

2022 – 2023 第一学期 实验报告

课程编号： 2801000060， 课程名称： 机器人学导论， 主讲教师： 邱国平

学号： 2022280485	姓名： 贾苏健	专业年级： 电子信息工程
----------------	---------	--------------

Coursework 1 (20%)

教师评语/成绩

目录

一、实验原理	3
1、单层感知器	3
2、特征量	3
3、训练过程	3
4、效果评估	6
5、多元分类	7
二、实验过程	8
1、实现过程	8
(1) 读取数据集	8
(2) 数据集图像的展示	9
(3) 数据集的划分	11
(4) 标签转化	11
(5) 训练函数	12
(6) 训练结果及其图像	13
(7) 验证集测试	14
(8) 验证集绘制图表	15
2、Python 实现区分数字 6 和其他数字	16
3、matlab 实现区分数字 6 和其他数字	16
4、matlab 实现区分数字 2 和数字 5	17
5、matlab 实现区分数字 1、2、5、7	17
6、matlab 实现区分数字 0 - 9	17
7、python 实现区分数字 0-9	17
8、遇到的问题、解决和一些细节	18
(1) 数据集读取与展示问题	18
(2) 数据集划分	18
(3) Matlab 预分配内存问题	19
(4) 求点积时可以借助 python 库函数降低时间复杂度	19
(5) 学习率选取问题	20
(6) 混淆矩阵绘制问题	20
9、对实验要求的回答	21
(1) 初始权重选取问题	21
(2) 结果评估（主要针对均方差值作为损失函数的结果）	21
三、实验总结	22
1、单层感知器性能评估	22
2、由单层感知器搭建多层感知器实现多元分类	22
3、使用 matlab 编程	22

一、实验原理

1、单层感知器

单层感知器(perceptron)是人工神经网络(Artificial neural network, ANN)的一种。利用单层感知器,我们能够自动识别和划分数据,例如在这个实验中我们对 MNIST 手写体数字进行了识别和划分。然而,需要注意的是,单层感知器主要适用于线性可分的问题,即能够将数据分为两类。在我们的实验中,我们尝试区分数字 6 和其他数字,以及区分数字 2 和数字 5

在使用感知器进行数据划分之前,我们必须对感知器进行训练。这个训练过程需要使用庞大的数据集。在训练过程中,也称为学习过程,我们首先为训练集的数据打上标签,然后根据数据集的特征不断更新权重,以更好地适应数据集的特性。

2、特征量

关于**特征量**,我理解的是能够完全且唯一表示一个数据的数值组合。举例来说,在 MNIST 手写体数据集中,每个数字图片的大小都是 28×28 , 总共有 784 个像素。这 784 个数值的组合足以完全且唯一地表示每张图片,被称为区分 MNIST 手写体的特征量。

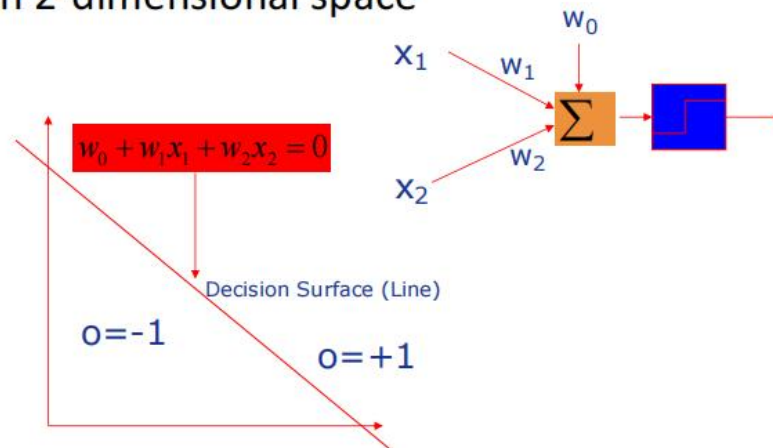
3、训练过程

为了更好地解释训练过程,我们以二维特征量为例。

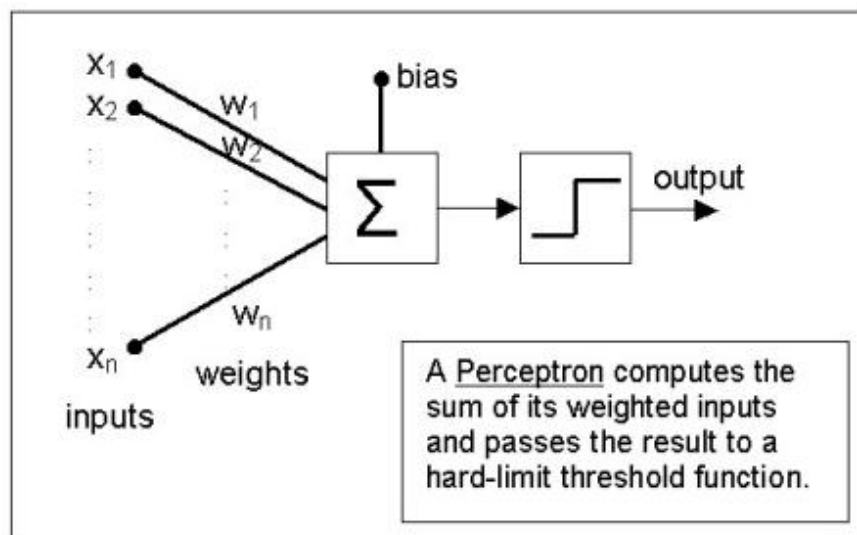
假设我们现在有足够大的数据集,每个数据都能够用两个特征量唯一而完全地表示,我们将这两个特征量表示为 x_1, x_2 。将这两个特征量以坐标的形式表示,得到一个点 (x_1, x_2) , 这个点所在的平面被称为**特征平面**(如果是多维特征量则称其为**特征空间**)。这样,数据集中的每个数据都可以在此特征平面内用一个点表示。我们的目标是将数据集中的所有数据进行识别并划分为两类,即在平面上对应的特征点划分为两类。

对于平面上的点划分成两类,我们可以通过一条曲线轻松地实现。这条曲线被称为**决策曲线**,多维特征空间则称为**决策平面**。曲线的一侧是一类,另一侧是另一类。为了判断某个点属于哪一类,我们可以将点的坐标带入曲线方程对应的函数,观察其结果的正负。然而,一般来说,复杂的曲线很难处理。因此,我们在这里只研究直线,如果一组数据可以被一条直线划分,就称其为**线性可分问题**。这个思路在多维特征空间同样适用。

- In 2-dimensional space



在二维平面中，表示一条直线其实需要两个参数，即 $y = kx + b$ 。但这里我们为了使每一个特征量的地位等价，我们用三个参数表示，方程形式为 $Ax + By + C = 0$ ，对应的函数是 $f(x_1, x_2) = w_2x_2 + w_1x_1 + w_0$ ，三个参数即其中的 w_2 、 w_1 和 w_0 。我们的目标是调整这三个参数，找到最合适的 w_2 、 w_1 和 w_0 ，使得数据集能够根据标签最大程度地被划分成两类。这里我们称 w_2 、 w_1 为**权重**， w_0 为**偏置**。



在寻找最适合的权重和偏置之前，我们必须首先获取一个重要的信息，即**标签**。标签代表了数据对应的实际类别。只有在获取了标签之后，我们才能通过不断调整权重，使得模型的输出 $f(x_1, x_2)$ 更加接近实际标签的值。对于二元分类，我们在图像上通常使用一条曲线将数据划分为两个侧区，曲线方程的函数值的正负决定了数据点所属的一侧。对于正负的区分，我们可以直接采用 $sign$ 符号函数，其中正值表示为 $+1$ ，负值表示为 -1 ，这个结果

和真实标签对比判断当前感知器判断结果是否正确。为了与之对应，我们通常将标签选择为 +1 和 -1，以适应二元分类的需求。

一切准备就绪后，可以开始对感知器进行训练。我们的目标是通过调整模型的权重，使得模型的输出 $f(x_1, x_2)$ 尽可能接近真实标签，或者说最小化输出 $f(x_1, x_2)$ 与真实标签的差值。这个差值的大小由损失函数（也称为代价函数）来衡量，损失函数的值越小，说明感知器的感知效果越好。由于感知器的感知效果只与权重和偏置有关，所以损失函数可以表示为 $W(w_0, w_1, w_2, \dots, w_n)$ ，其中 n 为特征量的维度。

为了表示错误程度，我们首先考虑了一种最简单的符号函数损失 $sign$ ，即

$$W(w_0, w_1, w_2, \dots, w_n) = sign((w_0 + \sum_{i=1}^n w_i x_i)) - d$$

，其中 d 表示数据的真实标签。这样的损失函数虽然计算简单且易于理解，但所有错误的样本不论错误程度大小都会面临相同的惩罚，大大降低了权重更新的准确性。且可能会面临梯度不连续、不可导等问题。所以不推荐使用。

1. Set the weights to small random values, e.g., in the range (-1, 1)

2. Present X_i and calculate

$$R = w_0 + \sum_{i=1}^n w_i x_i \quad o = sign(R) = \begin{cases} +1; & \text{if } R > 0 \\ -1, & \text{otherwise} \end{cases}$$

3. Update the weights

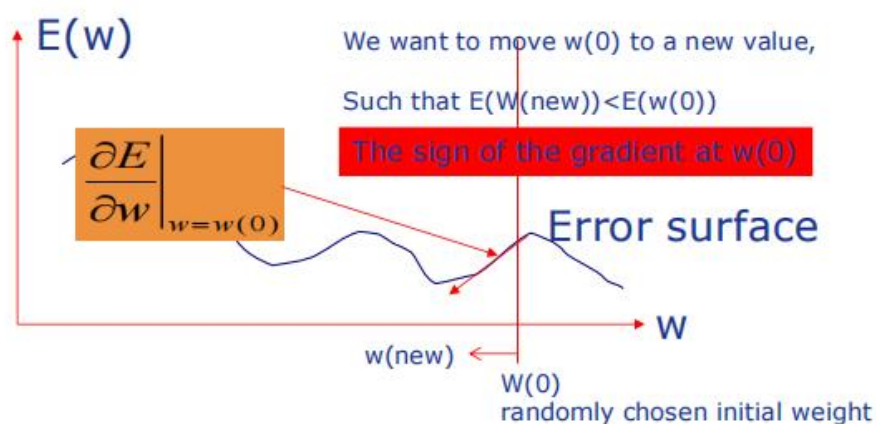
$$w_i \leftarrow w_i + \eta(d - o)x_i, i = 1, 2, \dots, n$$

$$0 < \eta < 1 \text{ is the training rate} \quad x_0 = 1 \text{ (constant)}$$

4. Repeat by going to step 2

为了更合理地进行权重更新，我们采用可求导的损失函数和梯度下降算法。常用的损失函数包括均方误差（MSE）、交叉熵损失、Hinge 损失、对数损失等。在这里，我们选择了均方误差作为损失函数，通过梯度下降来更新权重，从而根据数据的错误程度学习适当的权重，使得模型的输出尽可能接近实际标签。[1]

梯度下降的过程涉及到损失函数 $W(w_0, w_1, w_2, \dots, w_n)$ 大小的更新。以二维权重为例，可以在以 w_1 为横坐标， w_2 为纵坐标， W 为竖坐标的三维空间将其想象成在找到谷底最佳路径的思路。损失函数就像一个山坡，我们的目标是找到山坡最低的位置，即损失函数最小的地方。通过观察当前位置的坡度，即梯度，我们朝着梯度的反方向迈出一小步，这相当于更新了模型的位置。**学习率**则决定了迈步的大小，合适的学习率相当于选择适当的步子大小。我们不断重复这个过程，直到发现坡度足够平缓，即梯度很小，说明我们已经到达了山底，模型参数调整得差不多了。梯度下降是一种通过不断观察坡度并沿着最陡峭的方向调整位置的方法，最终找到让损失函数值最小的模型参数的过程。多维损失函数同理。



4、效果评估

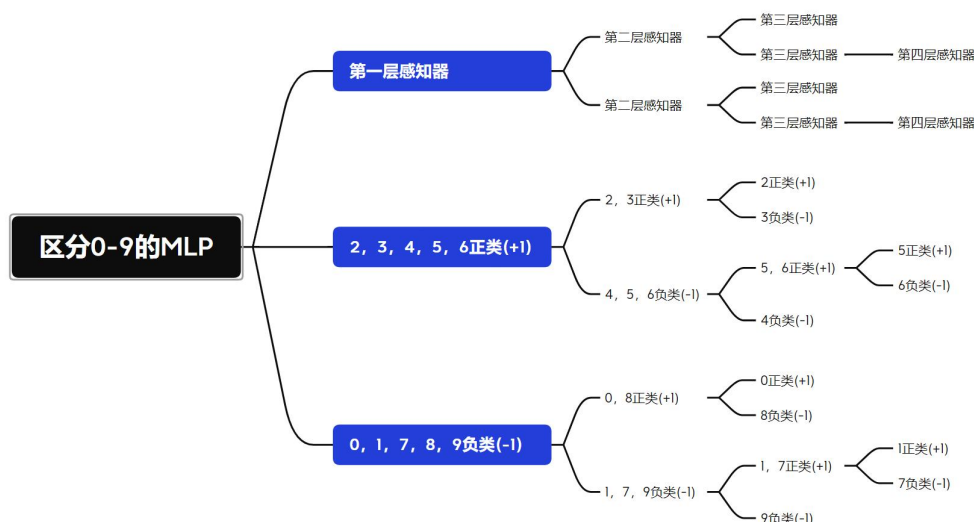
训练效果的展示可以通过绘制训练集错误率随时间变化的图表来实现。而对于预测集，了解到在二元分类中，主要使用 ROC 曲线和混淆矩阵进行评估。那么，ROC 曲线是一种衡量模型分类性能的图形工具，它显示了真正例率与假正例率之间的关系。而混淆矩阵则是一种表格，用于比较模型的实际预测结果与实际标签的差异。

混淆矩阵		真实值	
		Positive	Negative
预测值	Positive	TP	FP
	Negative	FN	TN

关于真正例率和假正例率的解释。真正例率（True Positive Rate, TPR）是指模型正确识别实际为正例的比例，计算方式为 $TP/(TP + FN)$ 。而假正例率（False Positive Rate, FPR）表示模型错误将实际为负例的样本识别为正例的比例，计算方式为 $FP/(FP + TN)$ 。在 ROC 曲线上，横轴是 FPR，纵轴是 TPR。ROC 曲线越靠近左上角，说明模型性能越好。混淆矩阵提供了更详细的分类结果，其中 TP 是真正例数量，FP 是假正例数量，FN 是假负例数量，TN 是真负例数量。综合利用这些指标和图表有助于全面评估分类模型的性能。

5、多元分类

单层感知器的原理和评估方式已经解释明了，然而需要注意的是，这种感知器仅适用于二元分类问题。如果我们的任务是实现多元分类，可以通过简单地构建多个单层感知器来实现，以下是我的一个实现思路。



二、实验过程

1、实现过程

(1) 读取数据集

在下载和使用数据集之前，首先应该了解数据集的结构，可通过以下网址获取：<http://yann.lecun.com/exdb/mnist/>。需要注意的是，在训练图像集和测试图像集中，前 16 个字节包含了 Magic Number 和 Number of Images，而并非实际的图像数据。同样，在训练标签集和测试标签集中，前 8 个字节包含了 Magic Number，而不是实际的标签数据。因此，在处理数据时需要将这些无关的部分舍去。详细信息可参考下图：

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

TEST SET LABEL FILE (t10k-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	10000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

TEST SET IMAGE FILE (t10k-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	10000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

Python 代码:

```
def load_mnist_images(filename):
    with open(filename, 'rb') as f:
        data = np.fromfile(f, dtype=np.uint8, count=-1)
        return data[16:].reshape(-1, 28*28) / 255.0

def load_mnist_labels(filename):
    with open(filename, 'rb') as f:
        data = np.fromfile(f, dtype=np.uint8, count=-1)
        return data[8:]
```


Matlab:

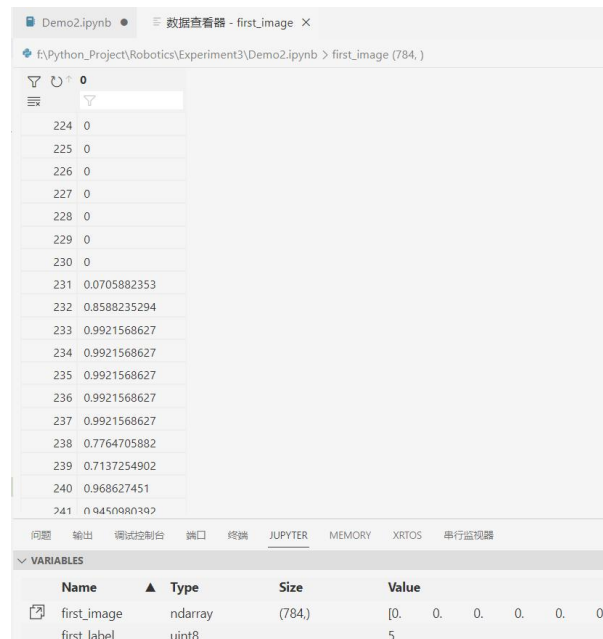
```
adaline.m x adaline_train.m x calculateSampleCount.m x task1.m x
1 function data = load_mnist_images(filename)
2 % 数据集图像加载函数
3 fid = fopen(filename, 'rb');
4 data = fread(fid, 'uint8');
5 fclose(fid);
6 data = data(17:end) / 255.0;
7 data = reshape(data, 28*28, []).';
8 end
```

注：此处读取时转置图像是没必要的

```
+3 task1_SLP.m x task2_MLP.m x task3.m x load_mnist_images.m
1 function data = load_mnist_labels(filename)
2 % 数据集标签加载函数
3 fid = fopen(filename, 'rb');
4 data = fread(fid, 'uint8');
5 fclose(fid);
6 data = data(9:end);
7 end
```

(2) 数据集图像的展示

可以在 VS Code 的变量查看器中观察到每个图像读取出来都是一个一维数组，具体如下图所示。在展示之前，需要将长度为 784 的一维数组转化为 28*28 的二维数组。这一步骤也可以直接集成到读取函数中。

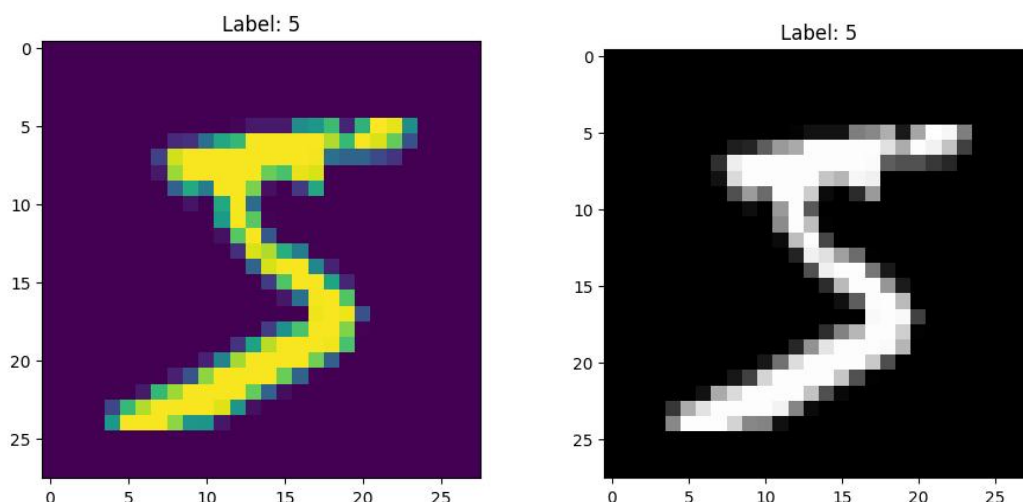


The screenshot shows the VS Code variable viewer for a Jupyter notebook. The variable 'first_image' is displayed as a 1D array of 784 elements. The first few elements are 0, and then there are some non-zero values. The variable 'first_label' is also shown as a 1D array with a single element, 5.

Name	Type	Size	Value
first_image	ndarray	(784,)	[0. 0. 0. 0. 0. 0. ...]
first_label	uint8	5	5

灰度图像通常是指每个像素只有一个灰度值，而不是多个颜色通道的彩色图像。对于 MNIST 数据集，每个图像都是**单通道的灰度图像**。使用灰度颜色映射 (cmap='gray') 在显示灰度图像时是合适的。这是因为灰度图像的每个像素值都是在黑色 (0) 和白色 (255 或 1.0) 之间的灰度级别。通过使用灰度颜色映射，可以在显示图像时更容易识别不同灰度级别的区别。

如果选择使用其他颜色映射，图像可能会以伪彩色显示，这可能不是你想要的效果，特别是对于单通道的灰度图像。具体显示效果如下图。



python:

```
# 将图像数据还原为28x28的形状
image_resaped = first_image.reshape(28, 28)

plt.imshow(image_resaped, cmap='gray') # 使用灰度颜色映射
plt.title(f"Label: {first_label}")
plt.show()
```

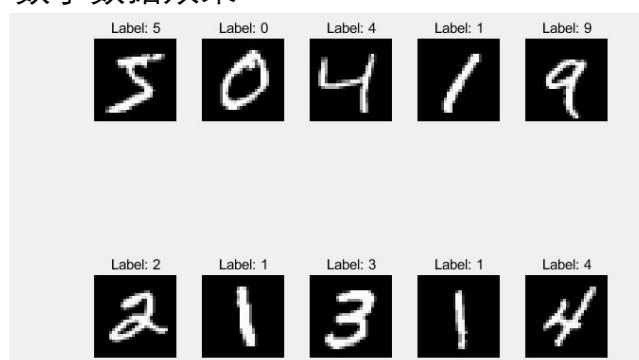
在上面的代码中，`plt.imshow(image_resaped, cmap='gray')` 用于使用灰度颜色映射显示图像。这确保了图像以黑白形式呈现，以反映灰度级别。

Matlab:

```
data_load

for i = 1:10
    subplot(2, 5, i);
    img = reshape(train_images_array(i, :), [28, 28]); % 将图像重新转换为 28x28 的形状
    imshow(img, 'InitialMagnification', 'fit'); % 转置图像矩阵并设置放大方式为适应窗口
    colormap(gray); % 灰度图
    title(['Label: ', num2str(train_labels_array(i))]);
end
```

Matlab 展示前十个数字数据效果:



(3) 数据集的划分

MNIST 官方网站已提供了训练集和测试集，实际上无需进行数据集划分。然而，本实验要求对训练数据集的 60000 个数据进行划分，分为自定义的训练集和测试集。CourseWork3 实验的具体要求如下：

guidelines:

- Split the data into a training set and a testing set.

实现这一要求的方法相对简单，只需对数据集进行随机打乱，然后按照特定比例划分为两部分。具体而言，代码通过对打乱后的数据集从前往后进行划分来实现。为了确保结果的~~可复现性~~，我在函数的输入变量中使用了 random_state 随机数种子。以下是代码的具体实现。

```
additional_questions.py x adline.py x
35 def split_dataset(x, y, test_size=0.2, random_state=None):
36     # 获取数据集的长度
37     data_size = len(x)
38
39     # 根据比例计算验证集的大小
40     val_size = int(test_size * data_size)
41
42     # 设置随机种子
43     if random_state is not None:
44         np.random.seed(random_state)
45
46     # 随机打乱数据集
47     indices = np.random.permutation(data_size)
48
49     # 划分数据集
50     val_indices = indices[:val_size]
51     train_indices = indices[val_size:]
52
53     # 获取划分后的数据
54     x_train, x_val = x[train_indices], x[val_indices]
55     y_train, y_val = y[train_indices], y[val_indices]
56
57     return x_train, x_val, y_train, y_val
```

(4) 标签转化

原始数据集的标签对应着数据所代表的数字，即 0-9。然而，在进行二元分类时，将真实数字直接作为标签效果并不理想。在二元分类中，我们通常通过在图像上绘制一条曲线来将数据划分为两个侧区，曲线方程的函数值的正负决定了数据点所属的一侧。为了在感知器中进行正负的区分，我们直接采用符号函数，其中正值表示为+1，负值表示为-1。这个结果与真实标签进行比较，从而判断当前感知器的判断结果是否正确。为了与这种方式相匹配，我们通常将标签选定为+1 和-1，以满足二元分类的需求。以下是针对不同实验要求进行的标签转化。

区分数字 6 和其他数字的标签转化。

```
170 train_label_binary = np.where(train_labels_array == 6, 1, -1)
```

区分数字 2 和数字 5 的标签转化。

```
4  train_label = (train_labels_array == 6) * 2 - 1;
5  test_label = (test_labels_array == 6) * 2 - 1;
```

区分数字 1、2、5、7 的标签转化。

% 提取数字 '2'、'5'、'1' 和 '7' 的样本

```
indices_2571 = find(train_labels_array == 2 | train_labels_array == 5 | train_labels_array == 1 | train_labels_array ==
```

% 将 '2' 和 '5' 视为正类 (+1), '1' 和 '7' 视为负类 (-1)

```
labels_parent = (labels == 2 | labels == 5) * 2 - 1;
```

区分数字 0-9, 不同层不同感知器对应的标签转化。

% 将 '2.3.4.5.6' 视为正类 (+1), '0.1.7.8.9' 视为负类 (-1)

```
labels_layer1 = (train_labels == 2 | ...  
                train_labels == 3 | ...  
                train_labels == 4 | ...  
                train_labels == 5 | ...  
                train_labels == 6) * 2 - 1;
```

% 将 '2.3' 视为正类 (+1), '4.5.6' 视为负类 (-1)

```
binary_23456 = (labels_23456 == 2 | labels_23456 == 3) * 2 - 1;
```

% 将 '0.8' 视为正类 (+1), '1.7.9' 视为负类 (-1)

```
binary_01789 = (labels_01789 == 0 | labels_01789 == 8) * 2 - 1;
```

% 将 '2' 视为正类 (+1), '3' 视为负类 (-1)

```
binary_23 = (labels_23 == 2) * 2 - 1;
```

% 将 '5.6' 视为正类 (+1), '4' 视为负类 (-1)

```
binary_456 = (labels_456 == 5 | labels_456 == 6) * 2 - 1;
```

% 将 '0' 视为正类 (+1), '8' 视为负类 (-1)

```
binary_08 = (labels_08 == 0) * 2 - 1;
```

% 将 '1.7' 视为正类 (+1), '9' 视为负类 (-1)

```
binary_179 = (labels_179 == 1 | labels_179 == 7) * 2 - 1;
```

% 将 '5' 视为正类 (+1), '6' 视为负类 (-1)

```
binary_56 = (labels_56 == 5) * 2 - 1;
```

% 将 '1' 视为正类 (+1), '7' 视为负类 (-1)

```
binary_17 = (labels_17 == 1) * 2 - 1;
```

(5) 训练函数

编写训练函数主要基于训练原理部分的理论基础。我创建了两个训练函数, 其中一个利用符号函数作为损失函数。以下是其原理和代码示例:

Perceptron – Training Algorithm

The Procedure is as follows

1. Set the weights to small random values, e.g., in the range $(-1, 1)$

2. Present X_i and calculate

$$R = w_0 + \sum_{j=1}^n w_j x_j, \quad o = \text{sig}n(R) = \begin{cases} +1, & \text{if } R > 0 \\ -1, & \text{otherwise} \end{cases}$$

3. Update the weights

$$w_i \leftarrow w_i + \eta(d - o)x_i, \quad i = 1, 2, \dots, n$$

$$0 < \eta < 1 \quad \text{is the training rate} \quad x_0 = 1 \text{ (constant)}$$

4. Repeat by going to step 2

```
47 # 训练函数  
48 def train(x, y, weights_init_val=None, learning_rate=0.01, epochs=100, stopping_threshold=0.018):  
49     input_variables_size = len(x[0])  
50     weight_i, weight_0 = initialize_weights(input_variables_size, weights_init_val)  
51  
52     errors = []  
53  
54     for epoch in range(epochs):  
55         error = 0  
56         for xi, target in zip(x, y):  
57             dot_product = sum(xi_j * weight_i_j for xi_j, weight_i_j in zip(xi, weight_i))  
58             prediction = np.sign(dot_product + weight_0)  
59             update = learning_rate * (target - prediction)  
60             weight_i += update * xi  
61             weight_0 += update  
62             error += int(update.item() != 0.0)  
63  
64         errors.append(error / len(y)) # 训练集的样本个数  
65  
66         if error / len(y) < stopping_threshold:  
67             break  
68  
69     return errors, weight_i, weight_0
```


Gradient of ADALINE Error Functions

$$E(W) = \frac{1}{2} \sum_{k=1}^K (d(k) - o(k))^2$$

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial E}{\partial w_i} \left(\frac{1}{2} \sum_{k=1}^K (d(k) - o(k))^2 \right) \\ &= \frac{1}{2} \sum_{k=1}^K \left(\frac{\partial E}{\partial w_i} (d(k) - o(k))^2 \right) \\ &= \frac{1}{2} \sum_{k=1}^K \left(2(d(k) - o(k)) \frac{\partial E}{\partial w_i} (d(k) - o(k)) \right) \\ &= \sum_{k=1}^K \left((d(k) - o(k)) \frac{\partial E}{\partial w_i} \left(d(k) - w_0 - \sum_{j=1}^n w_j x_j(k) \right) \right) \\ &= \sum_{k=1}^K ((d(k) - o(k))(-x_i(k))) \\ &= - \sum_{k=1}^K (d(k) - o(k)) x_i(k) \end{aligned}$$

```
71 def adaline_train(x, y, weights_init_val=None, learning_rate=0.01, epochs=100, stopping_threshold=0.010):
72     input_variables_size = len(x[0])
73     weight_i, weight_0 = initialize_weights(input_variables_size, weights_init_val)
74
75     errors = []
76
77     for epoch in range(epochs):
78         error = 0
79         for xi, target in zip(x, y):
80             dot_product = sum(xi_j * weight_i_j for xi_j, weight_i_j in zip(xi, weight_i))
81             prediction = dot_product + weight_0
82             update = learning_rate * (target - prediction)
83             weight_i += update * xi
84             weight_0 += update
85             error += (target - prediction) ** 2 # 使用平方损失函数
86
87     mean_squared_error = error / len(y)
88     errors.append(mean_squared_error)
89
90     if mean_squared_error < stopping_threshold:
91         break
92
93     return errors, weight_i, weight_0
```

(6) 训练结果及其图像

绘制训练图像相对简单，只需利用 matplotlib.pyplot 的绘图函数即可实现。需要注意的是，在预先分配内存时，如果固定了 error 数组的长度，那么 stopping threshold 将不会发挥作用。具体来说，即绘制测试集图像时，横坐标的长度不会随着 stopping threshold 的变化而变化。在训练截止时，未训练的迭代次数的 error 值都为 0，这些值仍然会被绘制在图像中，这样就不利于观察 stopping threshold 的效果。

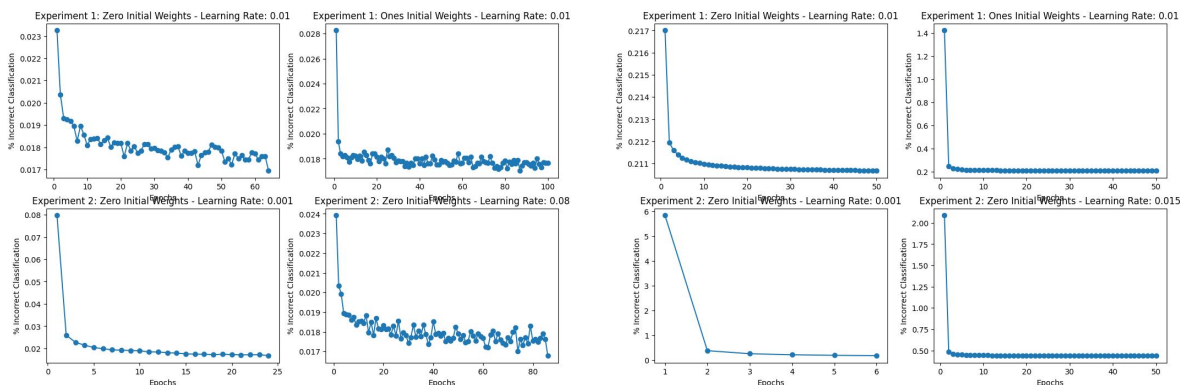
错误写法：`train_errors = np.zeros(epochs)` # 预分配内存

正确写法：`errors = []`

关于 stopping threshold 的确定，我首先将其设定为一个极小的值，然后绘制图像，观察 error 大致的小值。最后，将 stopping threshold（截止阈值）设置为这个值即可。

```
# 绘制训练集的错误率随时间变化的图表
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.plot(range(1, len(errors1) + 1), errors1, marker='o')
plt.xlabel('Epochs')
plt.ylabel('% Incorrect Classification')
plt.title('Experiment 1: Zero Initial Weights - Learning Rate: 0.01')
```

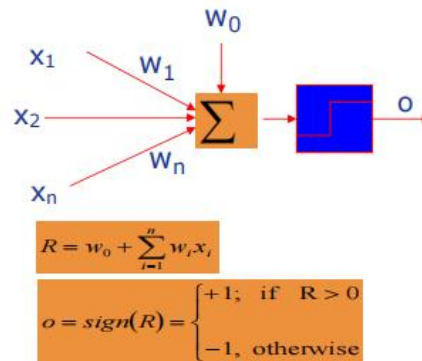
符号函数作为损失函数（截止阈值为 0.017）均方误差作为损失函数（截止阈值为 0.18）



(7) 验证集测试

验证集的测试过程涉及将感知器应用于测试集的数据（即 784 个特征量），计算得到的结果并取其符号。

从图像的角度来看，感知器可以被视为一个决策平面，其中将数据特征点的坐标带入相应的函数后，根据函数值的正负可以区分该特征点位于决策平面的哪一侧，从而实现二元分类。



对应代码：

```
# 使用训练好的参数进行预测
def predict_with_parameters(x, weight_i, weight_0):
    predictions = np.sign(np.dot(x, weight_i) + weight_0)
    return predictions

# 计算错误率
def calculate_error_rate(predictions, actual_labels):
    incorrect_samples = np.sum(predictions != actual_labels)
    error_rate = incorrect_samples / len(actual_labels)
    return error_rate
```

单层感知器的结果：

```
# 在测试集上验证不同实验的错误率
predict_label_1 = predict_with_parameters(test_images_array, final_weight_i_1, final_weight_0_1)
predict_label_2 = predict_with_parameters(test_images_array, final_weight_i_2, final_weight_0_2)
predict_label_3 = predict_with_parameters(test_images_array, final_weight_i_3, final_weight_0_3)
predict_label_4 = predict_with_parameters(test_images_array, final_weight_i_4, final_weight_0_4)
error_rate_1 = calculate_error_rate(predict_label_1, test_label)
error_rate_2 = calculate_error_rate(predict_label_2, test_label)
error_rate_3 = calculate_error_rate(predict_label_3, test_label)
error_rate_4 = calculate_error_rate(predict_label_4, test_label)

print("Experiment 1 Error Rate (Zero Initial Weights):", error_rate_1)
print("Experiment 1 Error Rate (Ones Initial Weights):", error_rate_2)
print("Experiment 2 Error Rate (Learning Rate: 0.001):", error_rate_3)
print("Experiment 2 Error Rate (Learning Rate: 0.08):", error_rate_4)
```

✓ 0.0s

```
Experiment 1 Error Rate (Zero Initial Weights): 0.0185
Experiment 1 Error Rate (Ones Initial Weights): 0.0164
Experiment 2 Error Rate (Learning Rate: 0.001): 0.0192
Experiment 2 Error Rate (Learning Rate: 0.08): 0.0168
```


(8) 验证集绘制图表

根据实验原理部分的了解，评估二元分类感知器的性能优劣主要通过混淆矩阵、ROC 曲线等指标。在本次实验中，我们选择使用混淆矩阵进行性能评价。

* 真正例 (True Positive, TP) : 模型正确地预测为正例的数量。
* 真负例 (True Negative, TN) : 模型正确地预测为负例的数量。
* 假正例 (False Positive, FP) : 模型错误地预测为正例的数量。
* 假负例 (False Negative, FN) : 模型错误地预测为负例的数量。

混淆矩阵的一般形式如下:

$$\begin{array}{cc} & \begin{matrix} TN & FP \end{matrix} \\ \begin{matrix} FN & TP \end{matrix} & \end{array}$$

根据混淆矩阵，我们可以计算一些评估指标，如准确率 (Accuracy)、精确率 (Precision)、召回率 (Recall) 和 F1 分数等。

* 准确率 (Accuracy) : $\frac{TP+TN}{TP+TN+FP+FN}$
* 精确率 (Precision) : $\frac{TP}{TP+FP}$
* 召回率 (Recall) : $\frac{TP}{TP+FN}$
* F1 分数 (F1 Score) : $2 \times \frac{Precision \times Recall}{Precision + Recall}$

定义函数:

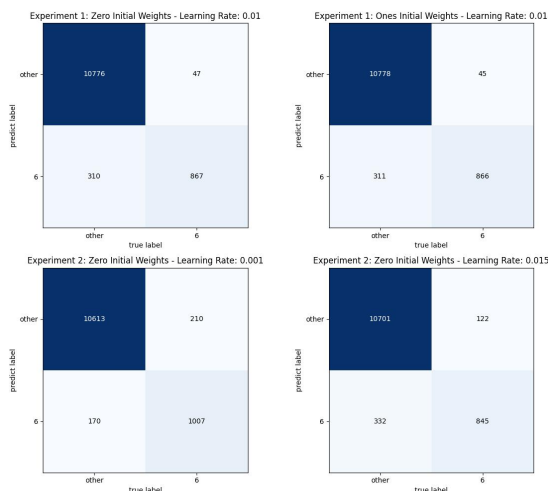
```
# 计算混淆矩阵
def calculate_confusion_matrix(verify, predict):
    true_positive = np.sum((verify == 1) & (predict == 1))
    false_positive = np.sum((verify == -1) & (predict == 1))
    true_negative = np.sum((verify == -1) & (predict == -1))
    false_negative = np.sum((verify == 1) & (predict == -1))

    return np.array([[true_negative, false_positive], [false_negative, true_positive]])

# 绘制混淆矩阵
def plot_confusion_matrix(confusion_matrix, title):
    plt.imshow(confusion_matrix, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(title)
    # plt.colorbar()
    # 这个 colorbar 可以在图像旁边显示颜色对应的数值
    classes = ['other', '6']
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)
    plt.xlabel('true label')
    plt.ylabel('predict label')

    # 这段代码是在绘制的矩阵中添加数字标签
    for i in range(len(classes)):
        for j in range(len(classes)):
            # 根据颜色深浅选择标签颜色
            text_color = 'white' if confusion_matrix[i, j] > confusion_matrix.max() / 2 else 'black'
            plt.text(j, i, str(confusion_matrix[i, j]), ha='center', va='center', color=text_color)
```

以均方误差损失函数为例:



2、Python 实现区分数字 6 和其他数字

对于这个任务有两种实现方式，一个是利用符号函数作为损失函数，另一种是利用均方差作为损失函数，下面是相关函数的介绍。

```
# 读取数据集图像
def load_mnist_images(filename):
# 读取数据集标签
def load_mnist_labels(filename):
# 划分数据集，得到训练集与测试集
def split_dataset(x, y, test_size=0.2, random_state=None):
# 初始化权重（实验要求）
def initialize_weights(input_variables_size, weights_init_val=None):
# 使用符号函数作为损失函数进行训练
def train(x, y, weights_init_val=None, learning_rate=0.01, epochs=100, stopping_threshold=0.018):
# 均方差作为损失函数进行训练
def adaline_train(x, y, weights_init_val, learning_rate = 0.01, epochs = 50, stopping_threshold = 0.01):
# 用训练得到的函数对测试集进行预测
def predict_with_parameters(x, weight_i, weight_0):
# 计算预测的争取率
def calculate_error_rate(predictions, actual_labels):
# 计算 roc 参数正真比例与假正比例
def calculate_roc_rate(verify, predict):
# 计算混淆矩阵
def calculate_confusion_matrix(verify, predict):
# 绘制混淆矩阵
def plot_confusion_matrix(confusion_matrix, title):
```

3、matlab 实现区分数字 6 和其他数字

函数文件：

```
% 读取数据集的图像
load_mnist_images.m
% 读取数据集的标签
load_mnist_labels.m
% 初始化感知器的权重
initialize_weights.m
% 符号函数作为损失函数
single_perceptron.m
% 均方差值作为损失函数
adaline_train.m
```

脚本文件：

```
% 展示前十张图片，验证数据读取
show_imshow.m
% 加载数据集图像及标签
data_load.m
% 符号函数作为损失函数的训练
```

Single_perceptron_train.m
% 均方差值作为损失函数的训练
adaline.m

4、 matlab 实现区分数字 2 和数字 5

函数文件：

% 计算数据集中各标签的数量（未在脚本文件引用，可直接调用调试）

calculateSampleCount.m

% 展示特定数字的图像（未在脚本文件引用，可直接调用调试）

displayDigitSamples.m

% 展示被错误分类的图像

displayMisclassifiedImages.m

脚本文件：

task1_SLP.m

5、 matlab 实现区分数字 1、2、5、7

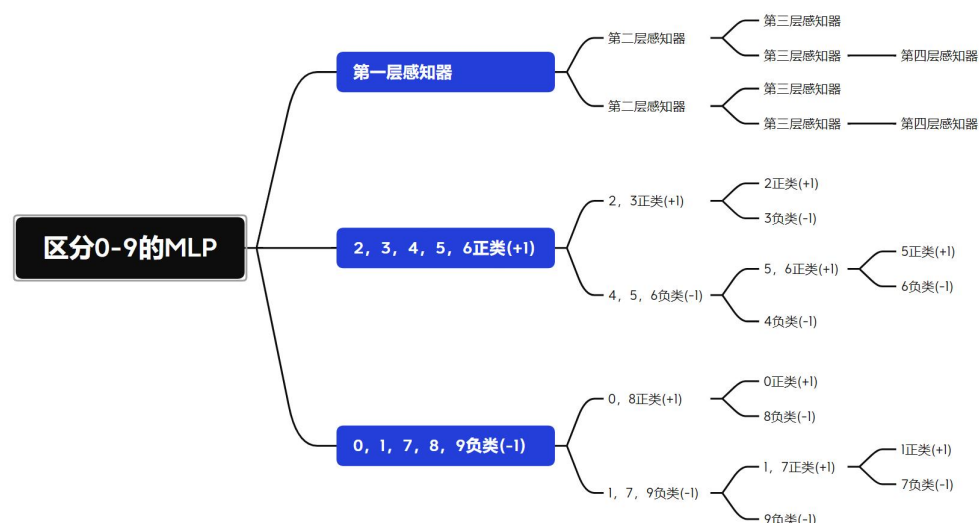
脚本文件：

task2_MLP.m

6、 matlab 实现区分数字 0 - 9

脚本文件：

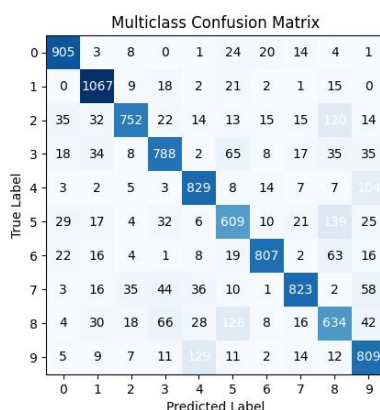
task3.m



7、 python 实现区分数字 0-9

实际上是 matlab 实现区分数字 0-9 直接转化成 python 的代码。实现思路如上。

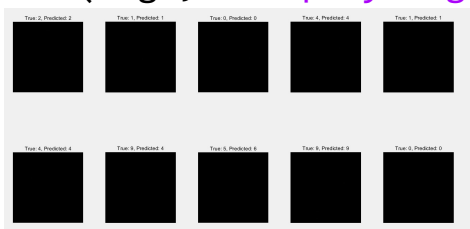
增加测试集图像的绘制。仍然使用混淆矩阵。



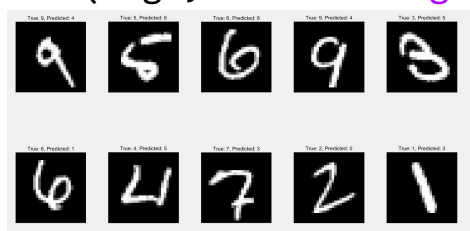
8、遇到的问题、解决和一些细节

(1) 数据集读取与展示问题

(python 具体见实验报告“实现过程”部分) 关于 matlab 展示, 这样是错误的: `imshow(img', 'DisplayRange', [0, 255]);`



这样是正确的: `imshow(img', 'InitialMagnification', 'fit');`



(2) 数据集划分

我添加了随机数种子, 保证数据集划分的可复现性。

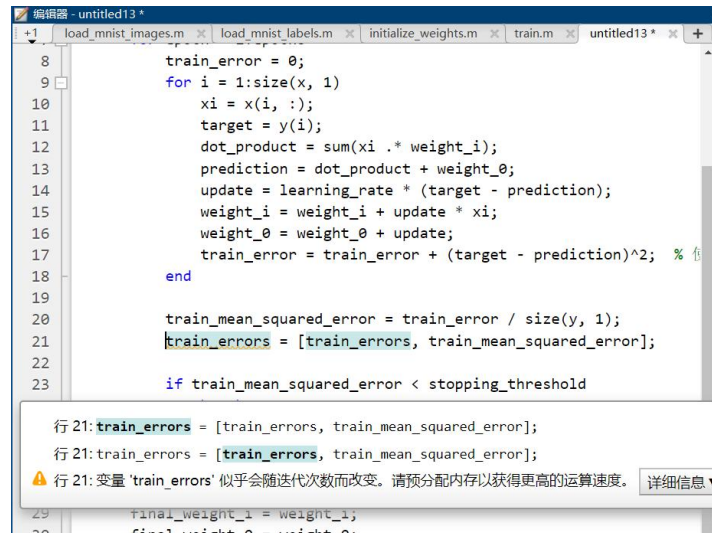
设置随机种子会影响后续对数据集的随机打乱。代码提供随机种子 `random_state`, 通过 `np.random.seed(random_state)` 这一行代码, 确保每次在执行 `np.random.permutation(data_size)` 这一步时, 得到的随机排列结果将是确定性的。如果没有设置随机种子, 那么每次运行代码时都将使用不同的随机种子, 导致不同的数据集排列结果。

```
# 设置随机种子
if random_state is not None:
    np.random.seed(random_state)

# 随机打乱数据集
indices = np.random.permutation(data_size)
```

(3) Matlab 预分配内存问题

未在函数定义里面对 `train_errors` 的长度进行定义，会出现以下警告。虽然程序依然可以运行，但会导致运算速度下降。



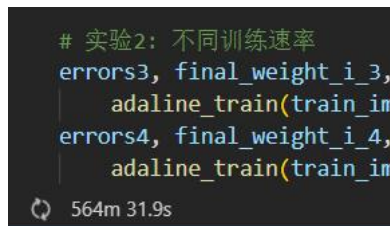
修改为以下表述，在调用 `train_errors` 之前对其数组长度进行说明。可以提高程序运行速度。（但此方法会带来弊端，即 `train_errors` 的长度不会随着 `stopping_threshold` 的改变而改变，导致训练错误变化曲线的横坐标长度不会变化，很难直观感受 `stopping_threshold` 的作用）

```
train_errors = zeros(1, epochs); % 预分配内存
```

Python 中可以在截断时加入这句代码解决此问题。

```
if train_mean_squared_error < stopping_threshold:
    train_errors = train_errors[:epoch + 1]
    # 截取数组长度，以适应变化的 stopping_threshold
    break
```

(4) 求点积时可以借助 python 库函数降低时间复杂度
原代码效果：



修改：

```
# dot_product = sum(xi_j * weight_i_j for xi_j, weight_i_j in zip(xi, weight_i))
# 运行时间太长了，我猜测是此处循环的问题，改用以下

dot_product = np.sum(xi * weight_i)
```

(5) 学习率选取问题

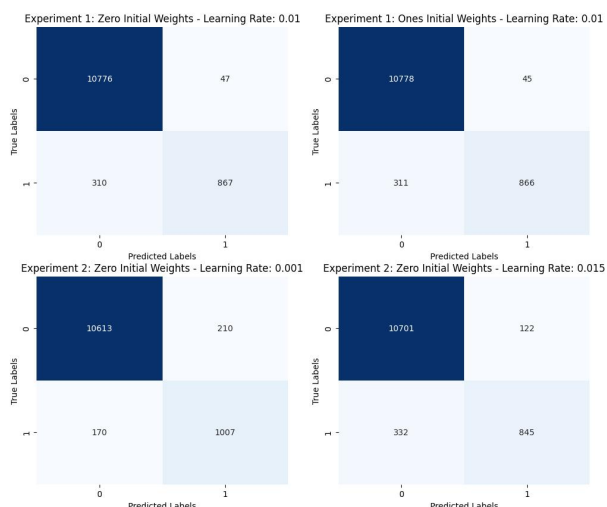
学习率的选取要符合实际。比如学习率为 0.08 时会导致数值过大而报错：

```
errors4, final_weight_i_4, final_weight_0_4 = \
    | adaline_train(train_images_array, train_label, weights_init_val=1, learning_rate=0.08, stopping_threshold=0.018)
37m 3.4s

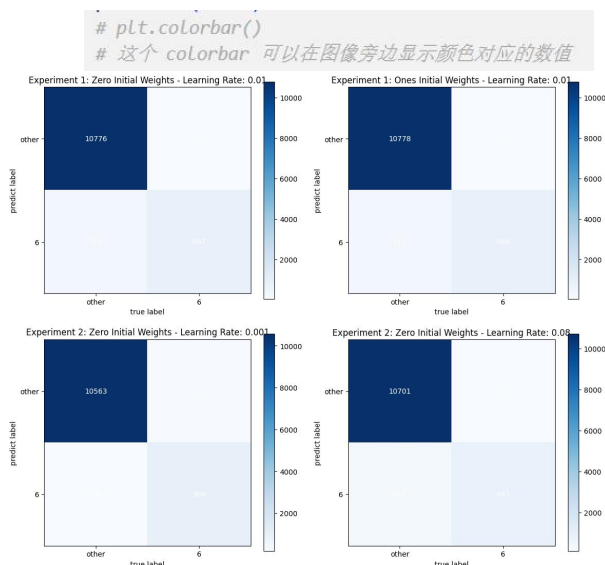
C:\Users\HP\AppData\Local\Temp\ipykernel_17928\3940178182.py:65: RuntimeWarning: overflow encountered in square
error += (target - prediction) ** 2 # 使用平方损失函数
C:\Users\HP\AppData\Local\Temp\ipykernel_17928\3940178182.py:60: RuntimeWarning: overflow encountered in scalar add
dot_product = sum(xi_j * weight_i_j for xi_j, weight_i_j in zip(xi, weight_i))
C:\Users\HP\AppData\Local\Temp\ipykernel_17928\3940178182.py:63: RuntimeWarning: invalid value encountered in multiply
weight_i += update * xi
C:\Users\HP\AppData\Local\Temp\ipykernel_17928\3940178182.py:60: RuntimeWarning: invalid value encountered in scalar multiply
dot_product = sum(xi_j * weight_i_j for xi_j, weight_i_j in zip(xi, weight_i))
```

(6) 混淆矩阵绘制问题

实际上，Python 已经提供了相关库函数用于绘制混淆矩阵，为了更深入地理解这一方法，我重新编写了一个混淆矩阵绘制的函数。首先，让我们来看一下使用 Python 库函数绘制混淆矩阵的方法：

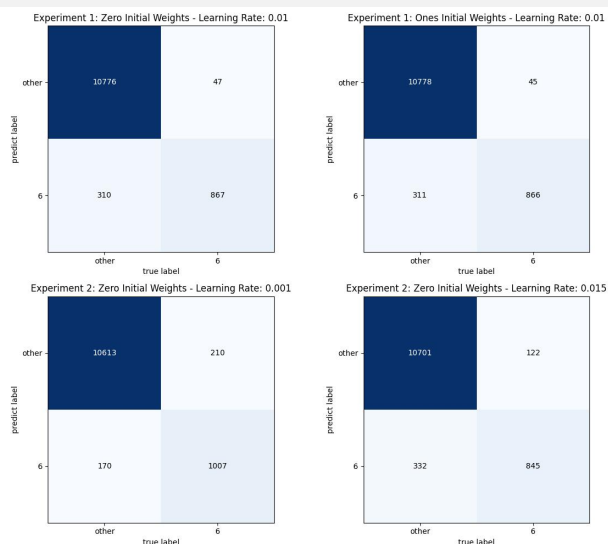


接下来是我第一个混淆矩阵绘制函数，我还添加了颜色条（colorbar）的绘制：



此外，有一个小细节是字体颜色的问题。我尝试使用最大值的一半和数据的平均值来区分颜色，发现使用最大值的一半更为合理。

```
# 这段代码是在绘制的矩阵中添加数字标签
for i in range(len(classes)):
    for j in range(len(classes)):
        # 根据颜色深浅选择标签颜色
        text_color = 'white' if confusion_matrix[i, j] > confusion_matrix.max() / 2 else 'black'
        text_color = 'white' if confusion_matrix[i, j] > confusion_matrix.mean() else 'black'
        plt.text(j, i, str(confusion_matrix[i, j]), ha='center', va='center', color=text_color)
```

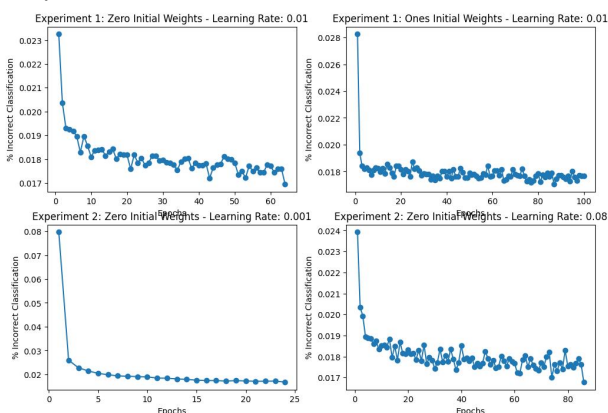


9、对实验要求的回答

(1) 初始权重选取问题

在我看来，当训练次数足够多的情况下，初始权重对结果的影响并不是特别显著。这是因为在我们的实验中，权重和损失函数之间存在唯一的局部最低点。根据梯度下降规则，通过足够多次的训练，模型将会收敛到这个独特的局部最低点，也就是最适合的权重组合。

(2) 结果评估（主要针对均方差值作为损失函数的结果）



这张图表呈现了四个实验中错误率随时间变化的情况。实验 1 主要探讨了不同初始权重对性能的影响，而实验 2 则关注了不同学习率的效果。从图

表中可以明显看出，随着时间的推移，所有实验的错误率都呈下降趋势。

在实验 1 中，观察到无论初始权重是全为 0 还是全为 1，误差率都迅速下降到一个较低水平并保持稳定。这一结果验证了初始权重选取的一部分猜想：在我们的实验中，权重和损失函数之间存在唯一的局部最低点。根据梯度下降规则，通过足够多次的训练，模型将会收敛到这个独特的局部最低点，即最适合的权重组合。至于未发生截断的情况，我推测可能是由于实验 1 中两个感知器的学习率较高，导致在截止阈值附近不断振荡。

实验 2 的结果显示，学习率较小的感知器能够在较少的训练次数内达到截止阈值，并且在该阈值附近不会出现振荡。另外，振荡的幅度较大，可能导致与阈值之间的差值较大。

三、实验总结

1、单层感知器性能评估

这次实验的总体结论表明，对感知器模型进行训练时初始权重和学习率的选择对性能产生了显著影响。实验 1 揭示了不同初始权重对模型性能的影响，指出无论是全为 0 还是全为 1 的初始权重，模型的错误率都在训练后迅速下降并趋于稳定。这结果验证了实验中权重和损失函数之间存在唯一的局部最低点，通过梯度下降规则能够有效地收敛到最适合的权重组合。

实验 2 则突出了学习率在训练过程中的关键作用。小学习率有助于在较少的训练次数内达到截止阈值，且在阈值附近不会引发振荡。相反，较大的学习率可能导致在截止阈值附近频繁振荡，甚至无法收敛。这强调了在实际应用中选择适当的学习率以确保训练的稳定性和有效性的重要性。

不过，实验 1 也提出了一个潜在的问题，即在某些情况下，高学习率可能导致振荡问题，阻碍了模型的收敛。因此，在实际应用中，需要仔细权衡和调整初始权重和学习率，以取得最佳性能。

性能评估还可以借助混淆矩阵等可视化的方法。

2、由单层感知器搭建多层感知器实现多元分类

通过结合多个层次标签和单层感知器，我们可以先把 0 到 9 这十个数字逐步分成不同的集合，然后取它们的唯一交集，从而实现多元分类。这就是构建多层感知器（MLP）的方法。

3、使用 matlab 编程

逐渐熟悉了 Matlab 的语法和功能，学会借助 Matlab 处理数据和进行编程。

【参考文献】

[1]模型评价之混淆矩阵、ROC 曲线与 AUC - 知乎 (zhihu.com)
<https://zhuanlan.zhihu.com/p/390518914>