人工智能实验 lab2

> PB18111707 吕瑞
>
> 2021/7/15

传统有监督模型

线性回归模型

模型构建思路

优化目标： $min_w(Xw-y)^2+\lambda||w||^2$

1. 对目标函数求导，得到：$2X^T(Y-XW)-2\lambda W\ (1)$
2. 令(1)式为 0，得到系数矩阵：$W'=(X^TX+\lambda I)^{(-1)}X^TY$
3. 预测结果矩阵：$Y=X'W'$

代码实现：

1. fit：使用 numpy 的矩阵运算函数库，计算系数矩阵

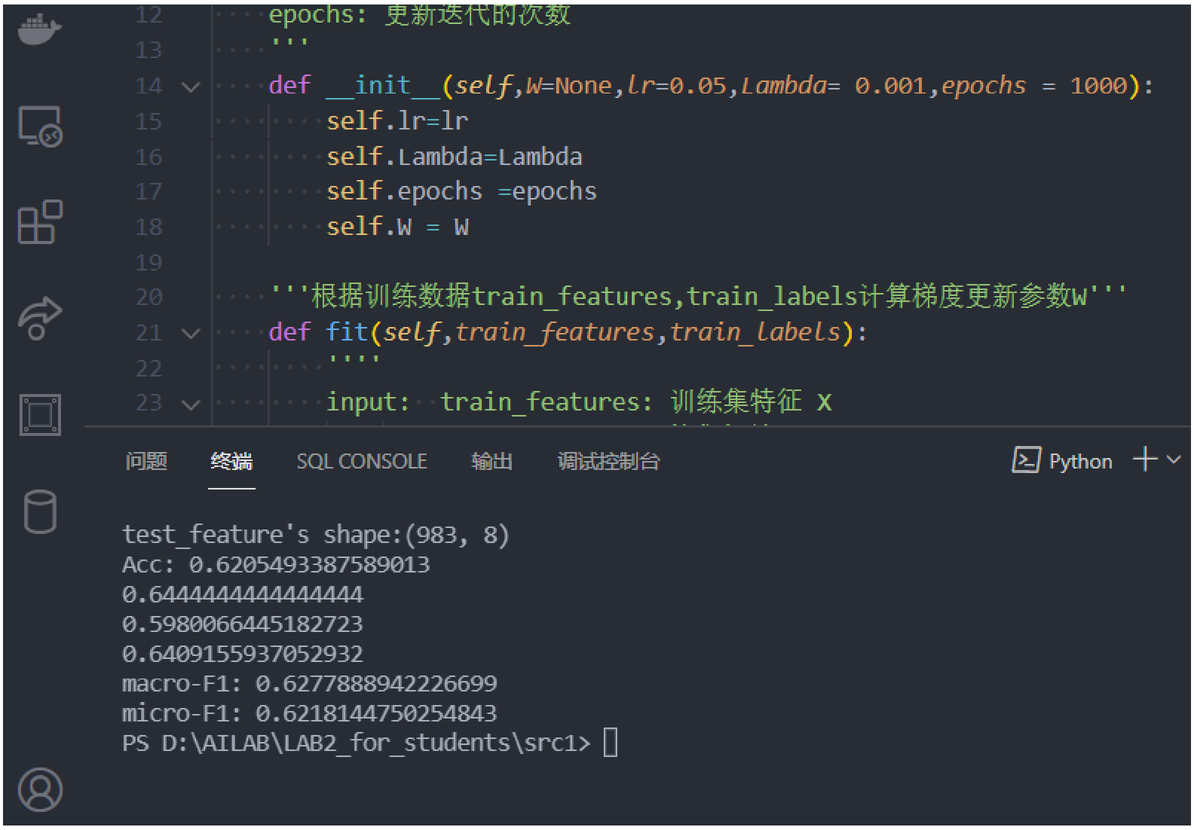
```
# 获得矩阵维数
n = np.shape(train_features)[1]
# w = (XTX + lambda*I)^-1 * XTY
w = np.matmul(np.matmul((np.matmul(train_features.T,train_features) + self.Lambda*np.mat(np.eye(n))).I,train_features.T),train_labels)
```

2. predict：矩阵乘法，求得预测结果

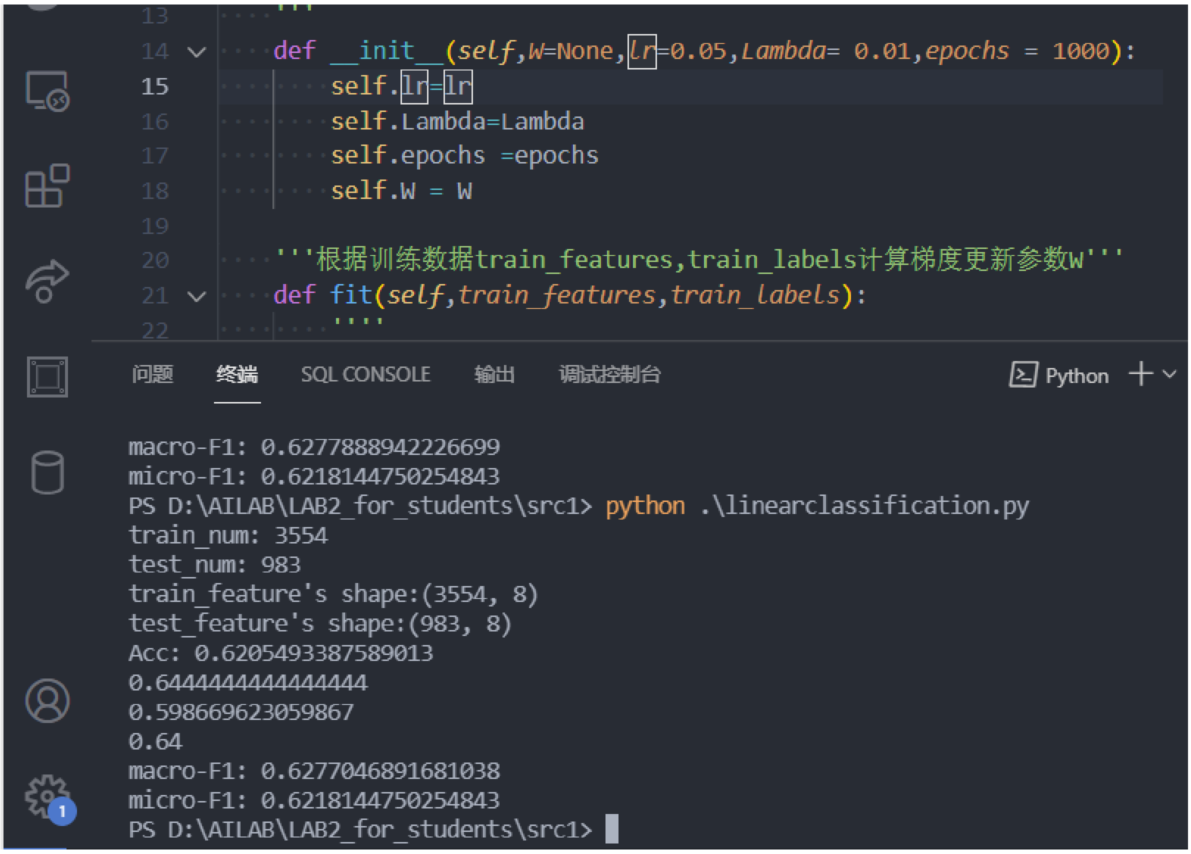
```python
 # Y = X'*W
y = np.matmul(test_features,self.W)
# 四舍五入
y_int = np.around(y)
```

调参结果记录：

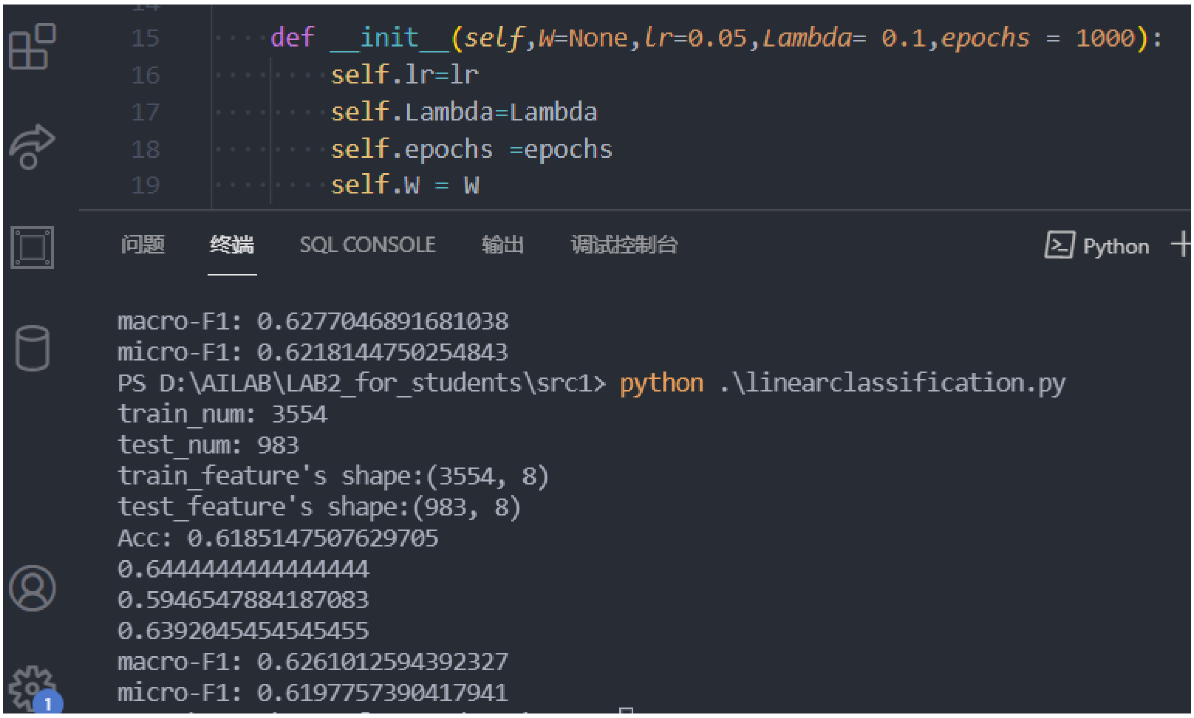
$\lambda = 0.001$



$\lambda = 0.01$



$\lambda = 0.1$



$\lambda = 1$

```python
    def __init__(self,W=None,lr=0.05,Lambda= 1,epochs = 1000):
        self.lr=lr
        self.Lambda=Lambda
        self.epochs =epochs
        self.W = W
```

问题　终端　SQL CONSOLE　输出　调试控制台　　Python

```
macro-F1: 0.6261012594392327
micro-F1: 0.6197757390417941
PS D:\AILAB\LAB2_for_students\src1> python .\linearclassification.py
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)
Acc: 0.6225839267548321
0.6333333333333333
0.5991091314031181
0.6496453900709219
macro-F1: 0.6273626182691244
micro-F1: 0.6235354049923587
PS D:\AILAB\LAB2_for_students\src1>
```

分析：λ 增大到 1 时，平均预测准确率开始上升，这意味着需要调整合适的惩罚系数来保证模型的泛化程度，防止模型过拟合。

朴素贝叶斯模型

模型构建思路

使用拉布拉斯平滑计算条件概率和先验概率

1. 条件概率：$P(c)=(|D_c|+1)/(|D|+N)$
2. 先验概率：$P(x_i|c)$

   离散属性：$P(x_i|c)=(|D_{c,x_i}|+1)/(|D_c|+N_i)$

   连续属性：考虑连续属性的概率密度函数，假定连续变量服从高斯分布，使用训练数据估计高斯分布的参数（均值 μ，方差 σ^2），此时的先验概率公式为：

   $P(x_i|c)=(1/\sqrt{2\pi}\sigma_{c,i})exp(-(x_i-\mu_{c,i})^2/2\sigma_{c,i}^2)$
3. 预测准则：

   $h_{nb}(x)=argmax_{c\in Y}P(c)\prod_{i=1}^{d}P(x_i|c)$

参数解析:

- D：表示训练集；
- N：特征类别总数；
- D_c：训练集中类别为 c 的数据；
- x：待预测的样本数据；
- x_i ：**是预测样本** x 在第 i 个属性上的取值；
- D_{c,x_i}：是 D_c 中第 i 个属性取 x_i 的样本组成的集合；
- d：属性的个数；
- N_i ：表示第 i 个属性可能的取值总数；

核心代码:

- 计算先验概率：

```python
for i in value_counts:
    pc = (value_counts[i]+1)/(num_of_samples+num_of_class)
    self.Pc[i] = pc
```

- 计算条件概率分布：

Pxc 数据表存储格式	特征 0（离散）	特征 1-8 （连续）
label = 1	字典：value_counts = {取值：取值个数}	字典：`` `{'mean':xx,'var':xx,'std':xx} ``
label = 2	字典：value_counts = {取值：取值个数}	字典：`` `{'mean':xx,'var':xx,'std':xx} ``
label = 3	字典：value_counts = {取值：取值个数}	字典：`` `{'mean':xx,'var':xx,'std':xx} ``

```python
data_CX = {}
for i in value_counts:
    data_DivideByLabel= df[df['label']==i]
    data_CX[i] = {}
    for j in feature_list:
        if feature_type[j] == 0:
            # 离散数据
            data_CX[i][j] =
dict(data_DivideByLabel[j].value_counts())
        else:
            # 连续数据
            data_CX[i][j] = {
                'mean':data_DivideByLabel[j].mean(),
                'var':data_DivideByLabel[j].var(),
                'std':data_DivideByLabel[j].std()
            }
    data_CX[i]['count'] = value_counts[i]
self.Pxc = data_CX
```

- 预测新样本：

```python
# 离散数据
res = res+np.log((self.Pxc[c][i][x[i]]+1)/(self.Pxc[c]
['count']+len(self.Pxc[c][i])))

# 连续数据
formula1 = np.exp(-(x[i]-self.Pxc[c][i]['mean'])**2/(2*self.Pxc[c][i]
['var']))
formula2 = 1/np.sqrt(2*np.pi*self.Pxc[c][i]['std'])
res = res+np.log(formula1*formula2)
```

测试结果记录

```
PS D:\AILAB\LAB2_for_students\src1> python .\nBayesClassifier.py
train_num: 3554
test_num: 983
train_feature's shape:(3554, 8)
test_feature's shape:(983, 8)
Acc: 0.6256358087487284
0.723658051689861
0.48358208955223888
0.6834804539722572
macro-F1: 0.6302401984047857
micro-F1: 0.6256358087487284
PS D:\AILAB\LAB2_for_students\src1>
```

SVM 模型

模型构建思路

对偶问题原型:

$max_\alpha \ \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{i=1}^{m}\alpha_i\alpha_j y_i y_j K(x_i,x_j)$

$s.t. \sum_{i=1}^{m}\alpha_i y_i = 0\ ;\ 0 \le \alpha_i \le C;$

二次规划问题的标准形式:

$min\ \frac{1}{2}x^T Px + q^T x;\ \ s.t. Gx \le h;\ \ Ax = b;$

对应转换:

- $Pij = y_i y_j K(x_i,x_j)$: 带核函数的软间隔 SVM
- $\alpha = (a_1, \ldots, a_m)$
- $q^T = (-1, \ldots, -1)_{1*m}$
- $A = (y_1, \ldots y_m)$
- $b = 0$
- $G = \begin{matrix} -I_m \\ I_m \end{matrix}$
- $h = (0, \ldots, 0, C, \ldots, C)^T_{1*2m}$: C 是软间隔参数

使用 cvxopt 求解二次规划问题:

```
sol = cvxopt.solvers.qp(P, q, G, h, A, b) # 求解标准二次规划问题
```

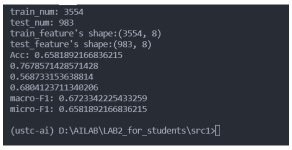
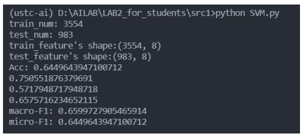
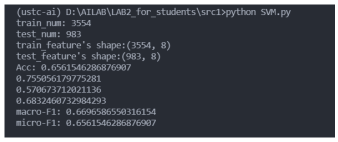
测试函数:

- $f(x) = \sum_{i=1}^{m}\alpha_i y_i K(x,x_i) + b$

b 的计算考量:

- $b = y_i - \sum_{j=1}^{m}\alpha_j y_j K(x_i,x_j)$
- 有三种方式:
  - 对 i 对应的结果取平均值 √: 该方法更能反映数据特征，更合适。
  - 随机选取值
  - 求最大值

实验结果记录

SVM 核函数	linear	poly	gauss
实验结果	 train_num: 3554 test_num: 983 train_feature's shape:(3554, 8) test_feature's shape:(983, 8) Acc: 0.6581892166836215 0.7678571428571428 0.568733153638814 0.6804123711340206 macro-F1: 0.6723342225433259 micro-F1: 0.6581892166836215 (ustc-ai) D:\AILAB\LAB2_for_students\src1>	 (ustc-ai) D:\AILAB\LAB2_for_students\src1>python SVM.py train_num: 3554 test_num: 983 train_feature's shape:(3554, 8) test_feature's shape:(983, 8) Acc: 0.6449643947100712 0.750551876379691 0.5717948717948718 0.6575716234652115 macro-F1: 0.6599727905465914 micro-F1: 0.6449643947100712	 (ustc-ai) D:\AILAB\LAB2_for_students\src1>python SVM.py train_num: 3554 test_num: 983 train_feature's shape:(3554, 8) test_feature's shape:(983, 8) Acc: 0.6561546286876907 0.755056179775281 0.570673712021136 0.6832460732984293 macro-F1: 0.6696586550316154 micro-F1: 0.6561546286876907

总结：我们希望样本在特征空间内线性可分，故而特征空间的好坏对支持向量机的性能至关重要。核函数隐式的定义了线性空间。若核函数选择不恰当，意味着将样本映射到了 不能最大程度线性可分的空间，故而可能导致性能不佳。

本次实验中，尝试了线性核，多项式核，高斯核，发现数据在线性核下对应的空间可分性更好。

深度学习

手写感知机模型并进行反向传播

模型构建思路

> [梯度推导参考文章]

函数准备：

- $f(x) = sigmoid(x) = 1/(1+e^{-x})$ ： 激活函数
- $f'(u_l) = sigmoid'(u_l) = y_l(1-y_l)$： 激活函数求导
- $y' = softmax(x_1, x_2, x_3) = (e^{x_1}, e^{x_3}, e^{x_3})/(e^{x_1}+e^{x_3}+e^{x_3})$ ： 归一化函数
- $E = loss(y, y') = -log(y_i'),\ i = y$： 用交叉熵函数作为损失函数；取向量中，index=label 位置的数的负 log
- $loss'(y, y') = y' - 1, i = y;\ \ loss'(y, y') = y', i \neq y;$： 损失函数的导数

前向传播：保存每一层的输出值，为反向传播做准备

- $y_l = f(u_l) = f(W_l y_{l-1} + b_l)$： l 是当前层的序号；

反向传播：关键在计算梯度

梯度：

- $grad_E = \partial E/\partial W_l = \delta_l y_{l-1}^T;$
- $grad_b = \partial E/\partial b_l = \delta_l;$
- $\delta_l = (W_{l+1}^T \delta_{l+1} o f'(u_l))$ l 层为隐层； $\delta_l = (loss'(y, y') o f'(u_l))$ l 层为输出层；

  o 表示矩阵或者向量中的对应元素相乘；

更新参数：

- $W_l := W_l - lr * grad_E$
- $b_l := b_l - lr * grad_b$

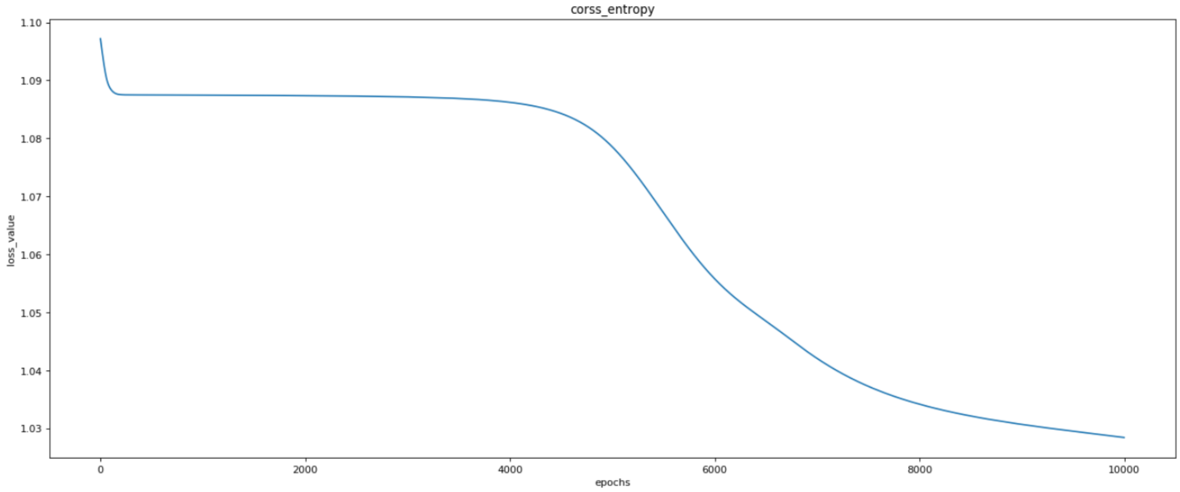
> 注意，为了让手动计算的梯度能和自动计算的梯度保持一致，需要将计算梯度和更新参数分为两个步骤进行，即先计算梯度，再统一更新参数；

> 如果在计算梯度的同时就更新参数，那么在计算下一层的梯度时，会使用已经更新过的参数，虽然这么做没有原则上的错误，但是在自动求导中并不会利用已经更新的参数计算梯度，故而此时就会造成两种方法得出的梯度值有些许差异。

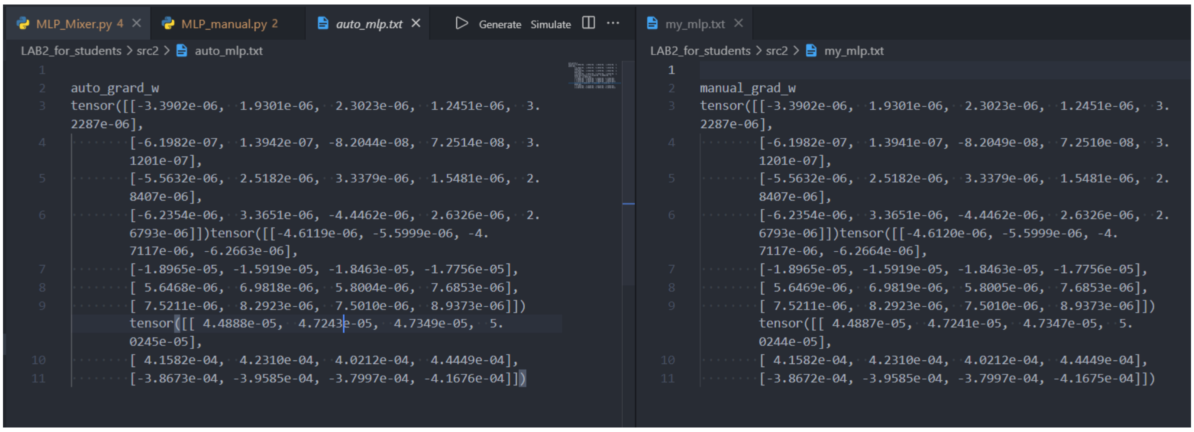
实验结果记录

loss 曲线:

$lr = 1, epochs = 10000, samples\_dim = 100$



手算 MLP 和自动求导的梯度对比:



从图上可以看出，手动实现的梯度计算和自动求导得出的梯度值完全相同。

复现 MLP-Mixer

模型架构图:

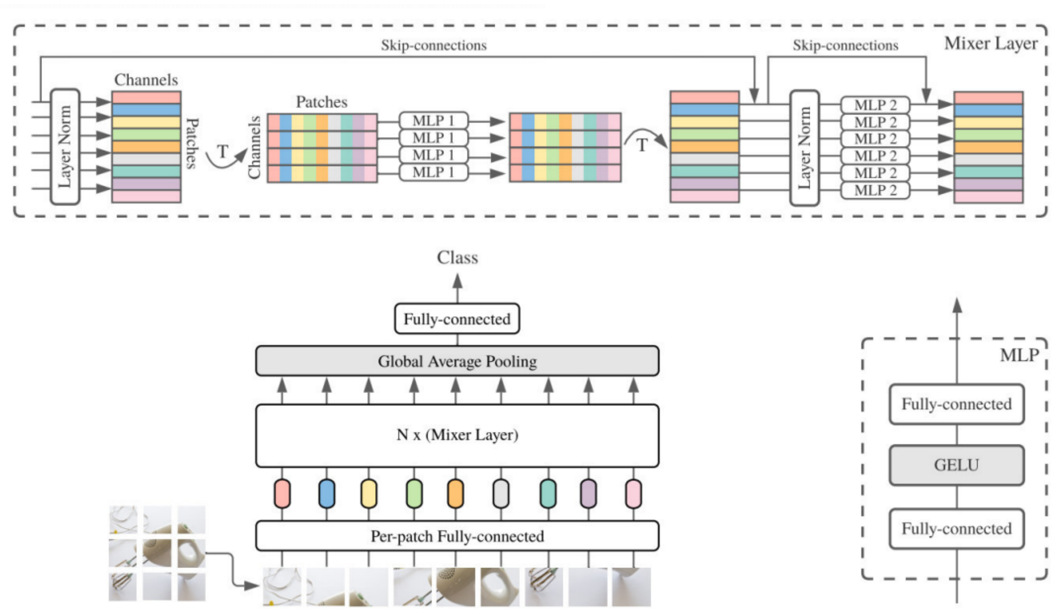


Figure 1: MLP-Mixer consists of per-patch linear embeddings, Mixer layers, and a classifier head. Mixer layers contain one token-mixing MLP and one channel-mixing MLP, each consisting of two fully-connected layers and a GELU nonlinearity. Other components include: skip-connections, dropout, layer norm on the channels, and linear classifier head.

Mixer利用了两种MLP层：

- channel-mixing MLPs：允许不同channels特征之间的交流；
- token-mixing MLPs：允许不同空间位置之间的交流。
- 这两个MLP层是交错的。

「图解读」

- 从图中caption部分可以看到。“Per-patch Fully-connected”我认为就是embedding层，比方说把一个32x32x3的彩色patch图片，全连接映射到128维度的序列。
- Mixer Layer就是文章提出的主要创新结构。其中，**每一个Mixer Layer包含一个token-mixing MLP 和一个channel-mixing MLP**，这两个结构都是由**两个全连接层和GELU激活函数组成。**
- 我们再来看上图的上面部分，体现了Mixer Layer的细节：首先，假设一个图片被分成了9个patch，然后每一个patch经过embedding，变成了一个128的向量。那么原图经过embedding，最终得到的是9x128这样的一个矩阵。
    1. 这个矩阵先经过LayerNorm，相当于是在128这个维度上进行归一化；
    2. 然后矩阵经过转置，变成128x9的样式；
    3. 经过第一个全联接层，这个MLP应该就是channel-mixing了，因为是对9这个patch维度进行计算；
    4. 然后再转置成9x128，再进行layer norm；
    5. 然后token-mixing channels，在128这个spatial维度上进行计算；
    6. 中间加了两个skip connection。

> [模型原理参考文章]()
>
> [复现代码参考文章]()

实验结果记录

对应参数：

```python
model = MLPMixer(num_class=10, patch_size=7, hidden_dim=8, tokens_mlp_dim=32,\
        channels_mlp_dim=32, depth=8, image_size=28, drop_rate=0.).to(device)  # 参数自己设定，其中depth必须大于1
```

> hidden_dim,token_mlp_dim,channels_mlp_dim 适当调大后，模型准确率高的惊人。
>
> AI 真的是个圈啊... 从最简单的感知机出发的神经网络模型，经过 CNN, RNN, Attention，Transtormer... 最后又回到了 MLP，奥卡姆剃刀永远的神。

```
(ustc-ai) D:\AILAB\LAB2_for_students\src2>python MLP_Mixer.py
Train Epoch: 0/5 [0/60000]      Loss: 2.370949
Train Epoch: 0/5 [12800/60000]  Loss: 1.395583
Train Epoch: 0/5 [25600/60000]  Loss: 0.860443
Train Epoch: 0/5 [38400/60000]  Loss: 0.609808
Train Epoch: 3/5 [51200/60000]  Loss: 0.187427
Train Epoch: 4/5 [0/60000]      Loss: 0.208335
Train Epoch: 4/5 [12800/60000]  Loss: 0.192250
Train Epoch: 4/5 [25600/60000]  Loss: 0.150283
Train Epoch: 4/5 [38400/60000]  Loss: 0.164434
Train Epoch: 4/5 [51200/60000]  Loss: 0.126458
Test set: Average loss: 0.1599   Acc 0.95
```

log

`np.mat(np.eye(n))`：构造 n 维单位矩阵；

`np.matmul(A,B)`：矩阵 A，B 相乘；

`np.hstack((a,b))`：水平合并两列 `array`；

> `np.concatenate((a,b),axis=1)` 水平合并两列；

`np.vstack((a,b))`：垂直合并若干列

> `np.concatenate((a,b),axis=0)`