Example code Document

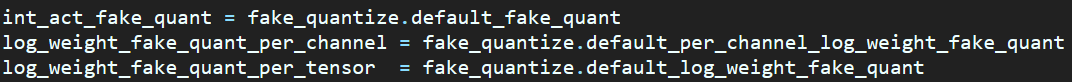
# Pytorch floatsd quantization

##### 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。

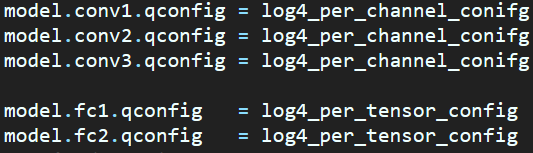


在MNIST的quantization中，activation的quantization configuration為int8，而weight的quantization configuration依據不同層的需要為floatSD4 per tensor或是floatSD4 per channel，為達到floatSD的quantization的目標，無論activation還是weight的config都是由實驗室自己產生的。

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



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。

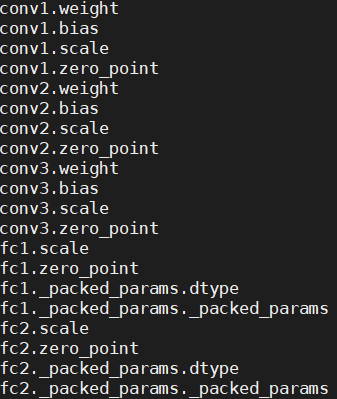


另外，如果要使用BN fusing的功能，也可以參考pytorch內建BN fusing的module，此功能可以幫助你將BN的參數並進去CONV裡面，如此在硬體實作上可以簡單許多。



##### 結果

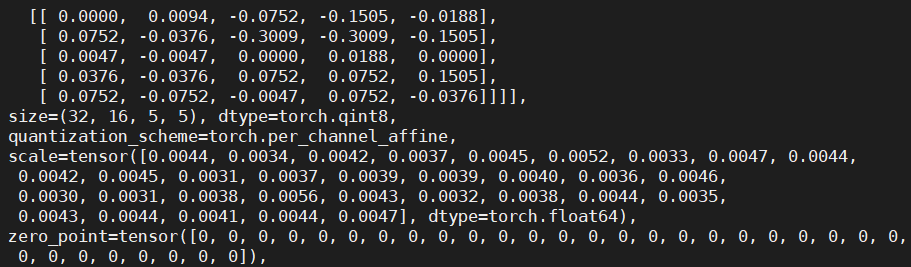
以MNIST的QAT為例，經過QAT後的model一樣可以使用pytorch內建的方式存成.pt或是.pth檔，而以MNIST quantization檔案為例，所儲存的參數如下。



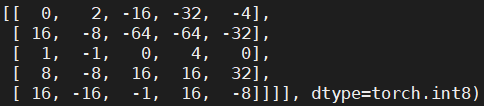
CONV layer會儲存weight, bias, scale和zero\_point，而fc layer會儲存scale, zero\_point, dtype和\_packed\_params。

|  |  |
| --- | --- |
| 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。

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, ，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。



##### 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:

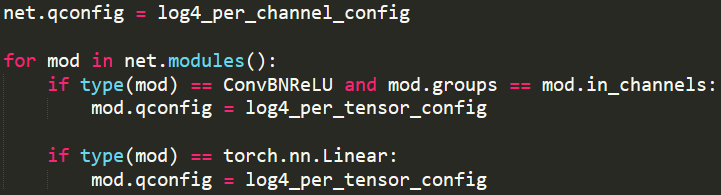
這是int\_quantization的source code，裡面是建立在pytorch上的quantization系統，所要使用的Qconfig也都可以從裡面找到，同學如果對quantization的流程有興趣的，也可以自行研究一下。

GET\_param\_in\_pytorch.py

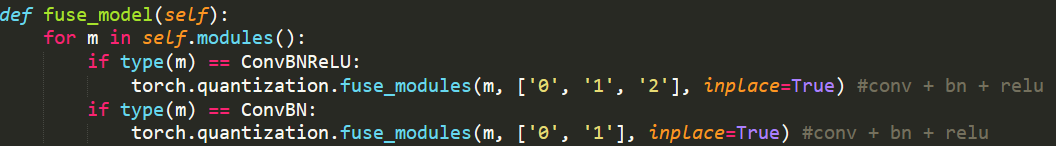
這是一個簡單的檔案，可以提供給你檢查你model所產生的參數是不是符合quantization的標準，也可以用來看model的參數為何。

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:



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。

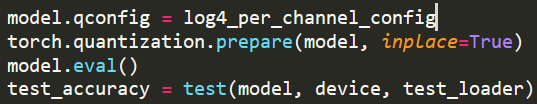


最後透過pytorch提供的convert，我們可以將模型參數用quantized後的形式，並進行儲存。

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。



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。