magic: Your Beginner's Guide to Setting...

Introduction:

✦ · 7 min read · Sep 12

1.4K 12

See more recommendations