智能之门

神经网络和深度学习入门

（基于Python的实现）

STEP 7 深度神经网络

第 14 章

搭建深度神经网络框架

14.1 神经网络框架设计

14.2 回归任务功能与实例

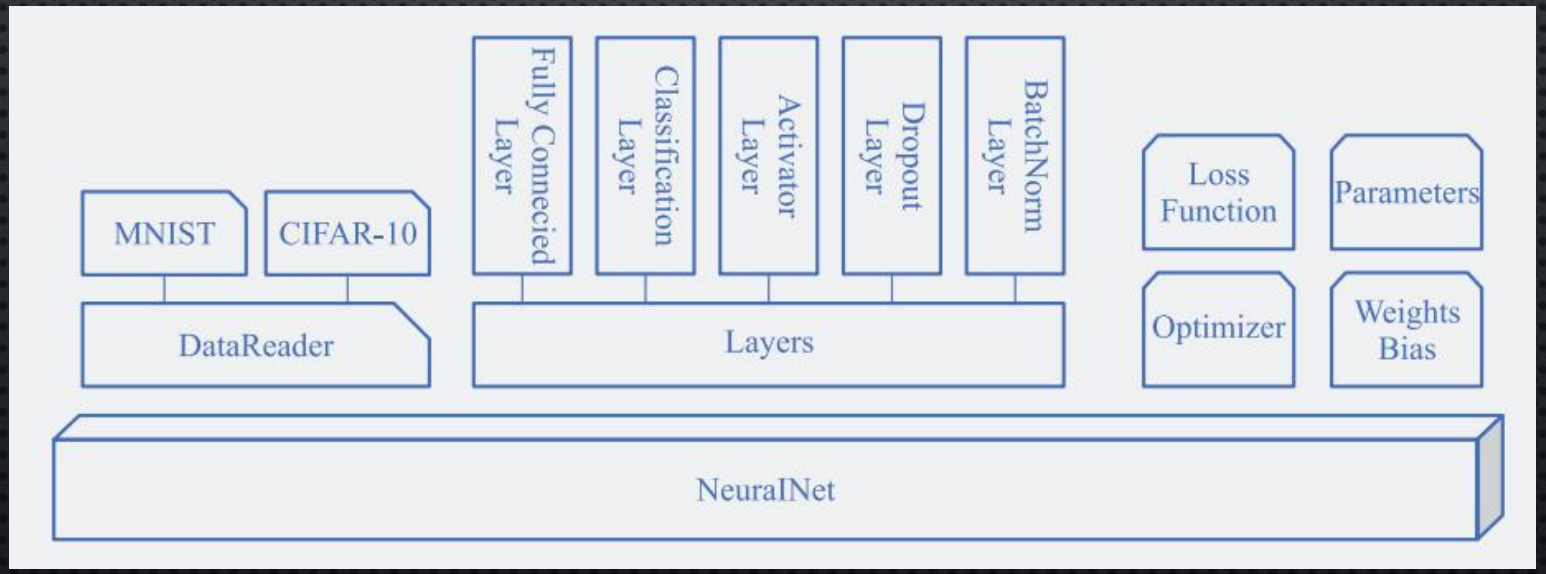
14.3 二分类任务功能与实例

14.4 多分类任务功能与实例

在前面的章节中，我们逐步学习了系统化的知识，本章我们将通过框架和几个实际案例，以更轻松地接触深度学习。

14.1 神经网络框架设计

比较第12章中的三层神经网络的代码，我们可以看到大量的重复之处。虽然三层网络比两层网络多了一层，在初始化、前向、反向、更新参数等四个环节有所不同，但却是有规律的。再加上前面章节中，为了实现一些辅助功能，我们已经写了很多类。所以，现在可以动手搭建一个深度学习的迷你框架了。



14.1 神经网络框架设计

- **NeuralNet**
  - Layers - 神经网络各层的容器，按添加顺序维护一个列表
  - Parameters - 基本参数，包括普通参数和超参
  - Loss Function - 提供计算损失函数值，存储历史记录并最后绘图的功能
  - LayerManagement() - 添加神经网络层
  - ForwardCalculation() - 调用各层的前向计算方法
  - BackPropagation() - 调用各层的反向传播方法
  - PreUpdateWeights() - 预更新各层的权重参数
  - UpdateWeights() - 更新各层的权重参数
  - Train() - 训练
  - SaveWeights() - 保存各层的权重参数
  - LoadWeights() - 加载各层的权重参数

- **Layer**
  - 每个Layer都包括以下基本方法：
  - ForwardCalculation() - 调用本层的前向计算方法
  - BackPropagation() - 调用本层的反向传播方法
  - PreUpdateWeights() - 预更新本层的权重参数
  - UpdateWeights() - 更新本层的权重参数
  - SaveWeights() - 保存本层的权重参数
  - LoadWeights() - 加载本层的权重参数

14.1 神经网络框架设计

- **Activator Layer**
  - Identity - 直传函数，即没有激活处理
  - Sigmoid, Tanh, ReLU
- **Classification Layer**
  - Sigmoid二分类, Softmax多分类
- **Parameters**
  - 学习率, 最大epoch，batch size
  - 损失函数定义
  - 初始化方法
  - 优化器类型
  - 停止条件
  - 正则类型和条件

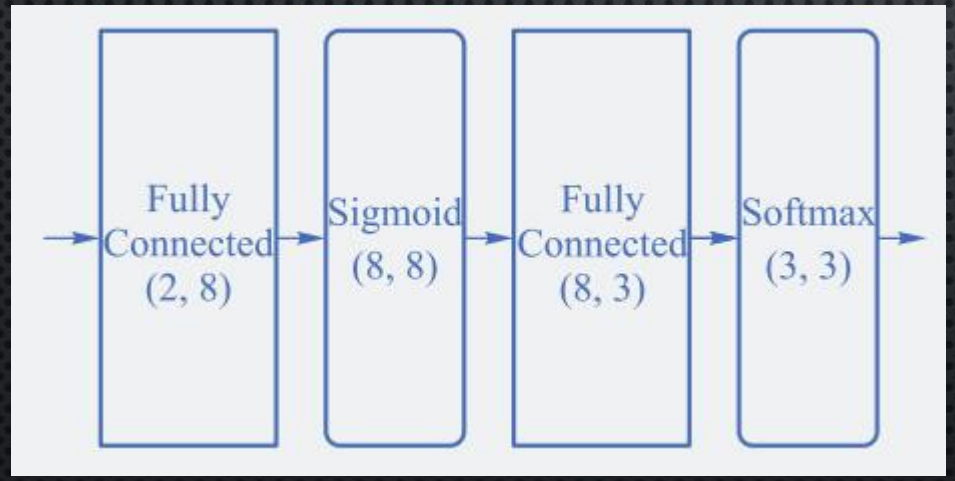
- **LossFunction**
  - 均方差函数
  - 交叉熵函数二分类、多分类
  - 记录损失函数，显示损失函数历史记录
  - 获得最小函数值时的权重参数
- **Optimizer**
  - SGD
  - Momentum
  - Nag
  - AdaGrad
  - AdaDelta
  - RMSProp
  - Adam

14.1 神经网络框架设计

- **WeightsBias**
  - 初始化
    - ✓ Zero, Normal, MSRA (HE), Xavier
    - ✓ 保存初始化值
    - ✓ 加载初始化值
  - Pre_Update - 预更新
  - Update - 更新
  - Save - 保存训练结果值
  - Load - 加载训练结果值
- **DataReader**
  - ReadData - 从文件中读取数据
  - NormalizeX - 归一化样本值
  - NormalizeY - 归一化标签值
  - GetBatchSamples - 获得批数据
  - ToOneHot - 标签值变成OneHot编码用于多分类
  - ToZeroOne - 标签值变成0/1编码用于二分类
  - Shuffle - 打乱样本顺序
- **从中派生出两个数据读取器：**
  - MnistImageDataReader - 读取MNIST数据
  - CifarImageReader - 读取Cifar10数据

14.2 回归任务功能与实例

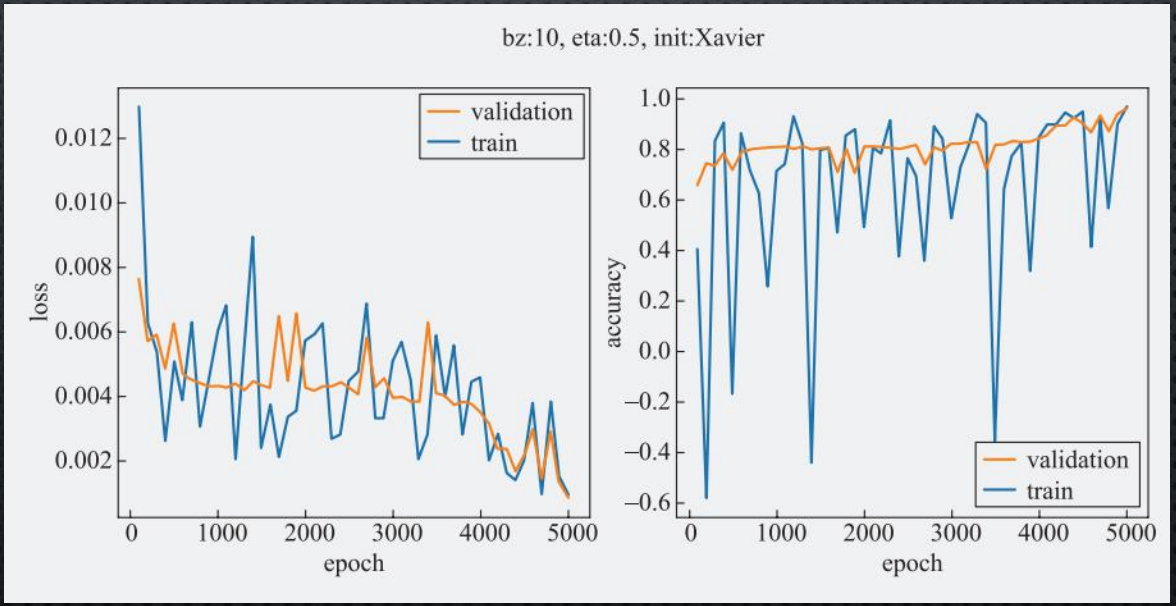
这个模型很简单，一个双层的神经网络，第一层后面接一个 Sigmoid 激活函数，第二层直接输出拟合数据。



- **超参数说明**
  - 输入层1个神经元，因为只有一个x值
  - 隐层4个神经元，对于此问题来说应该是足够了，因为特征很少
  - 输出层1个神经元，因为是拟合任务
  - 学习率=0.5
  - 最大epoch=10000轮
  - 批量样本数=10
  - 拟合网络类型
  - Xavier初始化
  - 绝对损失停止条件=0.001

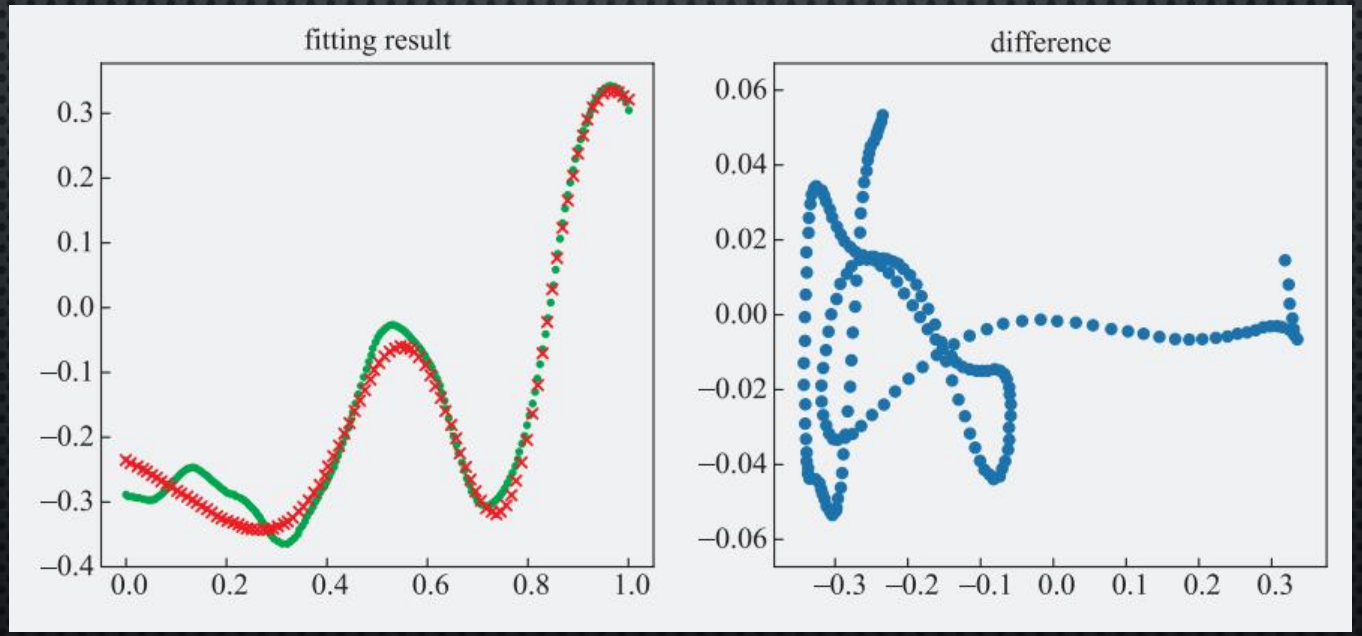
14.2 回归任务功能与实例

- **训练结果：**损失函数值在一段平缓期过后，开始陡降，这种现象在神经网络的训练中是常见的，最有可能的是当时处于一个梯度变化的平缓地带，算法在艰难地寻找下坡路，然后忽然就找到了。这种情况同时也带来一个弊端：我们会经常遇到缓坡，到底要不要还继续训练？



14.2 回归任务功能与实例

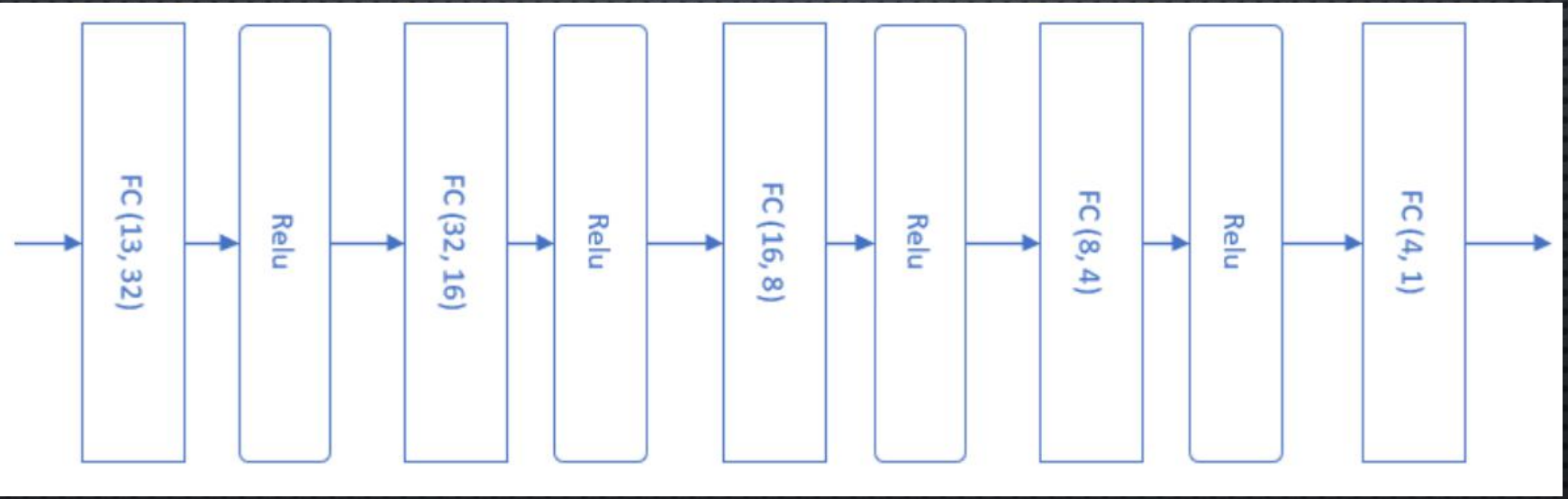
- 左侧子图是拟合的情况，绿色点是测试集数据，红色点是神经网路的推理结果，可以看到除了最左侧开始的部分，其它部分都拟合的不错。
- 右侧子图体现预测误差。



14.2 回归任务功能与实例

➢ 房价预测

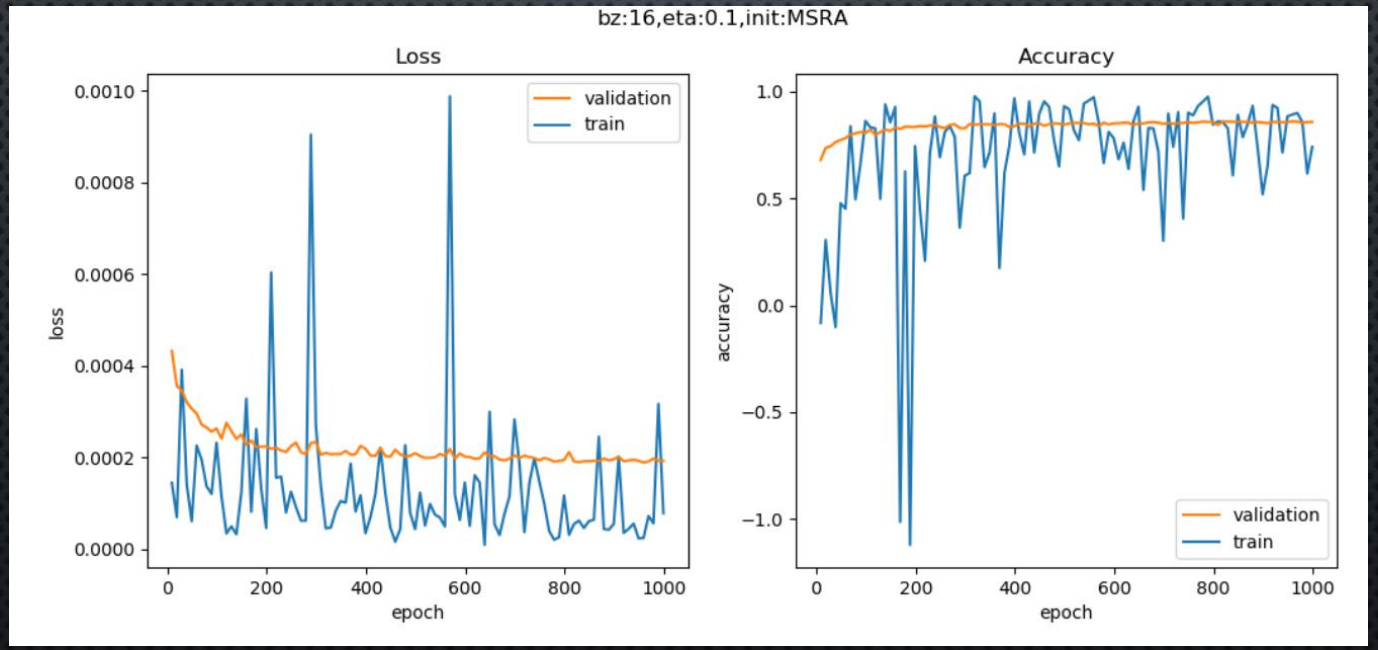
- 这个模型包含了四组全连接层-Relu层的组合，最后是一个单输出做拟合。



14.2 回归任务功能与实例

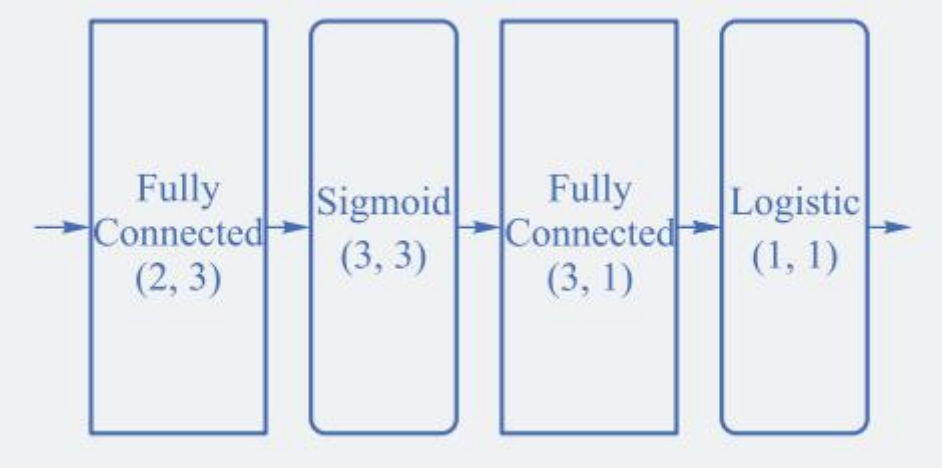
➢ 运行结果

- 损失函数值很快就降到了0.0002以下，然后就很缓慢地下降。而精度值在不断的上升，相信更多的迭代次数会带来更高的精度。



14.3 二分类任务功能与实例

本例同样是一个双层神经网络，但是最后一层要接一个 Logistic 二分类函数来完成二分类任务。

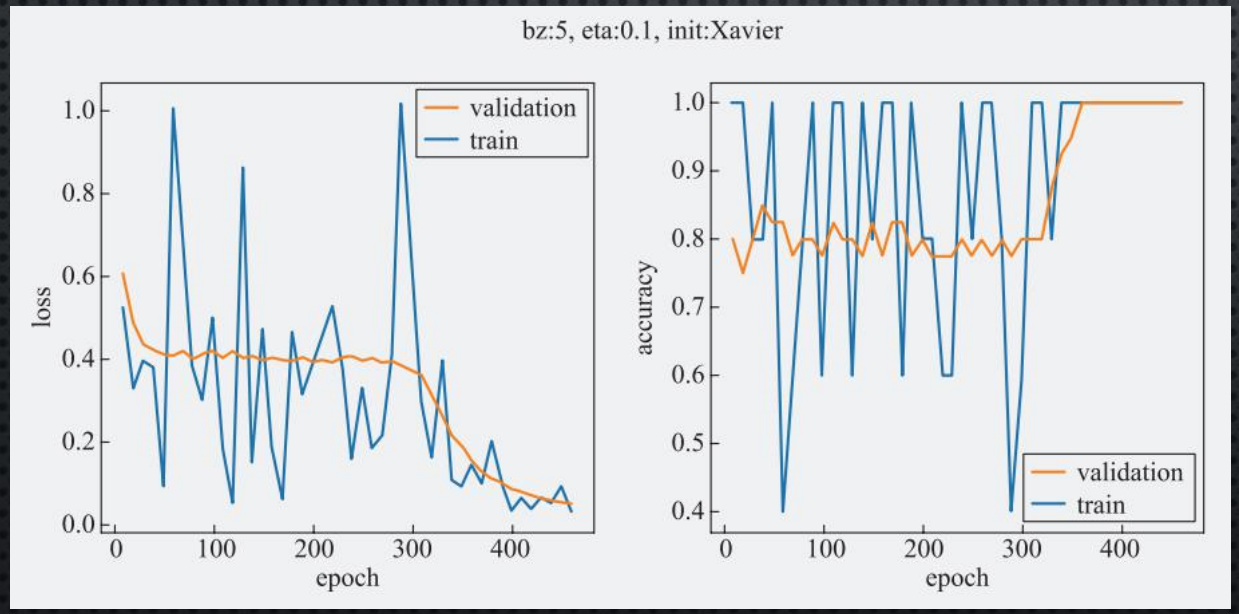


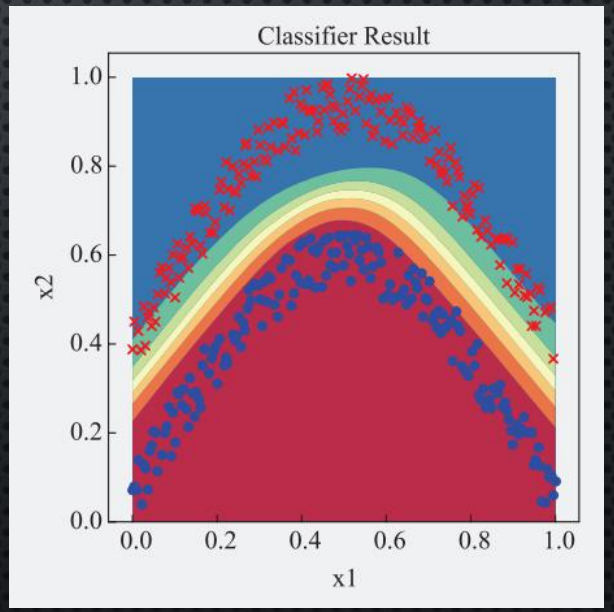
- **超参数说明**
  - 输入层神经元数为2
  - 隐层的神经元数为3，使用Sigmoid激活函数
  - 由于是二分类任务，所以输出层只有一个神经元，用Logistic做二分类函数
  - 最多训练1000轮
  - 批大小=5
  - 学习率=0.1
  - 绝对误差停止条件=0.02

14.3 二分类任务功能与实例

➢ 训练结果和测试结果

- 测试结果是 1.0，表示 100%正确，这初步说明迷你框架在这个基本案例的处理上工作得很好。

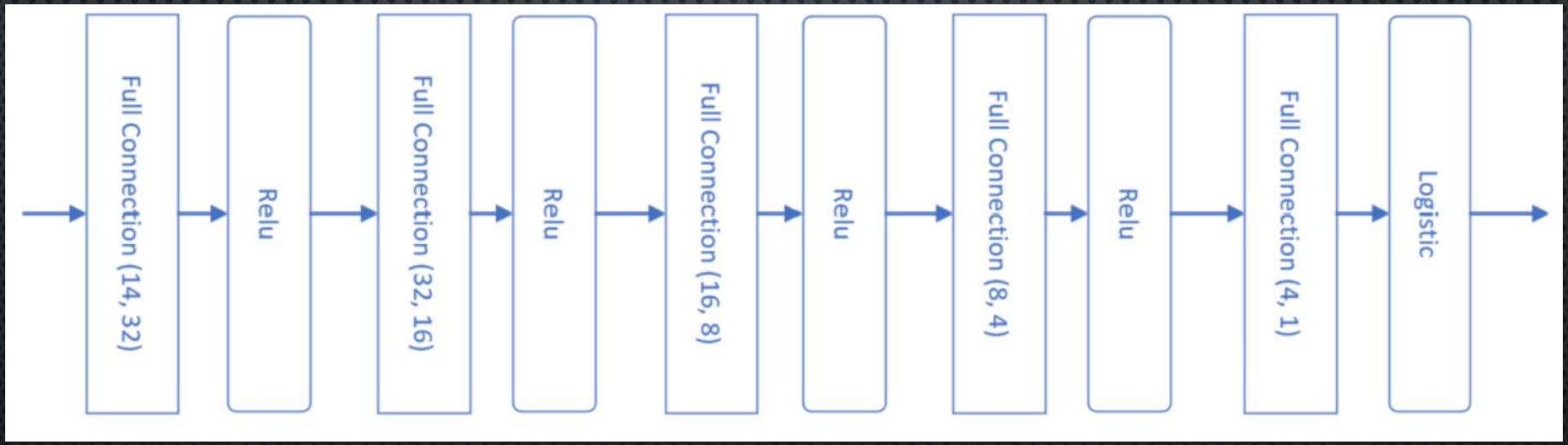




14.3 二分类任务功能与实例

➢ 收入调查与预测

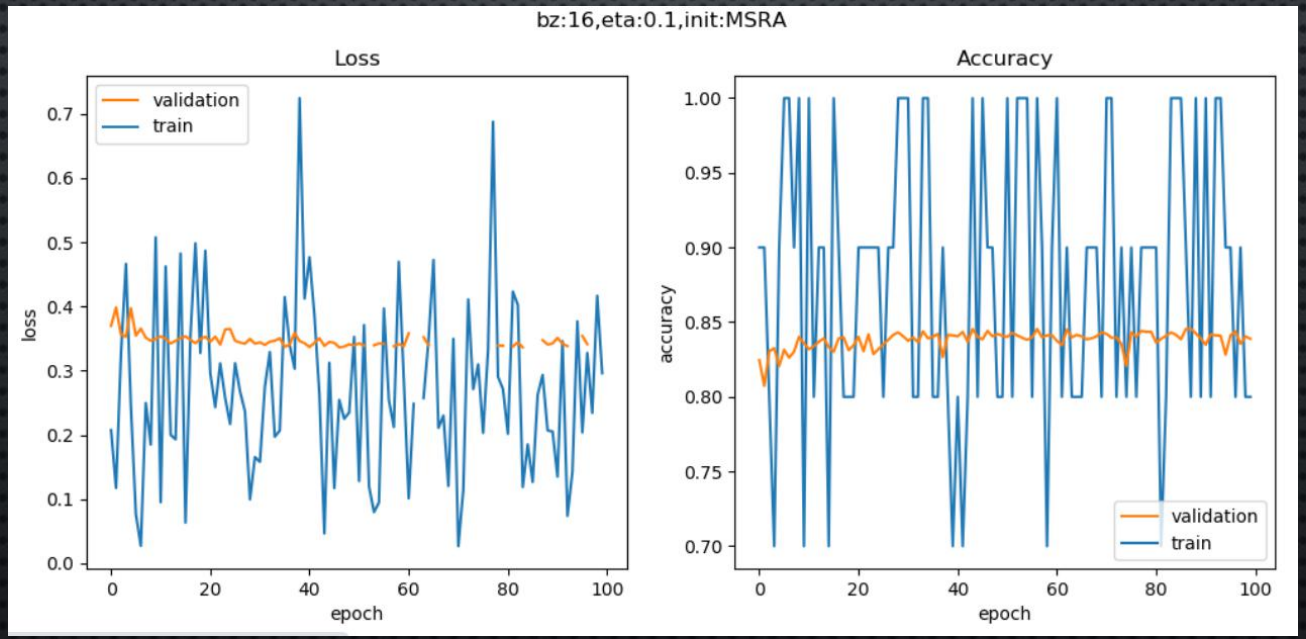
- 为了完成二分类任务，在最后接一个Logistic函数。



14.3 二分类任务功能与实例

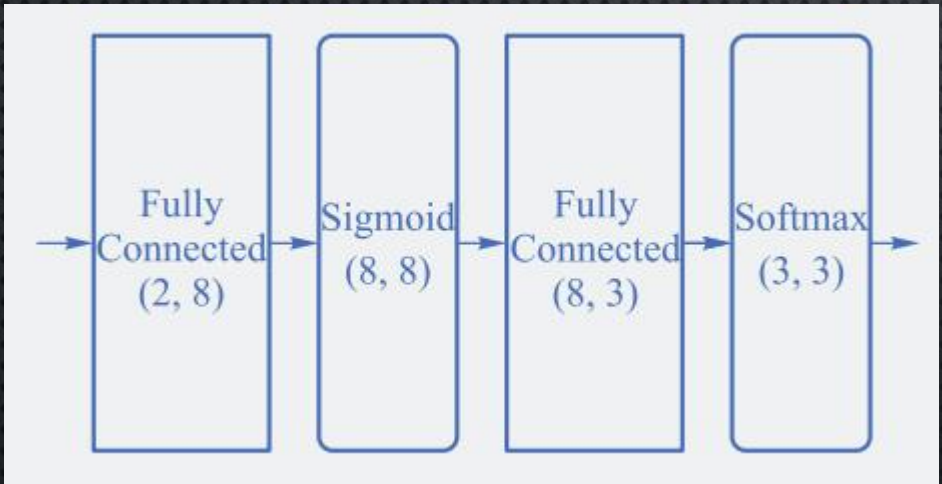
➢ 运行结果

- 左边是损失函数图，右边是准确率图。忽略测试数据的波动，只看红色的验证集的趋势，损失函数值不断下降，准确率不断上升。最后用独立的测试集得到的结果是84%。



14.4 多分类任务功能与实例

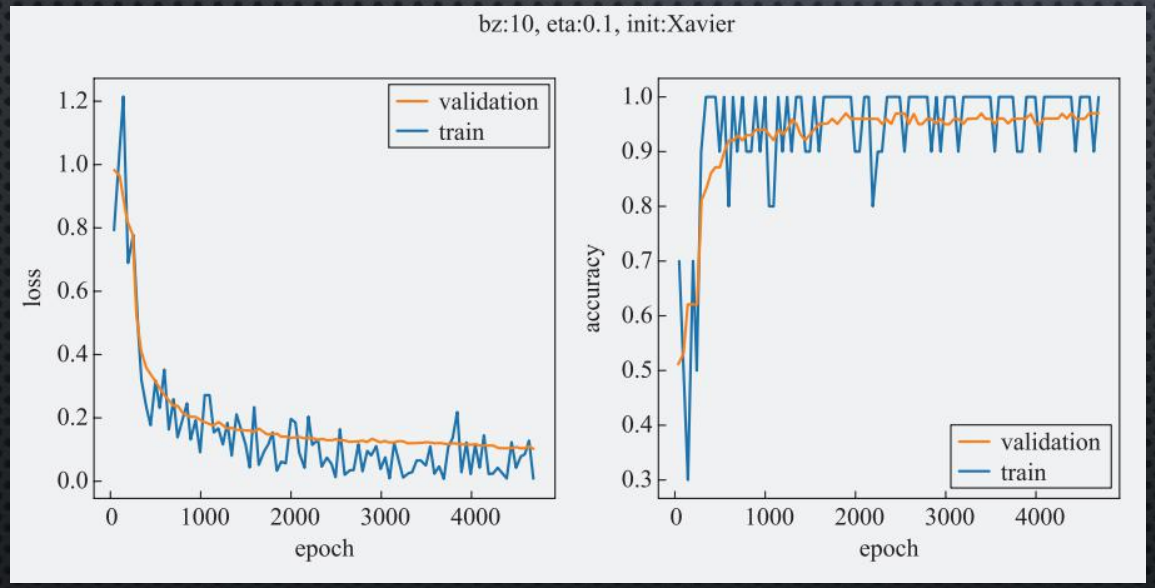
- **模型一**
  - 使用Sigmoid作为激活函数的两层网络，最后采用Softmax多分类函数。

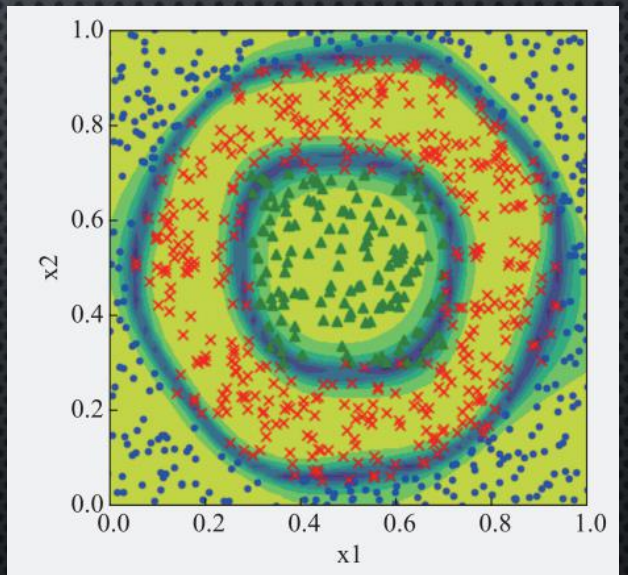


- **超参数说明**
  - 隐层8个神经元
  - 最大epoch=5000
  - 批大小=10
  - 学习率0.1
  - 绝对误差停止条件=0.08
  - 多分类网络类型
  - 初始化方法为Xavier

14.4 多分类任务功能与实例

- 训练结果和测试结果

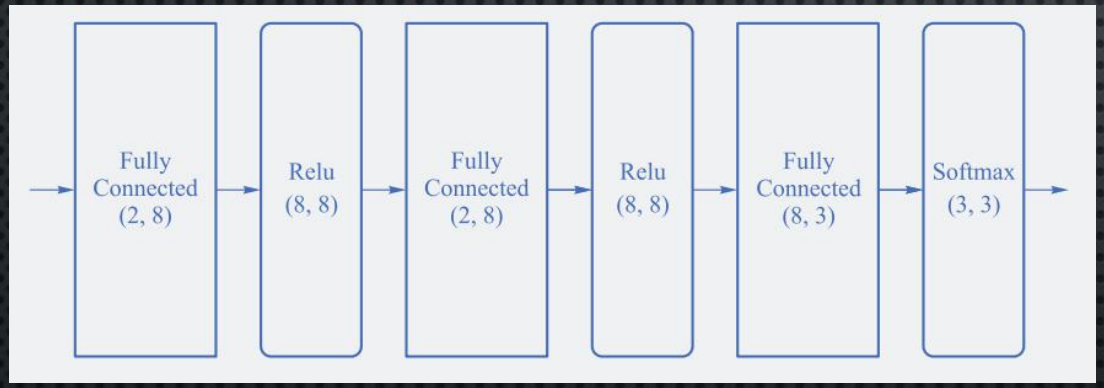




14.4 多分类任务功能与实例

➢ 模型二

- 使用ReLU作为激活函数的三层网络，最后采用Softmax多分类函数。

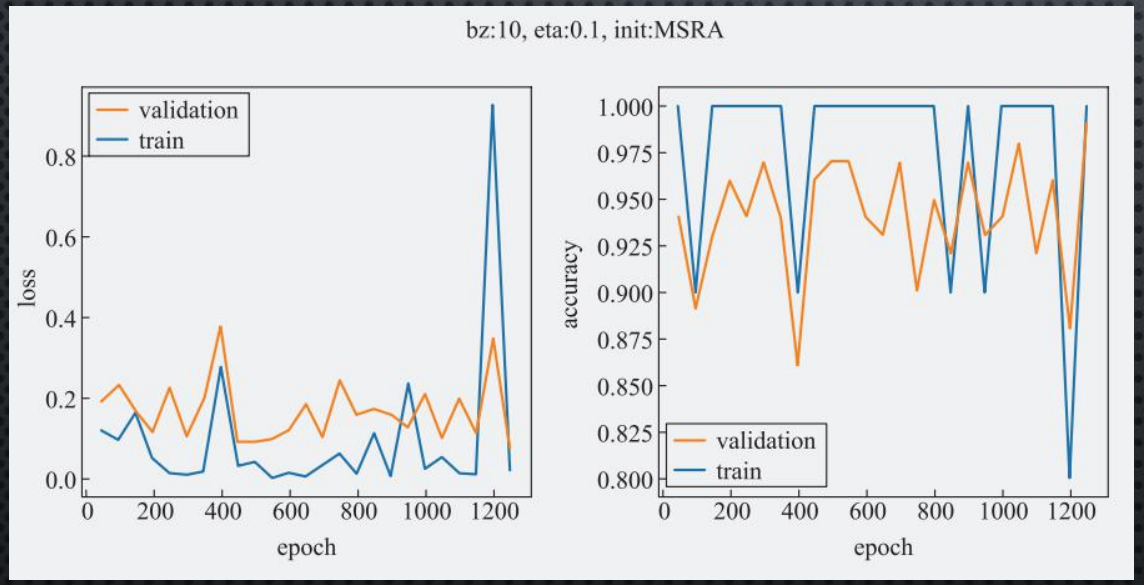


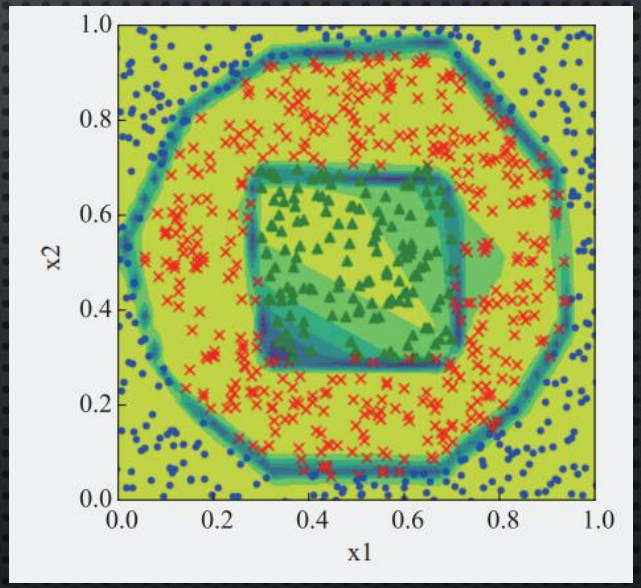
➢ 超参数说明

- 隐层8个神经元
- 最大epoch=5000
- 批大小=10
- 学习率0.1
- 绝对误差停止条件=0.08
- 多分类网络类型
- 初始化方法为MSRA

14.4 多分类任务功能与实例

- 训练结果和测试结果





14.4 多分类任务功能与实例

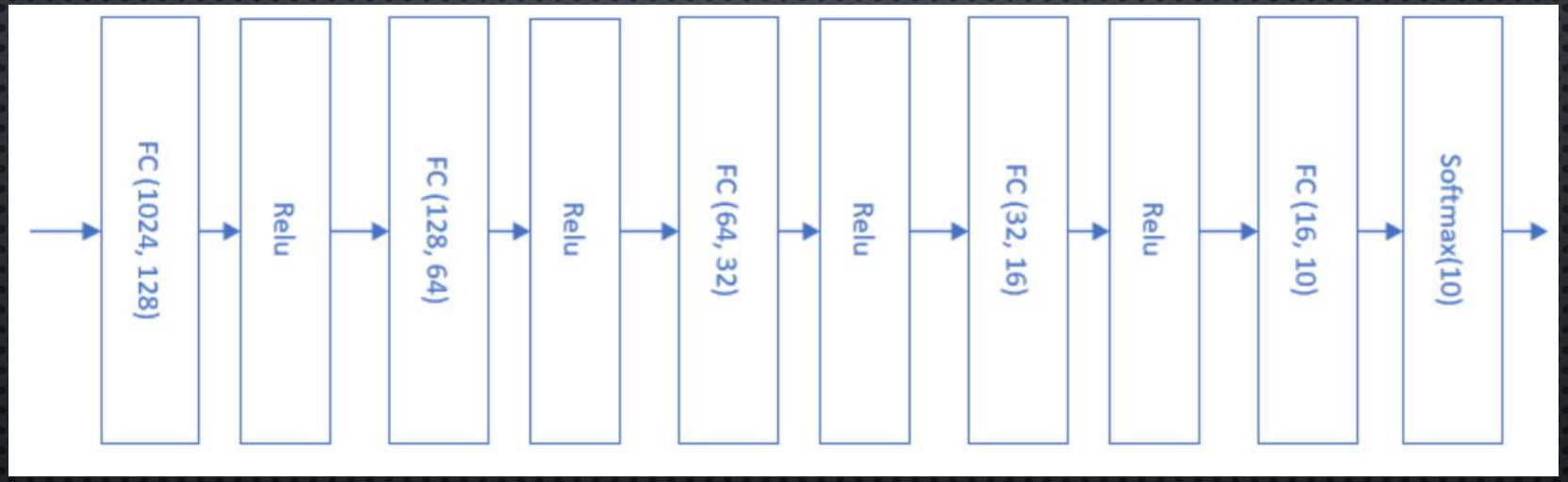
➢ 比较

- ReLU 是用分段线性拟合曲线，Sigmoid 有真正的曲线拟合能力，因而拟合边界更加平滑。
- 但是 Sigmoid 也有缺点，看分类的边界，使用 ReLU 函数的分类边界比较清晰，而使用 Sigmoid 函数的分类边界要平缓一些，过渡区较宽。
- 用一句简单的话来描述二者的差别：ReLU 能直则直，对方形边界适用；Sigmoid 能弯则弯，对圆形边界适用。

14.4 多分类任务功能与实例

➢ MNIST手写体识别

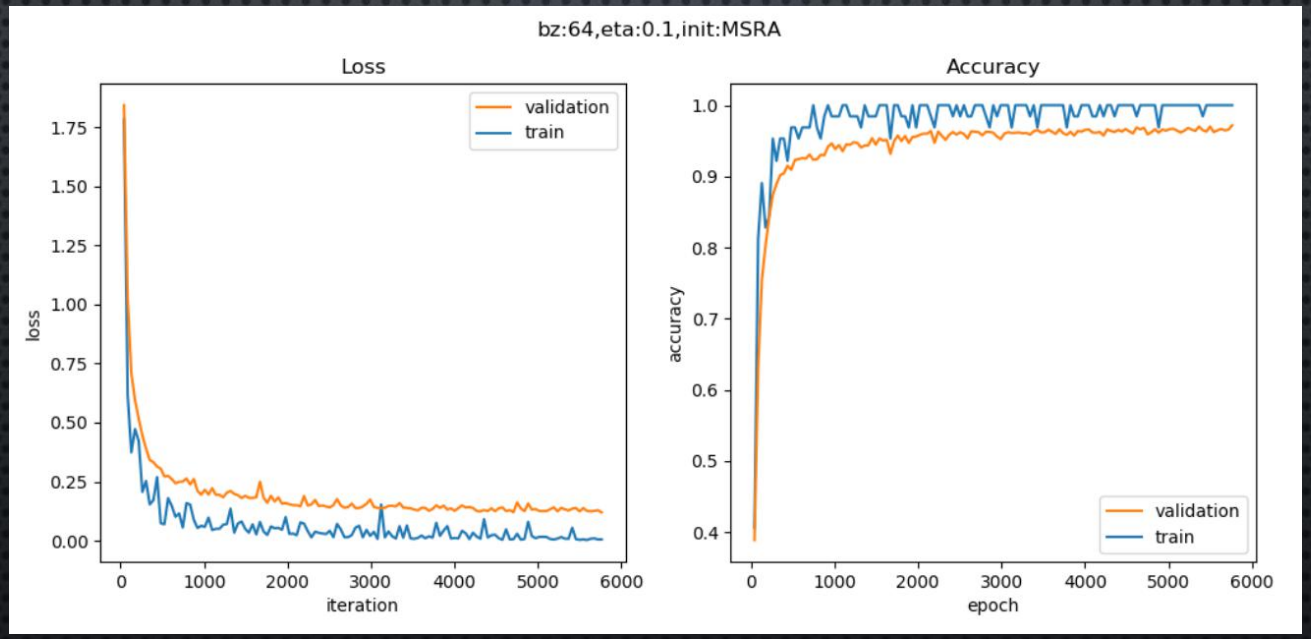
- 一共4个隐层，都用ReLU激活函数连接，最后的输出层接Softmax分类函数。



14.4 多分类任务功能与实例

➢ 运行结果

- 我们设计的停止条件是绝对Loss值低于0.12，所以迭代到6个epoch时，达到了0.119的损失值，就停止训练了。最终测试集上的准确率为96.97%。



THE END

谢谢！