SAVE, LOAD AND USE MODEL

一、内容

本部分将涉及如何保存和加载模型，包括三种方法：1、只保存和加载参数；2、保存和加载整个模型；3、在训练过程中保存和加载。

二、代码

1、只保存和加载参数

这样只需保存必要参数，使得模型大小减小，利于保存。

保存

```
torch.save(model.state_dict(), 'model_weights.pth')
```

这里保存上一部分内容中训练的模型。

加载

```
model = NeuralNetwork()
model.load_state_dict(torch.load('model_weights.pth'))
print(model)
```

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

2、保存和加载整个模型

保存

```
torch.save(model, 'model.pth')
```

加载完整模型

```
model = torch.load('model.pth')
print(model)
```

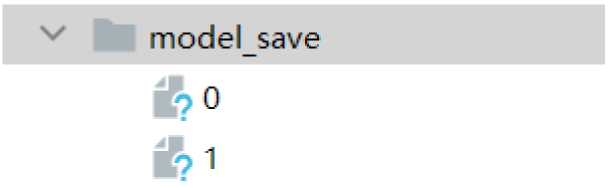
> 注意：这里不需要model = NeuralNetwork()构建模型结构，因为保存时已经存了模型的结构。

3、训练中保存

保存

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

epochs = 2
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader, model, loss_fn, optimizer)
    loss = test_loop(test_dataloader, model, loss_fn)
    torch.save({
        'epoch': t,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'loss': loss,
    }, os.path.join("model_save",str(t)))
print("Done!")
```



每一个epoch都会保存一次模型，并保存代化器。

```
model = NeuralNetwork()
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)

checkpoint = torch.load(os.path.join("model_save",str(0)))
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
loss_last = checkpoint['loss']

# 再接着从0号模型进行训练
model.train()
train_loop(train_dataloader, model, loss_fn, optimizer)
loss_now = test_loop(test_dataloader, model, loss_fn)
print("loss_last", loss_last)
print("loss_now", loss_now)
```

```
loss: 0.796059  [   64/60000]
loss: 0.338162  [ 6464/60000]
loss: 0.352287  [12864/60000]
loss: 0.463120  [19264/60000]
loss: 0.541893  [25664/60000]
loss: 0.507227  [32064/60000]
loss: 0.319748  [38464/60000]
loss: 0.460805  [44864/60000]
loss: 0.521575  [51264/60000]
loss: 0.641139  [57664/60000]
Test Error: 
 Accuracy: 87.1%, Avg loss: 0.352667 

loss_last 0.48903237672439265
loss_now 0.3526668991091282
```

接着从0号模型进行训练，这一次loss比上0号模型loss要小，说明再训练的模型性能是提高了的。