# 手写数字识别 A1-实验报告

## 一、任务目标：

### 1、基本实验要求完成：

a. **实现 softmax 分类器**

b. **实现全连接神经网络分类器**

c. 理解不同的分类器之间的区别，以及使用不同的更新方法优化神经网络 —— 通过实验结果等具体分析

### 2、进阶要求：

a. 尝试使用不同的损失函数和正则化方法，观察并分析其对实验结果的影响

b. 尝试使用不同的优化算法，观察并分析其对训练过程和实验结果的影响

## 二、实验实现过程：

### 1、实验平台概述：

本次实验采用 Python 完成。

依赖库有：

| 依赖库 | 版本 |
| --- | --- |
| numpy | 1.24.3 |
| torchvision | 0.14.0+cu116 |
| pandas | 1.5.1 |
| torch | 1.13.0+cu116 |
| cupy | 12.2.0 |

其中 torch、torchvision 仅用于数据集载入，不对数据集做任何处理， 不参与计算。

全过程使用 numpy 完成， 书写主要算法， 进行计算。

之后为将计算转移至 GPU 加速，采用 cupy 进行替换 numpy 进行迁移，效率提升巨大。

## 2、实验代码框架概述及实现原理：

### ① 数据集导入

```
import torch
import numpy as np
import PIL.Image as Image
import torchvision
import pandas as pd
import cv2
from os import walk
np.set_printoptions(threshold=np.inf)

#  为在某些准备环节上方便实验，torch等框架仅用于数据集读取操作，主体框架用numpy实现
def load_dataset(batch_size = 2, test_batch_size = 50):
    transform =
torchvision.transforms.Compose([torchvision.transforms.ToTensor()])
    path = './data/'
    batch_size = batch_size
    x_train, y_train, x_test, y_test = [], [], [], []
    trainData = torchvision.datasets.MNIST(path,train =
True,transform=transform,download = True)
    testData = torchvision.datasets.MNIST(path,transform=transform,train =
False)
    trainDataLoader = torch.utils.data.DataLoader(dataset=trainData,
batch_size=batch_size,shuffle=True)
    testDataLoader = torch.utils.data.DataLoader(dataset=testData,
batch_size=test_batch_size)

    for idx, data_batch in enumerate(trainDataLoader):
        x = data_batch[0].numpy()/255.0
        y = data_batch[1].reshape(1, -1)
        y = np.squeeze(y)
        Y = []
        for yl in y:
            yy = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
            yy[yl] = 1.0
            Y.append(yy)
        Y = np.array(Y)
        x_train.append(x)
        y_train.append(Y)

    for idx, data_batch in enumerate(testDataLoader):
        x = data_batch[0].numpy()/255.0
```

```
        y = data_batch[1].reshape(1, -1)
        y = np.squeeze(y)
        Y = []
        if test_batch_size == 1:
            for yl in [y]:
                yy = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
                yy[yl] = 1
                Y.append(yy)
        else:
            for yl in y:
                yy = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
                yy[yl] = 1
                Y.append(yy)
        Y = np.array(Y)
        x_test.append(x)
        y_test.append(Y)

    print(len(x_test))

    return x_train, y_train, x_test[:100], y_test[:100], x_test[100:], y_test[100:]

```

使用 torch， torchvision 封装一简单的数据集处理函数。先进行 MINIST 手写数字数据集的读取，然后划分训练、验证集、测试集。

对数据集我们手动做以下处理：

a. 将数据集原本的 tensor 类型转为 numpy 的 array 类型，方便运算。

b. 将数据集图像数值进行归一化

c. 将数据集标签进行独热码处理，方便训练时使用，计算 loss。

## ② math_function 类实现（封装常用的数学函数）

```
from math import *
import numpy as np

class math_fuction:
    def __init__(self):
        pass

    def softmax(self, x):
        e_x = np.exp(x - np.max(x))
        return e_x / e_x.sum(axis=1, keepdims=True)

```

```
    def softmax_derivative(self,x):
        s = self.softmax(x)
        jacobian_matrix = np.diag(s) - np.outer(s, s)
        return jacobian_matrix

    def relu(self, x):
        return np.maximum(0, x)

    def relu_derivative(self, x):
        return np.maximum(0, x+1e-15/np.fabs(x+1e-15))

    def binary_cross_entropy_loss(self, y_true, y_pred):
        epsilon = 1e-15  # 避免log(0)的情况
        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)  # 将预测值限制在[epsilon, 1-epsilon] 范围内
        loss = - (y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
        return np.mean(loss)

    def L1_loss(self, y_true, y_pred):
        return np.sum(np.abs(y_true-y_pred))


    def L2_loss(self, y_true, y_pred):
        return np.sum(np.power(y_true-y_pred, 2))

```

需要注意的事是为防止 softmax 计算溢出，将矩阵中元素减去最大值，约束范围。

交叉熵损失函数计算中也要避免 log（0） 的情况。

## ③ Optimizer（优化器类）的实现

为方便后续的代码书写，以及对比试验，我们将优化器单独抽象出来，封装不同的优化算法。

此部分主要为 **全连接神经网络** 的训练进行铺垫。

softmax 分类器由于其计算等较为简单，直接将各种优化算法集成在内部调用，减轻代码书写过程中接口对齐的负担。

主要实现并应用的是 **mini-batch GD， momentum， adam**

```
import cupy as np
from fuction_utils import math_fuction

class Optimizer:
```

```
    def __init__(self, model, type:str = 'mini-batch', lr:float = 5e-3, decay:float = 0.0) -> None:
        self.type = type
        self.lr = lr
        self.decay = decay
        self.epoch = 0

    def decay_process(self):
        self.lr = self.lr / (1 + self.decay)

    def optimize_params(self, model, dw:list[float], db:list[float], epoch):
        for i in range(0, model.num_layers):
            model.w[i] = model.w[i] - self.lr * dw[model.num_layers - i - 1].T
            model.b[i] = model.b[i] - self.lr * db[model.num_layers - i - 1]

        if self.epoch != epoch:
            self.epoch = epoch
            self.decay_process()

        return model.w, model.b


class Optimizer_Adam(Optimizer):
    def __init__(self, model, beta1:float = 0.9, beta2=0.999,  type: str = 'Adam', lr: float = 0.005, decay: bool = False) -> None:
        super().__init__(model, type, lr, decay)
        self.Vdw = []
        self.Vdb = []
        self.Sdw = []
        self.Sdb = []
        self.beta1 = beta1
        self.beta2 = beta2
        # print(self.type)

        for i in range(model.num_layers):
            self.Vdw.append(0)
            self.Vdb.append(0)
            self.Sdb.append(0)
            self.Sdw.append(0)


    def optimize_params(self, model, dw: list[float], db: list[float], epoch):
        for i in range(0, model.num_layers):
            self.Vdw[i] = self.beta * self.Vdw[i] + (1 - self.beta1) * dw[model.num_layers - i - 1].T
            self.Vdb[i] = self.beta * self.Vdb[i] + (1 - self.beta1) * db[model.num_layers - i - 1]
```

```
49              self.Sdw[i] = self.beta * self.Sdw[i] + (1 - self.beta2) *
    np.power(dw[model.num_layers - i - 1].T, 2)
50              self.Sdb[i] = self.beta * self.Sdb[i] + (1 - self.beta2) *
    np.power(db[model.num_layers - i - 1], 2)
51
52              model.w[i] = model.w[i] - self.lr * self.Vdw[i] /
    (np.power(self.Sdw[i], 1 / 2) + 0.00000001)
53              model.b[i] = model.b[i] - self.lr * self.Vdb[i] /
    (np.power(self.Sdb[i], 1 / 2) + 0.00000001)
54
55          if self.epoch != epoch:
56              self.epoch = epoch
57              self.decay_process()
58
59          return model.w, model.b
60
61
62  class Optimizer_Momentum(Optimizer):
63      def __init__(self, model, beta: float = 0.9, type: str = 'momentum', lr:
    float = 0.005, decay: bool = False) -> None:
64          super().__init__(model, type, lr, decay)
65          self.Vdw = []
66          self.Vdb = []
67          self.beta = beta
68
69          for i in range(model.num_layers):
70              self.Vdw.append(0)
71              self.Vdb.append(0)
72
73      def optimize_params(self, model, dw: list[float], db: list[float], epoch):
74          for i in range(0, model.num_layers):
75              self.Vdw[i] = self.beta * self.Vdw[i] + (1 - self.beta) *
    dw[model.num_layers - i - 1].T
76              self.Vdb[i] = self.beta * self.Vdb[i] + (1 - self.beta) *
    db[model.num_layers - i - 1]
77              model.w[i] = model.w[i] - self.lr * self.Vdw[i]
78              model.b[i] = model.b[i] - self.lr * self.Vdb[i]
79
80          if self.epoch != epoch:
81              self.epoch = epoch
82              self.decay_process()
83
84          return model.w, model.b
85
```

Mini-batch GD 实现原理：

$Require:\ lr$
$Require:\ \theta\ (initial\ prams)$
$while$ 停止标准为满足 $do:$
$1:$ 从训练集中采样$m$个样本 $\{x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(m)}\}$
$2:$ 计算梯度：$g \leftarrow \frac{1}{m}\nabla\sum_i L(f(x^{(i)};\theta), y^{(i)})$
$3:$ 更新权值 $\theta \leftarrow \theta - lr * g$

### Momentum 优化算法实现原理：

$Require:\ lr,\ \beta$
$Require:\ \theta\ (initial\ prams)$
$v = 0$
$while$ 停止标准为满足 $do:$
$1:$ 从训练集中采样$m$个样本 $\{x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(m)}\}$
$2:$ 计算梯度：$g \leftarrow \frac{1}{m}\nabla\sum_i L(f(x^{(i)};\theta), y^{(i)})$
$3:$ 速度更新 $v \leftarrow \beta v + g$
$4:$ 更新权值 $\theta \leftarrow \theta - lr * v$

### Adam 优化算法实现原理：

$Require:\ lr,\ \beta_1,\ \beta_2$
$Require:\ \theta\ (initial\ prams)$
$while$ 停止标准为满足 $do:$
$1:$ 从训练集中采样$m$个样本 $\{x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(m)}\}$
$2:$ 计算梯度：$g \leftarrow \frac{1}{m}\nabla\sum_i L(f(x^{(i)};\theta), y^{(i)})$
$3:$ 累计梯度：$v \leftarrow \beta_1 v + (1-\beta_1)g$
$4:$ 累计平方梯度：$r \leftarrow \beta_2 r + (1-\beta_2)g^2$
$5:$ 修正偏差：$\hat{v} = \frac{v}{1-{\beta_1}^t}\ ;\hat{r} = \frac{r}{1-{\beta_2}^t}$
$6:$ 更新权值：$\theta \leftarrow \theta - lr * \frac{\hat{v}}{\hat{r}^{0.5} + \delta}$

## ④ SoftMaxClassifier 类实现：

此类中实现 Softmax 分类器：

```
import numpy as np
from fuction_utils import math_fuction

f = math_fuction()
class SoftMaxClassifier:
```

```
    def __init__(self, lr=0.005, lossf='cross-entropy', optim='adam', beta=0.9) -> None:

        # 参数初始化
        np.random.seed(2023)
        self.w = np.random.randn(28*28, 10)
        self.b = np.zeros((10, 1))
        self.lr = lr
        self.lossf = lossf
        self.optim = optim
        self.num_layers = 1

        self.Vdw = 0
        self.Vdb = 0
        self.Sdw = 0
        self.Sdb = 0

        self.beta = 0.9

    def forward(self, x, bsz):
        x = np.array([item.reshape(-1, 1) for item in x])
        result = self.w.T @ x + self.b

        result = f.softmax(result)
        return result


    def backward(self, X, Y, bsz):

        result = self.forward(X, bsz)

        X = np.array([item.reshape(-1, 1) for item in X])
        Y = np.array([item.reshape(10, 1) for item in Y])
        dw, db = [], []
        if self.lossf == 'cross-entropy':
            loss = f.binary_cross_entropy_loss(Y, result)
            loss = loss.mean()
        elif self.lossf == 'L2':
            loss = f.L2_loss(Y, result)
            loss = loss.mean()

        if self.lossf == 'cross-entropy':
            dz = result - Y
        elif self.lossf == 'L2':
            dz = []
            for k in range(10):
                add = 0
```

```
                for j in range(10):
                    if j != k:
                        add += result[:,j] * (Y[:,j] - result[:,j])
                res = 2 * np.power(result[:,k], 2) * (Y[:,k] - result[:,k]) - 2 * result[:,k] * (Y[:,k] - result[:,k]) + 2 * result[:,k] * add
                dz.append(res)
            dz = np.array(dz).transpose(1,0,2)

        Dw = dz @ X.transpose((0, 2, 1))
        Dw = 1 / bsz * np.sum(Dw, axis=0)
        Db = 1 / bsz * np.sum(dz, axis=0)
        Db = np.sum(Db, axis=1, keepdims=True)

        return Dw, Db, loss

    def optimize_back(self, dw, db):
        if self.optim == 'adam':
            self.Vdw = self.beta * self.Vdw + (1 - self.beta) * dw.T
            self.Vdb = self.beta * self.Vdb + (1 - self.beta) * db
            self.Sdw = self.beta * self.Sdw + (1 - self.beta) * np.power(dw.T, 2)
            self.Sdb = self.beta * self.Sdb + (1 - self.beta) * np.power(db, 2)

            self.w = self.w - self.lr * self.Vdw / (np.power(self.Sdw, 1 / 2) + 0.00000001)
            self.b = self.b - self.lr * self.Vdb / (np.power(self.Sdb, 1 / 2) + 0.00000001)

        elif self.optim == 'mini-batch':
            self.w -= self.lr * dw.T
            self.b -= self.lr * db

        elif self.optim == 'momentum':
            self.Vdw = self.beta * self.Vdw + (1 - self.beta) * dw.T
            self.Vdb = self.beta * self.Vdb + (1 - self.beta) * db
            self.w = self.w - self.lr * self.Vdw
            self.b = self.b - self.lr * self.Vdb


    def predict(self, x, bsz):
        outputs = self.forward(x, bsz)
        cls = []
        for output in outputs:
            number = np.argmax(output)
            cls.append(number)

        return cls
```

```
95
96      def train_steps(self, X, Y, bsz, optim=None, epoch=None):
97          Dw, Db, loss = self.backward(X, Y, bsz)
98          self.optimize_back(Dw, Db)
99          return loss
100
```

在 Softmax 分类器中进行实现了相对应的 **前向计算函数， 后向梯度计算函数， （mini-batch，adam，momentum）三种梯度下降优化器计算， 用于测试的预测函数， 训练调用函数**等等。

**梯度计算依据损失函数的不同进行了不同的计算。**

同时内置了**学习率，优化选择方案，损失函数选择**，用于 adam 以及 momentum 的参数 **beta** 作为构造函数可选参数。

## ⑤ MLP 类实现（全连接神经网络）

在此类中对本实验中要进行多次对比实验的全连接神经网络分类器进行了实现与封装。

```
1  import numpy as np
2  from fuction_utils import math_fuction
3  import json
4  import os
5  from optimizer import Optimizer
6
7  f = math_fuction()
8
9  class MLP:
10     def __init__(self, num_layers, num_neus, lossf='cross-entropy',
   isBatchNorm=False, isL2=False, lambd=0.01, is_test=False):
11
12         if not isinstance(num_neus, list):
13             assert "num_neus应为列表类型对应每层神经元数量"
14
15         if len(num_neus) != num_layers:
16             assert "层数与每层神经元层数不一致"
17
18         self.num_layers = num_layers
19         self.num_neus = num_neus
20         self.isBatchNorm = isBatchNorm
21         self.is_test = is_test
22         self.lossf = lossf
23
24         # L2正则化
25         self.isL2 = isL2
26         self.lambd = lambd
```

```
27
28          # 根据信息初始化权重
29          self.w = []
30          self.b = []
31
32          # 用于前向传播时归一化输入
33          self.sigma2 = []
34          self.mu = []
35
36          # 保证参数随机初始化一致
37          np.random.seed(2023)
38
39          for i in range(num_layers):
40              if i != num_layers - 1:
41                  self.w.append(np.random.randn(num_neus[i], num_neus[i+1]))
42                  self.b.append(0.0)
43              else:
44                  self.w.append(np.random.randn(num_neus[i], 10))
45                  self.b.append(0.0)
46              self.Vdw.append(0.0)
47              self.Vdb.append(0.0)
48              self.Sdw.append(0.0)
49              self.Sdb.append(0.0)
50
51      def forward(self, x, bsz):
52          if self.isBatchNorm and not self.is_test:
53              result, z, L2_loss = self.forward_back_batchnorm(x, bsz)
54          elif self.isBatchNorm and self.is_test:
55              result, z, L2_loss = self.forward_pre_batchnorm(x, bsz)
56          else:
57              result, z, L2_loss = self.forward_normal(x, bsz)
58
59          return result, z, L2_loss
60
61      def forward_normal(self, x, bsz):
62          # 可定义网络结构为每层间嵌套relu函数作为激活函数，最后输出层后使用softmax函
    数计算概率
63          Z = []
64          L2_loss = 0
65
66          x = np.array([item.reshape(-1, 1) for item in x])
67          for i in range(self.num_layers):
68              result = self.w[i].T @ x + self.b[i]
69              Z.append(result)
70              x = f.relu(result)
71
72              if self.isL2:
```

```
                L2_loss += np.sum((self.lambd/(2*bsz)) *
                                  (self.w[i] @ self.w[i].T))

        result = f.softmax(result)

        return result, Z, L2_loss

    def forward_back_batchnorm(self, x, bsz):    # 用于训练反向传播的含有归一化输入
        Z = []
        L2_loss = 0
        x = np.array([item.reshape(-1, 1) for item in x])
        self.mu.clear()
        self.sigma2.clear()

        for i in range(self.num_layers):
            result = self.w[i].T @ x + self.b[i]
            # 归一化输入
            muL = (1 / bsz) * np.sum(result, axis=(0, 2), keepdims=True)
            sigmaL = (1 / bsz) * np.sum(np.power(result - muL, 2),
                                        axis=(0, 2), keepdims=True)
            z_norm = (result - muL) / (np.sqrt(sigmaL + 0.00000001))
            gamma, beta_1 = 1 * \
                np.sqrt(sigmaL + 0.00000001), muL + 0  # 此时的方差为1，均值为0
            result = np.multiply(z_norm, gamma) + beta_1
            self.mu.append(muL)
            self.sigma2.append(sigmaL)

            Z.append(result)
            x = f.relu(result)

            if self.isL2:
                L2_loss += np.sum((self.lambd/(2*bsz)) *
                                  (self.w[i] @ self.w[i].T))

        result = f.softmax(result)

        return result, Z, L2_loss

    def forward_pre_batchnorm(self, x, bsz):    # 用于测试的forward（含归一化）
        x = np.array([item.reshape(-1, 1) for item in x])
        result = 0
        for i in range(0, self.num_layers):
            result = self.w[i].T@x + self.b[i]
            # 归一化输入
            z_norm = (result - self.mu[i]) / \
                (np.sqrt(self.sigma2[i] + 0.00000001))
```

```
119             gamma, beta_1 = 1 * \
120                 np.sqrt(self.sigma2[i] + 0.00000001), self.mu[i] + 0
121             result = np.multiply(z_norm, gamma) + beta_1
122             x = f.relu(result)
123 
124         result = f.softmax(result)
125         return result, [], 0.0       # 保持接口统一
126 
127     def backward(self, X, Y, bsz):
128         if self.lossf == 'cross-entropy':
129             dw, db, loss = self.backward_crossentropy(X, Y, bsz)
130         elif self.lossf == 'L2':
131             dw, db, loss = self.backward_L2(X, Y, bsz)
132 
133         return dw, db, loss
134 
135     def backward_crossentropy(self, X, Y, bsz):
136         if len(X) != len(Y):
137             assert "数据与其标签长度不一致"
138         result, z, L2_loss = self.forward(X, bsz)
139         X = np.array([item.reshape(-1, 1) for item in X])
140         Y = np.array([item.reshape(10, 1) for item in Y])
141         dw, db = [], []
142         loss = f.binary_cross_entropy_loss(Y, result)
143         loss = loss.mean()  # 为方便损失度量，不添加L2正则化损失，但L2正则化若被启
    用会加入到梯度下降的计算中
144 
145         for i in range(self.num_layers - 1, 0, -1):
146             if i == self.num_layers - 1:
147                 dz = result - Y
148             else:
149                 dz = self.w[i + 1] @ dz * f.relu_derivative(z[i])
150 
151             if self.isL2:
152                 Dw = dz @ f.relu(z[i - 1].transpose((0, 2, 1))) + \
153                     (self.lambd / bsz) * self.w[i].T
154             else:
155                 Dw = dz @ f.relu(z[i - 1].transpose((0, 2, 1)))
156             Dw = np.sum(Dw, axis=0)
157             Db = np.sum(dz, axis=0)
158             Db = np.sum(Db, axis=1, keepdims=True)
159             dw.append(Dw)
160             db.append(Db)
161 
162         dz = self.w[1] @ dz * f.relu_derivative(z[0])
163 
164         if self.isL2:
```

```
            Dw = dz @ X.transpose((0, 2, 1)) + (self.lambd / bsz) * self.w[0].T
        else:
            Dw = dz @ X.transpose((0, 2, 1))

        Dw = np.sum(Dw, axis=0)
        Db = np.sum(dz, axis=0)
        Db = np.sum(Db, axis=1, keepdims=True)
        dw.append(Dw)
        db.append(Db)

        return dw, db, loss

    def backward_L2(self, X, Y, bsz):       # 使用L2损失的反向传播
        if len(X) != len(Y):
            assert "数据与其标签长度不一致"
        result, z, L2_loss = self.forward(X, bsz)
        X = np.array([item.reshape(-1, 1) for item in X])
        Y = np.array([item.reshape(10, 1) for item in Y])
        dw, db = [], []
        loss = f.L2_loss(Y, result)
        loss = loss/bsz # 为方便损失度量，不添加L2正则化损失，但L2正则化若被启用会加入到梯度下降的计算中

        for i in range(self.num_layers - 1, 0, -1):
            if i == self.num_layers - 1:
                dz = []
                for k in range(10):
                    add = 0
                    for j in range(10):
                        if j != k:
                            add += result[:,j] * (Y[:,j] - result[:,j])
                    res = 2 * np.power(result[:,k], 2) * (Y[:,k] - result[:,k]) - 2 * result[:,k] * (Y[:,k] - result[:,k]) + 2 * result[:,k] * add
                    dz.append(res)
                dz = np.array(dz).transpose(1,0,2)
            else:
                dz = self.w[i + 1] @ dz * f.relu_derivative(z[i])

            if self.isL2:
                Dw = dz @ f.relu(z[i - 1].transpose((0, 2, 1))) + \
                    (self.lambd / bsz) * self.w[i].T
            else:
                Dw = dz @ f.relu(z[i - 1].transpose((0, 2, 1)))
            Dw = np.sum(Dw, axis=0)
            Db = np.sum(dz, axis=0)
```

```
208             Db = np.sum(Db, axis=1, keepdims=True)
209             dw.append(Dw)
210             db.append(Db)
211 
212         dz = self.w[1] @ dz * f.relu_derivative(z[0])
213 
214         if self.isL2:
215             Dw = dz @ X.transpose((0, 2, 1)) + (self.lambd / bsz) *
    self.w[0].T
216         else:
217             Dw = dz @ X.transpose((0, 2, 1))
218 
219         Dw = np.sum(Dw, axis=0)
220         Db = np.sum(dz, axis=0)
221         Db = np.sum(Db, axis=1, keepdims=True)
222         dw.append(Dw)
223         db.append(Db)
224 
225         return dw, db, loss
226 
227     def predict(self, x, bsz):
228         outputs, Z, l2_loss = self.forward(x, bsz)
229 
230         cls = []
231         for output in outputs:
232             number = np.argmax(output)
233             cls.append(number)
234 
235         return cls
236 
237     def train_steps(self, X, Y, bsz, optimizer: Optimizer, epoch):
238         self.istest = False
239         dw, db, loss = self.backward(X, Y, bsz)
240         optimizer.optimize_params(self, dw, db, epoch)
241         return loss
242 
243     def save_weights(self, file_name='weights.json'):
244         print("Saving....")
245         if self.isBatchNorm:
246             weights = {
247                 'w': [weight.tolist() for weight in self.w],
248                 'b': [weight.tolist() for weight in self.b],
249                 'mu': [weight.tolist() for weight in self.mu],
250                 'sigma2': [weight.tolist() for weight in self.sigma2],
251             }
252         else:
253             weights = {
```

```
                'w': [weight.tolist() for weight in self.w],
                'b': [weight.tolist() for weight in self.b],
            }
        json_weights = json.dumps(weights)
        try:
            with open(file_name, 'w') as weight:
                weight.write(json_weights)
            print("Save Success!")
        except FileNotFoundError:
            os.mknod(file_name)
            self.save_weights(file_name)

    def load_weights(self, file_name='weights.json'):
        print("Loading Checkpoint File " + file_name + ".....")
        weights = {}
        with open(file_name, 'r') as weight:
            weights = json.load(weight)

        self.w = [np.array(weight) for weight in weights['w']]
        self.b = [np.array(weight) for weight in weights['b']]

        try:
            self.mu = [np.array(weight) for weight in weights['mu']]
            self.sigma2 = [np.array(weight) for weight in weights['sigma2']]
        except:
            print('并未启用归一化输入')
            self.isBatchNorm = False

        print("Loading Success!")

```

在本类中先对其主要属性分析：

a. 神经网络特征属性（层数，每层神经元数量）

b. 损失函数选择 （交叉熵，L2 损失）

c. 可选项（是否需要每层输入归一化（BatchNorm）， 是否进行 L2 正则化，是否处于测试状态）

d. 用于 L2 正则化的参数 lambd

主要函数有：

a. 前向传播计算函数

   i. 无 BatchNorm 的一般前向传播

   ii. 有 BatchNorm 的用于训练的前向传播

iii. 有 BatchNorm 的用于测试的前向传播

b. 后向梯度计算

i. 基于交叉熵损失函数

ii. 基于 L2 损失函数

iii. 包含 L2 正则化损失的梯度计算

c. 预测函数

d. 训练步骤调用函数

e. 模型参数的保存和载入函数

## ⑥ 分类器训练、测试实现

```
import numpy as npp
import cupy as np
import os
from datasets_prepare import load_dataset, selfmake_dataset_load
from optimizer import (Optimizer, Optimizer_Adam, Optimizer_Momentum)
from mlp import MLP
import matplotlib.pyplot as plt
import json
from softmax_classifier import SoftMaxClassifier
from concurrent.futures import ThreadPoolExecutor,wait,ALL_COMPLETED
import matplotlib.ticker as ticker
import math
from tqdm import tqdm

classifier_miniBatch = MLP(3, [28*28, 256, 30], isBatchNorm=False)
classifier_adam = MLP(3, [28*28, 256, 30], isBatchNorm=False)
classifier_momentum = MLP(3, [28*28, 256, 30], isBatchNorm=False)

classifier_miniBatch_withBatchNorm = MLP(3, [28*28, 256, 30], isBatchNorm=True)
classifier_adam_withBatchNorm = MLP(3, [28*28, 256, 30], isBatchNorm=True)
classifier_momentum_withBatchNorm = MLP(3, [28*28, 256, 30], isBatchNorm=True)

classifier_miniBatch_withL2 = MLP(3, [28*28, 256, 30], isBatchNorm=False,
isL2=True)
classifier_adam_withL2 = MLP(3, [28*28, 256, 30], isBatchNorm=False, isL2=True)
classifier_momentum_withL2 = MLP(3, [28*28, 256, 30], isBatchNorm=False,
isL2=True)

classifier_miniBatch_withL2andBN = MLP(3, [28*28, 256, 30], isBatchNorm=True,
isL2=True)
```

```
classifier_adam_withL2andBN = MLP(3, [28*28, 256, 30], isBatchNorm=True, isL2=True)
classifier_momentum_withL2andBN = MLP(3, [28*28, 256, 30], isBatchNorm=True, isL2=True)

classifier_miniBatch_withL2andBN_lossL2 = MLP(3, [28*28, 256, 30], isBatchNorm=True, isL2=True, lossf='L2')
classifier_adam_withL2andBN_lossL2 = MLP(3, [28*28, 256, 30], isBatchNorm=True, isL2=True, lossf='L2')
classifier_momentum_withL2andBN_lossL2 = MLP(3, [28*28, 256, 30], isBatchNorm=True, isL2=True, lossf='L2')

softmax_classifier_miniBatch = SoftMaxClassifier(optim='mini-batch')
softmax_classifier_adam = SoftMaxClassifier()
softmax_classifier_momentum = SoftMaxClassifier(optim='momentum', beta=0.9)

classifier_adam_lr5e_1 = MLP(3, [28*28, 256, 30], isBatchNorm=False, lr=5e-1)
classifier_adam_lr5e_2 = MLP(3, [28*28, 256, 30], isBatchNorm=False, lr=5e-2)
classifier_adam_lr5e_3 = MLP(3, [28*28, 256, 30], isBatchNorm=False, lr=5e-3)
classifier_adam_lr5e_4 = MLP(3, [28*28, 256, 30], isBatchNorm=False, lr=5e-4)

softmax_classifier_adam_lossL2 = SoftMaxClassifier(lr=5e-3,lossf='L2')
classifier_adam_lossL2 = MLP(3, [28*28, 256, 30], lossf='L2')

softmax_classifier_adam_lr5e_1 = SoftMaxClassifier(lr=5e-1)
softmax_classifier_adam_lr5e_2 = SoftMaxClassifier(lr=5e-2)
softmax_classifier_adam_lr5e_3 = SoftMaxClassifier(lr=5e-3)
softmax_classifier_adam_lr5e_4 = SoftMaxClassifier(lr=5e-4)

softmax_classifier_adam_beta03 = SoftMaxClassifier(beta=0.3)
softmax_classifier_adam_beta05 = SoftMaxClassifier(beta=0.5)
softmax_classifier_adam_beta07 = SoftMaxClassifier(beta=0.7)
softmax_classifier_adam_beta09 = SoftMaxClassifier(beta=0.9)

softmax_classifier_momentum_beta03 = SoftMaxClassifier(beta=0.3, optim='momentum')
softmax_classifier_momentum_beta05 = SoftMaxClassifier(beta=0.5, optim='momentum')
softmax_classifier_momentum_beta07 = SoftMaxClassifier(beta=0.7, optim='momentum')
softmax_classifier_momentum_beta09 = SoftMaxClassifier(beta=0.9, optim='momentum')

classifier_momentum_lr5e_1 = MLP(3, [28*28, 256, 30], isBatchNorm=False, lr=5e-1)
classifier_momentum_lr5e_2 = MLP(3, [28*28, 256, 30], isBatchNorm=False, lr=5e-2)
```

```
64 classifier_momentum_lr5e_3 = MLP(3, [28*28, 256, 30], isBatchNorm=False, lr=5e-
   3)
65 classifier_momentum_lr5e_4 = MLP(3, [28*28, 256, 30], isBatchNorm=False, lr=5e-
   4)
66
67 classifier_miniBatch20 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
68 classifier_miniBatch50 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
69 classifier_miniBatch100 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
70 classifier_miniBatch200 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
71 classifier_miniBatch300 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
72
73 classifier_momentum20 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
74 classifier_momentum50 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
75 classifier_momentum100 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
76 classifier_momentum200 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
77 classifier_momentum300 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
78
79 classifier_adam20 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
80 classifier_adam50 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
81 classifier_adam100 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
82 classifier_adam200 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
83 classifier_adam300 = MLP(3, [28*28, 256, 30], isBatchNorm=False)
84
85 classifier_adam_beta03 = MLP(3, [28*28, 256, 30], isBatchNorm=False, beta=0.3)
86 classifier_adam_beta05 = MLP(3, [28*28, 256, 30], isBatchNorm=False, beta=0.5)
87 classifier_adam_beta07 = MLP(3, [28*28, 256, 30], isBatchNorm=False, beta=0.7)
88 classifier_adam_beta09 = MLP(3, [28*28, 256, 30], isBatchNorm=False, beta=0.9)
89
90 classifier_momentum_beta03 = MLP(3, [28*28, 256, 30], isBatchNorm=False,
   beta=0.3)
91 classifier_momentum_beta05 = MLP(3, [28*28, 256, 30], isBatchNorm=False,
   beta=0.5)
92 classifier_momentum_beta07 = MLP(3, [28*28, 256, 30], isBatchNorm=False,
   beta=0.7)
93 classifier_momentum_beta09 = MLP(3, [28*28, 256, 30], isBatchNorm=False,
   beta=0.9)
94
95 mlp_small_size_adam = MLP(3, [28*28, 64, 10], isBatchNorm=False)
96 mlp_middle_size_adam = MLP(3, [28*28, 128, 25], isBatchNorm=False)
97 mlp_big_size_adam = MLP(3, [28*28, 256, 30], isBatchNorm=False)
98 mlp_huge_size_adam = MLP(3, [28*28, 512, 64], isBatchNorm=False)
99 mlp_Morehuge_size_adam = MLP(3, [28*28, 1024, 128], isBatchNorm=False)
100 mlp_hugest_size_adam = MLP(3, [28*28, 2048, 256], isBatchNorm=False)
101
102 mlp_overFit_size_adam = MLP(3, [28*28, 5096, 128], isBatchNorm=False)
103 mlp_overFit_size_adam_L2_001 = MLP(3, [28*28, 5096, 128], isBatchNorm=False,
   isL2=True, lambd=1e-2)
```

```
mlp_overFit_size_adam_L2_0001 = MLP(3, [28*28, 5096, 128], isBatchNorm=False, isL2=True, lambd=1e-3)
mlp_overFit_size_adam_L2_00001 = MLP(3, [28*28, 5096, 128], isBatchNorm=False, isL2=True, lambd=1e-4)
mlp_overFit_size_adam_Droup = MLP(3, [28*28, 2048, 256], isBatchNorm=False, isdropout=True, dropout=[0.0,0.1,0.1])

mlp_overFitadam_Droup_01 = MLP(3, [28*28, 5096, 128], isBatchNorm=False, isdropout=True, dropout=[0.0,0.1,0.1])
mlp_overFitadam_Droup_001 = MLP(3, [28*28, 5096, 128], isBatchNorm=False, isdropout=True, dropout=[0.0,0.01,0.01])

mlp_overFit_size_adam_DroupBN = MLP(3, [28*28, 2048, 256], isBatchNorm=True, isdropout=True, dropout=[0.0,0.1,0.1])
mlp_overFit_size_adam_DroupL2 = MLP(3, [28*28, 2048, 256], isBatchNorm=False, isdropout=True, isL2=True, dropout=[0.0,0.1,0.1])
mlp_overFit_size_adam_DroupL2BN = MLP(3, [28*28, 2048, 256], isBatchNorm=True, isdropout=True, isL2=True, dropout=[0.0,0.1,0.1])

mlp_short_length_adam = MLP(2, [28*28, 100], isBatchNorm=False)
mlp_normal_length_adam = MLP(3, [28*28, 100, 70], isBatchNorm=False)
mlp_long_length_adam = MLP(4, [28*28, 100, 70, 50], isBatchNorm=False)

mlp_longest_adam = MLP(5, [28*28, 100, 70, 50, 30], isBatchNorm=False)
mlp_longest_adam_BN = MLP(5, [28*28, 100, 70, 50, 30], isBatchNorm=True)
mlp_longest_adam_Droup = MLP(5, [28*28, 100, 70, 50, 30], isBatchNorm=False, isdropout=True, dropout=[0.0,0.1,0.1,0.1,0.1])
mlp_longest_adam_L2 = MLP(5, [28*28, 100, 70, 50, 30], isBatchNorm=False, isL2=True)

classifier_adam_dropout = MLP(3, [28*28, 256, 30], isBatchNorm=False, isdropout=True, dropout=[0.0,0.1,0.1])

classifiers = {
    # 'classifier_miniBatch' : classifier_miniBatch,
    # 'classifier_adam' : classifier_adam,
    # 'classifier_momentum' : classifier_momentum,

    # 'mlp_small_size_adam' : mlp_small_size_adam,
    # 'mlp_middle_size_adam' : mlp_middle_size_adam,
    # 'mlp_big_size_adam' : mlp_big_size_adam,
    # 'mlp_huge_size_adam' : mlp_huge_size_adam,
    # 'mlp_Morehuge_size_adam' : mlp_Morehuge_size_adam,
    # 'mlp_hugest_size_adam' : mlp_hugest_size_adam,

    'HUGE_adam' : mlp_overFit_size_adam,
    'HUGE_adam_L2_1e-2' : mlp_overFit_size_adam_L2_001,
```

```
    # 'HUGE_adam_L2_1e-3' : mlp_overFit_size_adam_L2_0001,
    'HUGE_adam_L2_1e-4' : mlp_overFit_size_adam_L2_00001,
    # 'mlp_HUGE_adam_Droup' : mlp_overFit_size_adam_Droup,
    # 'mlp_HUGE_adam_DroupBN':mlp_overFit_size_adam_DroupBN,
    # 'mlp_HUGE_adam_DroupL2' : mlp_overFit_size_adam_DroupL2,
    # 'mlp_HUGE_adam_DroupL2BN' : mlp_overFit_size_adam_DroupL2BN

    # 'mlp_short_length_adam' : mlp_short_length_adam,
    # 'mlp_normal_length_adam' : mlp_normal_length_adam,
    # 'mlp_long_length_adam' : mlp_long_length_adam,

    # 'mlp_longest_adam' : mlp_longest_adam,
    # 'mlp_longest_adam_BN' : mlp_longest_adam_BN,
    # 'mlp_longest_adam_Droup' : mlp_longest_adam_Droup,
    # 'mlp_longest_adam_L2' : mlp_longest_adam_L2,

    # 'classifier_miniBatch_withBatchNorm' :
classifier_miniBatch_withBatchNorm,
    # 'classifier_adam_withBatchNorm' : classifier_adam_withBatchNorm,
    # 'classifier_momentum_withBatchNorm' : classifier_momentum_withBatchNorm,

    # 'classifier_miniBatch_withL2' : classifier_miniBatch_withL2,
    # 'classifier_adam_withL2' : classifier_adam_withL2,
    # 'classifier_momentum_withL2' : classifier_momentum_withL2,

    # 'classifier_miniBatch_withL2andBN' : classifier_miniBatch_withL2andBN,
    # 'classifier_adam_withL2andBN' : classifier_adam_withL2andBN,
    # 'classifier_momentum_withL2andBN' : classifier_momentum_withL2andBN,
    # 'classifier_miniBatch_withL2andBN_lossL2' :
classifier_miniBatch_withL2andBN_lossL2,
    # 'classifier_adam_withL2andBN_lossL2' :
classifier_adam_withL2andBN_lossL2,
    # 'classifier_momentum_withL2andBN_lossL2' :
classifier_momentum_withL2andBN_lossL2,
    # 'softmax_classifier_miniBatch' : softmax_classifier_miniBatch,
    # 'softmax_classifier_adam' : softmax_classifier_adam,
    # 'softmax_classifier_momentum' : softmax_classifier_momentum,

    # 'classifier_adam_lr5e-1' : classifier_adam_lr5e_1,
    # 'classifier_adam_lr5e-2' : classifier_adam_lr5e_2,
    # 'classifier_adam_lr5e-3' : classifier_adam_lr5e_3,
    # 'classifier_adam_lr5e-4' : classifier_adam_lr5e_4,

    # 'classifier_momentum_lr5e_1' : classifier_momentum_lr5e_1,
    # 'classifier_momentum_lr5e_2' : classifier_momentum_lr5e_2,
    # 'classifier_momentum_lr5e_3' : classifier_momentum_lr5e_3,
    # 'classifier_momentum_lr5e_4' : classifier_momentum_lr5e_4,
```

```
183
184     # 'classifier_miniBatch16' : classifier_miniBatch20,
185     # 'classifier_miniBatch32' : classifier_miniBatch50,
186     # 'classifier_miniBatch64' : classifier_miniBatch100,
187     # 'classifier_miniBatch128' : classifier_miniBatch200,
188     # 'classifier_miniBatch256' : classifier_miniBatch300,
189
190     # 'classifier_momentum16' : classifier_momentum20,
191     # 'classifier_momentum32' : classifier_momentum50,
192     # 'classifier_momentum64' : classifier_momentum100,
193     # 'classifier_momentum128' : classifier_momentum200,
194     # 'classifier_momentum256' : classifier_momentum300,
195
196     # 'classifier_adam16' : classifier_adam20,
197     # 'classifier_adam32' : classifier_adam50,
198     # 'classifier_adam64' : classifier_adam100,
199     # 'classifier_adam128' : classifier_adam200,
200     # 'classifier_adam256' : classifier_adam300,
201
202     # 'softmax_classifier_adam_lr5e_1' : softmax_classifier_adam_lr5e_1,
203     # 'softmax_classifier_adam_lr5e_2' : softmax_classifier_adam_lr5e_2,
204     # 'softmax_classifier_adam_lr5e_3' : softmax_classifier_adam_lr5e_3,
205     # 'softmax_classifier_adam_lr5e_4' : softmax_classifier_adam_lr5e_4,
206
207     # 'softmax_classifier_adam_beta03' : softmax_classifier_adam_beta03,
208     # 'softmax_classifier_adam_beta05' : softmax_classifier_adam_beta05,
209     # 'softmax_classifier_adam_beta07' : softmax_classifier_adam_beta07,
210     # 'softmax_classifier_adam_beta09' : softmax_classifier_adam_beta09,
211
212     # 'classifier_adam_beta03' : classifier_adam_beta03,
213     # 'classifier_adam_beta05' : classifier_adam_beta05,
214     # 'classifier_adam_beta07' : classifier_adam_beta07,
215     # 'classifier_adam_beta09' : classifier_adam_beta09,
216
217     # 'classifier_momentum_beta03' : classifier_momentum_beta03,
218     # 'classifier_momentum_beta05' : classifier_momentum_beta05,
219     # 'classifier_momentum_beta07' : classifier_momentum_beta07,
220     # 'classifier_momentum_beta09' : classifier_momentum_beta09,
221
222     # 'softmax_classifier_adam_lossL2' : softmax_classifier_adam_lossL2,
223
224     # 'classifier_adam_lossL2' : classifier_adam_lossL2,
225
226     # 'classifier_adam_dropout': classifier_adam_dropout,
227 }
228
229 def test(classifier:MLP, x_test=None, y_test=None
```

```
          , test_bsz=200, dataset_prepared=False):

    if not dataset_prepared:
        x_train, y_train, x_test, y_test = load_dataset(300, test_bsz)

    Scr = []
    for x, y in zip(x_test, y_test):
        cls = classifier.predict(x, bsz=test_bsz)
        y_true = []
        for yy in y:
            y_true.append(npp.argmax(np.asnumpy(yy)))

        score = 0.0
        for cl, gtd in zip(cls, y_true):
            if cl == gtd:
                score += 1
        score /= len(y_true)
        Scr.append(score)

    Scr = np.array(Scr)
    Scr = np.mean(Scr)
    print("Mean Score: " + str(Scr))
    return Scr

def train(x_train, y_train, x_valid, y_valid, x_test, y_test, classifier : MLP, model_name : str, optim : Optimizer, batch_size=300, test_batch=50, epochs=10):

    J = []
    ep = []
    scr = []
    Step = []
    steps = 0
    test_scores = 0.0

    for epoch in range(epochs):
        classifier.is_test = False
        with tqdm(total=len(x_train)) as t:
            for x, y in zip(x_train, y_train):
                    t.set_description(model_name + "_Epoch %i" %epoch)
                    classifier.is_test = False
                    steps += 1
                    loss = classifier.train_steps(x, y, batch_size, optim, epoch)
                    t.set_postfix(Loss='%.4f'%loss)
                    J.append(loss)
                    Step.append(steps)
```

```
274
275                     t.update(1)
276
277         # if epoch % 5 == 0:
278         classifier.is_test = True
279         score = test(classifier, x_valid, y_valid, test_batch, True)
280
281         # if isinstance(classifier, MLP):
282         #     classifier.save_weights(file_name= './' + model_name
    +'_epoch_'+str(epoch)+'.json')
283         scr.append(score)
284         ep.append(epoch)
285     #     log_vars = {
286     #         'steps':steps,
287     #         'score':scr,
288     #         'epoch':epoch,
289     #         'loss':J
290     #     }
291     # log_vars = json.dumps(log_vars)
292
293     # log_File = './'+ model_name + '_train_log.json'
294     # with open(log_File, 'a') as f:
295     #     f.write(log_vars)
296     #     print('Save Log Success!')
297
298     classifier.is_test = True
299     test_scores = test(classifier, x_test, y_test, test_bsz=1,
    dataset_prepared=True)
300
301     return Step, J, ep, scr, test_scores
302
303 def single_muti_train(epochs=10, batch_size=300, test_batch=50, lr=0.005,
    png_title=''):
304     Steps = []
305     Js = []
306     eps = []
307     scrs = []
308     test_scores = []
309     x_train, y_train, x_valid, y_valid, x_test, y_test =
    load_dataset(batch_size, test_batch)
310     # x_test, y_test = selfmake_dataset_load()
311
312     for classifier_name, classifier in zip(list(classifiers.keys()),
    list(classifiers.values())):
313         if math.fabs(lr-5e-3) > 1e-6:
314             classifier.lr = lr
315
```

```
316         # 进行优化器判断
317         if classifier_name.count('miniBatch'):
318             optim = Optimizer(classifier, lr=classifier.lr)
319         elif classifier_name.count('adam'):
320             optim = Optimizer_Adam(classifier, lr=classifier.lr,
    beta=classifier.beta)
321         else:
322             optim = Optimizer_Momentum(classifier, lr=classifier.lr,
    beta=classifier.beta)
323 
324         Step, J, ep, scr, test_score = train(x_train, y_train, x_valid,
    y_valid, x_test, y_test, classifier, classifier_name, optim,
    batch_size=batch_size, epochs=epochs)
325 
326         Steps.append(Step)
327         Js.append(J)
328         eps.append(ep)
329         scrs.append(scr)
330         test_scores.append(test_score)
331 
332     plt.figure()
333     # loss map
334     for Step, J, classifier_name in zip(Steps, Js, list(classifiers.keys())):
335         plt.plot(Step, [j.get() for j in J], label=classifier_name)
336     plt.legend(loc='upper right')
337     plt.xlabel("steps:")
338     plt.ylabel("loss:")
339     plt.savefig(png_title + "_contrast_train_loss.png")
340     plt.show()
341 
342     plt.figure()
343     np.set_printoptions(precision=4)
344     plt.gca().yaxis.set_major_formatter(ticker.FormatStrFormatter('%.4f'))
345     # score map
346     for ep, scr, classifier_name in zip(eps, scrs, list(classifiers.keys())):
347         plt.plot(ep, [sc.get() for sc in scr], label=classifier_name)
348         for x, y in zip(ep, scr):
349             plt.text(x, y.get(), '%.4f'%y.get(),fontsize=8)
350     plt.legend(loc='upper right')
351     plt.xlabel("epochs:")
352     plt.ylabel("score:")
353 
354     plt.savefig(png_title + "_contrast_train_scr.png")
355     plt.show()
356 
357     plt.figure()
358     classifier_names = list(classifiers.keys())
```

```
359     plt.plot(range(len(list(classifiers.keys()))), [item.get() for item in
    test_scores])
360     plt.gca().yaxis.set_major_formatter(ticker.FormatStrFormatter('%.4f'))
361     for x, y in zip(list(range(len(list(classifiers.keys())))), test_scores):
362         plt.text(x, y.get(), y.get(),fontsize=8)
363     plt.xticks(range(len(list(classifiers.keys()))), classifier_names,
    fontsize=6)
364     plt.legend(loc='upper right')
365     plt.xlabel("epochs:")
366     plt.ylabel("score:")
367     plt.savefig(png_title + "_contrast_test_scr.png")
368     plt.show()
369
370 def muti_contrast_train(epochs=10, batch_size=300, test_batch=50, lr=0.005,
    png_title=''):
371     Steps = []
372     Js = []
373     eps = []
374     scrs = []
375     exec_list = []
376
377     if isinstance(batch_size, list):
378         Batch_sizes = batch_size.copy()
379         X_train, Y_train, X_valid, Y_valid = [], [], [], []
380         for batch_size in Batch_sizes:
381             x_train, y_train, x_valid, y_valid, x_test, y_test =
    load_dataset(batch_size, test_batch)
382             X_train.append(x_train)
383             Y_train.append(y_train)
384             X_valid.append(x_valid)
385             Y_valid.append(y_valid)
386
387         for x_train, y_train, x_valid, y_valid, bsz, classifier_name,
    classifier in zip(X_train, Y_train, X_valid, Y_valid, Batch_sizes,
388
          list(classifiers.keys()), list(classifiers.values())):
389             # x_test, y_test = selfmake_dataset_load()
390             pool = ThreadPoolExecutor(max_workers=8)
391
392             if math.fabs(lr-5e-3) > 1e-6:
393                 classifier.lr = lr
394
395             # 进行优化器判断
396             if classifier_name.count('miniBatch'):
397                 optim = Optimizer(classifier, lr=classifier.lr)
398             elif classifier_name.count('adam'):
```

```
                optim = Optimizer_Adam(classifier, lr=classifier.lr,
beta=classifier.beta)
            else:
                optim = Optimizer_Momentum(classifier, lr=classifier.lr,
beta=classifier.beta)

            future = pool.submit(train,x_train, y_train, x_valid, y_valid,
x_test, y_test, classifier, classifier_name, optim, batch_size=bsz,
epochs=epochs)
            exec_list.append(future)
        wait(exec_list, return_when=ALL_COMPLETED)

    else:
        x_train, y_train, x_valid, y_valid, x_test, y_test =
load_dataset(batch_size, test_batch)
        # x_test, y_test = selfmake_dataset_load()

        pool = ThreadPoolExecutor(max_workers=8)
        idx = 0
        for classifier_name, classifier in zip(list(classifiers.keys()),
list(classifiers.values())):
            if math.fabs(lr-5e-3) > 1e-6:
                classifier.lr = lr
            # 进行优化器判断
            if classifier_name.count('miniBatch'):
                optim = Optimizer(classifier, lr=classifier.lr)
            elif classifier_name.count('adam'):
                optim = Optimizer_Adam(classifier, lr=classifier.lr,
beta=classifier.beta)
            else:
                optim = Optimizer_Momentum(classifier, lr=classifier.lr,
beta=classifier.beta)

            future = pool.submit(train,x_train, y_train, x_valid, y_valid,
x_test, y_test, classifier, classifier_name, optim, batch_size=batch_size,
epochs=epochs)
            exec_list.append(future)

        wait(exec_list, return_when=ALL_COMPLETED)

    test_scores = []
    for exec in exec_list:
        Step, J, ep, scr, test_score = exec.result()

        Steps.append(Step)
        Js.append(J)
        eps.append(ep)
```

```
436         scrs.append(scr)
437         test_scores.append(test_score)
438 
439     plt.figure()
440     # loss map
441     for Step, J, classifier_name in zip(Steps, Js, list(classifiers.keys())):
442         plt.plot(Step, [j.get() for j in J], label=classifier_name)
443     plt.legend(loc='upper right')
444     plt.xlabel("steps:")
445     plt.ylabel("loss:")
446     plt.savefig(png_title + "_contrast_train_loss.png")
447     plt.show()
448 
449     plt.figure()
450     np.set_printoptions(precision=4)
451     plt.gca().yaxis.set_major_formatter(ticker.FormatStrFormatter('%.4f'))
452     # score map
453     for ep, scr, classifier_name in zip(eps, scrs, list(classifiers.keys())):
454         plt.plot(ep, [sc.get() for sc in scr], label=classifier_name)
455         for x, y in zip(ep, scr):
456             plt.text(x, y.get(), '%.4f'%y.get(),fontsize=8)
457     plt.legend(loc='upper right')
458     plt.xlabel("epochs:")
459     plt.ylabel("score:")
460 
461     plt.savefig(png_title + "_contrast_train_scr.png")
462     plt.show()
463 
464     plt.figure()
465     classifier_names = list(classifiers.keys())
466     plt.plot(range(len(list(classifiers.keys()))), [item.get() for item in
    test_scores])
467     plt.gca().yaxis.set_major_formatter(ticker.FormatStrFormatter('%.4f'))
468     for x, y in zip(list(range(len(list(classifiers.keys())))), test_scores):
469         plt.text(x, y.get(), y.get(),fontsize=8)
470     plt.xticks(range(len(list(classifiers.keys()))), classifier_names,
    fontsize=6)
471     plt.legend(loc='upper right')
472     plt.xlabel("epochs:")
473     plt.ylabel("score:")
474     plt.savefig(png_title + "_contrast_test_scr.png")
475     plt.show()
476 
477 if __name__ == '__main__':
478     muti_contrast_train(15, png_title='MLP_OverL2DroupRG', lr=0.0005,
    batch_size=64)
479     # single_muti_train(15, png_title='MLP_WidthRG', lr=0.0005, batch_size=64)
```

此部分为实验主要运行脚本，根据实验需要，对不同配置的分类器模型进行初始化添加到 **分类器集合** 词典中，进行运行即可自动调用分类器集合中所有的分类器模型进行训练与验证测试。训练过程中会保存训练日志以及模型权重（MLP），方便需要时调用。

在运行中，已将每步损失以及每轮测试得分（准确率）进行了保存，在最后将会把需要进行对比绘制的图像保存。

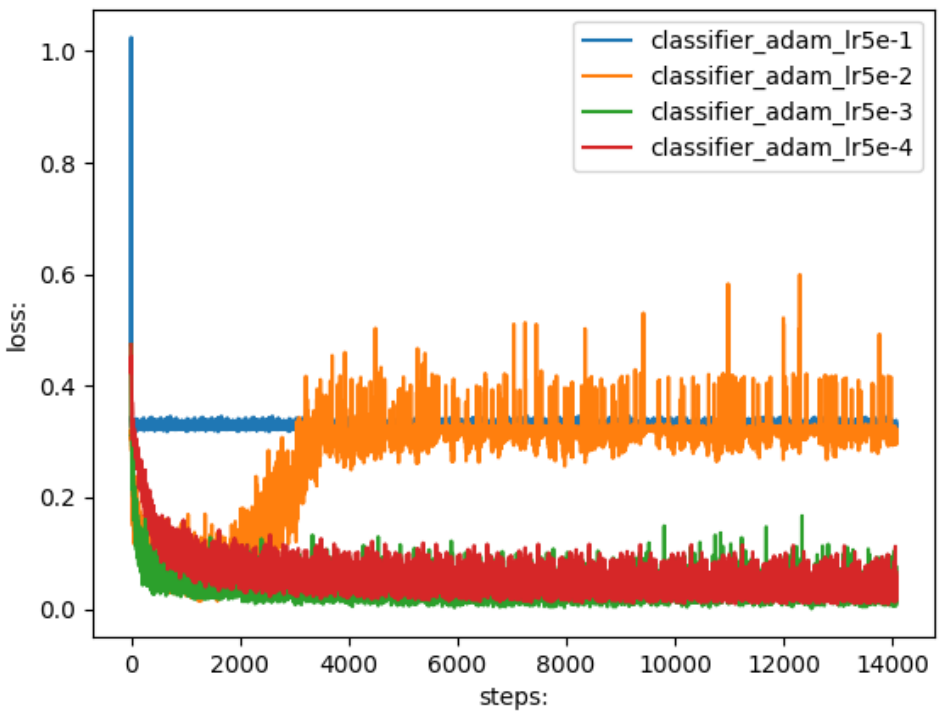
为加速训练，更快得到实验结果，对训练进行了多线程运行处理，将为每一个分类器的训练分配一个线程池线程，等待最后训练完毕调取所有得到结果。

## 3、具体实验步骤以及步骤结果分析：

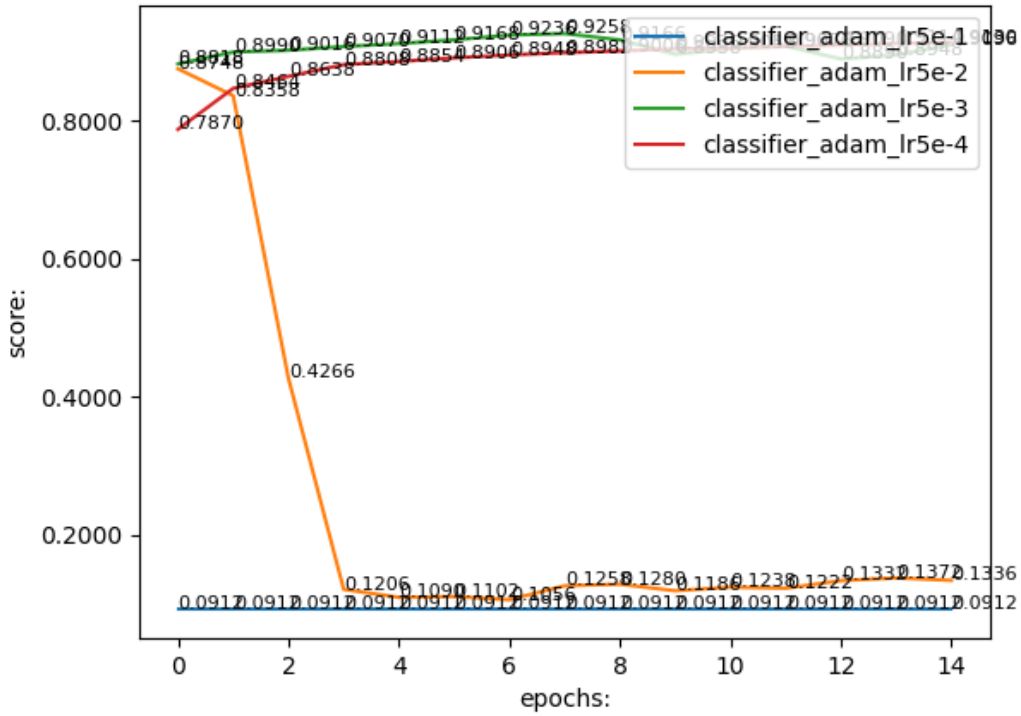
**默认在所有的实验中，我们将参数随机初始化，初始化种子确定为 2023**

### ① 分类器基本参数调校实验：

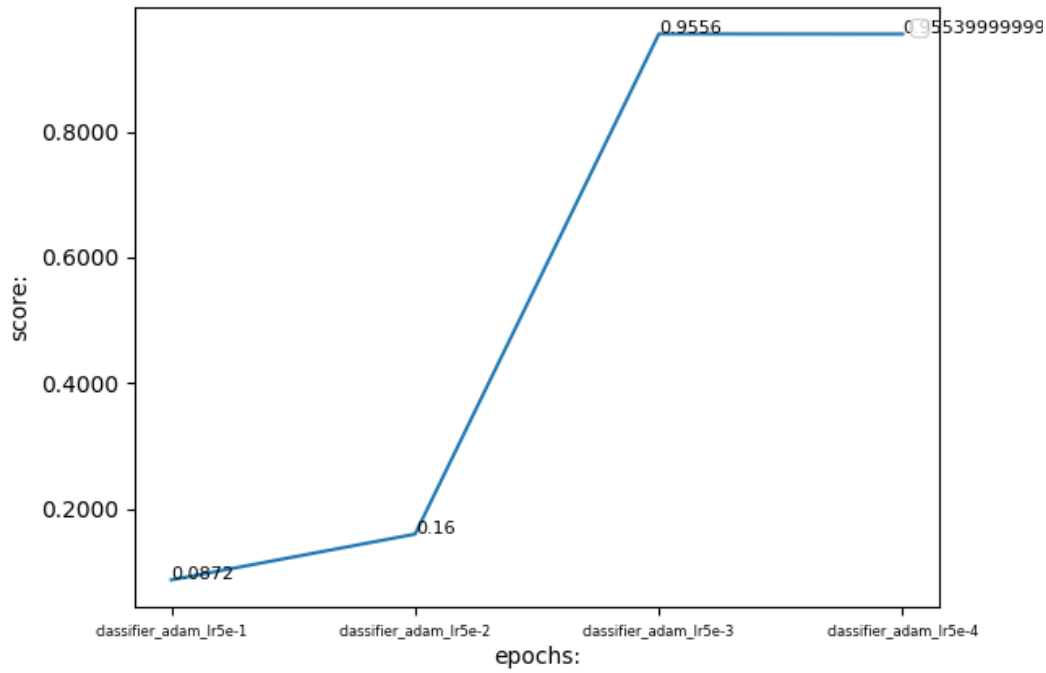
1. **学习率调校：**
   a. 在学习率调校中，统一采用 Adam 优化方法，BatchSize=64
   b. 都采用 MLP 分类器




学习率调校 Loss 图




学习率调校验证集 Score 图




学习率调校测试集 Score 图

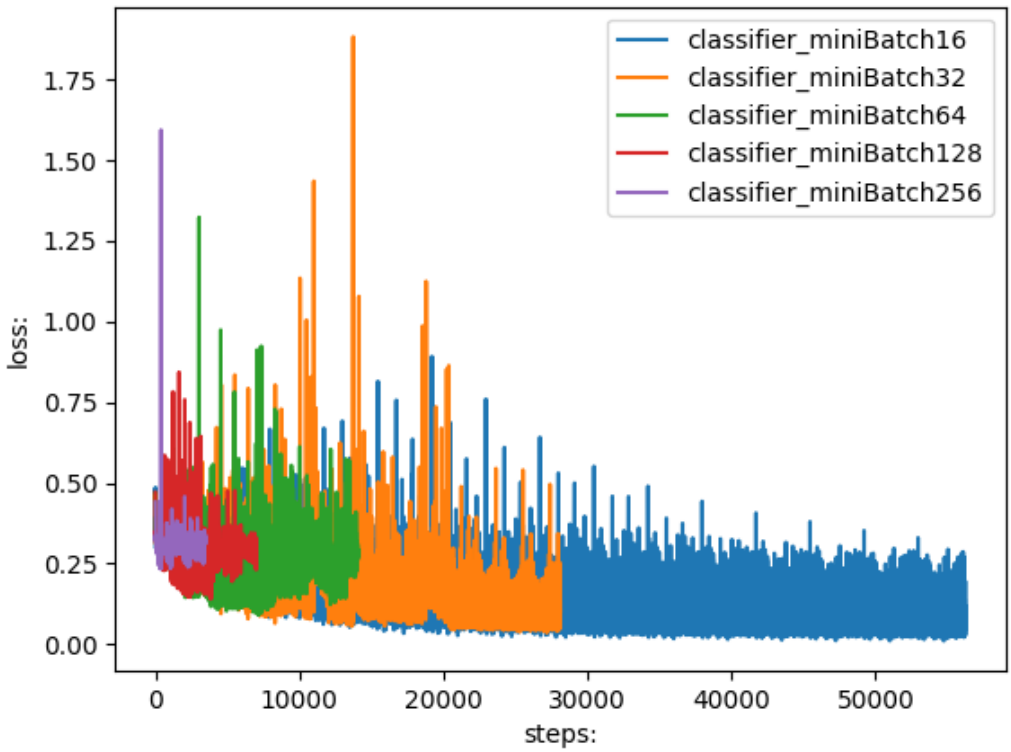
**结果分析：**

学习率过大时（5e-1，5e-2）情况下，损失下降过程有明显的下限阈值，且可能伴随不稳定的抖动，将导致模型权重无法学习到更优解，在梯度下降中反复徘徊。

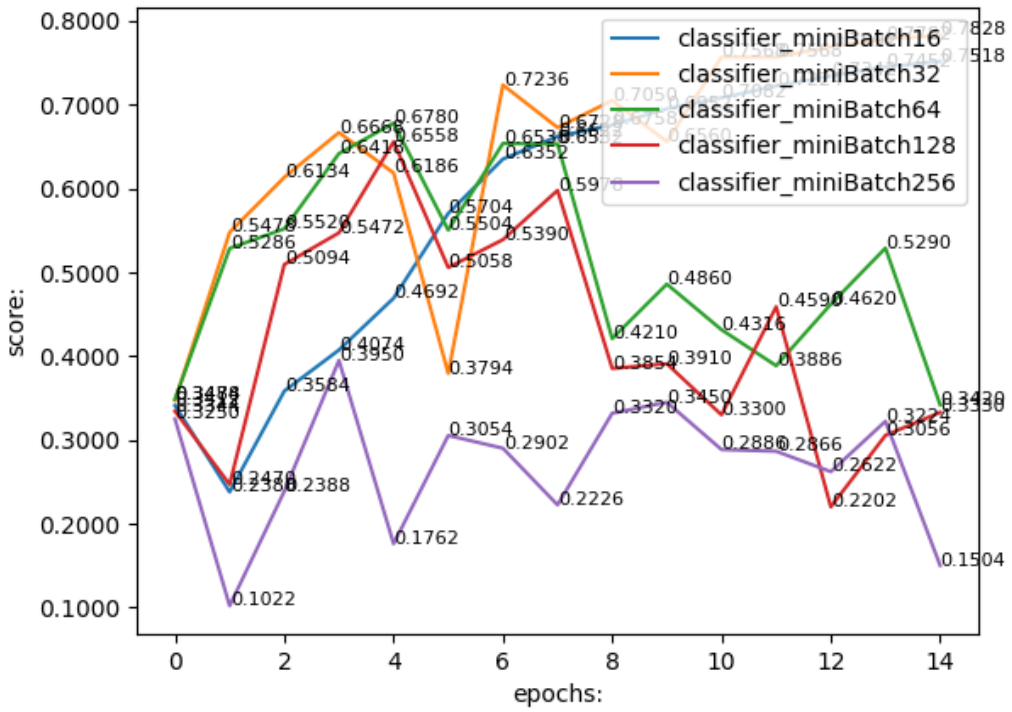
随着学习率的增大，模型损失下降速率也在增大，收敛速度变快，观察 lr=5e-3，5e-4 结果可知，lr=5e-3 时更快收敛到较高的验证集得分。

从测试集结果看，也能证明过高的学习率将导致模型学习效果变差。

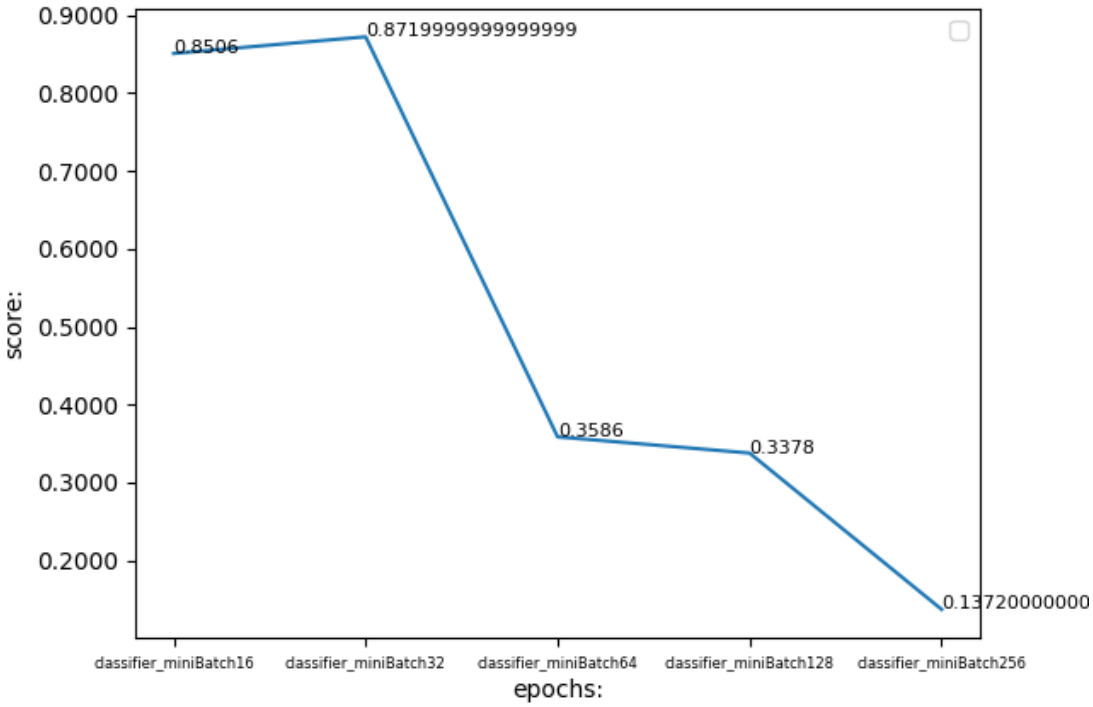
2. **batch-size 调校：**
   a. 在调校中，分别对三种下降方法进行了测试
   b. 学习率都固定为 5e-4
   c. 都采用 MLP 分类器




mini-Batch 下降方法下不同 Batch-size 损失




mini-Batch 下降方法下不同 Batch-size 验证集得分



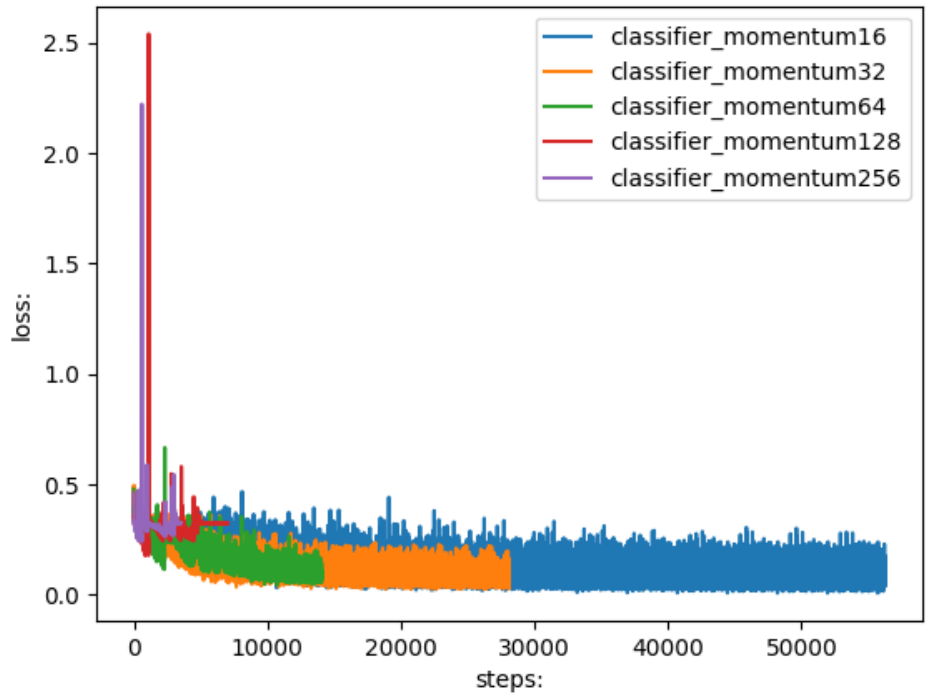

mini-Batch 下降方法下不同 Batch-size 测试集得分

**结果分析：**

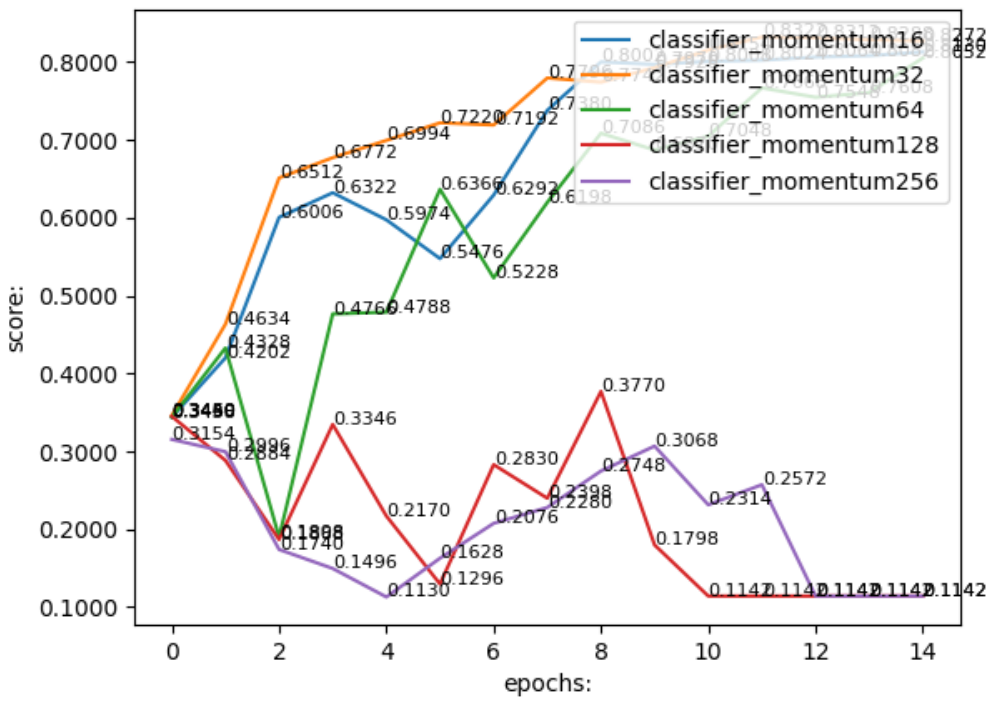
观察损失下降图，明显可知随批次大小增加，模型损失收敛速度增大，收敛下限提高。过大的批次大小导致模型无法达到更优解，不稳定性增加。但较小 batch-size 也带来损失的反复抖动，收敛不够平稳。

观察验证集每轮训练得分，batch-size=16 时上升趋势较为稳定，为 32 时总体呈现上升趋势但震荡幅度较大，不够稳定。其他情况下震荡较为明显，且得分有降低趋势。

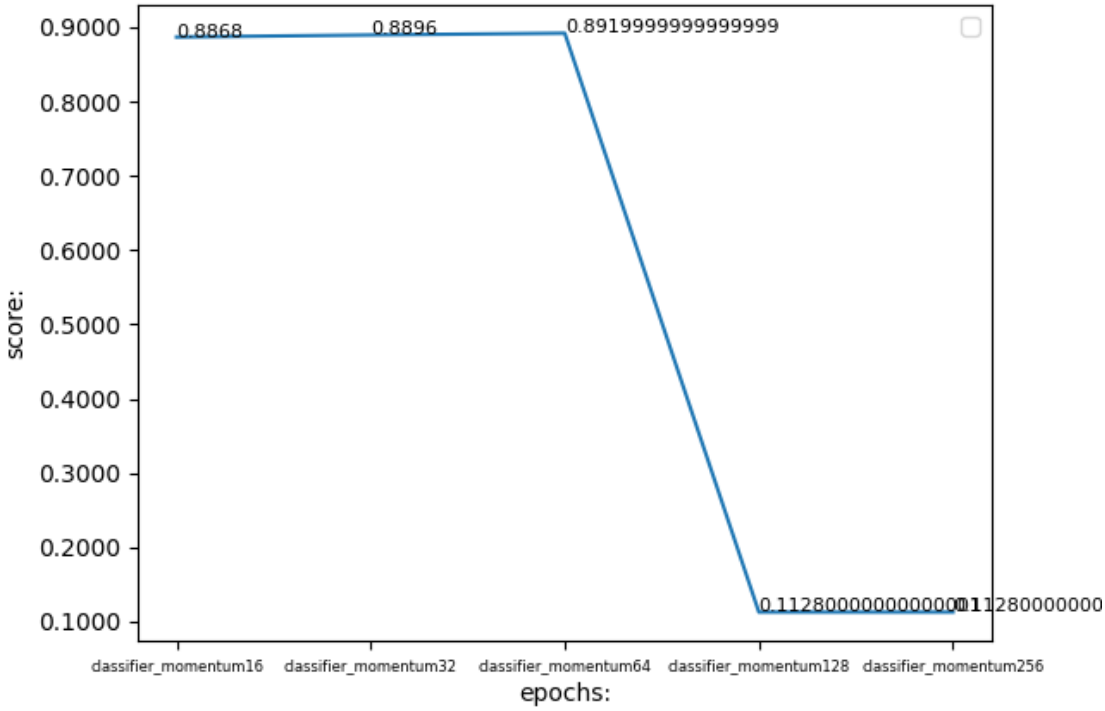
从测试集得分情况来看，batch-size=32 时得分最高，这可能是因为收敛速度更快，其学习时提取到的样本特征更加广泛等原因。大于 32 的 batch-size 的情况下，模型学习效果较差，得分较低。




momentum 下降方法下不同 Batch-size 下损失



momentum 下降方法下不同 Batch-size 下验证集分数



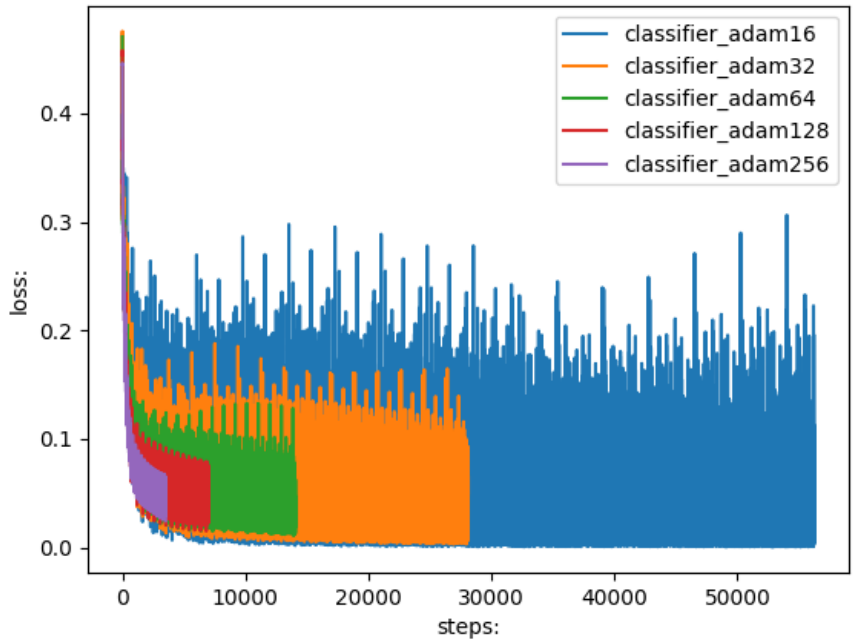

momentum 下降方法下不同 Batch-size 下测试集分数

**结果分析：**

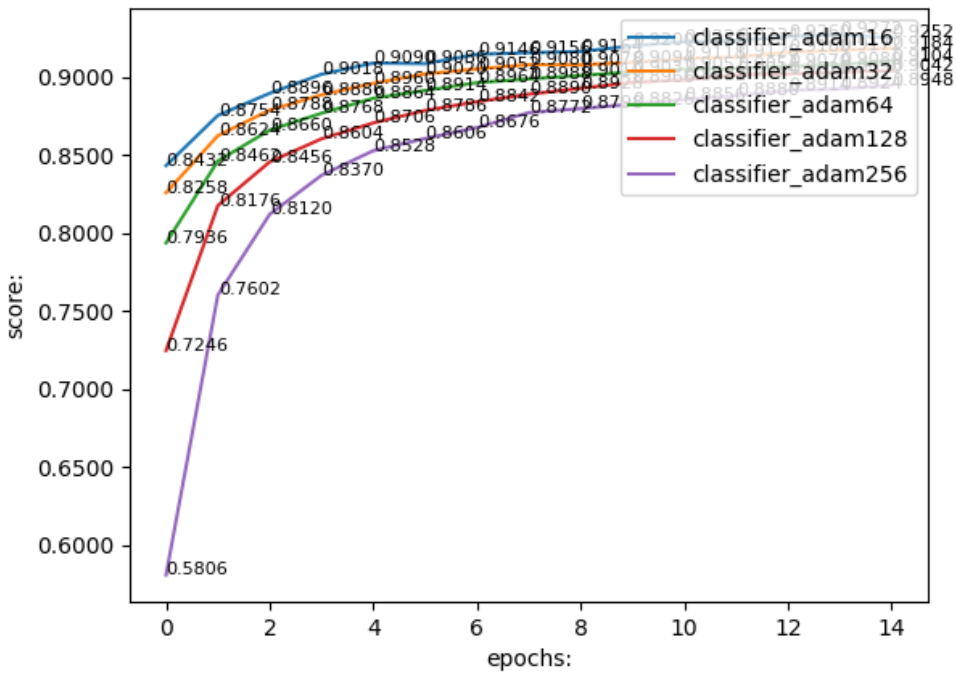
损失下降方面类似 mini-batch 的调试下降过程，但 momentum 优化方法对于 batch-size 的“容忍度”更高，即使 batch-size=64 情况下也可以正常损失下降。

观察验证集每轮训练得分，batch-size=32 时上升趋势最稳定，为 64 时总体上升，但震荡幅度较大，不够稳定。batch-size=16 时，验证集分数有明显波动，可能是因为 batch-size 较小，受噪声特征影响更大，导致了不稳定。

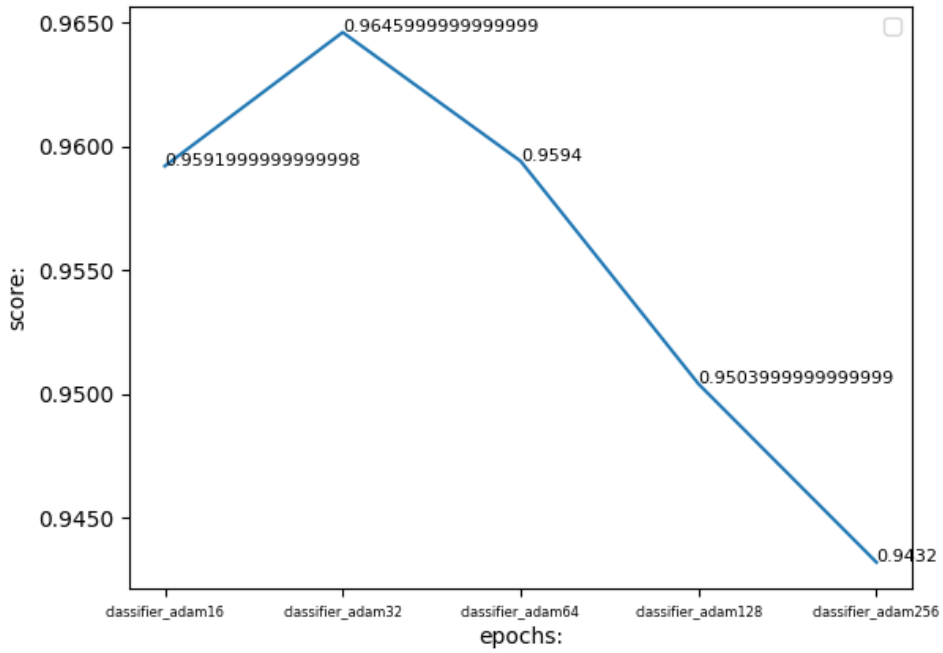
从测试集得分情况来看，batch-size=64 时得分最高，但与 16，32 时并无过多明显差异，可能由于其学习到了更广泛的样本特征以及收敛速度更快。




Adam 下降方法下不同 Batch-size 下损失




Adam 下降方法下不同 Batch-size 下验证集分数




Adam 下降方法下不同 Batch-size 下自制测试集分数

**结果分析：**

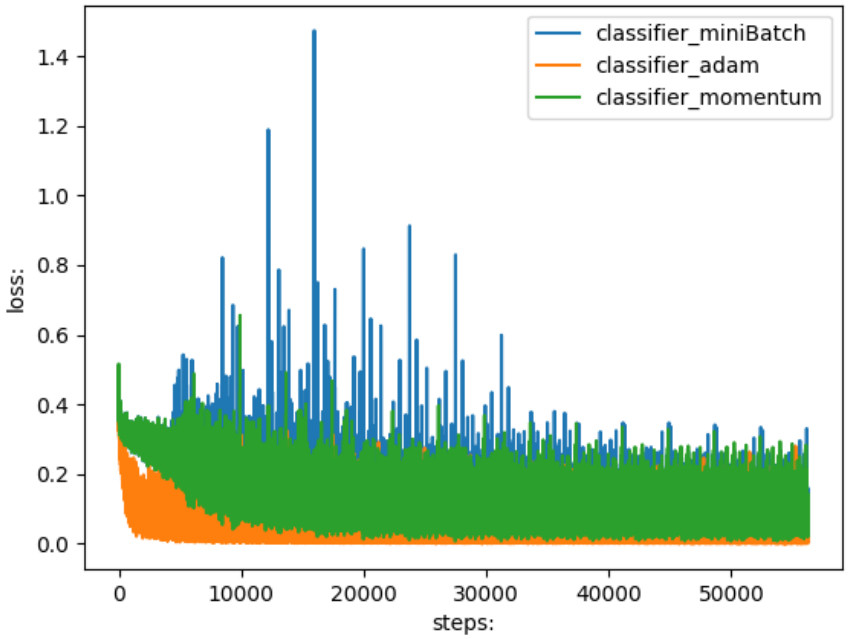
Adam 下降方法下，不同 batch-size 都能表现出稳定的损失下降和验证集分数增长。这可能是由于 Adam 优化方法具有更强的学习步长自适应性。

但 batch-size 较小时，明显可见 loss 的反复抖动，收敛不够平稳。

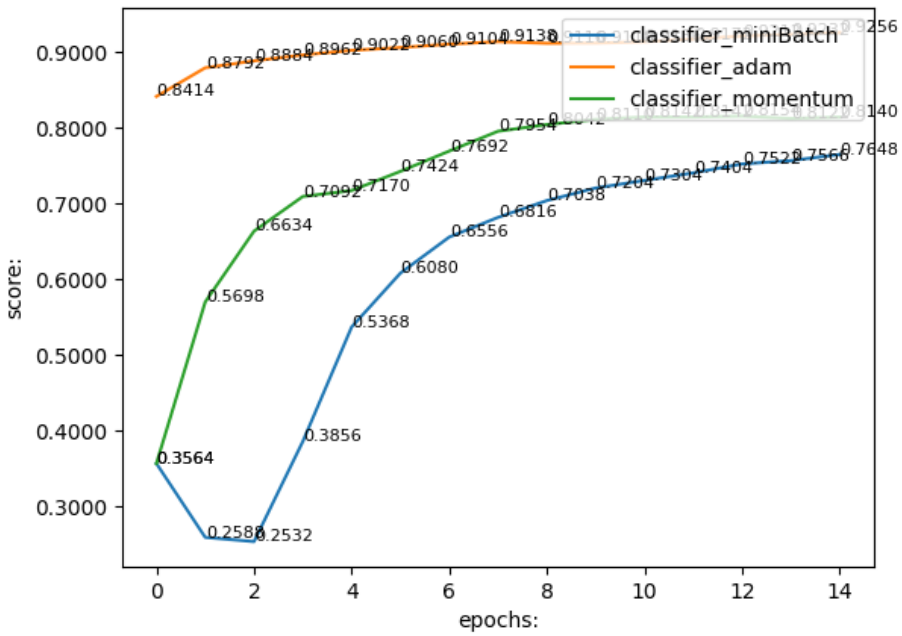
从测试集观察各 batch-size 情况下都有 0.94 以上的得分，可得出 Adam 优化方法对 batch-size 包容性更强的结论。

### ② 分类器优化方法对比实验（普通 mini-batch，adam，momentum）：

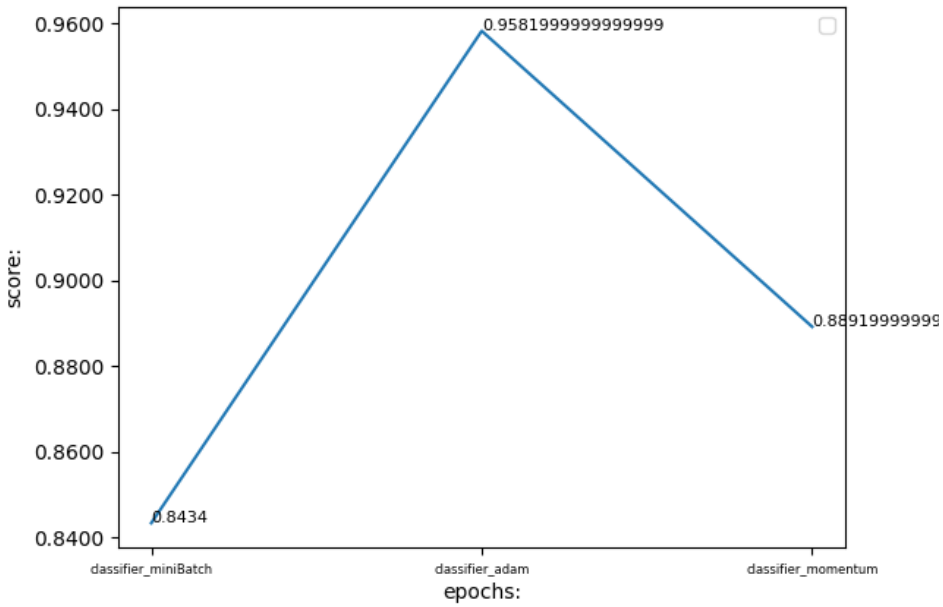
BatchSize=16, lr=5e-4:




各优化器损失图




各优化器验证集得分图




各优化器测试集得分图

**结果分析：**

观察损失下降图，明显可知 Adam 方法收敛速度 > momentum 方法收敛速度 > 普通 mini-batch 方法收敛速度

观察验证集每轮训练得分，Adam 方法每轮得分 > momentum 方法每轮得分 > 普通 mini-batch 方法每轮得分

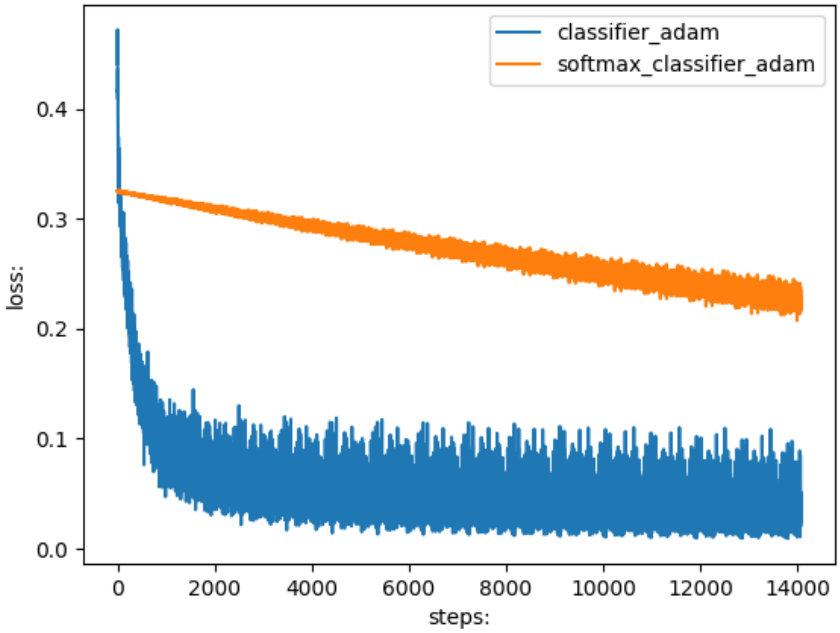
且从分数上升趋势来看 Adam 方法和 momentum 方法都较为平稳，mini-batch 方法经历过波动，不够稳定。

从测试集得分情况也同样 Adam 最优，momentum 次之，mini-batch 最次。

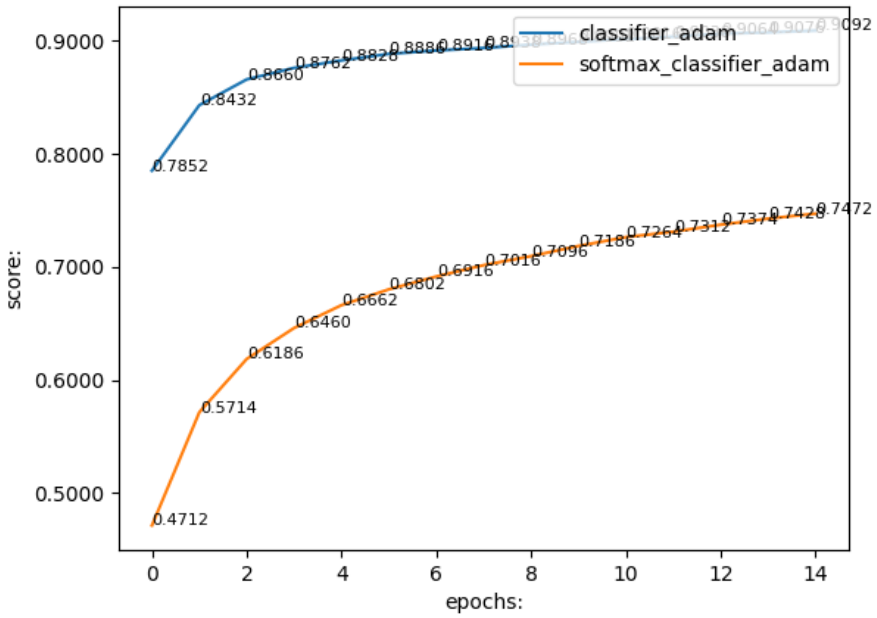
### ③ 分类器对比实验（Softmax - MLP）：

分类器对比实验采用 Softmax 分类器与 MLP 分类器进行对比

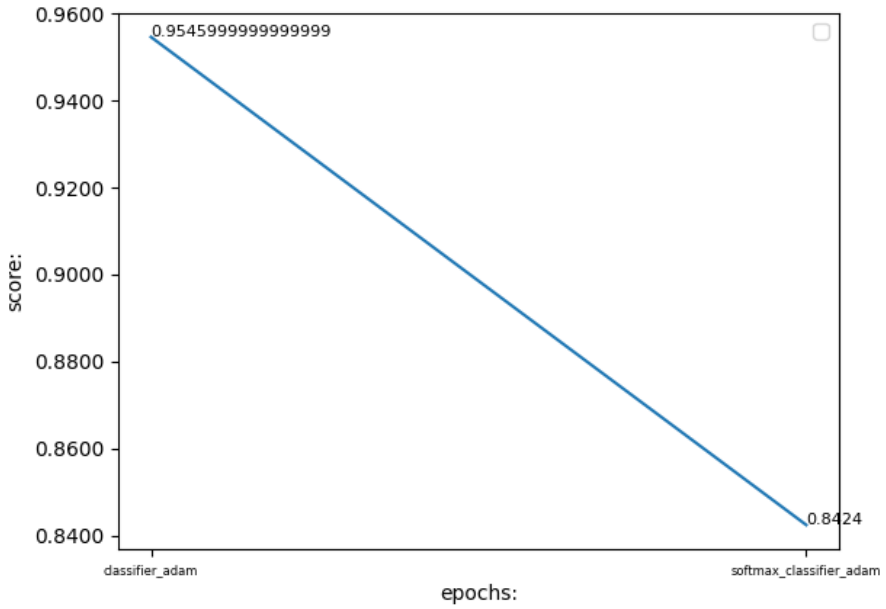
都采用 Adam 优化方法， BatchSize=64， lr=5e-4：




对比损失图




对比验证集得分图




对比测试集得分图

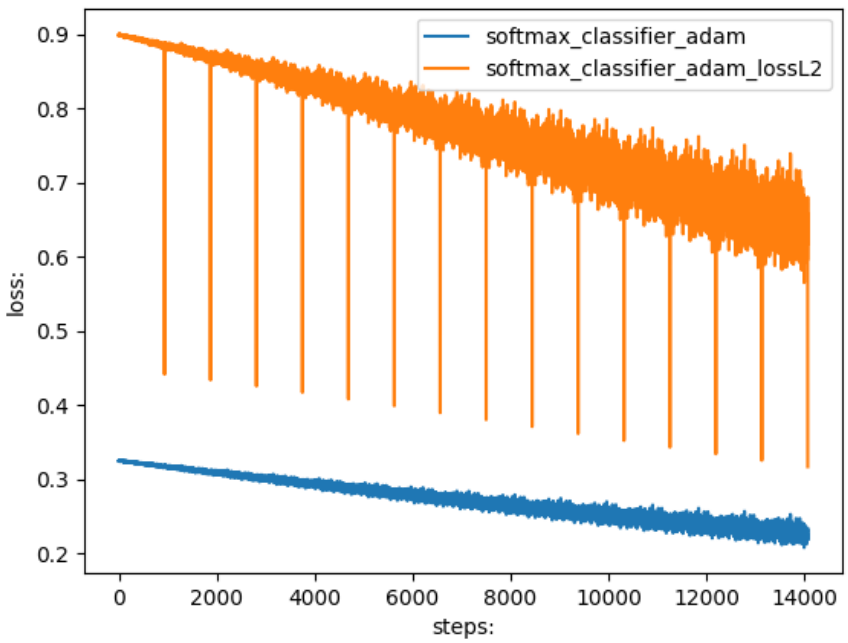
**结果分析：**

从各图明显可知 MLP 各方面性能明显强于普通 Softmax 分类器，收敛速度更快，准确度更高等等。

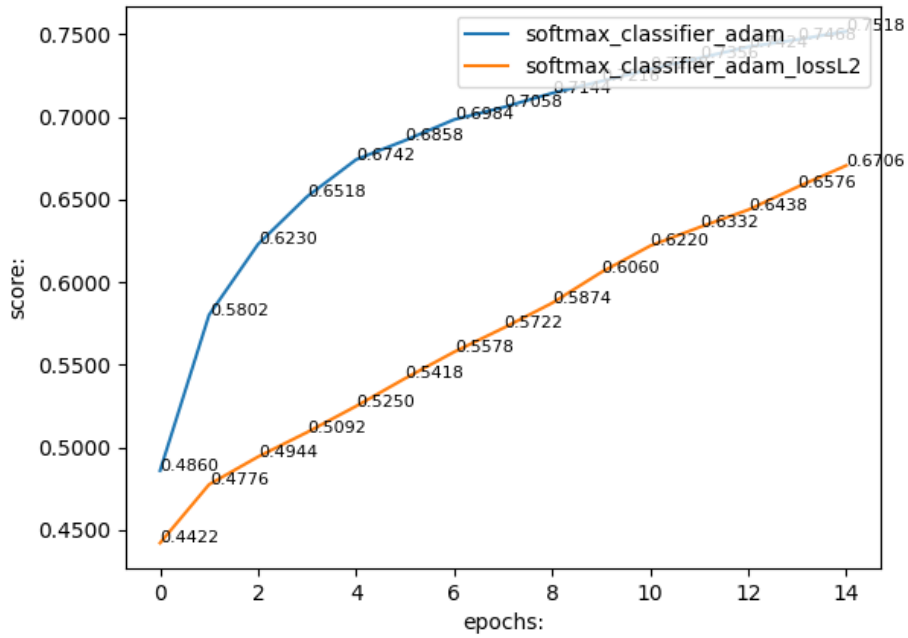
这可能是由于 MLP 可学习参数更多，且每层神经元有非线性激活函数，更易拟合各类特征等原因所导致。

### ④ 分类器损失函数对比实验（CrossEntropy - L2）：

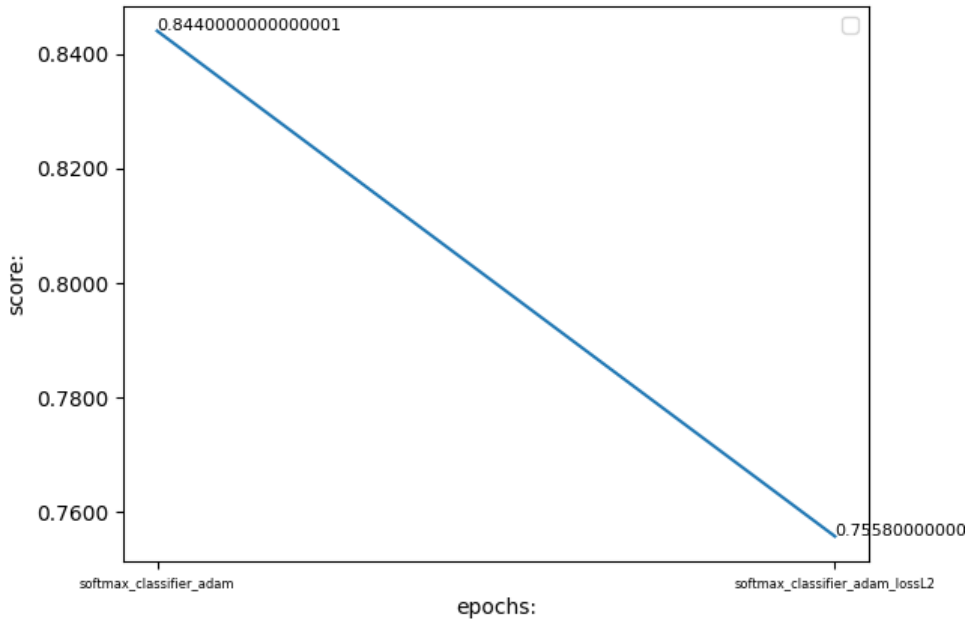
1. SoftMax 分类器：



交叉熵-L2 损失 下降损失对比图

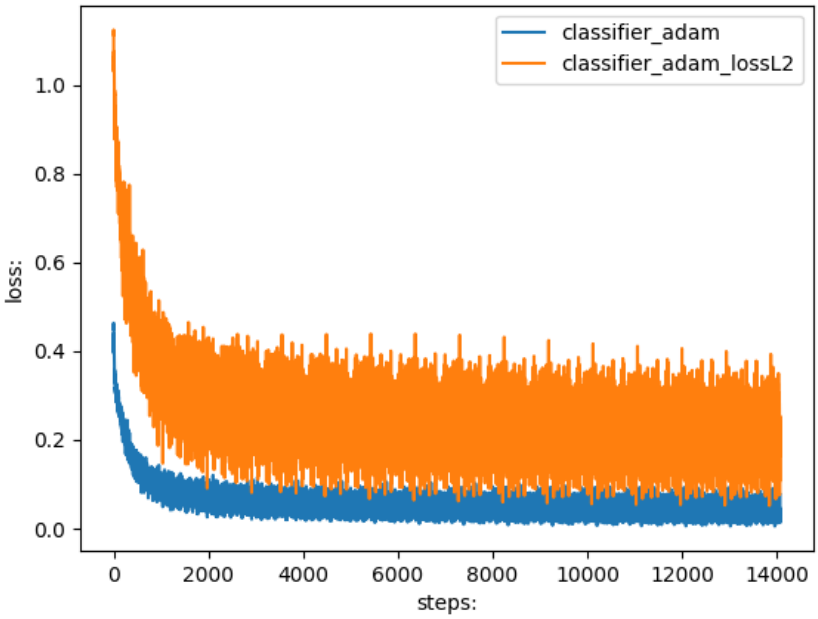



交叉熵-L2 损失 验证集得分对比图

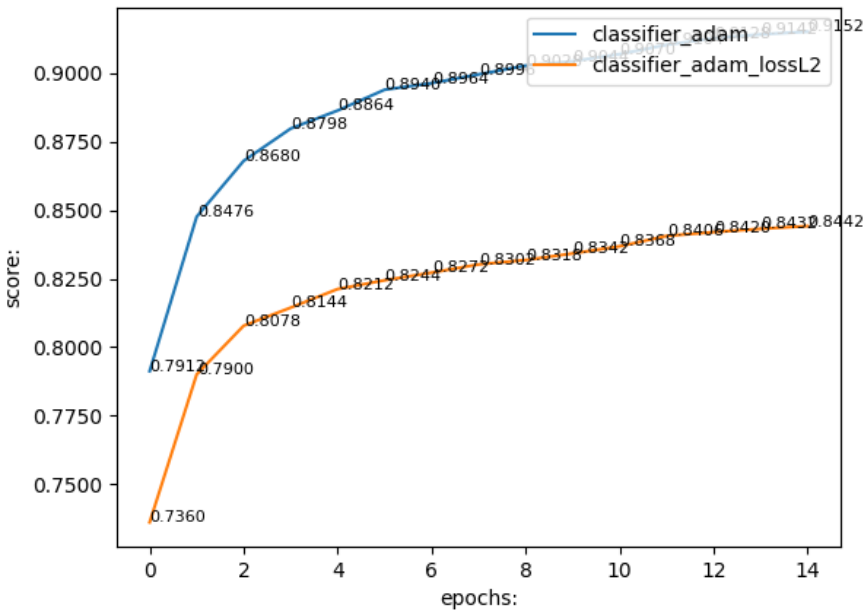



交叉熵-L2 损失 测试集得分对比图

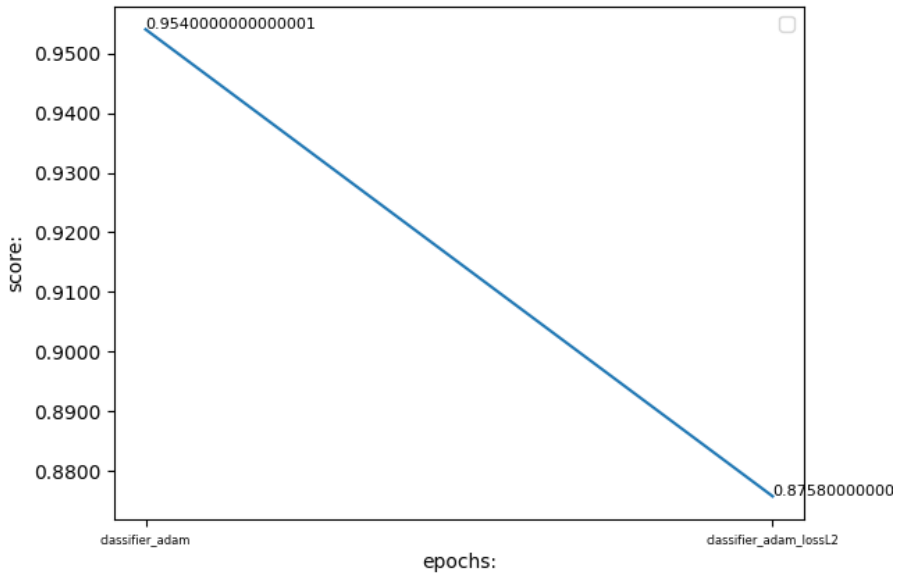
2. MLP 分类器：




交叉熵-L2 损失 下降损失对比图




交叉熵-L2 损失 验证集得分对比图




交叉熵-L2 损失 测试集得分对比图

**结果分析：**

在 Softmax 分类器上和 MLP 分类器上都有如下特征：

损失下降速度上，交叉熵损失明显优于了 L2 损失，收敛速度更快，震荡幅度更小，更平稳。

从训练所得精度上来看交叉熵损失也优于 L2 损失。

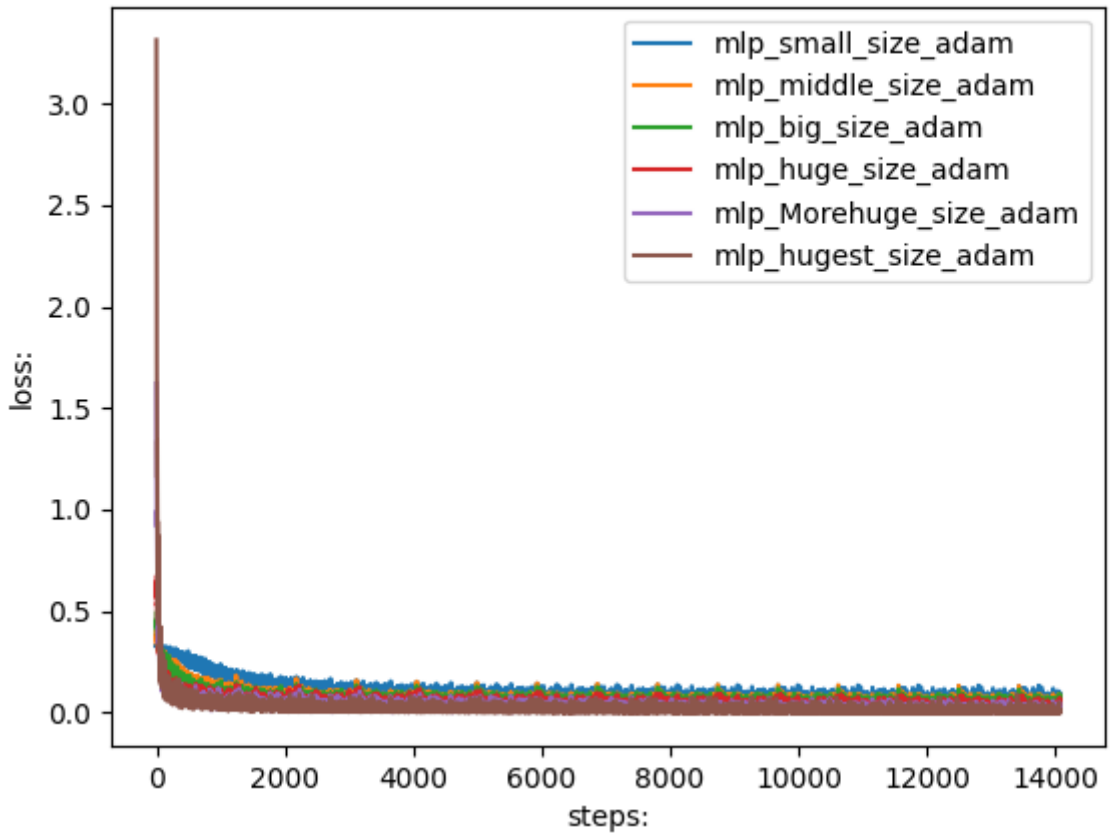
可得出在手写数字识别分类问题上，交叉熵损失更优于 L2 损失的结论。

### ⑤ MLP 网络结构影响对比：

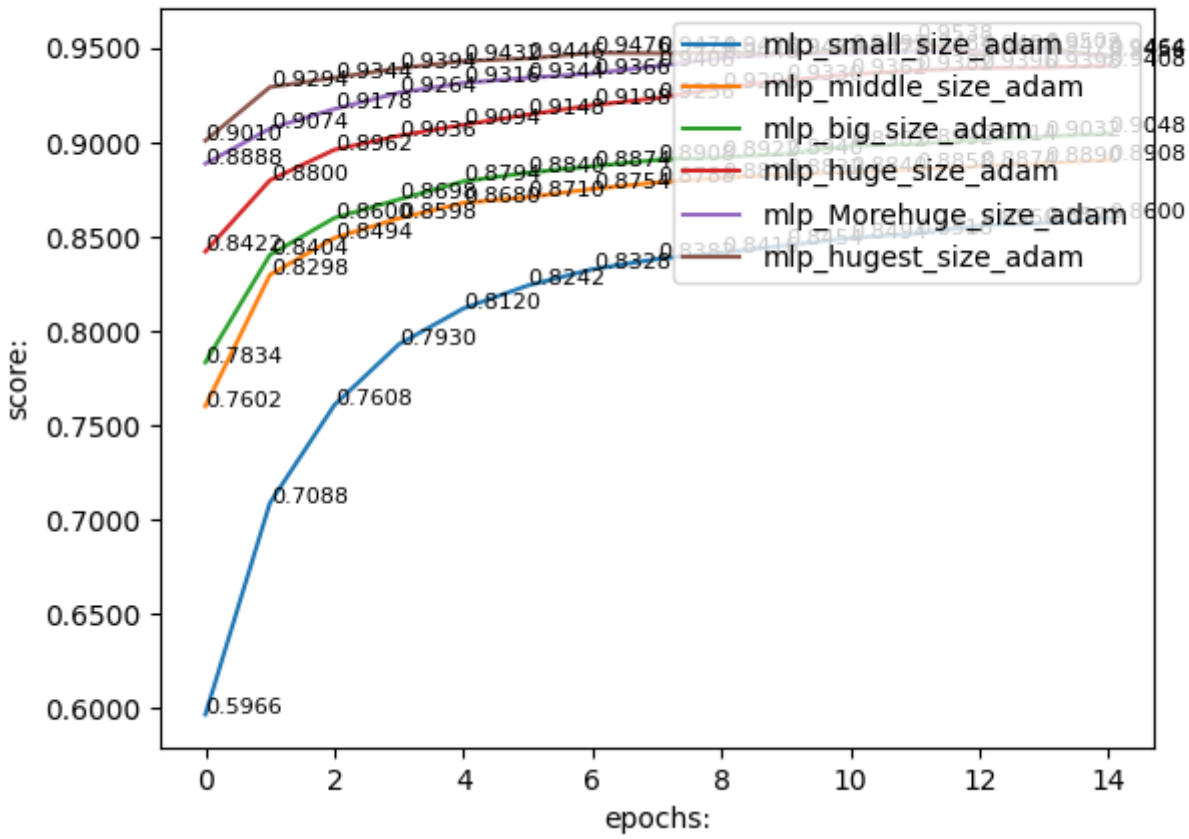
在该实验中，默认采用 Adam 优化方法，BatchSize=64， lr=5e-4

1. 网络宽度测试
   a. 将每层宽度设为如下数据：
      i. [28*28, 64, 10]（small）
      ii. [28*28, 128, 25]（middle）
      iii. [28*28, 256, 30]（huge）
      iv. [28*28, 512, 64]（Morehuge）
      v. [28*28, 1024, 128]（hugest）

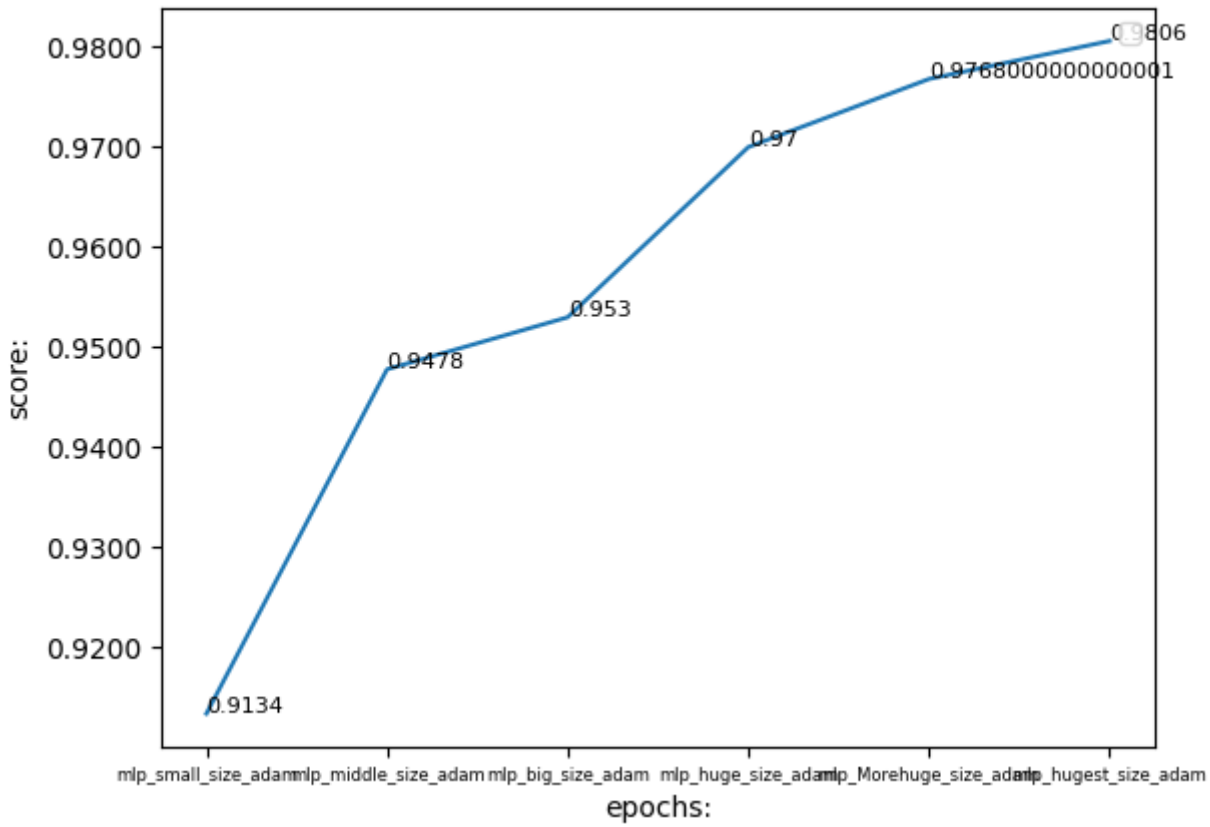
      有如下结果：


mlp_small_size_adam
mlp_middle_size_adam
mlp_big_size_adam
mlp_huge_size_adam
mlp_Morehuge_size_adam
mlp_hugest_size_adam
loss:
steps:
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
2000
4000
6000
8000
10000
12000
14000

损失图


mlp_small_size_adam
mlp_middle_size_adam
mlp_big_size_adam
mlp_huge_size_adam
mlp_Morehuge_size_adam
mlp_hugest_size_adam
score:
epochs:
0.6000
0.6500
0.7000
0.7500
0.8000
0.8500
0.9000
0.9500
0
2
4
6
8
10
12
14
0.5966
0.7088
0.7608
0.7930
0.8120
0.8242
0.8328
0.7602
0.7834
0.8298
0.8404
0.8494
0.8598
0.8680
0.8710
0.8754
0.8422
0.8800
0.8962
0.9036
0.9094
0.9148
0.9198
0.8888
0.9010
0.9074
0.9178
0.9264
0.9316
0.9344
0.9366
0.9294
0.9344
0.9394
0.9430
0.9446
0.9470

验证集得分图




测试集得分图

**结果分析：**

观察损失下降图，验证集得分图，随着网络神经元数量的增加，收敛速度增加，且精度更高。

从测试集得分来看，在该规模下，网络神经元数量与准确度成正比，hugest 网络达百分之 98 以上的准确率。

可有结论：在一定层次，一定规模下，网络神经元数量会影响模型能力，且成正比。

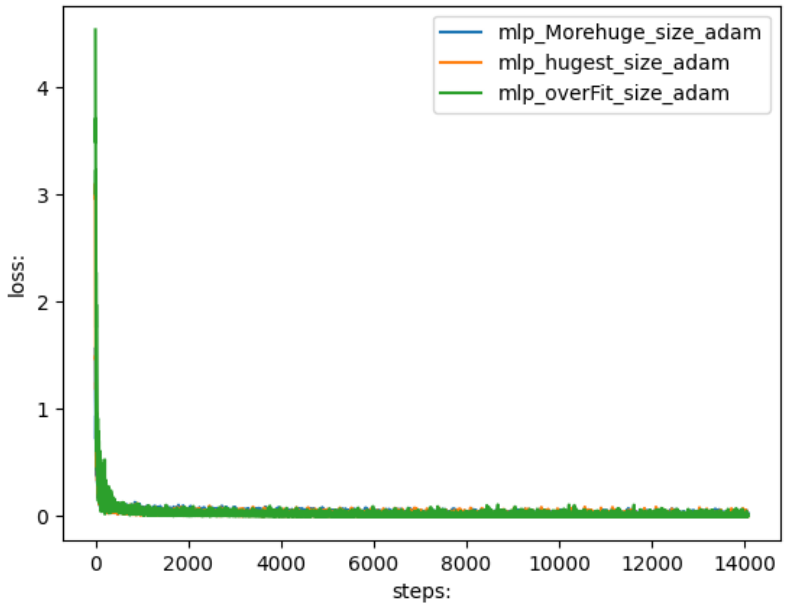
2. 尝试观察到过拟合现象

   a. 将网络每层神经元数量按如下配置：

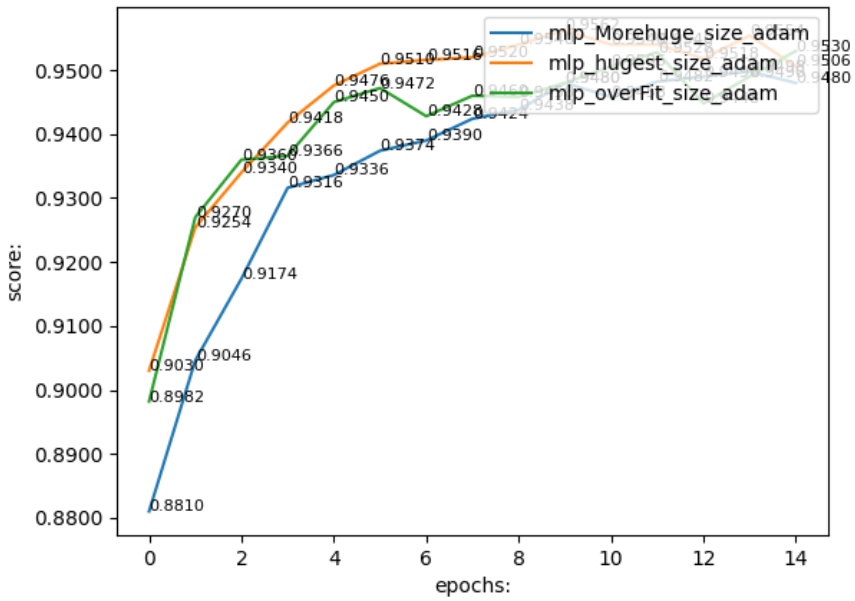
      i. [28*28, 1024, 128]（Morehuge）

      ii. [28*28, 2048, 256]（hugest）

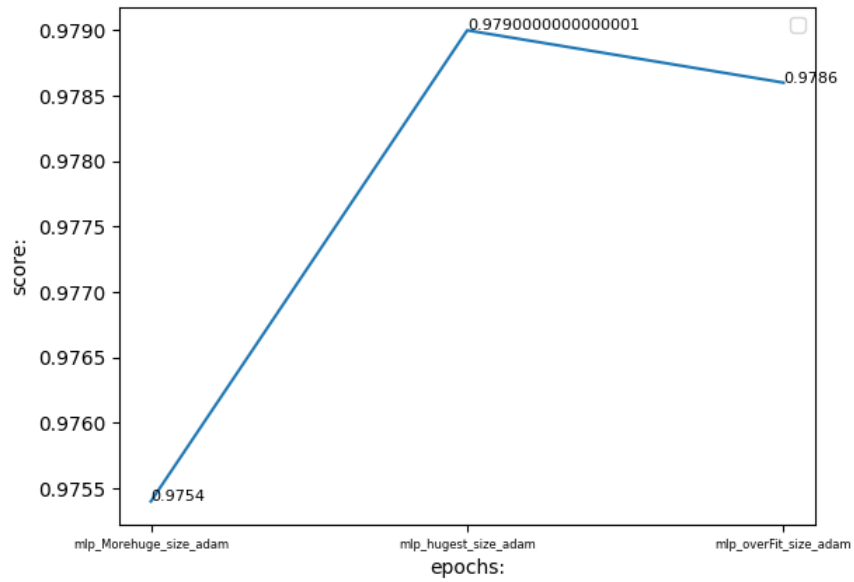
      iii. [28*28, 10240, 512]（overFit）


mlp_Morehuge_size_adam
mlp_hugest_size_adam
mlp_overFit_size_adam
loss:
steps:
0
1
2
3
4
0
2000
4000
6000
8000
10000
12000
14000

损失图


mlp_Morehuge_size_adam
mlp_hugest_size_adam
mlp_overFit_size_adam
score:
epochs:
0.8800
0.8900
0.9000
0.9100
0.9200
0.9300
0.9400
0.9500
0
2
4
6
8
10
12
14
0.8810
0.9030
0.8982
0.9046
0.9174
0.9254
0.9270
0.9316
0.9340
0.9366
0.9336
0.9374
0.9390
0.9418
0.9450
0.9476
0.9472
0.9510
0.9516

验证集得分图



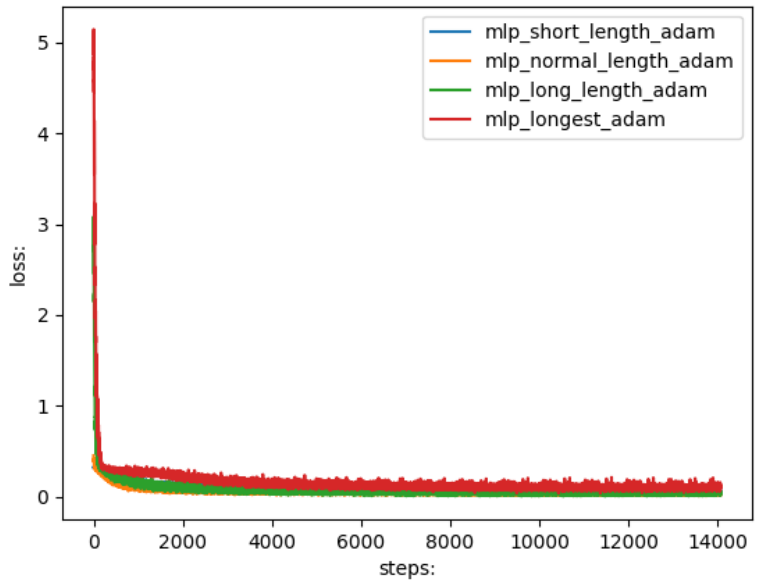

测试集得分图

**结果分析：**

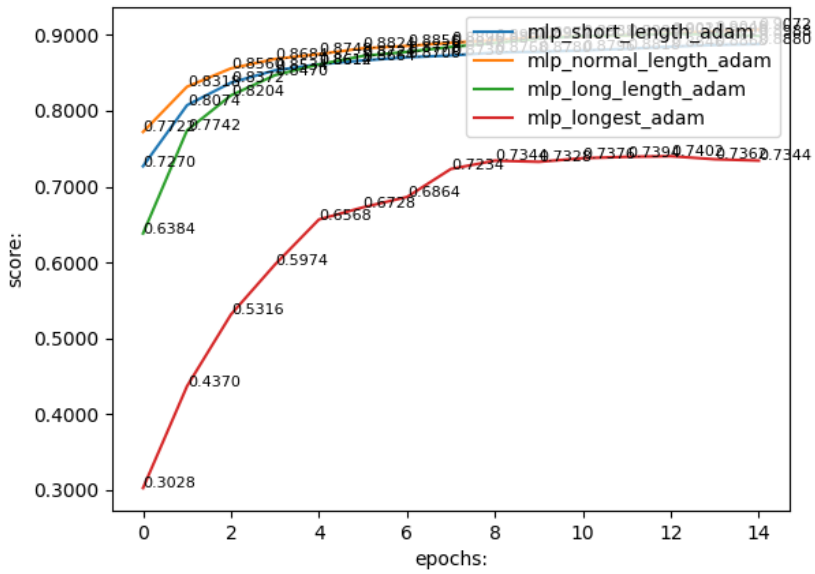
损失图上来看相差并不是很大。但观察验证集得分走向可知，overFit 尺寸下神经网络训练过程变得更加不稳定，可能是由于网络拟合能力的增强导致的对数据集噪声感知变强，出现过拟合。

在最终的测试集得分上来看，overFit 虽然尺寸增大很多，但却不如 hugest 尺寸下的网络。可以得知，网络学习能力的增加随神经元数量增加是有上限的，而且当神经元数量过多时反而可能导致泛化性降低，出现不稳定，过拟合。

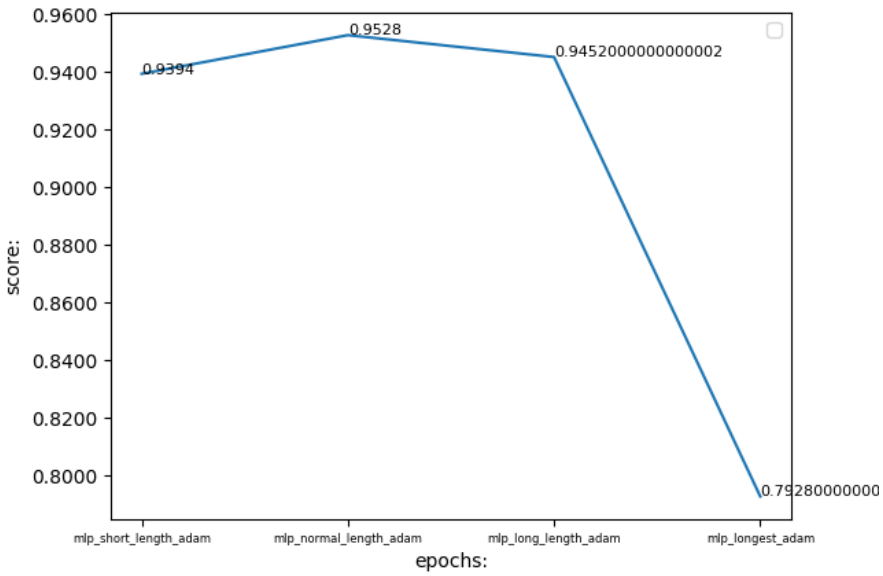
3. 增加网络深度观察实验：
    a. 考虑到计算资源问题，设置网络结构如下，未过多扩展神经元数量：
        i. [28*28, 100]
        ii. [28*28, 100, 70]
        iii. [28*28, 100, 70, 50]
        iv. [28*28, 100, 70, 50, 30]


mlp_short_length_adam
mlp_normal_length_adam
mlp_long_length_adam
mlp_longest_adam
loss:
steps:
0
1
2
3
4
5
0
2000
4000
6000
8000
10000
12000
14000

损失图


mlp_short_length_adam
mlp_normal_length_adam
mlp_long_length_adam
mlp_longest_adam
score:
epochs:
0.3000
0.4000
0.5000
0.6000
0.7000
0.8000
0.9000
0
2
4
6
8
10
12
14
0.7270
0.6384
0.3028
0.4370
0.5316
0.5974
0.6568
0.6728
0.6864
0.7234
0.7344

验证集得分图




测试集得分图

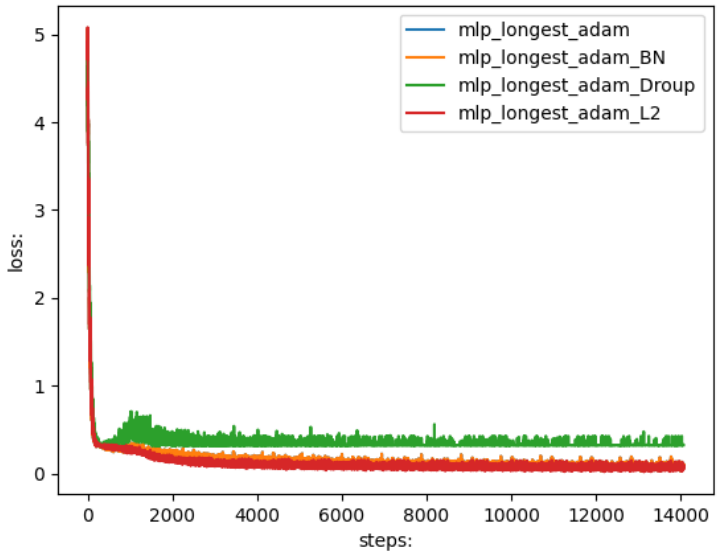
**结果分析：**

从损失下降图可知，longest 尺寸下神经网络收敛损失较高，并未有更好的效果。从验证集得分上来看同样如此，longest 因为层次过深，出现了难以训练的问题，可能出现了梯度消失问题。

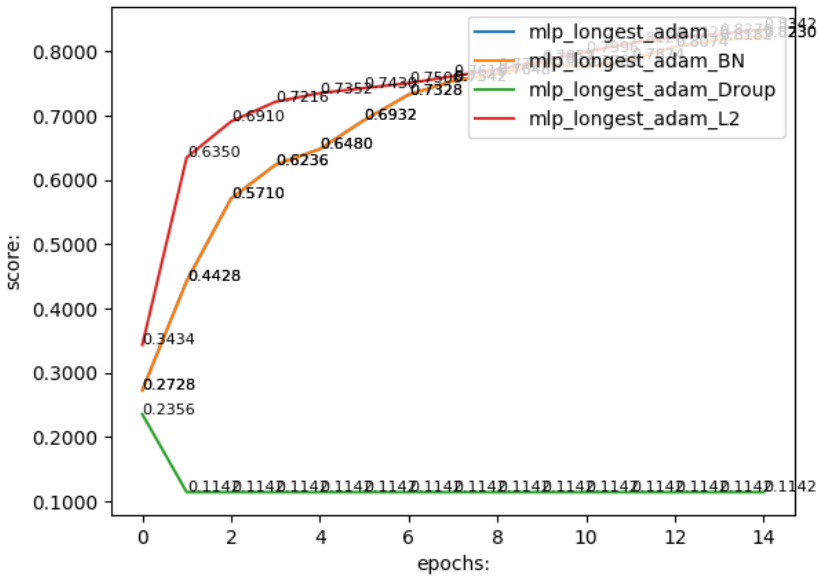
测试集得分也能说明该问题。

### ⑥ MLP 正则化方式影响对比以及正则化参数调校：

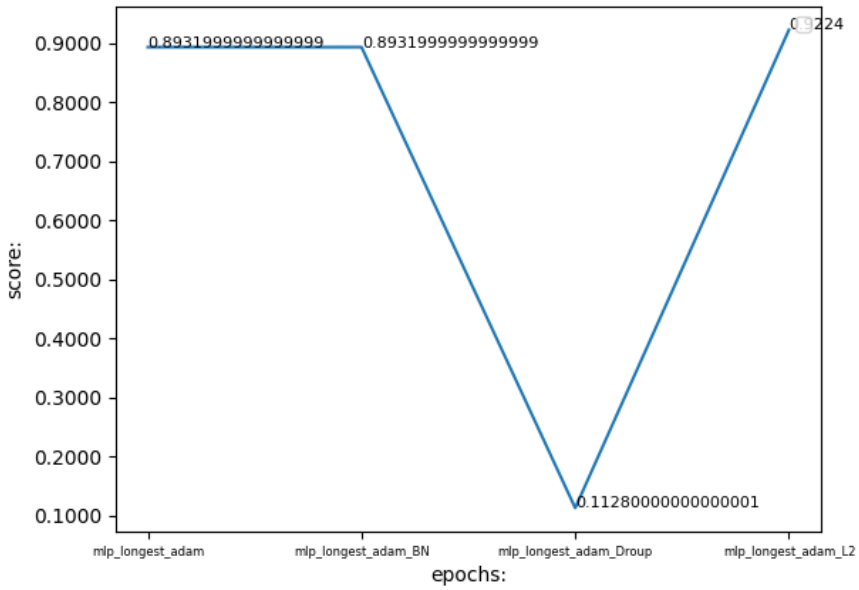
1. 正则化方式对深层网络的影响：
   a. 采用三种正则化方式：BatchNorm（在一些论文中指出可认为是一种正则化方式），Droupout， L2 正则化
   b. Droupout 参数：[0.0，0.1，0.1，0.1，0.1]（为每层 droupout 概率）
   c. L2 正则化参数：lambd = 0.01
   d. 网络结构为[28*28， 100， 70， 50， 30]（除最后一层为 10 个神经元的输出层）


mlp_longest_adam
mlp_longest_adam_BN
mlp_longest_adam_Droup
mlp_longest_adam_L2
loss:
steps:
0
1
2
3
4
5
0
2000
4000
6000
8000
10000
12000
14000

损失图


mlp_longest_adam
mlp_longest_adam_BN
mlp_longest_adam_Droup
mlp_longest_adam_L2
score:
epochs:
0.8000
0.7000
0.6000
0.5000
0.4000
0.3000
0.2000
0.1000
0
2
4
6
8
10
12
14
0.3434
0.6350
0.6910
0.7216
0.7352
0.2728
0.4428
0.5710
0.6236
0.6480
0.6932
0.7328
0.2356
0.1142

验证集得分图




测试集得分图

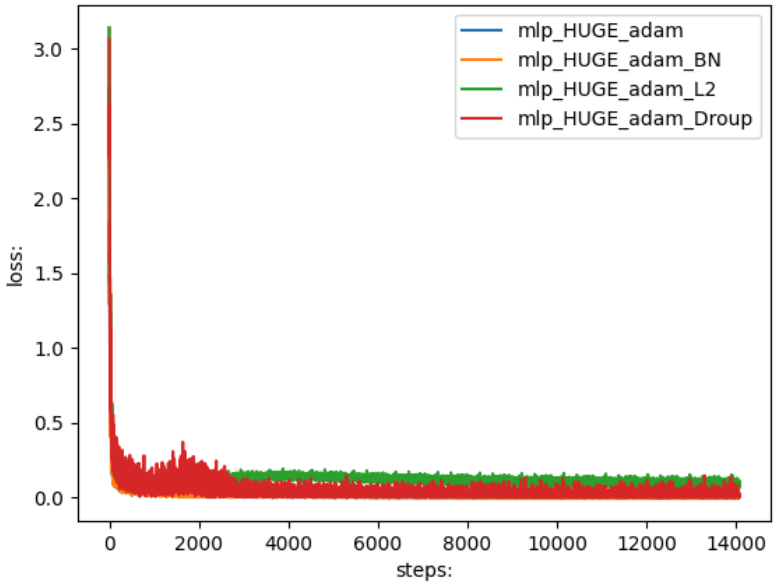
**结果分析：**

从损失下降层面上来讲，使用 L2 正则化和 BatchNorm 方式后，其梯度下降损失收敛值有所降低。

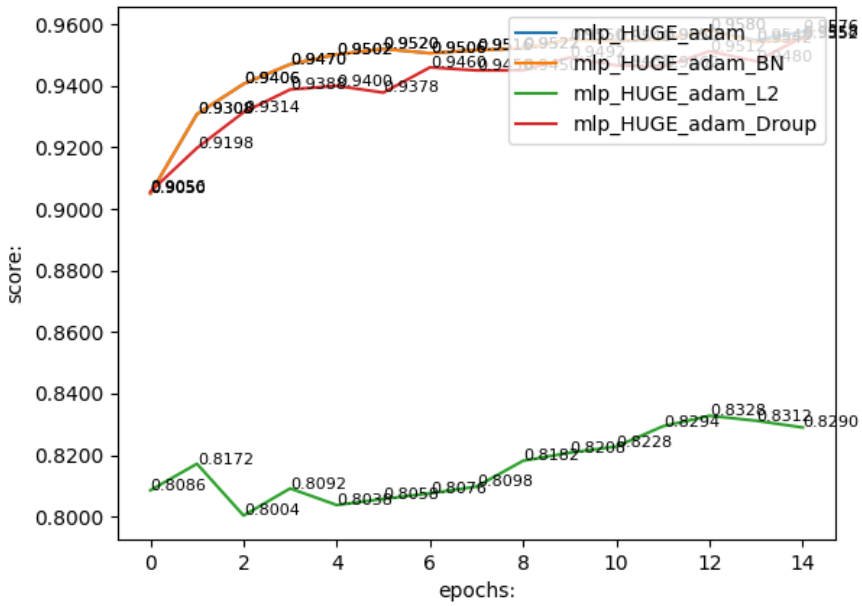
从验证集得分上来看，使用 L2 正则化和 BatchNorm 方式后，其得分上升速度更快。Droupout 可能由于丢弃率参数调试不合理导致网络没能正常训练。

最终测试集得分上来讲 L2 正则化下的网络取得了最佳的成绩。其泛化性能增强。

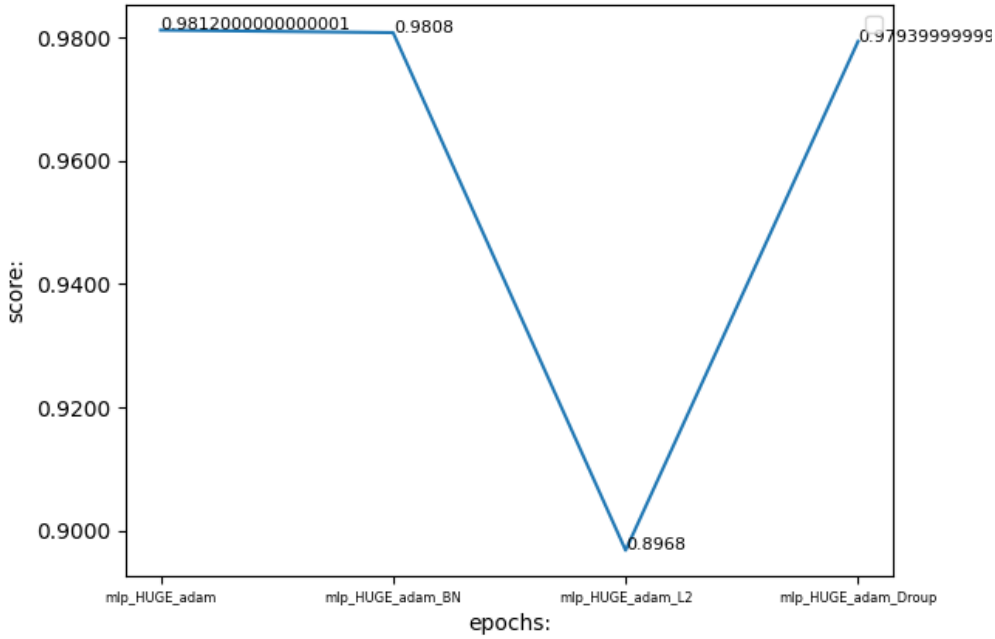
2. 正则化方式在较宽网络结构上影响：
   a. 采用三种正则化方式：BatchNorm， Droupout， L2 正则化
   b. Droupout 参数：[0.0，0.1，0.1] （为每层 droupout 概率）
   c. L2 正则化参数：lambd = 0.01
   d. 网络结构为[28*28， 2048， 256] （除最后一层为 10 个神经元的输出层）


mlp_HUGE_adam
mlp_HUGE_adam_BN
mlp_HUGE_adam_L2
mlp_HUGE_adam_Droup
loss:
steps:
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
2000
4000
6000
8000
10000
12000
14000

损失图


mlp_HUGE_adam
mlp_HUGE_adam_BN
mlp_HUGE_adam_L2
mlp_HUGE_adam_Droup
score:
epochs:
0.8000
0.8200
0.8400
0.8600
0.8800
0.9000
0.9200
0.9400
0.9600
0
2
4
6
8
10
12
14
0.9056
0.9198
0.9308
0.9314
0.9406
0.9470
0.9502
0.9520
0.9506
0.9460
0.9400
0.9378
0.8086
0.8172
0.8004
0.8092
0.8038
0.8058
0.8078
0.8098
0.8182
0.8208
0.8228
0.8294
0.8328
0.8312
0.8290

验证集得分图




测试集得分图

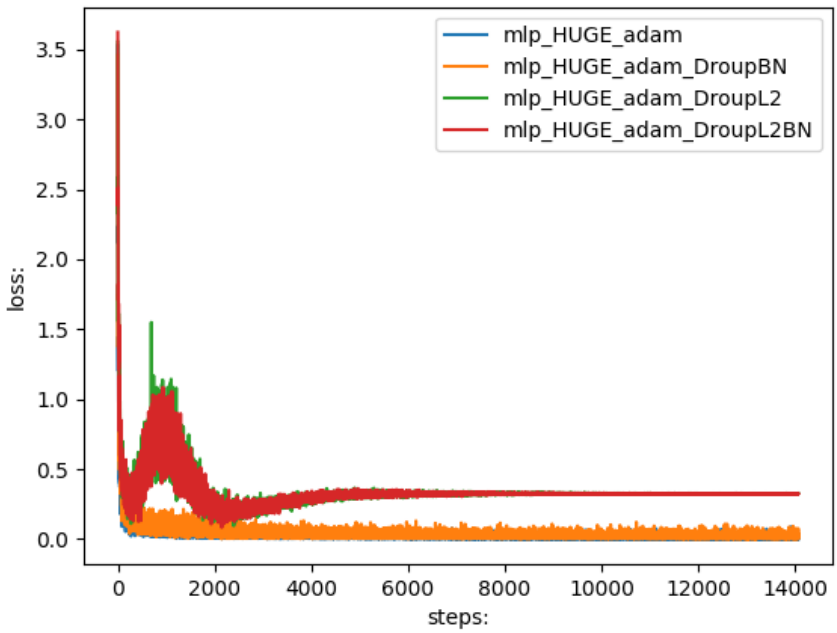
**结果分析：**

从损失下降上来看使用了 Droupout 正则化方法的神经网络损失震荡较为严重，收敛过程不够平稳。

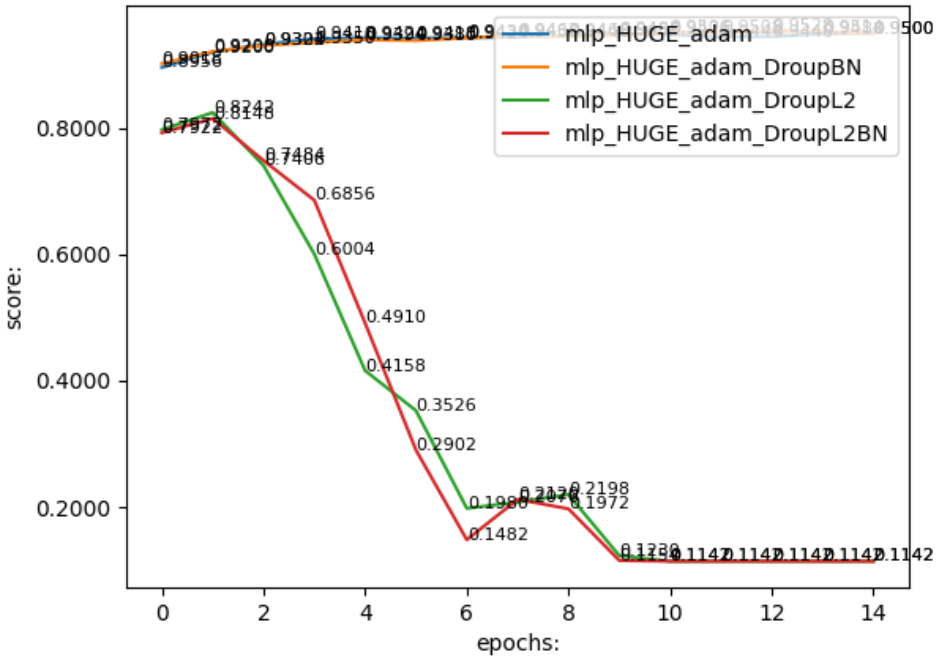
从验证机、测试集得分上来看 L2 正则化的当前参数对于模型拟合能力制约较强，使其得分不高。

3. 正则化方式在较宽网络结构上叠加影响：

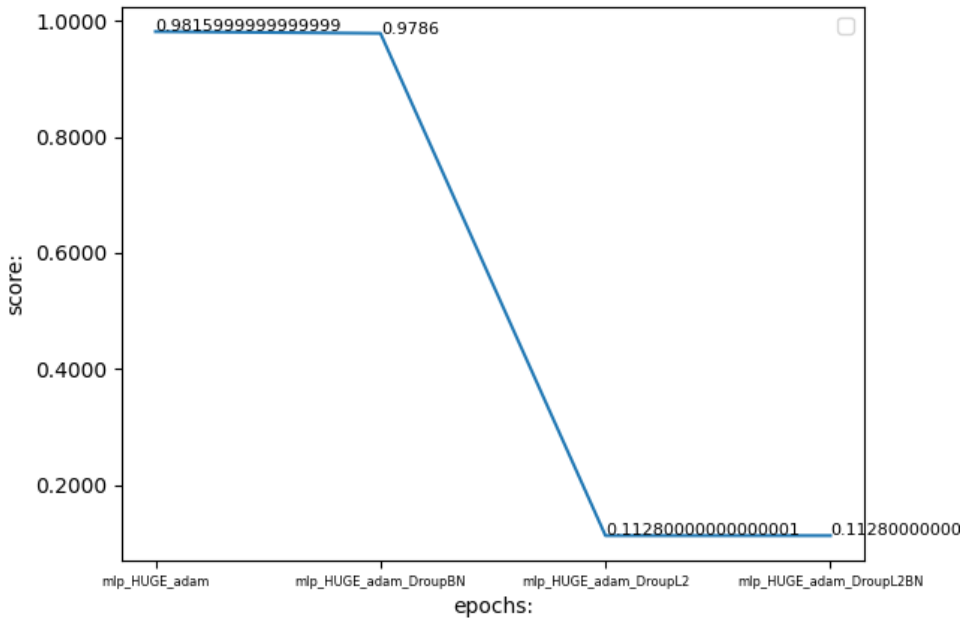
   a. 参数与之前相同，但进行了多种正则化方式混合使用



损失图




验证集得分图




测试集得分图

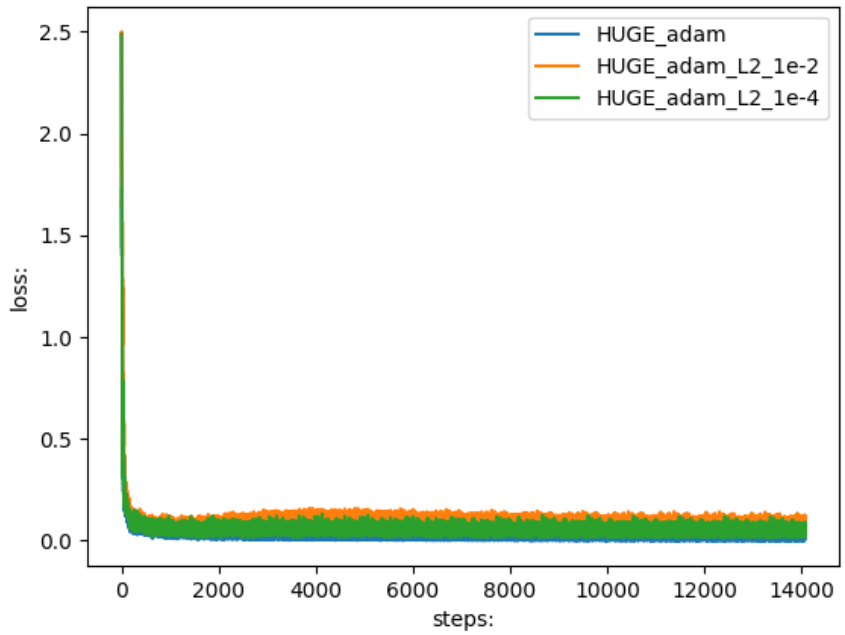
**结果分析：**

从以上结果可知，多种正则化方法的并用并没有带来更好的效果，这可能有参数调校上的问题，也有不适用相关方法上的问题。较好表现的是 Droupout 与 BatchNorm 混用方法。且留意到 Droupout 和 L2 正则化方法一旦混用导致了严重的崩溃问题。

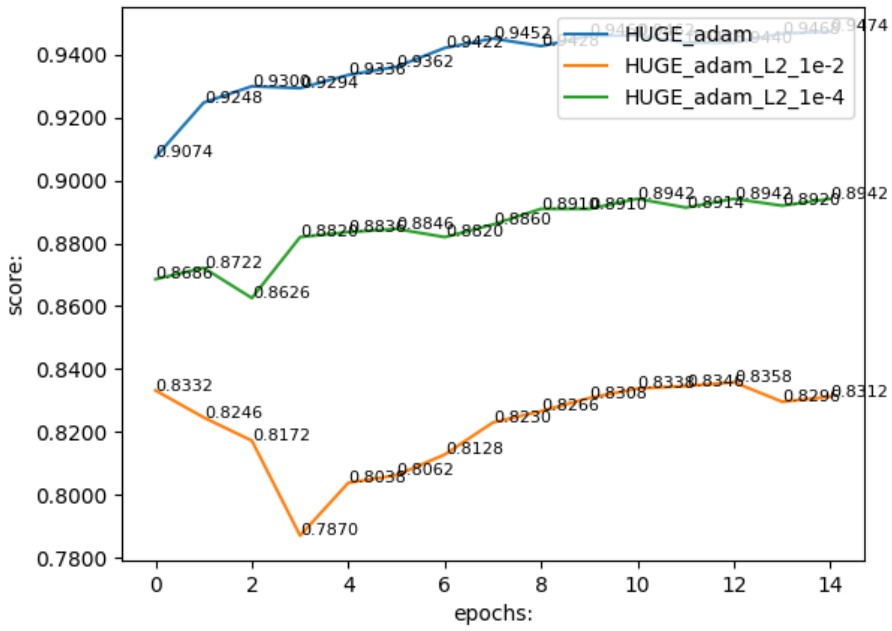
4. 正则化方式在较宽网络结构上参数调试：

a. 模型结构设为[28*28， 5096， 128]

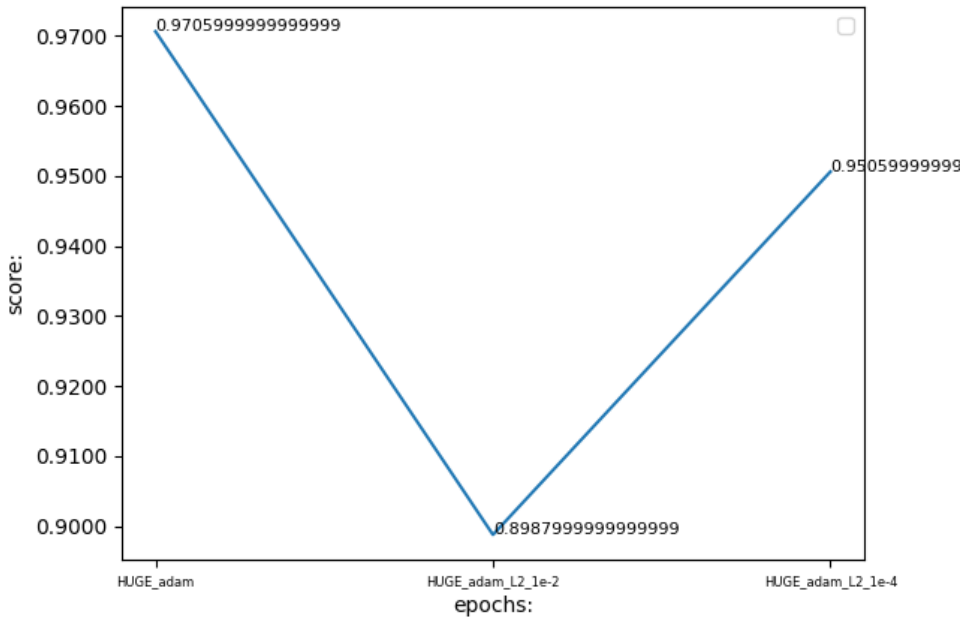
b. 对 L2 正则化参数 lambd 进行调试




损失图




验证集得分图

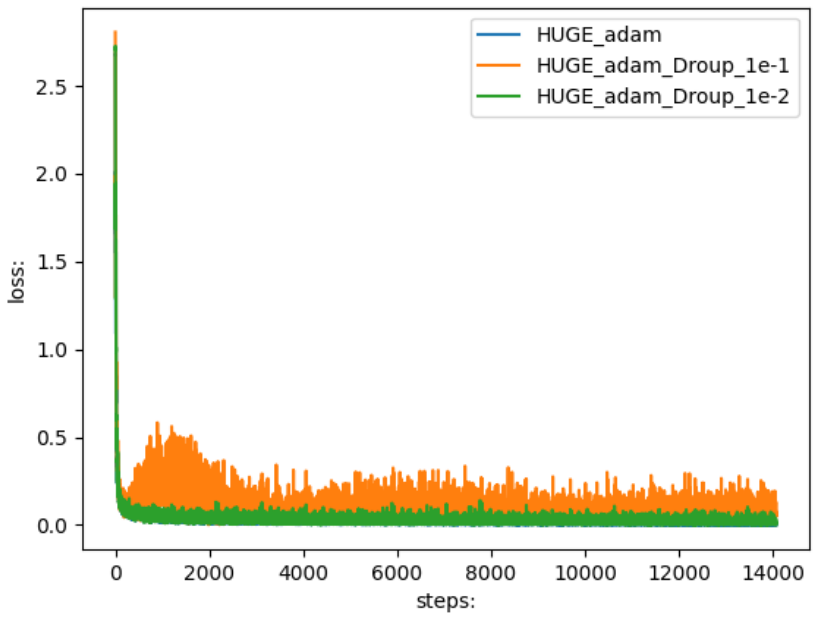



测试集得分图

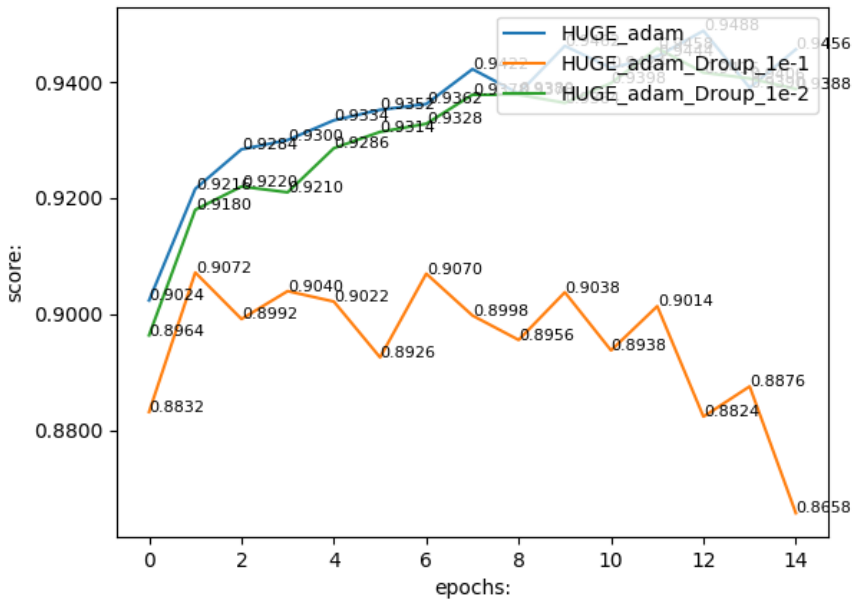
**结果分析：**

从验证集得分、测试集得分来看，随着 L2 正则化参数的增大，模型拟合能力跟随下降。L2 正则化在使用中需要注意超参数的严格调试，以得到最佳泛化性能和相对而言更好的拟合能力。

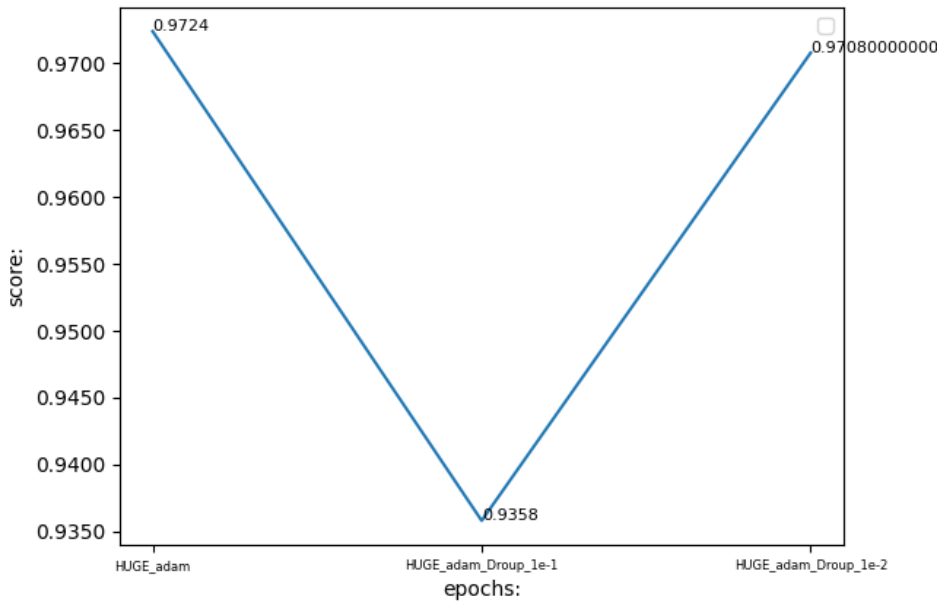
3. Droupout 正则化丢弃率进行调试




损失图




验证集得分图




测试集得分图

**结果分析：**

从 Droupout_1e-1 参属下网络的损失来看，较高的丢弃率可能会导致收敛过程不够平滑，发生震荡。

从验证集得分来看，随丢弃率的增高，模型拟合能力降低。在丢弃率为 0.01 情况下时，可发现其收敛过程更加平滑，震荡幅度更小。适当的丢弃率选择会使训练过程更加稳定，抵抗数据中的噪声。

从测试集得分来看，仍可得出随丢弃率的增高，模型拟合能力降低的结论。

## 三、实验结果总结：

### ① 分类器基本参数调校：

📌 mini-batch，momentum 优化方法下，都对梯度下降步长较为敏感，对于 batch-size，learn-rate 需要注意调试。adam 优化方法下包容性更强，但仍有相当的影响。batch-size，learn-rate 如果过高，将导致梯度下降难以达到更优解，过低将使收敛速度减慢。

### ② 分类器优化方法对比（普通 mini-batch，adam，momentum）：

💡 adam 由于其一定程度上的“自适应性”，在许多任务场景下都有较高的表现成绩，且参数调试上来讲更加容易一些。momentum 相比只做普通的梯度累加的 mini-batch 而言收敛速度更快，对于 batch-size，learn-rate 等参数包容性也更好一些。

### ③ 分类器对比实验（Softmax - MLP）

📌 在本任务场景下，MLP 分类器由于其可学习参数更多，有非线性激活函数等优势确实取得了更好的测试成绩。但在一些轻量化任务场景下，Softmax 也能取得百分之 80 以上的正确率，可进行一定程度上的判断。

### ④ 分类器损失函数对比（CrossEntropy - L2）：

❗ 在该多分类任务上，交叉熵损失确实优于使用 L2 损失的梯度下降方式，其收敛速度更快，收敛精度更高。

### ⑤ MLP 网络结构影响：

🎉 在实验中对网络进行了宽、深两层次的对比：

在一定程度上网络加宽，参数增多可使网络的拟合能力增强，取得更好的测试成绩。但如果参数过多确实容易出现过拟合的问题，对数据噪声更加敏感。

网络在一定程度上加深，可提高其识别效果，但如果过深也将导致难以训练的问题（梯度爆炸、梯度消失）。

### ⑥ MLP 正则化方式影响对比以及正则化参数调校：

💡 L2 对权重大小的约束参数，Droupout 方法的丢弃率需要谨慎调试。过高将导致模型拟合能力下降严重，发生欠拟合。过低也仍无法解决过拟合问题。

BatchNorm 可使梯度下降收敛更加平稳，顺滑。

## 四、结语：

在本次实验中，进行了大量的对比实验，和结果分析。为更快获得更多数据，尝试使用了 cupy 来进行 numpy 的计算，进行了调用 GPU 计算的尝试。

但在对比实验中可能存在着某些流程上的繁琐问题需要改进。对于 MLP 类，Softmax 类的实现仍有更好的解耦合代码方案可采用。MLP 类可采取尝试使用先完成层类，再聚合为 MLP 类这样的尝试，会使调试更加方便，规范。

感谢老师提供本次实验机会，使我更加熟悉底层的计算实现原理，对神经网络各种参数调试，正则化方法，网络结构等操作、问题有了更深的理解和体会。