# EXAMPLE CODE DOCUMENT

## PYTORCH FLOATSD QUANTIZATION

### 介紹

FLOATSD 是一個由闕志達實驗室學長發展的智能 QUANTIZATION 套件，目前支援 FLOATSD4、INT8 和多種 FLOATSD 8-BIT 形式，透過此套件，我們可以在 PYTORCH 建立 QUANTIZATION MODEL 進行訓練(QUANTIZATION AWARE TRAINING)，或是將已經用 FP32 PRETRAINED 完成的資料，進行 POST-QUANTIZATION 和 RETRAINING，以達到和 FP32 幾乎相同的正確率。

### 操作

基本中 floatSD 是繼承 quantization pytorch 的基礎，並而外添加新的功能和資料型態，在 quantization aware training(QAT) 中我們會先配置我們的 Qconfig(如下)，而 qconfig 的生成我們只需要從 fake_quantize.py 這個檔案中，include 你所需要使用的 quantization config。

```
int_act_fake_quant = fake_quantize.default_fake_quant
log_weight_fake_quant_per_channel = fake_quantize.default_per_channel_log_weight_fake_quant
log_weight_fake_quant_per_tensor  = fake_quantize.default_log_weight_fake_quant
```

在 MNIST 的 quantization 中，activation 的 quantization configuration 為 int8，而 weight 的 quantization configuration 依據不同層的需要為 floatSD4 per tensor 或是 floatSD4 per channel，為達到 floatSD 的 quantization 的目標，無論 activation 還是 weight 的 config 都是由實驗室自己產生的。

接下來，結合我們 activation 的 configuration 和 weight 的 configuration 產生要插入 model 的 Qconfig(如下)。

```
log4_per_channel_conifg = torch.quantization.QConfig(activation=int_act_fake_quant, weight=log_weight_fake_quant_per_channel)
log4_per_tensor_config  = torch.quantization.QConfig(activation=int_act_fake_quant, weight=log_weight_fake_quant_per_tensor)
```

分別產生 per_channel 和 per_tensor 兩種不同的 floatSD4 configuration，並依據我們的需要將不同的 quantization configuration 插入對應的 model 或是 layer(如下圖)，並使用 pytorch 內建的 prepare_qat() function，如此便在 pytorch 上成功建立了一個 quantization model。

```
model.conv1.qconfig = log4_per_channel_conifg
model.conv2.qconfig = log4_per_channel_conifg
model.conv3.qconfig = log4_per_channel_conifg

model.fc1.qconfig   = log4_per_tensor_config
model.fc2.qconfig   = log4_per_tensor_config
```

另外，如果要使用 BN fusing 的功能，也可以參考 pytorch 內建 BN fusing 的 module，此功能可以幫助你將 BN 的參數並進去 CONV 裡面，如此在硬體實作上可以簡單許多。

```
for m in self.modules():
    if type(m) == ConvBNReLU:
        torch.quantization.fuse_modules(m, [['conv', 'bn', 'relu']], inplace=True) #conv + bn + relu
```

## 結果

以 MNIST 的 QAT 為例，經過 QAT 後的 model 一樣可以使用 pytorch 內建的方式存成.pt 或是.pth 檔，而以 MNIST quantization 檔案為例，所儲存的參數如下。

```
conv1.weight
conv1.bias
conv1.scale
conv1.zero_point
conv2.weight
conv2.bias
conv2.scale
conv2.zero_point
conv3.weight
conv3.bias
conv3.scale
conv3.zero_point
fc1.scale
fc1.zero_point
fc1._packed_params.dtype
fc1._packed_params._packed_params
fc2.scale
fc2.zero_point
fc2._packed_params.dtype
fc2._packed_params._packed_params
```

CONV layer 會儲存 weight, bias, scale 和 zero_point，而 fc layer 會儲存 scale, zero_point, dtype 和 _packed_params。

| 參數名稱 | 意義與型態 |
| --- | --- |
| **CONV.weight** | QTensor 型態，在儲存參數的同時，也會儲存其 scale 和 zero_point，以此來轉換成 int8 形式。 |
| **CONV.bais** | INT32 型態，用來儲存 CONV 的 BIAS |
| **CONV.scale** | FP32 型態，用來儲存 CONV 結束後到下一層的 activation quantization 的 scale |
| **CONV.zero_point** | INT32 型態，用來儲存 CONV 結束後到下一層的 activation quantization 的 zero_point |
| **FC.scale** | FP32 型態，用來儲存 FC 結束後到下一層的 activation quantization 的 scale |
| **FC.zero_point** | INT32 型態，用來儲存 FC 結束後到下一層的 activation quantization 的 zero_point |

| | |
|---|---|
| **FC._packed_params** | 其中包含 FC layer 的 Weight 和 Bias，Weight 用 QTensor 的形式儲存，一樣包含了 weight 的 scale 和 zero_point，Bias 用 INT32 儲存 |

## 檢查

在 QTensor 中儲存的參數格式如下，最前面的是其 Tensor 儲存的值，而後會儲存其 scale 和 zero_point，下圖中為 per channel 的儲存方式，所以每一個 channel 會對應一個 zero_point 和 scale。

```
  [[ 0.0000,  0.0094, -0.0752, -0.1505, -0.0188],
   [ 0.0752, -0.0376, -0.3009, -0.3009, -0.1505],
   [ 0.0047, -0.0047,  0.0000,  0.0188,  0.0000],
   [ 0.0376, -0.0376,  0.0752,  0.0752,  0.1505],
   [ 0.0752, -0.0752, -0.0047,  0.0752, -0.0376]]],
size=(32, 16, 5, 5), dtype=torch.qint8,
quantization_scheme=torch.per_channel_affine,
scale=tensor([0.0044, 0.0034, 0.0042, 0.0037, 0.0045, 0.0052, 0.0033, 0.0047, 0.0044,
 0.0042, 0.0045, 0.0031, 0.0037, 0.0039, 0.0039, 0.0040, 0.0036, 0.0046,
 0.0030, 0.0031, 0.0038, 0.0056, 0.0043, 0.0032, 0.0038, 0.0044, 0.0035,
 0.0043, 0.0044, 0.0041, 0.0044, 0.0047], dtype=torch.float64),
zero_point=tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0]),
```

而 pytorch 也提供了一個很好用的 int_repr() function，透過此 function 我們可以將儲存在 Qtensor 中的資料轉換為 INT8 的形式，此外因為 FloatSD4 的 quantization 格式，我們可以讓參數只有可能為 0, $\pm 2, \pm 4, \pm 8, \pm 16, \pm 32$ ，如此就算是用 INT8 的格式表示，除了 signed bit，最多也是有一個 bit 不為 0。

```
[[[  0,   2, -16, -32,  -4],
  [ 16,  -8, -64, -64, -32],
  [  1,  -1,   0,   4,   0],
  [  8,  -8,  16,  16,  32],
  [ 16, -16,  -1,  16,  -8]]], dtype=torch.int8)
```

## 檔案解說

在提供的資料夾底下分別有 int_quantization 的 source code, CIFAR-10, MNIST 和一個 get_param_in_pytorch.py 檔案。

### INT_QUANTIZATION:

這是 int_quantization 的 source code，裡面是建立在 pytorch 上的 quantization 系統，所要使用的 Qconfig 也都可以從裡面找到，同學如果對 quantization 的流程有興趣的，也可以自行研究一下。

### GET_PARAM_IN_PYTORCH.PY

這是一個簡單的檔案，可以提供給你檢查你 model 所產生的參數是不是符合 quantization 的標準，也可以用來看 model 的參數為何。

### CIFAR-10

這資料夾提供簡單的 CIFAR-10 quantization aware training(QAT) example，所使用的 model 為 MobileNetV2，quantization config 如下，除了 ConvBNReLU 和 Linear layer 使用 log4_per_tenosr_quantization，其他層都使用 log4_per_channel_quantization:

```
net.qconfig = log4_per_channel_config

for mod in net.modules():
    if type(mod) == ConvBNReLU and mod.groups == mod.in_channels:
        mod.qconfig = log4_per_tensor_config

    if type(mod) == torch.nn.Linear:
        mod.qconfig = log4_per_tensor_config
```

在這檔案中已有提供 fuse module 簡單的使用方法，也可以作為參考。

```
def fuse_model(self):
    for m in self.modules():
        if type(m) == ConvBNReLU:
            torch.quantization.fuse_modules(m, ['0', '1', '2'], inplace=True) #conv + bn + relu
        if type(m) == ConvBN:
            torch.quantization.fuse_modules(m, ['0', '1'], inplace=True) #conv + bn + relu
```

最後透過 pytorch 提供的 convert，我們可以將模型參數用 quantized 後的形式，並進行儲存。

### MNIST

在 MNIST 資料夾下，我們提供 pytorch_mnist_QAT.py 和 pytorch_mnist_post_quant_and_retrain.py 兩個檔案，前面是針對 MNIST 做 QAT，基本上流程和 CIFAR-10 是完全相同的。

在後者的檔案中，我們先 train 好 FP32 的模型後，我們一樣先設計我們的 Qconfig，然後使用 pytorch 內建的 prepare 功能，將我們的 model 轉換成 quantized model，並且做 testing，如此就可以得到我們 post-quant 的正確率。

```
model.qconfig = log4_per_channel_config
torch.quantization.prepare(model, inplace=True)
model.eval()
test_accuracy = test(model, device, test_loader)
```

隨後，在參照 QAT 的方式，對 model 進行 retrain，在數個 epoch 後，model 可以迅速的回到原本的正確率。