人工智能基础 实验二

PB18151866 龚小航

【实验目的】

本次实验包含传统机器学习与深度学习两部分。通过本次实验初步了解机器学习的基本方法。

【实验内容】

在传统机器学习部分实现线性分类算法、朴素贝叶斯分类器以及 SVM 算法；在深度学习部分实现手写感知机模型并进行反向传播与复现 MLP-Mixer。

传统机器学习部分需要根据鲍鱼的物理测量属性预测鲍鱼的年龄，label 已经经过处理分为 1，2，3 三类，用各种算法在测试集上预测鲍鱼的年龄组别，并分析预测准确度。

深度学习部分首先需要实现四层感知机模型，神经元个数为 5，4，4，3，激活函数选为 sigmoid。在此基础上实现误差逆传播算法与梯度下降算法，使用随机输入与随机标签。而 MLP-Mixer 部分需要复现 MLP-Mixer 模型，实现手写数字图像分类识别。

【实验过程与运行结果】

一 、实现线性分类器

对鲍鱼数据集进行线性分类。鲍鱼数据集对一条训练数据 $(\vec{x}, xLabel)$ 共有 8 个属性和一个类别标签。线性分类器需要找到一个 $\vec{\omega}$ 使得所有训练数据到分割线的最小二乘距离最小。引入 L2 规范化项之后，线性分类问题等价于求解以下优化问题:

$$\min_{\omega}(X\omega - y)^2 + \lambda\|\omega\|^2$$

直接对 ω 求偏导并令其 $= 0$:

$$J(\omega) = (X\omega - y)^2 + \lambda\|\omega\|^2 = \omega^T X^T X\omega - 2\omega^T X^T Y + Y^T Y + \lambda\omega^T\omega$$

$$\begin{aligned}\frac{\partial J(\omega)}{\partial\omega} &= \frac{\partial(\omega^T X^T X\omega - 2\omega^T X^T Y + Y^T Y + \lambda\omega^T\omega)}{\partial\omega}\\ &= 2X^T X\omega - 2X^T Y + \lambda\omega\\ &= 2\left(X^T X + \frac{\lambda I}{2}\right)\omega - 2X^T Y\end{aligned}$$

$$\frac{\partial J(\omega)}{\partial\omega} = 0 \Leftrightarrow 2\left(X^T X + \frac{\lambda I}{2}\right)\omega - 2X^T Y = 0$$

$$\Rightarrow \omega = \left(X^T X + \frac{\lambda I}{2}\right)X^T Y$$

得到这个优化问题的解，因此只需要令 ω 取上述值即可完成最小二乘线性分类。

需要注意的是，在计算中已经把参数 b 吸收入向量形式 $\omega = (\omega; b)$，相应的，把训练集再在最右侧加上一列全 1。因此上述 X 写成矩阵形式时具有如下形式:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{18} & 1\\ x_{21} & x_{22} & \cdots & x_{28} & 1\\ \cdots & \cdots & \cdots & \cdots & \cdots\\ x_{m1} & x_{m2} & \cdots & x_{m8} & 1\end{pmatrix} = \begin{pmatrix} \vec{x_1}^T & 1\\ \vec{x_2}^T & 1\\ \cdots & \cdots\\ \vec{x_m}^T & 1\end{pmatrix}$$

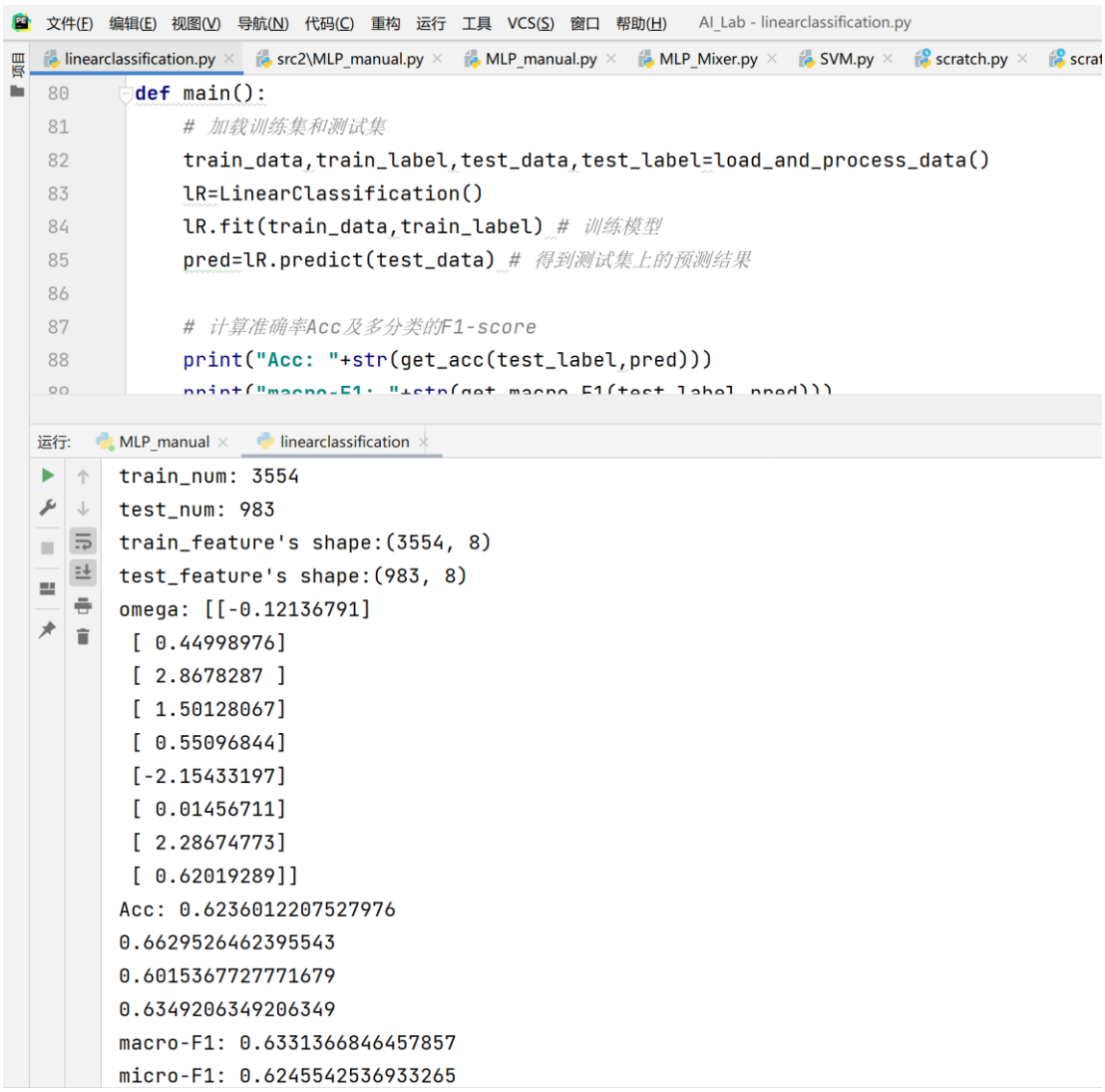
更新 ω 参数的过程如下所示，输入训练集，返回最小二乘分类参数。

```
def fit(self,train_features,train_labels):
    """..."""
    ...

    feature_rows = train_features.shape[0]
    feature_lines = train_features.shape[1]

    X = np.empty([feature_rows, feature_lines + 1], dtype = float)
    for i in range(0, feature_rows): # 0 <= i < feature_rows
        X[i][feature_lines] = 1
        for j in range(0,feature_lines):
            X[i][j] = train_features[i][j]
    #print(X)

    self.Omega = np.empty([feature_lines + 1 , 1],dtype = float)
    #print(self.Omega)

    I = np.eye(feature_lines + 1)
    ...
    self.Omega = lg.inv((X.transpose().dot(X) + (self.Lambda/2)*I)).dot(X.transpose()).dot(train_labels)
    print('omega:',self.Omega)
```

再利用得到的参数在测试集上作预测，输入测试数据集（无 label），返回对所有测试数据的预测类别:

```
'''根据训练好的参数对测试数据test_features进行预测，返回预测结果
预测结果的数据类型应为np数组，shape=(test_num,1) test_num为测试数据的数目'''
def predict(self,test_features):
    """..."""
    feature_rows = test_features.shape[0]
    feature_lines = test_features.shape[1]
    #print(feature_rows)
    pred = np.empty([feature_rows,1],dtype = float)

    X = np.empty([feature_rows, feature_lines + 1], dtype = float)
    for i in range(0, feature_rows): # 0 <= i < feature_rows
        X[i][feature_lines] = 1
        for j in range(0,feature_lines):
            X[i][j] = test_features[i][j]
    #print(X)

    pred = X.dot(self.Omega)

    pred_int = np.empty([feature_rows,1],dtype = int)

    for i in range(0,feature_rows):
        pred_int[i][0] = round(pred[i][0])
    #print(pred_int)

    return pred_int
```

最后运行统计正确率：



二、实现朴素贝叶斯分类器

朴素贝叶斯分类器主要需要实现每个类别的概率以及条件概率和先验概率。首先是计算每个 label 出现的概率，此处使用拉普拉斯平滑来处理没有在训练集中出现过的 label 的概率。计算式如下所示：

$$P(c) = \frac{|D_c| + 1}{|D| + N}$$

即训练集 D 中，类别（Label）为 c 出现的概率为训练集中类别为 c 的数据的条目数+1 除以训练集中所有的数据条目数|D|+Label 的总类别数|N|。分母上加|N|是为了平衡分子上为了平滑而加的 1

而对于先验概率，对于连续型和离散性数据需要分别处理。

对于离散型属性，利用下式计算先验概率：

$$P(x_i|c) = \frac{|D_{c,x_i}| + 1}{|D_c| + N_i}$$

其中 $|D_{c,x_i}|$ 表示类别为 c，且第 i 个属性值为 x_i 的数据条目数，N_i 表示第 i 个属性可能的取值数。

对于连续型数据，可以认为它服从正态分布，直接从训练集中得出这一条属性的 σ^2,μ 即可完全确定它的分布情况。需要注意的是，这一步需要根据 label 类别分组计算。比如训练集包含 10 种 label，那么对于一个连续型属性就要有 10 组 σ^2,μ 值。

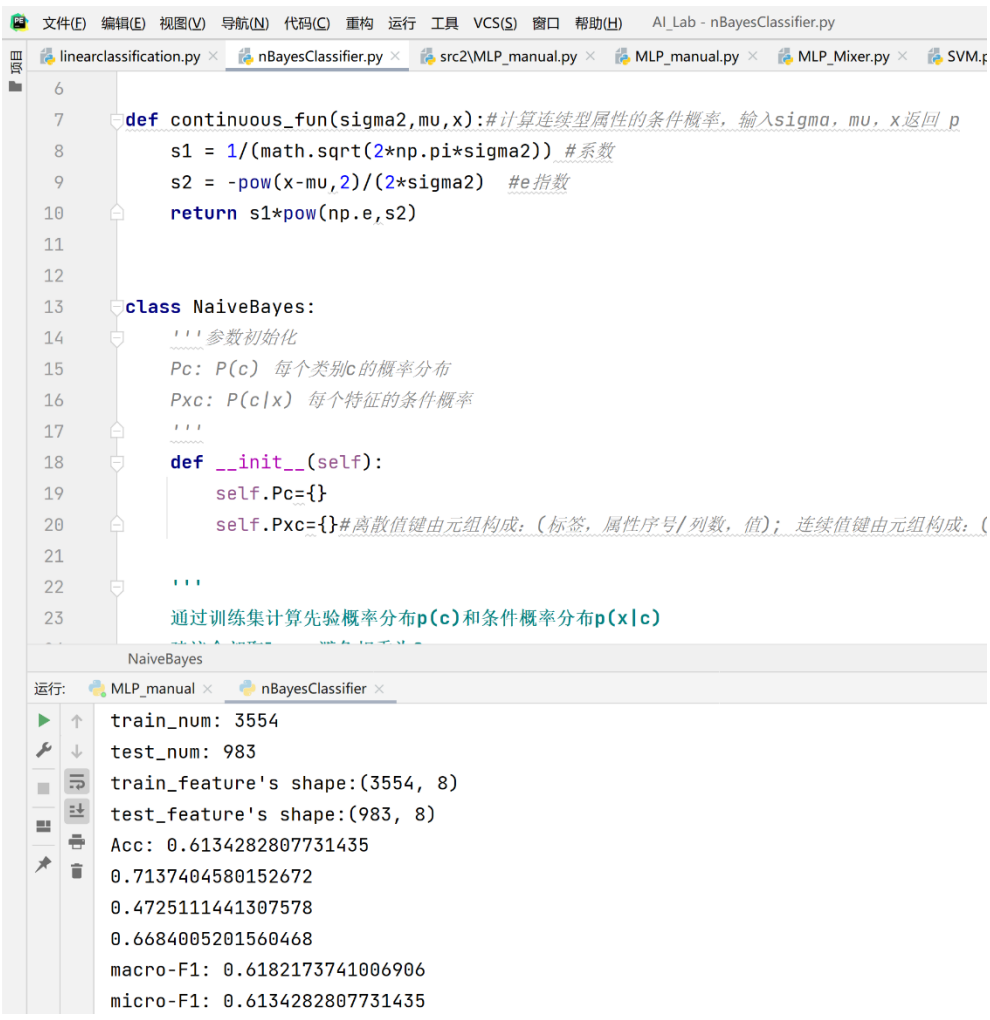
正态分布的函数定义在全局，输入 σ^2,μ,x 返回计算结果：

```
def continuous_fun(sigma2,mu,x):#计算连续型属性的条件概率，输入sigma，mu，x返回 p
    s1 = 1/(math.sqrt(2*np.pi*sigma2)) #系数
    s2 = -pow(x-mu,2)/(2*sigma2)  #e指数
    return s1*pow(np.e,s2)
```

有了所有的概率数据之后，就可以在测试集上预测分类结果。对于一条测试数据而言，判断它属于哪一类通过下式进行。对于连续型数据，累乘项直接用分布函数一步计算到位。

$$h(x) = \arg\max_c P(c) \prod_{i=1}^{d} P(x_i|c)$$

最后的运行结果与正确率如下所示：



三、实现 SVM 分类器

要求完成支持软间隔和核函数的 SVM。

需要满足的对偶问题可以写为：（已经化为标准形式）

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j \kappa(x_i, x_j) - \sum_{i=1}^{m}\alpha_i$$

$$\sum_{i=1}^{m} y_i\alpha_i = 0$$

$$-\alpha_i \le 0, \quad \alpha_i \le C$$

调用 cvxopt 模块引入二次规划求解包，qp 方法要求的参数（P,Q,G,H,A,B）形式如下：

$$\min_{x} Q^T x + \frac{1}{2}x^T P x$$

$$Gx \le H$$

$$Ax = B$$

按规则写出这六个 matrix 格式的矩阵。只需要一一对应即可。

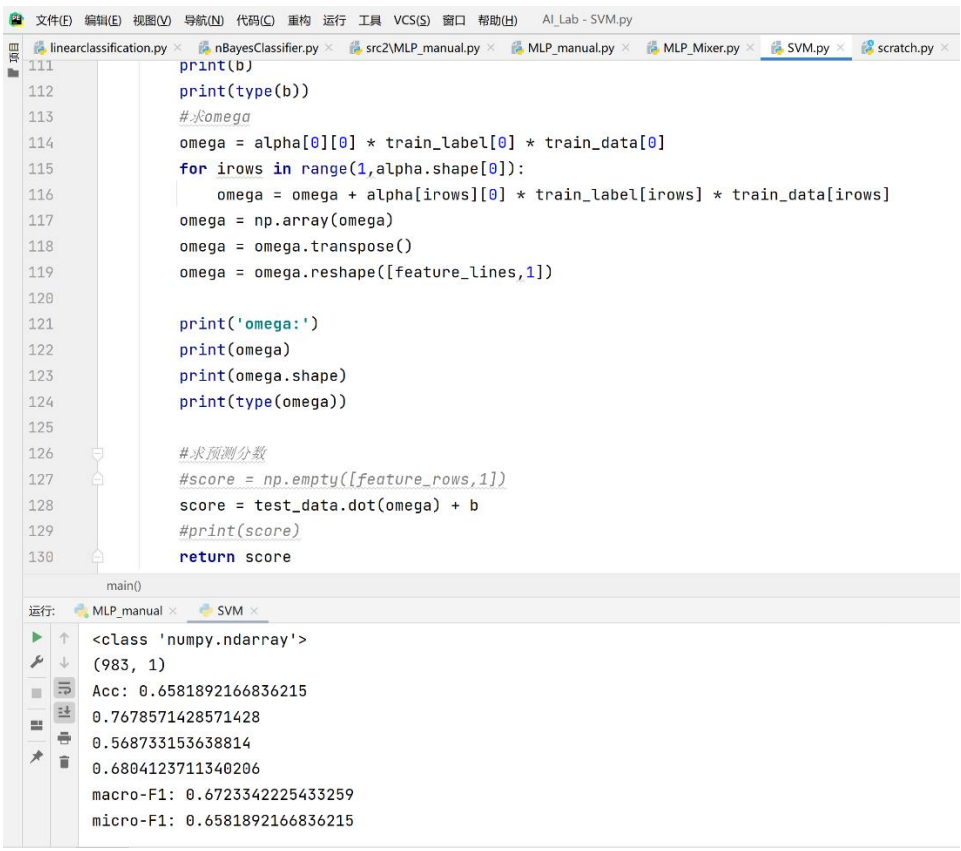
最后求解出 α，按以下方式计算 ω 与 b．ω的维度与一条数据(x)的维度相同, $S = \{i|\alpha_i > 0, i = 1,2,\dots m\}$

$$\vec{\omega} = \sum_{i=1}^{m}\alpha_i y_i \vec{x}_i$$

$$b = \frac{1}{|S|}\sum_{s\in S}\left(y_s - \sum_{i\in S}\alpha_i y_i \kappa(x_i, x_s)\right)$$

得出以上结果后，再对测试集直接进行 $f = \omega x + b$ 即可得到预测分数

最终得到的运行结果如下所示：



四、手写感知机模型并进行反向传播

本部分通过定义 MLP 模型并将其例化的方式实现。

首先定义隐藏层类 HiddenLayer 和回归层类 LogisticRegression．类中有属性 $\omega, input, output$ 。由于是 4 层感知机，例化两个隐层和一个回归层。根据以下公式在每一条训练数据读入时更新参数矩阵：

$$\boldsymbol{h}_1 = \boldsymbol{s}_1(\boldsymbol{W}_1\boldsymbol{x})$$
$$\boldsymbol{h}_2 = \boldsymbol{s}_2(\boldsymbol{W}_2\boldsymbol{h}_1)$$
$$\hat{\boldsymbol{y}} = \boldsymbol{s}_3(\boldsymbol{W}_3\boldsymbol{h}_2)$$
$$L = \ell(y, \hat{\boldsymbol{y}})$$

$$\frac{\partial L}{\partial \boldsymbol{W}_1} = (\boldsymbol{W}_2^{\mathrm{T}}(\boldsymbol{W}_3^{\mathrm{T}}(\ell'\boldsymbol{s}_3')\odot\boldsymbol{s}_2')\odot\boldsymbol{s}_1')\boldsymbol{x}^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{W}_2} = (\boldsymbol{W}_3^{\mathrm{T}}(\ell'\boldsymbol{s}_3')\odot\boldsymbol{s}_2')\boldsymbol{h}_1^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{W}_3} = (\ell'\boldsymbol{s}_3')\boldsymbol{h}_2^{\mathrm{T}}$$

$$\boldsymbol{s}_1 = \boldsymbol{s}_2 = \boldsymbol{\sigma}$$
$$\boldsymbol{\sigma}' = \boldsymbol{\sigma}(\boldsymbol{1} - \boldsymbol{\sigma})$$

$$\boldsymbol{s}_3(x_1, x_2, x_3) = \boldsymbol{Softmax}(x_1, x_2, x_3)$$
$$= \frac{1}{e^{x_1} + e^{x_2} + e^{x_3}}(e^{x_1}, e^{x_2}, e^{x_3})$$

$$\ell(y, \hat{\boldsymbol{y}}) = CrossEntropy(y, \hat{\boldsymbol{y}}) = -\log \hat{y}_i, i = y$$
$$(\ell'\boldsymbol{s}_3')_i = \begin{cases}\hat{y}_i - 1, i = y \\ \hat{y}_i, i \ne y\end{cases}$$

梯度下降算法

$$\boldsymbol{W}_i = \boldsymbol{W}_i - \eta\frac{\partial L}{\partial \boldsymbol{W}_i}$$

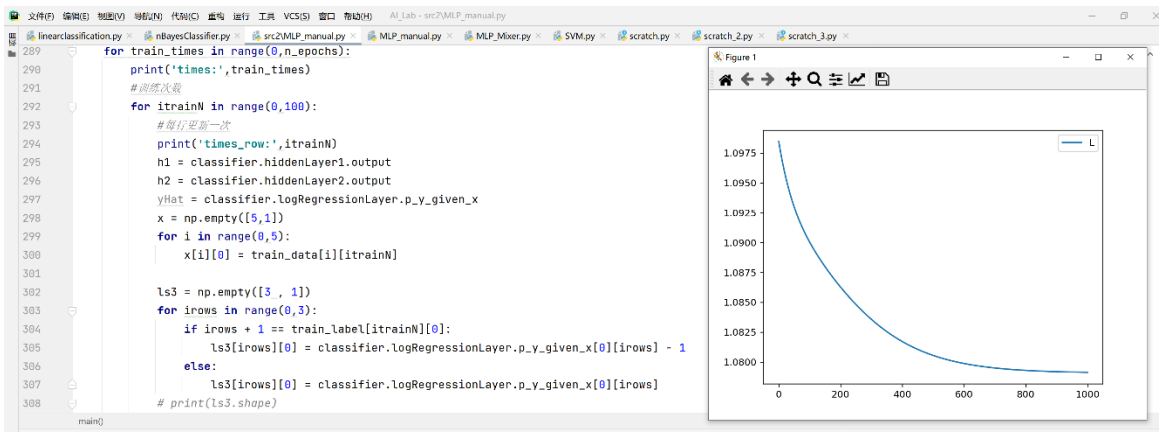
将所有训练数据都过一次称为一轮迭代。一轮迭代中参数矩阵更新 100 次。

更新参数部分实现如下所示，按上述算法逐步求出即可。

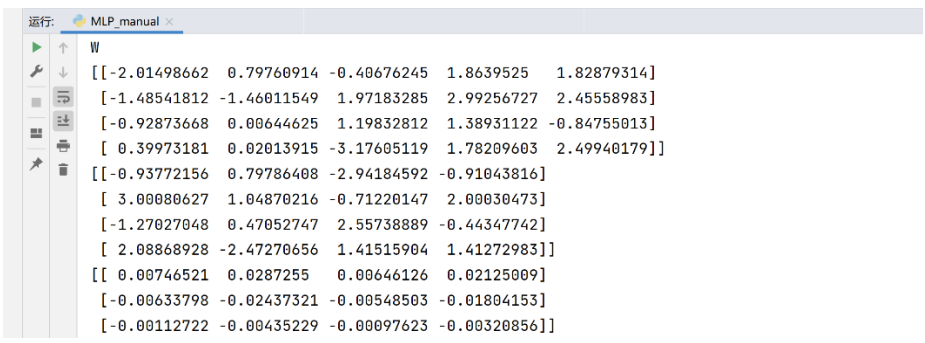
```
302 ls3 = np.empty([3 , 1])
303 for irows in range(0,3):
304     if irows + 1 == train_label[itrainN][0]:
305         ls3[irows][0] = classifier.logRegressionLayer.p_y_given_x[0][irows] - 1
306     else:
307         ls3[irows][0] = classifier.logRegressionLayer.p_y_given_x[0][irows]
308 ...
310 dLW3 = ls3.dot(h2.transpose())
311 ...
314
315 ds2 = np.empty([4,1])
316 w2h1 = classifier.hiddenLayer2.lin_output
317 for irows in range(0,4):
318     for ilines in range(0,1):
319         ds2[irows][ilines] = sigmoid(w2h1[irows][ilines]) * (1 - sigmoid(w2h1[irows][ilines]))
320 dLW2 =  ((classifier.logRegressionLayer.W.transpose().dot(ls3))*ds2).dot(h1.transpose())
321 ...
323
324 ds1 = np.empty([4,1])
325 w1h1 = classifier.hiddenLayer1.lin_output
326 for irows in range(0,4):
327     for ilines in range(0,1):
328         ds1[irows][ilines] = sigmoid(w1h1[irows][ilines]) * (1 - sigmoid(w1h1[irows][ilines]))
329 dLW1 = ((classifier.hiddenLayer2.W.transpose().dot(classifier.logRegressionLayer.W.transpose().dot(ls3) * ds2))*ds1).dot(x.transpose(
```

之后再将 $W_i = W_i - \eta dL/dW_i$ 更新参数即可。

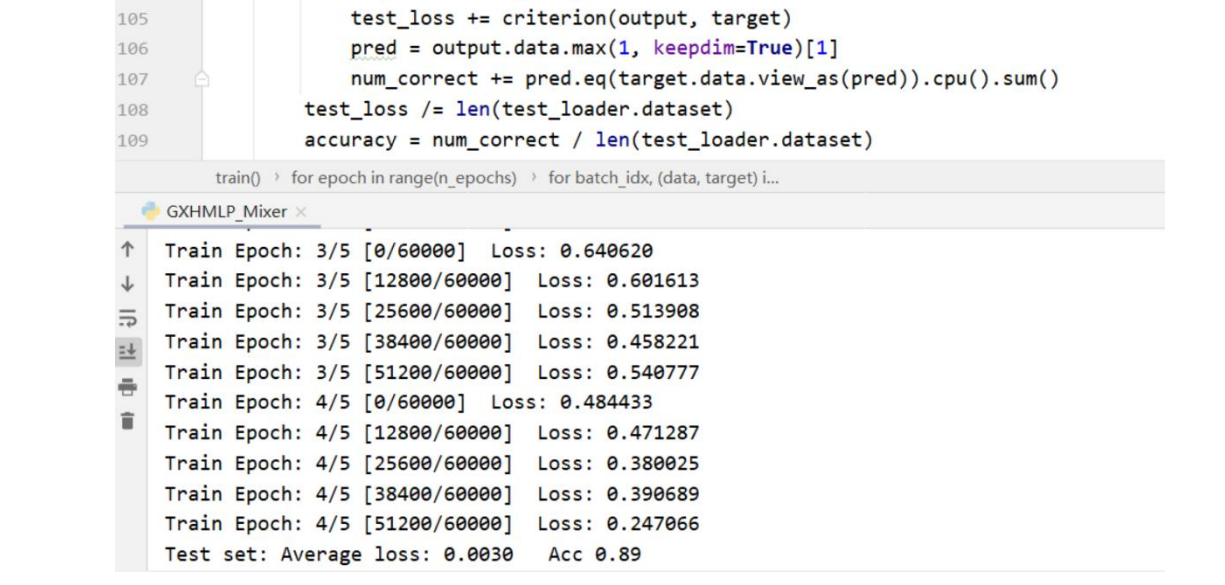
经过 1000 轮迭代，损失曲线如下所示：



而终端输出的最终 W1,W2,W3 矩阵如下所示：

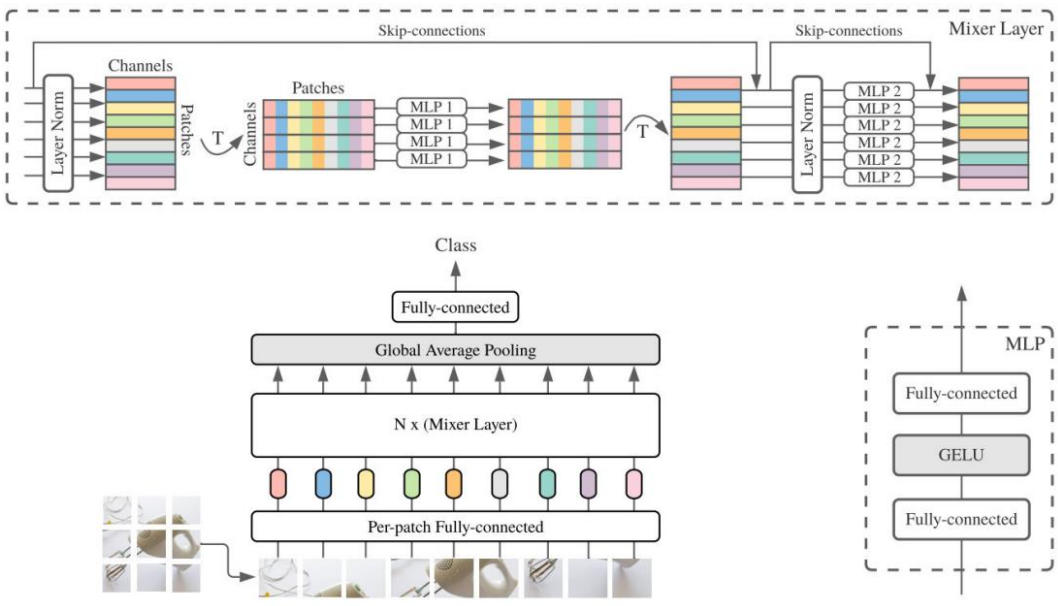


作为对比，以下是调用自动求导模块得到的结果，可以看出自动求导收敛速度较快。



五、复现 MLP-Mixer

此部分可以调用 torch 的所有功能。整体结构如下所示：



只需要根据每个填补区域的提示完成相应的功能。按照上述示例图例化各层即可。详细实现见附件代码。最终运行终端输出如下：