# RNN 做图像分类

```
import torch
from torch.autograd import Variable
from torch import nn
from torch.utils.data import DataLoader

from torchvision import transforms as tfs
from torchvision.datasets import MNIST
```

```
# 定义数据
data_tf = tfs.Compose([
    tfs.ToTensor(),
    tfs.Normalize([0.5], [0.5]) # 标准化
])

train_set = MNIST('./data', train=True, transform=data_tf)
test_set = MNIST('./data', train=False, transform=data_tf)

train_data = DataLoader(train_set, 64, True, num_workers=4)
test_data = DataLoader(test_set, 128, False, num_workers=4)
```

```
# 定义模型
class rnn_classify(nn.Module):
    def __init__(self, in_feature=28, hidden_feature=100, num_class=10,
num_layers=2):
        super(rnn_classify, self).__init__()
        self.rnn = nn.LSTM(in_feature, hidden_feature, num_layers) # 使用两层 lstm
        # 将最后一个 rnn 的输出使用全连接得到最后的分类结果
        self.classifier = nn.Linear(hidden_feature, num_class)

    def forward(self, x):
        """
        x 大小为 (batch, 1, 28, 28)，所以我们需要将其转换成 RNN 的输入形式，即 (28,
batch, 28)
        """
        x = x.squeeze() # 去掉 (batch, 1, 28, 28) 中的 1，变成 (batch, 28, 28)
        x = x.permute(2, 0, 1) # 将最后一维放到第一维，变成 (28, batch, 28)
        out, _ = self.rnn(x) # 使用默认的隐藏状态，得到的 out 是 (28, batch,
hidden_feature)
```

```
        out = out[-1, :, :] # 取序列中的最后一个，大小是 (batch, hidden_feature)
        out = self.classifier(out) # 得到分类结果
        return out
```

```
net = rnn_classify()
criterion = nn.CrossEntropyLoss()

optimzier = torch.optim.Adadelta(net.parameters(), 1e-1)
```

```
# 开始训练
from utils import train
train(net, train_data, test_data, 10, optimzier, criterion)
```

```
Epoch 0. Train Loss: 1.858605, Train Acc: 0.318347, Valid Loss: 1.147508, Valid Acc:
0.578125, Time 00:00:09
Epoch 1. Train Loss: 0.503072, Train Acc: 0.848514, Valid Loss: 0.300552, Valid Acc:
0.912579, Time 00:00:09
Epoch 2. Train Loss: 0.224762, Train Acc: 0.934785, Valid Loss: 0.176321, Valid Acc:
0.946499, Time 00:00:09
Epoch 3. Train Loss: 0.157010, Train Acc: 0.953392, Valid Loss: 0.155280, Valid Acc:
0.954015, Time 00:00:09
Epoch 4. Train Loss: 0.125926, Train Acc: 0.962137, Valid Loss: 0.105295, Valid Acc:
0.969640, Time 00:00:09
Epoch 5. Train Loss: 0.104938, Train Acc: 0.968450, Valid Loss: 0.091477, Valid Acc:
0.972805, Time 00:00:10
Epoch 6. Train Loss: 0.089124, Train Acc: 0.973481, Valid Loss: 0.104799, Valid Acc:
0.969343, Time 00:00:09
Epoch 7. Train Loss: 0.077920, Train Acc: 0.976679, Valid Loss: 0.084242, Valid Acc:
0.976661, Time 00:00:10
Epoch 8. Train Loss: 0.070259, Train Acc: 0.978795, Valid Loss: 0.078536, Valid Acc:
0.977749, Time 00:00:09
Epoch 9. Train Loss: 0.063089, Train Acc: 0.981093, Valid Loss: 0.066984, Valid Acc:
0.980716, Time 00:00:09
```

可以看到，训练 10 次在简单的 mnist 数据集上也取得的了 98% 的准确率，所以说 RNN 也可以做做简单的图像分类，但是这并不是他的主战场，下次课我们会讲到 RNN 的一个使用场景，时间序列预测。