高级人工智能大作业

姓　　名： 赵路路

学　　号： 2023202210120

小组成员： 王亚龙、沈静远、甘泽昊、赵家璇

编写日期： 2023 年 12 月 26 日

目 录

1 手写数字识别实验

1.1 实验要求

使用以下算法实现手写数字识别任务：

（1）KNN 算法，可以使用 sklearn 库，也可以自己实现。

（2）全连接的神经网络模型，可以使用 sklearn 库。

（3）全连接的神经网络模型，仅可使用 numpy 库，手动实现。

（4）CNN 卷积神经网络模型，可以使用 sklearn 库。

最终使用测试集评估算法的性能，比较各算法的差异，并形成实验报告：

（1）对于 KNN 算法，需设置不同的 K 值，比较结果差异；

（2）对于神经网络模型，需设置不同学习率、隐层节点数，比较结果差异；

（3）对比上述所有模型的性能差异。

1.2 数据集介绍

DBRHD （Pen-Based Recognition of Handwritten Digits）是 UCI 机器学习中心提供的手写数字数据集，他们使用一台 WACOM PL-100V 压感平板，连接到基于 Intel 486 的 PC 的串口收集手写样本，最终收集到来源于 44 位不同书写者的手写数字数据，每个数据条项包括 16 维特征及标签。

（1）训练集：来源于 30 位书写者的 7494 张手写数字特征及其对应标签。

（2）测试集：来源于 14 位书写者的 3498 张手写数字特征及其对应标签。

数据分布如图 1-1 所示：

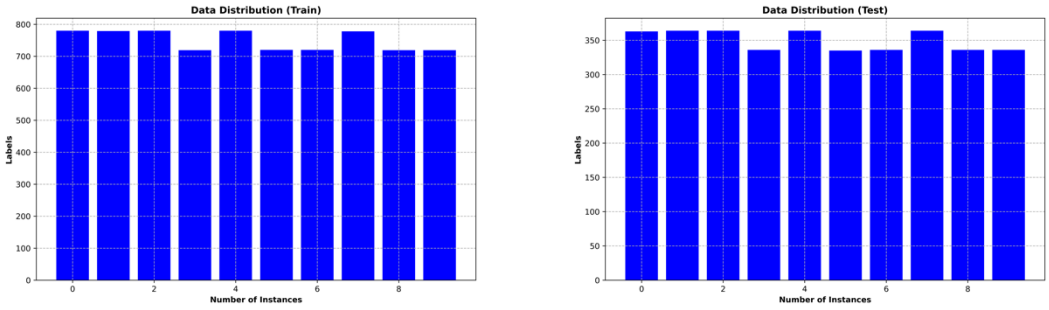

图 1-1 数据集分布柱状图

1.3 算法一：KNN

1.3.1 算法介绍

K 最近邻算法（K-Nearest Neighbors，KNN）是一种常用的监督学习算法，用于分类和回归问题。它基于样本之间的距离度量进行预测，即根据最邻近的 K 个训练样本的标签来确定新样本的标签。如图 1-2 所示，KNN 算法的基本思想是，如果一个样本在特征空间中的 K 个最近邻居中的大多数属于某个类别，则该样本也属于这个类别。

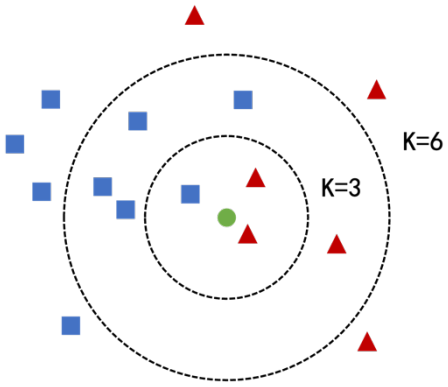

图 1-2 KNN 算法示意图

KNN 算法的主要步骤如下：

（1）准备数据集：收集训练样本数据集，包括输入特征和对应的标签。

（2）选择 K 值：确定 K 的取值，即要考虑的最近邻居的数量。K 值的选择会影响算法的性能，通常需要通过交叉验证等方法进行选择。

（3）计算距离：对于一个新的测试样本，计算它与训练样本集中每个样本之间的距离。常用的距离度量方法包括欧氏距离、曼哈顿距离等。

（4）选择最近邻居：根据距离大小选择与测试样本最近的 K 个训练样本。

（5）进行预测：对于分类问题，根据 K 个最近邻居的标签进行投票，将得票最多的类别作为测试样本的预测类别。对于回归问题，可以取 K 个最近邻居的平均值作为测试样本的预测值。

KNN 算法简单易用、无需训练过程、对异常值不敏感。然而，由于需要计算测试样本与所有训练样本之间的距离，KNN 算法的计算复杂度较高。此外，特征空间的维度较高时，需进行降维或特征选择等处理。

1.3.2 代码实现

（1）sklearn 库实现

如图 1-3 所示，初始化 KNN 分类器 KNeighborsClassifier，输入训练集数据，对测试集中数据进行类别预测，计算测试集上的准确率。

```
1 个用法  zllwhu
@get_time
def train_and_predict():
    for i in k:
        print("当前 k 值: " + str(i))
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train, y_train)
        y_pred = knn.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        result_accuracy.append(accuracy)
        print("当前精度: " + str(accuracy))
```

图 1-3 KNN 算法 sklearn 实现代码图

设置 K 值为 1～7 重复实验，得到分类准确率结果如图 1-4 所示。

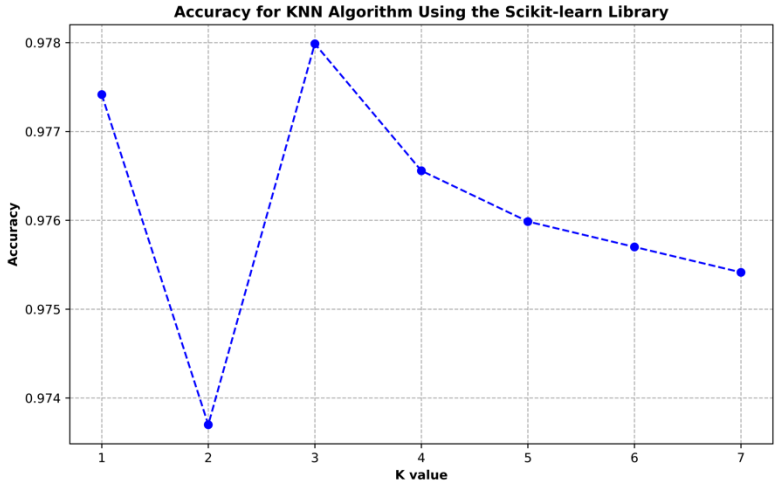

图 1-4 KNN 算法 sklearn 实现结果图

（2）手动实现

如图 1-5 所示，构建一个 KNN 分类器，定义预测函数。预测函数需首先计算输入的待预测样本与训练集所有样本之间的距离（定义为欧式距离），然后按照距离从小到大进行排序，选取前 K 个最近的训练样本点，

根据这 K 个点的标签投票表决待预测样本的标签。

```
zllwhu
class KNN:
    zllwhu
    def __init__(self, k, X_train, y_train):
        self.k = k
        self.X_train = X_train
        self.y_train = y_train

    zllwhu
    def predict(self, X_test):
        distances = []
        for i in range(len(self.X_train)):
            distance = euclidean_distance(self.X_train[i], X_test)
            distances.append((distance, self.y_train[i]))
        distances.sort(key=lambda x: x[0])
        k_nearest = distances[:self.k]
        k_nearest_labels = [label for (_, label) in k_nearest]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]
```

图 1-5 KNN 算法手动实现代码图

```
        return most_common[0][0]
```

设置 K 值为 1～7 重复实验，得到分类准确率结果如图 1-6 所示。

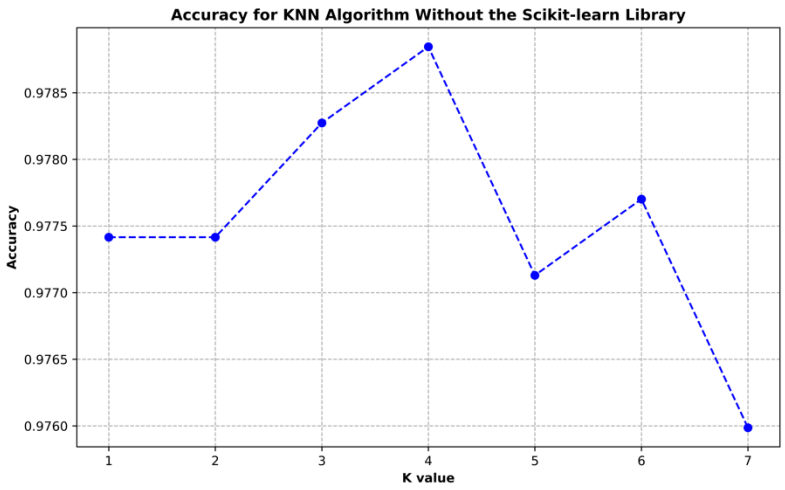

图 1-6 KNN 算法手动实现结果图

1.3.3 实验结果

统计 sklearn 和手动实现 KNN 算法测试结果，如表 1-1 和表 1-2 所示。

表 1-1 KNN 算法测试结果表（准确率）

实现方式	K 值						
	1	2	3	4	5	6	7
sklearn	0.9774	0.9737	0.9780	0.9766	0.9760	0.9757	0.9754
手动	0.9774	0.9774	0.9783	0.9788	0.9771	0.9777	0.9760

表 1-2 KNN 算法测试结果表（错误数）

实现方式	K 值						
	1	2	3	4	5	6	7
sklearn	79	91	76	81	83	85	86
手动	79	79	75	74	80	78	83

重复实验多次，统计并计算 sklearn 和手动实现的 KNN 算法对测试集样本进行预测的平均耗时，如表 1-3 所示。

表 1-3 KNN 算法测试平均耗时表

实现方式	平均耗时/秒
sklearn	2.8612
手动	1376.2428

通过统计并分析实验结果，可以发现 sklearn 实现和手动实现的 KNN 算法，在准确率指标上的表现差距极小，但手动实现的计算耗时远高于 sklearn 实现，原因分析如下：

（1）sklearn 库实现 KNN 算法时会对训练数据随机打乱（shuffle），在前 K 个最近邻样本点投票表决时，出现相同票数会随机选择最终输出标签；在手动实现中，对于相同票数情况，选择在训练集中顺序靠前的标签输出，故二者在准确率指标上略有差异。

（2）在计算性能上，由于 sklearn 库经过高度优化，采用 Cython 等底层实现方法提高运行效率。此外，sklearn 库在计算时采用并行化技术加速运算。

1.4 算法二：全连接神经网络

1.4.1 算法介绍

全连接神经网络（Fully Connected Neural Network，FCNN），也被称为多层感知机（Multilayer Perceptron，MLP），是一种经典的人工神经网络模型。它由多个神经元层组成，其中每个神经元与前一层的所有神经元连接，并将其输入加权求和后通过激活函数进行非线性转换。

如图 1-7 所示，为全连接神经网络的一般结构示意。

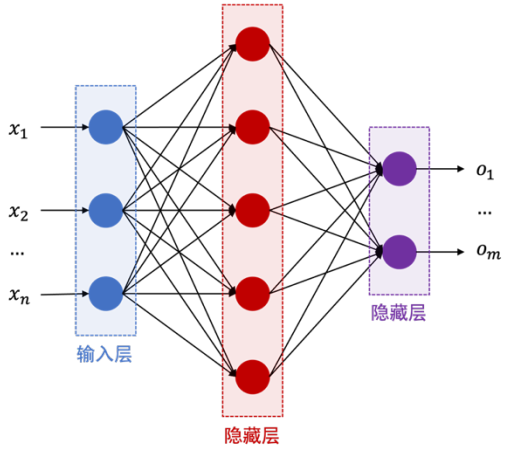

图 1-7 全连接神经网络一般结构示意图

（1）输入层（Input Layer）：接受输入数据的层。每个输入特征都对应一个输入神经元。

（2）隐藏层（Hidden Layers）：中间层，位于输入层和输出层之间。每个隐藏层包含多个神经元，这些神经元与前一层的所有神经元相连接。

（3）输出层（Output Layer）：产生模型的输出结果。输出层的神经元数量取决于任务的要求，例如二分类问题通常使用一个神经元，多分类问题可能使用多个神经元。

全连接神经网络的训练过程主要包括前向传播和反向传播两个阶段：

（1）前向传播（Forward Propagation）：在前向传播中，输入数据通过网络的各个层，每个神经元将其输入加权求和并应用激活函数。这样，网络逐层计算输出，直到达到输出层，产生模型的预测结果。

（2）反向传播（Backpropagation）：在反向传播中，通过比较模型的预测

结果和真实标签，计算损失函数的梯度。然后，梯度从输出层向输入层传播，根据链式法则更新每个神经元的权重，以最小化损失函数。

全连接神经网络在许多领域广泛应用，包括图像分类、语音识别、自然语言处理等。然而，全连接神经网络的参数量随着网络层数和神经元数量的增加而增加，容易导致过拟合问题。为了应对这个问题，可以使用正则化技术、Dropout、批归一化等方法来提高模型的泛化能力。

1.4.2 代码实现

（1）sklearn 实现

如图 1-8 所示，通过两层循环的设置，使用 sklearn 库提供的多层感知机模型 MLPClassifier 构建不同学习率和隐藏层神经元个数的全连接神经网络。此外需要注意，本次试验采用随机梯度下降优化器、固定学习率的模式，而非 Adam 优化器、自适应调整学习率的模式。激活函数选择使用 sigmoid 型函数，最大迭代轮次为 2000 轮。使用训练数据对模型进行训练，将训练得到的模型用于测试数据的测试，计算模型的分类准确率记录。

```
1 个用法  zllwhu *
@get_time
def train_and_predict():
    for j in range(len(learning_rate)):
        print("当前学习率: " + str(learning_rate[j]))
        for i in range(len(hidden_layer_neural_unit)):
            print("当前隐层神经元数量: " + str(hidden_layer_neural_unit[i]))
            model = MLPClassifier(hidden_layer_sizes=hidden_layer_neural_unit[i],
                                  activation='logistic',
                                  solver='sgd',
                                  random_state=None,
                                  learning_rate='constant',
                                  learning_rate_init=learning_rate[j],
                                  max_iter=2000,
                                  early_stopping=False,
                                  shuffle=False)
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)
            accuracy = accuracy_score(y_test, y_pred)
            result_accuracy[j].append(accuracy)
            print("当前精度: " + str(accuracy))
```

图 1-8 全连接神经网络 sklearn 实现代码图

设置学习率为 0.1、0.01、0.001、0.0001，隐藏层神经元个数为 500、1000、1500、2000，重复实验，得到分类准确率结果如图 1-9 所示。

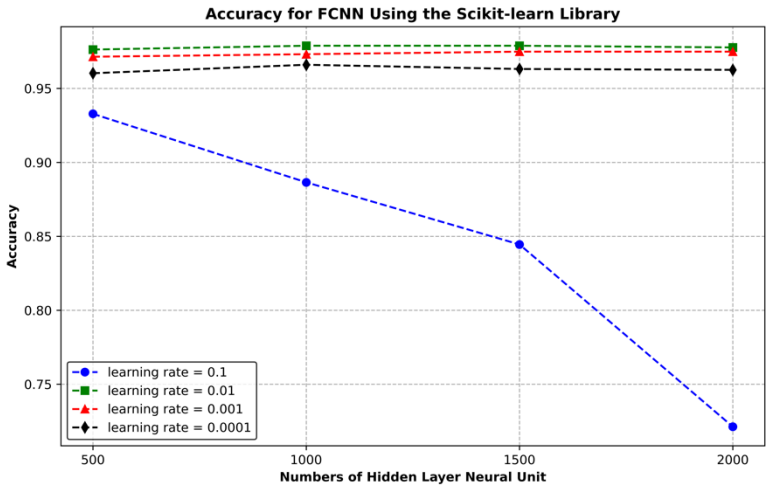

图 1-9 全连接神经网络 sklearn 实现结果图

（2）手动实现

如图 1-10 所示，定义全连接神经网络类 NeuralNetwork，定义成员变量输入尺寸、隐藏层尺寸、输出尺寸、学习率、最大迭代轮数、分组大小。

```
zllwhu *
class NeuralNetwork:
    zllwhu *
    def __init__(self, input_size, hidden_size, output_size,
                 learning_rate, max_iter, batch_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate
        self.max_iter = max_iter
        self.batch_size = batch_size
        self.weights1 = None
        self.biases1 = None
        self.weights2 = None
        self.biases2 = None
```

图 1-10 全连接神经网络手动实现类定义代码图

如图 1-11 所示，定义前向传播过程，依次计算输入数据的加权和与非线性变换得到的激活值，计算第二层的加权和与非线性变换得到的激活值，最终将两层激活值返回。

zllwhu
```
def forward_propagation(self, X):
    Z1 = np.dot(X, self.weights1) + self.biases1
    A1 = self.sigmoid(Z1)
    Z2 = np.dot(A1, self.weights2) + self.biases2
    A2 = self.sigmoid(Z2)
    return A1, A2
```

图 1-11 全连接神经网络手动实现前向传播代码图

如图 1-12 所示，定义反向传播过程，依次计算输出层权重和偏差的梯度，隐藏层权重和偏差的梯度，返回计算得到的梯度用于参数更新。

zllwhu
```
def backward_propagation(self, X, y, A1, A2):
    m = X.shape[0]
    dZ2 = A2 - y
    dW2 = np.dot(A1.T, dZ2) / m
    dB2 = np.mean(dZ2, axis=0, keepdims=True)
    dZ1 = np.dot(dZ2, self.weights2.T) * A1 * (1 - A1)
    dW1 = np.dot(X.T, dZ1) / m
    dB1 = np.mean(dZ1, axis=0, keepdims=True)
    return dW1, dB1, dW2, dB2
```

图 1-12 全连接神经网络手动实现反向传播代码图

如图 1-13 所示，定义参数更新过程，更新操作将根据梯度和学习率的乘积，以梯度下降的方式沿着损失函数的梯度方向调整神经网络的参数，以尽量减小损失。

zllwhu
```
def update_parameters(self, dW1, dB1, dW2, dB2):
    self.weights1 -= self.learning_rate * dW1
    self.biases1 -= self.learning_rate * dB1
    self.weights2 -= self.learning_rate * dW2
    self.biases2 -= self.learning_rate * dB2
```

图 1-13 全连接神经网络手动实现参数更新代码图

如图 1-14 所示，定义模型训练过程，包括参数初始化、迭代更新、前向传播、反向传播和参数更新过程，通过不断迭代重复训练模型。

```
zllwhu
def train(self, X, y):
    self.initialize_parameters()
    num_samples = X.shape[0]

    for _ in range(self.max_iter):
        for batch_start in range(0, num_samples, self.batch_size):
            batch_end = batch_start + self.batch_size
            X_batch = X[batch_start:batch_end]
            y_batch = y[batch_start:batch_end]

            A1, A2 = self.forward_propagation(X_batch)
            dW1, dB1, dW2, dB2 = self.backward_propagation(X_batch, y_batch, A1, A2)
            self.update_parameters(dW1, dB1, dW2, dB2)
```

图 1-14 全连接神经网络手动实现模型训练代码图

如图 1-15 所示，定义模型实例化、训练、预测和结果记录过程。

```
1 个用法  zllwhu
@get_time
def train_and_predict():
    for j in range(len(learning_rate)):
        print("当前学习率: " + str(learning_rate[j]))
        for i in range(len(hidden_layer_neural_unit)):
            print("当前隐层神经元数量: " + str(hidden_layer_neural_unit[i]))
            output_size = y_train_encoded.shape[1]
            model = NeuralNetwork(X_train.shape[1], hidden_layer_neural_unit[i], output_size,
                                  learning_rate=learning_rate[j], max_iter=2000, batch_size=200)
            model.train(X_train, y_train_encoded)
            y_pred = model.predict(X_test)
            accuracy = np.mean(y_pred == y_test)
            result_accuracy[j].append(accuracy)
            print("当前精度: " + str(accuracy))
```

图 1-15 全连接神经网络手动实现模型实例化代码图

设置学习率为 0.1、0.01、0.001、0.0001，隐藏层神经元个数为 500、1000、1500、2000，重复实验，得到分类准确率结果如图 1-16 所示。

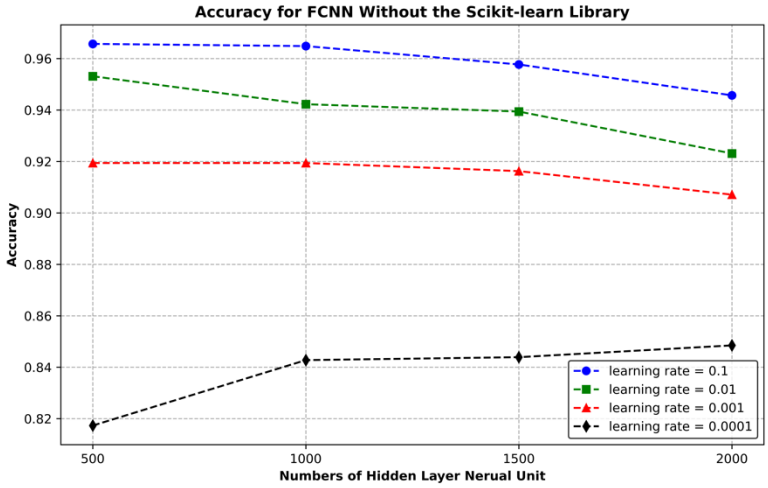

图 1-16 全连接神经网络手动实现结果图

1.4.3 实验结果

统计 sklearn 和手动实现的全连接神经网络准确率测试结果，如表 1-4 和表 1-5 所示。

表 1-4 全连接神经网络测试结果表（sklearn 实现）

学习率	隐藏层神经元数量			
	500	1000	1500	2000
0.1	0.9328	0.8865	0.8445	0.7213
0.01	0.9763	0.9788	0.9788	0.9777
0.001	0.9714	0.9731	0.9748	0.9748
0.0001	0.9603	0.9660	0.9631	0.9626

表 1-5 全连接神经网络测试结果表（手动实现）

学习率	隐藏层神经元数量			
	500	1000	1500	2000
0.1	0.9657	0.9648	0.9577	0.9457
0.01	0.9531	0.9423	0.9394	0.9231
0.001	0.9194	0.9194	0.9162	0.9071
0.0001	0.8173	0.8428	0.8439	0.8485

重复实验多次，统计并计算 sklearn 和手动实现的全连接神经网络对测试集样本进行预测的平均耗时，如表 1-6 所示。

表 1-6 全连接神经网络测试平均耗时表

实现方式	平均耗时/秒
sklearn	791.3919
手动	3960.6146

通过统计并分析实验结果，可以发现 sklearn 实现和手动实现的全连接神经网络，学习率较大时，在准确率指标上的表现差距较小；学习率较小时，在准确率指标上的表现差距较大。此外，手动实现的计算耗时远高于 sklearn 实现。原因分析如下：

（1）sklearn 库使用小批量随机梯度下降等优化方式加快了模型收敛速度。

（2）在计算性能上，sklearn 库经过高度优化提高运行效率，且使用并行化技术加速运算。

1.5 算法三：卷积神经网络

1.5.1 算法介绍

卷积神经网络（Convolutional Neural Networks，CNN）是一类主要用于处理具有网格结构数据，如图像和视频的深度学习模型。CNN 在计算机视觉领域取得了很大成功，其设计灵感来自生物学中对视觉系统的理解，尤其是视觉皮层中的神经元的工作方式。

如图 1-17 所示，为卷积神经网络的一般结构示意。

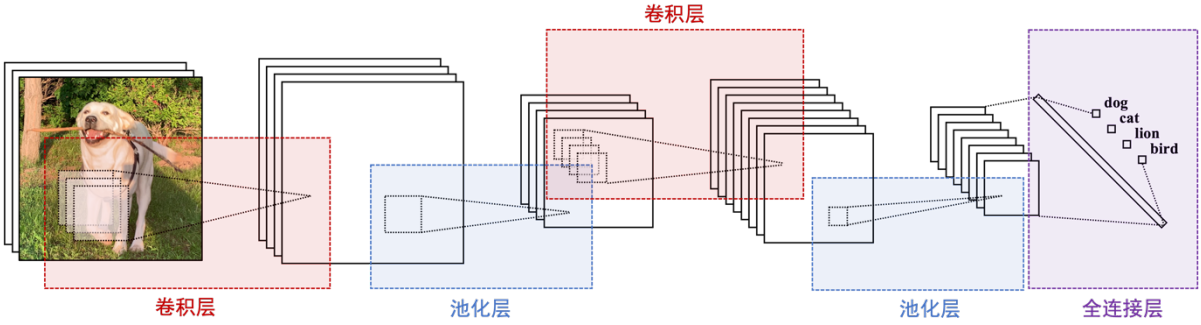

图 1-17 卷积神经网络一般结构示意图

（1）输入层（Input Layer）：用于接收输入数据，通常表示为图像。每个输入节点对应图像中的一个像素值或通道。

（2）卷积层（Convolutional Layer）：卷积层是 CNN 的核心组件。它通过在输入图像上滑动卷积核，学习图像中的特征。卷积操作可以有效地捕捉图像的局部模式，如边缘、纹理等。

（3）池化层（Pooling Layer）：池化层用于降低特征图的空间维度，减少计算复杂度，并提取图像的重要特征。常见的池化操作包括最大池化和平均池化。

（4）全连接层（Fully Connected Layer）：全连接层通常位于网络的顶部，将前面层的输出转换为最终的输出。全连接层的每个节点都与前一层的所有节点相连，通过权重连接。在图像分类任务中，全连接层通常用于输出类别概率。

（5）输出层（Output Layer）：输出层产生最终的预测结果。

如表 1-7 所示，是一些常见的卷积神经网络及其特点。

表 1-7 常见卷积神经网络表

网络名	网络特点
LeNet-5	LeNet-5 是早期的卷积神经网络，主要用于手写数字识别。包含卷积层、池化层和全连接层。
AlexNet	AlexNet 是在 2012 年 ImageNet 图像分类比赛中获胜的模型，推动了深度学习在计算机视觉中的应用。具有 8 个卷积层和 3 个全连接层。
VGGNet	VGGNet 的核心思想是使用多个 3x3 的小卷积核来代替一个大的卷积核，以增加网络的深度。VGGNet 有 16 层或者 19 层的深度。
GoogLeNet	GoogLeNet 使用了 Inception 模块，该模块同时使用多个不同大小的卷积核，提供了不同尺度的信息。具有 22 层的深度。
ResNet	ResNet 是使用残差块（Residual Block）的模型，通

	过引入残差连接，解决了深度网络中梯度消失问题。ResNet 赢得了 2015 年 ImageNet 图像分类比赛。
MobileNet	MobileNet 是一种轻量级的卷积神经网络，专注于在资源受限的环境中运行，如移动设备。使用深度可分离卷积来减少参数数量。
EfficientNet	EfficientNet 基于网络缩放的思想，通过在深度、宽度和分辨率上进行均衡调整，以在相对较少的参数下提高性能。

1.5.2 代码实现

如图 1-18 所示，使用 pytorch 库定义卷积神经网络类 CNN1D，构建网络结构为卷积层、激活层、池化层、全连接层。

其中，如图 1-19 所示，卷积层是尺寸为 3 的一维卷积核，步长为 1，零填充为 1；池化层是尺寸为 2 的池化核，步长为 2。

```
2 个用法  zllwhu
class CNN1D(nn.Module):
    zllwhu
    def __init__(self):
        super(CNN1D, self).__init__()

        self.conv1 = nn.Conv1d( in_channels: 1,  out_channels: 16, kernel_size=3, stride=1, padding=1)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool1d(kernel_size=2, stride=2)

        self.fc1 = nn.Linear(16 * 8,  out_features: 128)
        self.fc2 = nn.Linear( in_features: 128,  out_features: 10)
```

图 1-18 卷积神经网络类定义代码图

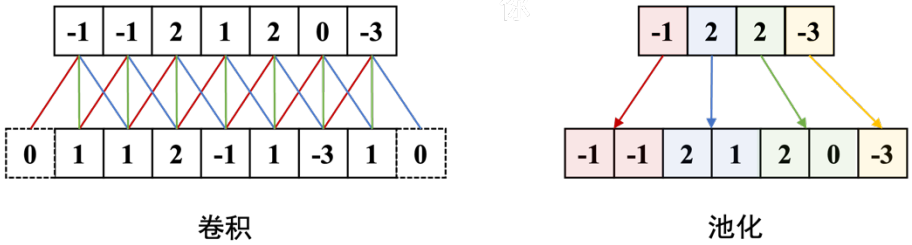

图 1-19 卷积和池化操作示意图

如图 1-20 所示，定义模型训练过程，包括选取批次、前向传播、反向传播和参数优化、计算平均损失并记录。

```
zllwhu
def forward(self, x):
    x = self.conv1(x)
    x = self.relu(x)
    x = self.maxpool(x)
    x = x.view(x.size(0), -1)
    x = self.fc1(x)
    x = self.relu(x)  # 添加ReLU激活函数
    x = self.fc2(x)
    return x
```

图 1-20 卷积神经网络模型训练代码图

如图 1-21 所示，定义模型评估过程，使用训练得到的模型对测试集数据进行预测，计算准确率并记录。

```
1 个用法  zllwhu *
@get_time
def train_and_predict():
    for epoch in tqdm(range(num_epochs)):
        model.train()  # 设置模型为训练模式
        total_loss = 0.0
        # 批量训练
        for i in range(0, len(X_train), batch_size):
            inputs = X_train[i:i + batch_size]
            labels = y_train[i:i + batch_size]
            # 梯度清零
            optimizer.zero_grad()
            # 前向传播
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            # 反向传播和优化
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        # 计算平均损失
        avg_loss = total_loss / (len(X_train) // batch_size)
        # 记录训练信息
        losses.append(avg_loss)
```

图 1-21 卷积神经网络模型评估代码图

1.5.3 实验结果

使用 Adam 优化器，批次大小 200，训练 30 轮后，平均交叉熵损失如图 1-22 所示。模型最终分类准确率为 0.9671，训练耗时 1.6274 秒。

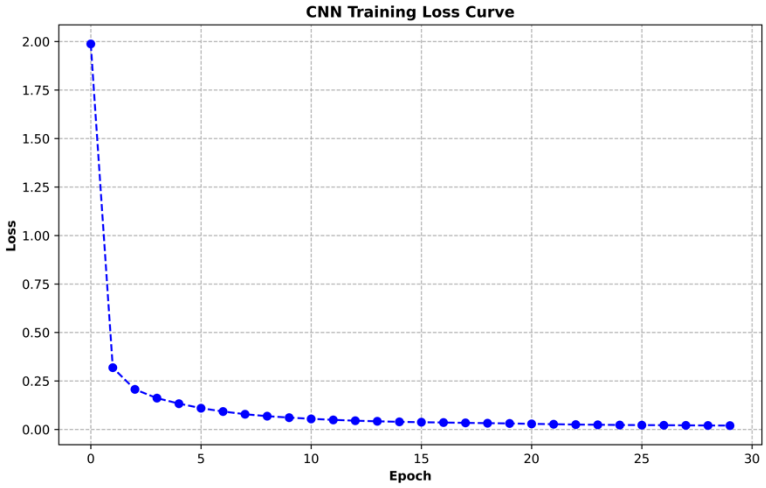

图 1-22 卷积神经网络 Loss 曲线图

2 小组论文分享报告

2.1 摘要

我们小组分享的论文是 NeurIPS 2022 的 Chain-of-Thought Prompting Elicits Reasoning in Large Language Models，这篇论文提出了思维链（Chain of Thought，COT），即通过一系列中间推理步骤，提升大语言模型的复杂推理能力。本文是 COT 的开山之作。下面我们将从背景介绍、方案涉及、实验测试和总结与展望四个方面介绍这篇论文。

姓名	分工
赵路路	论文阅读、PPT 制作
王亚龙	论文阅读、PPT 制作
沈静远	论文阅读、课堂汇报
甘泽昊	论文阅读、报告撰写
赵家璇	论文阅读、报告撰写

2.2 背景介绍

2.2.1 现有问题

自然语言处理（Natural Language Processing，NLP）领域近年来因语言模型而产生了革命性的变化。现有的研究表明扩大语言模型的规模可以为其带来一系列的好处，如提高性能和样本效率等。然而，仅扩大模型大小不足以在算术、常识和符号推理等具有挑战性的任务上实现高性能。

2.2.2 现有方案

文章受到以下两种想法的启发，提出了思维链，来解锁大语言模型的推理能力：

（1）算术推理技术可以从中间推理步骤中受益，研究表明中间推理步骤可以由从头开始训练的方式或者微调预训练的模型两种方式来生成。

（2）大预言模型通过上下文提示进行小样本学习有着令人兴奋的前景，也就是说通过几个演示任务的输入-输出示例来“提示”模型。

上述两种方案虽然都取得了不错的效果，但是都有十分关键的局限性。首先对于前者而言，通过训练和微调的方法创建大量高质量的推理依据的时间成本是非常高的；而对于小样本学习的方案来说，其在需要推理能力的任务上效果并不是很好，并且性能通常不会随着语言模型规模的增加而显著改善。

文章将这两种思想的优势结合起来，同时避免了它们的局限性。具体来说，文章首先改进了传统的提示，提出了由三元组（〈输入，思维链，输出〉）构成的思维链提示，并将其应用到推理任务中。思维链是一系列中间的自然语言推理步骤，这些步骤会导致最终的输出。

文章对算术、常识和符号推理基准进行了实证评估，表明文章提出的思维链提示优于标准提示，有时甚至达到惊人的程度。如图 2-1 是在数学单词问题 GSM8K 上的评测结果：

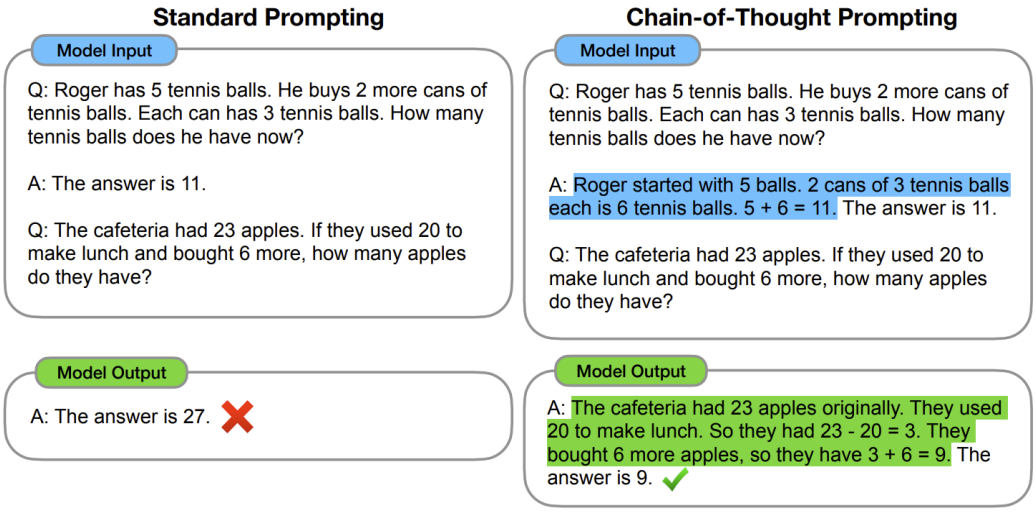

图 2-1 数学问题数据集评测结果示例图

使用 PaLM 540B 的思维链提示大大优于标准提示，并达到了目前最优秀的性能表现。仅提示的方法很重要，因为它不需要大型训练数据集，而且单个模型检查点可以执行许多任务而不会失去通用性。这项工作强调了大型语言模型如何通过几个带有任务自然语言数据的例子进行学习（例如，通过大型训练数据集自动学习输入和输出的基本模式）。文章强调了大语言模型通过任务相关的自然语言提示进行学习的重要性，因为它不需要大

型训练数据集，而且单个模型检查点可以执行许多任务从而不失通用性。

2.3 方案设计

接下来具体讲解思维链提示，当我们自己在解决复杂推理任务时，我们也会将问题分解为多个中间步骤，然后一步步解决直到给出最终的答案。文章的目标也是赋予语言模型类似思维链的能力，从而解决复杂的问题。具体来说，在标准提示的示例中加入一系列连贯的中间推理步骤，作为思维链，最终形成思维链提示。

作为促进语言模型推理的方法，思维链提示具有以下几个优点：

（1）思维链原则上来说可以让模型将多步任务分解为多个中间步骤，这意味着可以将额外的计算资源分配给需要更多推理的步骤。

（2）思维链为模型的生成行为提供了可解释性，表明模型是如何得出特定的答案的，并且提供了调试推理路径出错位置的可能性。

（3）思维链适用各种推理任务，如数学、常识推理等，通用性好，并且可能适用于人类可以通过自然语言解决的任何任务。

（4）思维链应用简单，只需要将思维链的示例序列注入到小样本标准提示中，就可以提高模型的推理能力。

2.4 实验测试

实验分为算术推理实验、常识推理实验和符号推理实验三个部分。

2.4.1 算术推理实验

在算术推理的实验中，作者选择了 5 个数学问题数据集。5 个数学问题分别为数学应用题的 GSM8K 基准、具有不同结构的数学应用题的 SVAMP 数据集，各种数学应用题的 ASDiv 数据集，代数应用题的 AQuA 数据集，以及 MAWPS 基准。在这 5 个数学问题上，作者人工设计了一套 8 个带有 COT 推理链条的 few-shot 样例，而且作者在五个数据集中统一使用了这 8 个带有 COT 推理链条的 few-shot 样例。作者在以下五种大模型下做了测试：GPT3、LaMDA、PaLM、UL2 20B、Codex.

算术推理实验结果如图 2-2 所示，黑线代表不使用 COT 的标准问题，蓝线代表使用了 COT 的测试问题。

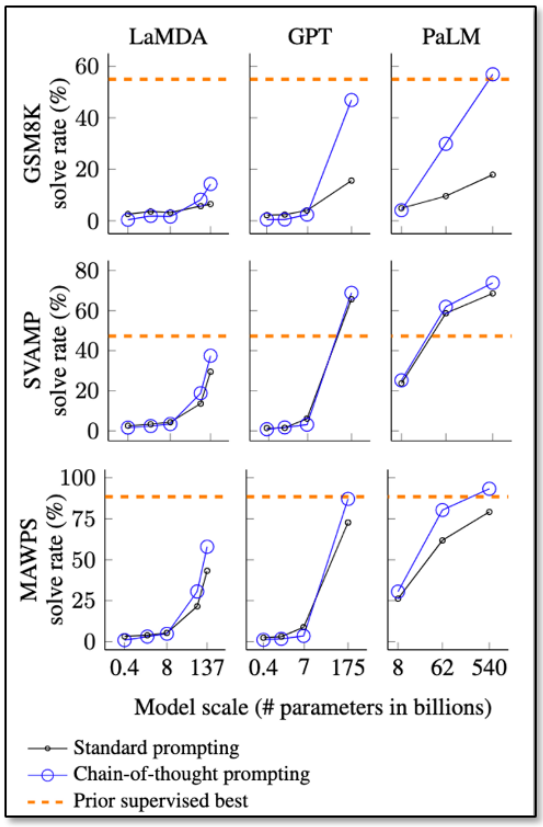

图 2-2 算术推理实验结果图

我们可以分析得到三大结论：

（1）COT 的能力是模型规模的产物。对于小规模模型，COT 对性能提升没有积极影响，定性分析中发现小规模模型产生了流畅但不合逻辑的 COT。

（2）COT 对复杂问题的性能提升更大。例如，对于 GSM8K 数据集中的数学问题，最大规模的 GPT-3 和 PaLM 模型的性能在使用 COT 后提升超过一倍。

（3）与基于特定任务微调模型的技术相比，COT 的性能提升更显著。

此外，为了更好理解 COT 的工作原理，作者手动检查了思维链生成的结果，在正确结果上，其思维链在逻辑和数学上都是正确的，在错误结果上，一半的思维链几乎正确，其余的错的离谱。此外为了探索模型规模对

COT 的影响，作者对 PaLM 62B 和 PaLM 540B 进行了类似的分析，发现后者在 COT 上修复了前者很大部分的缺失和语义理解错误。

使用思维链提示所观察到的好处自然提出了一个问题：是否可以通过其它类型的 prompt 来实现相同的性能改进？如图 2-3 展示了三种 COT 变体的消融实验。在消融实验中，作者尝试将 COT 修改为仅列出等式、替换为等长的省略号和将 COT 置于答案的后面。

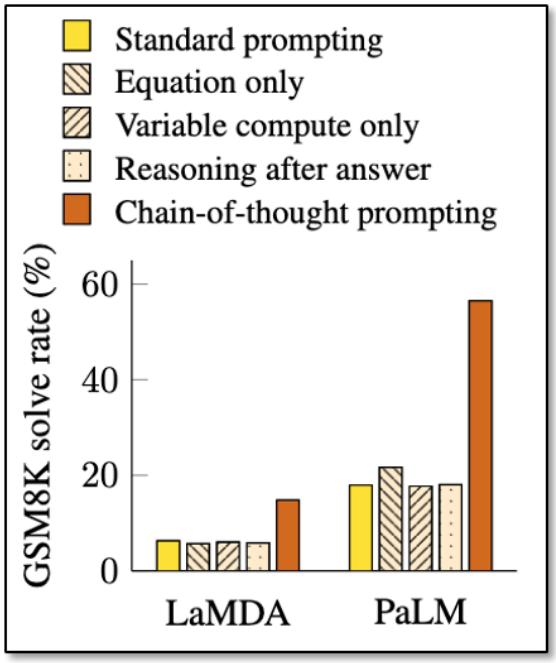

图 2-3 消融实验结果图

从图中我们可以看到：

（1）仅列出等式：一个可能解释思维链有效性的原因是它生成了用于评估的数学方程。因此，一种 COT 的变体是在给出答案之前仅输出数学方程。研究表明，仅输出方程式对于解决语义推理问题并不十分有帮助，因为它无法完全展现问题的解答过程。

（2）替换为等长的省略号：思维链允许模型在更复杂的问题上投入更多时间，那么这是否意味着性能提升归因于增加的计算量呢？为了探究这一点，研究者将 COT 替换为等长度的点序列。结果显示，这种方法与基线模型的表现相似，说明增加计算量并非 COT 成功的关键因素。

（3）将 COT 置于答案后面：最后一个变体是在模型给出答案之后附加思维链，以此来判断模型是否真的依赖 COT 来生成答案。这种方法的表现与基线模型大致相同，这表明模型确实依赖于 COT 来得出答案。

在鲁棒性实验中，不同的 prompt 信息会对大模型生成的结果产生巨大影响，例如打乱 few-shot 示例的顺序，就会将 GPT-3 在 SST-2 上的性能从 54.3%提升到 93.4%。因此作者尝试让多人编写不同的 COT、采用更简洁形式的 COT、从训练集中随机抽取示例，来验证 COT 的鲁棒性。

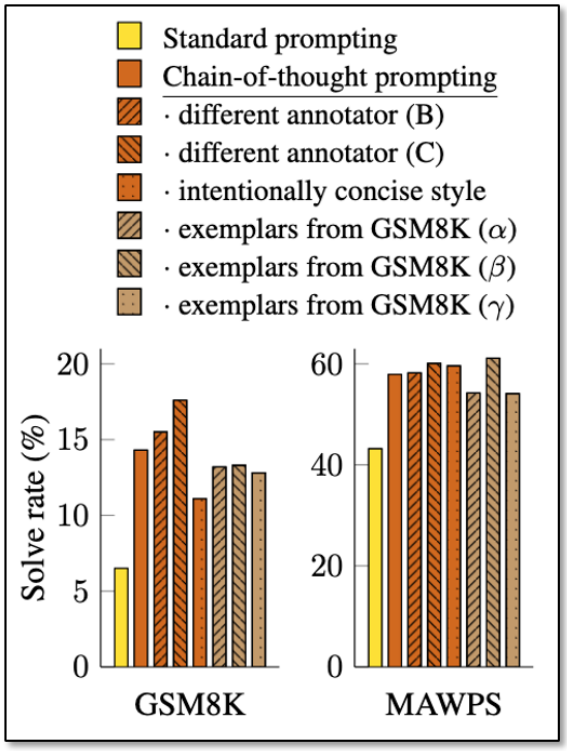

图 2-4 鲁棒性实验结果图

以上实验结果表明，尽管采用了不同的方法，但 COT 的性能并未出现较大的损失，从而证明了其鲁棒性。

2.4.2 常识推理实验

虽然 COT 特别适合解决数学问题，但是自然语言提示理论上在常识推理上更为实用。在常识推理实验中，作者在 CSQA、StrategyQA、Date、Sports、SayCan 这五个数据集上进行了实验，prompt 设置和数学推理实验中设置一致。结果如图 2-5 所示：

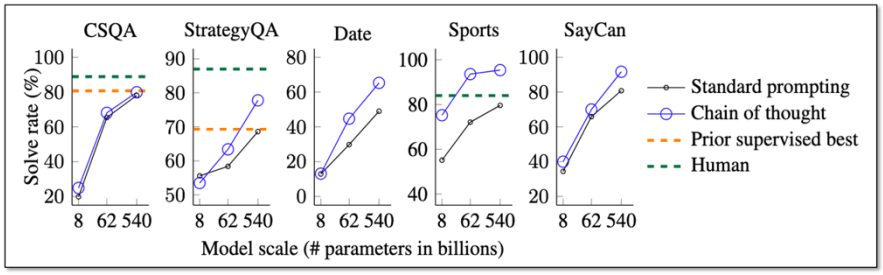

图 2-5 常识推理实验结果图

实验结果表明，COT 可以提高语言模型执行需要常识推理能力任务的性能。这项研究展示了 COT 提示在处理涉及广泛常识推理问题上的有效性，证明了它不仅适用于数学类问题，也能够有效处理涉及物理和人类互动的常识性问题。这强调了 COT 方法在促进大型语言模型进行复杂推理任务时的重要作用。

2.4.3 符号推理实验

最后评估 COT 在符号推理上的表现。符号推理任务对人类来说很简单，但是对标准的大模型来说是一个有挑战的任务。作者选取了词尾字母拼接和硬币翻转两个任务，前者要求模型将名字中最后一个字母拼接起来，后者要求模型回答硬币翻转特定次数后的正反。实验设置和前两个实验相同，结果如图 2-6 所示：

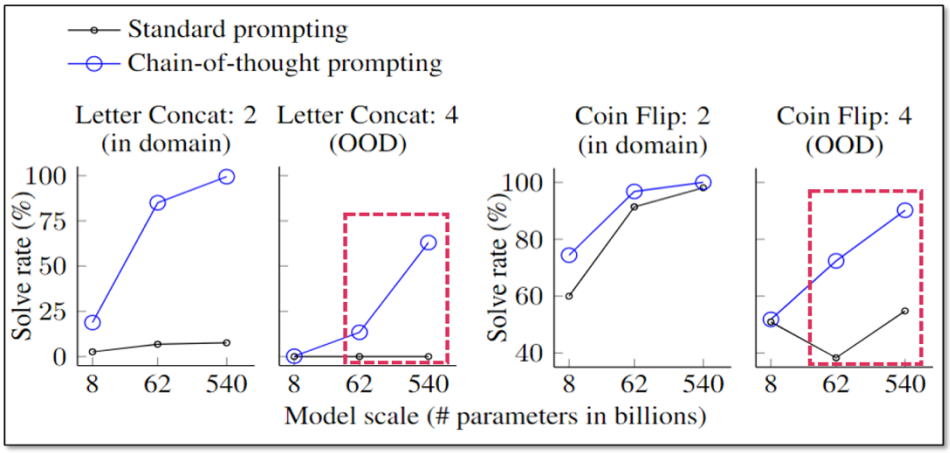

图 2-6 符号推理实验结果图

实验表明 PaLM 540B 模型使用思维链提示几乎达到了 100%的解决率，这在标准提示下已足以解决硬币翻转任务，但对于 LaMDA 137B 模型则不尽然。这些领域内评估被视为“玩具任务”，因为模型只需在测试时重复少

量样本中提供的思维链步骤。然而，在领域外任务中，标准提示表现不佳，而思维链提示使模型能够处理超出已见思维链长度的任务，尽管性能略有下降。

总的来说，这部分实验强调了思维链提示在帮助语言模型处理复杂符号推理任务方面的有效性，尤其是在领域外任务中，它显著提高了模型处理更长序列的泛化能力。

2.5 总结与展望

2.5.1 总结

文章探索了思维链提示在大型语言模型中引发多步推理行为的简单机制。在所有实验中，思维链推理都是通过提示现成的语言模型来简单地引出，过程中没有对语言模型进行微调，并大大提高了语言模型在算术推理、常识推理和符号推理上的性能，这些任务在标准提示中是失败的。通过进一步扩大模型规模，有望激发基于自然语言的推理方法的研究。

由于模型规模的限制，思维链推理的出现一直是热门话题。对于标准提示具有平坦的性能缩放曲线的许多推理任务，思维链提示使得性能曲线升高。思维链提示似乎扩展了大语言模型能够成功执行的任务集——换句话说，标准的提示仅提供大语言模型能力的下限。这也引发了文章作者的一系列思考：

（1）随着模型规模的增加，还期望模型理解能力能够提升多少？

（2）还有什么其他的提示方法能扩展语言模型可以解决的任务范围？

此外，文章还提出了一些局限性。

（1）尽管思维链模拟了人类推理的思维过程，但这并不能说明神经网络是否真的进行了“推理”，这仍是一个悬而未决的问题。

（2）思维链不能保证正确的推理路径，这可能导致错误的答案。

（3）尽管在 few-shot 设置中，用思维链手动扩充样本的成本很小，但这种方式的注释成本对于微调操作来说可能令人望而却步（这个可以通过合成数据生成或零样本泛化来克服）。

2.5.2 展望

思维链推理仅在大型模型上有明显的性能提高，这使得其在实际应用中成本很高，进一步的研究可以探索如何在较小的模型中利用思维链推理。

2.6 阅读总结

本文作为思维链的开山之作，理论简单且易于理解，实验设置丰富，并且消融实验构造巧妙，是一篇无论是写作还是实验都特别值得借鉴的工作。当然作为开山之作，肯定有很多值得改进的地方，比如本文是手动构建的 COT，这种方式质量难以保证，并且效率不高，成本巨大，一个省时省力的工作是利用 chatGPT 等优质大语言模型自动构建 COT，当然除了构建的形式不同，COT 本身也有很多可以做的工作，比如多阶段的 COT 会进一步提升模型的效果。强化学习+COT 也能进一步提高 COT 的质量等。

3 个人技术报告《联邦学习隐私保护前沿技术报告》

3.1 联邦学习技术

2016 年是人工智能技术走向成熟的历史性元年。随着 AlphaGo 击败顶尖的人类围棋选手，我们真正见证了人工智能的巨大潜力，并开始期待更复杂、更尖端的人工智能技术可以应用在更多应用领域，如无人驾驶、智慧医疗、电子金融等。如今，人工智能技术已经在各行各业展现出其卓越的优势。

随着人工智能应用的普及，涌现了一些亟待解决的痛点问题，如数据隐私、社会不平等、算法歧视等。这些问题引发了对人工智能的担忧，同时人们对资源的分配和使用也提出了更多关切。联邦学习的起源与人工智能的发展和隐私保护的需求密切相关。人工智能的进一步发展受到两方面的限制：

（1）数据隐私保护的挑战：随着人工智能应用的不断发展，涉及大规模敏感数据的训练需求也急剧增加。传统的中央化训练模型方法，需要将数据集中到中央服务器上，这引发了用户隐私和数据泄露的担忧。

（2）分布式数据的现实：很多场景下，数据分布在不同的设备上，例如智能手机、物联网设备等，这使得传统中央化学习的方式难以直接应用。

为了解决传统中央化学习中存在的隐私问题，谷歌的研究团队首次提出了联邦学习的概念[1]。联邦学习的核心思想是在保护用户隐私的前提下，通过在本地设备上进行模型训练，只将模型的更新信息进行聚合，从而实现在分布式数据上进行模型训练。

联邦学习可以形式化定义[2]如下：

N个数据持有者$\{\mathcal{F}_1, \dots, \mathcal{F}_N\}$希望通过整合各自的数据$\{\mathcal{D}_1, \dots, \mathcal{D}_N\}$训练一个模型。传统方式是将数据拼接$\mathcal{D} = \mathcal{D}_1 \cup \dots \cup \mathcal{D}_N$得到模型$\mathcal{M}_{SUM}$。联邦学习方式是用户$\mathcal{F}_i$不向其他用户披露自己的数据$\mathcal{D}_i$得到模型$\mathcal{M}_{FED}$，且模型精度与传统方式接近，即：

$$\forall \delta, |\mathcal{V}_{FED} - \mathcal{V}_{SUM}| < \delta \tag{1}$$

3.2 纵向联邦学习隐私保护

联邦学习可以分为两类，即横向联邦学习和纵向联邦学习。纵向联邦学习主要适用于拥有略微不同用户空间和不同特征集的数据所有者。从根本上讲，纵向联邦学习是从多个公司或组织中汇总这些不同特征的过程。

举例说明，如图 3-1 所示，银行 A 想建立一个预测客户信用的模型。电子商务公司 B 和 A 在同一个城市，所以他们的客户群相似。A 有客户的财务信息，B 有客户的在线消费信息。即 A 和 B 具有不同的客户数据特征。因此，B 可以利用其信息帮助 A 建立纵向联邦学习模型，该模型可以准确地评估客户的信用评级。在此过程中，应保护参与方的隐私性和用户数据的安全性[3]。

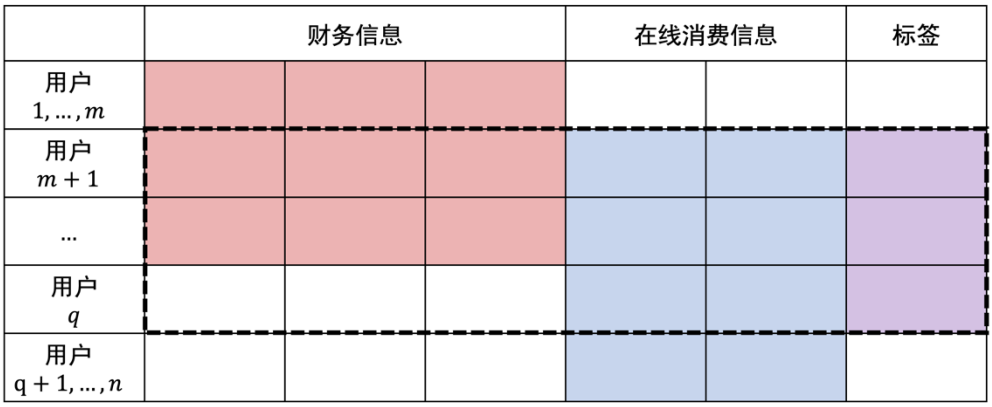

图 3-1 银行 A 和电子商务公司 B 之间的纵向联邦学习示意图

用户数据的安全性是纵向联邦学习的关键属性之一。大多数纵向联邦学习模型使用同态加密（Homomorphic Encryption，HE）方法[4]对包含样本特征的传输数据进行加密。同态加密对所有参与方的加密数据执行简单的计算（如加法、乘法），解密结果与直接计算数据的结果一致。作为一个经典的纵向联邦学习模型，Hardy 等人于 2017 年在纵向联邦学习中引入了 Paillier 同态加密方法[5]。在这个模型中，参与者可以传输加密的中间结果，以有效隐藏本地数据，当他们从所有参与方汇总特征时。然而，同态加密的缺陷在于计算成本较高，难以将这种安全方法应用于多方参与的情况。

随着纵向联邦学习参与者数量的增加，交互变得更加复杂，通信和计算成本可能更高。此外，同态加密只能对加密数据执行简单的计算，其他计算只能通过近似公式替代，导致准确度低于传统机器学习训练结果。

除了同态加密的缺点外，纵向联邦学习模型还存在由服务器引起的信息泄露风险。现有的纵向联邦学习模型需要一个身份验证服务器来协调和传输加密信息，但在这个过程中存在信息泄露的风险。同时，由于模型结构复杂，所有参与方都必须与协调者进行交互，造成较高的通信成本。

3.3 基于秘密分享的纵向联邦学习隐私保护方案

为解决上述两个问题，Shi 等人于 2022 年提出了一种基于秘密共享的高效多方纵向联邦学习方法（Multi-participant Vertical Federated Learning based on Secret Sharing，MVFLS）。

首先，在改进的多方纵向联邦学习中使用秘密共享替代同态加密，从而大大提高了通信和计算效率。其次，提出的模型可以在不进行近似的情况下计算准确的梯度和损失函数公式，与现有的纵向联邦学习方法相比，模型的准确性得到提升。最后，MVFLS 方法依赖于具有标签信息的一个参与方来协调所有参与方的计算，无需专用服务器，减少了通信轮数并防止第三方恶意发送错误信息。

3.3.1 问题形式化定义

对于N条数据样本$D=\{(x_i,y_i|i=1,\dots,N)\}$，其中$x_i\in R^{d\times1}$分发给$M$个参与方$P\{p_i|i=1,\dots,M\}$。参与方$p_M\in P$持有数据的标签，即持有数据$D_{p_M}=\{x^{p_M};y\}$，其余参与方持有数据$D_{p_a}=\{x^{p_a}\}$。目标是使用所有参与方持有的数据，在不泄露隐私的条件下建立一个联合机器学习模型：

$$\underset{\theta_{p_1},\dots,\theta_{p_M}}{\operatorname{argmin}}\ \mathcal{L}=\frac{1}{N}\sum_{p_a\in P}f\left(\theta_{p_a};D_{p_a}\right)+\frac{\lambda}{2}\sum_{p_a\in P}\left\|\theta_{p_a}\right\|^2 \tag{2}$$

其中，θ_{p_a}为参与方p_a的训练参数，$f(\cdot)$的线性回归和逻辑回归表达式如下：

$$f\left(\theta_{p_a}; D_{p_a}\right) = \left\| \sum_{p_a \in P} \theta_{p_a} x^{p_a} - y \right\|^2 \tag{3}$$

$$f\left(\theta_{p_a}; D_{p_a}\right) = \log\left(1 + e^{-y \sum \theta_{p_a} x^{p_a}}\right) - y \sum_{p_a \in P} \theta_{p_a} x^{p_a} \tag{4}$$

简化表达$u^{p_a} = \theta_{p_a} x^{p_a}$，回归方程（3）和（4）的梯度为：

$$\frac{\partial \mathcal{L}}{\partial \theta_{p_a}} = 2x^{p_a} \left\| \sum_{p_a \in P} u^{p_a} - y \right\| + \lambda \sum_{p_a \in P} \left|\left|\theta_{p_a}\right|\right| \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_{p_a}} = x^{p_a} \left(y - \frac{e^{\sum u^{p_a}}}{1 + e^{\sum u^{p_a}}} \right) \tag{6}$$

根据梯度下降法，学习率为η时，参数更新如下：

$$\theta'_{p_a} = \theta_{p_a} - \eta \frac{\partial \mathcal{L}}{\partial \theta_{p_a}} \tag{7}$$

因此，在学习过程中，各参与方需要获得$Y = \sum u^{p_a} - y$或$G = y - \frac{e^{\sum u^{p_a}}}{1+e^{\sum u^{p_a}}}$，才能通过线性回归或逻辑回归更新参数。在保证算法效率的前提下，获得没有隐私泄露的聚合数据Y或G，需要探索安全的数据聚合策略。

3.3.2 基于秘密分享的安全聚合

为确保在数据聚合过程中不泄露私人信息，MVFLS 方案设计了一种新颖的安全聚合方法，其中秘密共享意味着将一个秘密S分割成若干部分，其中一部分共享可以重构S。此方法依赖于一种已有的安全聚合方法[6]，并扩展了传统的秘密共享方法。

假设存在n个参与方，每个参与方持有数据u_a，目标是服务方在不获得u_a的前提下计算它们的和。如果u_a以特定方式被屏蔽，参与者将掩码值传输到服务器，则可以安全地计算。具体来说，我们假设每对用户(p_a, p_b)都协商一个随机数S_{p_a}或S_{p_b}（$S_{p_a} = S_{p_b}$），作为掩码隐藏实际值，即：

$$y_a = u_a + S_{p_a}, y_b = u_b - S_{p_b} \tag{8}$$

这样，服务器可以通过计算所有参与者输入的总和来计算聚合值：

$$y_a = u_a + \sum_{a<b;p_b \in P} S_{a,b} - \sum_{a>b;p_b \in P} S_{a,b} \tag{9}$$

$$Y = \sum_{p_a \in P} y_a = \sum_{p_a \in P} u_a \tag{10}$$

随机数的协商可以使用 Diffie-Hellman 密钥交换协议[7]完成。对于素数阶群G_q，生成元为g，协商的两个参与方分别选取随机数$sk_a < q$和$sk_b < q$作为私钥，计算$pk_a = g^{sk_a} \ mod \ q$和$pk_b = g^{sk_b} \ mod \ q$作为公钥，协商秘密值为$S_{p_a,p_b} = pk_b^{sk_a} = pk_a^{sk_b} = S_{p_b,p_a}$。更进一步，为减少通信开销，采用伪随机数发生器，将协商秘密值作为输入，获得最终使用的掩码值隐藏原始数据，即$PRG(S_{p_a,p_b}) = PRG(S_{p_b,p_a})$。

尽管基于加性秘密分享的安全聚合策略解决了用户数据私密性的需求，但依旧依赖于中心化的服务器。

3.3.3 无服务纵向联邦学习

本节将介绍如何将中心化纵向联邦学习转化为无服务纵向联邦学习。为叙述简洁，以三参与方p_a, p_b, p_c为例（其中仅p_c具有标签信息）。参与方p_a和参与方p_b协商掩码值，并将数据隐藏后发送给参与方p_c，即：

$$y_{p_a} = u_{p_a} + \ PRG(S_{p_a,p_b}), y_{p_b} = u_{p_b} + \ PRG(S_{p_b,p_a}) \tag{11}$$

参与方p_c据此聚合信息，计算$Y = \sum u^{p_a} - y$或$G = y - \frac{e^{\sum u^{p_a}}}{1+e^{\sum u^{p_a}}}$，通过线性回归或逻辑回归更新参数，并将更新后的参数返回给另外两个参与方。

在这种情况下，三参与方合作在纵向联邦学习范式下建立机器学习模型。此外，每一方计算的梯度与他们在没有隐私保护的情况下收到的损失相同，因此模型是无损的。接下来，继续探索如何扩展该模型的使用范围。

3.3.4 多方纵向联邦学习

将上述基于秘密分享的安全聚合策略与无服务纵向联邦学习策略融合，扩展至多方场景。在模型训练过程中，仅一方持有标签信息，则其作为秘密安全聚合执行方，其他参与方需将数据经隐藏后发送给该方，即：

$$y_{p_a} = u_{p_a} + \sum_{p_b \in P; p_b \neq p_M} \left(\sum_{a<b} PRG(S_{p_a,p_b}) - \sum_{a>b} PRG(S_{p_b,p_a}) \right) \tag{12}$$

$$Y = \sum_{p_a \in P; p_a \neq p_M} y_{p_a} + u_{p_M} - y = \sum_{p_a \in P} u_{p_a} - y \tag{13}$$

这种方法可以实现多方纵向联邦学习训练。然而，纵向联邦学习模型的效率取决于隐私方法的通信和计算成本。这种方法要求不持有标签信息的参与方与其余的参与者交互，需要存储他人的公钥值，因此这种关键协议协议不可避免地会浪费大量的存储空间和时间。因此，如图 3-2 所示，本文对该策略提出了两个改进。参与方只与邻域节点协商秘密值用于掩码隐藏数据。

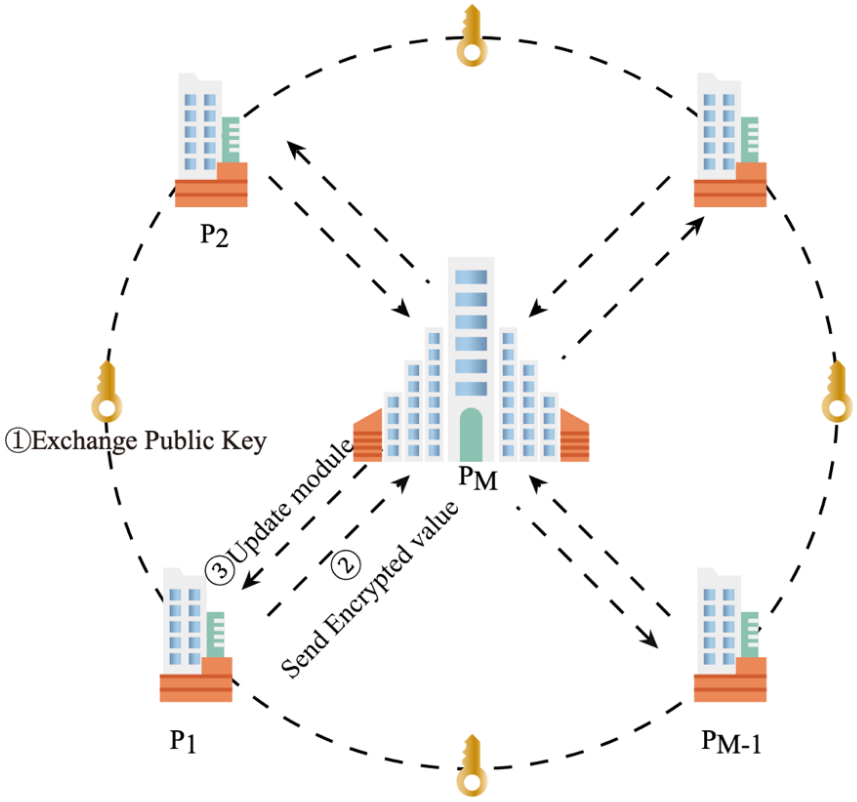

图 3-2 安全多方纵向联邦学习示意图

此外，为了降低通信成本，该模型可以在每一轮训练中减少样本数量。在每一轮开始时，选择一部分随机样本，参与方只计算所选样本的数据。

3.4 总结

本文介绍了人工智能技术在大数据背景下面临的数据隐私保护挑战。为了解决传统中央化学习中存在的隐私问题，联邦学习应运而生，代表了一种新兴的机器学习范式，为在敏感数据场景下的模型训练提供了有效的解决方案。更进一步地，本文介绍了一种基于秘密共享的多方纵向联邦学习方案（MVFLS）。该方法采用秘密共享解决了纵向联邦学习中的隐私保护问题，相较于基于同态加密的现有方法，具有更低的通信成本和更高的计算效率。

3.5 参考文献

[1] McMahan B, Moore E, Ramage D, et al. Communication-efficient learning of deep networks from decentralized data[C]//Artificial intelligence and statistics. PMLR, 2017: 1273-1282.

[2] YANG Q, LIU Y, CHEN T, et al. Federated Machine Learning[J/OL]. ACM Transactions on Intelligent Systems and Technology, 2019: 1-19. http://dx.doi.org/10.1145/3298981. DOI:10.1145/3298981.

[3] Shi H, Jiang Y, Yu H, et al. MVFLS: multi-participant vertical federated learning based on secret sharing[J]. The Federate Learning, 2022: 1-9.

[4] Paillier P. Public-key cryptosystems based on composite degree residuosity classes[C]//International conference on the theory and applications of cryptographic techniques. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999: 223-238.

[5] Hardy S, Henecka W, Ivey-Law H, et al. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption[J]. arXiv preprint arXiv:1711.10677, 2017.

[6] Bonawitz K, Ivanov V, Kreuter B, et al. Practical secure aggregation for privacy-preserving machine learning[C]//proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017:

1175-1191.

[7] Diffie W, Hellman M E. New directions in cryptography[M]//Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman. 2022: 365-390.