EAI Lab6
Quantization

Advisor：CCTsai

TA：林泳陞

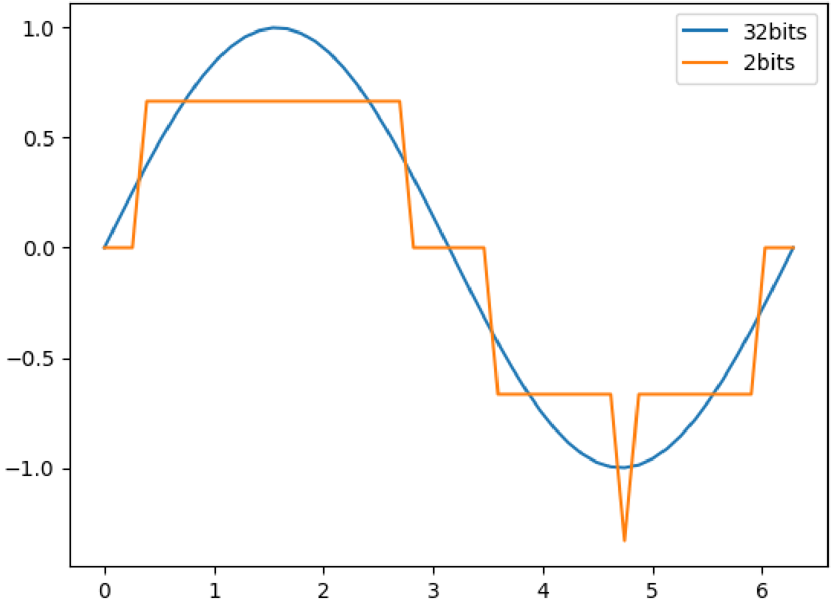
Outline

- Introduction
  - Quantization
  - PTQ and QAT

- Task
  - Example code
  - Task

Introduction - Quantization

- Quantization is a method used to reduce the model size and computation requirements by replacing floating-point 32-bit (FP32) representations with lower bit precision, such as FP16, INT8 or even less bits. However, reducing the number of bits used to represent values can lead to a loss of accuracy. The primary objective is to strike a balance between accuracy and computational resources.



Introduction - Quantization

- The simplest way to implement quantization is to map the original values to an integer range of 0 to 255 or -128 to 127, and then map these 256 integers back to the original value range to minimize the error.
  For example:

```
FP32: tensor([ 0.6004, -1.0151,  1.1885,  0.9948, -0.3187, -1.1111,  0.7827, -1.1217])
INT8: tensor([ 0.5979, -1.0147,  1.1868,  0.9966, -0.3171, -1.1143,  0.7791, -1.1234])
INT8 integer representation: tensor([  62., -116.,  127.,  106.,  -39., -127.,   82., -128.])
```

- Therefore, we need to find a method to implement quantization using scale factor and zero point.

Introduction - Quantization

- q = round($\frac{r}{s} + Z$), where $s = \frac{\beta - \alpha}{\beta_q - \alpha_q}$ and $Z = round(\alpha_q - \frac{\alpha}{s})$
- For weight, [α, β] is weight range, and [α_q, β_q] is usually [-128, 127] for int8 quantization.
- For activation, [α, β] is output range, and [α_q, β_q] is usually [-128, 127] for int8 quantization.
- r_q = (q – Z) * s

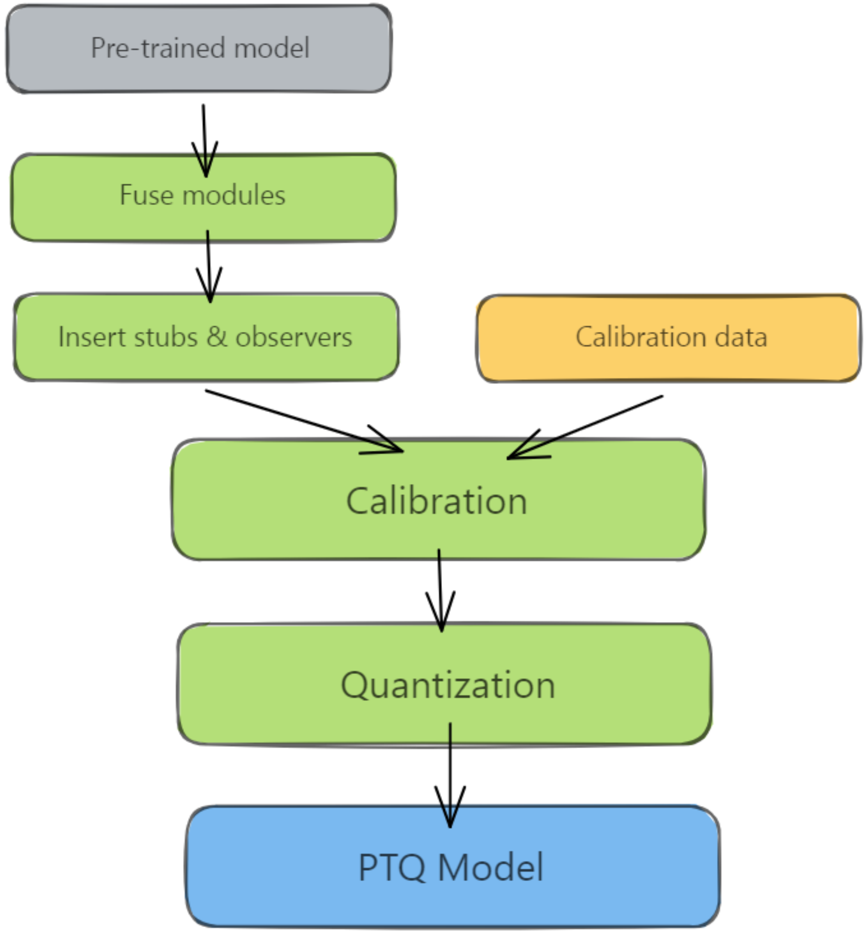
```
FP32: tensor([ 0.6004, -1.0151,  1.1885,  0.9948, -0.3187, -1.1111,  0.7827, -1.1217])
INT8: tensor([ 0.5979, -1.0147,  1.1868,  0.9966, -0.3171, -1.1143,  0.7791, -1.1234])
INT8 integer representation: tensor([  62., -116.,  127.,  106.,  -39., -127.,   82., -128.])
```

Introduction - Quantization

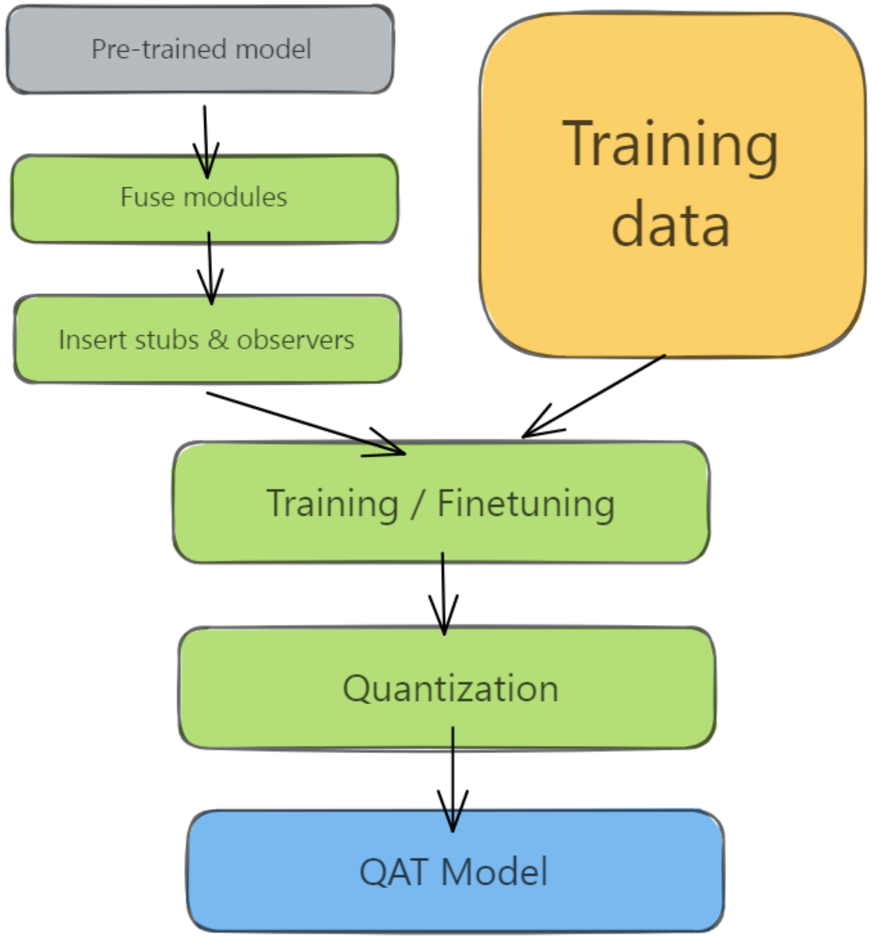
- q = clamp[round($\frac{r}{s} + Z$), -128, 127], where $s = \frac{\beta - \alpha}{\beta_q - \alpha_q}$ and $Z = round(\alpha_q - \frac{\alpha}{s})$
- For clip method, [α_q, β_q] is [-256, 255] for int8 quantization.
- For weight, [α, β] is weight range.
- For activation, [α, β] is output range
- r_q = (q – Z) * s

```
FP32: tensor([ 1.4243, -0.7382, -1.0199,  0.6312, -0.4644, -1.2151,  1.1037,  0.2785])
INT8: tensor([ 0.7644, -0.5527, -0.5527,  0.6301, -0.4649, -0.5527,  0.7644,  0.2789])
INT8 integer representation: tensor([ 127., -128., -128.,  101., -111., -128.,  127.,   33.])
```

Introduction - PTQ

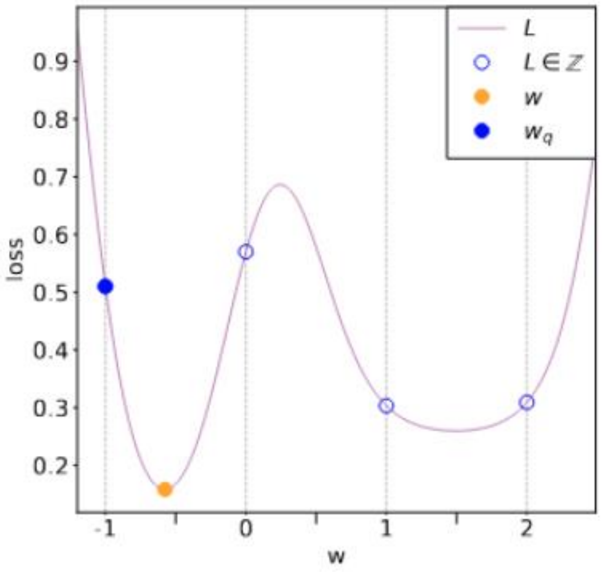


Introduction - QAT



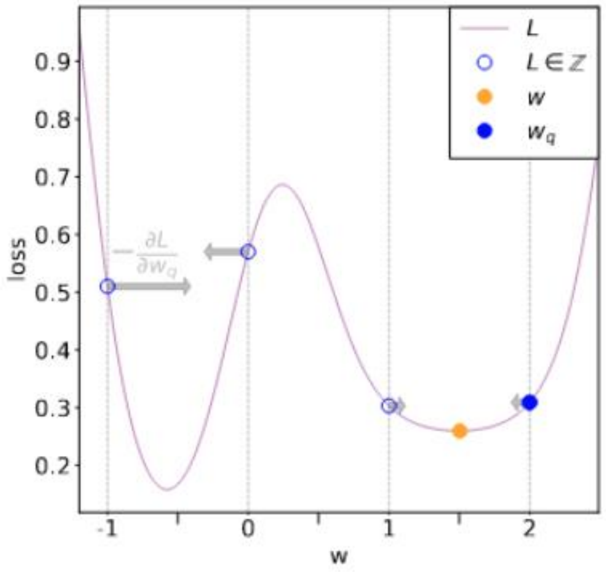
Introduction - QAT

Bad performance



(a) Post training quantization

Good performance



(b) After quantization aware fine-tuning

Example code

```
2秒 [23] compare(model=FP32_model, device="cpu", test_loader=test_loader)

========================================= PERFORMANCE =============================================
Size of the model(MB): 0.423146

Accuracy: 8391/10000 (84%)

2秒 [24] compare(model=PTQ_model, device="cpu", test_loader=test_loader)

========================================= PERFORMANCE =============================================
Size of the model(MB): 0.110646

Accuracy: 8392/10000 (84%)

2秒 [25] compare(model=QAT_model, device="cpu", test_loader=test_loader)

========================================= PERFORMANCE =============================================
Size of the model(MB): 0.110646

Accuracy: 8469/10000 (85%)
```

Example code

Quantize layer by layer

```
========================================= PERFORMANCE =============================================

Accuracy: 6655/10000 (67%)
```

Quantize at the same time

```
========================================= PERFORMANCE =============================================

Accuracy: 8396/10000 (84%)
```

MSE

```
MSE of layer quantize_per_tensor is 0.5495123863220215
MSE of layer nn1.relu is 1.3325119018554688
MSE of layer nn2.relu is 1.7615503072738647
MSE of layer dequantize is 14.916387557983398
```

Example code

```
def  Calculate_scale_zero_point(x,  mode="normal"):
    if  mode  ==  "normal":
        '''
        請完成以下程式碼
        '''


    elif  mode  ==  "clip":
        '''
        請完成以下程式碼
        '''


    return  scale,  zero_point
```

參考5、6頁的算法完成scale factor及zero point的計算

Example code

```
self.tensor = x
self.scale = scale
self.zero_point = zero_point
```

```
def _quantize(self, mode):
    if mode == "normal":
        self.qtensor_int = #請完成以下程式碼
        self.qtensor = #請完成以下程式碼

    elif mode == "clip":
        self.qtensor_int = #請完成以下程式碼
        self.qtensor_int = #請完成以下程式碼    clamp qtensor_int
        self.qtensor = #請完成以下程式碼
```

參考5、6頁的算法完成quantize的計算

Example code

```
[ ]  scale_dic = []
     zero_dic = []

     #Calibrate to compute s、z of all layer at the same time
     for batch in train_loader:
         input, label = batch
         for node in ['x', 'relu', 'relu_1', 'nn3']:
             extractor = feature_extraction.create_feature_extractor(model, [node]).cpu()
             output = extractor(input)[node]
             q_min, q_max = -128, 127
             min_val, max_val = np.min(output.detach().numpy()), np.max(output.detach().numpy())
             scale = (max_val - min_val) / (q_max - q_min)
             zero = round(q_min - min_val / scale)
             q = Quantize_per_tensor(output, scale=scale, zero_point=zero, mode="normal")
             scale_dic.append(scale)
             zero_dic.append(zero)
         break

     print(scale_dic)
     print(zero_dic)
```

與Example當中quantize at the same time的作法一樣，先計算每個layer的scale、zero point

Example code

```
[ ]  #define evaluate function
     def Evaluate(model, loader):
         total = 0
         correct = 0
         with torch.no_grad():
             for data in loader:
                 images, labels = data
                 outputs = model(images)
                 # the class with the highest energy is what we choose as prediction
                 _, predicted = torch.max(outputs.data, 1)
                 total += labels.size(0)
                 correct += (predicted == labels).sum().item()

         test_loss = 0

         print("========================================= PERFORMANCE =============================================")
         print('\nAccuracy: {}/{} ({:.0f}%)\n'.format( correct, total,100. * correct / total))
```

```
[ ]  #Normal quantize
     Evaluate(Quantized_normal_model, test_loader)
```

```
[ ]  #Clip quantize
     Evaluate(Quantized_clip_model, test_loader)
```

將兩種quantize計算方式的結果print出來並且放在結報中

Task

- Example code 30% (no need to write code)
  - Show accuracy comparison of FP32 model, PTQ model and QAT model
  - Self quantization
    - Show the MSE for each layer using two methods: quantizing layer by layer and quantizing all layers at the same time.
    - Show the difference of output distribution of above two method.
- Practice to implement quantization function 40% (Page 5)
  - Normal quantization
  - Clip quantization ($[\alpha_q, \beta_q]$ is [-256, 255] but we only use [-128, 127])
- Report and Question 30%

Task – cont.

- Report and Question 30%
  - Explain why the performance of quantize layer by layer is worse by using MSE and output distribution in the example code.
  - Besides the methods of scale factor and zero point mentioned in this lab, please provide the other example to determine the quantize value.