人工智慧模型設計與應用 Lab6

NM6121030 余振揚

1. Accuracy Comparison of FP32 Model, PTQ Model and QAT Model:

```
compare(model=FP32_model, device="cpu", test_loader=test_loader)

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

Accuracy: 8299/10000 (83%)
```

```
compare(model=PTQ_model, device="cpu", test_loader=test_loader)

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

Accuracy: 8287/10000 (83%)
```

```
compare(model=QAT_model, device="cpu", test_loader=test_loader)

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

Accuracy: 8464/10000 (85%)
```

三個模型的性能非常相近，而在量化感知訓練（QAT）相較於直接進行量化（PTQ）稍微優越是合理的。這是因為直接進行量化可能會導致一些精度損失，而經過量化感知訓練後，模型有機會補償這些損失，甚至使其性能超越原始的浮點數模型。

2. Self Quantization:

- Quantize layer by layer:

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

Accuracy: 6138/10000 (61%)
```

- Quantize all layers at the same time:

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

Accuracy: 8271/10000 (83%)
```

- MSE:

```
MSE of layer quantize_per_tensor is 0.582194447517395
MSE of layer nn1.relu is 1.2625266313552856
MSE of layer nn2.relu is 1.5307530164718628
MSE of layer dequantize is 11.38371467590332
```

3. Implement Quantization:

```python
def Calculate_scale_zero_point(x, mode="normal"):
  if mode == "normal":
    '''
    請完成以下程式碼
    '''
    q_min, q_max = -128, 127  # int8
    min_val, max_val = np.min(x.detach().numpy()), np.max(x.detach().numpy()) # get min/max value of x

    scale = (max_val - min_val) / (q_max - q_min) # calculate scale
    zero_point = round(q_min - min_val / scale) # calculate zero_point

  elif mode == "clip":
    '''
    請完成以下程式碼
    '''
    q_min, q_max = -256, 255
    min_val, max_val = np.min(x.detach().numpy()), np.max(x.detach().numpy()) # get min/max value of x

    scale = (max_val - min_val) / (q_max - q_min) # calculate scale
    zero_point = round(q_min - min_val / scale)

  return scale, zero_point

def _quantize(self, mode):
  if mode == "normal":
    self.qtensor_int = torch.round(self.tensor / self.scale + self.zero_point)  # q = round(r/s + zp)
    self.qtensor = (self.qtensor_int - self.zero_point) * self.scale  # rq = (q - zp) * scale

  elif mode == "clip":
    self.qtensor_int = torch.round(self.tensor / self.scale + self.zero_point)
    self.qtensor_int = torch.clamp(self.qtensor_int, -128, 127) # clamp to [-128, 127]
    self.qtensor = (self.qtensor_int - self.zero_point) * self.scale
```

4. Quantization Results:

- Normal:

```python
#Normal quantize
Evaluate(Quantized_normal_model, test_loader)
```

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

Accuracy: 8442/10000 (84%)
```

- Clip:

```python
#Clip quantize
Evaluate(Quantized_clip_model, test_loader)
```

```
========================================= PERFORMAN

Accuracy: 8427/10000 (84%)
```