



## 自制大模型推理框架-第7次课程-模型的量化

# 自制大模型 推理框架

带你从零写一个支持LLama2/3推理  
支持Cuda加速和int8量化的大模型框架

## 课程亮点

### 一、项目整体架构和设计

学习架构思维，我们将在这里学习算子、张量等关键数据结构的设计。

### 二、支持LLama2模型结构

对LLama模型的权重读取和构建，用KV Cache等机制支持加速推理。

### 三、模型的量化

为了减少显存的占用，我们会一起开发int8模型量化模块，包括量化模型导出和量化算子支持。

### 四、Cuda基础和算子实现

带你学Cuda并实战大模型算子的实现，每个算子都将讲解原理，从零手写。

### 五、用推理框架做点有趣的事情

文本生成，生成故事，多轮对话。



现价优惠 扫码马上报名

# 视频

[📄 LLama模型的量化.mkv](#)

重置后的视频

[📄 重置后的第7次课程.mp4](#)

## 本节使用到的int8权重文件

[https://huggingface.co/fushenshen/lesson\\_model/blob/main/chat\\_q8.bin](https://huggingface.co/fushenshen/lesson_model/blob/main/chat_q8.bin)

用的时候需要注意一点，我们需要修改代码目录demo/main.cpp中的最后一个参数为true，表示当前加载的是一个量化后的模型，请切记这一点，如果不改的话会导致程序运行段错误。

```
1  model::LLama2Model model(tokenizer_path, checkpoint_path, false);
```

## 导出

我们为课程/项目提供了一个模型导出工具，可能同学们还没用过。在这里要先感谢万能的Andrej karpathy，是他为我们的项目提供了一套模型权重文件的导出工具。我们先来看看这套工具（请见 `tools/export.py`）的逻辑：

### 1. 使用 `transformers` 库加载 `llama` 结构的模型

```
1  hf_model = AutoModelForCausalLM.from_pretrained(model_path)
2  hf_dict = hf_model.state_dict()
```

### 2. 从模型的配置信息 `config.json` 构造模型参数

```
1  if any(['config.json' in path for path in os.listdir("./")]):
2      with open(os.path.join("./", 'config.json'), 'r') as f:
3          config_json = json.load(f)
4          config.dim = config_json["hidden_size"]
5          config.n_layers = config_json["num_hidden_layers"]
6          config.n_heads = config_json["num_attention_heads"]
7          config.n_kv_heads = config_json["num_key_value_heads"]
8          config.vocab_size = config_json["vocab_size"]
9          config.hidden_dim = config_json["intermediate_size"]
10         config.norm_eps = config_json["rms_norm_eps"]
11         config.max_seq_len = config_json["max_position_embeddings"]
```

3. 根据配置信息创建一个待导出的模型，并从Hugging Face的权重列表中逐一加载权重，将其赋值给该模型；

```
1 model = Transformer(config)
2
3 model.tok_embeddings.weight =
  nn.Parameter(hf_dict['model.embed_tokens.weight'])
4 model.norm.weight = nn.Parameter(hf_dict['model.norm.weight'])
```

4. 为导出的模型配置权重，权重来自Hugging Face的预训练权重；

```
1 for layer in model.layers: # 把预训练的权重放到model的各层中
2     i = layer.layer_id
3     layer.attention_norm.weight = nn.Parameter(hf_dict[f'model.layers.
  {i}.input_layernorm.weight'])
4     ...
5     ...
6     layer.feed_forward.w2.weight = nn.Parameter(hf_dict[f'model.layers.
  {i}.mlp.down_proj.weight'])
7     layer.feed_forward.w3.weight = nn.Parameter(hf_dict[f'model.layers.
  {i}.mlp.up_proj.weight'])
```

5. 开始导出模型的权重，这里我们需要导出的是int8权重

### 5.1 首先打开输出的权重文件，filepath是我们要导出的量化模型权重路径

```
1 out_file = open(filepath, 'wb')
```

### 5.2 导出模型的配置参数

```
1 hidden_dim = model.layers[0].feed_forward.w1.weight.shape[0]
2 p = model.params
3 shared_classifier = torch.equal(model.tok_embeddings.weight,
  model.output.weight)
4 # legacy format uses negative/positive vocab size as a shared classifier
  flag
5 if not shared_classifier:
6     p.vocab_size = -p.vocab_size
7 n_kv_heads = p.n_heads if p.n_kv_heads is None else p.n_kv_heads
8 group_size = 64
```

```

9
10 header = struct.pack('iiiiiii', p.dim, hidden_dim, p.n_layers, p.n_heads,
11                        n_kv_heads, p.vocab_size, p.max_seq_len, group_size)
12 out_file.write(header)

```

**5.3 struct.pack**简单理解就是把这几个参数(**p.dim, hidden\_dim, p.n\_layers, p.n\_heads, n\_kv\_heads**)打包到一起, 紧凑地放到header中, 比如这几个配置是32, 64, 128, 256, 16, 那么pack后就是这几个数字的二进制数据。

举个例子, 32就是 `0b100000`, 64就是 `0b1000000`, 我们按照每个数字占用4个字节将它们写入到二进制文件 `out_file` 中, 我们随后需要在C++中把它给读出来。在第10行代码中的, `struct.pack('iiiiiii')`表示我们将这8个参数每个都是以int类型排布到header变量的内部存储空间中, 每个参数占用4个字节。

```

1 import struct
2 demo_struct = struct.pack('iiii', 399, 7, 21, 33)
3 print(demo_struct.hex())
4
5 输出: 8f010000070000001500000021000000

```

让我们来逐个解析这些数字:

- `399` 在十六进制中是 `018f`。在小端模式下, 它会被存储为 `8f010000`。01占用一个字节, 8f占用一个字节。
- `7` 在十六进制中是 `07`, 在小端模式下, 它会被存储为 `07000000`。
- `21` 在十六进制中是 `15`, 同样地, 在小端模式下, 它会被存储为 `15000000`。
- `33` 在十六进制中是 `21`, 在小端模式下, 它会被存储为 `21000000`。

因此, 当打印 `demo_struct` 的十六进制形式时, 你会看到每个整数都被转换成了一个4字节的十六进制数, 并且由于小端模式, 最高字节 (most significant byte) 出现在最后。这就是为什么我们会得到 `8f010000070000001500000021000000` 这样的输出。

## 6. 模型参数部分的量化和导出

```

1 for layer in model.layers:
2     q, s, err = quantize_q80(layer.attention.wq.weight, group_size)
3     serialize_int8(out_file, q)
4     serialize_fp32(out_file, s) # layer.attention.wq.weight共有w个权重

```



我们采用了按组量化的方法。具体来说，将一组浮点数 [3,5,2,4] 分成两个组，每个组中包含两个 fp32 数据：在[[3,5], [2,4]]中，我们计算每个组的最大值，得到 [5,4]。我们将 int8 数据类型的最大可表示值 qmax 设定为 127。基于此，我们计算每个子集的量化比例（scale），即：scale = (5 / 127, 4 / 127)  $\approx$  (0.03937008, 0.03149606)，分别记作 scale1 和 scale2，qmax 表示 int8 的一个最大值，是 127。

我们将两组中的四个浮点值分别用各自组的量化比例进行量化：quant value 1 = round(3 / scale1)，quant value 2 = round(5 / scale1)，quant value 3 = round(2 / scale2)，我们用宽泛的方式进行分析，在逐组量化中，假设我们有 W 个权重，我们将它分成 G 组，每组的权重个数是 W/G 个。在每组中我们都求得一个最大值，记作 RMAX，共有 group G 个。再举个实际的例子，例如现在有 4 个权重数据，分别是 [1, 3, 5, -1]，每组的权重是两个。我们求出每组的最大值分别是 3 和 5，根据对称量化公式

$$Scale = \frac{|r_{max}|}{|q_{max}|}$$

这里的量化系数是按组来求得的，随后我们再对输入数据进行量化，其中 r 表示原始浮点数据，q 表示量化后输出的整型数据。

$$q = Round(\frac{r}{scale})$$

第一组：q = r = [3,5] / 该组的量化系数 = 76.19999756, 126.99999594，round 之后 76, 127

第二组：q = r = [2,4] / 该组的量化系数 = 63.50000603, 127.00001207，round 之后 64, 127

- 首先将 [3,5,2,4] 4 个权重分成两组，每组是 2 个 fp32 数据。
- 随后对每组求出一个最大值分别 5, 4
- 随后对各组求出 scale，qmax = 127，scale 各组分别等于 0.03937008, 0.03149606
- 量化的值 round(3/0.039) = 76，round(2/0.031) = 64
- 对应到代码，我们就是

```
1  for layer in model.layers:
2      q, s, err = quantize_q80(layer.attention.wq.weight, group_size)
3      serialize_int8(out_file, q)
4      serialize_fp32(out_file, s)
```

其中 q 是量化后的整型数据，s 是求得的量化系数。比如说原先要保存 1024 个 float32 数据，如果不量化的话需要占用 4096 个字节。如果现在进行量化，并且组的大小 (group size) 为 64，也就是每 64 个浮点数据共用一个量化系数 scale。那么现在占用的只有 1024 个 int8 数据，加上 1024 / 64 = 16 个 float32 数据 (scale，每组 64 个浮点权重共享)，总共占用 1024 + 64 字节，大大减少了空间的占用。当完成以上的步骤时，我们输出的权重文件大致有以下的排布。

```

2  part 1
3  模型的配置参数
4  -----
5  part 2
6  w1 layer: [int8权重参数] × layer num + [权重系数] × layer num
7  w2 layer: [int8权重参数] × layer num + [权重系数] × layer num
8  -----
9  part 3
10  不参与量化的权重
11  -----

```

如上是模型导出后的模型权重文件的数据排布情况。

1. 首先是模型的配置参数，包括 `layer num`，`hidden num` 等信息。
2. 随后就是模型中各层的参数，依次存放的是**各类型各层**量化后的权重和每组共享的量化系数。
3. 最后就是不参加量化的权重，例如 `embedding table` 和 `rmsnorm` 层的权重等。


## 命令行导出权重的办法

```

1  python export_llama.py --version 3 --hf TinyLlama/TinyLlama-1.1B-Chat-v1.0
    chat_q8.bin

```

`TinyLlama/TinyLlama-1.1B-Chat-v1.0` 是我们指定的 `huggingface` 模型名称，这里要注意的是只能选取LLama系列的模型。**export\_llama.py**是我们要用到的导出脚本，用于llama2模型的导出。

 在使用导出的权重文件后，由于是LLama2的推理阶段，所以我们需要关闭Llama3的选项，也就是需要设置

```
-DLLAMA3_SUPPORT=OFF
```

其中export\_llama2.py和config.json文件位于KuiperLlama项目的tools文件夹下。

## 加载

### 加载参数

我们先来看看模型的权重参数配置部分是怎么被读取的，在 `model.cpp` 中我们直接从二进制权重文件的首部读取模型的配置文件。也就是刚才我们在Python端导出的一组参数，它们分别是`p.dim`, `hidden_dim`, `p.n_layers`, `p.n_heads`, `n_kv_heads`, `p.vocab_size`, `p.max_seq_len`, `group_size`

```

1  auto config = ModelConfig{};
2  if (fread(&config, sizeof(ModelConfig), 1, file) != 1) {
3      return error::ModelParseError(
4          "Failed to retrieve the configuration information from the model "
5          "file.");
6  }

```

我们来看一下 `ModelConfig` 的结构，我们这里是从二进制模型权重文件头部中读取相关的配置信息。并存放在这个结构中。

```

1  struct ModelConfig {
2      int32_t dim = 0;
3      int32_t hidden_dim = 0;
4      int32_t layer_num = 0;
5      int32_t head_num = 0;
6      int32_t kv_head_num = 0;
7      int32_t vocab_size = 0;
8      int32_t seq_len = 0;
9  };

```

我们在视频中来看看这部分读取后是什么样子的，和存放进去时候的数值是不是保持相同的。

## 加载权重

我们来看看当文件被打开后，是如何从二进制模型文件中加载权重的。打开大型文件的方法我们会在以后的课时中讲到，这里我们先看看在文件打开后是如何加载的，以 `query` 层为例。

```

1  for (int32_t i = 0; i < config->layer_num; ++i) {
2      auto wq = std::make_shared<op::MatmulLayer>(device_type_, dim, dim, true);
3      wq->set_group_size(group_size_);
4      wq->set_weight(0, {dim, dim}, this->raw_model_data_->weight(pos),
5          cpu_device_type);
6      llama_layers_->wq_layers_.push_back(wq);
7      pos = pos + dim * dim + wq->get_scale_num() * sizeof(float);
8  }

```

其中 `pos` 指向我们当前权重文件中的偏移位置，`wq` 是新初始化出来的线性层，我们在 `set_weight` 中对它完成权重赋值。我们还记得在  $N \times N$  个权重之后还存放了  $N \times N / \text{GROUP SIZE}$  个量化系数，所以我们需要用两部分把它读出来。



```

1  base::Status LayerParam::set_weight(int32_t idx, const std::vector<int32_t>&
   dims, const void* weight_ptr, base::DeviceType device_type) {
2
3      size_t size = std::accumulate(dims.begin(), dims.end(), sizeof(float),
   std::multiplies<>());
4      // 将模型权重文件中的权重数据赋值给buffer
5      std::shared_ptr<base::Buffer> buffer =
6          std::make_shared<base::Buffer>(size, nullptr, (void*)(weight_ptr),
   true);
7      if (device_type != base::DeviceType::kDeviceUnknown) {
8          buffer->set_device_type(device_type);
9      }

```

首先我们将权重 `weight_ptr`（维度为 `dims`）读取到 `Buffer` 结构中，这里的权重赋值直接用了指针复用的方式，而不是直接重新拷贝 `dims` 维度的权重数据到 `tensor` 中。为了存放系数数据我们还在 `Layer` 算子类中增加了一个 `scale` 类型为 `Tensor` 的变量，这就是我们在前文中说到的量化系数。

```

1  base::Status LayerParam::set_weight(...) {
2      ...
3      ...
4      tensor::Tensor weight(base::DataType::kDataTypeInt8, dims);
5      weight.set_device_type(device_type);
6      CHECK(weight.assign(buffer));
7      weights_.at(idx) = weight;
8  }

```

随后读取 `scale_nums` 个权重系数

```

1  int32_t scale_nums = weight_size / group_size_;
2  scales_ = tensor::Tensor{base::DataType::kDataTypeFp32, scale_nums, false,
   nullptr, reinterpret_cast<float*>((int8_t*)weight_ptr + weight_size)};
3
4  scales_.set_device_type(device_type);

```

权重系数开始的位置就如同我们上文所说的那样，是在权重结束作为开始的（摆放在某一层权重数据结束的位置上），也就是 `weight_ptr + weight_size` 的位置，`scales_num` 等于权重的个数除以每组的权重数量，之所以这么算是因为权重每 `group_size` 个共享一个系数。

## 计算

在完成上述的权重和系数读取之后，我们每一个 MatmulLayer 中都有一个 weight 张量（tensor），和系数数据（scales）。那么应该如何来完成计算呢？我们现在知道的信息是每个 Matmul 层中有M个权重数据为 int8 类型，且有 scale\_num 个权重系数，这里我们假设矩阵大小为2×4且 group size 等于2，原先在完成一个矩阵相乘是这样的。

weight				input	
3.1	2.1	5.1	6.3	1.6	
3.1	2.1	5.1	6.3	1.7	
				1.8	
				2.5	
quant weight					
127	86	103	127		
127	86	103	127		
scale					
0.024409449		0.049606			
0.024409449		0.049606			
		3.1	2.1	m= 3.1	scale = m/127 = 0.024409
		quant等于			
		3.1 / scale = 127			
		2.1 / scale = 86			
计算的时候					
quant weight先乘以系数					
3.1	2.099	5.109449	6.3		
3.1	2.099	5.109449	6.3		

具体的计算因为会涉及到CUDA部分所以会放到后面讲解，我们会先将quant weight中的每个整型权重按照它原本所属的组(group)去乘以每组相关的量化系数，得到浮点权重矩阵，随后再将浮点权重矩阵乘以输入矩阵得到最终的结果。

