華中科技大學

计 算 机 视 觉 课 程 报 告

题目： 基于前馈神经网络的分类任务设计

学　　号　1111

姓　　名　thezmmm

专　　业　数据科学与大数据技术

班　　级　000

指 导 教 师　杨卫

计算机科学与技术学院

目　录

一　实验要求

1. 实验目标:
   设计一个前馈神经网络,将给定的二维高斯数据集进行分类。
2. 数据集描述:
   下载并导入名为”dataset.csv” 的数据集。该数据集包含四类二维高斯数据和它们的标签。数据集中的每个样本由两个特征值 (〚data〛_1, 〚data〛_2) 和对应的分类标签组成。
3. 数据预处理:
   随机排序: 首先对数据集进行随机排序, 以确保训练集和测试集的样本分布均匀。划分训练集和测试集: 将排序后的数据集划分为训练集和测试集。其中 90% 的数据将用于训练,剩下的 10% 用于测试。
4. 神经网络设计:
   前馈神经网络: 设计一个前馈神经网络, 用于分类任务。隐藏层: 至少包含一层隐藏层, 可以根据需要选择隐藏层的数量和神经元的数量。输入层: 输入层的节点数量应与数据集中的特征数量相匹配, 即两个。输出层: 输出层的节点数量应与分类任务的类别数量相匹配,即四个。
5. 训练和测试:
   使用划分好的训练集对神经网络进行训练。使用划分好的测试集对神经网络进行测试和评估。可以选择适当的训练算法和评估指标,如反向传播算法和准确率。
6. 实验结果:
   给出实验结果的详细记录,包括训练过程中的损失值变化和测试集上的准确率等。可以使用图表、表格或其他形式展示实验结果。

二　实验内容

2.1　数据集划分

数据集的准备过程如下:

1. 加载数据集:
   使用 ‘pd.read_csv()‘函数加载名为”dataset.csv” 的数据集文件, 并将其存储在一个名为 ‘data‘的变量中。
2. 随机排序数据集:
   使用 ‘sample()‘函数和 ‘frac‘参数设置为 1, 对数据集进行随机排序。‘frac=1‘表示保留全部数据, 实现了对整个数据集的洗牌操作。通过 ‘reset_index(drop=True)‘将重置索引, 确保数据集的索引是连续的。
3. 提取特征和标签:
   从排序后的数据集中提取特征和标签。将 ‘data‘中名为’data1’ 和’data2’ 的列提取出来, 使用 ‘values‘属性将其转换为 NumPy 数组, 并将其存储在一个名为 ‘features‘的变量中。同时, 将 ‘data‘中名为’label’ 的列提取出来, 使用 ‘values‘属性将其转换为 NumPy 数组, 并将其存储在一个名为 ‘labels‘的变量中。这样, ‘features‘变量将包含二维高斯数据的特征, 而 ‘labels‘变量将包含对应的分类标签。
4. 划分训练集和测试集:
   使用 ‘train_test_split()‘函数将数据集划分为训练集和测试集。将 ‘features‘和 ‘labels‘作为输入参数, 通过 ‘test_size‘参数设置测试集的比例, 这里设置为 0.1 表示测试集占总样本数的 10%。通过 ‘random_state‘参数设置随机种子, 以确保划分结果的可重复性。将划分后的训练集特征、测试集特征、训练集标签和测试集标签分别存储在 ‘train_features‘、‘test_features‘、‘train_labels‘和 ‘test_labels‘变量中。

```
# 加载数据集
data = pd.read_csv('dataset.csv')

# 随机排序数据集
data = data.sample(frac=1).reset_index(drop=True)

# 提取特征和标签
features = data[['data1', 'data2']].values
labels = data['label'].values - 1

# 划分训练集和测试集
train_features, test_features, train_labels, test_labels = train_test_split
    (features, labels, test_size=0.1, random_state=42)
```

2.2 神经网络模型

```
# 定义神经网络的架构
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation=activation_function, input_shape
        =(2,)),
    tf.keras.layers.Dense(32, activation=activation_function),
    tf.keras.layers.Dense(4, activation='softmax')
])
```

这段代码定义了神经网络的架构,它是一个序列模型(Sequential Model)。

1. 第一层是一个全连接层(Dense Layer),它有 64 个神经元。这一层使用了一个激活函数(activation_function),实验中使用了 ReLU 和 sigmoid 两种激活函数。激活函数的作用是引入非线性性质,以便网络可以学习复杂的模式和关系。这一层的输入形状为 (2,),表示输入数据的维度是 2。
2. 第二层也是一个全连接层,有 32 个神经元,并且也使用了相同的激活函数。这一层没有指定输入形状,因为它会自动适应前一层的输出形状作为输入。
3. 最后一层是一个全连接层,有 4 个神经元,并使用了 softmax 激活函数。softmax 函数将神经元的输出转化为概率分布,用于多分类问题。这一层的输出表示模型对输入数据属于每个类别的概率分布。

这个神经网络的输入层有 2 个特征,输出层有 4 个类别。中间的两个全连接层为网络提供了学习和表示数据的能力。整个网络的结构是一个前馈神经网络,每一层的输出作为下一层的输入,信息从前向后传播。实验中根据具体的需求,调整了网络的层数、每层的神经元数量以及激活函数等参数,从而获得不同的实验结果以供分析。

2.3 网络参数

2.3.1 网络层数

指定神经网络的层数,即网络中包含的全连接层的数量。在给定的代码中,神经网络有 3 层,包括输入层、一个隐藏层和输出层。实验过程中尝试了两层神经网络。

2.3.2 神经元数量

指定每个全连接层中的神经元数量之和。其中,每一层的神经元数量都为 2^n。

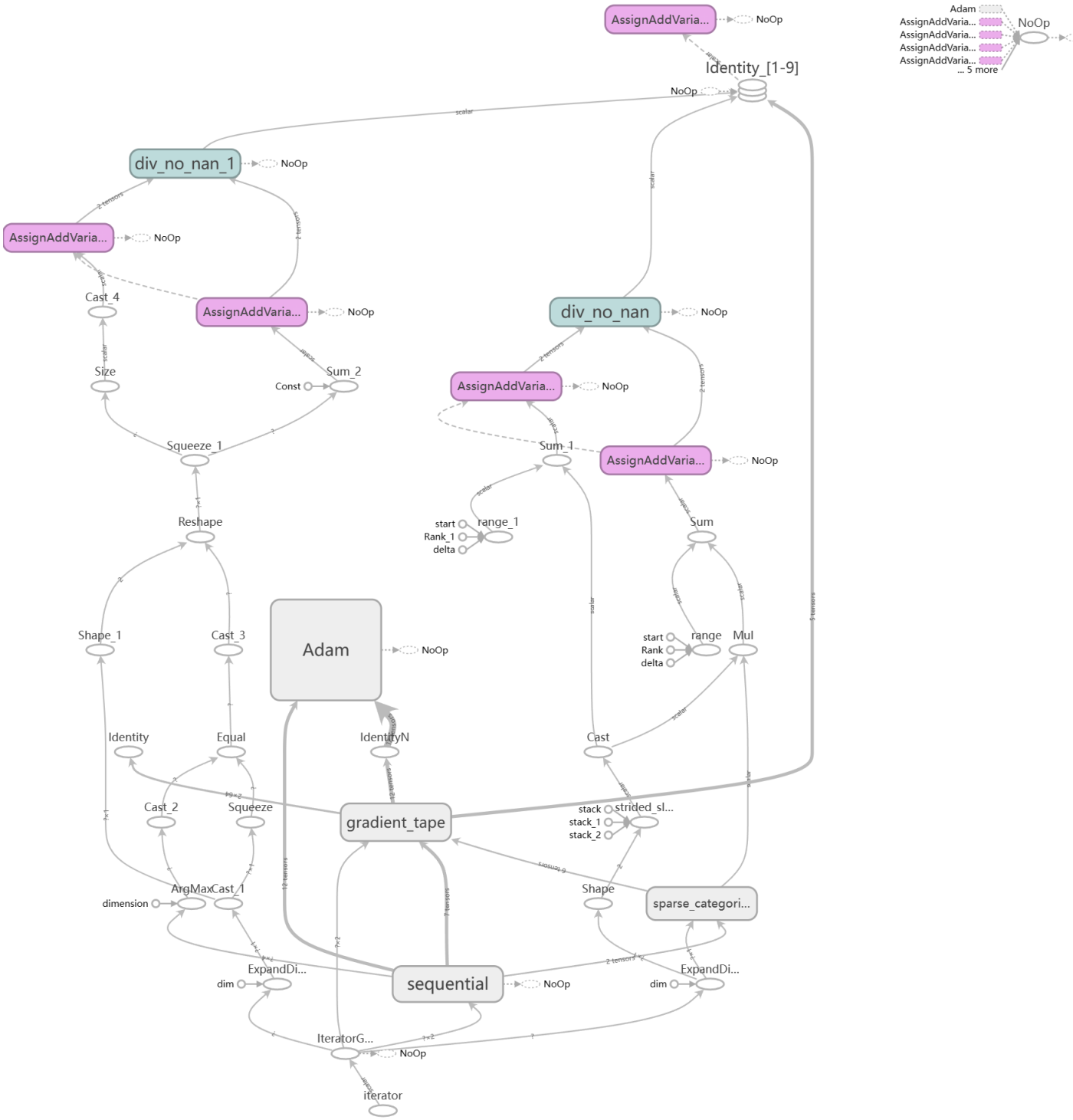

AssignAddVaria...
NoOp
Identity_[1-9]
div_no_nan_1
AssignAddVaria...
Cast_4
Size
Sum_2
Const
Squeeze_1
Reshape
Shape_1
Cast_3
Identity
Equal
Cast_2
Squeeze
ArgMaxCast_1
dimension
ExpandDi...
dim
IteratorG...
iterator
Adam
IdentityN
gradient_tape
sequential
div_no_nan
Sum_1
range_1
start
Rank_1
delta
Sum
range
Rank
Mul
Cast
stack
stack_1
stack_2
strided_sl...
Shape
sparse_categori...
Adam
AssignAddVaria...
AssignAddVaria...
AssignAddVaria...
AssignAddVaria...
... 5 more
NoOp

图 2-1　神经网络架构图

2.3.3 激活函数

激活函数是应用于每个神经元的非线性函数。在实验过程中,使用了三种激活函数:ReLU(修正线性单元),Sigmoid 和 tanh。ReLU 对于隐藏层常用,它将负输入值设置为零,并保持正输入值不变。Sigmoid 函数则将输入值映射到 0 到 1 之间的连续值,常用于二分类问题或输出层的概率估计。tanh 函数的图像与 sigmoid 十分相似,它的输出取值是(-1, 1),也具有同样的缺点,梯度消失,计算复杂。

2.3.4 优化器

优化器决定了神经网络在训练过程中如何更新权重以最小化损失函数。在实验过程中,使用了 Adam 优化器,它是一种基于梯度的优化算法,结合了动量方法和自适应学习率。

2.3.5 损失函数

损失函数用于衡量模型预测与实际标签之间的差异。在实验过程中,使用了稀疏分类交叉熵损失函数(sparse_categorical_crossentropy),适用于多分类问题,并且标签是整数形式。

2.3.6 训练轮数

指定神经网络在训练数据上迭代的次数。在实验过程中,网络将在训练数据上进行 10 个轮次的迭代。

2.3.7 批量大小

指定每个训练批次中使用的样本数量。在实验过程中,每个批次包含 32 个样本。

三　实验结果

3.1　迭代次数的影响

根据提供的实验结果，可以分析迭代次数(Epochs)对网络性能的影响。

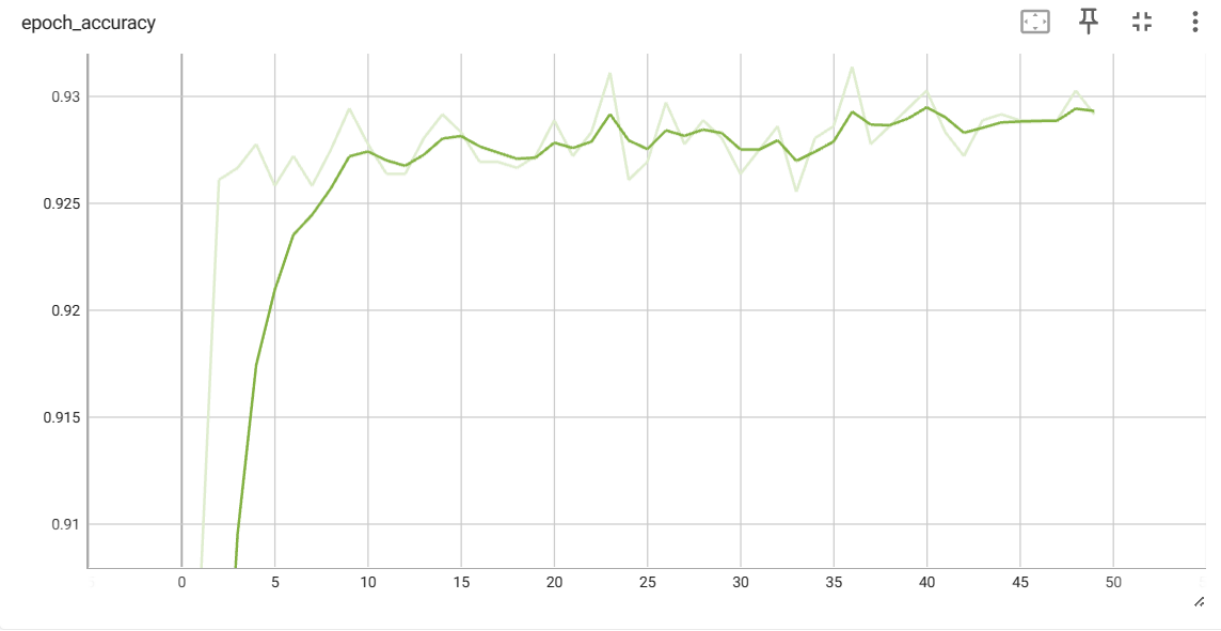


图 3-1　epoch_accuracy

1. 迭代次数对准确度的影响：
    - 随着迭代次数的增加，准确度逐渐提高，从初始的 0.788888872 逐渐增加到最高的 0.931388915。
    - 在迭代次数较低时，准确度的提升较为明显，但随着迭代次数的增加，提升幅度逐渐减小。
    - 在最后几次迭代中，准确度相对稳定，没有明显的进一步改善。
2. 迭代次数对损失函数的影响：
    - 随着迭代次数的增加，损失函数的值逐渐减小，从初始的 0.83936125 逐渐降低到最低的 0.202553749。
    - 在迭代次数较低时，损失函数的下降较为明显，但随着迭代次数的增加，下降速度逐渐变慢。

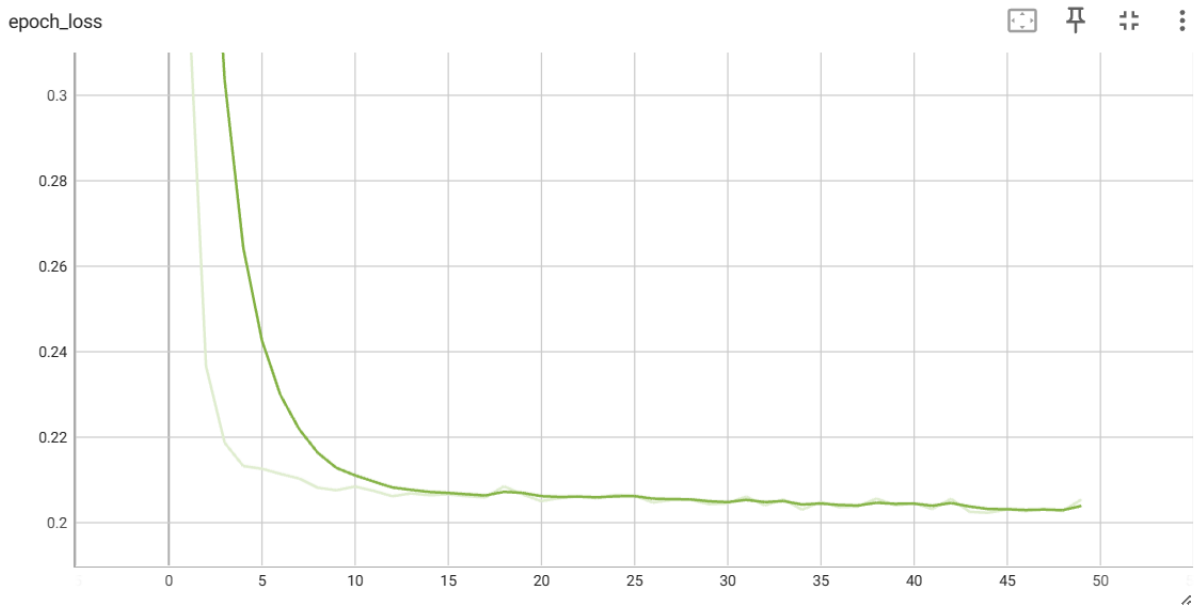


图 3-2　epoch_loss

- 在最后几次迭代中，损失函数的值相对稳定，没有明显的进一步改善。

根据实验结果，随着迭代次数的增加，模型的性能有所改善，准确度提高，损失函数降低。

在较低的迭代次数时，模型可能还没有充分学习数据集的特征和模式，因此准确度较低，损失函数较高。

随着迭代次数的增加，模型逐渐学习到更多的特征和模式，准确度提高，损失函数降低。

在一定程度上，模型的性能可能会随着迭代次数的增加而趋于稳定，准确度和损失函数的改善幅度减小。

3.2　激活函数的影响

根据实验结果，可以对三种不同的激活函数（ReLU、Sigmoid 和 Tanh）对网络性能的影响进行分析。下面是对每个激活函数的影响进行详细分析：

1. ReLU 激活函数：

   准确率：在 10 个训练轮次中，准确率从初始值 0.738 逐渐增加到 0.924，显示出较好的收敛性。然而，准确率在后续的轮次中没有明显的提升，可能暗示网络已经接近其性能上限。

   损失值：损失值从初始值 0.901 下降到 0.220，在每个轮次中都有显著的减少。这表明模型在训练过程中有效地减小了预测值与实际标签之间的差距。

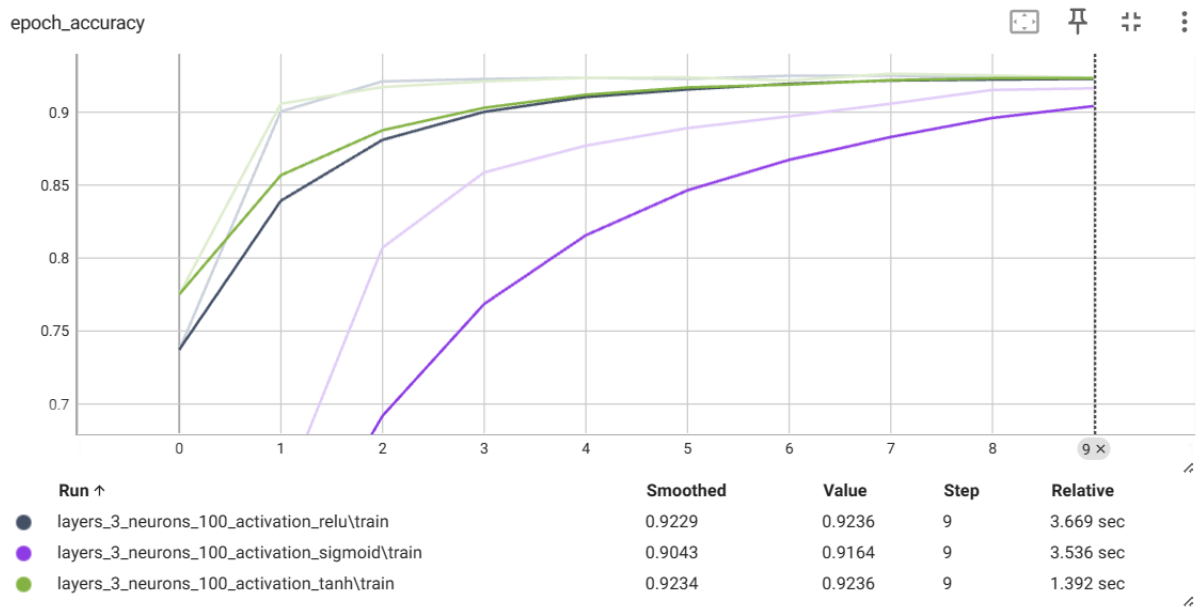


图 3-3　不同激活函数的准确度

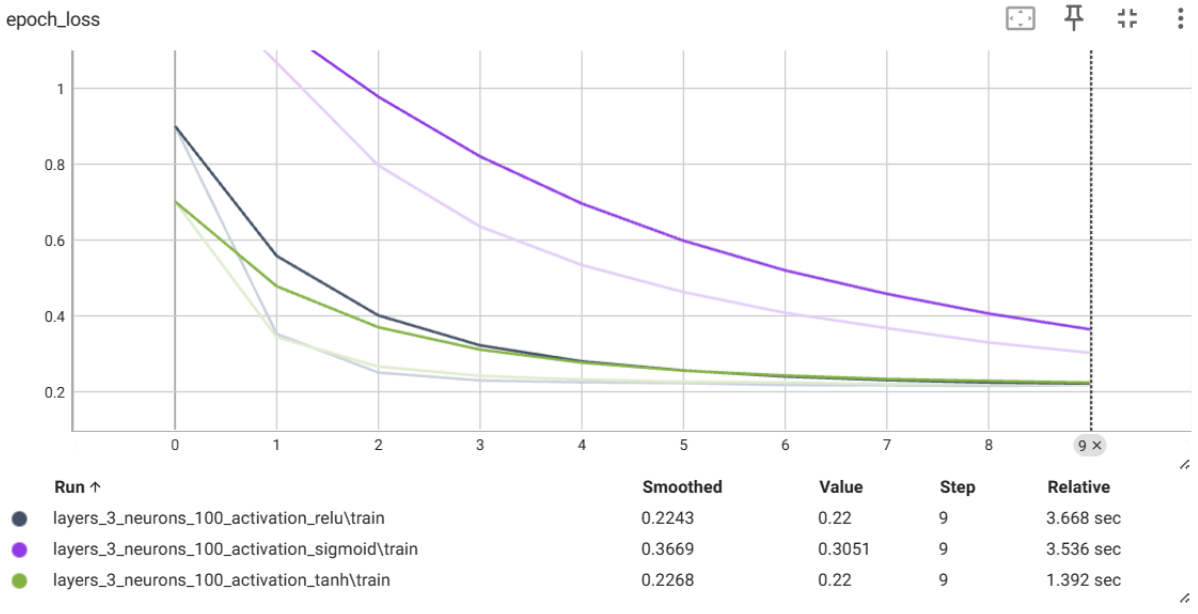


图 3-4　不同激活函数的损失值

2. Sigmoid 激活函数:

   准确率: 在 10 个训练轮次中, 准确率从初始值 0.469 逐渐增加到 0.916。相对于 ReLU 函数, Sigmoid 函数的准确率增长速度较慢, 并且在后续轮次中的增长也相对较小。

   损失值: 损失值从初始值 1.330 下降到 0.305, 在每个轮次中都有显著的减少。然而, 相对于 ReLU 函数, Sigmoid 函数的损失值要高得多。

3. Tanh 激活函数:

   准确率: 在 10 个训练轮次中, 准确率从初始值 0.776 逐渐增加到 0.924。Tanh 函数的准确率增长速度与 ReLU 函数相当, 但整体性能略低于 ReLU 函数。

   损失值: 损失值从初始值 0.703 下降到 0.220, 在每个轮次中都有显著的减少。Tanh 函数的损失值与 ReLU 函数相当。

在这个特定的实验设置中, ReLU 激活函数表现出最佳的性能, 具有较高的准确率和较低的损失值。这可能是因为 ReLU 函数能够更好地处理梯度消失问题, 并在训练过程中提供更好的非线性拟合能力。

Sigmoid 激活函数在准确率和损失值方面的表现相对较差, 可能是因为其饱和性导致了梯度消失问题, 并且在深层网络中较难优化。

Tanh 激活函数的性能介于 ReLU 和 Sigmoid 之间, 虽然能够处理梯度消失问题, 但相对于 ReLU 函数, 它的性能略低。

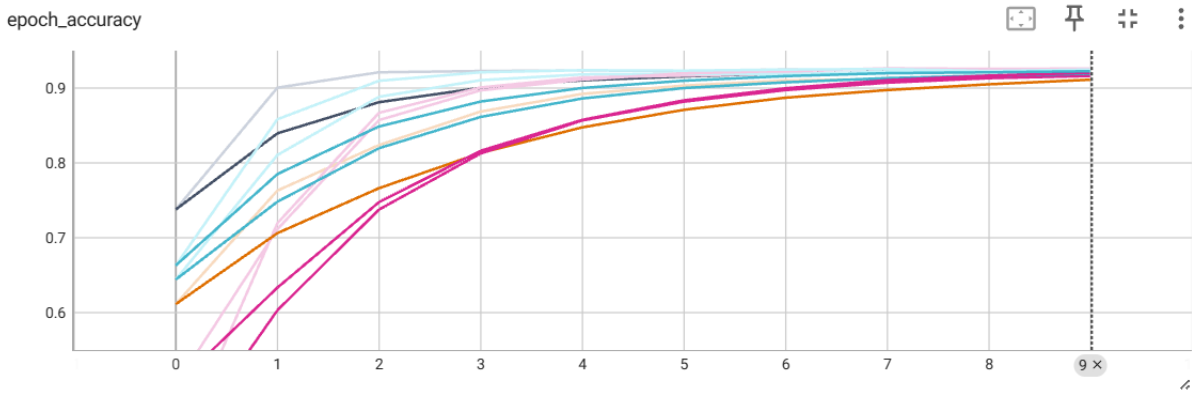
总的来说, 激活函数对神经网络的性能有明显的影响。在实验结果中, ReLU 激活函数在这个特定的任务上表现最好, 但对于不同的数据集和任务, 不同的激活函数可能会有不同的效果。因此, 在选择激活函数时, 需要根据具体情况和实验结果进行评估和选择。

3.3 网络层数及神经元数量的影响

3.3.1 神经元数量的影响

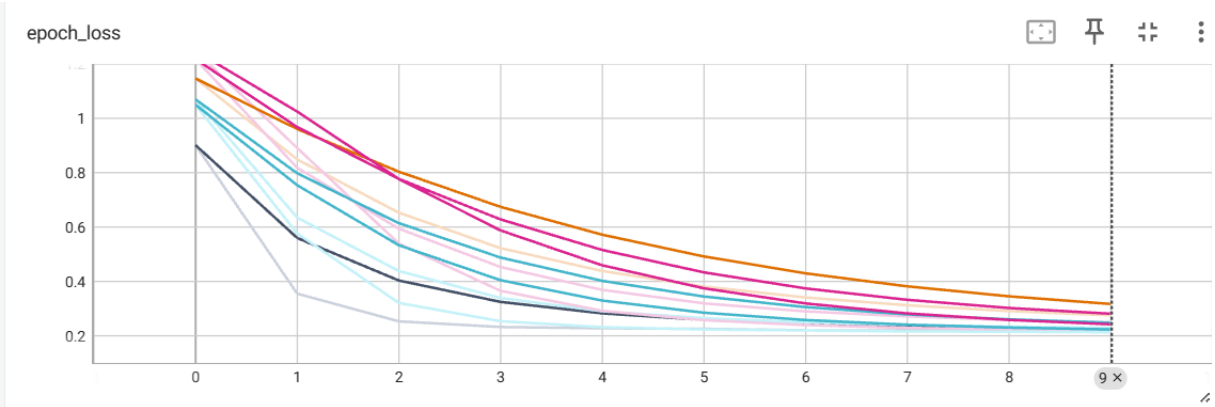
在 2 层网络中, 增加神经元数量从 20 到 36 时, 准确性 (accuracy) 有所提高, 从 0.920277774 增加到 0.920555532。然而, 当神经元数量增加到 68 时, 准确性提高到 0.92305553, 比 36 神经元更高。这表明增加神经元数量可以改善网络性能, 但超过一定点后, 进一步增加神经元数量可能带来较小的性能改善。

在 3 层网络中, 增加神经元数量从 28 到 52 时, 准确性从 0.926111102 提高到 0.924722195。然而, 当神经元数量增加到 100 时, 准确性下降到 0.219961464, 比 52 神经元更低。这表明增加神经元数量可以提高性能, 但超过一定点后, 可能会导致过拟合和性能下降。



Run	Smoothed	Value	Step	Relative
layers_2_neurons_20_activation_relu\train	0.9112	0.9203	9	2.978 sec
layers_2_neurons_36_activation_relu\train	0.9162	0.9206	9	2.878 sec
layers_2_neurons_68_activation_relu\train	0.9193	0.9231	9	2.523 sec
layers_3_neurons_100_activation_relu\train	0.9229	0.9236	9	3.669 sec
layers_3_neurons_28_activation_relu\train	0.9204	0.9261	9	3.652 sec
layers_3_neurons_52_activation_relu\train	0.9225	0.9247	9	3.27 sec

图 3-5　不同网络层数和神经元数量的准确度



Run	Smoothed	Value	Step	Relative
layers_2_neurons_20_activation_relu\train	0.3171	0.2752	9	2.978 sec
layers_2_neurons_36_activation_relu\train	0.2812	0.2498	9	2.878 sec
layers_2_neurons_68_activation_relu\train	0.2479	0.229	9	2.523 sec
layers_3_neurons_100_activation_relu\train	0.2243	0.22	9	3.668 sec
layers_3_neurons_28_activation_relu\train	0.2425	0.2187	9	3.651 sec
layers_3_neurons_52_activation_relu\train	0.2234	0.2131	9	3.27 sec

图 3-6　不同网络层数和神经元数量的损失值

3.3.2 网络层数的影响

在 2 层网络中，网络的准确性逐渐提高，从 0.920277774 到 0.92305553，随着神经元数量的增加。这说明在一定范围内，增加网络层数可以提高性能。

在 3 层网络中，准确性从 0.926111102 到 0.924722195，随着神经元数量的增加而略微下降。这表明增加网络层数不一定会带来更好的性能，而是取决于其他因素，如神经元数量和数据集的复杂性。

3.3.3 总结

综合考虑神经元数量和网络层数的影响，我们可以得出以下结论：

1. 增加神经元数量可以提高网络性能，但超过一定点后，进一步增加可能带来较小的改进或导致性能下降。
2. 增加网络层数可以在一定范围内提高性能，但过多的层数不一定会带来更好的结果。
3. 在确定最佳神经元数量和网络层数时，需要综合考虑模型的准确性、过拟合风险和计算资源的限制。

四　总结

本次实验设计了基于前馈神经网络的四分类神经网络模型，分析了神经网络性能受到的多个因素的影响，包括迭代次数、激活函数、网络层数、神经元数量。以下是对这些因素的概括性总结：

1. 迭代次数：增加迭代次数可以改善神经网络性能，提高准确度并降低损失函数。然而，性能的改善幅度会逐渐减小，需要在时间和性能改善之间进行权衡。
2. 激活函数：激活函数的选择对网络性能至关重要。常见的激活函数有 ReLU、Sigmoid、Tanh 和 Leaky ReLU 等，具体选择要根据任务和数据特性进行调整。
3. 网络层数：增加网络层数可以增强网络的表达能力，但同时也增加了过拟合的风险。适当的网络层数取决于任务的复杂性和可用的训练数据量。
4. 神经元数量：适当的神经元数量可以提高网络的表达能力，但过多的神经元可能导致过拟合。选择合适的神经元数量需要在模型复杂性和计算资源之间进行平衡。

综上所述，选择适当的迭代次数、激活函数、网络层数、神经元数量和模型架构是优化神经网络性能的关键因素，需要综合考虑任务要求和数据特性来进行调整和优化。