中山大学计算机学院

人工智能

本科生实验报告

（2022 学年春季学期）

课程名称：Artificial Intelligence

教学班级	**信息与计算科学班**	专业（方向）	**信息与计算科学**
学号	**21311359**	姓名	**何凯迪**

一、 实验题目

在 MNIST 手写数字数据集完成图像分类任务，在测试集完成测试，计算准确率。

要求：

设计多层感知机(至少两层全连接层)，并使用激活函数，选择合适的损失函数，并手动推导参数更新公式，利用训练集完成网络训练，并在测试集上计算准确率。应该包含参数更新公式的推导，损失的可视化展示，以及学习率对准确率影响的可视化展示。

二、 实验内容

1. 算法原理

多层感知机是一种基本的前馈神经网络，由多个全连接层组成。

输入层：多层感知机的输入层接受原始数据作为输入特征，每个输入特征对应一个输入神经元。

隐藏层：多层感知机可以包含一个或多个隐藏层。每个隐藏层由多个神经元组成，每个神经元都与上一层的所有神经元相连，并且每个连接都有一个对应的权重。隐藏层通过学习逐渐提取输入数据中的高级特征。

输出层：多层感知机的输出层通常包含一个或多个神经元，每个神经元对应一个输出类别。输出层的激活函数通常根据问题的要求而定，例如对于二分类问题可以使用 sigmoid 函数，对于多分类问题可以使用 softmax 函数。

前向传播：多层感知机的前向传播通过逐层计算神经元的加权和，并将加权和输入激活函数进行非线性变换，得到每个神经元的输出。

反向传播：在前向传播过程中，通过计算损失函数，可以得到模型的预测值与真实标签之间的差异。为了最小化这个差异，需要使用反向传播算法计算损失函数对于模型参数的梯度。

参数更新：通过梯度下降等优化算法，根据计算得到的梯度，更新模型的参数。更新参数的过程可以通过设置学习率和其他超参数来控制。

迭代训练：重复执行前向传播、反向传播和参数更新的步骤，直到达到预定的训练轮数或收敛条件。

激活函数：

激活函数是深度学习模型中的一种非线性函数，作用于神经网络的每个神经元上，用于引入非线性特性和增强模型的表达能力。通常包括以下步骤：

输入信号的加权求和：激活函数的输入是神经元的加权输入，即输入信号与对应连接权重的线性组合。

非线性变换：激活函数对输入信号进行非线性变换，将其映射到一个新的值域。这种非线性变换可以引入非线性特性，从而使神经网络能够对复杂的数据模式进行建模。

输出信号的传递：激活函数的输出作为下一层神经元的输入信号，通过神经网络进行传递。

损失函数：

损失函数是用于衡量模型预测值与真实标签之间差异的函数，它是训练模型的关键部分。通常包括以下步骤：

定义损失函数的形式：根据问题类型和模型设计的需求，选择适当的损失函数形式。不同问题类型和模型类型可能需要不同的损失函数。

计算模型的预测值：通过前向传播将输入数据输入模型，得到模型的预测值。预测值的具体计算方式取决于模型的架构和输出类型。

计算预测值与真实标签之间的差异：将模型的预测值与真实标签进行比较，计算它们之间的差异。这个差异衡量了模型的预测性能。

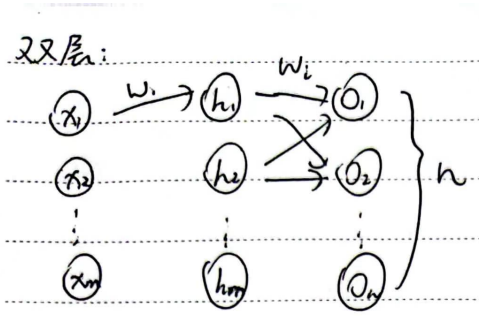
计算损失函数的值：根据差异的大小，使用损失函数的具体计算公式将预测值与真实标签的差异转化为一个标量值。这个标量值表示了模型的预测与真实标签之间的误差程度。

损失函数的优化：根据损失函数的值，使用优化算法（如梯度下降）来更新模型的参数，以最小化损失函数。通过迭代训练过程，不断调整模型的参数，使损失函数的值不断减小，以提高模型的预测性能。

梯度下降：

梯度下降算法的核心思想是通过迭代优化参数，使模型的损失函数逐渐减小，从而找到使损失函数最小化的参数值。通过反复计算梯度并更新参数，模型在训练过程中逐渐调整参数，从而改善预测结果。

推导参数更新公式：



1. 输出层参数更新：(以w_1为例)

误差 $E_{total} = \sum_{k=1}^{n} \frac{1}{2}(t_k - output_k)^2$

$$\frac{\partial E_{total}}{\partial w_i} = \frac{\partial E_{total}}{\partial output_1} \cdot \frac{\partial output_1}{\partial IN_{01}} \cdot \frac{\partial IN_{01}}{\partial w_i} \quad ①$$

$$\frac{\partial E_{total}}{\partial output_1} = \frac{\partial \frac{1}{2}(t_1 - output_1)^2}{\partial output_1} = output_1 - t_1 \quad ②$$

激活函数采用 sigmoid：$h(x) = \frac{1}{1+e^{-x}}$ $\therefore h'(x) = h(x)\cdot(1-h(x))$

$$\therefore \frac{\partial output_1}{\partial IN_{01}} = output_1 \cdot (1 - output_1) \quad ③$$

$$IN_{01} = \sum w \cdot output_h \quad \therefore \frac{\partial IN_{01}}{\partial w_i} = output_{h_1} \quad ④$$

将①~④联立起来

$w_i = w_i - \eta \cdot \frac{\partial E_{total}}{\partial w_i} = w_i - \eta(output_1 - t_1)\cdot output_1(1-output_1)\cdot output_{h_1}$ η为学习率

隐藏层参数更新（W_1、W_2为例）

$$\frac{\partial E_{total}}{\partial W_1} = \frac{\partial E_{total}}{\partial out_{puth_1}} \cdot \frac{\partial out_{puth_1}}{\partial Inh_1} \cdot \frac{\partial Inh_1}{\partial W_1} \quad ①$$

$$\frac{\partial out_{puth_1}}{\partial Inh_1} = out_{puth_1}(1 - out_{puth_1}) \quad ②$$

$$\frac{\partial Inh_1}{\partial W_1} = x_1 \quad ③$$

$$\frac{\partial E_{total}}{\partial out_{puth_1}} = \frac{\partial E_{o_1}}{\partial out_{puth_1}} + \cdots + \frac{\partial E_{o_n}}{\partial out_{puth_1}} = \sum_{i=1}^{n} \frac{\partial E_{o_i}}{\partial out_{puth_1}}$$

$$= \sum_{i=1}^{n} \left(\frac{\partial E_{o_i}}{\partial out_{put_i}} \cdot \frac{\partial out_{put_i}}{\partial Inh_1} \cdot \frac{\partial Inh_i}{\partial out_{puth_1}} \right)$$

$$= \left(\sum_{i=1}^{n} \delta_i \cdot W_{h_i} \right)$$

其中 δ_i 为 $\frac{\partial E_{total}}{\partial Inh_{o_i}}$，$W_{h_i}$ 为隐藏层到输出层的权重.

$$\therefore W_1 = W_1 - \eta \cdot \frac{\partial E_{total}}{W_1} = \delta_{h_1} \cdot x_1 \left(\sum_{i=1}^{n} \delta_i \cdot W_{h_i} \right) + out_{puth_1}(1 - out_{puth_1})$$

2. 伪代码

调用 tensorflow 库实现如下：

```
# 加载MNIST数据集
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 数据预处理
x_train = x_train.reshape((-1, 784)) / 255.0
x_test = x_test.reshape((-1, 784)) / 255.0
y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)

# 定义模型
model = Sequential([
    Dense(256, activation='sigmoid', input_shape=(784,)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# 编译模型
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 训练模型
history = model.fit(x_train, y_train, batch_size=128, epochs=10, validation_split=0.1)

# 在测试集上评估模型
loss, accuracy = model.evaluate(x_test, y_test)
print("测试集上的准确率：", accuracy)

# 可视化损失函数
```

手动实现（关键部分）如下：

```
输入：训练集(train_data)，其中存储输入向量；正确结果(train_label)；
      初始化的输入层-隐藏层参数矩阵(w)和隐藏层-输出层参数矩阵(v)；偏置(offset1, offset2)。
输出：经过训练后的输入层-隐藏层参数矩阵(w)和隐藏层-输出层参数矩阵(v)，以及更新后的偏置(offset1, offset2)。

函数 Train()：对于每个 cnt 在范围(len(train_data)) 内：
# 前向传播计算
hidden_value = train_data[cnt] * w + offset1 # 计算隐藏层的输入值
hidden_result = h(hidden_value) # 隐藏层的输出结果
out_value = hidden_result * v + offset2 # 计算输出层的输入值
out_result = h(out_value) # 输出层的输出结果，h为激活函数
  # 计算误差的导数
  E = 真实值 - out_result  # 计算误差
  delta_output = E * out_result * (1 - out_result)  # 输出层的导数
  delta_hidden = hidden_result * (1 - hidden_result) * np.dot(v, delta_output)  # 隐藏层的导数

# 权重和偏置的更新
for i in range(output_num):
    v = v - output_learn_rate * delta_output[i] * hidden_result  # 更新隐藏层-输出层的参数矩阵v
for i in range(hidden):
    w = w - input_learn_rate * delta_hidden[i] * train_data[cnt]  # 更新输入层-隐藏层的参数矩阵w
offset2 += output_learn_rate * delta_output  # 更新输出层的偏置offset2
offset1 += input_learn_rate * delta_hidden  # 更新隐藏层的偏置offset1
```

3. 关键代码展示（带注释）

```
def load_images(filename):
    with gzip.open(filename, 'rb') as f:
        f.read(16)   # 跳过文件头部信息
        buf = f.read()   # 读取图像数据
        data = np.frombuffer(buf, dtype=np.uint8)  # 转换为 NumPy 数组
        data = data.reshape(-1, 28*28)             # 重塑为二维数组
    return data

def load_labels(filename):
    with gzip.open(filename, 'rb') as f:
        f.read(8)     # 跳过文件头部信息
        buf = f.read() # 读取标签数据
        data = np.frombuffer(buf, dtype=np.uint8)   # 转换为 NumPy 数组
    return data

# 数据预处理
x_train = x_train / 255.0  # 归一化像素值到 0-1 范围
x_test = x_test / 255.0    # 归一化像素值到 0-1 范围
y_train = tf.keras.utils.to_categorical(y_train)    #对标签进行 one-hot 编码
y_test = tf.keras.utils.to_categorical(y_test)      #对标签进行 one-hot 编码

# 定义模型
model = Sequential([
    Dense(256, activation='sigmoid', input_shape=(784,)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# 编译模型(使用多分类交叉熵损失 categorical_crossentropy)
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# 训练模型（指定每个批次的大小为 128，迭代训练 10 个周期（epochs），将测试集划分为训练集的 10%
用于验证模型性能）
history = model.fit(x_train, y_train, batch_size=128, epochs=10, validation_split=0.1)

# 可视化学习率对准确率的影响
for lr in learning_rates:
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
loss='categorical_crossentropy', metrics=['accuracy'])
    model.fit(x_train, y_train, batch_size=128, epochs=5, verbose=0)
    _, accuracy = model.evaluate(x_test, y_test)
    accuracies.append(accuracy)
```

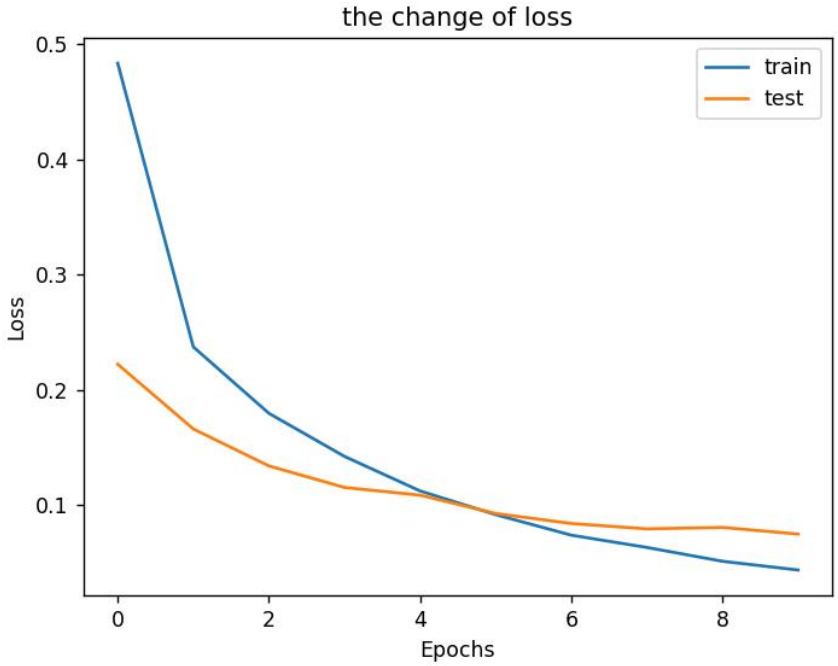
4. 创新点&优化（如果有）

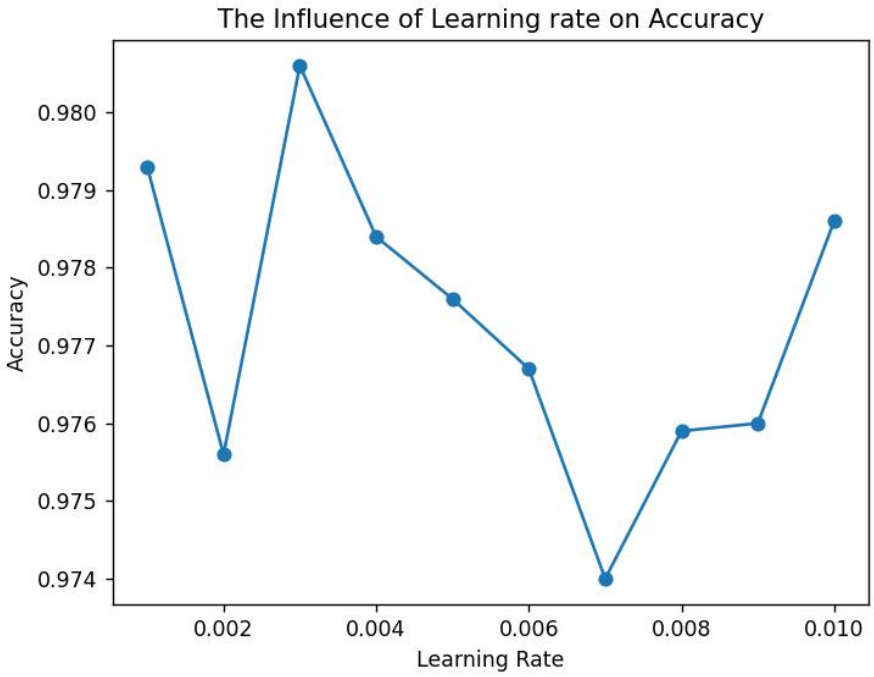
对不同激活函数的组合做了预实验（sigmoid+sigmoid+softmax，relu+relu+softmax，sigmoid+relu+softm，relu+sigmoid+softmax），并进行多次实验挑选出平均正确率最高的一个组合。

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

（激活函数使用 sigmoid+relu+softmax，迭代 10 次，展示某次结果）





```
预测值:  0 真实值:  0
预测值:  1 真实值:  1
预测值:  2 真实值:  2
预测值:  3 真实值:  3
预测值:  4 真实值:  4
预测值:  5 真实值:  5
预测值:  6 真实值:  6
313/313 [==============================] - 0s 801us/step - loss: 0.0729 - accuracy: 0.9793
313/313 [==============================] - 0s 837us/step - loss: 0.0997 - accuracy: 0.9756
313/313 [==============================] - 0s 821us/step - loss: 0.1017 - accuracy: 0.9806
313/313 [==============================] - 0s 814us/step - loss: 0.1137 - accuracy: 0.9784
313/313 [==============================] - 0s 813us/step - loss: 0.1230 - accuracy: 0.9776
313/313 [==============================] - 0s 811us/step - loss: 0.1206 - accuracy: 0.9767
313/313 [==============================] - 0s 828us/step - loss: 0.1305 - accuracy: 0.9740
313/313 [==============================] - 0s 824us/step - loss: 0.1130 - accuracy: 0.9759
313/313 [==============================] - 0s 794us/step - loss: 0.1428 - accuracy: 0.9760
313/313 [==============================] - 0s 945us/step - loss: 0.1175 - accuracy: 0.9786
PS D:\python文件\exp10>
```

2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

预实验结果对比如下：

（对于多类别分类任务，默认选择 softmax 作为输出层激活函数）

Accuracy	0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009	0.01	Average
type1	0.9785	0.9803	0.9775	0.9763	0.9766	0.9735	0.9777	0.975	0.9709	0.9755	0.97618
type2	0.9802	0.9758	0.977	0.9783	0.9771	0.9729	0.9709	0.9746	0.9754	0.9767	0.97589
type3	0.98	0.9761	0.9763	0.9756	0.9753	0.9775	0.9772	0.9739	0.976	0.9765	0.97644
type4	0.9814	0.9759	0.9786	0.9773	0.9734	0.9752	0.9721	0.9743	0.9688	0.9729	0.97499

type1:sigmoid+sigmoid+softmax
type2:relu+relu+softmax
type3:sigmoid+relu+softmax
type4:relu+sigmoid+softmax

最终挑选 type3 作为实际实验的激活函数组合。

四、 参考资料

https://blog.csdn.net/bestrivern/article/details/101755218