作业要求：

1.补全网络代码，并运行手写数字识别项目。以出现最后的图片和预测结果为准。（65分）

2.保留原本的multilayer_perceptron网络定义（自己补全完的），自己定义一个卷积网络并运行成功。以出现最后的图片和预测结果为准（45分）

首先导入必要的包

numpy---------->python第三方库，用于进行科学计算

PIL------------> Python Image Library,python第三方图像处理库

matplotlib----->python的绘图库 pyplot:matplotlib的绘图框架

os------------->提供了丰富的方法来处理文件和目录

```
#导入需要的包
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import os
import paddle
print("基于Paddle的版本号为："+paddle.__version__)
```

```
基于Paddle的版本号为：2.0.0
```

Step1：准备数据。

(1)数据集介绍

MNIST数据集包含60000个训练集和10000测试数据集。分为图片和标签，图片是28*28的像素矩阵，标签为0~9共10个数字。

(2)transform函数是定义了一个归一化标准化的标准

(3)train_dataset和test_dataset

paddle.vision.datasets.MNIST()中的mode='train'和mode='test'分别用于获取mnist训练集和测试集

transform=transform参数则为归一化标准

```
#导入数据集Compose的作用是将用于数据集预处理的接口以列表的方式进行组合。
#导入数据集Normalize的作用是图像归一化处理，支持两种方式： 1. 用统一的均值和标准差值对图像的每个通道进行归一化处理；
#                                                          2. 对每个通道指定不同的均值和标准差值进行归一化处理。
from paddle.vision.transforms import Compose, Normalize
transform = Compose([Normalize(mean=[127.5],std=[127.5],data_format='CHW')])
# 使用transform对数据集做归一化
print('下载并加载训练数据')
train_dataset = paddle.vision.datasets.MNIST(mode='train', transform=transform)
test_dataset = paddle.vision.datasets.MNIST(mode='test', transform=transform)
print('加载完成')
```

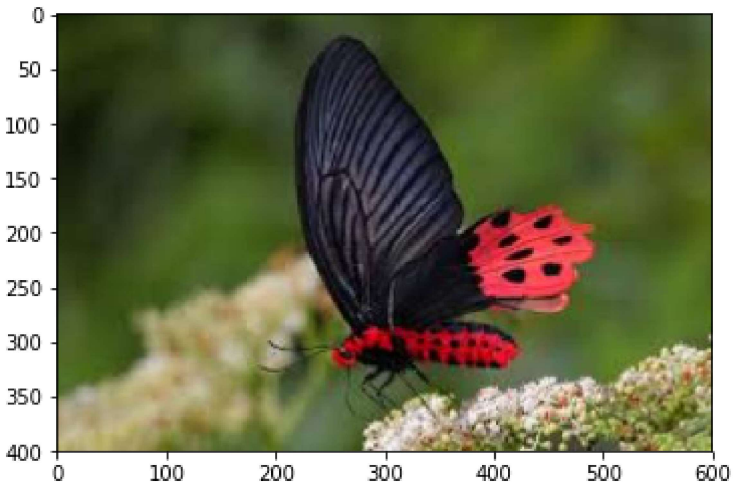
```
下载并加载训练数据
```

```
Cache file /home/aistudio/.cache/paddle/dataset/mnist/train-images-idx3-ubyte.gz not found, downloading htt
Begin to download

Download finished
Cache file /home/aistudio/.cache/paddle/dataset/mnist/train-labels-idx1-ubyte.gz not found, downloading htt
Begin to download
........
Download finished
Cache file /home/aistudio/.cache/paddle/dataset/mnist/t10k-images-idx3-ubyte.gz not found, downloading http
Begin to download

Download finished
Cache file /home/aistudio/.cache/paddle/dataset/mnist/t10k-labels-idx1-ubyte.gz not found, downloading http
Begin to download
..
Download finished


加载完成
```

```
?plt.figure
```

```
#让我们一起看看数据集中的图片是什么样子的
train_data0, train_label_0 = train_dataset[2][0],train_dataset[2][1]
train_data0 = train_data0.reshape([28,28])
plt.figure(figsize=(1,1))
print(plt.imshow(train_data0, cmap=plt.cm.binary))
print('train_data0 的标签为: ' + str(train_label_0))
```

```
AxesImage(9,9;55.8x54.36)
train_data0 的标签为: [4]
```




```
#让我们再来看看数据样子是什么样的吧
print(train_data0)
```

```
[[-1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.        ]
 [-1.         -1.         -1.         -1.         -1.         -1.
```

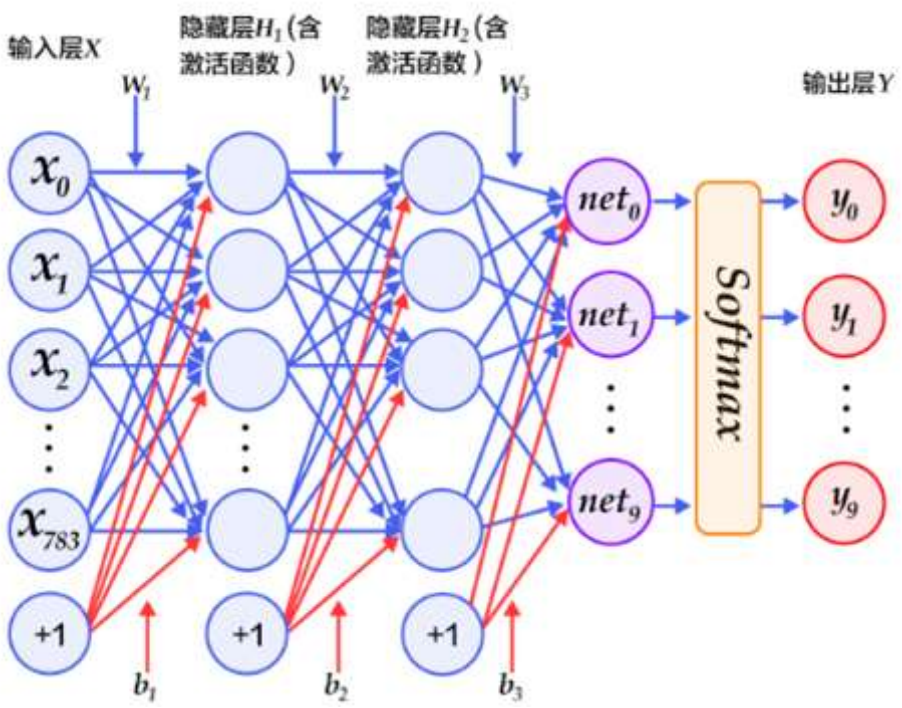
```
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -0.9764706  -0.85882354 -0.85882354 -0.85882354 -0.01176471  0.06666667
   0.37254903 -0.79607844  0.3019608   1.           0.9372549  -0.00392157
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -0.7647059  -0.7176471  -0.2627451   0.20784314
   0.33333334  0.9843137   0.9843137   0.9843137   0.9843137   0.9843137
   0.7647059   0.34901962  0.9843137   0.8980392   0.5294118  -0.49803922
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -0.6156863   0.8666667   0.9843137   0.9843137   0.9843137
   0.9843137   0.9843137   0.9843137   0.9843137   0.9843137   0.96862745
  -0.27058825 -0.35686275 -0.35686275 -0.56078434 -0.69411767 -1.
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -0.85882354  0.7176471   0.9843137   0.9843137   0.9843137
   0.9843137   0.9843137   0.5529412   0.42745098  0.9372549   0.8901961
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -0.37254903  0.22352941 -0.16078432  0.9843137
   0.9843137   0.60784316 -0.9137255  -1.          -0.6627451   0.20784314
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -0.8901961  -0.99215686  0.20784314
   0.9843137  -0.29411766 -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.           0.09019608
   0.9843137   0.49019608 -0.9843137  -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -0.9137255
   0.49019608  0.9843137  -0.4509804  -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.        ]
 [-1.          -1.          -1.          -1.          -1.          -1.
  -1.          -1.          -1.          -1.          -1.          -1.
  -0.7254902   0.8901961   0.7647059   0.25490198 -0.15294118 -0.99215686
```

```
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -0.3647059   0.88235295  0.9843137   0.9843137  -0.06666667
 -0.8039216  -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -0.64705884  0.45882353  0.9843137   0.9843137
  0.1764706  -0.7882353  -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -0.8745098  -0.27058825  0.9764706
  0.9843137   0.46666667 -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.          0.9529412
  0.9843137   0.9529412  -0.49803922 -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -0.6392157   0.01960784  0.43529412  0.9843137
  0.9843137   0.62352943 -0.9843137  -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -0.69411767  0.16078432  0.79607844  0.9843137   0.9843137   0.9843137
  0.9607843   0.42745098 -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -0.8117647  -0.10588235
  0.73333335  0.9843137   0.9843137   0.9843137   0.9843137   0.5764706
 -0.3882353  -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -0.81960785 -0.48235294  0.67058825  0.9843137
  0.9843137   0.9843137   0.9843137   0.5529412  -0.3647059  -0.9843137
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -1.         -1.
 -0.85882354  0.34117648  0.7176471   0.9843137   0.9843137   0.9843137
  0.9843137   0.5294118  -0.37254903 -0.92941177 -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -0.5686275   0.34901962
  0.77254903  0.9843137   0.9843137   0.9843137   0.9843137   0.9137255
  0.04313726 -0.9137255  -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.          0.06666667  0.9843137
  0.9843137   0.9843137   0.6627451   0.05882353  0.03529412 -0.8745098
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
[-1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.         -1.         -1.
 -1.         -1.         -1.         -1.        ]
```

```
 [-1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.        ]
 [-1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.         -1.         -1.        ]]
```

Step2.网络配置

以下的代码判断就是定义一个简单的多层感知器，一共有三层，两个大小为100的隐层和一个大小为10的输出层，因为MNIST数据集是手写0到9的灰度图像，类别有10个，所以最后的输出大小是10。最后输出层的激活函数是Softmax，所以最后的输出层相当于一个分类器。加上一个输入层的话，多层感知器的结构是：输入层-->>隐层-->>隐层-->>输出层。



请补全网络代码

```
# 定义多层感知器
#动态图定义多层感知器
class multilayer_perceptron(paddle.nn.Layer):
    def __init__(self):#网络
        super(multilayer_perceptron,self).__init__()
        self.flatten = paddle.nn.Flatten()
        self.linear_1 = paddle.nn.Linear(in_features=784,out_features=500)
        self.linear_2 = paddle.nn.Linear(in_features=500,out_features=10)
        self.relu = paddle.nn.ReLU()
        self.dropout = paddle.nn.Dropout(0.2)

    def forward(self, x):#传递
        y = self.flatten(x)
        y = self.linear_1(y)
        y = self.relu(y)
        y = self.dropout(y)
        y = self.linear_2(y)
        return y
```

```
#请在这里定义卷积网络的代码
#注意：定义完成卷积的代码后，后面的代码是需要修改的！
```

```
LeNet= multilayer_perceptron()
from paddle.metric import Accuracy

# 用Model封装模型
model = paddle.Model(LeNet)

# 定义损失函数
optim = paddle.optimizer.Adam(learning_rate=0.001, parameters=model.parameters())

# 配置模型
model.prepare(optim,paddle.nn.CrossEntropyLoss(),Accuracy())

# 训练保存并验证模型
model.fit(train_dataset,test_dataset,epochs=2,batch_size=64,save_dir='multilayer_perceptron',verbose=1)
```

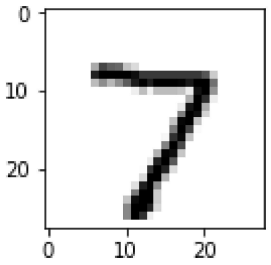
```
The loss value printed in the log is the current step, and the metric is the average value of previous step
Epoch 1/2
step 938/938 [==============================] - loss: 0.2053 - acc: 0.9019 - 8ms/step
save checkpoint at /home/aistudio/multilayer_perceptron/0
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous ste
step 157/157 [==============================] - loss: 0.0531 - acc: 0.9533 - 7ms/step
Eval samples: 10000
Epoch 2/2
step 938/938 [==============================] - loss: 0.0988 - acc: 0.9509 - 8ms/step
save checkpoint at /home/aistudio/multilayer_perceptron/1
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous ste
step 157/157 [==============================] - loss: 0.0087 - acc: 0.9595 - 7ms/step
Eval samples: 10000
save checkpoint at /home/aistudio/multilayer_perceptron/final
```

```
# 训练保存并验证模型
model.fit(train_dataset,test_dataset,epochs=2,batch_size=64,save_dir='multilayer_perceptron',verbose=1)
```

```
#获取测试集的第一个图片
test_data0, test_label_0 = test_dataset[0][0],test_dataset[0][1]
test_data0 = test_data0.reshape([28,28])
plt.figure(figsize=(2,2))
#展示测试集中的第一个图片
print(plt.imshow(test_data0, cmap=plt.cm.binary))
print('test_data0 的标签为: ' + str(test_label_0))
#模型预测
result = model.predict(test_dataset, batch_size=1)
#打印模型预测的结果
print('test_data0 预测的数值为：%d' % np.argsort(result[0][0])[0][-1])
```

```
AxesImage(18,18;111.6x108.72)
test_data0 的标签为: [7]
Predict begin...
step 10000/10000 [==============================] - 2ms/step
Predict samples: 10000
test_data0 预测的数值为：7


/opt/conda/envs/python35-paddle120-env/lib/python3.7/site-packages/numpy/lib/type_check.py:546: Deprecation
```

```
  'a.item() instead', DeprecationWarning, stacklevel=1)
```




```
?np.argsort
```

```
#获取测试集的任一图片
import random
for i in range(5):
    a=int(random.random()*10)
    print(a)

    test_data0, test_label_0 = test_dataset[a][0],test_dataset[a][1]
    test_data0 = test_data0.reshape([28,28])
    plt.figure(figsize=(2,2))
    #展示这张图片
    print(plt.imshow(test_data0, cmap=plt.cm.binary))
    print('test_data0 的标签为: ' + str(test_label_0))
    #模型预测
    result = model.predict(test_dataset, batch_size=1)
    #打印模型预测的结果
    #print(np.argsort(result))
    print('test_data0 预测的数值为: %d' % np.argsort(result[0][a])[0][-1])
```

```
7
AxesImage(18,18;111.6x108.72)
test_data0 的标签为: [9]
Predict begin...
step 10000/10000 [==============================] - 2ms/step
Predict samples: 10000
test_data0 预测的数值为: 9
6
AxesImage(18,18;111.6x108.72)
test_data0 的标签为: [4]
Predict begin...
step 10000/10000 [==============================] - 2ms/step
Predict samples: 10000
test_data0 预测的数值为: 4
4
AxesImage(18,18;111.6x108.72)
test_data0 的标签为: [4]
Predict begin...
step 10000/10000 [==============================] - 2ms/step
Predict samples: 10000
test_data0 预测的数值为: 4
8
AxesImage(18,18;111.6x108.72)
test_data0 的标签为: [5]
Predict begin...
step 10000/10000 [==============================] - 2ms/step
Predict samples: 10000
test_data0 预测的数值为: 6
0
```

```
AxesImage(18,18;111.6x108.72)
test_data0 的标签为: [7]
Predict begin...
step 10000/10000 [==============================] - 1ms/step
Predict samples: 10000
test_data0 预测的数值为: 7
```