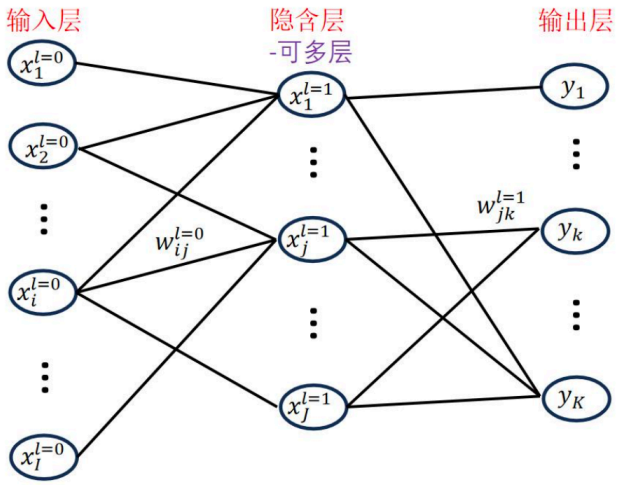
人工智能基础算法 第三次作业

王怡丰 2024310877

1

基于如图所示的神经网络



对于隐含层节点的输出

$$x_j^{l=1} = s(\sum_{i=1}^{I} x_i^{l=0} w_{ij}^{l=0})$$

进一步，对于输出层的结果

$$y_k = s(\sum_{j=1}^{J} x_j^{l=1} w_{jk}^{l=1})$$

其中

$$s(\alpha) = \frac{1}{1+e^{-\alpha}}$$

该神经网络的损失函数为

$$f(w) = \frac{1}{2}\sum_{k=1}^{K}(y_k - d_k)^2$$

反向传播算法即让该损失函数的值最小

$$\min_{w} f(w) = \frac{1}{2}\sum_{k=1}^{K}(y_k - d_k)^2$$

现在推导反向传播算法的更新公式。假设已知t时刻的输出值y_k和参考值$w_{ij}^{l=0}$，$w_{jk}^{l=1}$，以下给出更新$t+1$时刻，参数$w_{ij}^{l=0}(t+1)$，$w_{jk}^{l=1}(t+1)$的值的计算过程。利用梯度下降方法，首先对于$w_{jk}^{l=1}$，损失函数关于其的导数为

$$\frac{\partial f}{\partial w_{jk}^{l=1}}=\frac{\partial}{\partial w_{jk}^{l=1}}\frac{1}{2}\sum_{k=1}^{K}(y_k-d_k)^2$$

$$=\frac{\partial y_k}{\partial w_{jk}^{l=1}}\cdot\frac{\partial}{\partial y_k}\frac{1}{2}\sum_{k=1}^{K}(y_k-d_k)^2$$

$$=(y_k-d_k)\frac{\partial}{\partial w_{jk}^{l=1}}s(\sum_{j=1}^{J}x_j^{l=1}w_{jk}^{l=1})$$

$$=(y_k-d_k)\frac{\partial\sum_{j=1}^{J}x_j^{l=1}w_{jk}^{l=1}}{\partial w_{jk}^{l=1}}\frac{\partial s(\sum_{j=1}^{J}x_j^{l=1}w_{jk}^{l=1})}{\partial\sum_{j=1}^{J}x_j^{l=1}w_{jk}^{l=1}}$$

$$=(y_k-d_k)x_j^{l=1}s(\sum_{j=1}^{J}x_j^{l=1}w_{jk}^{l=1})(1-s(\sum_{j=1}^{J}x_j^{l=1}w_{jk}^{l=1}))$$

$$=(y_k-d_k)y_k(1-y_k)x_j^{l=1}$$

令$\delta_k^{l=2}=(y_k-d_k)y_k(1-y_k)$，则有

$$\frac{\partial f}{\partial w_{jk}^{l=1}}=x_j^{l=1}\delta_k^{l=2}$$

假设此时的步长为λ，则有

$$w_{jk}^{l=1}(t+1)=w_{jk}^{l=1}(t)-\lambda\frac{\partial f}{\partial w_{jk}^{l=1}}=w_{jk}^{l=1}(t)-\lambda x_j^{l=1}\delta_k^{l=2}$$

而对于$w_{jk}^{l=0}$，损失函数关于其的导数为

$$\frac{\partial f}{\partial w_{ij}^{l=0}}=\frac{\partial}{\partial w_{ij}^{l=0}}\frac{1}{2}\sum_{k=1}^{K}(y_k-d_k)^2$$

$$=\sum_{k=1}^{K}\frac{\partial y_k}{\partial w_{ij}^{l=0}}\cdot\frac{\partial}{\partial y_k}\frac{1}{2}(y_k-d_k)^2$$

$$=\sum_{k=1}^{K}(y_k-d_k)\frac{\partial}{\partial w_{ij}^{l=0}}s(\sum_{j=1}^{J}x_j^{l=1}w_{jk}^{l=1})$$

$$=\sum_{k=1}^{K}(y_k-d_k)\frac{\partial\sum_{j=1}^{J}x_j^{l=1}w_{jk}^{l=1}}{\partial w_{ij}^{l=0}}\frac{\partial s(\sum_{j=1}^{J}x_j^{l=1}w_{jk}^{l=1})}{\partial\sum_{j=1}^{J}x_j^{l=1}w_{jk}^{l=1}}$$

$$=\sum_{k=1}^{K}\delta_k^{l=2}\frac{\partial x_j^{l=1}w_{jk}^{l=1}}{\partial w_{ij}^{l=0}}$$

$$=\sum_{k=1}^{K}\delta_k^{l=2}w_{jk}^{l=1}\frac{\partial s(\sum_{i=1}^{I}x_i^{l=0}w_{ij}^{l=0})}{\partial w_{ij}^{l=0}}$$

$$=\sum_{k=1}^{K}\delta_k^{l=2}w_{jk}^{l=1}x_i^{l=0}x_j^{l=1}(1-x_j^{l=1})$$

令$\delta_j^{l=1} = x_j^{l=1}(1 - x_j^{l=1}) \sum_{k=1}^{K} \delta_k^{l=2} w_{jk}^{l=1}$，则有

$$\frac{\partial f}{\partial w_{ij}^{l=0}} = x_i^{l=0} \delta_j^{l=1}$$

假设此时的步长为λ，则有

$$w_{ij}^{l=0}(t+1) = w_{ij}^{l=0}(t) - \lambda \frac{\partial f}{\partial w_{ij}^{l=0}} = w_{ij}^{l=0}(t) - \lambda x_i^{l=0} \delta_j^{l=1}$$

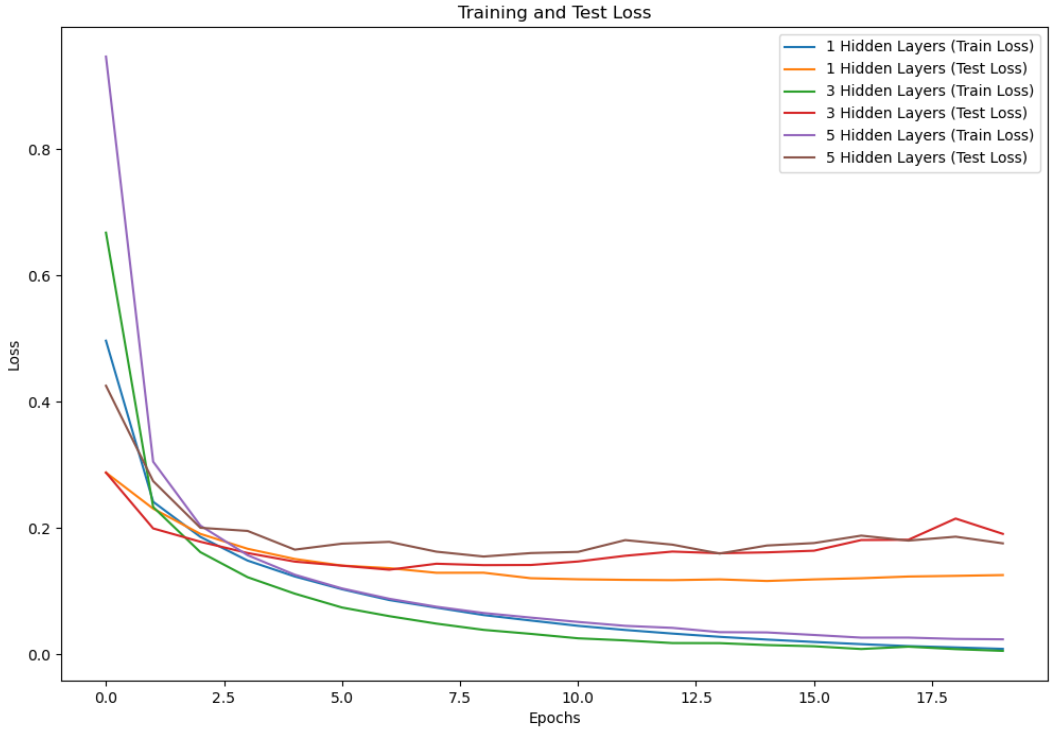
2

利用`tensorflow`库建立神经网络，使用`Adam`优化器训练神经网络，损失函数为`categorical_crossentropy`分类交叉熵损失，分别设置隐含层的层数为1，3，5层，以Sigmoid函数作为激活函数。训练过程中，每次计算同时处理20个数据（`batch_size`）。共（对训练集数据）训练20次，记录针对训练集、测试集的损失函数、正确率随训练次数的变化关系。

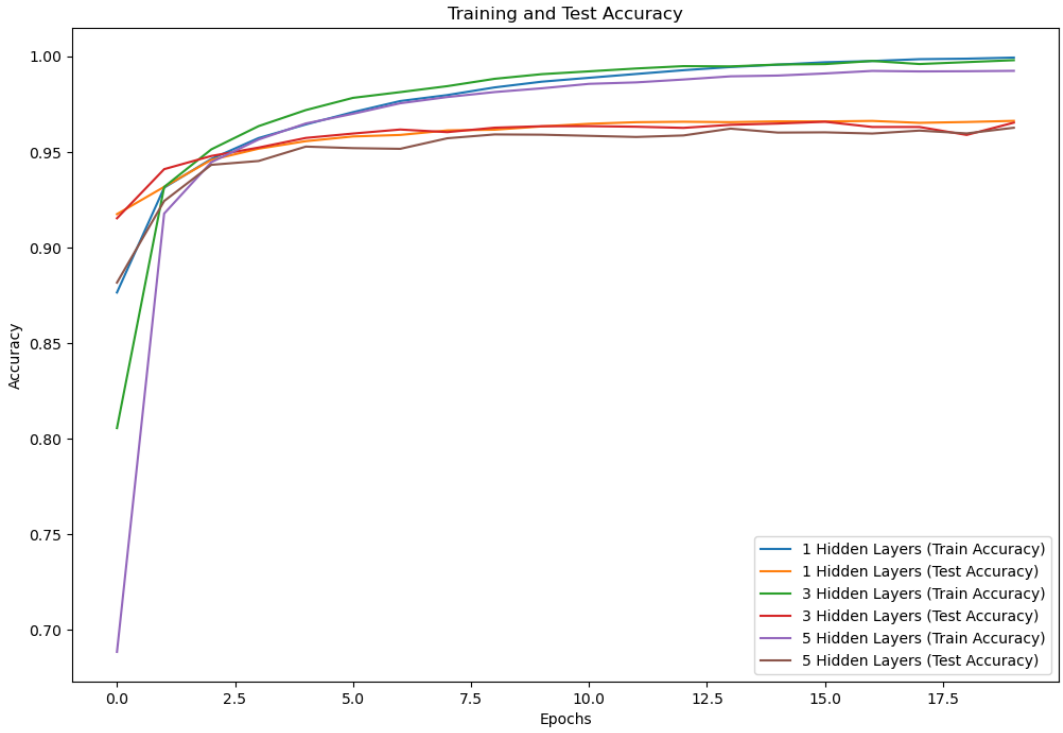
其中部分关键的代码如下，包括训练函数中对构建、编译模型并训练模型的部分

```
def train_model(hidden_layers):
    # 构建模型
    model = tensorflow.keras.models.Sequential()
    model.add(tensorflow.keras.layers.Dense(100, activation='sigmoid', input_shape=
(784,)))  # 输入层到第一个隐含层
    for _ in range(hidden_layers - 1):  # 添加剩余的隐含层
        model.add(tensorflow.keras.layers.Dense(100, activation='sigmoid'))
    model.add(tensorflow.keras.layers.Dense(10, activation='softmax'))  # 输出层

    # 编译模型
    model.compile(optimizer='adam',  # 使用随机梯度下降作为优化器
                  loss='categorical_crossentropy',  # 多分类交叉熵损失函数
                  metrics=['accuracy'])

    # 训练模型
    for epoch in range(20):  # 可以根据需要调整epoch数量
        history = model.fit(X_train, y_train,
                            epochs=1,  # 每次只训练一个epoch
                            batch_size=20,  # 批量大小
                            verbose=0)  # 不打印进度条
```

损失函数关于训练次数的关系为



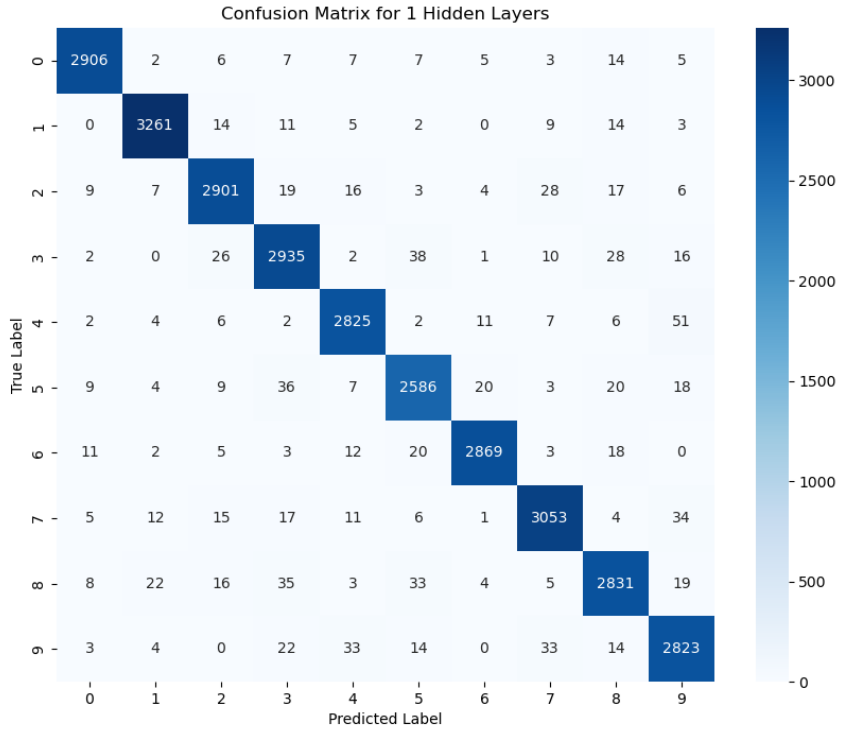
正确率关于训练次数的关系为

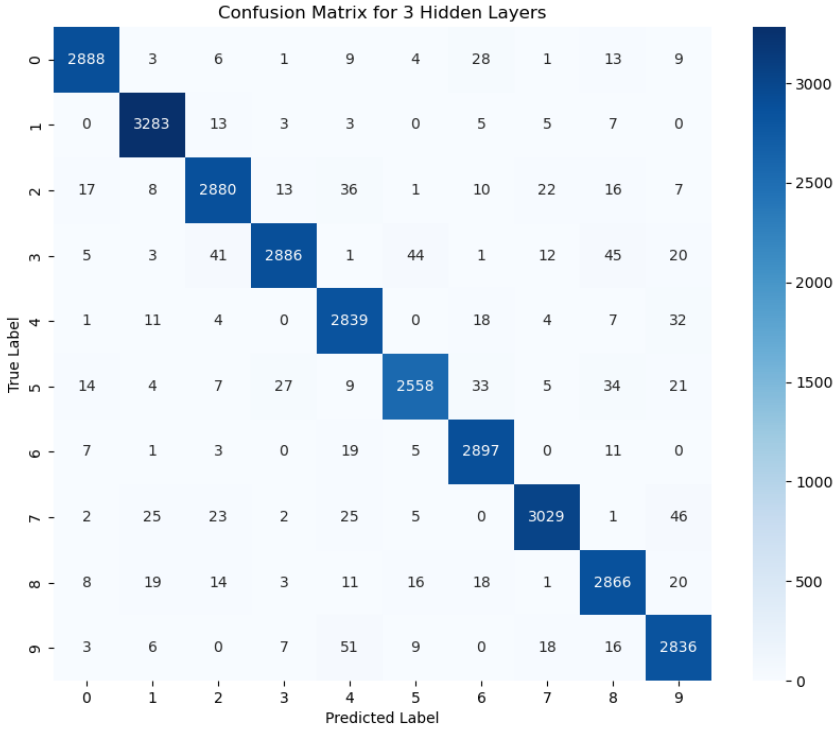


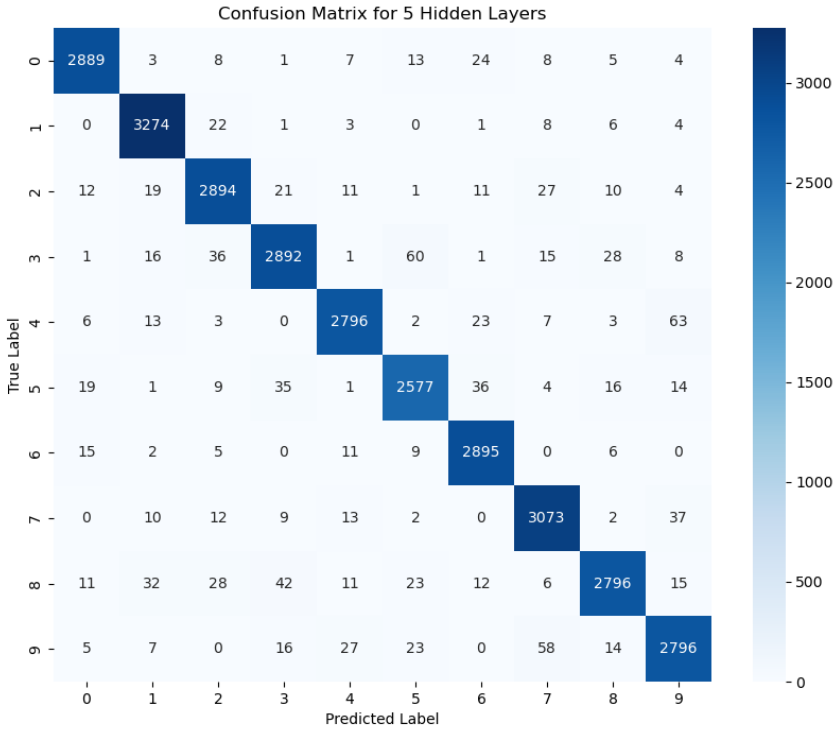
完成20次训练后，所花费的时间和正确率如下所示

层数	1	3	5
正确率	96.63%	96.54%	96.27%
时间/s	38.49	43.65	48.65

三种情况的混淆矩阵依次为





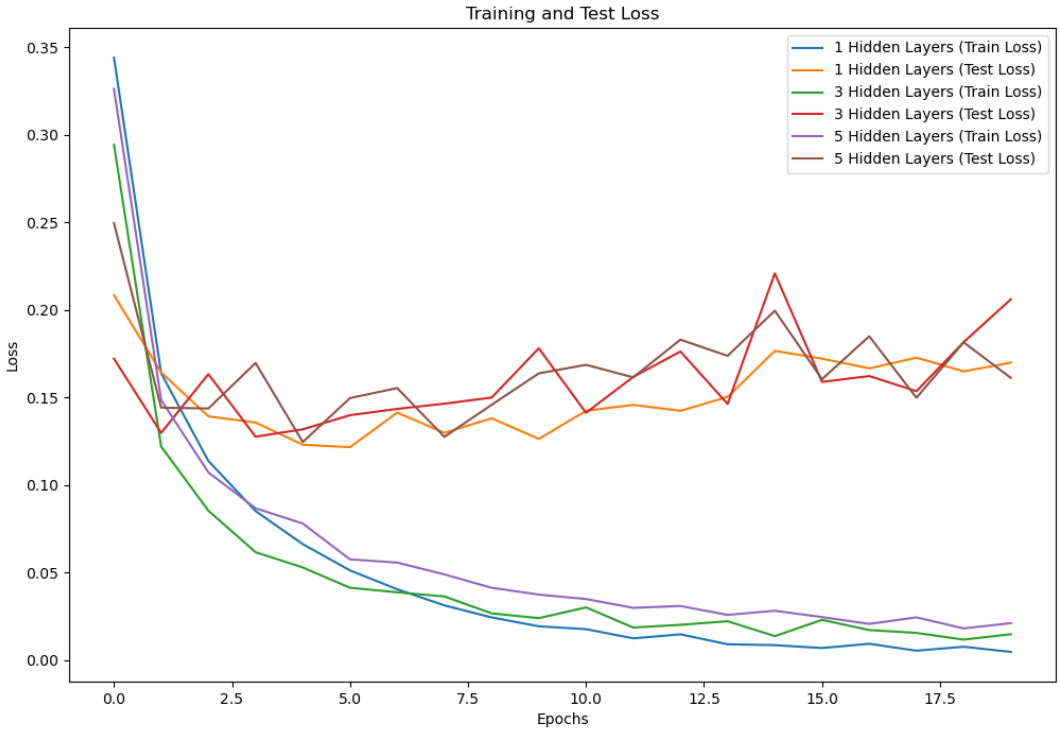


随着层数的增加，运算时间逐渐变长，正确率逐渐降低但变化不大；随着训练次数的增加，损失函数逐渐减小，正确率逐渐升高，但在达到一定值后逐渐停止增加、减小并开始一定的震荡。

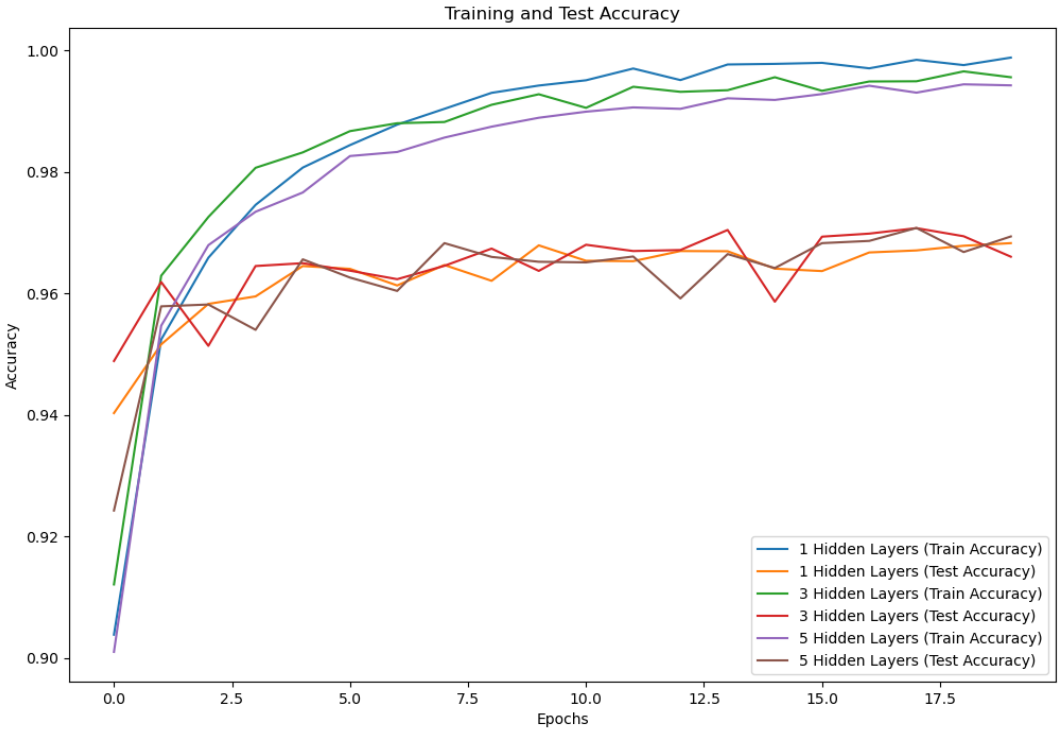
3

以ReLU函数作为激活函数，重复上述过程

损失函数关于训练次数的关系为



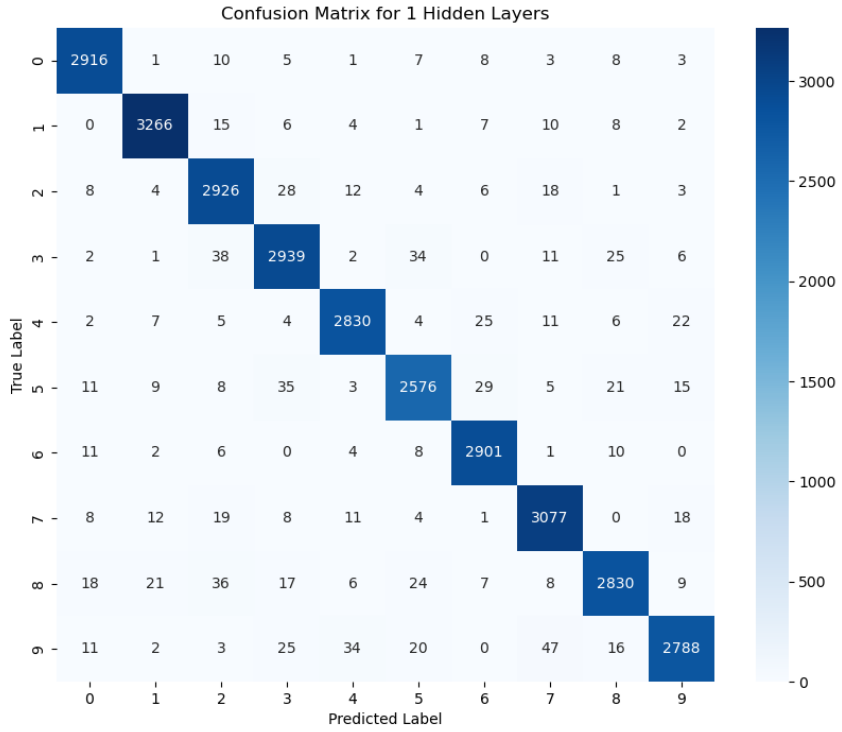
正确率关于训练次数的关系为

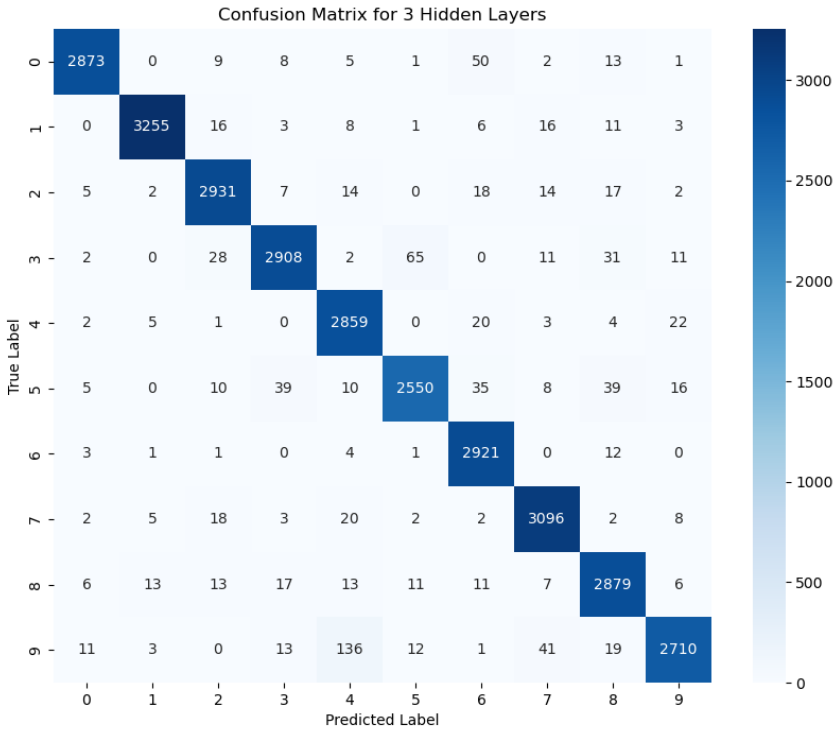


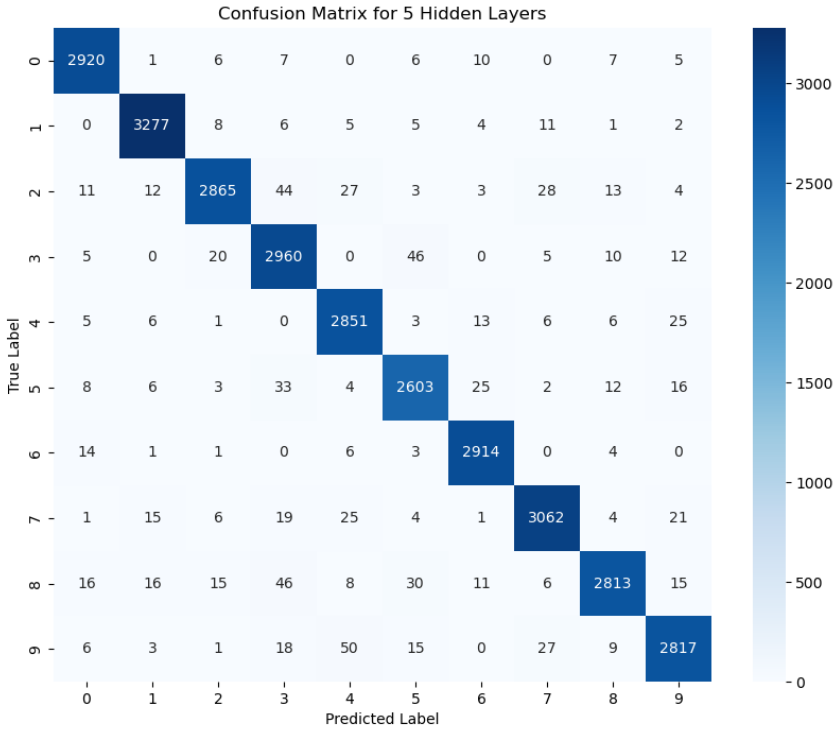
完成20次训练后，所花费的时间和正确率如下所示

层数	1	3	5
正确率	96.83%	96.61%	96.94%
时间/s	40.05	44.16	48.76

三种情况的混淆矩阵依次为





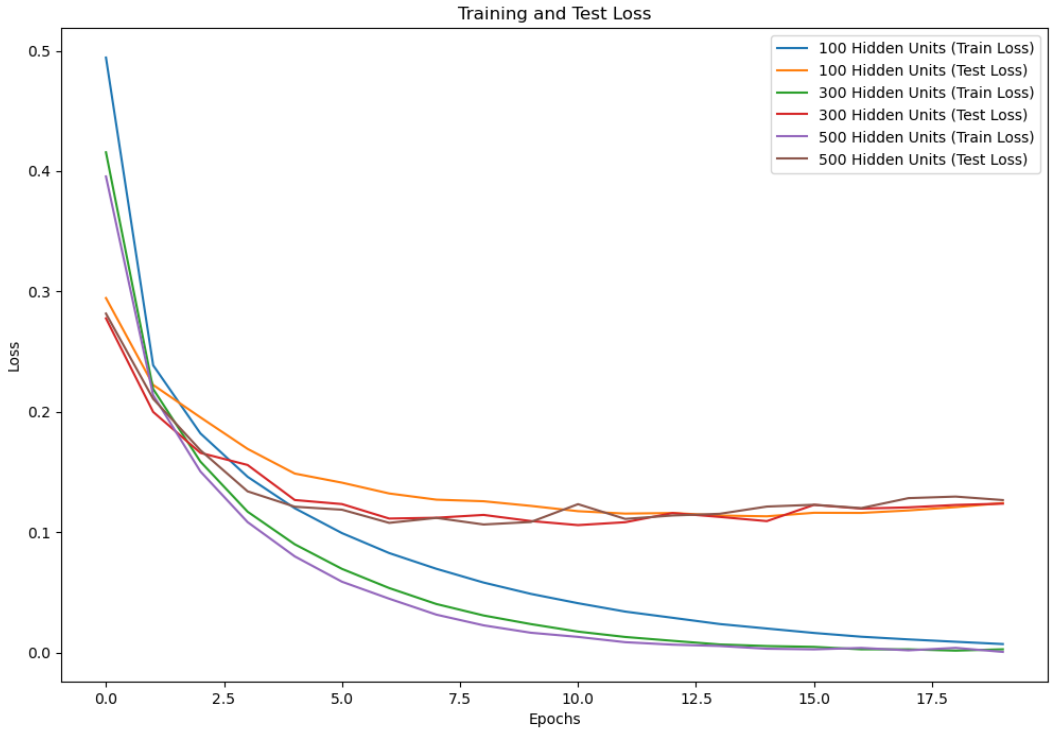


三种层数的正确率差别不大，时间和正确率对比Sigmoid函数均略有提升，但也比较接近。

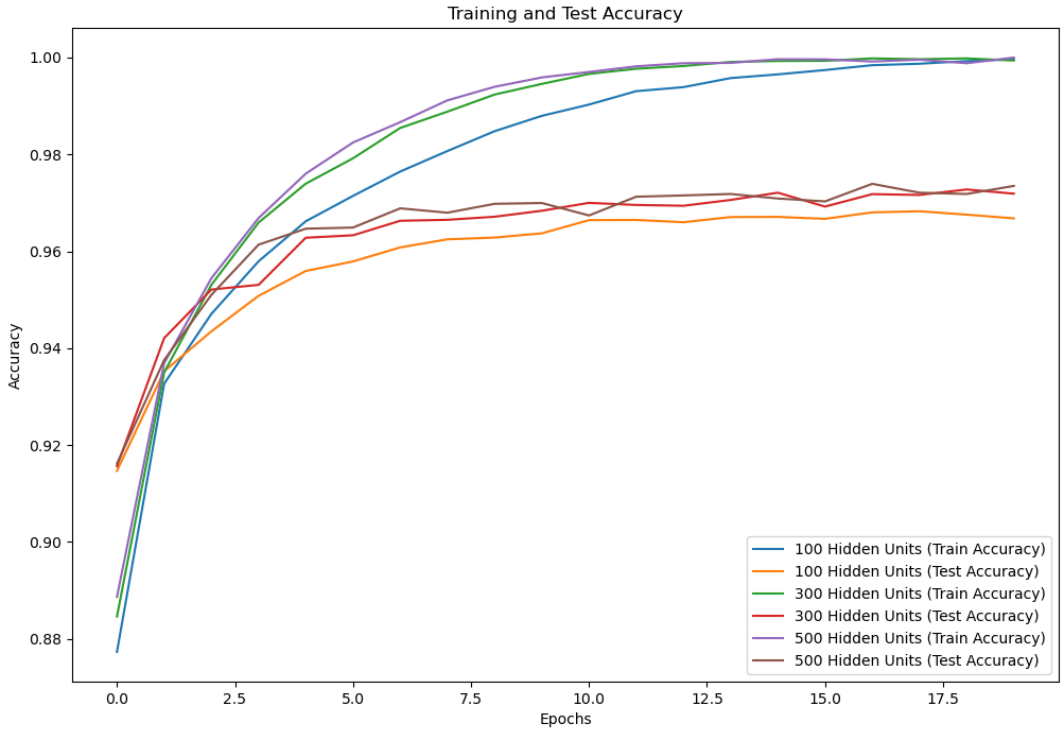
4

以Sigmoid作为激活函数，隐含层的层数为1层， 分别设置隐含层变量的个数为100、300、500，重复上述过程。

损失函数关于训练次数的关系为



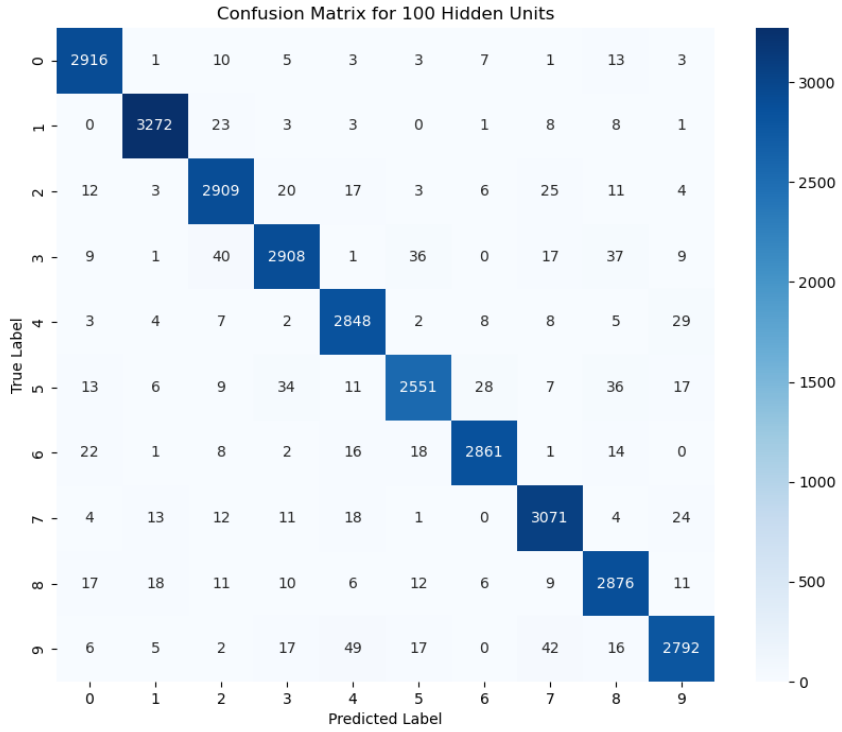
正确率关于训练次数的关系为

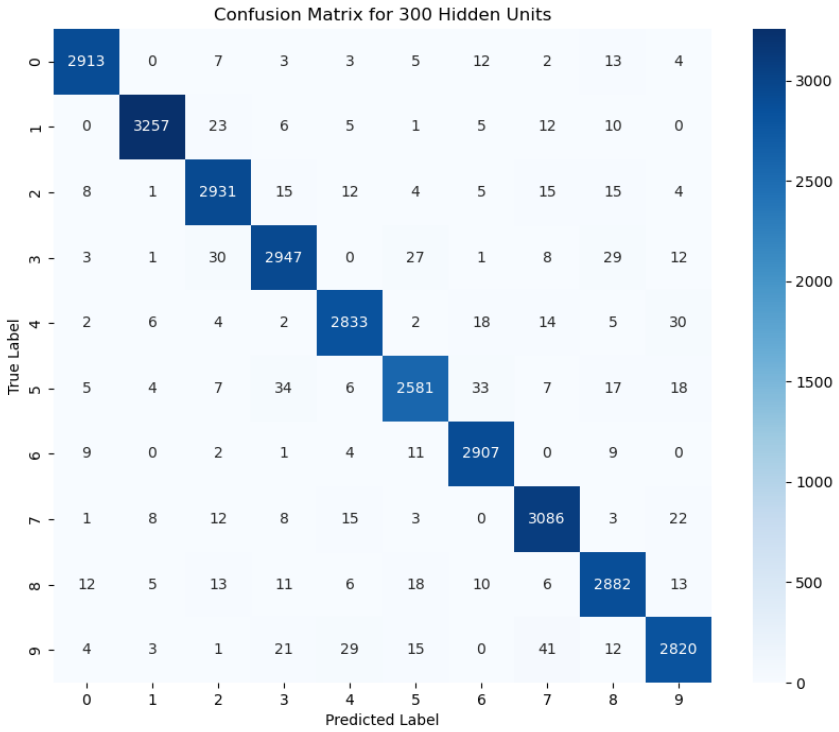


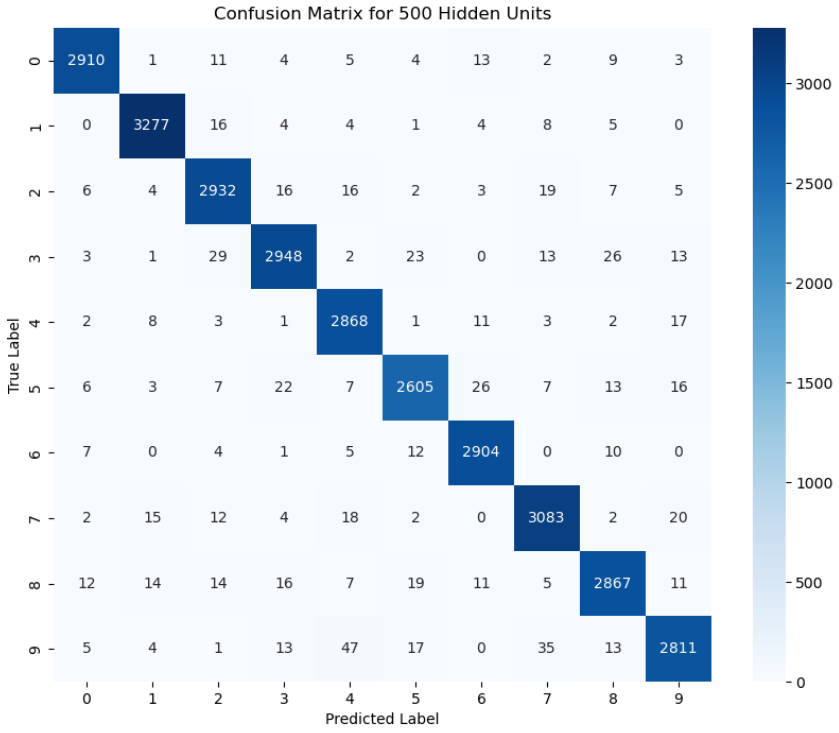
完成20次训练后，所花费的时间和正确率如下所示

变量个数	100	300	500
正确率	96.68%	97.19%	97.35%
时间/s	39.78	50.81	61.78

三种情况的混淆矩阵依次为







随着隐含层节点数量的增加，计算时间明显变长，正确率升高但变化不大。

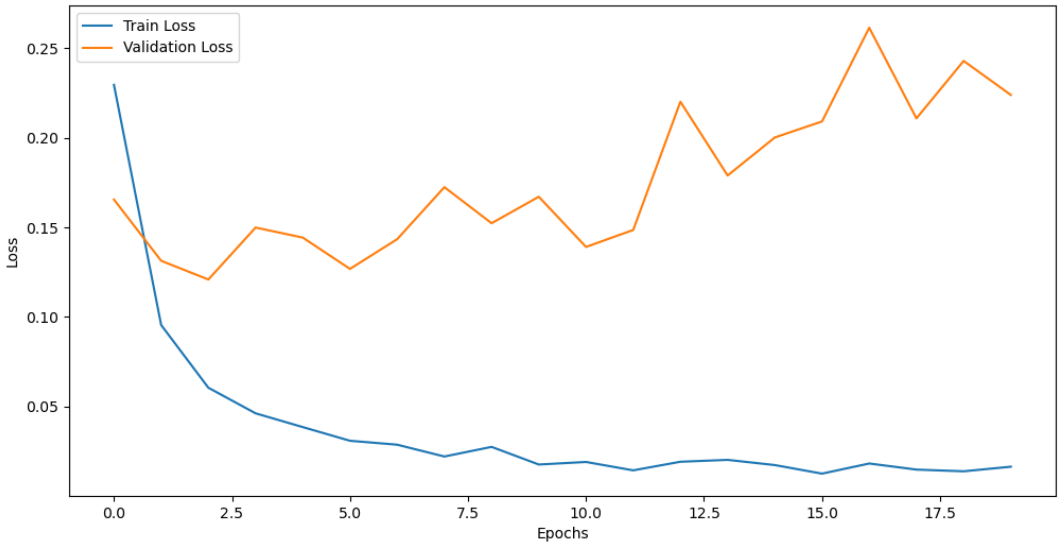
5

构建带有单个残差块的神经网络，包含两个全连接层，其中第一个全连接层的激活函数为ReLU，第二个不应用任何激活函数，隐含层数为1，包含100个节点，重复上述过程。

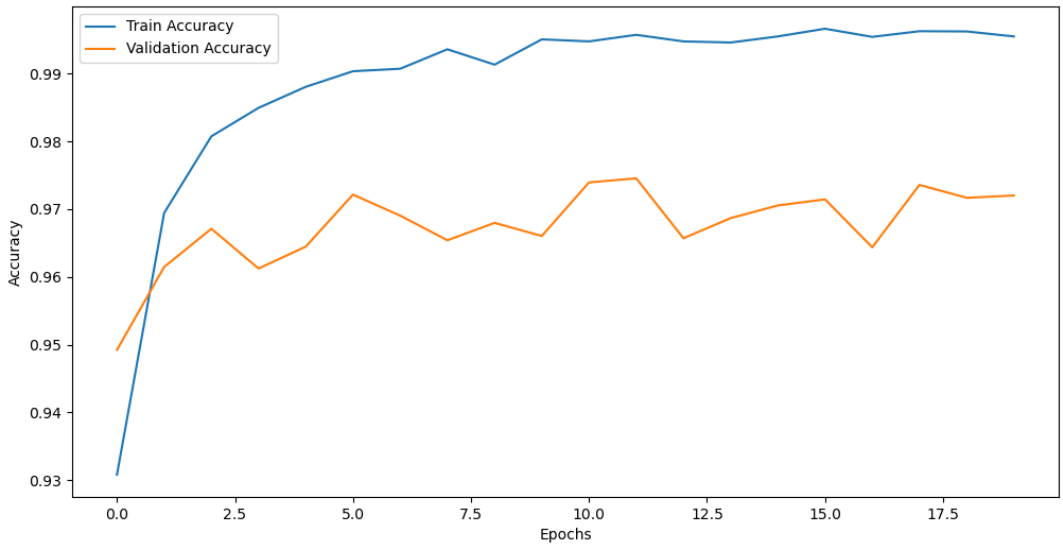
其中部分关键的代码如下，包括定义残差块、定义神经网络模型的部分

```
# 定义残差块
class ResidualBlock(tensorflow.keras.layers.Layer):
    def __init__(self, units=100):
        super(ResidualBlock, self).__init__()
        self.dense1 = tensorflow.keras.layers.Dense(units, activation='relu')
        self.dense2 = tensorflow.keras.layers.Dense(units, activation=None)

    def call(self, inputs):
        x = self.dense1(inputs)
        x = self.dense2(x)
        return tensorflow.nn.relu(x + inputs)

# 定义神经网络模型
def create_model(input_shape=(784,)):
    model = tensorflow.keras.models.Sequential([
        tensorflow.keras.layers.Input(shape=input_shape),
        ResidualBlock(784),  # 使用784个节点以保持维度一致
        tensorflow.keras.layers.Dense(10, activation='softmax')  # 输出层
    ])
    return model
```

损失函数关于训练次数的关系为



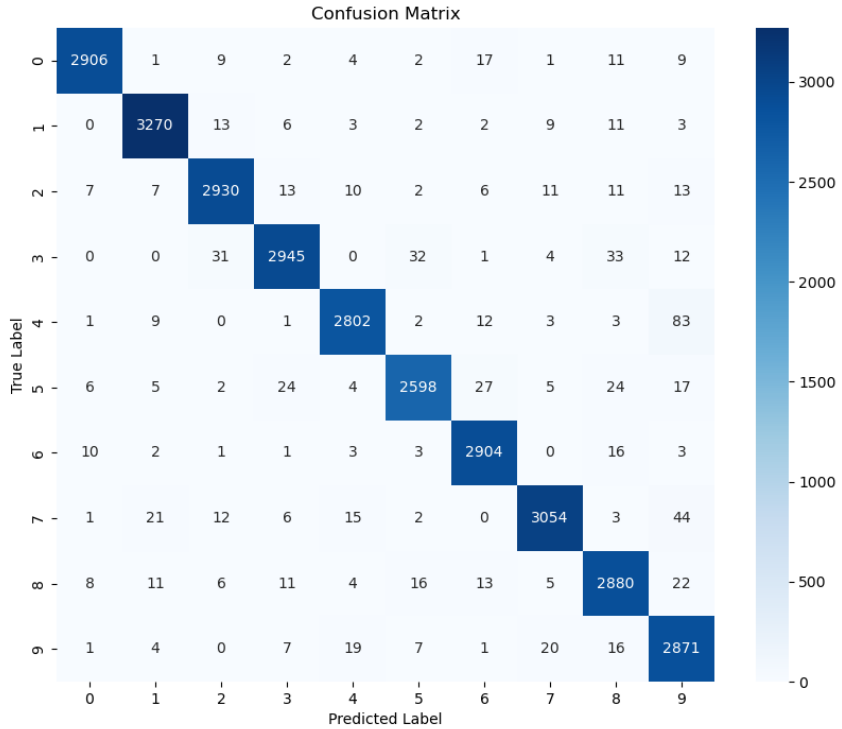
正确率关于训练次数的关系为



完成20次训练后，所花费的时间和正确率如下所示

正确率	时间/s
97.20%	106.63

混淆矩阵为



相对于层数、节点数相同的一般神经网络，残差神经网络训练时间更长，正确率也更高；不过，残差神经网络在训练次数较少的时候就展现出了非常好的正确率和较小的损失函数值，也即其所需的训练次数较小，而当训练次数不断增加，训练结果反而可能变差。

6

隐含层的节点数

基于`task4`可以发现，更多的节点数往往意味着显著更大的计算代价；但另一方面，在一定范围内，更多的节点数也能提高神经网络的准确性，这可能是因为更多的节点能抓取更多的样本特征，从而进行更准确的分类；但若节点数过多，神经网络过于复杂，也可能导致神经网络不仅学习了训练样本中有用的信息，还学习了其噪声和偏差，导致其在训练集上表现很好，但在测试集上表现较差，出现过拟合问题。

隐含层层数

基于`task2`、`task3`可以发现，更大的隐含层数量同样意味着更大的计算代价；另一方面，更多的隐含层加强了神经网络逐层提取不同层次的抽象特征、学习到更多特征的能力，但这并不直接意味着更高的判断准确率，因为更多的隐含层可能导致欠拟合，且可能更难对特征进行学习。

激活函数

对比`task2`、`task3`可以发现，不同的激活函数对神经网络的能力也有一定的影响，每种激活函数有其优势和不足，因而有对应的适用范围。例如，Sigmoid广泛用于二分类问题，但存在梯度消失问题：当输入值很大或很小时，梯度接近于零，导致权重更新非常缓慢，因而不适用于深层网络，因为容易导致梯度消失；ReLU函数具有计算简单且高效的优点，并解决了梯度消失问题，在正值区域梯度为1，加速了训练过程，但可能会导致“死亡ReLU”问题，即某些神经元可能永远输出0，从而不再参与训练。

残差神经网络

基于`task5`可以发现，相比于层数、节点数相同的一般神经网络，残差神经网络的计算资源消耗较大，但正确率更好、所需训练次数更少；最重要的是，残差神经网络允许梯度直接从输出层传递到前面的层，绕过了中间层，从而缓解了梯度消失和梯度爆炸的问题，这使得网络能够有效地训练更深的结构，具有更强的表达能力，能够捕捉更复杂的特征。