目录

[基于前馈神经网络的分类任务设计 1](#_Toc153736927)

[一、神经网络架构 1](#_Toc153736928)

[模型结构 1](#_Toc153736929)

[二、训练过程 2](#_Toc153736930)

[数据预处理 2](#_Toc153736931)

[训练阶段 2](#_Toc153736932)

[评估阶段 3](#_Toc153736933)

[训练和测试损失及准确率 3](#_Toc153736934)

[结果分析 5](#_Toc153736935)

[三、其它尝试 5](#_Toc153736936)

[1.修改神经元数量 5](#_Toc153736937)

[2.增加网络层数 6](#_Toc153736938)

[四、总结 6](#_Toc153736939)

# 基于前馈神经网络的分类任务设计

## 一、神经网络架构

### 模型结构

本实验使用的神经网络模型为GaussianClassifier，这是一个简单的全连接网络。具体架构如下：

* 输入层：接受大小为input\_size的输入。
* 隐藏层：一个全连接层（fc1），包含hidden\_size个神经元，后接ReLU激活函数。
* 输出层：一个全连接层（fc2），输出大小为num\_classes，对应于分类任务的类别数。
* *class* GaussianClassifier(*nn*.*Module*):
* *def* *\_\_init\_\_*(self, input\_size, hidden\_size, num\_classes):
* super(GaussianClassifier, *self*).*\_\_init\_\_*()
* *self*.fc1 = nn.Linear(input\_size, hidden\_size)
* *self*.relu = nn.ReLU()
* *self*.fc2 = nn.Linear(hidden\_size, num\_classes)
* *def* forward(self, x):
* out = *self*.fc1(x)
* out = *self*.relu(out)
* out = *self*.fc2(out)
* return out

超参数：2个特征，100个隐藏层神经元，4个类别

input\_size = 2

hidden\_size = 100

num\_classes = 4

该前馈神经网络通过ReLU激活函数增加非线性，使得模型能够学习更复杂的数据表示。

## 二、训练过程

### 数据预处理

包括随机排序、划分训练集和测试集等。

from sklearn.model\_selection import train\_test\_split

……

# 随机排序

data\_shuffled = data.sample(frac=1).reset\_index(drop=*True*)

X = data\_shuffled[['data1', 'data2']].values

y = data\_shuffled['label'].values - 1  # 减1是因为pytorch的loss函数要求类别从0开始

# 10%用于测试，90%用于训练

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42)

### 训练阶段

        model.train()

        running\_loss\_train = 0.0

        correct\_train = 0

        total\_train = 0

        for inputs, labels in train\_loader:

            optimizer.zero\_grad()

            outputs = model(inputs)

            loss = criterion(outputs, labels)

            loss.backward()

            optimizer.step()

            running\_loss\_train += loss.item() \* inputs.size(0)

            \_, predicted = torch.max(outputs, 1)

            total\_train += labels.size(0)

            correct\_train += (predicted == labels).sum().item()

        epoch\_loss\_train = running\_loss\_train / *len*(train\_loader.dataset)

        train\_accuracy = 100 \* correct\_train / total\_train

### 评估阶段

        model.eval()

        running\_loss\_test = 0.0

        correct\_test = 0

        total\_test = 0

        with torch.no\_grad():

            for inputs, labels in test\_loader:

                outputs = model(inputs)

                loss = criterion(outputs, labels)

                running\_loss\_test += loss.item() \* inputs.size(0)

                \_, predicted = torch.max(outputs, 1)

                total\_test += labels.size(0)

                correct\_test += (predicted == labels).sum().item()

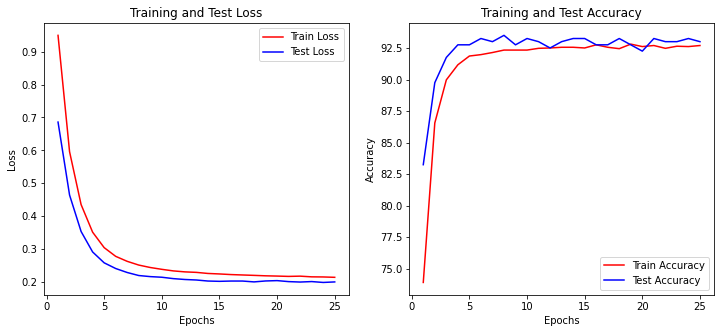
        epoch\_loss\_test = running\_loss\_test / *len*(test\_loader.dataset)

        test\_accuracy = 100 \* correct\_test / total\_test

### 训练和测试损失及准确率

在25个训练周期（epoch）中，模型在训练集和测试集上的损失（Loss）和准确率（Accuracy）如下所示：

| Epoch | Training Loss | Test Loss | Training Accuracy | Test Accuracy |
| --- | --- | --- | --- | --- |
| 1 | 0.9781 | 0.7136 | 71.31% | 87.75% |
| 2 | 0.5679 | 0.4765 | 89.61% | 89.25% |
| 3 | 0.4065 | 0.3697 | 91.19% | 90.50% |
| 4 | 0.3300 | 0.3118 | 91.81% | 90.25% |
| 5 | 0.2911 | 0.2860 | 92.00% | 91.00% |
| 6 | 0.2699 | 0.2642 | 92.36% | 91.25% |
| 7 | 0.2544 | 0.2553 | 92.36% | 91.75% |
| 8 | 0.2466 | 0.2444 | 92.50% | 91.75% |
| 9 | 0.2395 | 0.2423 | 92.36% | 91.50% |
| 10 | 0.2339 | 0.2348 | 92.58% | 92.00% |
| 11 | 0.2302 | 0.2308 | 92.61% | 92.25% |
| 12 | 0.2266 | 0.2314 | 92.58% | 91.75% |
| 13 | 0.2253 | 0.2330 | 92.67% | 91.75% |
| 14 | 0.2236 | 0.2306 | 92.53% | 91.75% |
| 15 | 0.2222 | 0.2255 | 92.50% | 92.50% |
| 16 | 0.2189 | 0.2231 | 92.64% | 92.25% |
| 17 | 0.2181 | 0.2222 | 92.78% | 92.50% |
| 18 | 0.2159 | 0.2238 | 92.69% | 91.75% |
| 19 | 0.2159 | 0.2203 | 92.75% | 92.25% |
| 20 | 0.2154 | 0.2209 | 92.42% | 92.25% |
| 21 | 0.2137 | 0.2199 | 92.61% | 92.50% |
| 22 | 0.2127 | 0.2179 | 92.69% | 92.25% |
| 23 | 0.2129 | 0.2219 | 92.64% | 92.00% |
| 24 | 0.2126 | 0.2202 | 92.72% | 92.00% |
| 25 | 0.2115 | 0.2212 | 92.67% | 92.25% |
| 24 | 0.2126 | 0.2202 | 92.72% | 92.00% |
| 25 | 0.2115 | 0.2212 | 92.67% | 92.25% |
|  |  |  |  |  |



### 结果分析

**快速初始学习**：从第1个epoch到第2个epoch，模型在测试集上的准确率显著提高，从87.75%跃升到89.25%，同时训练损失和测试损失都显著下降，表明模型能够迅速从数据中学习。

**稳定收敛**：随着训练的进行，模型的训练损失和测试损失逐渐下降，并趋于稳定。训练集和测试集的准确率也逐渐提高并稳定在92%左右，表明模型对数据具有良好的泛化能力。

**过拟合情况**：观察训练损失和测试损失可以发现，两者始终保持较为接近的水平，没有出现显著的过拟合现象。这表明模型复杂度适中，能够很好地适应数据集的特性。

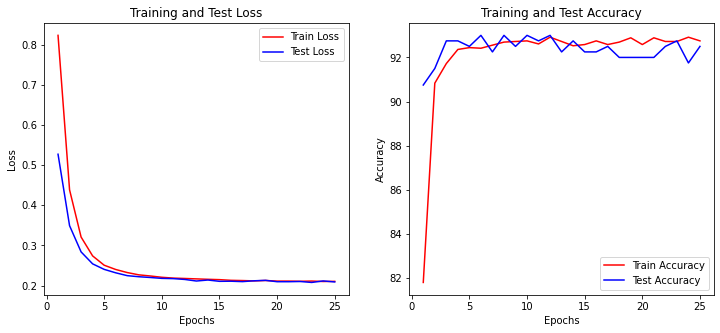
**性能瓶颈**：尽管模型性能稳定，但准确率的提升在后期逐渐放缓，这可能意味着模型出现了性能瓶颈，需要进一步的参数调优和结构改进来提升性能。

## 三、其它尝试

### 1.增加神经元数量

将神经元数量从100改为200，训练结果如下：

| Epoch | Training Loss | Test Loss | Training Accuracy | Test Accuracy |
| --- | --- | --- | --- | --- |
| 1 | 0.8230 | 0.5271 | 81.81% | 90.75% |
| 2 | 0.4384 | 0.3492 | 90.83% | 91.25% |
| 3 | 0.3209 | 0.2893 | 91.72% | 92.75% |
| … | … | … | … | … |
| 25 | 0.2106 | 0.2091 | 92.75% | 92.50% |



可以看到增加神经元数量很大程度上提高了模型前期的学习速度，模型更快速地到达了稳定阶段，并且随着训练的深入，200个神经元的模型并没有比100个神经元的模型预测准确率有明显提高，但最终的损失值略有降低。

### 2.增加网络层数

增加一层隐藏层

*class* GaussianClassifier(*nn*.*Module*):

*def* *\_\_init\_\_*(self, input\_size, hidden\_size, num\_classes):

        super(GaussianClassifier, *self*).*\_\_init\_\_*()

        # 第一个隐藏层

*self*.fc1 = nn.Linear(input\_size, hidden\_size)

        # 新增的第二个隐藏层

*self*.fc2 = nn.Linear(hidden\_size, hidden\_size)

        # 输出层

*self*.fc3 = nn.Linear(hidden\_size, num\_classes)

*self*.relu = nn.ReLU()

*def* forward(self, x):

        out = *self*.fc1(x)

        out = *self*.relu(out)

        # 通过第二个隐藏层

        out = *self*.fc2(out)

        out = *self*.relu(out)

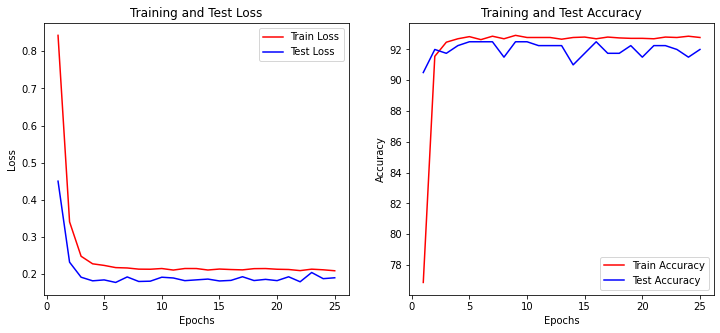
        # 通过输出层

        out = *self*.fc3(out)

        return out

后（神经元数量为100），训练结果如下：

| Epoch | Training Loss | Test Loss | Training Accuracy | Test Accuracy |
| --- | --- | --- | --- | --- |
| 1 | 0.8429 | 0.4501 | 76.86% | 90.50% |
| 2 | 0.3405 | 0.2319 | 91.56% | 92.00% |
| 3 | 0.2478 | 0.1914 | 92.47% | 91.75% |
| … | … | … | … | … |
| 25 | 0.2087 | 0.1899 | 92.78% | 92.00% |



可以看到虽然每层只有100个神经元，但有2个隐藏层的模型（深度模型）比1个隐藏层、200个神经元的模型（宽度模型）在处理具有空间结构和复杂决策边界的数据集（如本实验的二维高斯数据集）的任务中表现更好。

## 四、总结

本实验使用的GaussianClassifier模型在给定的任务中表现良好，能够在保证不过拟合的前提下达到较高的准确率。

在训练初期（Epoch1），测试集准确率暂时高于训练集准确率可能是由于模型尚未足够复杂以拟合训练数据以及训练数据中的噪声干扰；随着训练的深入（Epoch2到Epoch10），模型在训练数据上短暂出现过拟合，表现为训练准确率高于测试准确率，这反映了模型对训练数据特征的精确学习，但缺乏泛化能力。随着训练的继续进行（Epoch10及以后），训练准确率与测试准确率之间的差距逐渐缩小，模型泛化能力得到提高。

然而，模型的性能提升在后期趋于平缓，表明可能需要进一步的策略来突破性能瓶颈。未来工作可以包括调整模型结构、尝试不同的优化算法或学习率调整等，以期望获得更优的性能。