2 天前, 2021/12/31

搭建折线图分类任务神经网络笔记

实现目标：

从零开始搭建神经网络实现分类任务！！

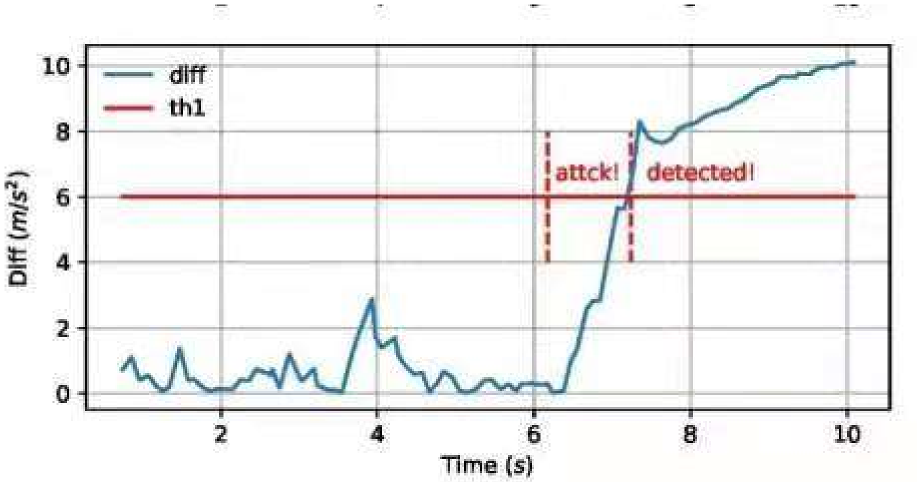


图 7 全加速攻击下估算加速度与参考值的比较

手动构造这种图片构建上图数据集（图片里不带坐标轴，纯折线），然后随我感觉划分成正常与异常，然后训练。

过程规划：

- 资源环境准备
- 知识学习
- 实践过程
- 效果展示
- 总结

准备（环境、网站）：

环境准备：

` pytorch_ 1.8.0_py3.6_cuda10.2_cudnn7.6.5_0 `

资源准备：

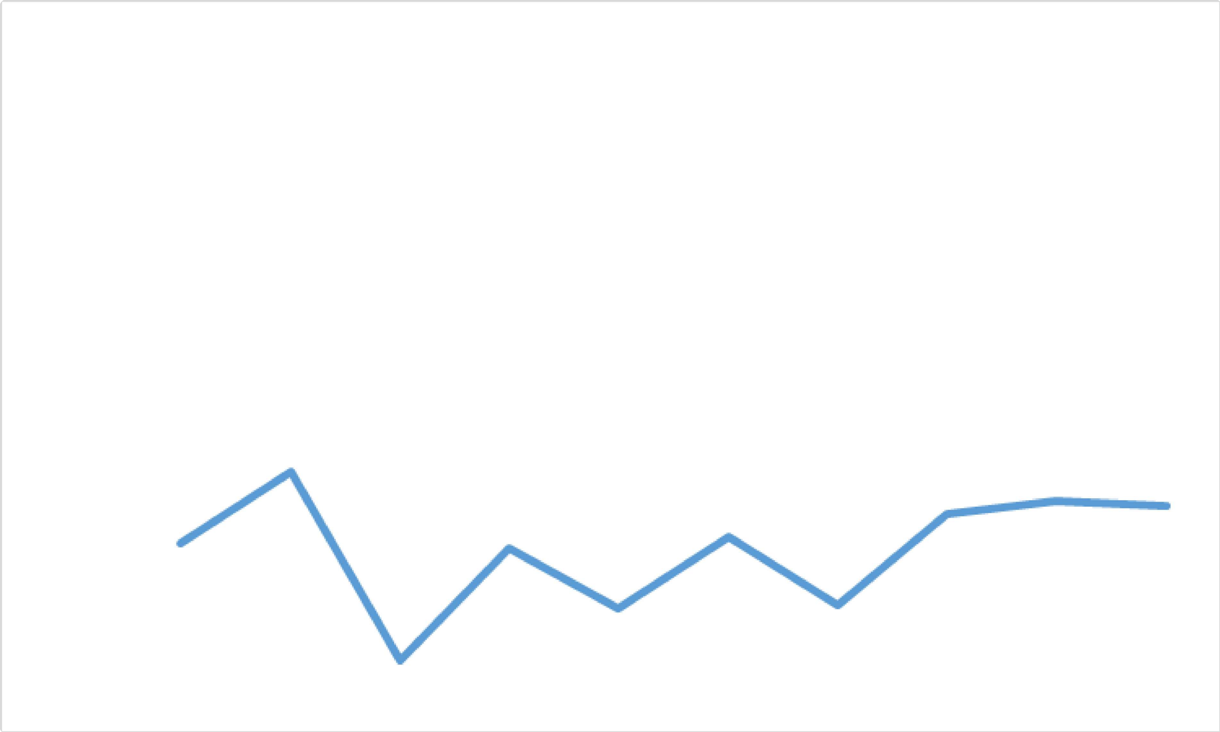
[cifar10数据集教程](https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutorial-py)
[Fashion_MNIST数据集教程](https://pytorch.org/tutorials/beginner/basics/data_tutorial.html)

[PyTorch官方文档](https://pytorch123.com/)

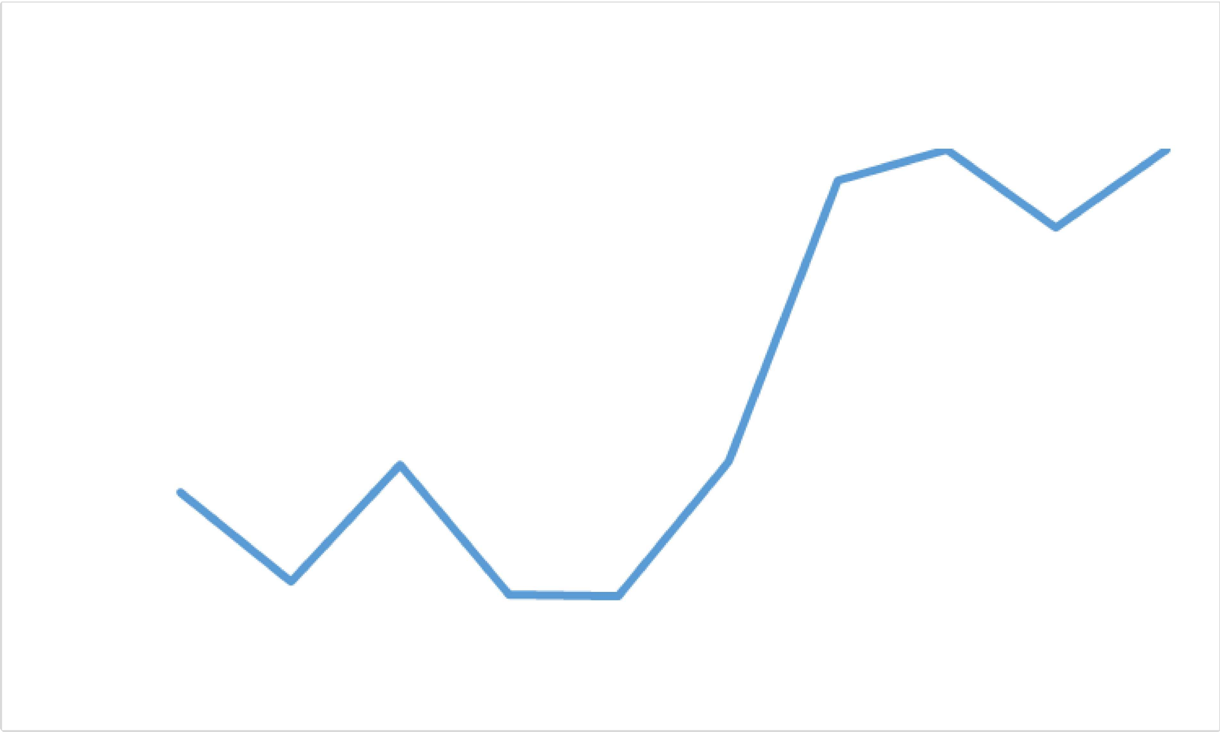
数据集准备：（存疑）

两个类型的折线图共100张：

normal：



abnormal：



实现阶段：

知识学习：

通过pytorch实现神经网络训练需要一下环节：

- 首先是数据集的处理，处理为你想要的数据结构；

在数据处理阶段，声明了MyDataset类，其目的是声明自建数据集供模型调用，主要有init参数、`__getitem__`和`__len__`三个不可获取的模块。其中`__getitem__`实现类函数对数据集的调用，`__len__`是声明数据集的规模，区别于loader的长度。
以下是代码具体实现：

```
# 数据处理class
class MyDataset(torch.utils.data.Dataset): #创建自己的类：MyDataset,这个类是继承的torch.utils.data.Dataset
    def __init__(self,root, datatxt, transform=None, target_transform=None): #初始化一些需要传入的参数
        fh = open(root + datatxt, 'r') #按照传入的路径和txt文本参数，打开这个文本，并读取内容
        imgs = []                    #创建一个名为img的空列表，一会儿用来装东西
        for line in fh:              #按行循环txt文本中的内容
            line = line.rstrip()       # 删除 本行string 字符串末尾的指定字符，这个方法的详细介绍自己查询python
            words = line.split()   #通过指定分隔符对字符串进行切片，默认为所有的空字符，包括空格、换行、制表符等
            imgs.append((words[0],int(words[1]))) #把txt里的内容读入imgs列表保存，具体是words几要看txt内容而定
                                        # 很显然，根据我刚才截图所示txt的内容，words[0]是图片信息，words[1]是lable
        self.imgs = imgs
        self.transform = transform
        self.target_transform = target_transform

    #用于按照索引读取每个元素的具体内容
    def __getitem__(self, index):
        fn, label = self.imgs[index] #fn是图片path #fn和label分别获得imgs[index]也即是刚才每行中word[0]和word[1]的信息
        img = Image.open(root+fn).convert('RGB') #按照path读入图片from PIL import Image # 按照路径读取图片

        if self.transform is not None:
            img = self.transform(img) #是否进行transform
        return img,label  #return很关键，return回哪些内容，那么我们在训练时循环读取每个batch时，就能获得哪些内容

    def __len__(self): #这个函数也必须要写，它返回的是数据集的长度，也就是多少张图片，要和loader的长度作区分
        return len(self.imgs)
```

- 通过torch搭建网络的`Dataset`和`DataLoader`；

首先是配置torchvision.transforms,对加载的数据进行预处理：

```
train_transform = transforms.Compose([  #打包各操作函数
    transforms.RandomRotation(10), #以指定的角度选装图片。（-10，10）旋转角度范围
    transforms.RandomHorizontalFlip(), # 随机水平翻转给定的PIL.Image,概率为0.5。即：一半的概率翻转，一半的概率不翻转。
    transforms.Resize(224),
    transforms.CenterCrop(224), #将给定的PIL.Image进行中心切割，得到给定的size，size可以是tuple，(target_height, target_width)。size也可以是一个Integer，在这种情况下，切出来的图片的形状是正方形。
    # transforms.Grayscale(), # 灰度话
    transforms.ToTensor(),
                                 # transforms.ToTensor()函数的作用是将原始的PILImage格式或者numpy.array格式的数据格式化为可被pytorch快速处理的张量类型。
                                 # 先由HWC转置为CHW格式；
                                 # 再转为float类型；
                                 # 最后，每个像素除以255,从0-255变换到0-1之间。
    transforms.Normalize(mean = [0.458,0.456,0.406],std = [0.229,0.224,0.225])
])
test_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    # transforms.Grayscale(), # 灰度话
    transforms.ToTensor(),
    transforms.Normalize(mean = [0.458,0.456,0.406],std = [0.229,0.224,0.225])
])


"""
 那transform.Normalize()是怎么工作的呢？以上面代码为例，ToTensor()能够把灰度范围从0-255变换到0-1之间，而后面的transform.Normalize()则把0-1变换到(-1,1).具体地说，对每个通道而言，Normalize执行以下操作：
    image=(image-mean)/std
 其中mean和std分别通过(0.5,0.5,0.5)和(0.5,0.5,0.5)进行指定。原来的0-1最小值0则变成(0-0.5)/0.5=-1，而最大值1则变成(1-0.5)/0.5=1.
"""
```

加载Dataset和DataLoader：

```
#根据自己定义的那个勒MyDataset来创建数据集！注意是数据集！而不是loader迭代器
train_data=MyDataset(root,'train_data.txt', transform=train_transform)
test_data=MyDataset(root,'test_data.txt', transform=test_transform)
```

```
trainloader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
                                          shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(test_data, batch_size=batch_size,
                                         shuffle=False, num_workers=2)
"""
torch.utils.data.DataLoader：数据加载器，自定义dataset封装在其中。组合数据集和采样器，并在数据集上提供单进程或多进程迭代器。
参数：

dataset (Dataset) – 加载数据的数据集。
batch_size (int, optional) – 每个batch加载多少个样本(默认: 1)。
shuffle (bool, optional) – 设置为True时会在每个epoch重新打乱数据(默认: False).
sampler (Sampler, optional) – 定义从数据集中提取样本的策略。如果指定，则忽略shuffle参数。
num_workers (int, optional) – 用多少个子进程加载数据。0表示数据将在主进程中加载(默认: 0)
collate_fn (callable, optional) –
pin_memory (bool, optional) –
drop_last (bool, optional) – 如果数据集大小不能被batch size整除，则设置为True后可删除最后一个不完整的batch。如果设为False并且数据集的大小不能被batch size整除，则最后一个batch将更小。(默认: False)
"""
```

- 构建网络；

声明超参数：

```
batch_size = 1
classes = ('0', '1')
```

构建卷及神经网络，包括两层卷基层三层全连接层，通过relu激活，最后通过softmax归一化输出结果：

```
class CNNModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3,6,3,1)
        self.conv2 = nn.Conv2d(6,16,3,1)
        self.fc1 = nn.Linear(54*54*16,120)
        self.fc2 = nn.Linear(120,84)
        self.fc3 = nn.Linear(84,2)
    def forward(self,X):
```

```
        X = F.relu(self.conv1(X))
        X = F.max_pool2d(X,2,2)
        X = F.relu(self.conv2(X))
        X = F.max_pool2d(X,2,2)
        X = X.view(-1,54*54*16)
        X = F.relu(self.fc1(X))
        X = F.relu(self.fc2(X))
        X = self.fc3(X)
        return F.log_softmax(X,dim = 1)
```

备选方案经典的LeNet网络：

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet,self).__init__()
        self.feature = nn.Sequential(
            nn.Conv2d(1,6,5),
            nn.Sigmoid(),
            nn.MaxPool2d(2,2),
            nn.Conv2d(6,16,5),
            nn.Sigmoid(),
            nn.MaxPool2d(2,2)
        )
        self.classifer = nn.Sequential(
            nn.Linear(4*4*16,120),
            nn.Sigmoid(),
            nn.Linear(120,84),
            nn.Sigmoid(),
            nn.Linear(84,10),
        )
    def forward(self,x):
        feature = self.feature(x)
        classifer = self.classifer(feature.view(feature.shape[0],-1))
        return classifer
```

- 选择Loss function和优化器；

设定交叉熵损失函数和Adam优化器：

```
```

```
criterion = nn.CrossEntropyLoss() #交叉熵损失函数
# optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9) #动量的SGD优化器
optimizer = torch.optim.Adam(net.parameters(),lr=0.001)
```

- 执行训练，打印结果并保存权重；

训练过程及保存权重结果：

```
import time
import math
start = time.time()
epochs = 10
train_losses = []
train_correct = []
test_losses = []
test_correct = []

for i in range(epochs):
    trn_corr = 0
    tst_corr = 0
    for b,(X_train,Y_train) in enumerate(trainloader):
        b+=1
        X_train = X_train.cuda()
        Y_train = Y_train.cuda()
        Y_pred = net(X_train)
        loss = criterion(Y_pred,Y_train)
        predictions = torch.max(Y_pred.data,1)[1]
        trn_corr+= (predictions == Y_train).sum()
        if b%125==0:
            print(f'Epoch {i+1} batch:{b} [{b*10}/{18750}] loss:{loss.item():.2f} accuracy:
{(trn_corr.item()*100)/(10*b):.2f}')
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    train_losses.append(loss)
    train_correct.append(trn_corr)
    with torch.no_grad():
        for(X_test,Y_test) in testloader:
            X_test = X_test.cuda()
```

```
            Y_test = Y_test.cuda()
            Y_val = net(X_test)
            predictions = torch.max(Y_val.data,1)[1]
            tst_corr+= (predictions == Y_test).sum()
        loss = criterion(Y_val,Y_test)
        test_losses.append(loss)
        test_correct.append(tst_corr)
end = time.time()
print(f'Train Duration : {(end-start)//60} minutes {math.ceil((end-start)%60)} seconds')



print('---------------------------------------------------------------------------------')
plt.plot(train_losses,label = 'Train Losses')
plt.plot(test_losses,label = 'Test Losses')
plt.legend()
print('---------------------------------------------------------------------------------')
plt.plot([i for i in range(1,11)],[100*i/18743 for i in train_correct],label = 'Training Accuracy')
plt.plot([i for i in range(1,11)],[100*i/6251 for i in test_correct],label = 'Testing Accuracy')
plt.legend()

PATH = './mode1.pth'
torch.save(net.state_dict(), PATH)
```

- 替换训练阶段的训练过程

```
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
    time.sleep(2)
    plt.close()
    # print ('it is ok')

#测试
```

```
# 读取测试样例
dataiter = iter(testloader)
images, labels = dataiter.next()
print('labels:',labels.shape)
print('img:',images)
# print images展示正确gt
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(1)))

# 加载网络权重
# net = Net()
net = CNNModel()
net.load_state_dict(torch.load('/home/cxking/datasets/fashion_data/pickle_cifar10_data/cifar_net.pth'))

outputs = net(images)

# 取最大概率的类别为结果
_, predicted = torch.max(outputs, 1)
print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                              for j in range(1)))

# 查看每个类别的准确率
# prepare to count predictions for each class
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

# again no gradients needed
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1
```

```
# print accuracy for each class
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print("Accuracy for class {:5s} is: {:.1f} %".format(classname,
                                                   accuracy))

```
```

- 效果展示

测试：

总准确率

```
```
output:
/home/cxking/datasets/line_data_new/train_data.txt
/home/cxking/datasets/line_data_new/test_data.txt
num_of_trainData: 84
num_of_testData: 21
labels: torch.Size([1])
img: tensor([[[[1.8188, 1.8188, 1.8188, ..., 1.8188, 1.8188, 1.8188],
          [2.2812, 2.2812, 2.2812, ..., 2.2812, 2.2812, 2.2812],
          [2.3668, 2.3668, 2.3668, ..., 2.3668, 2.3668, 2.3668],
          ...,
          [2.3668, 2.3668, 2.3668, ..., 2.3668, 2.3668, 2.3668],
          [2.2812, 2.2812, 2.2812, ..., 2.2812, 2.2812, 2.2812],
          [1.8188, 1.8188, 1.8188, ..., 1.8188, 1.8188, 1.8188]],

         [[1.8683, 1.8683, 1.8683, ..., 1.8683, 1.8683, 1.8683],
          [2.3410, 2.3410, 2.3410, ..., 2.3410, 2.3410, 2.3410],
          [2.4286, 2.4286, 2.4286, ..., 2.4286, 2.4286, 2.4286],
          ...,
          [2.4286, 2.4286, 2.4286, ..., 2.4286, 2.4286, 2.4286],
          [2.3410, 2.3410, 2.3410, ..., 2.3410, 2.3410, 2.3410],
          [1.8683, 1.8683, 1.8683, ..., 1.8683, 1.8683, 1.8683]],

         [[2.0823, 2.0823, 2.0823, ..., 2.0823, 2.0823, 2.0823],
          [2.5529, 2.5529, 2.5529, ..., 2.5529, 2.5529, 2.5529],
          [2.6400, 2.6400, 2.6400, ..., 2.6400, 2.6400, 2.6400],
          ...,
          [2.6400, 2.6400, 2.6400, ..., 2.6400, 2.6400, 2.6400],
```

```
         [2.5529, 2.5529, 2.5529,  ..., 2.5529, 2.5529, 2.5529],
         [2.0823, 2.0823, 2.0823,  ..., 2.0823, 2.0823, 2.0823]]]])
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
GroundTruth:      1
Predicted:      1
Accuracy of the network on the 100 test images: 85 %
```

分类准确率：

```
(torch180) cxking@cxking:~/project/Line_chart_recognition$ python test.py
/home/cxking/datasets/line_data_new/train_data.txt
/home/cxking/datasets/line_data_new/test_data.txt
num_of_trainData: 84
num_of_testData: 21
labels: torch.Size([1])
img: tensor([[[[1.8188, 1.8188, 1.8188,  ..., 1.8188, 1.8188, 1.8188],
          [2.2812, 2.2812, 2.2812,  ..., 2.2812, 2.2812, 2.2812],
          [2.3668, 2.3668, 2.3668,  ..., 2.3668, 2.3668, 2.3668],
          ...,
          [2.3668, 2.3668, 2.3668,  ..., 2.3668, 2.3668, 2.3668],
          [2.2812, 2.2812, 2.2812,  ..., 2.2812, 2.2812, 2.2812],
          [1.8188, 1.8188, 1.8188,  ..., 1.8188, 1.8188, 1.8188]],

         [[1.8683, 1.8683, 1.8683,  ..., 1.8683, 1.8683, 1.8683],
          [2.3410, 2.3410, 2.3410,  ..., 2.3410, 2.3410, 2.3410],
          [2.4286, 2.4286, 2.4286,  ..., 2.4286, 2.4286, 2.4286],
          ...,
          [2.4286, 2.4286, 2.4286,  ..., 2.4286, 2.4286, 2.4286],
          [2.3410, 2.3410, 2.3410,  ..., 2.3410, 2.3410, 2.3410],
          [1.8683, 1.8683, 1.8683,  ..., 1.8683, 1.8683, 1.8683]],

         [[2.0823, 2.0823, 2.0823,  ..., 2.0823, 2.0823, 2.0823],
          [2.5529, 2.5529, 2.5529,  ..., 2.5529, 2.5529, 2.5529],
          [2.6400, 2.6400, 2.6400,  ..., 2.6400, 2.6400, 2.6400],
          ...,
          [2.6400, 2.6400, 2.6400,  ..., 2.6400, 2.6400, 2.6400],
          [2.5529, 2.5529, 2.5529,  ..., 2.5529, 2.5529, 2.5529],
          [2.0823, 2.0823, 2.0823,  ..., 2.0823, 2.0823, 2.0823]]]])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
GroundTruth:      1
Predicted:      1
Accuracy for class 0      is: 40.0 %
Accuracy for class 1      is: 100.0 %
```

完成效果：

总结：

Python 编程学习 学术 +