**2024-2025学年第二学期计算机22A1-A3班人工智能模块**

**《机器学习》期末大作业**

**班级： 计算机22A4**

**学号： 2022015232**

**姓名： 孙春辉**

**2025年3月**

# BACK PROGRESSION NUTERAL NETWORK算法

## 基本算法原理

BP 神经网络分为正向传播和反向传播两个阶段，在正向传播阶段，输入样本从输入层开始，经过隐藏层加权和激活函数处理之后，信号正向传播到输出层。若输出层与实际输出结果不一致，则根据计算出的误差开始进行反向传播。并且在反向传播阶段更新学习参数权值，重复上述过程将参数进行优化以达到最好预测方式中。

可以将输入的各种类型的输入值转化成为数值、特征值，并且将其进行标签化，如 one-hot 标签等，用于激励函数的输出值比对，然后将输入的特征值进行多层 BP 神经网络得到其输出值，并且暂留过程中的隐藏层的参数，用于下一次的使用。在经过各个隐藏层之后输出的最终结果和训练集合的实际结果one-hot标签进行比对，用均方误差L\_2范数来求其最小误差，并且根据这样的，当我们在对其求误差的时候，可以将其上一层的权重作为未知数，并且对其均方误差函数进行求导，之后我们会得到一个常数值，并且通过选择的激励函数进行数值的映射，将其映射到我们所需要的范围，并且，将最终的梯度值，按照我们所规定的梯度学习参数进行最后的调整，将其应用到上一层的学习参数之中，完成根据结果的迭代，并且依次向前传递，直到传递到最后。

经过多次迭代我们将获得一个多层或者单层的学习参数矩阵W和偏置矩阵B，这就是模型训练的最终结果，而后我们可以根据已经训练好的模型用来预测我们想要预测的非线性的问题的概率解。最后的输出结果会是一个概率矩阵，最终我们通过找到其中的最大的概率值的索引，并且通过最开始的one-hot编码或者其他编码的反向映射，知道其预测结果。

## 算法模型选择条件

起初，在看到这个模型可以有反馈，本以为该模型可以解决所有的非线性问题，便粗暴的认为其可以解决所有问题，但是起始并非如此，因为它只能够存在所有问题有解，并且当问题的解是可以通过数值映射或者可以使用一些方法来将结果变换的问题才有可能去实现最终结果的分类，意思就是需要先进行分类才可以解决，但是如果有问题其无法进行分类或者其结果无法进行映射则不可能存在可解的可能性，此外如果问题的解是不可举例或者不可以在最之前得到确切的答案的情况下也是不可以进行训练的因为没有可以反馈的基准值，在反向传播的情况下无法计算其误差最小化，所以就会存在无解的情况。

问题选择的情况下，在于输入数据的特征值是可以转化成为数值的情况，因为前向传播是以方程及导数的计算作为传播途径的，所以如果问题的输入是以文本含义或者信息的方式进行输入，那么值的解就是不可能进行计算的所以不能选择该种问题，并且也无法选择。

其次最终所要预测的方面，其结果也一定要是可以映射到数值层面，并且其最终结果是一种可以通过规律的概率进行计算的，也就是一定程度的有序，可以不是一种线性图表的有序，但是一定是有分类的情况下才可以解决该种问题，并且进行反馈。

激励函数的选择，我们一定需要将其根据我们所需要的编码形式进行选择，我们的输出值一定要存在于该种激励函数的输出值中，并且其输出值的范围是可以决定的。

## 算法模型调参 模型参数

在BP神经网络中处理数据的输入部分一般存在两种参数，一个是梯度，一个是迭代次数。

**梯度部分：**

**梯度消失：**梯度趋近于零，网络权重无法更新或更新的很微小，网络训练再久也不会有效果；

**梯度爆炸：**梯度呈指数级增长，变的非常大，然后导致网络权重的大幅更新，使网络变得不稳定。

梯度剪切这个方案主要是针对梯度爆炸提出的，其思想是设置一个梯度剪切阈值，然后更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。这可以防止梯度爆炸；权重正则化也是解决度爆炸的手段，比较常见的是L1正则，和L2正则，正则化是通过对网络权重做正则，防止w过大，限制过拟合。

**迭代次数：**

由于其模型算法依赖于模型数据，以及迭代顺序，有时候可能会根据很多同一结果值的数据模型输入导致最终的学习参数W和B矩阵都导致了最终的结果直接会偏向其特定某一值，所以我们根据其特性将梯度参数降低并且在选取输入数据的方式上，更加均衡的去选择这样的数据输入，根据本算法模型本身的前向传播和反向传播的反馈机制实现其最最终的准确率较为高 的结果。

# BP神经网络 解决识别手写数字问题

## 问题需求分析

**数据处理部分：**

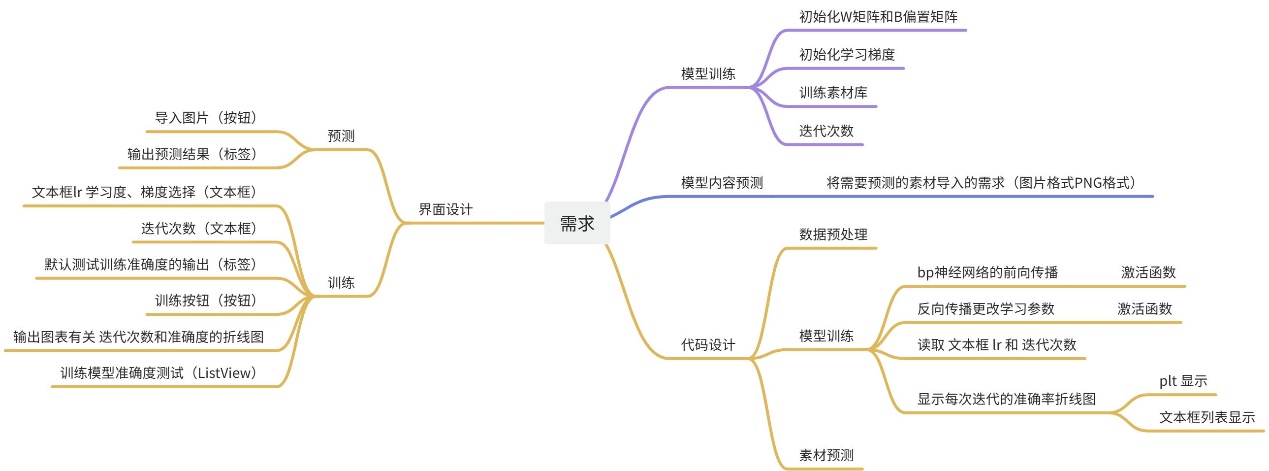
1. **图片是更改为数值类型**
2. **多通道数值处理为单通道数值**

**算法实现部分：**

1. **学习参数获取**
2. **迭代次数获取**
3. **训练模型**
4. **预测结果**

**结果展示：**

1. **图片显示**
2. **结果预测显示**



# 问题解决

## 数据处理

* + 1. **文件路径获取（文件）：**

def open\_png(self) :  
 # getOpenFileNames 和 getOpenFileName的区别 一个是多个一个是单个文件  
 filename, \_ = QFileDialog.getOpenFileName(  
 self,  
 "选择文件",# title  
 "", # starting dir,  
 "图片文件 (\*.png \*.jpg \*.jpeg \*.bmp \*.gif)"  
 )

通过系统自带的打开文件的方式进行文件路径的获取用于之后的文件打开和预览。

* + 1. **图片文件转换为数据数组**

def img\_to\_arr(self,filename) : # 将 需要预测的图片转化为 数组形式  
 arr = []  
 fh = open(filename)  
 im = Image.open(filename)  
 for i in range(0,32) :  
 for j in range(0, 32):  
 pix = im.getpixel((i,j)) # RGB 三通道 颜色值  
 pixs = pix[0] + pix[1] + pix[2] # 转化为单通道颜色值  
 if pixs == 0 :  
 arr.append(1)  
 else :  
 arr.append(0)  
 print(self.predict(arr,self.W1,self.W2))  
 return np.argmax(self.predict(arr,self.W1,self.W2))

用作于特征值并且将其用于模型训练和数据预测的数据输入

* + 1. 训练模型的数据处理

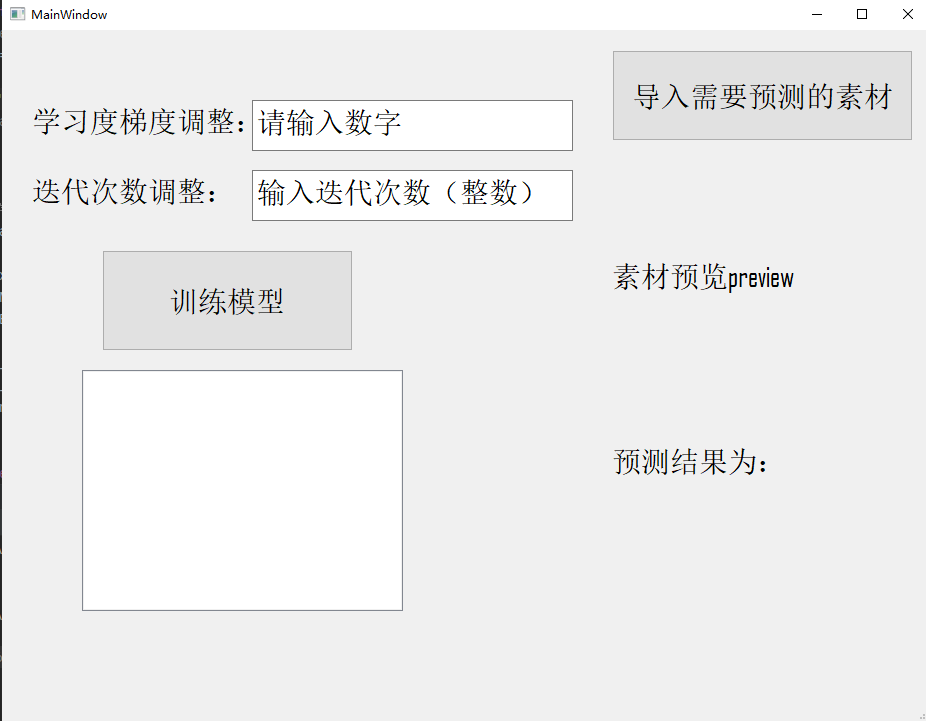
def img\_to\_txt(self,png\_path, txt\_path):  
 png\_list = listdir(png\_path + "/" + 'png')  
 for f in png\_list:  
 fname = f.split(".")[0]  
 im = Image.open(png\_path + "/" + 'png' + "/" + f)  
 fh = open(txt\_path + "/" + fname + ".txt", "w")  
 for m in range(0, 32):  
 for n in range(0, 32):  
 pix = im.getpixel((n, m))  
 pixs = pix[0] + pix[1] + pix[2]  
 if pixs == 0:  
 fh.write("1")  
 else:  
 fh.write("0")  
 fh.write("\n")  
 fh.close()  
  
def get\_split\_data(self):  
 txt\_list = listdir(self.txt\_path)  
  
 for txt in txt\_list:  
 try:  
 shutil.move(self.txt\_path + "/" + txt, self.test\_path)  
 except:  
 pass

## 界面设计

* + 1. **界面设计类：**

# -\*- coding: utf-8 -\*-  
  
# Form implementation generated from reading ui file 'sch.ui'  
#  
# Created by: PyQt5 UI code generator 5.15.9  
#  
# WARNING: Any manual changes made to this file will be lost when pyuic5 is  
# run again. Do not edit this file unless you know what you are doing.  
  
from PyQt5.QtCore import QStringListModel # Add this import  
from PyQt5 import QtCore, QtGui, QtWidgets  
  
  
class Ui\_MainWindow(object):  
 def setupUi(self, MainWindow):  
 MainWindow.setObjectName("MainWindow")  
 MainWindow.resize(929, 691)  
 font = QtGui.QFont()  
 font.setFamily("Agency FB")  
 font.setPointSize(22)  
 MainWindow.setFont(font)  
 self.centralwidget = QtWidgets.QWidget(MainWindow)  
 self.centralwidget.setObjectName("centralwidget")  
 self.predict\_Button = QtWidgets.QPushButton(self.centralwidget)  
 self.predict\_Button.setGeometry(QtCore.QRect(610, 20, 301, 91))  
 self.predict\_Button.setObjectName("predict\_Button")  
 self.preview\_picture = QtWidgets.QLabel(self.centralwidget)  
 self.preview\_picture.setGeometry(QtCore.QRect(610, 120, 301, 251))  
 self.preview\_picture.setObjectName("preview\_picture")  
 self.output\_num = QtWidgets.QLabel(self.centralwidget)  
 self.output\_num.setGeometry(QtCore.QRect(610, 370, 301, 121))  
 self.output\_num.setObjectName("output\_num")  
 self.label\_3 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_3.setGeometry(QtCore.QRect(30, 50, 221, 81))  
 self.label\_3.setObjectName("label\_3")  
 self.label\_4 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_4.setGeometry(QtCore.QRect(30, 120, 221, 81))  
 self.label\_4.setObjectName("label\_4")  
 self.Train\_Button = QtWidgets.QPushButton(self.centralwidget)  
 self.Train\_Button.setGeometry(QtCore.QRect(100, 220, 251, 101))  
 self.Train\_Button.setObjectName("Train\_Button")  
  
 self.listView = QtWidgets.QListView(self.centralwidget)  
 self.listView.setGeometry(QtCore.QRect(80, 340, 321, 241))  
 self.listView.setObjectName("listView")  
 self.log\_model = QStringListModel()  
 self.listView.setModel(self.log\_model)  
  
 self.iter\_num = QtWidgets.QTextEdit(self.centralwidget)  
 self.iter\_num.setGeometry(QtCore.QRect(250, 140, 321, 51))  
 self.iter\_num.setObjectName("iter\_num")  
 self.lr\_num = QtWidgets.QTextEdit(self.centralwidget)  
 self.lr\_num.setGeometry(QtCore.QRect(250, 70, 321, 51))  
 self.lr\_num.setObjectName("lr\_num")  
 MainWindow.setCentralWidget(self.centralwidget)  
 self.statusbar = QtWidgets.QStatusBar(MainWindow)  
 self.statusbar.setObjectName("statusbar")  
 MainWindow.setStatusBar(self.statusbar)  
  
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
 def retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))  
 self.predict\_Button.setText(\_translate("MainWindow", "导入需要预测的素材"))  
 self.preview\_picture.setText(\_translate("MainWindow", "素材预览preview"))  
 self.output\_num.setText(\_translate("MainWindow", "预测结果为："))  
 self.label\_3.setText(\_translate("MainWindow", "学习度梯度调整："))  
 self.label\_4.setText(\_translate("MainWindow", "迭代次数调整："))  
 self.Train\_Button.setText(\_translate("MainWindow", "训练模型"))  
 self.iter\_num.setHtml(\_translate("MainWindow", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"  
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"  
"p, li { white-space: pre-wrap; }\n"  
"</style></head><body style=\" font-family:\'Agency FB\'; font-size:22pt; font-weight:400; font-style:normal;\">\n"  
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">输入迭代次数（整数）</p></body></html>"))  
 self.lr\_num.setHtml(\_translate("MainWindow", "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/strict.dtd\">\n"  
"<html><head><meta name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"  
"p, li { white-space: pre-wrap; }\n"  
"</style></head><body style=\" font-family:\'Agency FB\'; font-size:22pt; font-weight:400; font-style:normal;\">\n"  
"<p style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px;\">请输入数字</p></body></html>"))

* + 1. **界面展示**：



* + 1. **界面输入及运行结果：**





可以通过上面两种结果对于同一输入数据，出现了不同的预测结果我们可以了解到学习度梯度和迭代次数对于模型训练以及，最后的数据的预测是有很大的关系的。

## 算法设计

def predict(self,X\_data, W1, W2):  
 L1 = self.sigmoid(np.dot(X\_data, W1))  
 L2 = self.sigmoid(np.dot(L1, W2))  
 return L2  
  
def sigmoid(self,x):  
 return 1 / (1 + np.exp(-x))  
  
def dsigmoid(self,x):  
 s = 1 / (1 + np.exp(-x))  
 return s \* (1 - s)  
  
def train(self,X\_train, y\_train, X\_test, y\_test) :  
  
 # 定义神经网络由三层组成，每一层的神经元个数分别为1024、50、10，其中，1024像素即1024通道输入，隐藏层神经元个数为50，输出层为0~9，所以是10个  
 # 输入层到隐藏层的权值为W1，隐藏层到输出层的权值为W2  
 iter = self.iter  
 lr = self.lr  
 W1 = np.random.random((1024, 50)) - 0.5  
 W2 = np.random.random((50, 10)) - 0.5  
 accx = []  
 accy = []  
 for it in range(iter):  
 # 随机选取一个数据  
 # k = np.random.randint(X\_train.shape[0])  
 k = it % X\_train.shape[0]  
 X\_data = X\_train[k]  
  
 # 把数据变为矩阵形式  
 X\_data = np.atleast\_2d(X\_data)  
  
 # 利用BP神经网络算法公式更新W1和W2  
 L1 = self.sigmoid(np.dot(X\_data, W1))  
 L2 = self.sigmoid(np.dot(L1, W2))  
  
 L2\_delta = (y\_train[k] - L2) \* self.dsigmoid(np.dot(L1, W2))  
 L1\_delta = np.dot(L2\_delta, W2.T) \* self.dsigmoid(np.dot(X\_data, W1))  
  
 # 更新权值  
 W2 += lr \* np.dot(L1.T, L2\_delta)  
 W1 += lr \* np.dot(X\_data.T, L1\_delta)  
  
 # 每训练1000次，预测一次准确率  
  
 if it % 1000 == 0:  
 output = self.predict(X\_test, W1, W2)  
 pre\_y = np.argmax(output, axis=1)  
 acc = np.mean(np.equal(pre\_y, y\_test))  
 accx.append(acc)  
 accy.append(it)  
 new\_entry = f"[迭代次数：{it}; 准确度 ： {acc}]"  
 current\_logs = self.log\_model.stringList()  
 current\_logs.append(new\_entry)  
 self.log\_model.setStringList(current\_logs)  
 self.listView.scrollToBottom()  
 # print("iterations:", it, "accuracy:", acc)  
 print(accx)  
 print(accy)  
 plt.plot(accy,accx)  
 plt.show()  
 # print('suceess 训练结束')  
 new\_entry = f"[训练结束success]"  
 current\_logs = self.log\_model.stringList()  
 current\_logs.append(new\_entry)  
 self.log\_model.setStringList(current\_logs)  
 self.listView.scrollToBottom()  
 self.W2=W2  
 self.W1=W1

* 1. **整体代码：**

from PyQt5 import QtCore, QtGui, QtWidgets  
import sch  
import sys  
from PyQt5.QtCore import QLibraryInfo  
from PyQt5.QtWidgets import \*  
from PyQt5.QtCore import \*  
from PyQt5.QtCore import QUrl  
from PyQt5.QtGui import \*  
from PyQt5.QtGui import QDesktopServices  
from PyQt5.QtWidgets import QTableWidgetItem, QFileDialog, QMessageBox  
import pandas as pd  
import numpy as np  
from PIL import Image  
from os import listdir  
from sklearn.datasets import load\_digits  
from sklearn.preprocessing import LabelBinarizer  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import classification\_report, confusion\_matrix  
import matplotlib.pyplot as plt  
import shutil  
  
class T(sch.Ui\_MainWindow,QtWidgets.QMainWindow) :  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
  
 self.predict\_Button.clicked.connect(self.open\_png)  
 self.Train\_Button.clicked.connect(self.train\_entry)  
  
 def open\_png(self) :  
 # getOpenFileNames 和 getOpenFileName的区别 一个是多个一个是单个文件  
 filename, \_ = QFileDialog.getOpenFileName(  
 self,  
 "选择文件",# title  
 "", # starting dir,  
 "图片文件 (\*.png \*.jpg \*.jpeg \*.bmp \*.gif)"  
 )  
  
 # 成员变量 - 存储  
 if not filename: # 用户取消选择  
 return  
 pixmap = QPixmap(filename)  
 if pixmap.isNull(): # 检查是否加载成功  
 QMessageBox.warning(self, "错误", "无法加载图片文件！")  
 return  
 self.preview\_picture.setPixmap(pixmap)  
 self.preview\_picture.setScaledContents(True) # 自适应QLabel大小  
 self.output\_num.setNum(self.img\_to\_arr(filename))  
  
 def img\_to\_arr(self,filename) : # 将 需要预测的图片转化为 数组形式  
 arr = []  
 fh = open(filename)  
 im = Image.open(filename)  
 for i in range(0,32) :  
 for j in range(0, 32):  
 pix = im.getpixel((i,j)) # RGB 三通道 颜色值  
 pixs = pix[0] + pix[1] + pix[2] # 转化为单通道颜色值  
 if pixs == 0 :  
 arr.append(1)  
 else :  
 arr.append(0)  
 print(self.predict(arr,self.W1,self.W2))  
 return np.argmax(self.predict(arr,self.W1,self.W2))  
  
  
  
  
  
 def train\_entry(self):  
 lr = self.lr\_num.toPlainText()  
 iter = self.iter\_num.toPlainText()  
 if not iter.isnumeric() or not self.IsFloatNum(lr) :  
 QMessageBox.warning(self, "错误", "输入不是浮点数和整数!")  
 return  
 self.lr = float(lr)  
 self.iter = int(iter)  
 self.img\_path = "./img" # 手写体数字图片路径  
 self.txt\_path = "./txt" # 转换后的数字矩阵的保存路径  
 self.train\_path = "./train/"  
 self.test\_path = "./test/"  
 self.img\_to\_txt(self.img\_path, self.txt\_path)  
 self.get\_split\_data()  
 X\_train, y\_train = self.get\_traindata()  
 X\_test, y\_test = self.get\_testdata()  
  
 y\_test = np.array(list(map(int, y\_test)))  
 y\_train = LabelBinarizer().fit\_transform(y\_train)  
 self.train(X\_train, y\_train, X\_test, y\_test)  
  
  
 # 建立训练数据  
 def get\_traindata(self):  
 labels = []  
 train\_file = listdir(self.train\_path)  
  
 trainarr = np.zeros((len(train\_file), 1024))  
  
 for i in range(0, len(train\_file)):  
 this\_label = train\_file[i].split("\_")[0]  
 if len(this\_label) != 0:  
 labels.append(int( train\_file[i].split(".")[0].split("\_")[0] ))  
 trainarr[i, :] = self.data\_to\_array(self.train\_path + train\_file[i])  
 print(this\_label+" : " + train\_file[i])  
  
 return trainarr, labels  
  
 def get\_testdata(self):  
  
 test\_files = listdir(self.test\_path)  
 true\_label\_set = []  
 test\_set = np.zeros((len(test\_files), 1024))  
  
 for i in range(0, len(test\_files)):  
 true\_label = (int)(test\_files[i].split("\_")[0])  
 testarr = self.data\_to\_array(self.test\_path + test\_files[i])  
 true\_label\_set.append(true\_label)  
 test\_set[i, :] = testarr  
 return test\_set, true\_label\_set  
  
 def data\_to\_array(self,fname):  
 arr = []  
 fh = open(fname)  
 for i in range(0, 32):  
 thisline = fh.readline()  
 # print(thisline)  
 for j in range(0, 32):  
 arr.append(int(thisline[j]))  
 return arr  
  
 def img\_to\_txt(self,png\_path, txt\_path):  
 png\_list = listdir(png\_path + "/" + 'png')  
 for f in png\_list:  
 fname = f.split(".")[0]  
 im = Image.open(png\_path + "/" + 'png' + "/" + f)  
 fh = open(txt\_path + "/" + fname + ".txt", "w")  
 for m in range(0, 32):  
 for n in range(0, 32):  
 pix = im.getpixel((n, m))  
 pixs = pix[0] + pix[1] + pix[2]  
 if pixs == 0:  
 fh.write("1")  
 else:  
 fh.write("0")  
 fh.write("\n")  
 fh.close()  
  
 def get\_split\_data(self):  
 txt\_list = listdir(self.txt\_path)  
  
 for txt in txt\_list:  
 try:  
 shutil.move(self.txt\_path + "/" + txt, self.test\_path)  
 except:  
 pass  
  
 def predict(self,X\_data, W1, W2):  
 L1 = self.sigmoid(np.dot(X\_data, W1))  
 L2 = self.sigmoid(np.dot(L1, W2))  
 return L2  
  
 def sigmoid(self,x):  
 return 1 / (1 + np.exp(-x))  
  
 def dsigmoid(self,x):  
 s = 1 / (1 + np.exp(-x))  
 return s \* (1 - s)  
  
 def train(self,X\_train, y\_train, X\_test, y\_test) :  
  
 # 定义神经网络由三层组成，每一层的神经元个数分别为1024、50、10，其中，1024像素即1024通道输入，隐藏层神经元个数为50，输出层为0~9，所以是10个  
 # 输入层到隐藏层的权值为W1，隐藏层到输出层的权值为W2  
 iter = self.iter  
 lr = self.lr  
 W1 = np.random.random((1024, 50)) - 0.5  
 W2 = np.random.random((50, 10)) - 0.5  
 accx = []  
 accy = []  
 for it in range(iter):  
 # 随机选取一个数据  
 # k = np.random.randint(X\_train.shape[0])  
 k = it % X\_train.shape[0]  
 X\_data = X\_train[k]  
  
 # 把数据变为矩阵形式  
 X\_data = np.atleast\_2d(X\_data)  
  
 # 利用BP神经网络算法公式更新W1和W2  
 L1 = self.sigmoid(np.dot(X\_data, W1))  
 L2 = self.sigmoid(np.dot(L1, W2))  
  
 L2\_delta = (y\_train[k] - L2) \* self.dsigmoid(np.dot(L1, W2))  
 L1\_delta = np.dot(L2\_delta, W2.T) \* self.dsigmoid(np.dot(X\_data, W1))  
  
 # 更新权值  
 W2 += lr \* np.dot(L1.T, L2\_delta)  
 W1 += lr \* np.dot(X\_data.T, L1\_delta)  
  
 # 每训练1000次，预测一次准确率  
  
 if it % 1000 == 0:  
 output = self.predict(X\_test, W1, W2)  
 pre\_y = np.argmax(output, axis=1)  
 acc = np.mean(np.equal(pre\_y, y\_test))  
 accx.append(acc)  
 accy.append(it)  
 new\_entry = f"[迭代次数：{it}; 准确度 ： {acc}]"  
 current\_logs = self.log\_model.stringList()  
 current\_logs.append(new\_entry)  
 self.log\_model.setStringList(current\_logs)  
 self.listView.scrollToBottom()  
 # print("iterations:", it, "accuracy:", acc)  
 print(accx)  
 print(accy)  
 plt.plot(accy,accx)  
 plt.show()  
 # print('suceess 训练结束')  
 new\_entry = f"[训练结束success]"  
 current\_logs = self.log\_model.stringList()  
 current\_logs.append(new\_entry)  
 self.log\_model.setStringList(current\_logs)  
 self.listView.scrollToBottom()  
 self.W2=W2  
 self.W1=W1  
  
  
 def IsFloatNum(self,str):  
 s = str.split('.')  
 if len(s) > 2:  
 return False  
 else:  
 for si in s:  
 if not si.isdigit():  
 return False  
 return True  
  
if \_\_name\_\_ == '\_\_main\_\_' :  
 app = QtWidgets.QApplication(sys.argv)  
 win = T()  
  
 win.show()  
 sys.exit(app.exec())