## 5.5 深度学习案例

一个完整的深度学习过程主要包括构建网络、定义训练函数、定义测试函数、训练和测试、结果展示几个过程。本节给出两个基于PyTorch的深度学习案例，来展示基于PyTorch的深度学习全过程。

### 5.5.1 函数近似

本例用一个深度神经网络来近似一元二次函数，待近似的一元二次函数为

|  |  |
| --- | --- |
|  | （5-50） |

使用全连接前馈式深度神经网络，各层节点数为输入层：1，第1隐藏：20，第2隐层：40，第3隐层20，输出层：1；损失函数使用均方误差损失（MSE）；优化器使用随机梯度下降（SGD），全部代码如下：

【代码5-1】深度神经网络逼近一元二次函数代码

# In[导入包]

import numpy as np

import torch

import torch.nn as nn

import matplotlib.pyplot as plt

# In[超参数]

LR = 1e-3

BATCH\_SIZE = 32

EPOCHS = 40

# In[原函数]

def fun(x):

return x\*x+3\*x+4

x = np.linspace(-np.pi,np.pi,100)

y = fun(x)

# In[创建神经网络]

class NeuNet(nn.Module):

def \_\_init\_\_(self,in\_size,out\_size):

nn.Module.\_\_init\_\_(self)

self.flatten = nn.Flatten()

self.layers = nn.Sequential(

nn.Linear(in\_size,20),

nn.ReLU(),

nn.Linear(20,40),

nn.ReLU(),

nn.Linear(40,20),

nn.ReLU(),

nn.Linear(20,out\_size),

)

def forward(self,x):

self.flatten(x)

return self.layers(x)

model = NeuNet(1,1)

# In[损失函数和优化器]

loss = torch.nn.MSELoss()

opt = torch.optim.SGD(model.parameters(),lr=LR)

# In[训练函数]

def train(model,loss,opt):

x\_batch = -np.pi+2\*np.pi\*np.random.rand(BATCH\_SIZE,1) # 训练输入

y\_tar\_batch = fun(x\_batch) # 目标输出

x\_batch = torch.from\_numpy(x\_batch).float() # 数据格式转换

y\_tar\_batch = torch.from\_numpy(y\_tar\_batch).float() # 数据格式转换

y\_pre\_batch = model(x\_batch).float() # 预测输入

loss\_fn = loss(y\_tar\_batch,y\_pre\_batch) # 损失函数

model.train() # 声明训练

opt.zero\_grad() # 梯度归零

loss\_fn.backward() # 误差反向传播

opt.step() # 参数调整

# In[测试函数]

def test(model):

model.eval()

with torch.no\_grad():

y\_pre\_test = model(torch.from\_numpy(x).float().unsqueeze(dim=1))

loss\_value = loss(torch.from\_numpy(y).float(),y\_pre\_test.float())

print('loss\_fn = ',loss\_value)

return loss\_value

# In[训练和测试]

Loss = []

for i in range(EPOCHS):

print('EPOCH {}--------------'.format(i))

train(model,loss,opt)

loss\_value = test(model)

Loss.append(loss\_value)

print('DONE')

# In[作图比较]

with torch.no\_grad():

y\_test = model(torch.from\_numpy(x).float().unsqueeze(dim=1))

y\_test = y\_test.squeeze().numpy()

plt.figure(1)

plt.plot(Loss)

plt.xlabel('EPOCHS')

plt.ylabel('Loss')

plt.title('Loss via EPOCHS')

plt.savefig('loss.jpg')

plt.figure(2)

plt.plot(x,y,label='real')

plt.plot(x,y\_test,label='approximated')

plt.xlabel('x')

plt.ylabel('y')

plt.title('Real vs approximated graph')

plt.legend()

plt.savefig('graph.jpg')

plt.show()

使用print(model)命令可以查看构建的神经网络结构如下：

NeuNet(

(flatten): Flatten(start\_dim=1, end\_dim=-1)

(layers): Sequential(

(0): Linear(in\_features=1, out\_features=20, bias=True)

(1): ReLU()

(2): Linear(in\_features=20, out\_features=40, bias=True)

(3): ReLU()

(4): Linear(in\_features=40, out\_features=20, bias=True)

(5): ReLU()

(6): Linear(in\_features=20, out\_features=1, bias=True)

)

)

程序运行结果如图5-16所示：

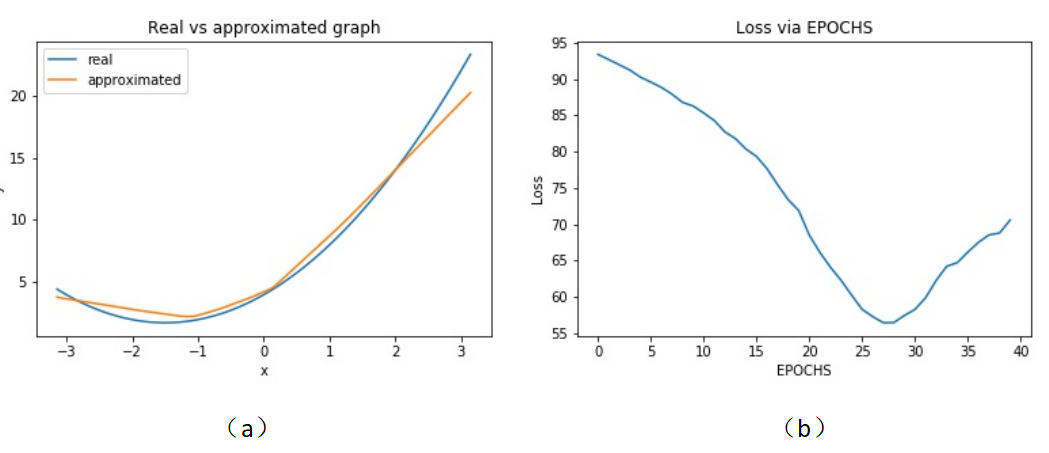


图 5-16 函数逼近运行结果

### 5.5.2 数字图片识别

本小节给出一个用深度神经网络识别FashionMNIST图片库的案例。深度神经网络共有4层，节点数分别为28\*28（这也是FashionMNIST图片的像素尺寸），512，512和10；因为是分类问题，损失函数使用交叉熵损失函数（CrossEntropyLoss），优化器使用随机梯度下降（SGD），全部代码如下：

【代码5-2】深度神经网络数字图片识别代码

# In[导入包]

import torch

import torch.nn as nn

from torch.utils.data import DataLoader

from torchvision import datasets

from torchvision.transforms import ToTensor

import matplotlib.pyplot as plt

# In[超参数]

BATCH\_SIZE = 64

LR = 1e-3

EPOCHS = 5

# In[数据下载]

training\_data = datasets.FashionMNIST(

root="data",

train=True,

download=True,

transform=ToTensor(),

)

test\_data = datasets.FashionMNIST(

root="data",

train=False,

download=True,

transform=ToTensor(),

)

# In[数据加载]

train\_dataloader = DataLoader(training\_data,batch\_size=BATCH\_SIZE)

test\_dataloader = DataLoader(test\_data,batch\_size=BATCH\_SIZE)

# In[创建网络]

device = 'cuda' if torch.cuda.is\_available() else 'cpu'

class NeuralNetwork(nn.Module):

def \_\_init\_\_(self):

nn.Module.\_\_init\_\_(self)

self.flatten = nn.Flatten()

self.linear\_relu\_stack=nn.Sequential(

nn.Linear(28\*28,512),

nn.ReLU(),

nn.Linear(512,512),

nn.ReLU(),

nn.Linear(512,10)

)

def forward(self,x):

x = self.flatten(x)

logits = self.linear\_relu\_stack(x)

return logits

model = NeuralNetwork().to(device)

# In[损失函数和优化器]

loss\_fn = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model.parameters(),lr=LR)

# In[训练函数]

def train(dataloader,model,loss\_fn,optimizer):

size = len(dataloader.dataset)

model.train()

for batch, (X,y) in enumerate(dataloader):

X,y = X.to(device),y.to(device)

pred = model(X)

loss = loss\_fn(pred,y)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if batch%100 ==0:

loss, current = loss.item(),batch\*len(X)

print(f"loss: {loss:>7f} [{current:>5d}/{size:>5d}]")

# In[测试函数]

def test(dataloader,model,loss\_fn):

size = len(dataloader.dataset)

num\_batches = len(dataloader)

model.eval()

test\_loss,correct = 0,0

with torch.no\_grad():

for X,y in dataloader:

X,y = X.to(device),y.to(device)

pred = model(X)

test\_loss += loss\_fn(pred,y).item()

correct += (pred.argmax(1)==y).type(torch.float).sum().item()

test\_loss /= num\_batches

correct /= size

print(f"Accuracy: {(100\*correct):>0.1f}%, Avg loss: {test\_loss:>8f}")

# In[模型训练和测试]

for t in range(EPOCHS):

print(f"Epoch {t+1}\n--------------------")

train(train\_dataloader,model,loss\_fn,optimizer)

test(test\_dataloader,model,loss\_fn)

print("Done!")

# In[训练结果展示]

classes = training\_data.classes

model.eval()

x,y = test\_data[0][0],test\_data[0][1]

with torch.no\_grad():

pred = model(x)

predicted,actual = classes[pred[0].argmax(0)],classes[y]

print(f'Predicted: "{predicted}", Actual: "{actual}"')

使用print(model)函数可以得到神经网络模型的拓扑结构如下：

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)

)

)

程序运行结果如下：

------Epoch 1------

loss: 2.308854 [ 0/60000]

loss: 2.285936 [ 6400/60000]

loss: 2.274783 [12800/60000]

loss: 2.276982 [19200/60000]

loss: 2.243695 [25600/60000]

loss: 2.230343 [32000/60000]

loss: 2.230508 [38400/60000]

loss: 2.203054 [44800/60000]

loss: 2.198021 [51200/60000]

loss: 2.176432 [57600/60000]

Accuracy: 46.0%, Avg loss: 2.161742

------Epoch 2------

Accuracy: 58.9%, Avg loss: 1.907152

------Epoch 3------

Accuracy: 61.6%, Avg loss: 1.540300

------Epoch 4------

Accuracy: 63.3%, Avg loss: 1.265945

------Epoch 5------

Accuracy: 64.5%, Avg loss: 1.097099

Done!

Predicted: "Ankle boot", Actual: "Ankle boot"