homework#2: 理论题

By: 黄哲昊 (518021910660)

2020 年 10 月

1 理论题

1.1 现假设样本来自三个类，某次训练中的一个 batch 包含 3 个训练样本x_1，x_2，x_3，分别来自第 1，2，3 类

1. 试推导采用单热向量编码时该 batch 交叉熵损失函数表达式。(提示：设该 batch 对应网络输出为$\boldsymbol{y_1}$,$\boldsymbol{y_2}$,$\boldsymbol{y_3}$)
2. 如果网络输出为$\boldsymbol{y_1}$= $(0.65, 0.43, 0.11)$，$\boldsymbol{y_2}$= $(0.05, 0.51, 0.18)$，$\boldsymbol{y_3}$= $(0.33, 0.21, 0.72)$，计算交叉熵损失函数值。

1.1.1

假设第 i 个训练样本对应网络输出为$\boldsymbol{y_i}$= (y_{i1}, y_{i2}, y_{i3})，则属于第 j 类 (j=1,2,3) 的概率为

$$P(j|\boldsymbol{y_i}) = \frac{e^{y_{ij}}}{\sum_{k=1}^{3} e^{y_{ik}}} \tag{1}$$

三个样本真实值的单热向量输出分别为$\hat{\boldsymbol{y_1}}$=$(1, 0, 0)$，$\hat{\boldsymbol{y_2}}$=$(0, 1, 0)$，$\hat{\boldsymbol{y_3}}$=$(0, 0, 1)$。则这个 batch 的交叉熵损失为：

$$\mathcal{L} = \sum_{i=1}^{3} H(\hat{y}_i, P(\boldsymbol{y_i})) \tag{2}$$

$$= -(\log \frac{\hat{\boldsymbol{y_1}}(e^{\boldsymbol{y_1}})^T}{\sum_{k=1}^{3} e^{y_{1k}}} + \log \frac{\hat{\boldsymbol{y_2}}(e^{\boldsymbol{y_2}})^T}{\sum_{k=1}^{3} e^{y_{2k}}} + \log \frac{\hat{\boldsymbol{y_3}}(e^{\boldsymbol{y_3}})^T}{\sum_{k=1}^{3} e^{y_{3k}}}) \tag{3}$$

$$= -(\log \frac{e^{y_{11}}}{\sum_{k=1}^{3} e^{y_{1k}}} + \log \frac{e^{y_{22}}}{\sum_{k=1}^{3} e^{y_{2k}}} + \log \frac{e^{y_{33}}}{\sum_{k=1}^{3} e^{y_{3k}}}) \tag{4}$$

1.1.2

将网络输出$\boldsymbol{y_1}$= $(0.65, 0.43, 0.11)$，$\boldsymbol{y_2}$= $(0.05, 0.51, 0.18)$，$\boldsymbol{y_3}$= $(0.33, 0.21, 0.72)$，代入上式 (4)，得：

$$\mathcal{L} = -(\log \frac{e^{0.65}}{e^{0.65} + e^{0.43} + e^{0.11}} + \log \frac{e^{0.51}}{e^{0.05} + e^{0.51} + e^{0.18}} + \log \frac{e^{0.72}}{e^{0.33} + e^{0.21} + e^{0.72}}) \tag{5}$$

$$= 2.54692 \tag{6}$$

1.2 假设输入有 2 个样本 x_1，x_2

1. 请画出计算 x_1，x_2 标准差的详细计算图；
2. 标出当 $x_1 = -1$，$x_2 = 3$ 时，输出对图中每个节点输入变量的梯度值，并求出 x_1，x_2 总的梯度值。

1.2.1

计算 x_1，x_2 标准差的计算图如下：

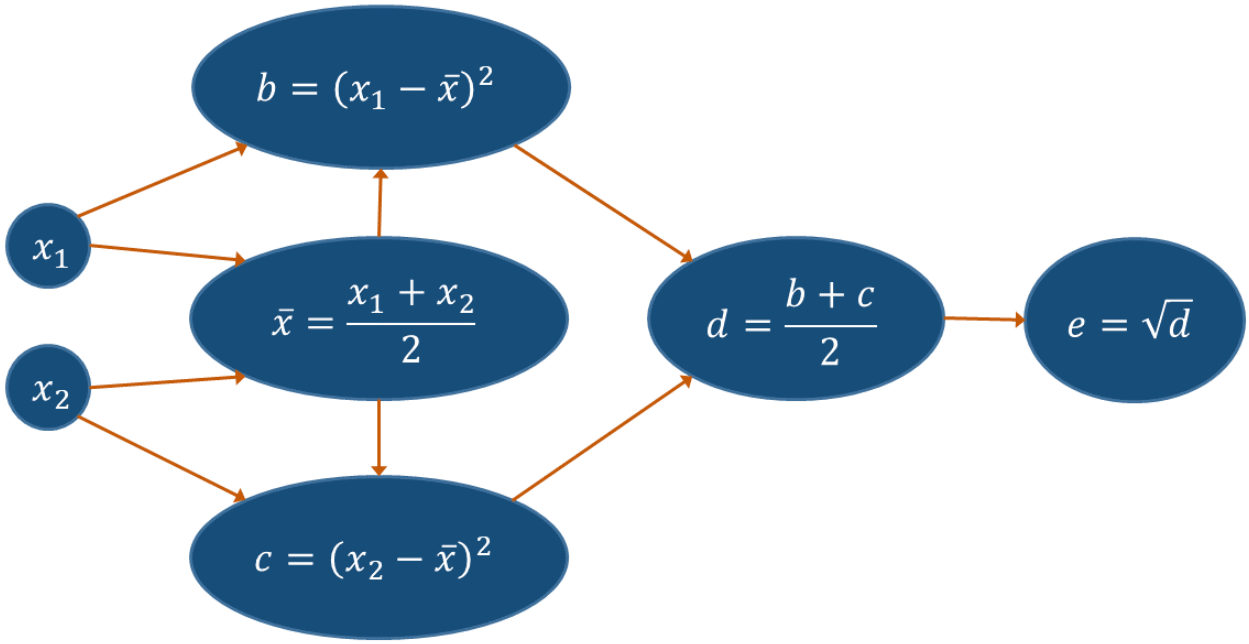


图 1: 计算图

1.2.2

当 $x_1 = -1$，$x_2 = 3$ 时，正向传播和反向传播后得到的计算图如图所示，其中红色数字表示每个节点的输出结果，蓝色数字则表示每个节点计算得到的上游和下游梯度：

对梯度进行计算得到：

$$\frac{\partial e}{\partial \bar{x}} = \frac{\partial e}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}\frac{\partial b}{\partial \bar{x}} + \frac{\partial e}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial c}\frac{\partial c}{\partial \bar{x}} \tag{7}$$

$$= 1 \times \frac{1}{4} \times \frac{1}{2} \times 4 + 1 \times \frac{1}{4} \times \frac{1}{2} \times (-4) \tag{8}$$

$$= 0 \tag{9}$$

$$\frac{\partial e}{\partial x_1} = \frac{\partial e}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}\frac{\partial b}{\partial x_1} + \frac{\partial e}{\partial \bar{x}}\frac{\partial \bar{x}}{\partial x_1} \tag{10}$$

$$= 1 \times \frac{1}{4} \times \frac{1}{2} \times (-4) + 0 \times \frac{1}{2} \tag{11}$$

$$= -0.5 \tag{12}$$

$$\frac{\partial e}{\partial x_2} = \frac{\partial e}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}\frac{\partial b}{\partial x_1} + \frac{\partial e}{\partial \bar{x}}\frac{\partial \bar{x}}{\partial x_2} \tag{13}$$

$$= 1 \times \frac{1}{4} \times \frac{1}{2} \times 4 + 0 \times \frac{1}{2} \tag{14}$$

$$= 0.5 \tag{15}$$

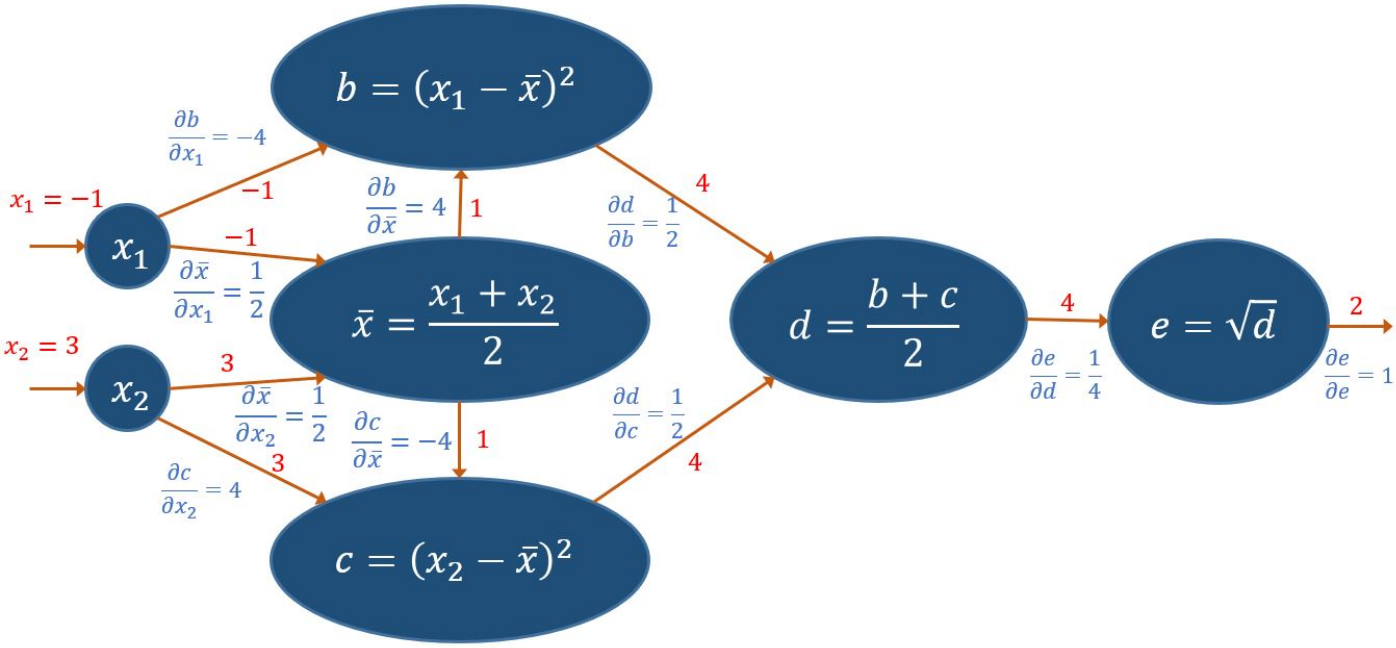

$b = (x_1 - \bar{x})^2$
$\bar{x} = \frac{x_1 + x_2}{2}$
$c = (x_2 - \bar{x})^2$
$d = \frac{b + c}{2}$
$e = \sqrt{d}$
$x_1 = -1$
$x_2 = 3$
$\frac{\partial b}{\partial x_1} = -4$
$\frac{\partial \bar{x}}{\partial x_1} = \frac{1}{2}$
$\frac{\partial \bar{x}}{\partial x_2} = \frac{1}{2}$
$\frac{\partial c}{\partial x_2} = 4$
$\frac{\partial b}{\partial \bar{x}} = 4$
$\frac{\partial c}{\partial \bar{x}} = -4$
$\frac{\partial d}{\partial b} = \frac{1}{2}$
$\frac{\partial d}{\partial c} = \frac{1}{2}$
$\frac{\partial e}{\partial d} = \frac{1}{4}$
$\frac{\partial e}{\partial e} = 1$
−1
−1
3
3
1
1
4
4
4
2

图 2: 梯度图

作业 2 实践题结果与分析 黄哲昊 518021910660

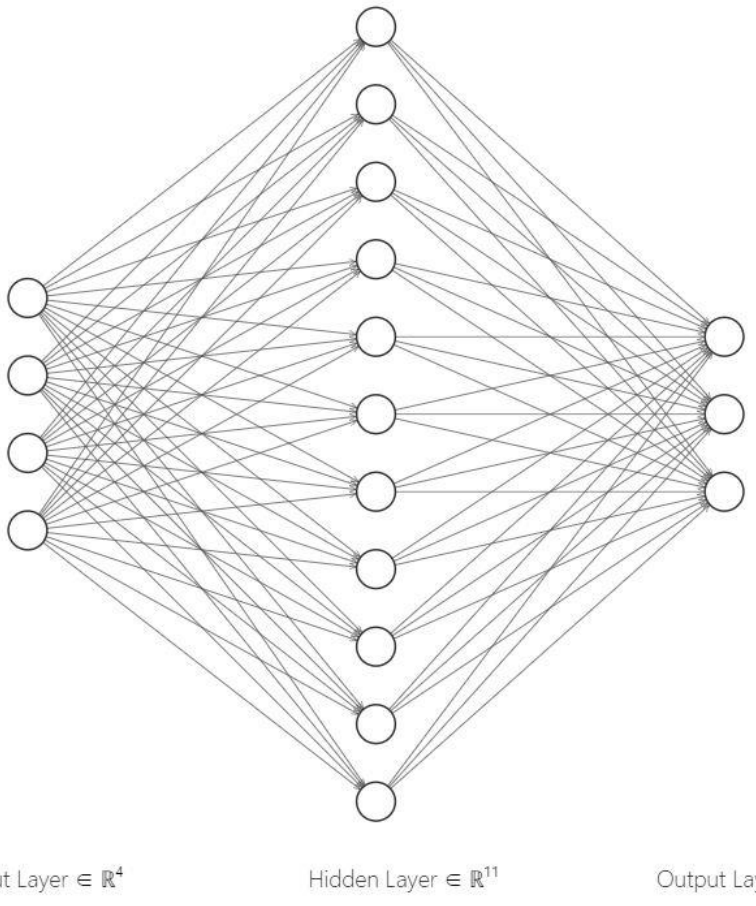
实践题 1 中我使用的是 pytorch 深度学习框架。

实践题 1-1

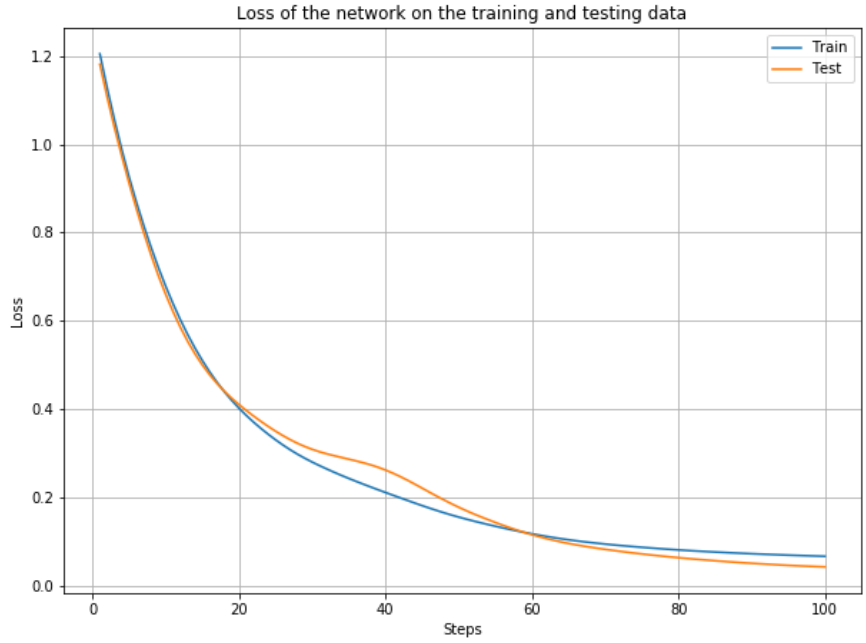
实现一个三层神经网络，并使用 iris 数据集前 80%训练、后 20%测试，要求测试错误率小于 5%，分析至少三种非线性激活函数的影响。

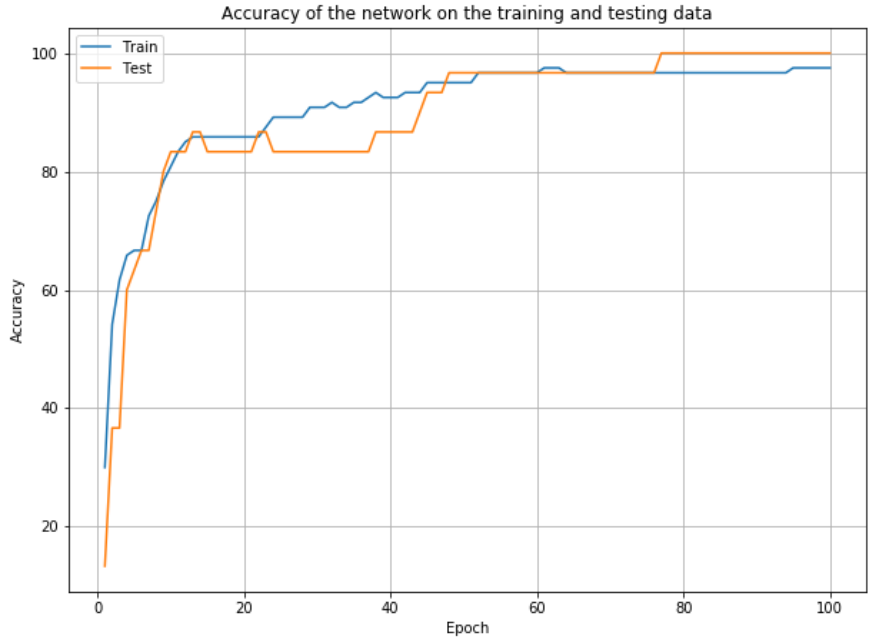
1. 使用的三层神经网络结构为：

- 输入层维度为 4（数据特征维度为 4）
- 隐藏层维度为 11
- 输出层维度为 3（数据分为 3 类）



2. ReLU 激活函数结果

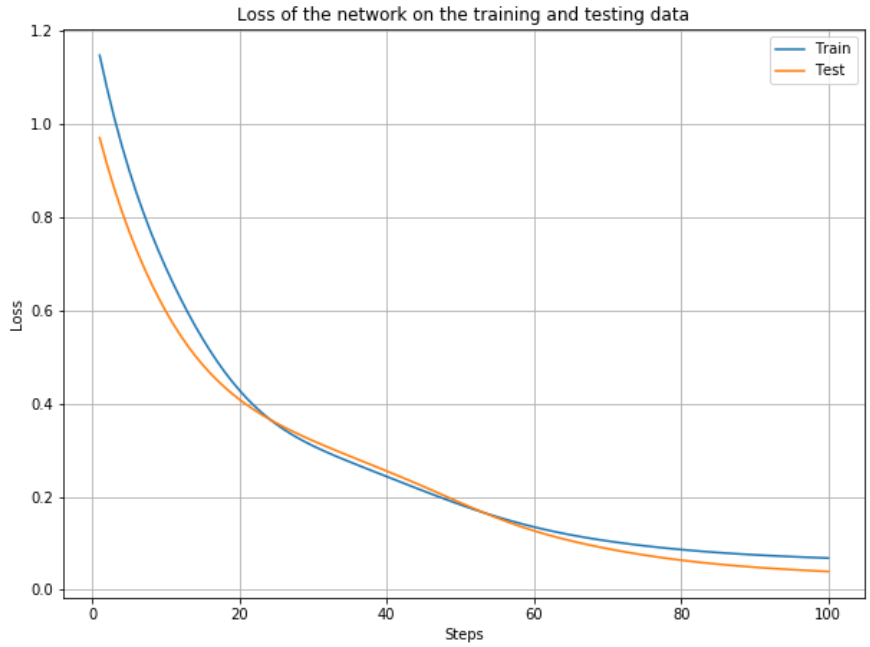


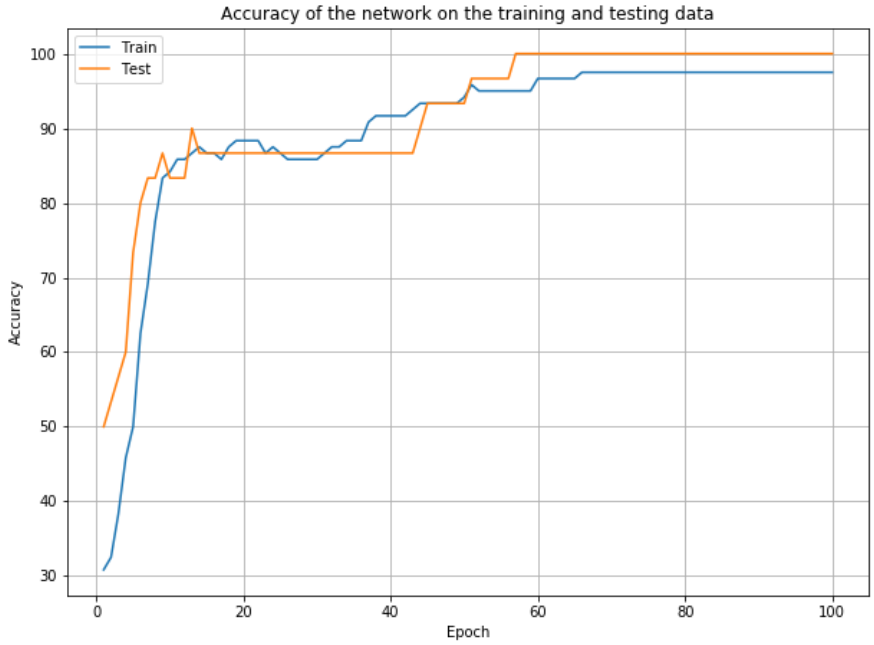


在 iris 数据集上最终达到 100%正确率。

```
[epoch: 97]|training loss: 0.067396| testing loss: 0.043769
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
[epoch: 98]|training loss: 0.066870| testing loss: 0.043086
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
[epoch: 99]|training loss: 0.066356| testing loss: 0.042449
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
[epoch: 100]|training loss: 0.065855| testing loss: 0.041845
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
Finished Training
```

3. ELU 激活函数结果

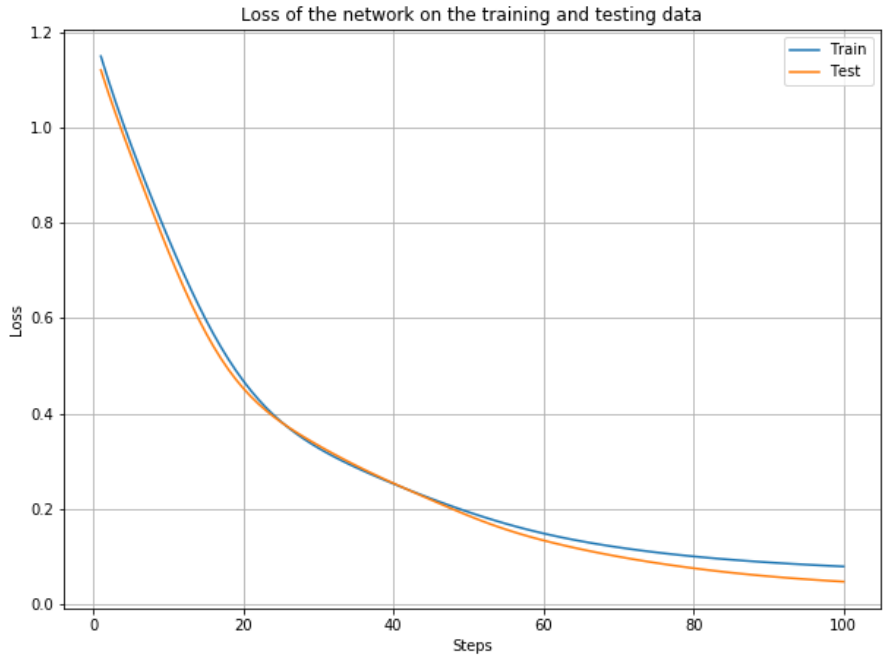


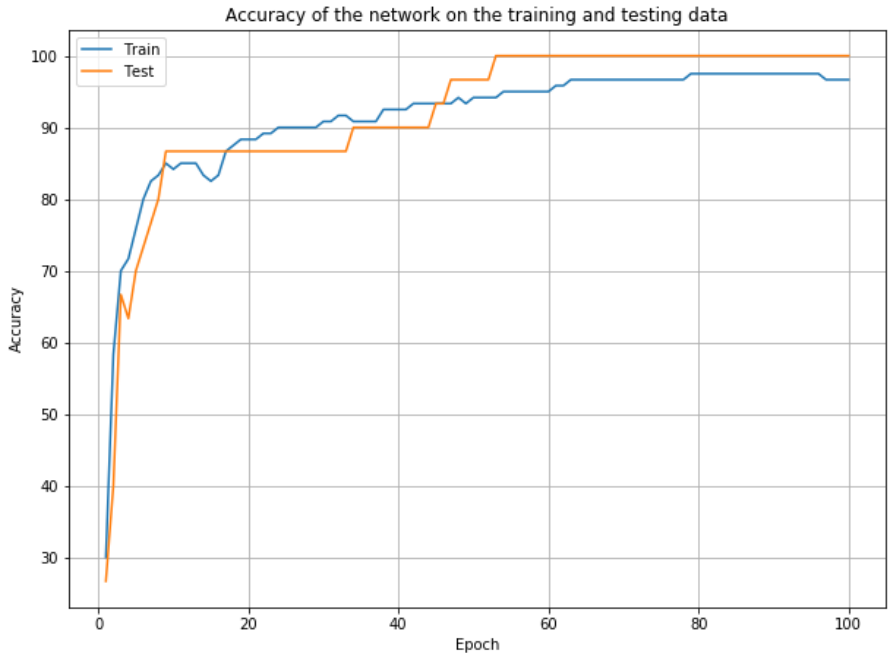


在 iris 数据集上最终达到 100%正确率。

```
[epoch: 97]|training loss: 0.070279| testing loss: 0.042022
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
[epoch: 98]|training loss: 0.069664| testing loss: 0.041215
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
[epoch: 99]|training loss: 0.069074| testing loss: 0.040443
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
[epoch: 100]|training loss: 0.068507| testing loss: 0.039704
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
Finished Training
```

4. Tanh 激活函数结果

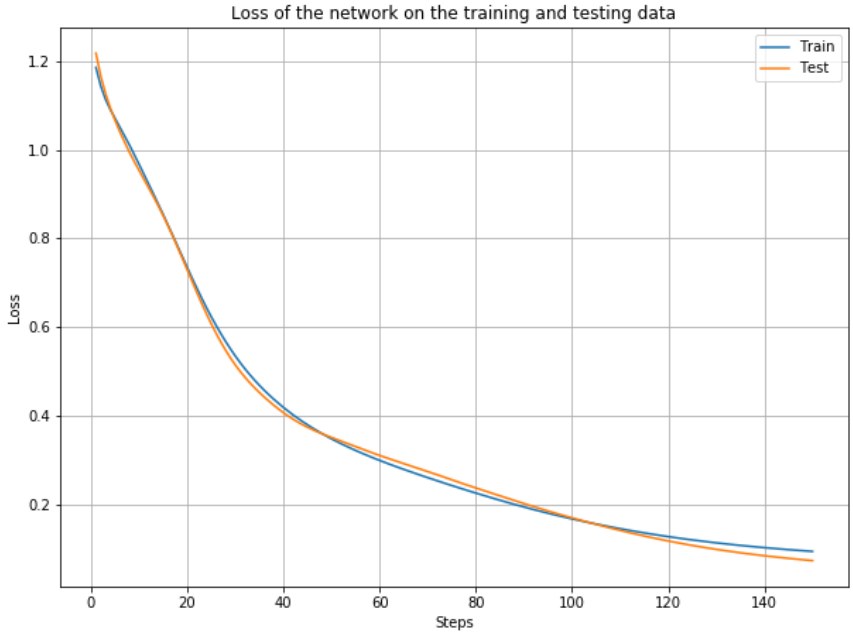


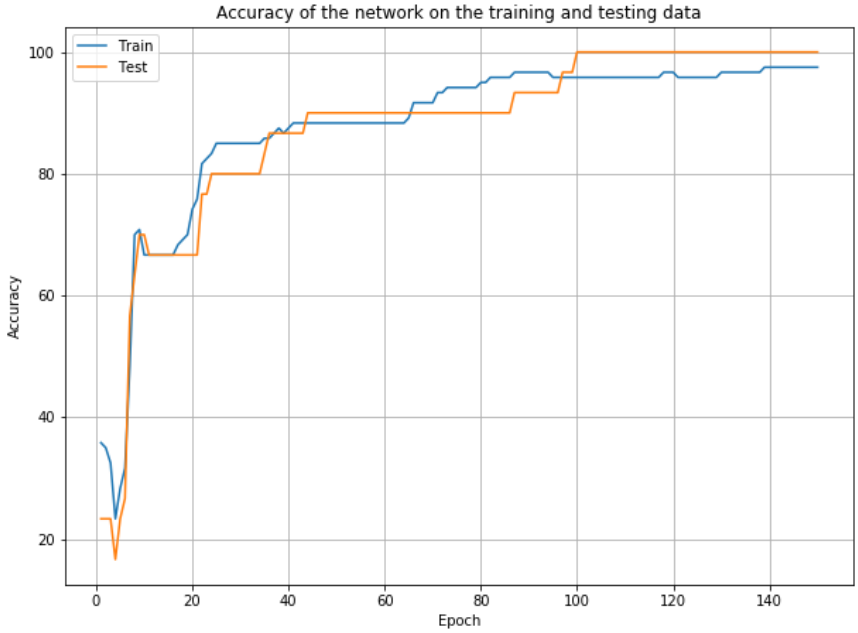


在 iris 数据集上最终达到 100%正确率。

```
[epoch: 97]|training loss: 0.080685| testing loss: 0.049429
Accuracy of the network on the train data: 96.667 %
Accuracy of the network on the test data: 100.000 %
[epoch: 98]|training loss: 0.079901| testing loss: 0.048420
Accuracy of the network on the train data: 96.667 %
Accuracy of the network on the test data: 100.000 %
[epoch: 99]|training loss: 0.079143| testing loss: 0.047452
Accuracy of the network on the train data: 96.667 %
Accuracy of the network on the test data: 100.000 %
[epoch: 100]|training loss: 0.078410| testing loss: 0.046524
Accuracy of the network on the train data: 96.667 %
Accuracy of the network on the test data: 100.000 %
Finished Training
```

5. Sigmoid 激活函数结果





在 iris 数据集上最终达到 100%正确率。

```
[epoch: 147]|training loss: 0.096495| testing loss: 0.076361
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
[epoch: 148]|training loss: 0.095707| testing loss: 0.075337
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
[epoch: 149]|training loss: 0.094938| testing loss: 0.074338
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
[epoch: 150]|training loss: 0.094187| testing loss: 0.073365
Accuracy of the network on the train data: 97.500 %
Accuracy of the network on the test data: 100.000 %
Finished Training
```

6. 分析与讨论

通过调整四种激活函数的参数，最终使用{ReLU、ELU、Tanh、Sigmoid}四种激活函数均在 iris 数据集上取得了 100%的测试正确率。

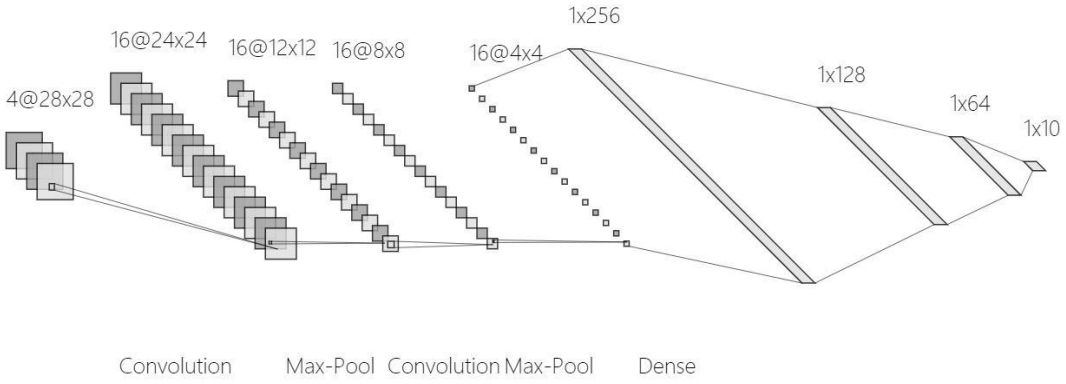
但这四者还是有很大区别的：

- Sigmoid 函数饱和会导致梯度消失，当神经元的激活值在接近 0 或 1 时，就会产生饱和，在这些区域梯度几乎为 0，这就会导致梯度消失；而且 Sigmoid 函数的输出不是零中心的。基于以上两个问题，通常在预处理输入值时会将输入值进行均值方差归一化，使得网络能够进行有效训练
- Tanh 解决了 Sigmoid 的输出不是零中心的问题，但仍然存在饱和的问题，因此也需要对输入输出进行均值方差归一化，或者假如 batch normalization 层，尽可能保证每层网络的输入具有均值较小、零中心的分布。
- ReLU 相较于 Sigmoid 和 Tanh 函数，ReLU 对于随机梯度下降的收敛有巨大的加速作用，ReLU 求导几乎不用任何计算量，但其问题在于 ReLU 单元不可逆，而且当网络较浅时，ReLU 无法有效地提升网络模型的非线性，而导致模型表现能力下降。
- ELU 融合了 Sigmoid 和 ReLU，其左侧具有软饱和性，右侧无饱和性，其输出均值接近于零，因此收敛更快。

实践题 1-2

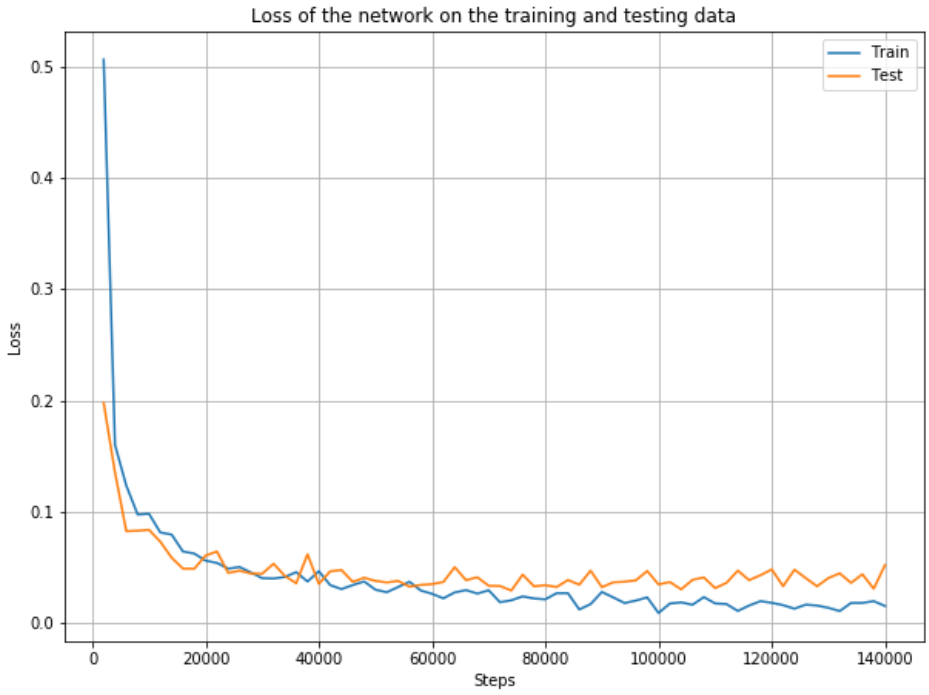
设计并实现一个深度学习网络结构，能够在 MNIST 数据集上（前 6w 个训练，后 1w 个测试）获得至少 99%的测试精度。

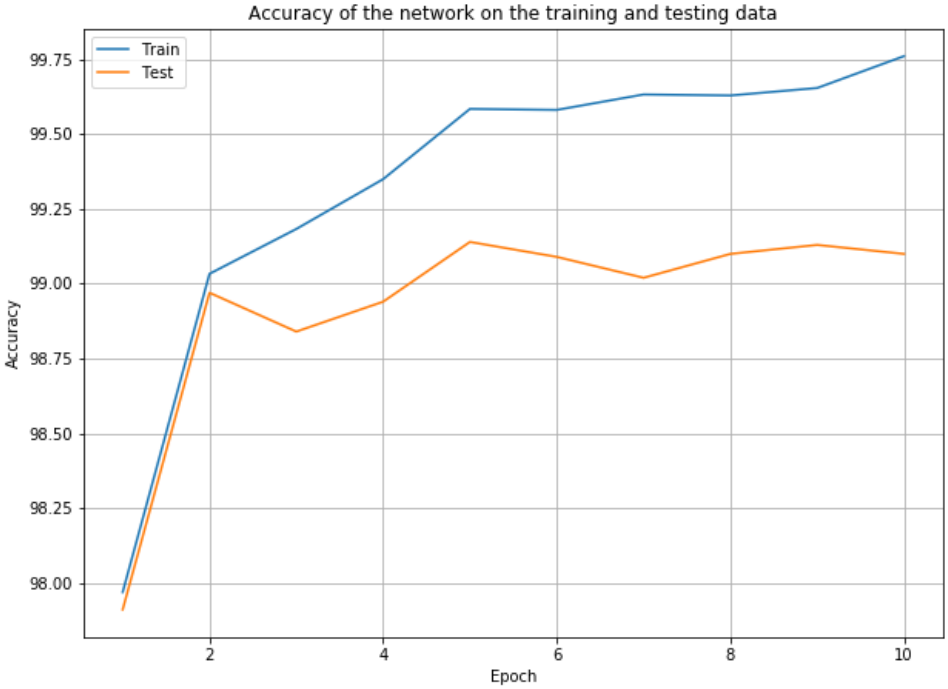
1. MNIST 数据集
   MNIST 数据集（Mixed National Institute of Standards and Technology database）是美国国家标准与技术研究院收集整理的大型手写数字数据库，包含 60000 个示例的训练集以及 10000 个示例的测试集。
2. 深度网络结构
   采用了两层卷积层及三层全连接层，为防止训练时过拟合，在最后一层全连接层前添加了一层 dropout 层，但在预测过程中不会启用 dropout 层。其网络结构可视化如下图所示：



3. 训练过程中由于是分类问题，因此采用交叉熵损失函数作为误差判断依据，并使用 Adam 优化器在反向梯度传播时对网络参数进行优化，并且将训练数据 batch size 设置为 4，一共训练 10 个 epoch，期间没有使用渐进下降学习率，学习率设置为 0.0003。
4. 最终在 MNIST 测试集上稳定达到 99%以上的正确率，训练集及测试集误差曲线和精度曲线如下图所示。

```
Accuracy of the network on the train data: 99.633 %
Accuracy of the network on the test data: 99.020 %
[8,  2000]|training loss: 0.009224| testing loss: 0.034836
[8,  4000]|training loss: 0.017741| testing loss: 0.037054
[8,  6000]|training loss: 0.018785| testing loss: 0.030438
[8,  8000]|training loss: 0.016601| testing loss: 0.039015
[8, 10000]|training loss: 0.023506| testing loss: 0.041147
[8, 12000]|training loss: 0.017826| testing loss: 0.031548
[8, 14000]|training loss: 0.017253| testing loss: 0.036296
 80%|████████████████████████████████████████████████████████████████████████████
| 8/10 [17:52<04:27, 133.89s/it]
Accuracy of the network on the train data: 99.630 %
Accuracy of the network on the test data: 99.100 %
[9,  2000]|training loss: 0.011076| testing loss: 0.047255
[9,  4000]|training loss: 0.015945| testing loss: 0.038717
[9,  6000]|training loss: 0.019871| testing loss: 0.043154
[9,  8000]|training loss: 0.018383| testing loss: 0.048362
[9, 10000]|training loss: 0.016250| testing loss: 0.033273
[9, 12000]|training loss: 0.013110| testing loss: 0.048132
[9, 14000]|training loss: 0.016731| testing loss: 0.040557
 90%|████████████████████████████████████████████████████████████████████████████
          | 9/10 [20:06<02:13, 133.89s/it]
Accuracy of the network on the train data: 99.655 %
Accuracy of the network on the test data: 99.130 %
[10,  2000]|training loss: 0.015920| testing loss: 0.033214
[10,  4000]|training loss: 0.013894| testing loss: 0.040604
[10,  6000]|training loss: 0.010953| testing loss: 0.044823
[10,  8000]|training loss: 0.018243| testing loss: 0.036250
[10, 10000]|training loss: 0.018306| testing loss: 0.044021
[10, 12000]|training loss: 0.019850| testing loss: 0.031017
[10, 14000]|training loss: 0.015532| testing loss: 0.052287
100%|████████████████████████████████████████████████████████████████████████████
                    | 10/10 [22:18<00:00, 133.88s/it]
Accuracy of the network on the train data: 99.762 %
Accuracy of the network on the test data: 99.100 %
Finished Training
```

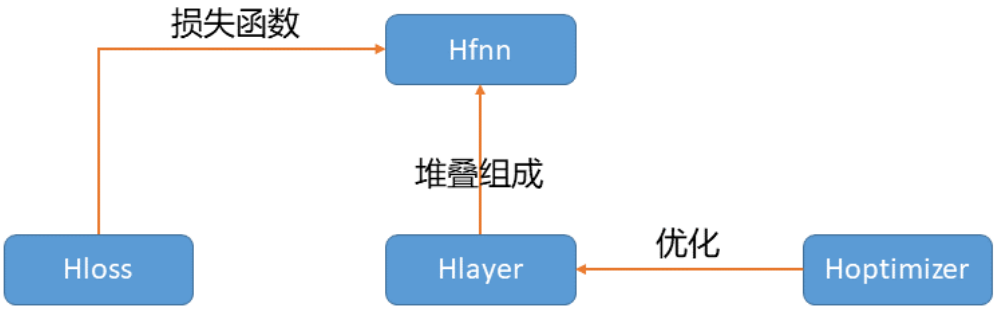

Loss of the network on the training and testing data
Train
Test
Loss
Steps
0.0
0.1
0.2
0.3
0.4
0.5
0
20000
40000
60000
80000
100000
120000
140000


Accuracy of the network on the training and testing data
Train
Test
Accuracy
Epoch
98.00
98.25
98.50
98.75
99.00
99.25
99.50
99.75
2
4
6
8
10

实践题 2

1）仅使用 numpy 实现三层神经网络 BP 训练算法（输入 d 维向量，中间 h 个隐含神经元，输出 c>1 类单热向量编码，隐含层使用 sigmoid 激活函数，输入输出层使用线性激活函数），损失函数用均方误差或者交叉熵。
2）在 iris 数据集上对 1）中实现的算法测试，并与实践题一结果进行比较。

使用 numpy 实现三层神经网络时，我参照了 tensorflow 构建训练模型时静态计算图的构建方式，采用自顶向下模块化设计方法。



其中 Hlayer 表示层父类，Hloss 表示损失函数父类，Hoptimizer 表示优化器父类，当需要进行新的网络层或损失函数或优化器设计时，就继承对应父类属性。

在实现过程中设计完成：

- 输入层：Hinput
- Sigmoid 层：Hsigmoid
- 线性全连接层：Hlinear
- 交叉熵损失函数：HCrossEntropy
- Sgd 随机梯度下降优化器：Hsgd

最终通过 Hfnn 神经网络类将三个模块进行封装，Hfnn 有三个成员变量分别为 self.sequential 用于存储堆叠的实例化 Hlayer 对象；self.len_sequential 用于记录网络堆叠层深度；self.criterion 用于存储网络损失函数 Hloss 对象。

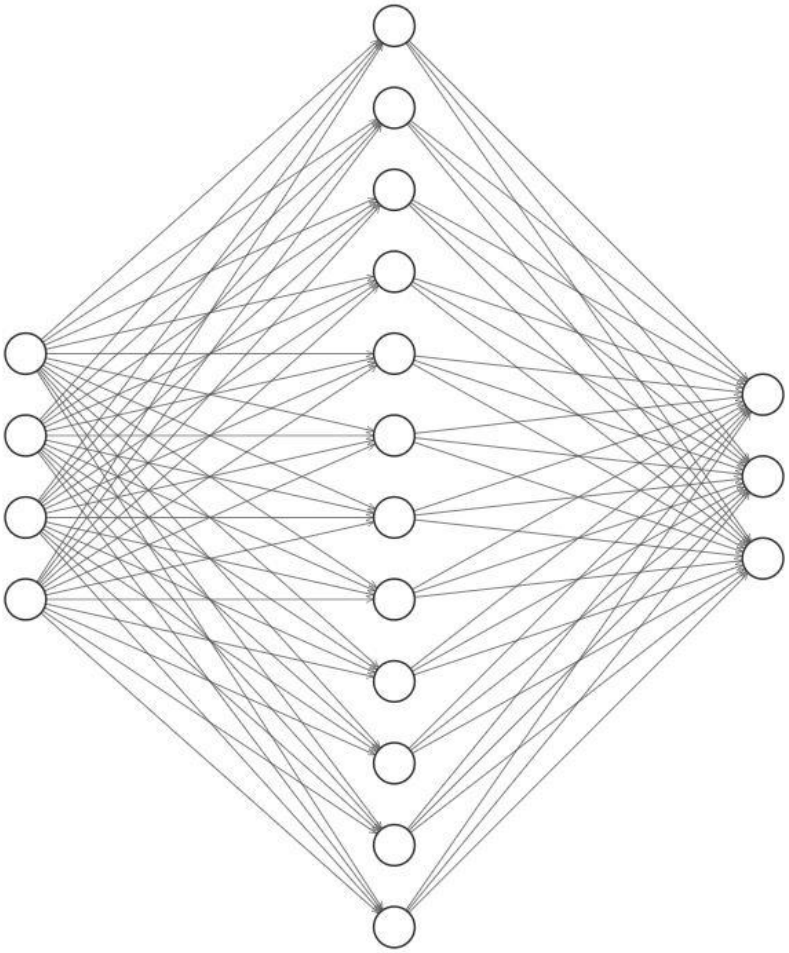
其包含以下成员函数：

- add_layer 函数用于添加实例化层
- add_criterion 函数用于添加损失函数
- 可直接调用网络实例化对象计算输出结果
- loss 函数得到网络损失函数输出
- predict 函数得到网络分类编号输出
- backward 函数反向传播计算网络可训练层梯度
- step 函数对网络可训练层使用指定优化器进行参数更新
- summary 函数可以打印网络实例化对象基本信息

神经网络实例化时采用以下结构：

- 输入层维度为 4（数据特征维度为 4）
- 隐藏层维度为 12

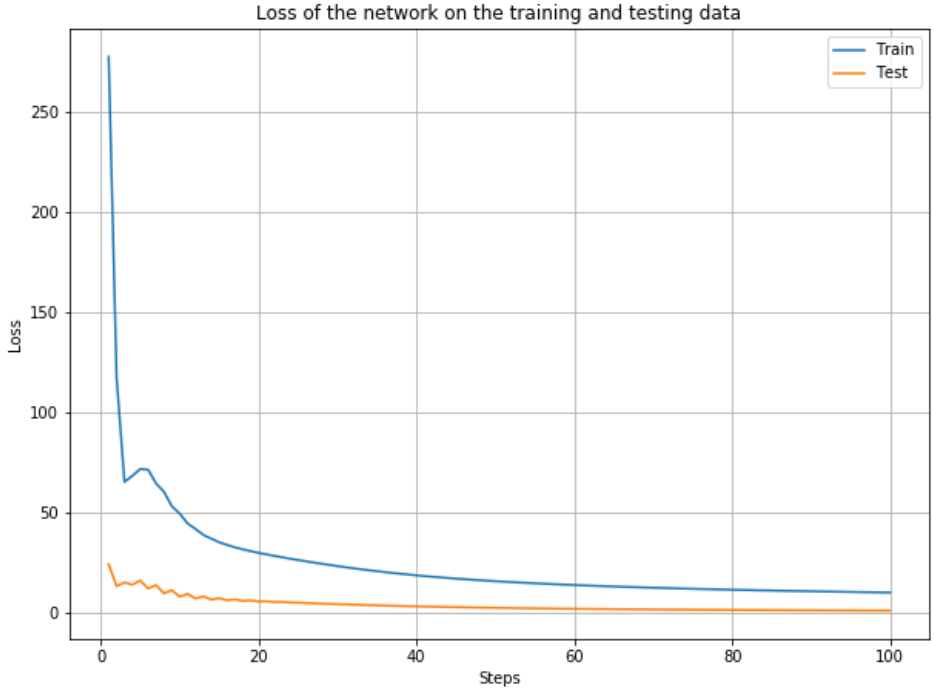
- 输出层维度为 3（数据分为 3 类）

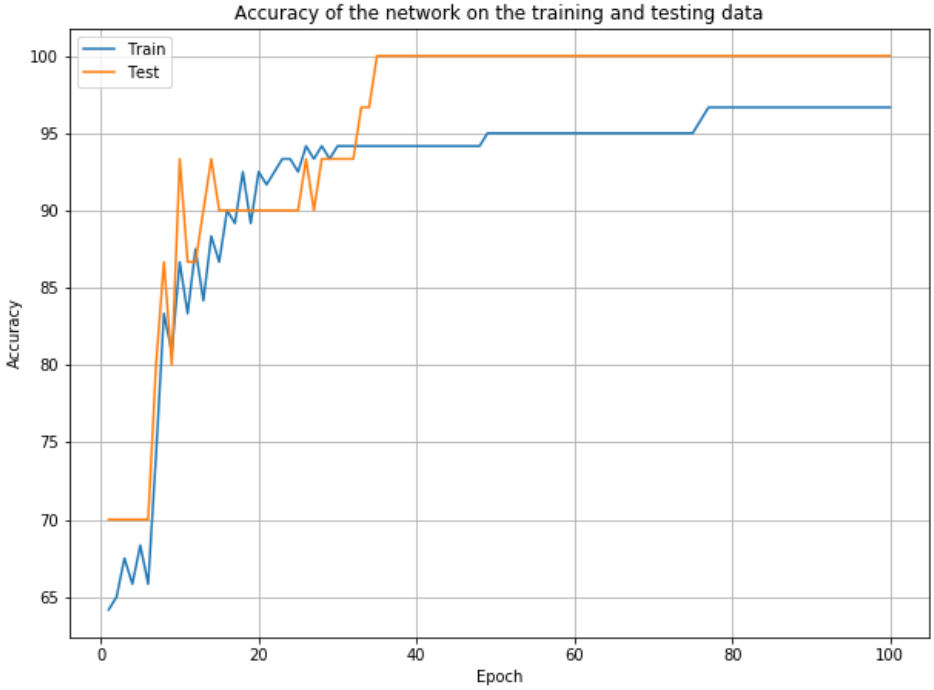


在 iris 数据集上也能达到 100%正确率：

```
[epoch: 97]|training loss: 10.606480| testing loss: 1.444284
Accuracy of the network on the train data: 96.667 %
Accuracy of the network on the test data: 100.000 %
[epoch: 98]|training loss: 10.547669| testing loss: 1.430832
Accuracy of the network on the train data: 96.667 %
Accuracy of the network on the test data: 100.000 %
[epoch: 99]|training loss: 10.490032| testing loss: 1.417697
Accuracy of the network on the train data: 96.667 %
Accuracy of the network on the test data: 100.000 %
[epoch: 100]|training loss: 10.433533| testing loss: 1.404868
Accuracy of the network on the train data: 96.667 %
Accuracy of the network on the test data: 100.000 %
Finished Training
```

以下是网络在训练集和测试集上的误差曲线以及精度曲线。





从训练以及在测试集上的结果可以看出，使用 numpy 实现的三层神经网络也能够非常有效地收敛得到精准预测 iris 数据集。

同时我设计的模块化可堆叠的网络架构，可拓展性相比较于直接设计三层神经网络更强，可以按需求堆叠不同神经网络层，同时也可以通过继承三个顶层模块父类，设计新的网络层添加到网络中。损失函数和优化器也为以后的进一步设计提供了良好的接口。

总结

在此次作业中，我采用了 pytorch 深度学习框架自己搭建了神经网络在简单数据集 iris 以及深度学习入门数据集 MNIST 分别进行了训练测试，对于深度学习网络的搭建以及调整网络结构以及训练参数在测试集上获取更高正确率有了更深刻的体会。

作业第二部分自己使用 numpy 实现三层神经网络更是给予了我一次学习实现深度学习框架，模仿其设计思想构建自己的深度学习体系框架的机会，在此过程中虽然遇到了一些编程实现的困难，但在查阅资料和与同学共同讨论的过程中不仅对神经网络的实现有了更加详细的了解，对其正向传播计算及反向传播计算梯度的优化方法也有了更清晰的思路和认识。