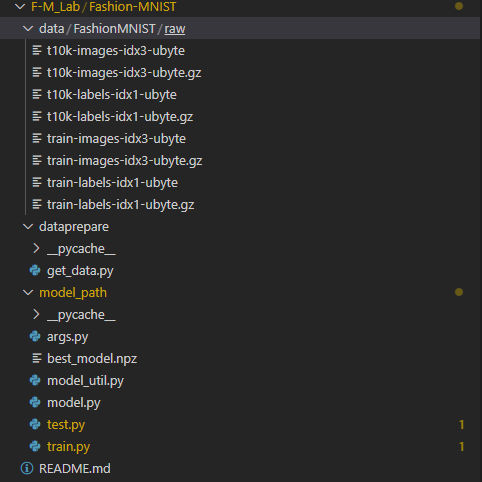
手工搭建三层神经网络分类器实验报告

一.项目结构：

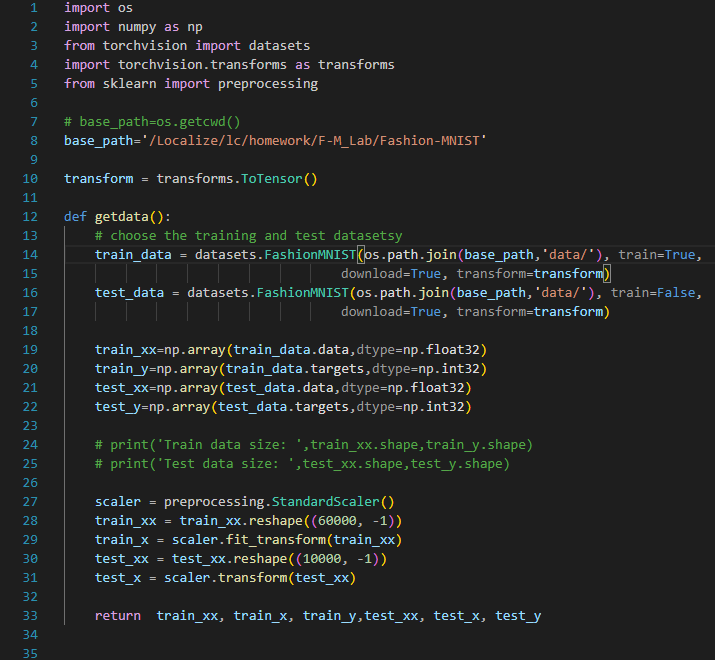


Data 目录保存下载的数据集

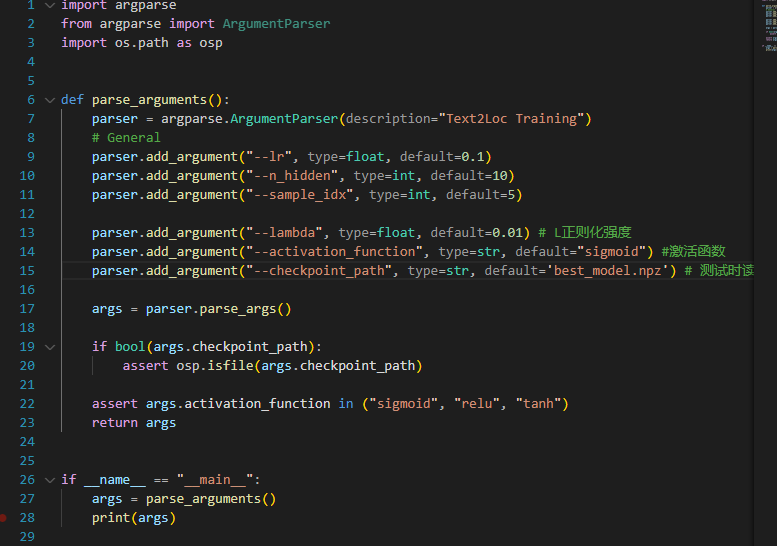
Dataprepare 数据集处理

Model\_path 目录下包含模型（库）、训练、测试和参数文件

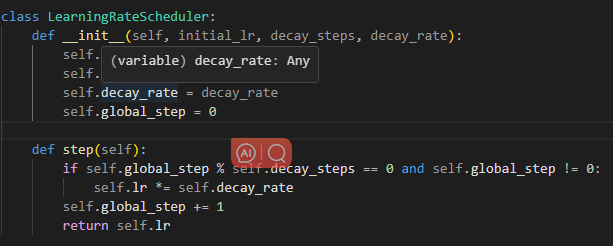
二．代码介绍



数据处理返回训练集测试集数据



设置超参数用于训练测试



学习率下降函数

class NeuralNet():

    """只有一层隐藏的MLP"""

    def \_\_init\_\_(self,input\_size,hidden\_size,output\_size):

        self.W\_h=np.random.uniform(size=(input\_size,hidden\_size),high=0.01,low=-0.01)

        self.b\_h=np.zeros(hidden\_size)

        self.W\_o = np.random.uniform(size=(hidden\_size, output\_size), high=0.01, low=-0.01)

        self.b\_o = np.zeros(output\_size)

        self.output\_size = output\_size

    def forward(self,X,keep\_activations=False):

        z\_h=np.dot(X,self.W\_h)+self.b\_h

        h = sigmoid(z\_h)  #激活函数

        z\_o = np.dot(h, self.W\_o) + self.b\_o

        y = softmax(z\_o)

        if keep\_activations:

            return y, h, z\_h

        else:

            return y

    def loss(self, X, y):

        return nll(one\_hot(self.output\_size, y), self.forward(X))

    def grad\_loss(self, x, y\_true):    #反向传播设计

        y, h, z\_h = self.forward(x, keep\_activations=True)

        grad\_z\_o = y - one\_hot(self.output\_size, y\_true)

        grad\_W\_o = np.outer(h, grad\_z\_o)

        grad\_b\_o = grad\_z\_o

        grad\_h = np.dot(grad\_z\_o, np.transpose(self.W\_o))

        grad\_z\_h = grad\_h \* dsigmoid(z\_h)

        grad\_W\_h = np.outer(x, grad\_z\_h)

        grad\_b\_h = grad\_z\_h

        grads = {"W\_h": grad\_W\_h, "b\_h": grad\_b\_h,

                 "W\_o": grad\_W\_o, "b\_o": grad\_b\_o}

        return grads

    def train(self, x, y, learning\_rate):    #SGD优化器

        grads = self.grad\_loss(x, y)

        self.W\_h = self.W\_h - learning\_rate \* grads["W\_h"]

        self.b\_h = self.b\_h - learning\_rate \* grads["b\_h"]

        self.W\_o = self.W\_o - learning\_rate \* grads["W\_o"]

        self.b\_o = self.b\_o - learning\_rate \* grads["b\_o"]

    def predict(self, X):

        if len(X.shape) == 1:

            return np.argmax(self.forward(X))

        else:

            return np.argmax(self.forward(X), axis=1)

    def accuracy(self, X, y):

        y\_preds = np.argmax(self.forward(X), axis=1)

        return np.mean(y\_preds == y)

    def save\_weights(self, file\_name):

        np.savez(file\_name, W1=self.W\_h, B1 =self.b\_h,W2=self.W\_o, B2 =self.b\_o)

    def load\_weights(self, file\_name):

        data = np.load(file\_name)

        self.W\_h=data['W1']

        self.b\_h=data['B1']

        self.W\_o = data['W2']

        self.b\_o = data['B2']

    def get\_weights(self,file\_name):

        data = np.load(file\_name)

        print('W\_h:',data['W1'])

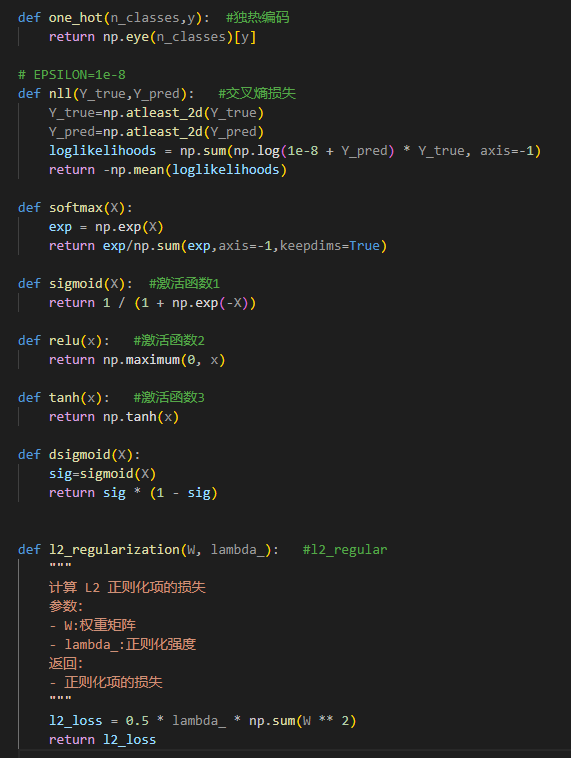
        print('b\_h:',data['B1'])

        print('W\_o:',data['W2'])

        print('b\_o:',data['B2'])

        return data['W1'],data['B1'],data['W2'],data['B2']

模型以及功能代码



激活函数，交叉熵损失，L2正则化

def plot\_prediction(model, sample\_idx=0, classes=range(10)):

    fig, (ax0, ax1) = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))

    ax0.imshow(test\_xx[sample\_idx].reshape(28,28))   #28\*28图片

    ax0.set\_title("True image label: %d" % test\_y[sample\_idx]); #标签

    ax1.bar(classes, one\_hot(len(classes), test\_y[sample\_idx]), label='true')

    ax1.bar(classes, model.forward(test\_x[sample\_idx]), label='prediction', color="red")

    ax1.set\_xticks(classes)

    prediction = model.predict(test\_x[sample\_idx])

    ax1.set\_title('Output probabilities (prediction: %d)'

                  % prediction)

    ax1.set\_xlabel('Digit class')

    ax1.legend()

    plt.show()

图片可视化以及预测结果可视化函数

def eval():

    n\_features = train\_x.shape[1]

    n\_classes = len(np.unique(train\_y))

    model = NeuralNet(n\_features, n\_hidden, n\_classes)

    model.load\_weights(checkpoint\_file) # 读取模型的权重

    W1, b1, W2, b2 =model.get\_weights(checkpoint\_file)  #权重参数可视化(参数查找)

    print('TEST Accuracy:',model.accuracy(test\_x, test\_y))  #测试结果

    if sample\_idx is not None:

        plot\_prediction(model, sample\_idx=sample\_idx)

    # 可视化权重矩阵的直方图

    plt.figure(figsize=(10, 10))

    plt.subplot(2, 2, 1)

    plt.hist(W1.flatten(), bins=50)

    plt.title('Layer 1 Weights Histogram')

    plt.xlabel('Weight Value')

    plt.ylabel('Frequency')

    plt.subplot(2, 2, 2)

    plt.hist(b1.flatten(), bins=50)

    plt.title('Layer 1 bias Histogram')

    plt.xlabel('Weight Value')

    plt.ylabel('Frequency')

    plt.subplot(2, 2, 3)

    plt.hist(W2.flatten(), bins=50)

    plt.title('Layer 2 Weights Histogram')

    plt.xlabel('Weight Value')

    plt.ylabel('Frequency')

    plt.subplot(2, 2, 4)

    plt.hist(b2.flatten(), bins=50)

    plt.title('Layer 2 bias Histogram')

    plt.xlabel('Weight Value')

    plt.ylabel('Frequency')

    plt.tight\_layout()

    plt.show()

测试函数，权重数据可视化

def train():

    print("Evaluation of the untrained model:")

    n\_features = train\_x.shape[1]

    n\_classes = len(np.unique(train\_y))

    model = NeuralNet(n\_features, n\_hidden, n\_classes)

    model.loss(train\_x, train\_y)

    model.accuracy(train\_x, train\_y)

    losses, accuracies, accuracies\_test = [], [], []

    losses.append(model.loss(train\_x, train\_y))

    accuracies.append(model.accuracy(train\_x, train\_y))

    accuracies\_test.append(model.accuracy(test\_x, test\_y))

    print("Random init: train loss: %0.5f, train acc: %0.3f, test acc: %0.3f"

        % (losses[-1], accuracies[-1], accuracies\_test[-1]))

    best\_acc= 0

    lr\_scheduler = LearningRateScheduler(initial\_lr=learning\_rate, decay\_steps=20, decay\_rate=0.8)

    for epoch in range(10):

        # 计算测试集准确率

        test\_acc = model.accuracy(test\_x, test\_y)

        # print(f"Epoch #{epoch + 1}, Test Accuracy: {test\_acc}")

        for step in range(2):

            lr = lr\_scheduler.step()

            for i, (x, y) in enumerate(zip(train\_x, train\_y)):

                model.train(x, y, lr)

            # print(f"Epoch {epoch + 1}, Step {step + 1}, Learning Rate: {lr}")

            # 保存最佳模型

            if test\_acc > best\_acc:

                best\_acc = test\_acc

                if os.path.exists(checkpoint\_file):

                    os.remove(checkpoint\_file)

                model.save\_weights(checkpoint\_file)

        # 记录训练损失和准确率

        losses.append(model.loss(train\_x, train\_y))

        accuracies.append(model.accuracy(train\_x, train\_y))

        # accuracies\_test.append(test\_acc)

        accuracies\_test.append(model.accuracy(test\_x, test\_y))

        print("Epoch #%d, train loss: %0.5f, train acc: %0.3f, test acc: %0.3f"

            % (epoch + 1, losses[-1], accuracies[-1],accuracies\_test[-1]))

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)

    plt.plot(losses)

    plt.title("Training loss");

    plt.subplot(1, 2, 2)

    plt.plot(accuracies, label='train')

    plt.plot(accuracies\_test, label='test')

    plt.ylim(0, 1.1)

    plt.ylabel("accuracy")

    plt.legend(loc='best');

    plt.show()

    if sample\_idx is not None:

        plot\_prediction(model, sample\_idx=sample\_idx)

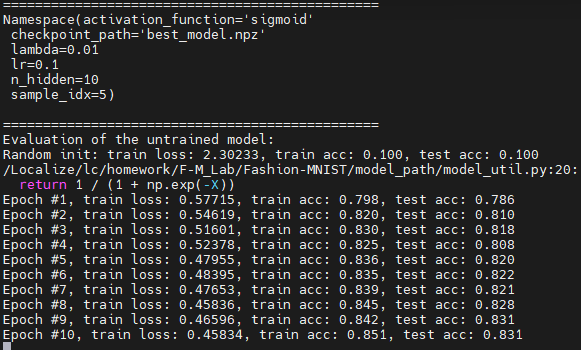
训练函数，loss 以及acc 输出以及可视化

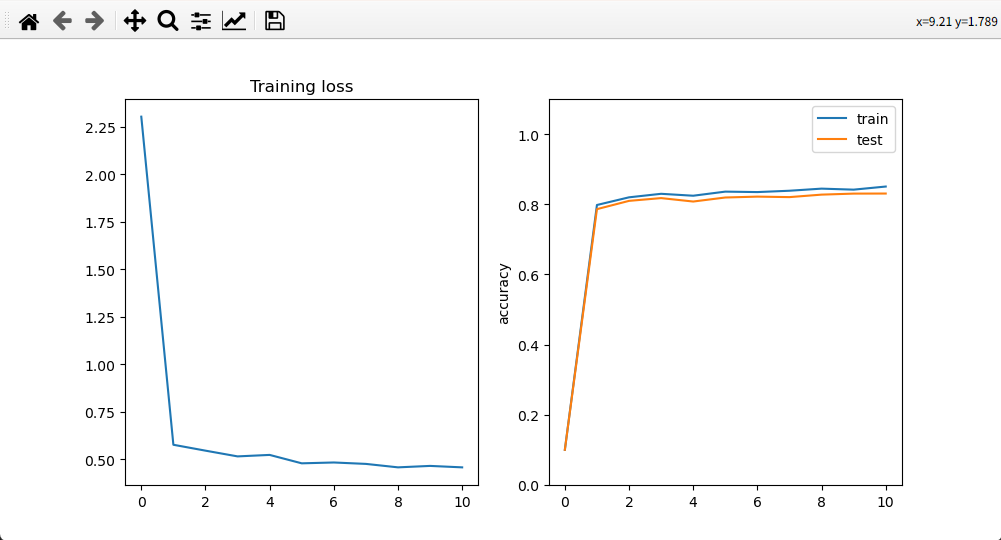
1. 实验

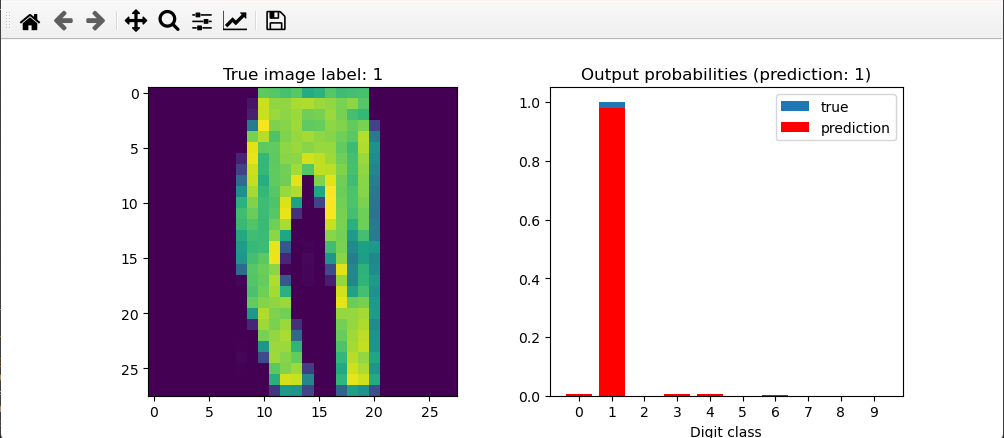
参考readme文件

1. 结果

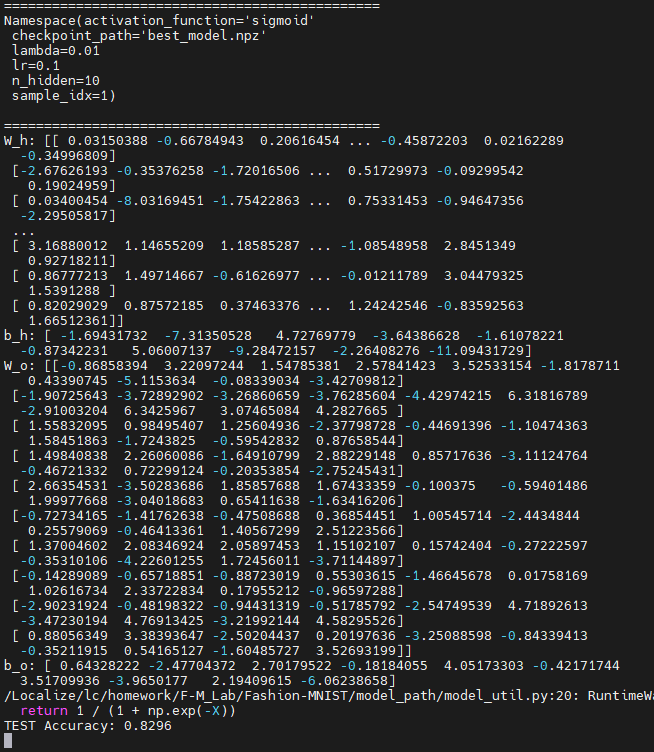
训练



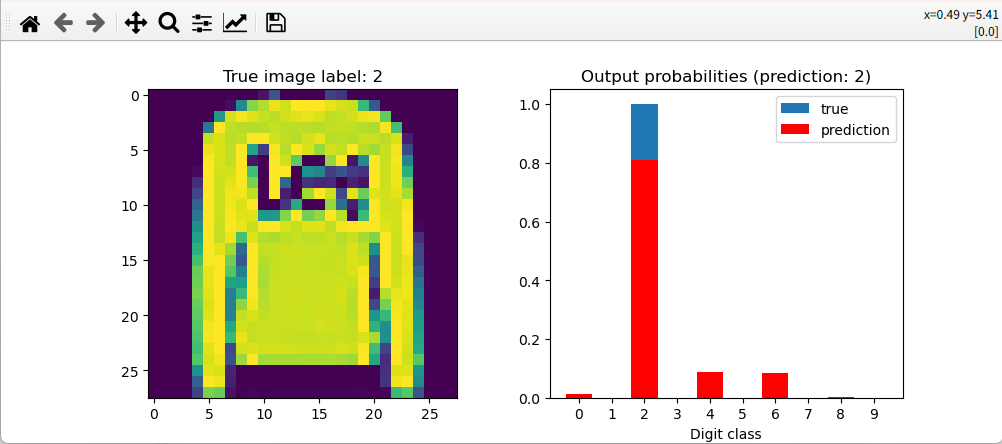




测试



预测



权重可视化

