数学建模实验报告模版

实验三　数据拟合模型

姓 名: 王 世 炟
学 号: PB20151796
学 院: 大数据学院

001139.01 数学建模
(春季, 2023)

中国科学技术大学

2023 年 4 月 26 日

目 录

1 摘要

本文使用基础的神经网络模型实现根据体长和翼长对于昆虫数据集进行分类，并进行了训练和测试，具有较高的准确率和鲁棒性。并分析了有关参数对于实验效果的影响。

2 前言

随着人工智能技术的不断发展，神经网络模型在各个领域得到了广泛的应用。其中，利用神经网络模型进行分类是其中的一个重要应用方向。昆虫分类作为生物学研究的重要分支之一，对于了解生物多样性和生态系统的结构和功能具有重要意义。本实验旨在利用神经网络模型对昆虫进行分类，通过对昆虫体长和翼长等特征的提取和分析，构建适合分类的神经网络模型，并进行训练和测试。通过本实验的研究，可以进一步理解深度学习在分类上的应用。

3 问题分析

本实验的主要问题是如何利用神经网络模型对昆虫数据集进行分类。具体来说，需要解决以下几个问题：

1. 如何利用神经网络模型完成多分类问题；
2. 如何选择合适的神经网络结构和激活函数，以提高分类准确率和鲁棒性；
3. 如何评估神经网络模型的性能，并分析其影响因素。

4 理论基础

4.1 神经网络

神经网络是一种模拟人脑神经元之间相互连接的计算模型，它可以通过学习和训练来实现对数据的分类、预测和识别等任务。神经网络由多个神经元组成，每个神经元接收多个输入信号，通过加权和和激活函数的处理后输出一个结果。神经元之间通过连接权重相互连接，形成多层的网络结构。神经网络的训练过程是通过反向传播算法来更新连接权重，使得网络的输出结果与实际结果之间的误差最小化。神经网络具有自适应性、非线性映射和容错性等特点，可以应用于图像识别、语音识别、自然语言处理、预测分析等领域。

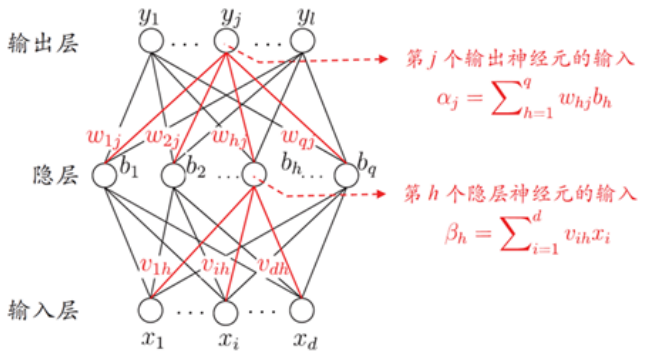


图 1: 神经网络

4.2 激活函数

神经网络激活函数是神经元的非线性变换函数，它将神经元的输入信号转换为输出信号，使得神经网络可以处理非线性问题。常见的激活函数包括 sigmoid 函数、ReLU 函数、tanh 函数等。

sigmoid 函数是一种常用的激活函数，它将输入信号映射到 0 到 1 之间的值，具有平滑的 S 形曲线。sigmoid 函数的输出值在 0 和 1 之间，可以用于二分类问题的输出层。但是，sigmoid 函数在输入信号较大或较小时，梯度会趋近于 0，导致梯度消失的问题。

ReLU 函数是一种近年来广泛使用的激活函数，它将输入信号映射到 0 和正无穷之间的值，具有简单的线性形式。ReLU 函数的输出值在输入信号大于 0 时为输入信号本身，在输入信号小于等于 0 时为 0，可以有效地解决梯度消失的问题。

tanh 函数是一种双曲正切函数，它将输入信号映射到-1 到 1 之间的值，具有 S 形曲线。tanh 函数的输出值在 0 附近变化，可以用于多分类问题的输出层。但是，tanh 函数也存在梯度消失的问题。

不同的激活函数适用于不同的神经网络结构和任务，选择合适的激活函数可以提高神经网络的性能和鲁棒性。

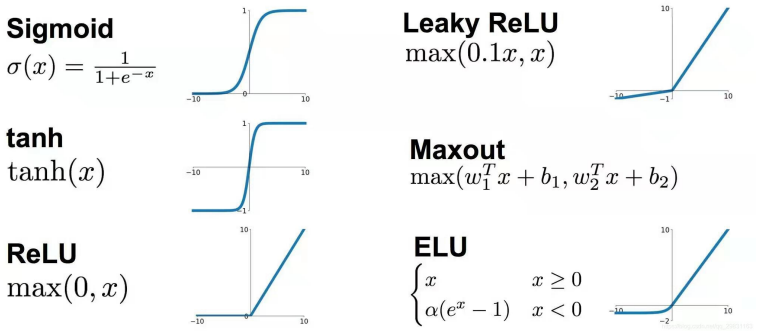


图 2: 激活函数

4.3 多分类

利用神经网络解决多分类问题是神经网络应用的重要领域之一。神经网络可以通过多个输出神经元来实现多分类任务，每个输出神经元对应一个类别，输出值表示该样本属于该类别的概率。在训练过程中，通常采用交叉熵损失函数来衡量预测结果与实际结果之间的误差，并通过反向传播算法来更新连接权重。

为了实现输出为某一类别的概率，我们需要对最后一层线性层的输出进行处理使得最终输出在 [0,1] 范围内。通常采用 **softmax() 函数**:

softmax 函数是一种常用的激活函数，主要用于多分类问题中。它将一个向量映射为另一个向量，使得每个元素的值都在 0 到 1 之间，并且所有元素的和为 1。softmax 函数的公式如下：

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

其中，z 是一个向量，j 表示向量中的第 j 个元素，K 表示向量的长度。$\sigma(z)_j$ 表示向量 z 中第 j 个元素经过 softmax 函数处理后的值。那么我们预测的结果就是输出的概率最大值所对应的类别。

5 模型建立

我们使用 PyTorch 手写前馈神经网络，包含一个输入层、numlayers-1 个隐藏层和一个输出层。我们使用 ReLU/tanh/sigmoid 三者之一作为激活函数，使用交叉熵作为损失函数，使用 Adam 优化器进行参数更新。在训练过程中，我们记录了模型在训练集上的损失，并绘制了损失曲线。(为了保证可复现，给定了初始化网络参数的随机种子)

其中输入层有 2 个神经元，输出层有 3 个神经元。可调的参数有隐藏层层数，隐藏层宽度，学习率，激活函数。

给出网络模型的代码，见附录A。

6 结果（与对比）

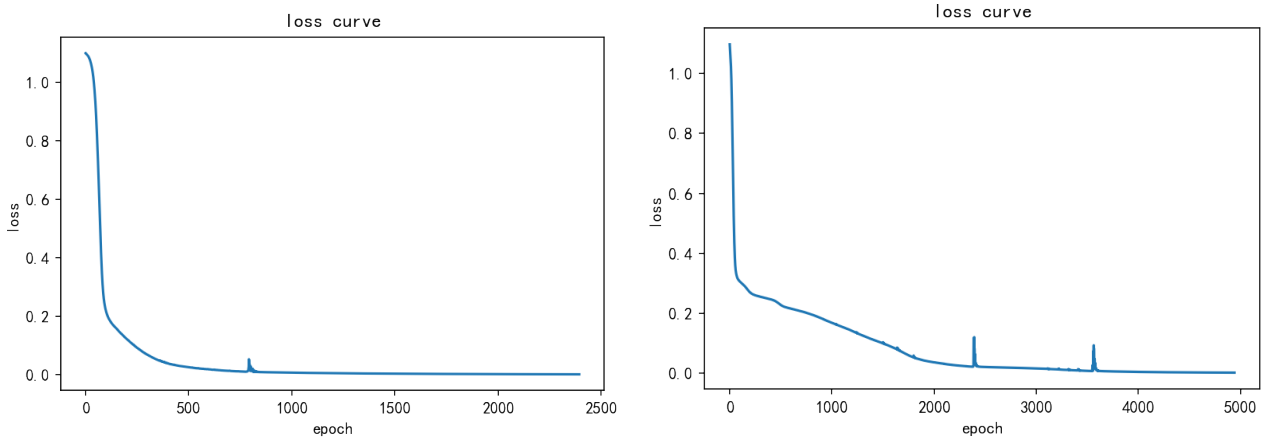
6.1 最优结果与损失曲线

首先给出两组数据集在最优参数下的预测结果：

数据集	epoch	train loss	test accuracy
无噪声	2395	0.000999	1.000000
有噪声	3997	0.000999	0.971429

表 1: 实验结果

以及两条损失曲线：

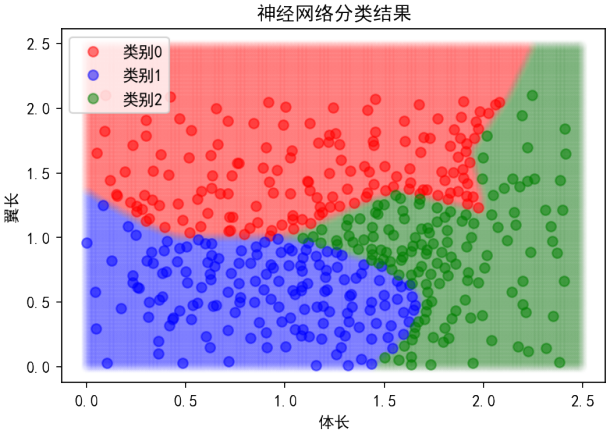


(a)dataset without noise epoch-loss curve

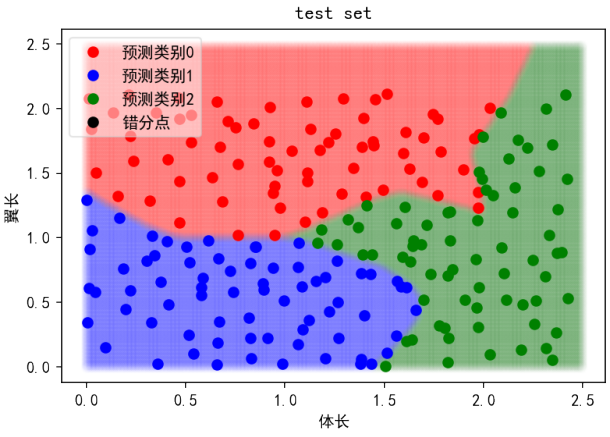
(b)dataset with noise epoch-loss curve

然后我们对神经网络的训练结果进行可视化，可以得到如下结果：

无噪声数据：

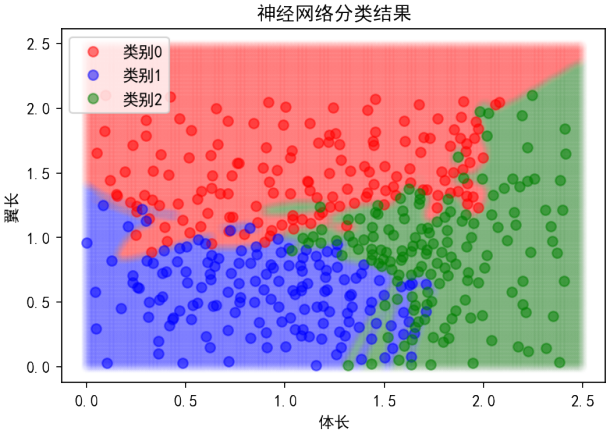


(c) 训练集以及神经网络决策边界

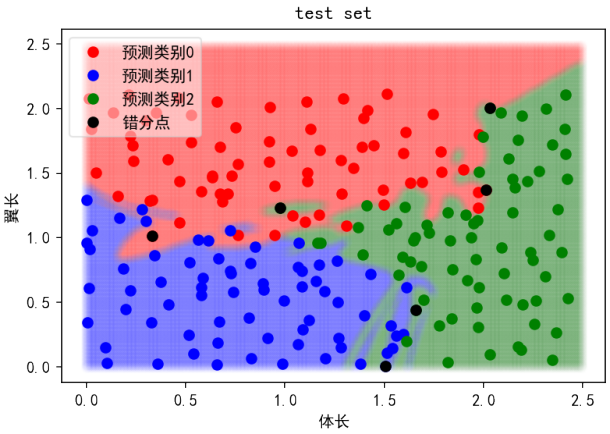


(d) 测试集以及神经网络决策边界

有噪声数据：



(e) 训练集以及神经网络决策边界



(f) 测试集以及神经网络决策边界

从上图可见，对于无噪声数据，神经网络能够完美的找出决策边界，在测试集上的准确率达到了 100%；而对于有噪声的数据，神经网络无法做到完全的识别噪声点，会去用更复杂的结构去拟合包括噪声点在内的数据。例如在图 e 中红色区域中的绿色区域，在肉眼上看是较为明显的噪声区域，但是神经网络并不能分辨这一点，仍将这一区域划分错误。

6.2 参数分析

我们将训练好的模型在验证集上进行测试，以 Cross Entropy Loss(交叉熵) 作为网络性能指标。然后，对网络深度、学习率、网络宽度、激活函数等模型超参数进行调整，再重新训练、测试，并分析对模型性能的影响。

下面是对于固定其他参数取值 (较为合适)，分别对于**激活函数**、**学习率**、**网络深度**、**网络宽度**的不同取值得到的训练集上的误差曲线：

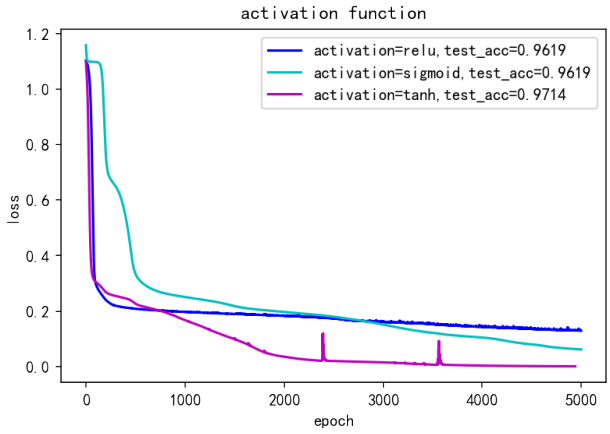


图 3: Activation Function

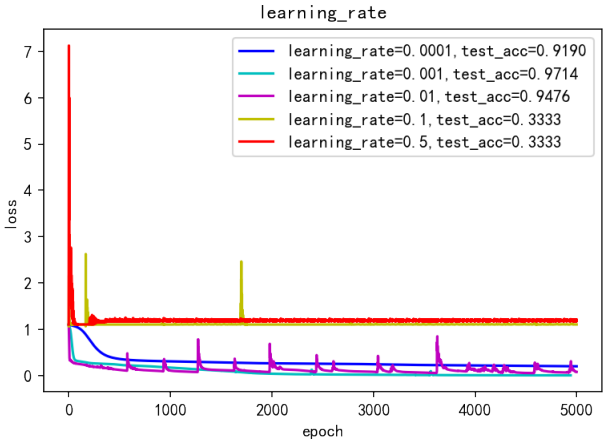


图 4: Learning Rate

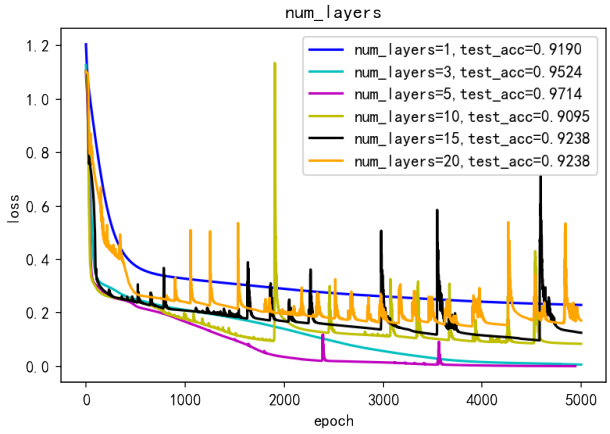


图 5: Num_layers

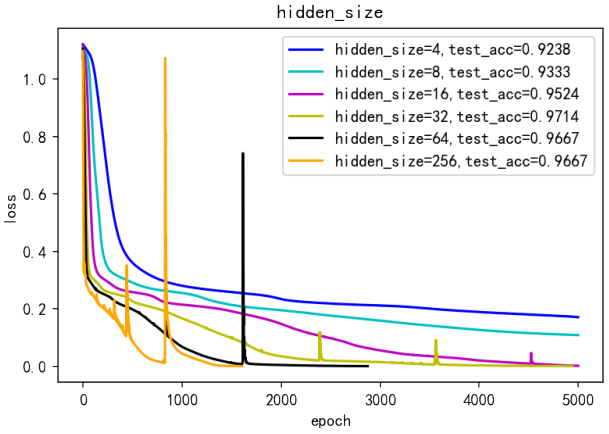


图 6: Hidden Size

由上图3可见，整体来看 tanh 激活函数在本数据集上效果最好。事实上，不同的激活函数对模型性能的影响不同，ReLU 通常是一个不错的选择，但是在某些情况下，其他激活函数可能更适合。但是如果网络层数加深的话，sigmoid 和 tanh 函数会导致梯度消失。采用 relu 函数有时可以避免此问题。

从上图4可看出，lr=0.5 与 0.3 时，损失值很大，且不再下降，迭代陷入局部最优。lr=0.0001 时，损失还未降到最低，收敛速度很慢。当**学习率过大**时，每次参数更新的步长会很大，可能会导致模型在训练过程中出现震荡或不稳定的情况，甚至可能会导致模型无法收敛。此外，学习率过大还可能会导致模型跳过全局最优解，而陷入局部最优解。当**学习率过小**时，每次参数更新的步长会很小，可能会导致模型在训练过程中收敛速度过慢，甚至可能会导致模型无法收敛。此外，学习率过小还可能会导致模型陷入局部最优解，而无法达到全局最优解。

由上图5可见，在有限范围内增大网络层数确实会带来性能的提升，但如果深度过大，则会出现过拟合的现象，导致测试集上的准确率很低。

同样的，由上图6可见，对于网络宽度在有限范围内增大网络层数确实会带来性能的提升，但如果宽度过大，则会出现过拟合的现象 (当 hidden size 达到 256 时，loss 的表现反而不如 32)，导致测试集上的准确率很低。

7 结论

本次实验通过对于昆虫体长和翼长的分析，使用神经网络模型对数据进行训练和测试，取得了不错的效果，在无噪声数据集上达到了 100% 的准确率，在有噪声数据集上达到了 97.14% 的准确率；并对于有关参数进行了调整和测试，得到了以下结论：

1. 激活函数：激活函数是神经网络中非常重要的组成部分，它的作用是将输入信号转换为输出信号。常见的激活函数有 sigmoid、ReLU、tanh 等。不同的激活函数对神经网络的性能有不同的影响。例如，sigmoid 函数在输入较大或较小时会出现梯度消失的问题，导致训练困难；而 ReLU 函数可以有效地避免梯度消失问题，加速训练过程。

2. 学习率：学习率是神经网络中控制权重更新速度的参数，它的大小会影响神经网络的收敛速度和性能。如果学习率过大，会导致权重更新过快，可能会错过最优解；如果学习率过小，会导致收敛速度过慢，训练时间过长。因此，需要根据具体问题选择合适的学习率。

3. 网络深度：网络深度是指神经网络中隐藏层的层数，它的大小会影响神经网络的复杂度和性能。一般来说，增加网络深度可以提高神经网络的表达能力，但也会增加训练时间和过拟合的风险。因此，需要根据具体问题选择合适的网络深度。

4. 网络宽度：网络宽度是指神经网络中每层的神经元个数，它的大小会影响神经网络的复杂度和性能。一般来说，增加网络宽度可以提高神经网络的表达能力，但也会增加训练时间和过拟合的风险。因此，需要根据具体问题选择合适的网络宽度。

8 问题

本次实验按照要求并没有进行数据的预处理。但是如果我们在进行训练之前对于数据进行**降噪**，会得到更好的结果。常用的降噪方法包括：均值滤波、中值滤波、小波变换、自适应滤波等。也可以先对数据进行**聚类**，找出离群点作为噪声点。或者可以通过观察数据，自定义一些规则选择噪声。下面给出一个不太成熟的规则：如果一个点周围的 k 个点内颜色与它不同的点的个数超过 αk，则该点是噪声点。这里面的 k, α 均是可调整的参数。

也可以加入正则化项，用于防止过拟合。

参考文献

[1] 周志华. 机器学习 [M]. 北京: 清华大学出版社, 2016.

附录 A 代码

```python
# 定义前馈神经网络模型
class FeedForwardNet(nn.Module):
    '''
    input_size: 输入层的维度
    hidden_size: 隐藏层的维度
    output_size: 输出层的维度
    num_layers: 隐藏层的层数
    activation: 激活函数
    '''
    def __init__(self, input_size, hidden_size, output_size, num_layers, activation):
        torch.manual_seed(0) # 设置随机种子
        super().__init__() # 调用父类的构造函数

        self.num_layers = num_layers

        # 定义输入层
        self.input_layer = nn.Linear(input_size, hidden_size)

        # 定义隐藏层
        self.hidden_layers = nn.ModuleList()
        for i in range(num_layers-1):
            self.hidden_layers.append(nn.Linear(hidden_size, hidden_size))

        # 定义输出层
        self.output_layer = nn.Linear(hidden_size, output_size)

        # 定义激活函数
        if activation == 'relu':
            self.activation = nn.ReLU()
        elif activation == 'sigmoid':
            self.activation = nn.Sigmoid()
        else:
            self.activation = nn.Tanh()

        # 定义softmax函数
        # self.sm=nn.Softmax(dim=1)

    # 前向传播
    def forward(self, x):
        out = self.input_layer(x)
        out = self.activation(out)

        for i in range(self.num_layers-1):
            out = self.hidden_layers[i](out)
            out = self.activation(out)

        out = self.output_layer(out)

        return out

# 定义训练函数和测试函数
def fit(model, optimizer, criterion, x, y, epochs, tol=1e-2):
    losses = []
    for epoch in tqdm(range(epochs)):
        optimizer.zero_grad()
```

```
56          y_pred = model(x)
57          loss = criterion(y_pred, y)
58          loss.backward()
59          optimizer.step()
60          losses.append(loss.item())
61          # if (epoch + 1) % 200 == 0:
62          #     print('epoch: {}, train_loss: {:.6f}'.format(epoch+1, loss.item()))
63          if epoch >= 2:
64              if losses[-1] < tol:
65                  print('epoch: {}, train_loss: {:.6f}'.format(epoch+1, loss.item()))
66                  return losses
67      print('epoch: {}, train_loss: {:.6f}'.format(epochs, loss.item()))
68      return losses
69
70  def predict(model, x):
71      with torch.no_grad():
72          y_pred = model(x)
73      return y_pred
```