Token Generation Pipeline for LLM

Use case with HuggingFace

Objectifs

Due to the revolution of LLM model in the world of AI, companies want to integrate its performances into their services.

However the use of the API can be expensive and the models are too large to fit on regular devices. The solution will be to split parameters over to disk and ram.

Documentations

The solution is divided on several parts and parameters and we use some high level packages developed by HuggingFace based on the following documentations:

1. https://huggingface.co/blog/accelerate-large-models

2. https://huggingface.co/docs/accelerate/usage_guides/big_modeling

3. https://huggingface.co/docs/transformers/main/main_classes/quantization

Solution

Many aspects of the model have to be taken with consideration like the parameters format and its architecture.

However, when you load a model directly with HuggingFace package, you load the architecture and the parameters.

So, the main idea is to start with an empty model and make sure on which device to store each layer (submodule).

1. We define the checkpoint of the model we want to load.
2. We load the empty model
3. We define the device mapping
4. We configure a quantization object
5. We load the model form HuggingFace set up with the previous steps and an offload folder.

Example

We load the model Bloom-3B, with 3 billions of parameters. It needs 24 Go in the RAM to be able to be loaded. However, the RAM of the T4 colab gpu is 15 Go.

The solution organize the memory ingestion between the CPU RAM, GPU RAM and the drive.

```python
# define model checkpoint
checkpoint = "bigscience/bloom-3b"

# define no splitting block name for device mapping
no_split_block = 'BloomBlock'

# load empty model
config = AutoConfig.from_pretrained(checkpoint)
with init_empty_weights():
    model = AutoModelForCausalLM.from_config(config)

# device block mapping
device_map = infer_auto_device_map(model,
                                   no_split_module_classes=[no_split_block],
                                   dtype='float16')

# set up a quantization
quantization_config = BitsAndBytesConfig(llm_int8_enable_fp32_cpu_offload=True,
                                         llm_int8_threshold=6.0,
                                         llm_int8_skip_modules=["lm_head"])

# load model from shraded files
model = AutoModelForCausalLM.from_pretrained(checkpoint,
                                             device_map=device_map,
                                             offload_folder="offload",
                                             offload_state_dict=True,
                                             load_in_8bit=True,
                                             quantization_config=quantization_config,
                                             )

# load tokenizer
tokenizer = AutoTokenizer.from_pretrained(checkpoint)

# generate inputs
prompt = "The quick brown fox"
max_length = 50
inputs = tokenizer(prompt, return_tensors="pt").to('cuda')

# get outputs
outputs = model.generate(inputs["input_ids"], max_length=max_length)

# decode outputs
decoded_outputs = tokenizer.batch_decode(outputs, skip_special_tokens=True)

# print outputs
print(decoded_outputs)
```

Conclusion

This work was done on a very short amount of time (1 week) and I needed to explore very new technologies that never stops to be improved.

With the GPU device offered by Google Colab the pipeline works for “small” LLM models - with more than 6B of parameters the colab session crashes.

The pipeline depends a lot on the parameters format and can lose a part of its performance.