Homework 2

Wu Benli

522031910763

Contents

1 Introduction

In this Homework, you shall yourself explore more on how to implement quantization, and deploy LLM on Nano with the multiple online resources.
We sincerely hope you enjoy your journey through this homework as we do. Have fun!

2 Q1: Quantization in PyTorch (85')

2.1 Problem Formulation

We've discussed quantization in class.
Now, try to quantize your own network in Homework 1 Q5 on your own laptop computer.
You shall then be able to answer the following questions:

1. (5') Q: Screenshot the results of "print(qconfig)". Explain the parameters of it. e.g., explain what is "per_channel_symmetric".

2. (10') Q: Implement "fuse_model" function for your model, explain how "Conv2d", "BatchNorm2d", and "ReLU" are fused. Screenshot the results of "print(model)" after you fuse it.

3. (10') Q: Calibrate and quantize your model in PyTorch. Screenshot the results of "print(model)" after you quantize it. Explain why we need calibration phase, and what we are observing during that phase.

4. (20') Q: Measure the size, inference accuracy, and inference time of your model after quantization. Compare the size, time, and accuracy before and after quantization.

5. (40') Q: Submit your quantization code, whether it's runnable or not. (Even it can't work, we attribute some points to you.)

6. (Bonus +15') Q: Quantized model can't be directly deployed on nano, we need onnx workaround. We have a line of "torch.onnx.export" in our "example.py". Search online how to install and run the "quant.onnx" you export, and deploy that stuff onto your Nano board with onnx. If onnx is impossible on nano, write in the report to inform me, and I will attributes bouns to you too. (No hints nor guides, search yourself! hahahaha!)

(**Hint**: It's suggested that you install a CPU version of Pytorch on your own computer if you don't have a powerful Nvidia GPU installed.
"pip install torch==2.0.1 torchvision==0.15.2 –index-url https://download.pytorch.org/whl/cpu" to install a cpu version pytorch on your computer, if you previously have no pytorch installed.)
(**Hint**: Refer to https://pytorch.org/docs/stable/quantization.html#post-training-static-quantization and https://pytorch.org/tutorials/advanced/static_quantization_tutorial.html for more information of how to quantize a model in PyTorch. Also, see our "example.py" and "model.py" provided.)

2.2 Solution

(Your answers go here!)
1.
From the screenshot, we can see the program use QConfig
The API is torch.ao.quantization.qconfig.QConfig(activation,weight). It provides settings (observer classes) for activations and weights respectively.

activation=HistogramObserver.with_args(dtype=torch.quint8,reduce_range=True)
HistogramObserver records the running histogram of tensor values along with min/max values. "dtype" argument to the quantize node needed to implement the reference model spec. "reduce_range" reduces the range of the quantized data type by 1 bit.
weight=PerChannelMinMaxObserver.with_args(dtype=torch.qint8, qscheme=torch.per_channel_symmetric)
PerChannelMinMaxObserver is a Observer module for computing the quantization parameters based on the running per channel min and max values.
"qscheme"=torch.per_channel_symmetric means that we quantify weight per channel instead of per tensor and the zero_point=0(symmetric).

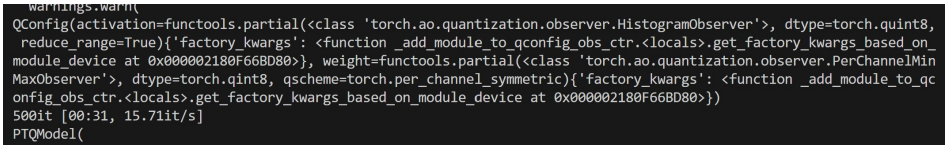

Figure 1: qconfig

2.
We can see "ConvReLU2d". This layer is fused by "Conv2d","BatchNorm2d" and "ReLU". As the TA say, the "BatchNorm2d" may be fused in "ReLU". There are some layers named "Identity()". It just output the input. It connect the layer before it and the layer after it.

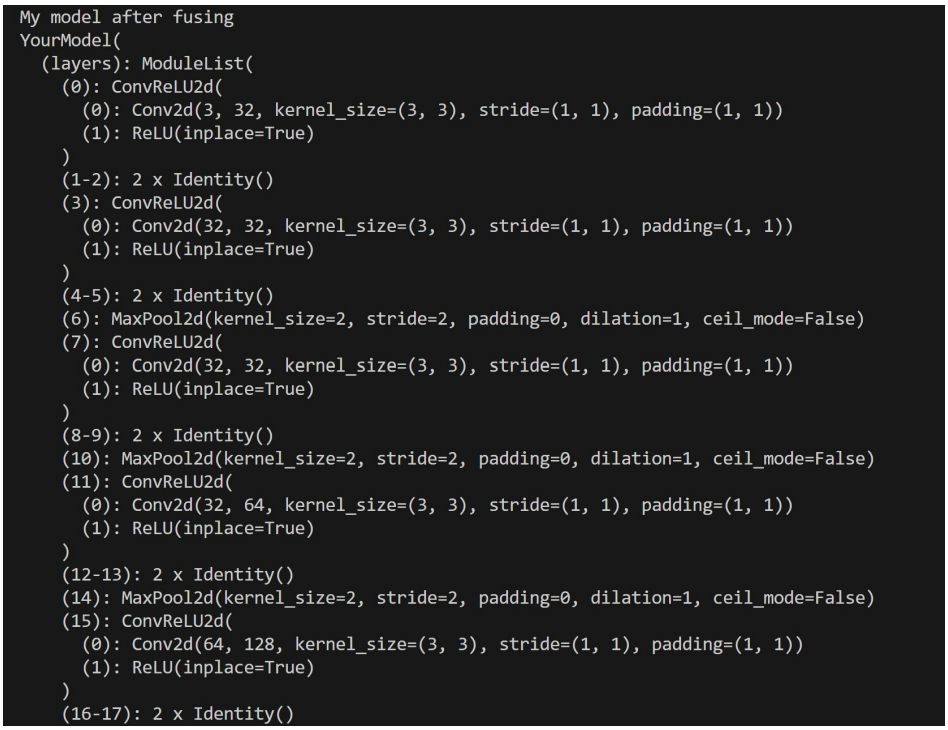

Figure 2: screenshot of model after fusing

3.
OK, I print three models in the program to make it clearer.
After preparation, we can see some observers in the model. So the process adds observers among layers. But the values of "min_value" and "max_value" are "-inf".
After calibration, we can see the values of "min_value" and "max_value" changed. It means that calibration phase can help us to observe data and get appropriate values of scale and zero_point in activation. That's why we need calibration.

```
after prepare
PTQModel(
  (fp_model): YourModel(
    (layers): ModuleList(
      (0): ConvReLU2d(
        (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (activation_post_process): HistogramObserver(min_val=inf, max_val=-inf)
      )
      (1-2): 2 x Identity()
      (3): ConvReLU2d(
        (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (activation_post_process): HistogramObserver(min_val=inf, max_val=-inf)
      )
      (4-5): 2 x Identity()
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (7): ConvReLU2d(
        (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (activation_post_process): HistogramObserver(min_val=inf, max_val=-inf)
      )
      (8-9): 2 x Identity()
      (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (11): ConvReLU2d(
        (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (activation_post_process): HistogramObserver(min_val=inf, max_val=-inf)
      )
```

Figure 3: model after preparation

```
after calibration
PTQModel(
  (fp_model): YourModel(
    (layers): ModuleList(
      (0): ConvReLU2d(
        (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (activation_post_process): HistogramObserver(min_val=0.0, max_val=10.419854164123535)
      )
      (1-2): 2 x Identity()
      (3): ConvReLU2d(
        (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (activation_post_process): HistogramObserver(min_val=0.0, max_val=13.995031356811523)
      )
      (4-5): 2 x Identity()
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (7): ConvReLU2d(
        (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): ReLU(inplace=True)
        (activation_post_process): HistogramObserver(min_val=0.0, max_val=10.28312873840332)
```

Figure 4: model after calibration

```
PTQModel(
  (fp_model): YourModel(
    (layers): ModuleList(
      (0): QuantizedConvReLU2d(3, 32, kernel_size=(3, 3), stride=(1, 1), scale=0.04528425633907318, zero_point=0, pa
dding=(1, 1))
      (1-2): 2 x Identity()
      (3): QuantizedConvReLU2d(32, 32, kernel_size=(3, 3), stride=(1, 1), scale=0.09028998762369156, zero_point=0, p
adding=(1, 1))
      (4-5): 2 x Identity()
      (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (7): QuantizedConvReLU2d(32, 32, kernel_size=(3, 3), stride=(1, 1), scale=0.0495002456009388, zero_point=0, pa
dding=(1, 1))
      (8-9): 2 x Identity()
      (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (11): QuantizedConvReLU2d(32, 64, kernel_size=(3, 3), stride=(1, 1), scale=0.03826827555894852, zero_point=0,
padding=(1, 1))
      (12-13): 2 x Identity()
      (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (15): QuantizedConvReLU2d(64, 128, kernel_size=(3, 3), stride=(1, 1), scale=0.04976480081677437, zero_point=0,
 padding=(1, 1))
      (16-17): 2 x Identity()
      (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (19): QuantizedConvReLU2d(128, 256, kernel_size=(3, 3), stride=(1, 1), scale=0.0724952295422554, zero_point=0,
 padding=(1, 1))
      (20): Identity()
      (21): Flatten(start_dim=1, end_dim=-1)
      (22): QuantizedLinear(in_features=1024, out_features=100, scale=0.700404942035675, zero_point=78, qscheme=torc
h.per_channel_affine)
```

Figure 5: model after quantization

4.
We can see:
The size of model before quantization is 4 times that after quantization.
The model after quantization takes less time.
The accuracy of the two is about the same.

Figure 6: model before quantization

Figure 7: model after quantization

3 Q2: Advanced Topic, LLM on Nano (15')

3.1 Problem Formulation

1. (12') Follow our instructions in LLM-README.md to deploy a LLM (Qwen1.5-4B) on Nano. Measure time and space it consumes. Have fun chatting with it! Screenshot your conversions.

   The resource LLM takes up.

   (Hint: If you can't deploy it, screenshot the step you get stuck and we will attribute points to you!)

2. (3') Tell me how many bit are this model's weights quantized into? 4 bits? 8 bits? No quantization? Where did you find that info?

3.2 Solution

(Your answer here!)
1.

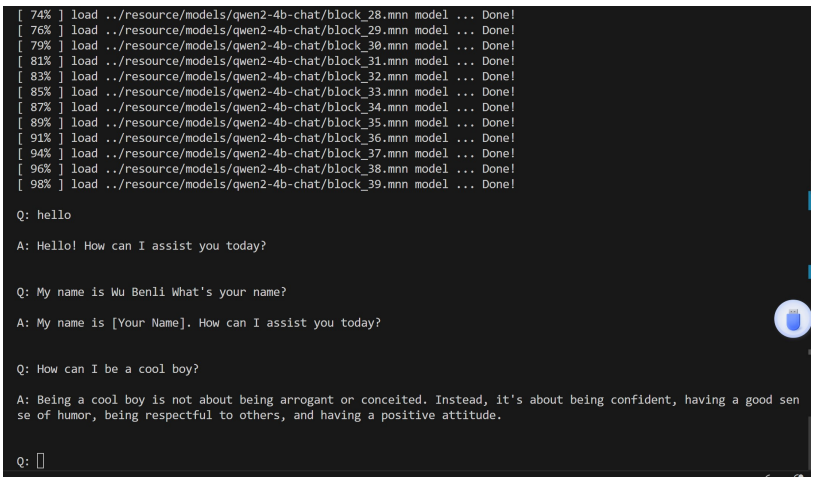

Figure 8: chat with the model

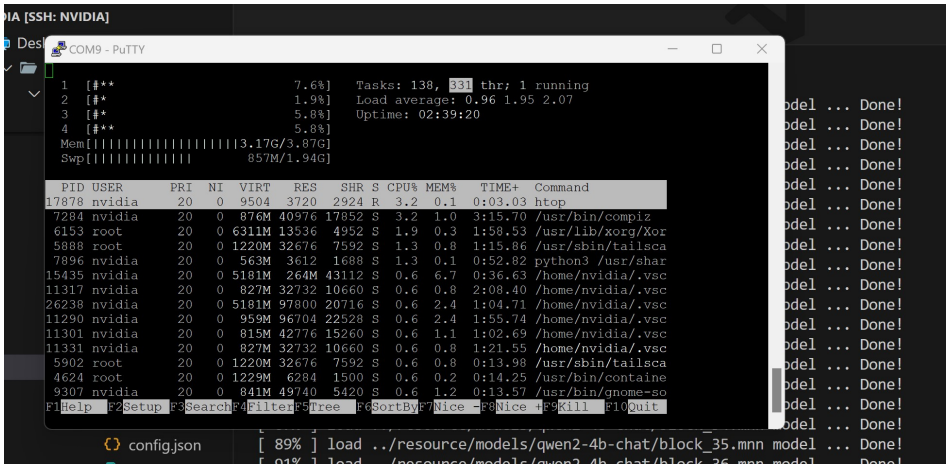

Figure 9: resource manager

2.
I guess it's 8 bits.
Because I find that there is the code "parser.add_argument("–load_in_8bit", action='store_true')" in qwen.py.

```
parser.add_argument("--load_in_8bit", action='store_true')
parser.add_argument("--cot", action='store_true')
args = parser.parse_args()
```

Figure 10: code

4 What have you learned?

4.1 Problem Formulation

(Your answer here!)
When I put BN layers after ReLU layers, the result is better.
But it can't be fused.
I don't know how order affects the result.

4.2 Tell me what you have learned

(Your answer here!)
I learned the basic knowledge about how to apply quantification in pytorch, such as qconfig, fuse and calibration.
Ok, I also deploy a LLM (Qwen1.5-4B) on Nano. It's fun.

5 Acknowledgement

(Your answer here!)
Thanks to the teacher and the teaching assistant for their guidance and assistance !