机器学习课程报告
MLPNN

一、MLPNN 任务描述

基于 MNIST 数据集，设计并实现具有两种不同误差函数的 MLPNN 进行分类。
这里我们使用两种不同的方法去进行 MLPNN 分类
第一种是自己构建 MLPNN
第二种是使用 skearn 提供的 MLPNN 分类器

二、MLPNN 实验准备

（一）导入必要的包

```
import numpy as np
import pandas as pd

np.random.seed(0)
import random

from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.utils import to_categorical
from keras.datasets import mnist

import matplotlib.pyplot as plt
import seaborn as sns
```

keras 以 TensorFlow 和 Theano 作为后端封装，是一个专门用于深度学习的 python 模块。
包含了全连接层，卷积层，池化层，循环层，嵌入层等等等，常见的深度学习模型。
包含用于定义损失函数的 Losses，用于训练模型的 Optimizers，评估模型的 Metrics，定义激活函数的 Activations，防止过拟合的 Regularizers 等功能

三、MNIST 数据集描述

（一）导入 MNIST 数据集

使用 csv 文件导入

```
train = pd.read_csv("train.csv")
```

或者调库导入

```
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
```

一共统计了来自 250 个不同的人手写数字图片，其中 50%是高中生，50%来自人口普查局的工作人员。该数据集的收集目的是希望通过算法，实现对手写数字的识别。

注意：前 5000 个比后 5000 个要规整，这是因为前 5000 个数据来自于美国人口普查局的员工，而后 5000 个来自于大学生。

（二） MNIST 数据集特征

```
print(train_images.shape, test_images.shape)
```

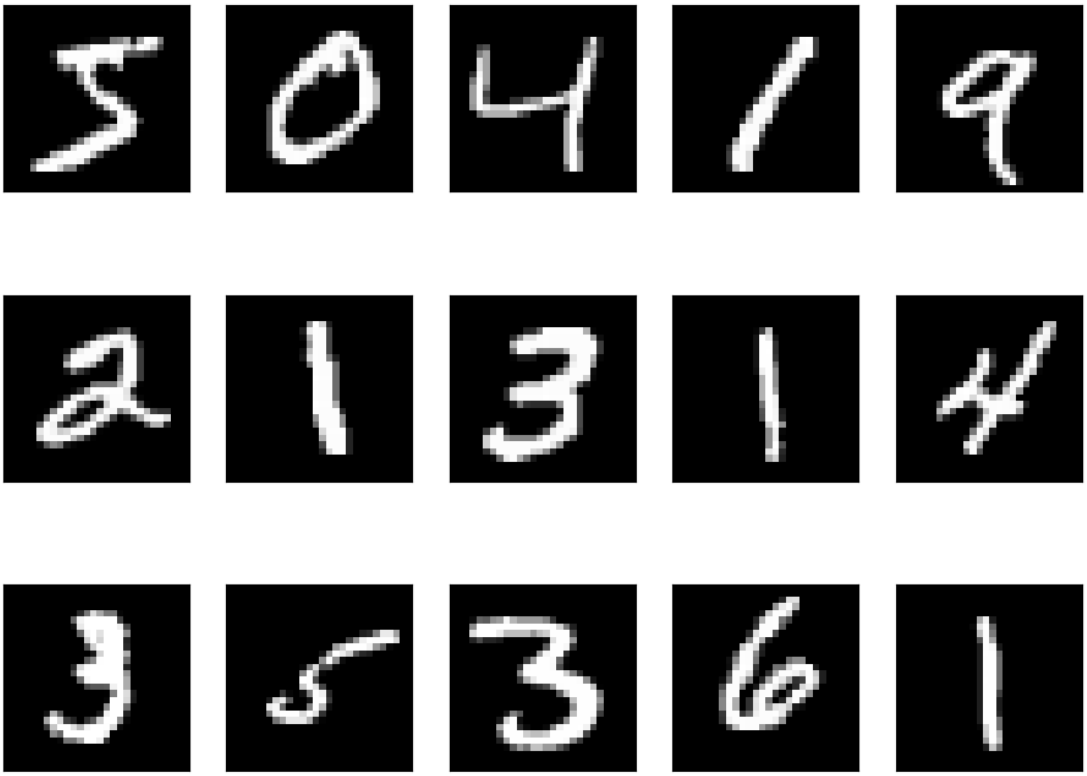
```
(60000, 28, 28) (10000, 28, 28)
```

训练集 train 一共包含了 60000 张图像和标签，而测试集一共包含了 10000 张图像和标签。idx3 表示 3 维，ubyte 表示是以字节的形式进行存储的，t10k 表示 10000 张测试图片（test10000）。每张图片是一个 28*28 像素点的 0 ~ 9 的手写数字图片，图像像素值为 0 ~ 255，越大该点越白。

（三） MNIST 数据集内容

```
#画图 3 行 5 列
n_row=3
n_col=5
plt.figure(figsize=(1.8*n_col,2.4*n_row))
plt.subplots_adjust(bottom=0,left=.01,right=.99,top=.90,hspace=.35)
for i in range(n_row*n_col):
    plt.subplot(n_row,n_col,i+1)
    plt.imshow(train_images[i],cmap=plt.cm.gray)#plt.cm.gray 看灰度图
    plt.xticks(())#去掉 x 坐标
    plt.yticks(())


plt.show()
```

展示了前 15 个数据的内容



（四）以灰度值查看数字图像

```
# 灰度函数返回数字
def plot_digit(digit, dem = 28, font_size = 12):
    max_ax = font_size * dem

    fig = plt.figure(figsize=(13, 13))
    plt.xlim([0, max_ax])
    plt.ylim([0, max_ax])
    plt.axis('off')
    black = '#000000'

    for idx in range(dem):
        for jdx in range(dem):

            t = plt.text(idx * font_size, max_ax - jdx*font_size,
digit[jdx][idx], fontsize = font_size, color = black)
            c = digit[jdx][idx] / 255.
            t.set_bbox(dict(facecolor=(c, c, c), alpha = 0.5, edgecolor
= 'black'))

    plt.show()
```

定义完成以灰度值返回数字图像函数，下面随机选取图像进行输出

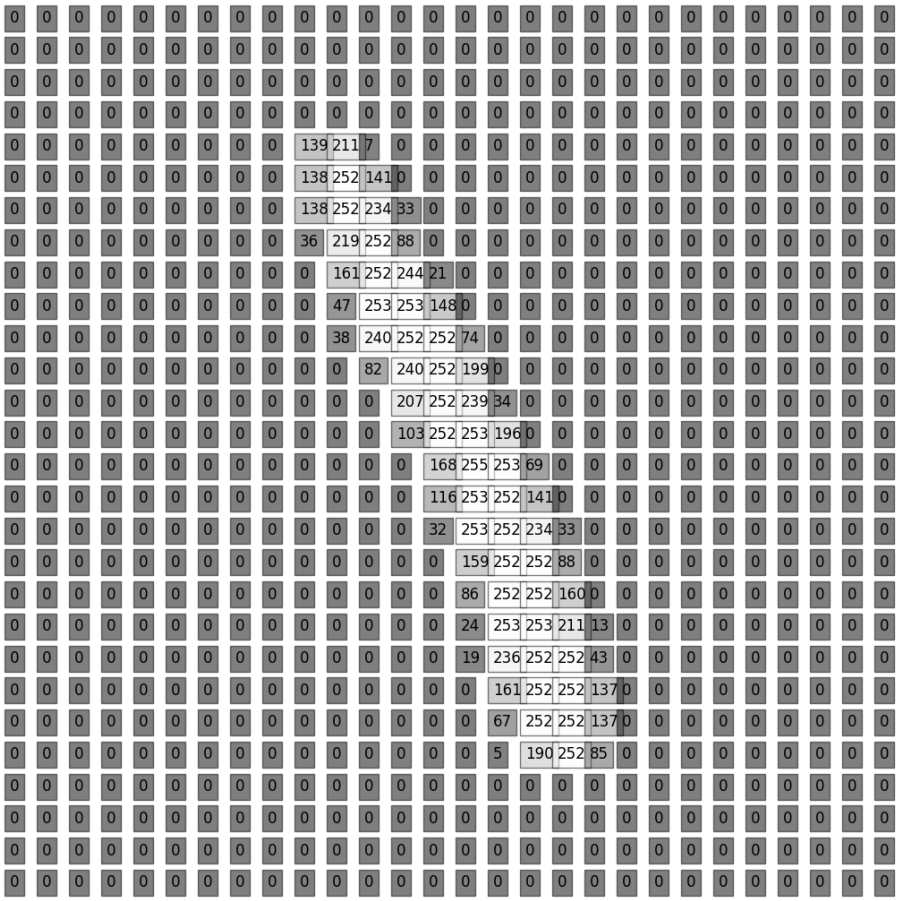
```
train_images_plot = train_images.reshape(-1, 28, 28)

rand_number = random.randint(0, len(train_labels))
print(train_labels[rand_number])
plot_digit(train_images_plot[rand_number])
```

random.randint 产生随机整数。返回从低（包括）到高（不包括）的随机整数。 从“半开”区间 [low, high) 中指定 dtype 的“离散均匀”分布返回随机整数。

标签为:

```
1
```

输出图像为:

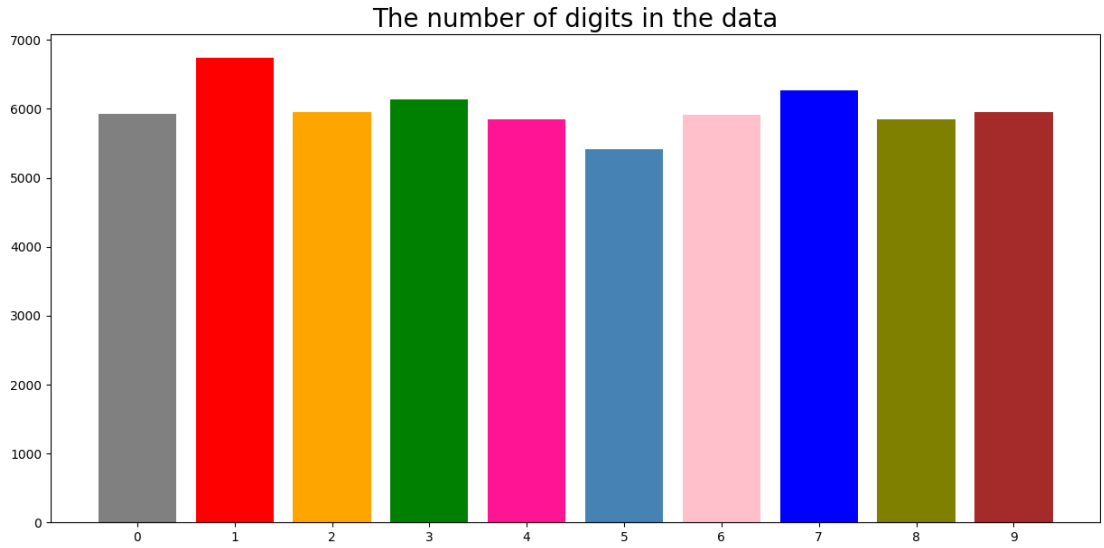


（五）查看不同标签数量

```
digit_range = np.arange(10)

y_train=pd.Series(train_labels)

val = y_train.value_counts().index
cnt = y_train.value_counts().values
mycolors = ['red', 'blue', 'green', 'orange', 'brown', 'grey',
'pink', 'olive', 'deeppink', 'steelblue']
```

```
plt.figure(figsize = (15, 7))
plt.title("The number of digits in the data", fontsize = 20)
plt.xticks(range(10))
plt.bar(val, cnt, color = mycolors);
```

在 pandas 中，value_counts 常用于数据表的计数及排序，它可以用来查看数据表中，指定列里有多少个不同的数据值，并计算每个不同值有在该列中的个数，同时还能根据需要进行排序。



四、数据预处理

（一）MLPNN--MNIST 数据集

1.MNIST 数据集形状变换

```
train_images = train_images.reshape(60000, 28 * 28).astype('float32')
/ 255
test_images = test_images.reshape(10000, 28 * 28).astype('float32') /
255
```

reshape()函数可以对数组的结构进行改变。

astype 方法：通用函数，可以用于把 dataframe 中的任何列转换成其他类型。需要将 MNIST 数据集变换为一个形状为(60000, 28 * 28)，取值范围为 0~1 的数组。

（二）MNIST 数据集选取部分数据

```
x_train, y_train = train_images[:7000], train_labels[:7000]
x_test, y_test = test_images[:3000], test_labels[:3000]
```

选取 7500 个训练数据，2500 个测试数据，

（三）MNIST 数据集 PCA 降维

```
pca = PCA(n_components=100)#把原来点压缩为 100 个维度
x_train_pca = pca.fit_transform(x_train)
```

```
y_train_pca = y_train
x_test_pca = pca.transform(x_test)
y_test_pca = y_test
```

主成分分析，提取数据的主要成分，剔除数据中相对次要的成分，换句话说 PCA 的目标是降维，就是剔除数据次要成分的维度。剔除数据中相对次要的成分，一次来压缩数据量的大小、降低数据变量的复杂度。

高维数据中变量之间的关系是不可见的。一个合理的方法，在降低维度的同时，尽量的减少数据信息的损失，这样对于数据的处理是可以接受的。

```
print(pca_x_train.shape,pca_x_test.shape)
```

```
(7500, 100) (2500, 100)
```

把原来的像素点压缩为 100 个维度

五、MLPNN 算法描述

（一）感知机模型

已知感知机由两层神经元组成，故感知机模型的公式可表示为

$$y = f(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \theta)$$

其中，$\boldsymbol{x} \in \mathrm{R}^n$ 为样本的特征向量，是感知机模型的输入；$\boldsymbol{w}$, θ 是感知机模型的参数，$\boldsymbol{w} \in \mathrm{R}^n$ 为权重，θ 为阈值。假定 f 为阶跃函数，那么感知机模型的公式可进一步表示为

$$y = \mathrm{sgn}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \theta) = \begin{cases} 1, & \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \theta \geq 0 \\ 0, & \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \theta < 0 \end{cases}$$

由于 n 维空间中的超平面方程为

$$w_1x_1 + w_2x_2 + \cdots + w_nx_n + b = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0$$

所以此时感知机模型公式中的 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \theta$ 可以看作是 n 维空间中的一个超平面，通过它将 n 维空间划分为 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \theta \geq 0$ 和 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \theta < 0$ 两个子空间，落在前一个子空间的样本对应的模型输出值为 1，落在后一个子空间的样本对应的模型输出值为 0，以此来实现分类功能。

（二）感知机学习策略

给定一个线性可分的数据集 T，感知机的学习目标是求得能对数据集 T 中的正负样本完全正确划分的分离超平面：

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \theta = 0$$

假设此时误分类样本集合为 $M \subseteq T$，对任意一个误分类样本$(\boldsymbol{x}, y) \in M$来说，当 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \theta$

≥ 0 时，模型输出值为 $\hat{y}=1$，样本真实标记为 $y=0$；反之，当 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-\theta<0$ 时，模型输出值为 $\hat{y}=0$，样本真实标记为 $y=1$。综合两种情形可知，以下公式恒成立

$$(\hat{y}-y)(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-\theta)\geq 0$$

所以，给定数据集 T，其损失函数可以定义为：

$$L(\boldsymbol{w},\theta)=\sum_{\boldsymbol{x}\in M}(\hat{y}-y)(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}-\theta)$$

显然，此损失函数是非负的。如果没有误分类点，损失函数值是 0。而且，误分类点越少，误分类点离超平面越近，损失函数值就越小。因此，给定数据集 T，损失函数 $L(\boldsymbol{w},\theta)$ 是关于 $\boldsymbol{w},\theta$ 的连续可导函数。

（三）感知机学习算法

感知机模型的学习问题可以转化为求解损失函数的最优化问题，具体地，给定数据集

$$T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_N,y_N)\}$$

其中 $\boldsymbol{x}_i\in\mathrm{R}^n$，$y_i\in\{0,1\}$，求参数 $\boldsymbol{w},\theta$，使其为极小化损失函数的解：

$$\min_{\boldsymbol{w},\theta}L(\boldsymbol{w},\theta)=\min_{\boldsymbol{w},\theta}\sum_{\boldsymbol{x}_i\in M}(\hat{y}_i-y_i)(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i-\theta)$$

其中 $M\subseteq T$ 为误分类样本集合。若将阈值 θ 看作一个固定输入为 -1 的“哑节点”，即

$$-\theta=-1\cdot w_{n+1}=x_{n+1}\cdot w_{n+1}$$

那么 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i-\theta$ 可化简为

$$\begin{aligned}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_{\boldsymbol{i}}-\theta&=\sum_{j=1}^{n}w_jx_j+x_{n+1}\cdot w_{n+1}\\&=\sum_{j=1}^{n+1}w_jx_j\\&=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_{\boldsymbol{i}}\end{aligned}$$

其中 $\boldsymbol{x}_{\boldsymbol{i}}\in\mathrm{R}^{n+1}$，$\boldsymbol{w}\in\mathrm{R}^{n+1}$。根据该式，可将要求解的极小化问题进一步简化为

$$\min_{\boldsymbol{w}}L(\boldsymbol{w})=\min_{\boldsymbol{w}}\sum_{\boldsymbol{x}_{\boldsymbol{i}}\in M}(\hat{y}_i-y_i)\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_{\boldsymbol{i}}$$

假设误分类样本集合 M 固定，那么可以求得损失函数 $L(\boldsymbol{w})$ 的梯度为：

$$\nabla_{\boldsymbol{w}}L(\boldsymbol{w})=\sum_{\boldsymbol{x}_{\boldsymbol{i}}\in M}(\hat{y}_i-y_i)\boldsymbol{x}_{\boldsymbol{i}}$$

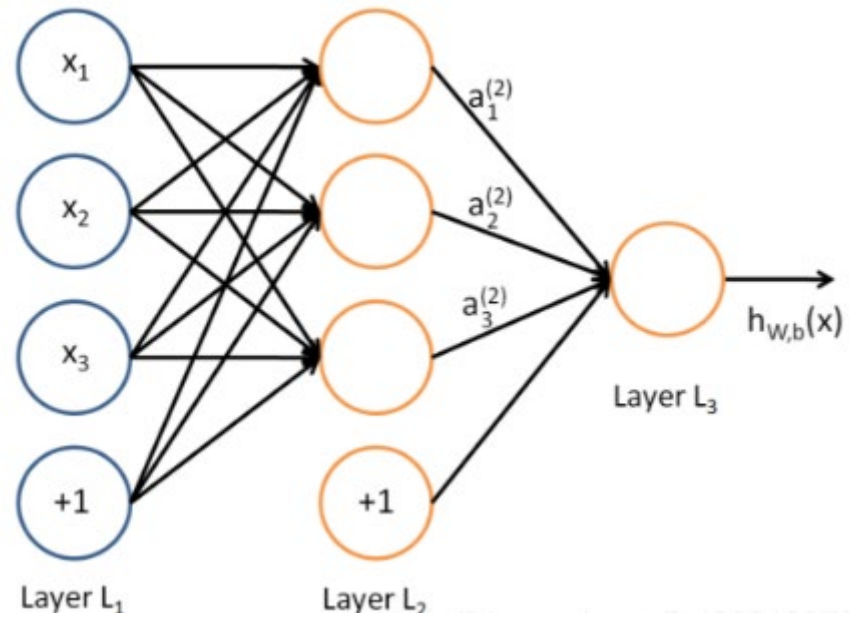
感知机的学习算法具体采用的是随机梯度下降法，也就是极小化过程中不是一次使 M 中所有误分类点的梯度下降，而是一次随机选取一个误分类点使其梯度下降。所以权重 $\boldsymbol{w}$ 的更新公式为

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta \boldsymbol{w}$$

$$\Delta \boldsymbol{w} = -\eta(\hat{y}_i - y_i)\boldsymbol{x}_i = \eta(y_i - \hat{y}_i)\boldsymbol{x}_i$$

相应地，$\boldsymbol{w}$ 中的某个分量 w_i 的更新公式即为公式

$$\Delta w_i = \eta(y - \hat{y})x_i$$



六、实验过程：构建的 MLPNN

（一）构建 MLPNN

Sequential 模型可以构建非常复杂的神经网络，包括全连接神经网络、卷积神经网络(CNN)、循环神经网络(RNN)、等等。Sequential 通过堆叠许多层，构建出 MLPNN。

```
model = Sequential()
#具有 30 个神经元的层
model.add(Dense(30, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(20, activation='relu'))
model.add(Dropout(0.2))
#最终分类层，每个输出类有 1 个神经元。 Softmax 划分每个类的概率。
model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

model.summary()输出模型各层的参数状况。通过这些参数，可以看到模型各个层的组成。也能看到数据经过每个层后，输出的数据维度。

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense (Dense)                (None, 30)                23550

 dropout (Dropout)            (None, 30)                0

 dense_1 (Dense)              (None, 20)                620

 dropout_1 (Dropout)          (None, 20)                0

 dense_2 (Dense)              (None, 10)                210
```

dense 表示全连接层，全连接层神经网络的 Param，说明的是每层神经元权重的个数。Param = （输入数据维度+1）* 神经元个数，之所以要加 1，是考虑到每个神经元都有一个 Bias。

（二）配置模型

model.compile()方法用于在配置训练方法时，告知训练时用的优化器、损失函数和准确率评测标准

```
model.compile(optimizer = 优化器,
              loss = 损失函数,
              metrics = ["准确率"])
```

```
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

交叉熵损失函数通过计算以下和来计算示例的损失:

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i$$

根据公式我们可以发现，因为 yi,要么是 0，要么是 1。而当 yi 等于 0 时，结果就是 0，当且仅当 yi 等于 1 时，才会有结果。也就是说 categorical_crossentropy 只专注与一个结果，因而它一般配合 softmax 做单标签分类。

（三）进行训练

model.fit( ) ： 将训练数据在模型中训练一定次数，返回 loss 和测量指标

```
batch_size = 32
epochs = 30
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
```

```
                    verbose=1,
                    validation_data=(x_val, y_val))
```

batch_size　每一个 batch 的大小（批尺寸），即训练一次网络所用的样本数
epochs　迭代次数，即全部样本数据将被轮替多少次，轮完训练停止
verbose　0:不输出信息；1:显示进度条(一般默认为 1)；2:每个 epoch 输出一行记录；
validation_data　指定验证数据，该数据将覆盖 validation_spilt 设定的数据
部分截图：

```
Epoch 1/30
1182/1182 [==============================] - 3s 2ms/step - loss: 0.7794 - accuracy: 0.7543 - val_loss: 0.2966 -
val_accuracy: 0.9102
Epoch 2/30
1182/1182 [==============================] - 2s 2ms/step - loss: 0.4378 - accuracy: 0.8685 - val_loss: 0.2377 -
val_accuracy: 0.9329
Epoch 3/30
1182/1182 [==============================] - 2s 2ms/step - loss: 0.3781 - accuracy: 0.8897 - val_loss: 0.2233 -
val_accuracy: 0.9333
```

七、实验过程：sklearn 中 MLPNN

（一）定义显示函数

因为在使用 sklearn 中使用了两种不同的激活函数，有部分代码可以重用
ClaReport 函数用来输出分类报告

```
def ClaReport(ytest,predic):
    # 召回率、准确率、F1
    print('precision:%.4f' % precision_score(y_true=ytest,
y_pred=predic,average= 'macro'))
    print('recall:%.4f' % recall_score(y_true=ytest,
y_pred=predic,average= 'macro'))
    print('F1:%.4f' % f1_score(y_true=ytest, y_pred=predic,average=
'macro'))
```

sns.heatmap 用颜色编码的矩阵来绘制矩形数据，MatVisual 函数用来绘制可视化的混淆矩阵

```
def MatVisual(conf):
    fig, ax = plt.subplots(figsize=(10, 9))

    sns.heatmap(conf,annot=True,ax=ax) #plot heatmap


    ax.set_title('Confusion Matrix \n\n')
    ax.set_xlabel('predict')
    ax.set_ylabel('true') #

    plt.xlabel('predicted label')
    plt.ylabel('true label')
    plt.show()
```

（二）两层 MLPNN，激活函数为 relu

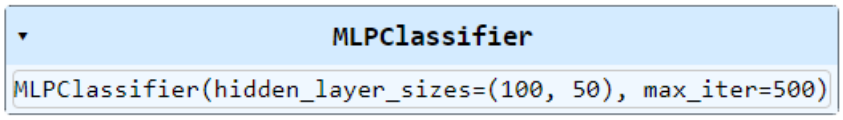
1.定义 MLPNN

```
mlp=MLPClassifier(hidden_layer_sizes=(100,50),activation="relu",max_iter=500)
```

激活函数：{'identity'，'logistic'，'tanh'，'relu'}，默认'relu' 隐藏层的激活函数：'identity'，无操作激活，对实现线性瓶颈很有用，返回 f（x）= x；'logistic'，logistic sigmoid 函数，返回 f（x）= 1 / （1 + exp（-x））；'tanh'，双曲 tan 函数，返回 f（x）= tanh（x）；'relu'，整流后的线性单位函数，返回 f（x）= max（0，x）

2.训练 MLPNN

```
mlp.fit(x_train_pca,y_train_pca)
```



（三）两层 MLPNN，激活函数为 logistic sigmoid

1.定义 MLPNN

```
mlp=MLPClassifier(hidden_layer_sizes=(100,50),activation="logistic",max_iter=500)
```

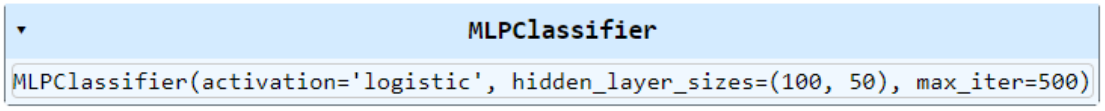
hidden_layer_sizes 第 i 个元素表示第 i 个隐藏层中的神经元数量。

hidden_layer_sizes=(100,50)两个隐藏层，第一个 100 个神经元，第二个 50 个神经元。还能再加。

max_iter：int，optional，默认值 200。最大迭代次数。solver 迭代直到收敛（由'tol'确定）或这个迭代次数。对于随机解算器（'sgd'，'adam'），请注意，这决定了时期的数量（每个数据点的使用次数），而不是梯度步数。

2.训练 MLPNN

```
mlp.fit(x_train_pca,y_train_pca)
```



八、实验结果分析

（一）评价指标

1.准确率（Accuracy）

所有预测正确的（包括正类和负类）占总的的比例。

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + TN + FP} = \frac{TP + TN}{P + N}$$

2.精确率（Precision）

查准率。即正确预测为正类的占全部预测为正类的的比例。

精确率是针对我们预测结果而言的，它表示的是预测为正类的样本中有多少是真正的正类样本。那么预测为正类就有两种可能了，一种就是把正类预测为正类(TP)，另一种就是把负类预测为正类(FP)。

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

3.召回率（Recall）

正确预测为正类的占全部实际为正类的的比例。召回率是针对我们原始样本而言的，它表示的是全体样本中的所有正类样本有多少被预测正确了。

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{P}}$$

4.混淆矩阵

混淆矩阵是对分类问题的预测结果的总结。使用计数值汇总正确和不正确预测的数量，并按照每个类别进行细分，这是混淆矩阵的关键所在。混淆矩阵显示了额分类模型在进行预测时会对那一部分产生混淆，不仅能了解分类模型所犯的错误，更重要的是可以了解发生错误的类型。这是这种对结果的分解客服了仅使用分类准确率所带来的局限性。

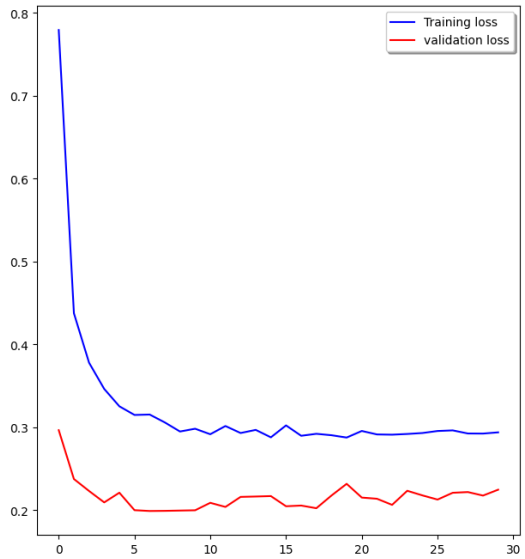
（二）构建的 MLPNN 结果分析

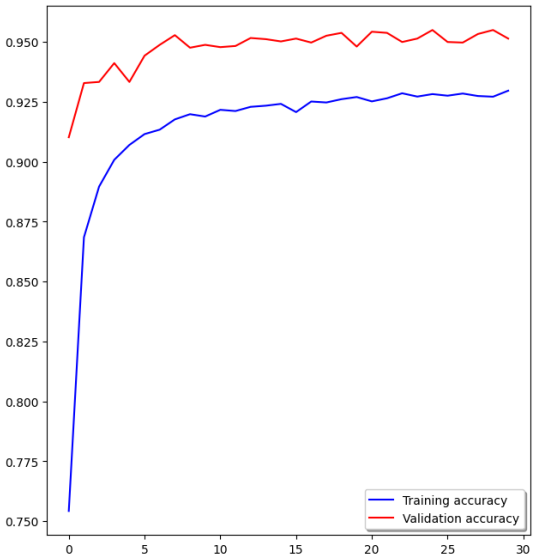
1.查看 loss 与 accuracy

plt.subplots 会产生一个 figure 和一系列的 subplots 的。 用户不需要每次都设置所有属性。

```
fig, ax = plt.subplots(1,2, figsize=(16,8))
ax[0].plot(history.history['loss'], color='b', label="Training loss")
ax[0].plot(history.history['val_loss'], color='r', label="validation
loss",axes =ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training
accuracy")
ax[1].plot(history.history['val_accuracy'],
color='r',label="Validation accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```

在分类问题中，可能准确率更加的直观，也更具有可解释性，更重要，但是它不可微，无法直接用于网络训练，因为反向传播算法要求损失函数是可微的。而损失函数一个很好的性质就是可微，可以求梯度，运用反向传播更新参数。即首选损失不能直接优化(比如准确率)时，可以使用与真实度量类似的损失函数。





交叉熵损失函数：交叉熵输出的是正确标签的似然对数，和准确率有一定的关系，但是取值范围更大。

准确率：在分类问题中，准确率的定义简单粗暴，就是看预测的类别是否和真实的类别一致。

2.测试集结果

model.evaluate 输入数据和标签,输出损失和精确度， 输入数据(data)和真实标签(label),然后将预测结果与真实标签相比较,得到两者误差并输出.

```
score = model.evaluate(x_val, y_val, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.22485488653182983
Test accuracy: 0.9514285922050476
```

通过模型在的 accuracy ， 计算模型正确分类的样本数与总样本数之比以衡量模型的效果，目标是对模型的效果进行度量。

通过损失函数的计算，更新模型参数，目标是为了减小优化误差，即在损失函数与优化算法的共同作用下，减小模型的经验风险。

（三）sklearn 中 MLPNN 结果分析

1.激活函数为 Relu

（1）分类结果

model.predict 输入测试数据,输出预测结果

model.predict 不需要,只是单纯输出预测结果,全程不需要真实标签的参与.

```
predictions=mlp.predict(x_test_pca)#做预测
print(classification_report(y_test_pca,predictions)
```

classification_report()是 python 在机器学习中常用的输出模型评估报告的方法。

	precision	recall	f1-score	support
0	0.92	0.98	0.95	271
1	0.97	0.99	0.98	340
2	0.93	0.92	0.92	313
3	0.92	0.93	0.93	316
4	0.91	0.94	0.92	318
5	0.94	0.89	0.91	283
6	0.93	0.94	0.93	272
7	0.93	0.91	0.92	306
8	0.91	0.85	0.88	286
9	0.91	0.92	0.92	295
accuracy			0.93	3000
macro avg	0.93	0.93	0.93	3000
weighted avg	0.93	0.93	0.93	3000

accuracy,macro avg（宏平均）:算术平均
weighted avg（加权平均）：除开本身的比例，还要算上该种类样本占所有样本的比例。

```
ClaReport(y_test,predictions)
```

```
precision:0.9261
recall:0.9259
F1:0.9257
```

precision 是表示预测为正样本中，被实际为正样本的比例。可以看出 precision 是考虑的正样本被预测正确的比例．其计算公式为：P = TP / (TP + FP)

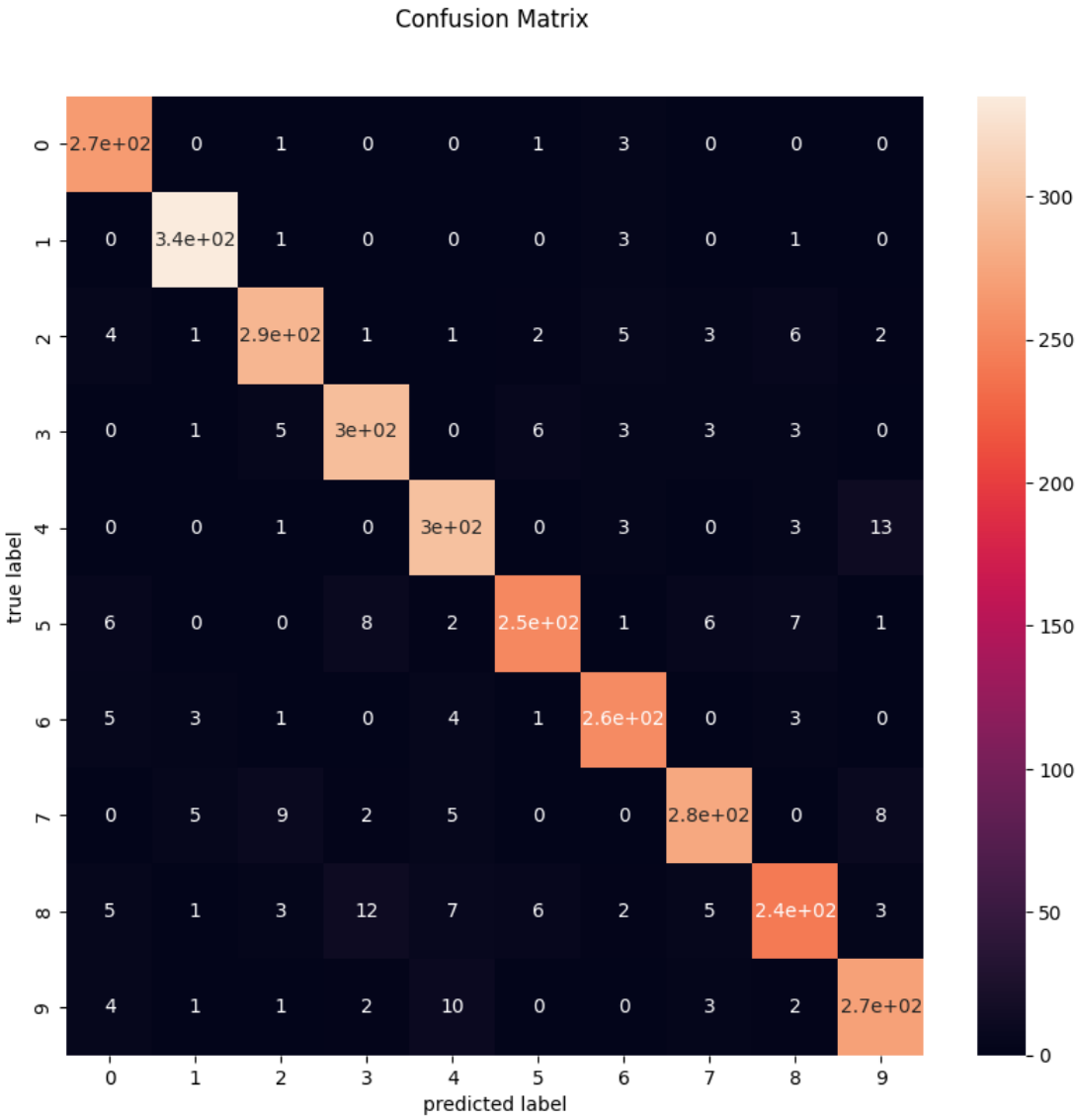
（2）混淆矩阵

```
confmat = confusion_matrix(y_true=y_test_pca, y_pred=predictions)  #输出混淆矩阵
print(confmat)
```

```
[[266   0   1   0   0   1   3   0   0   0]
 [  0 335   1   0   0   0   3   0   1   0]
 [  4   1 288   1   1   2   5   3   6   2]
 [  0   1   5 295   0   6   3   3   3   0]
 [  0   0   1   0 298   0   3   0   3  13]
 [  6   0   0   8   2 252   1   6   7   1]
 [  5   3   1   0   4   1 255   0   3   0]
 [  0   5   9   2   5   0   0 277   0   8]
 [  5   1   3  12   7   6   2   5 242   3]
 [  4   1   1   2  10   0   0   3   2 272]]
```

混淆矩阵是对分类问题的预测结果的总结。使用计数值汇总正确和不正确预测的数量，并按照每个类别进行细分，这是混淆矩阵的关键所在。

```
MatVisual(confmat)
```



2.激活函数为 logistic sigmoid

（1）分类结果

```
predictions=mlp.predict(x_test_pca)#做预测
print(classification_report(y_test_pca,predictions))
```

	precision	recall	f1-score	support
0	0.91	0.98	0.95	271
1	0.96	0.98	0.97	340
2	0.90	0.91	0.91	313
3	0.90	0.90	0.90	316
4	0.91	0.93	0.92	318
5	0.89	0.87	0.88	283
6	0.92	0.90	0.91	272
7	0.92	0.90	0.91	306
8	0.91	0.85	0.88	286
9	0.90	0.90	0.90	295
accuracy			0.91	3000
macro avg	0.91	0.91	0.91	3000
weighted avg	0.91	0.91	0.91	3000

精确率（accuracy）和召回率（recall）计算公式的分子都是 TP 也就是正样本被预测为正样本的数量，可知其为正样本的精确率和正样本的召回率．而准确率（accuracy）主要表征的是整体预测正确的比例．

```
ClaReport(y_test,predictions)
```

```
precision:0.9126
recall:0.9124
F1:0.9122
```

召回率是表示实际为正样本中，预测为正样本的比例。

准确率表示所有的预测样本中，预测正确的比例。

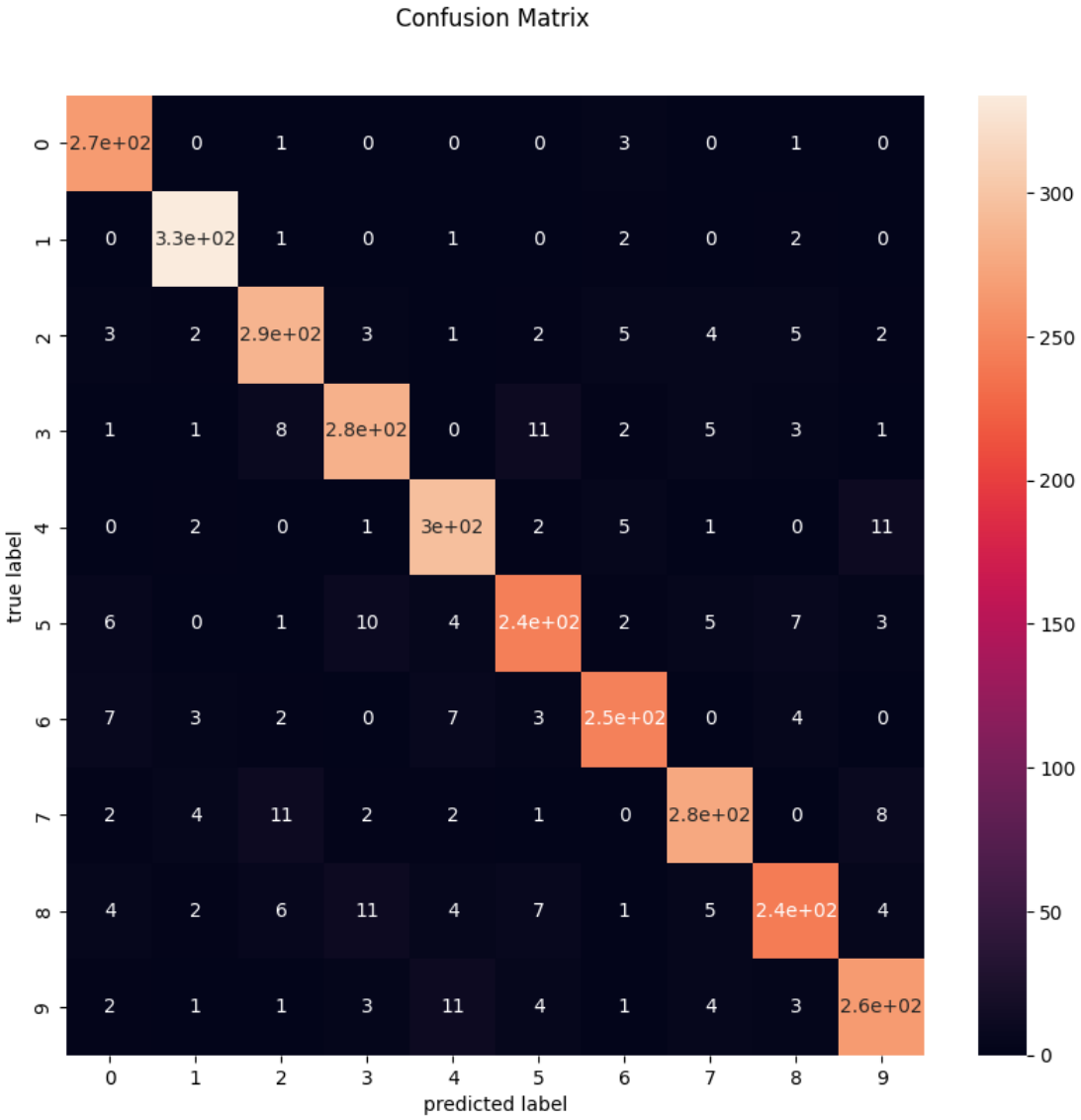
（2）混淆矩阵

```
confmat = confusion_matrix(y_true=y_test_pca, y_pred=predictions)  #输出混淆矩阵
print(confmat)
```

```
[[266   0   1   0   0   0   3   0   1   0]
 [  0 334   1   0   1   0   2   0   2   0]
 [  3   2 286   3   1   2   5   4   5   2]
 [  1   1   8 284   0  11   2   5   3   1]
 [  0   2   0   1 296   2   5   1   0  11]
 [  6   0   1  10   4 245   2   5   7   3]
 [  7   3   2   0   7   3 246   0   4   0]
 [  2   4  11   2   2   1   0 276   0   8]
 [  4   2   6  11   4   7   1   5 242   4]
 [  2   1   1   3  11   4   1   4   3 265]]
```

混淆矩阵显示了额分类模型在进行预测时会对那一部分产生混淆，不仅能了解分类模型所犯的错误，更重要的是可以了解发生错误的类型。这是这种对结果的分解客服了仅使用分类准确率所带来的局限性。

```
MatVisual(confmat)
```



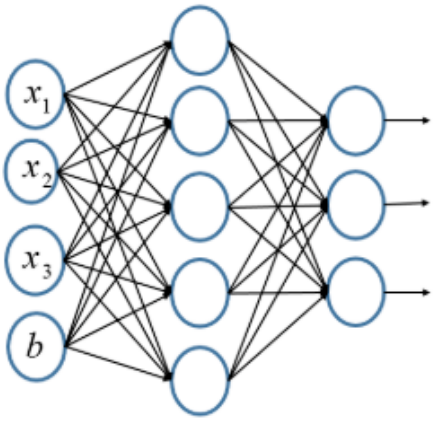
九、总结

（一）多层感知机

多层感知机(MLP, Multilayer Perceptron)也叫人工神经网络(ANN, Artificial Neural Network)，除了输入输出层，它中间可以有多个隐层。

一层神经元越多，导致模型很宽，这叫做浅度学习。这样的模型非常容易过拟合，不好训练。

MLP 并没有限定隐层的数量，对于输出层神经元的个数也没有限制，所以我们可以根据各自的需求选择合适的隐层层数。



（二）激活函数

引入隐藏层的神经网络可以等价于仅含输出层的单层神经网络的问题，在于全连接层只是对数据做仿射变换（affine transformation），而多个仿射变换的叠加仍然是一个仿射变换。解决问题的方法之一是引入非线性变换，对隐藏变量使用按元素运算的非线性函数进行变换，再作为下一个全连接层的输入。

使用激活函数，能够给神经元引入非线性因素，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以利用到更多的非线性模型中。