20220509-机器学习

1.过程描述

1.1机器学习

softmax回归

Python | 复制代码

```python
%matplotlib inline
import torch
import torchvision
from torch.utils import data
from torchvision import transforms
from d2l import torch as d2l
d2l.use_svg_display()

#读取数据集
trans=transforms.ToTensor()
mnist_train=torchvision.datasets.FashionMNIST(root="../data",train=True,transform=trans,download=True)
mnist_test=torchvision.datasets.FashionMNIST(root="../data",train=False,transform=trans,download=True)

def get_fashion_mnist_labels(labels):
    text_labels=['t-shirt', 'trouser', 'pullover', 'dress', 'coat','sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [text_labels[int(i)] for i in labels]

def show_images(imgs,num_rows,num_cols,titles=None,scale=1.5):
    figsize=(num_cols*scale,num_rows*scale)
    _,axes=d2l.plt.subplots(num_rows,num_cols,figsize=figsize)
    axes=axes.flatten()
    for i,(ax,img) in enumerate(zip(axes,imgs)):
        if torch.is_tensor(img):
            ax.imshow(img.numpy())
        else:
            ax.imshow(img)
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        if titles:
            ax.set_title(titles[i])
    return axes

X,y=next(iter(data.DataLoader(mnist_train,batch_size=18)))
show_images(X.reshape(18,28,28),2,9,titles=get_fashion_mnist_labels(y))

#读取小批量
batch_size=256
def get_dataloader_workers():
    return 4
train_iter=data.DataLoader(mnist_train,batch_size,shuffle=True,num_workers=get_dataloader_workers())

```

```
timer=d2l.Timer()
for X,y in train_iter:
    continue
f'{timer.stop():.2f} sec'

#整合所有组件
def load_data_fashion_mnist(batch_size,resize=None):
    trans=[transforms.ToTensor()]
    if resize:
        trans.insert(0,transforms.Resize(resize))
    trans=transforms.Compose(trans)

mnist_train=torchvision.datasets.FashionMNIST(root="../data",train=True,
transform=trans,download=True)

 mnist_test=torchvision.datasets.FashionMNIST(root="../data",train=False
,transform=trans,download=True)
    return (data.DataLoader(mnist_train,batch_size,shuffle=True,
                            num_workers=get_dataloader_workers()),
            data.DataLoader(mnist_test,batch_size,shuffle=False,
                           num_workers=get_dataloader_workers()))

train_iter,test_iter=load_data_fashion_mnist(32,resize=64)
for X,y in train_iter:
    print(X.shape,X.dtype,y.shape,y.dtype)
    break

import torch
from IPython import display
from d2l import torch as d2l

batch_size=256
train_iter,test_iter=d2l.load_data_fashion_mnist(batch_size)

#初始化模型参数
num_inputs=784
num_outputs=10
W=torch.normal(0,0.01,size=(num_inputs,num_outputs),requires_grad=True)
b=torch.zeros(num_outputs,requires_grad=True)

#定义softmax操作
def softmax(X):
    X_exp=torch.exp(X)
    partition=X_exp.sum(1,keepdims=True)
    return X_exp/partition

X=torch.normal(0,1,(2,5))
X_prob=softmax(X)
```

```
X_prob,X_prob.sum(1)

#定义模型
def net(X):
    return softmax(torch.matmul(X.reshape((-1,W.shape[0])),W)+b)

#定义损失函数
y=torch.tensor([0,2])
y_hat=torch.tensor([[0.1,0.3,0.6],[0.3,0.2,0.5]])
y_hat[[0,1],y]

def cross_entropy(y_hat,y):
    return -torch.log(y_hat[range(len(y_hat)),y])
cross_entropy(y_hat,y)

#分类的准确度
def accuracy(y_hat,y):
    if len(y_hat.shape)>1 and y_hat.shape[1]>1:
        y_hat=y_hat.argmax(axis=1)
    cmp=y_hat.type(y.dtype)==y
    return float (cmp.type(y.dtype).sum())

accuracy(y_hat,y)/len(y)

def evaluate_accuracy(net,data_iter):
    if isinstance(net,torch.nn.Module):
        net.eval()
    metric=Accumulator(2)
    with torch.no_grad():
        for X,y in data_iter:
            metric.add(accuracy(net(X),y),y.numel())
    return metric[0]/metric[1]

class Accumulator:
    def __init__(self,n):
        self.data=[0.0]*n
    def add(self,*args):
        self.data=[a+float(b) for a,b in zip(self.data,args)]
    def reset(self):
        self.data=[0.0]*len(self.data)
    def __getitem__(self,idx):
        return self.data[idx]

evaluate_accuracy(net,test_iter)

#训练
def train_epoch_ch3(net,train_iter,loss,updater):
    #将模型设置为训练模式
```

```
    if isinstance(net,torch.nn.Module):
        net.train()
    #训练损失总和、训练准确度综合、样本数
    metric=Accumulator(3)
    for X,y in train_iter:
        y_hat=net(X)
        l=loss(y_hat,y)
        if isinstance(updater,torch.optim.Optimizer):
            updater.zero_grad()
            l.mean().backward()
            updater.step()
        else:
            l.sum().backward()
            updater(X.shape[0])
        metric.add(float(l.sum()),accuracy(y_hat,y),y.numel())
    return metric[0]/metric[2],metric[1]/metric[2]

class Animator:
    def __init__(self, xlabel=None, ylabel=None, legend=None, xlim=None,
                 ylim=None, xscale='linear', yscale='linear',
                 fmts=('-', 'm--', 'g-.', 'r:'), nrows=1,
ncols=1,figsize=(3.5, 2.5)):
        if legend is None:
            legend = []
        d2l.use_svg_display()
        self.fig, self.axes = d2l.plt.subplots(nrows, ncols,
figsize=figsize)
        if nrows * ncols == 1:
            self.axes = [self.axes, ]
            # 使用lambda函数捕获参数
        self.config_axes = lambda: d2l.set_axes(
            self.axes[0], xlabel, ylabel, xlim, ylim, xscale, yscale,
legend)
        self.X, self.Y, self.fmts = None, None, fmts
    def add(self, x, y):
        # 向图表中添加多个数据点
        if not hasattr(y, "__len__"):
            y = [y]
        n = len(y)
        if not hasattr(x, "__len__"):
            x = [x] * n
        if not self.X:
            self.X = [[] for _ in range(n)]
        if not self.Y:
            self.Y = [[] for _ in range(n)]
        for i, (a, b) in enumerate(zip(x, y)):
            if a is not None and b is not None:
                self.X[i].append(a)
```

```
                    self.Y[i].append(b)
            self.axes[0].cla()
            for x, y, fmt in zip(self.X, self.Y, self.fmts):
                self.axes[0].plot(x, y, fmt)
            self.config_axes()
            display.display(self.fig)
            display.clear_output(wait=True)

    def train_ch3(net,train_iter,test_iter,loss,num_epochs,updater):
        animator=Animator(xlabel='epoch', xlim=[1, num_epochs], ylim=[0.3,
    0.9],
                          legend=['train loss', 'train acc', 'test acc'])
        for epoch in range(num_epochs):
            train_metrics = train_epoch_ch3(net, train_iter, loss, updater)
            test_acc = evaluate_accuracy(net, test_iter)
            animator.add(epoch + 1, train_metrics + (test_acc,))
        train_loss, train_acc = train_metrics
        assert train_loss < 0.5, train_loss
        assert train_acc <= 1 and train_acc > 0.7, train_acc
        assert test_acc <= 1 and test_acc > 0.7, test_acc

    lr=0.1
    def updater(batch_size):
        return d2l.sgd([W,b],lr,batch_size)

    num_epochs=10
    train_ch3(net,train_iter,test_iter,cross_entropy,num_epochs,updater)

    #预测
    def predict_ch3(net,test_iter,n=6):
        for X,y in test_iter:
            break
        trues=d2l.get_fashion_mnist_labels(y)
        preds=d2l.get_fashion_mnist_labels(net(X).argmax(axis=1))
        titles=[true+'\n'+pred for true,pred in zip(trues,preds)]
        d2l.show_images(X[0:n].reshape((n,28,28)),1,n,titles=titles[0:n])
    predict_ch3(net,test_iter)
```

2.结果输出

今天只把softmax回归大致看完了，其中比较疑惑的一个点是为什么损失函数要用交叉熵而非最小二次方差。后来看到一篇文章说多分类问题更关心的是输出层预测值的概率分布与真实值的分布的差异，而非概率之间的绝对差异，感觉有点道理。感觉整体而言算法思路还是比较清楚的，难点在于实现过程中各种矩阵以及向量的表示以及加速计算，看下来确实比较费劲。