# 7.2 RNN 手写字体分类

import seaborn as sns

sns.set(font\_scale=1.5, style="white")

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import time

import copy

import torch

from torch import nn

import torch.nn.functional as F

import torch.optim as optim

import torchvision

import torch.utils.data as Data

from torchvision import transforms

import hiddenlayer as hl

train\_data = torchvision.datasets.FashionMNIST( # MNIST的下载不了，这里换成FashionMNIST

root="D:\pythoncode\learn/a\deep Learning\FashionMNIST", # 数据的路径

train=True, # 只使用训练数据集

# 将数据转化为torch使用的张量,取汁范围为［0，1］

transform=transforms.ToTensor(),

# download=True

download=False

)

# 定义一个数据加载器

train\_loader = Data.DataLoader(

dataset=train\_data, # 使用的数据集

batch\_size=64, # 批处理样本大小

shuffle=True, # 每次迭代前打乱数据

num\_workers=0,

)

# 准备需要使用的测试数据集

test\_data = torchvision.datasets.FashionMNIST(

root="D:\pythoncode\learn/a\deep Learning\FashionMNIST", # 数据的路径

train=False, # 不使用训练数据集

transform=transforms.ToTensor(),

download=False # 因为数据已经下载过，所以这里不再下载

)

# 定义一个数据加载器

test\_loader = Data.DataLoader(

dataset=test\_data, ## 使用的数据集

batch\_size=64, # 批处理样本大小

shuffle=True, # 每次迭代前打乱数据

num\_workers=0,

)

# 可视化训练数据集的一个batch的样本来查看图像内容

for step, (b\_x, b\_y) in enumerate(test\_loader):

if step > 0:

break

# 输出训练图像的尺寸和标签的尺寸，都是torch格式的数据

print(b\_x.shape)

print(b\_y.shape)

# 可视化训练数据集的一个batch的样本来查看图像内容

for step, (b\_x, b\_y) in enumerate(train\_loader):

if step > 0:

break

# 输出训练图像的尺寸和标签的尺寸，都是torch格式的数据

print(b\_x.shape)

print(b\_y.shape)

class RNNimc(nn.Module):

def \_\_init\_\_(self, input\_dim, hidden\_dim, layer\_dim, output\_dim):

"""

input\_dim:输入数据的维度(图片每行的数据像素点)

hidden\_dim: RNN神经元个数

layer\_dim: RNN的层数

output\_dim:隐藏层输出的维度(分类的数量)

"""

super(RNNimc, self).\_\_init\_\_()

self.hidden\_dim = hidden\_dim ## RNN神经元个数

self.layer\_dim = layer\_dim ## RNN的层数

# RNN

self.rnn = nn.RNN(input\_dim, hidden\_dim, layer\_dim,

batch\_first=True, nonlinearity='relu')

# 连接全连阶层

self.fc1 = nn.Linear(hidden\_dim, output\_dim)

def forward(self, x):

# x:[batch, time\_step, input\_dim]

# 本例中time\_step＝图像所有像素数量／input\_dim

# out:[batch, time\_step, output\_size]

# h\_n:[layer\_dim, batch, hidden\_dim]

out, h\_n = self.rnn(x, None) # None表示h0会使用全0进行初始化

# 选取最后一个时间点的out输出

out = self.fc1(out[:, -1, :])

return out

# 模型的调用

input\_dim = 28 # 图片每行的像素数量

hidden\_dim = 128 # RNN神经元个数

layer\_dim = 1 # RNN的层数

output\_dim = 10 # 隐藏层输出的维度(10类图像)

MyRNNimc = RNNimc(input\_dim, hidden\_dim, layer\_dim, output\_dim)

print(MyRNNimc)

# RNNimc(

# (rnn): RNN(28, 128, batch\_first=True)

# (fc1): Linear(in\_features=128, out\_features=10, bias=True)

# )

# 可视化卷积神经网络

# 输入:[batch, time\_step, input\_dim]

hl\_graph = hl.build\_graph(MyRNNimc, torch.zeros([1, 28, 28]))

hl\_graph.theme = hl.graph.THEMES["blue"].copy()

print(hl\_graph)

# <hiddenlayer.graph.Graph object at 0x0000022805862C50>

# trace, out = torch.jit.get\_trace\_graph(model, args) 改为

# trace, out = torch.jit.\_get\_trace\_graph(model, args)

# Traceback (most recent call last):

# File "D:\pythoncode\learn\a\deep\_learning7.2.py", line 92, in <module>

# hl\_graph = hl.build\_graph(MyRNNimc, torch.zeros([1, 28, 28]))

# File "D:\anaconda3\envs\deeplearning\lib\site-packages\hiddenlayer\graph.py", line 136, in build\_graph

# import\_graph(g, model, args)

# File "D:\anaconda3\envs\deeplearning\lib\site-packages\hiddenlayer\pytorch\_builder.py", line 50, in import\_graph

# trace, out = torch.jit.get\_trace\_graph(model, args)

# AttributeError: module 'torch.jit' has no attribute 'get\_trace\_graph'

# pip install --upgrade hiddenlayer 版本不适配，更新包后解决

# Traceback (most recent call last):

# File "D:\pythoncode\learn\a\deep\_learning7.2.py", line 92, in <module>

# hl\_graph = hl.build\_graph(MyRNNimc, torch.zeros([1, 28, 28]))

# File "D:\anaconda3\envs\deeplearning\lib\site-packages\hiddenlayer\graph.py", line 136, in build\_graph

# import\_graph(g, model, args)

# File "D:\anaconda3\envs\deeplearning\lib\site-packages\hiddenlayer\pytorch\_builder.py", line 53, in import\_graph

# torch\_graph = trace.graph()

# AttributeError: 'torch.\_C.Graph' object has no attribute 'graph'

#

# 进程已结束,退出代码1

# 将可视化的网络保存为图片,默认格式为pdf

hl\_graph.save("deep Learning/MyRNNimc\_hl.png", format="png")

# 对模型进行训练

optimizer = torch.optim.RMSprop(MyRNNimc.parameters(), lr=0.0003)

criterion = nn.CrossEntropyLoss() # 损失函数

train\_loss\_all = []

train\_acc\_all = []

test\_loss\_all = []

test\_acc\_all = []

num\_epochs = 30

for epoch in range(num\_epochs):

print('Epoch {}/{}'.format(epoch, num\_epochs - 1))

MyRNNimc.train() # 设置模型为训练模式

corrects = 0

train\_num = 0

for step, (b\_x, b\_y) in enumerate(train\_loader):

# input :[batch, time\_step, input\_dim]

xdata = b\_x.view(-1, 28, 28)

output = MyRNNimc(xdata)

pre\_lab = torch.argmax(output, 1)

loss = criterion(output, b\_y)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

loss += loss.item() \* b\_x.size(0)

corrects += torch.sum(pre\_lab == b\_y.data)

train\_num += b\_x.size(0)

# 计算经过一个epoch的训练后在训练集上的损失和精度

train\_loss\_all.append(loss / train\_num)

train\_acc\_all.append(corrects.double().item() / train\_num)

print('{} Train Loss: {:.4f} Train Acc: {:.4f}'.format(

epoch, train\_loss\_all[-1], train\_acc\_all[-1]))

# 设置模型为验证模式

MyRNNimc.eval()

corrects = 0

test\_num = 0

for step, (b\_x, b\_y) in enumerate(test\_loader):

# input :[batch, time\_step, input\_dim]

xdata = b\_x.view(-1, 28, 28)

output = MyRNNimc(xdata)

pre\_lab = torch.argmax(output, 1)

loss = criterion(output, b\_y)

loss += loss.item() \* b\_x.size(0)

corrects += torch.sum(pre\_lab == b\_y.data)

test\_num += b\_x.size(0)

# 计算经过一个epoch的训练后在测试集上的损失和精度

test\_loss\_all.append(loss / test\_num)

test\_acc\_all.append(corrects.double().item() / test\_num)

print('{} Test Loss: {:.4f} Test Acc: {:.4f}'.format(

epoch, test\_loss\_all[-1], test\_acc\_all[-1]))

# 可视化模型训练过程中

plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)

plt.plot(train\_loss\_all, "ro-", label="Train loss")

plt.plot(test\_loss\_all, "bs-", label="Test loss")

plt.legend()

plt.xlabel("epoch")

plt.ylabel("Loss")

plt.subplot(1, 2, 2)

plt.plot(train\_acc\_all, "ro-", label="Train acc")

plt.plot(test\_acc\_all, "bs-", label="Test acc")

plt.xlabel("epoch")

plt.ylabel("acc")

plt.legend()

plt.show()

# Epoch 0/29

# 0 Train Loss: 0.0004 Train Acc: 0.5699

# 0 Test Loss: 0.0021 Test Acc: 0.6780

# Epoch 1/29

# 1 Train Loss: 0.0004 Train Acc: 0.7064

# 1 Test Loss: 0.0010 Test Acc: 0.7219

# Epoch 2/29

# 2 Train Loss: 0.0003 Train Acc: 0.7540

# 2 Test Loss: 0.0015 Test Acc: 0.7325

# Epoch 3/29

# 3 Train Loss: 0.0003 Train Acc: 0.7894

# 3 Test Loss: 0.0005 Test Acc: 0.7751

# Epoch 4/29

# 4 Train Loss: 0.0004 Train Acc: 0.8133

# 4 Test Loss: 0.0004 Test Acc: 0.7698

# Epoch 5/29

# 5 Train Loss: 0.0002 Train Acc: 0.8268

# 5 Test Loss: 0.0008 Test Acc: 0.8166

# Epoch 6/29

# 6 Train Loss: 0.0002 Train Acc: 0.8362

# 6 Test Loss: 0.0014 Test Acc: 0.8012

# Epoch 7/29

# 7 Train Loss: 0.0003 Train Acc: 0.8424

# 7 Test Loss: 0.0005 Test Acc: 0.8235

# Epoch 8/29

# 8 Train Loss: 0.0002 Train Acc: 0.8477

# 8 Test Loss: 0.0005 Test Acc: 0.8454

# Epoch 9/29

# 9 Train Loss: 0.0001 Train Acc: 0.8530

# 9 Test Loss: 0.0022 Test Acc: 0.8516

# Epoch 10/29

# 10 Train Loss: 0.0002 Train Acc: 0.8561

# 10 Test Loss: 0.0007 Test Acc: 0.8188

# Epoch 11/29

# 11 Train Loss: 0.0003 Train Acc: 0.8598

# 11 Test Loss: 0.0004 Test Acc: 0.8067

# Epoch 12/29

# 12 Train Loss: 0.0002 Train Acc: 0.8626

# 12 Test Loss: 0.0009 Test Acc: 0.8482

# Epoch 13/29

# 13 Train Loss: 0.0002 Train Acc: 0.8648

# 13 Test Loss: 0.0005 Test Acc: 0.8472

# Epoch 14/29

# 14 Train Loss: 0.0001 Train Acc: 0.8675

# 14 Test Loss: 0.0010 Test Acc: 0.8538

# Epoch 15/29

# 15 Train Loss: 0.0002 Train Acc: 0.8705

# 15 Test Loss: 0.0007 Test Acc: 0.8608

# Epoch 16/29

# 16 Train Loss: 0.0002 Train Acc: 0.8722

# 16 Test Loss: 0.0004 Test Acc: 0.8603

# Epoch 17/29

# 17 Train Loss: 0.0001 Train Acc: 0.8733

# 17 Test Loss: 0.0003 Test Acc: 0.8603

# Epoch 18/29

# 18 Train Loss: 0.0002 Train Acc: 0.8732

# 18 Test Loss: 0.0003 Test Acc: 0.8569

# Epoch 19/29

# 19 Train Loss: 0.0002 Train Acc: 0.8755

# 19 Test Loss: 0.0007 Test Acc: 0.8581

# Epoch 20/29

# 20 Train Loss: 0.0001 Train Acc: 0.8797

# 20 Test Loss: 0.0002 Test Acc: 0.8634

# Epoch 21/29

# 21 Train Loss: 0.0002 Train Acc: 0.8810

# 21 Test Loss: 0.0002 Test Acc: 0.8504

# Epoch 22/29

# 22 Train Loss: 0.0002 Train Acc: 0.8812

# 22 Test Loss: 0.0004 Test Acc: 0.8580

# Epoch 23/29

# 23 Train Loss: 0.0002 Train Acc: 0.8828

# 23 Test Loss: 0.0012 Test Acc: 0.8704

# Epoch 24/29

# 24 Train Loss: 0.0002 Train Acc: 0.8843

# 24 Test Loss: 0.0012 Test Acc: 0.8625

# Epoch 25/29

# 25 Train Loss: 0.0002 Train Acc: 0.8853

# 25 Test Loss: 0.0009 Test Acc: 0.8566

# Epoch 26/29

# 26 Train Loss: 0.0003 Train Acc: 0.8856

# 26 Test Loss: 0.0004 Test Acc: 0.8722

# Epoch 27/29

# 27 Train Loss: 0.0002 Train Acc: 0.8864

# 27 Test Loss: 0.0013 Test Acc: 0.8721

# Epoch 28/29

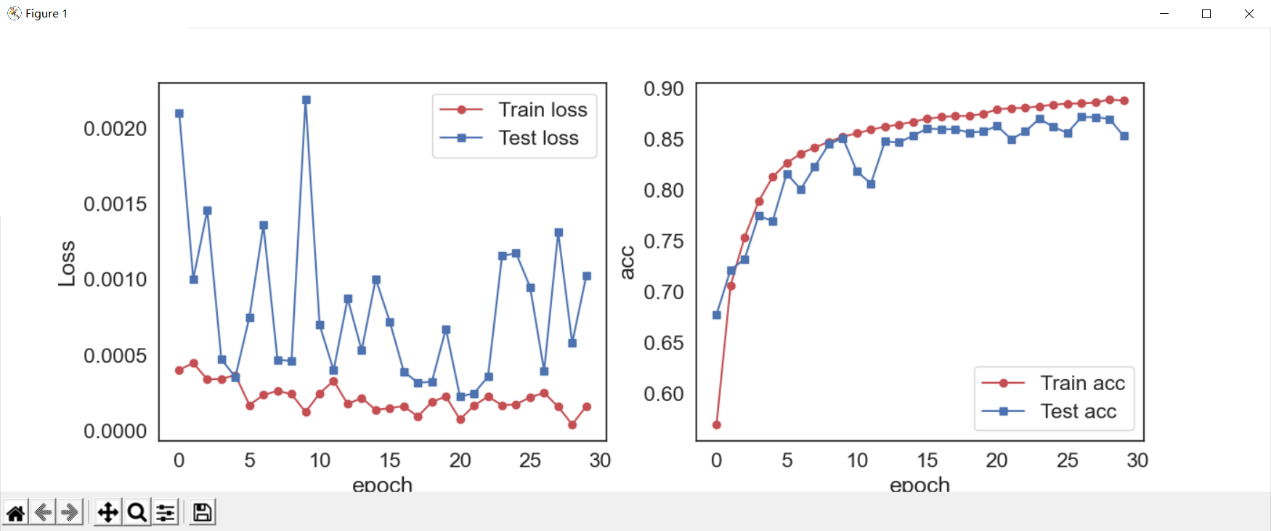
# 28 Train Loss: 0.0000 Train Acc: 0.8894

# 28 Test Loss: 0.0006 Test Acc: 0.8703

# Epoch 29/29

# 29 Train Loss: 0.0002 Train Acc: 0.8884

# 29 Test Loss: 0.0010 Test Acc: 0.8536



# 我最后Test Acc: 0.87 左右，课本上用的数据集是MNIST，Test Acc 是0.98附近 ，因为我用的是更复杂的FashionMNIST，所以准确率会低一些。