空气质量四分类任务：经典与量子机器学习模型设计与性能分析

作者[1]，作者[2]

（1. 作者详细单位，省市邮编；2. 作者详细单位，省市邮编;）

[1] 作者详细单位，省市邮编；[2] 作者详细单位，省市邮编；

摘　要: 摘要内容。概括地陈述论文研究的目的、方法、结果、结论，要求 200～300 字。应排除本学科领域已成为常识的内容；不要把应在引言中出现的内容写入摘要，不引用参考文献；不要对论文内容作诠释和评论。不得简单重复题名中已有的信息。用第三人称，不使用“本文”、“作者”等作为主语。使用规范化的名词术语，新术语或尚无合适的汉文术语的，可用原文或译出后加括号注明。除了无法变通之外，一般不用数学公式和化学结构式，不出现插图、表格。缩略语、略称、代号，除了相邻专业的读者也能清楚理解的以外，在首次出现时必须加括号说明。结构严谨，表达简明，语义确切。

关键词：关键词 1；关键词 2；关键词 3；关键词 4

中图分类号：作者本人填写　　**文献标识码**：A

引言内容。引言作为论文的开场白，应以简短的篇幅介绍论文的写作背景和目的，以及相关领域内前人所做的工作和研究概况，说明本研究与前人工作的关系，目前研究的热点、存在的问题及作者工作的意义。1、开门见山，不绕圈子。避免大篇幅地讲述历史渊源和立题研究过程。2、言简意赅，突出重点。不应过多叙述同行熟知的及教科书中的常识性内容，确有必要提及他人的研究成果和基本原理时，只需以引用参考文献的形势标出即可。在引言中提示本文的工作和观点时，意思应明确，语言应简练。3、引言的内容不要与摘要雷同，也不是摘要的注释。4、引言最好不要有插图、列表和数学公式。

1 概述

空气污染是全球性环境问题，对人类健康和生态系统产生了深远的影响。准确评估空气质量并预测污染等级对于制定环境保护政策和改善居民生活质量至关重要。本次比赛旨在利用量子机器学习技术，基于多维度的环境数据，预测空气质量等级，为城市环境管理提供决策支持。

我们的代码结构如下：

- `Data` 文件夹：存放数据集 `train_data.csv` 和 `test_data.csv`。
- `Utils.py`：实现数据读取、预处理等的工具类和函数。
- `ClassicalModel.py`：实现经典机器学习模型的网络框架。
- `QuantumModel.py`：实现量子机器学习模型的网络框架。
- `Trainer.py`：实现模型训练和评估。
- `Main.py`：主程序，负责调用各个模块，进行数据加载、模型训练和评估。

2 数据探索与预处理

本任务中提供了 `train_data.csv` 与 `test_data.csv` 两个数据集。两个文件具有相同的结构，从左到右依次为：温度（$°C$）、湿度（%）、$PM_{2.5}$ 浓度（$\mu g/m^3$）、PM_{10} 浓度（$\mu g/m^3$）、NO_2 浓度（ppb）、SO_2 浓度（ppb）、CO 浓度（ppm）、到最近工业区的距离（km）、人口密度（人 $/km^2$）、空气质量。其中，前九列是特征，最后一列即空气质量是标签，共有四类：Good、Moderate、Poor 和 Hazardous，在之后我们会将其映射为 $0 \sim 3$。

为了读取文件，我们在 `Utils.py` 文件中实现了 `CSVReader` 类。该类的构造函数读入一个数据集文件路径，自动加载文件中所有数据，包括列名、数据维度、缺失值等。通过调用函数，也可以查看读入数据的统计信息，例如均值和方差等。这个类中有以下方法：

- `__init__(self, filePath: str)`：构造函数，传入数据集文件路径，自动读取数据集。
- `__loadData__(self)`：读取数据集文件，没有返回值。会被构造函数自动调用，用户不得手动调用。
- `getData(self)`：以二元组返回数据集的特征和标签。
- `checkBasicInfo(self)`：输出检查数据集的基本信息，包括维度、缺失值等。
- `showStatistics(self)`：输出数据集的统计信息，包括均值、方差等。
- `read(filePath: str)`：静态方法，传入数据集文件路径，返回数据集特征和标签。用于快速读取数据集。

一般而言，为了模型能够更好地学习特征，我们会对数据进行预处理。由于数据量较小，简单的预处理即可完成需求。我们在 `Utils.py` 文件中实现了 `Math` 静态类。这个类中只有一个静态函数：

- `normalize(data: np.ndarray)`：对数据进行归一化处理，返回归一化后的数据。

这是为了保证不会修改 CSVReader 类中的数据。在 `Trainer.py` 文件中，`Trainer` 类读入数据后会自动调用 `Math.normalize` 函数对数据进行归一化处理。

3 经典神经网络模型

3.1 算法原理

我们实现了两种经典神经网络结构作为基准模型：残差网络和多层感知器。

3.1.1 SmallResNet（小型残差网络）

残差网络是一种深度学习架构，主要用于解决深层神经网络中的梯度消失和梯度爆炸问题。它的核心思想是引入“残差学习”机制，通过残差连接（Skip Connection）让网络中的信号可以绕过某些层直接传播。

- 残差块（**Residual Block**）：

$$\text{output} = \text{ReLU}(\text{Linear}_2(\text{ReLU}(\text{Linear}_1(x))) + x)$$

- 主网络:

$$\begin{aligned} x_1 &= \text{ReLU}(\text{Linear}_{\text{input}}(x)) \\ x_2 &= \text{BasicBlock}(x_1) \\ x_3 &= \text{Linear}_{\text{output}}(x_2) \\ \text{output} &= \text{Softmax}(x_3) \end{aligned}$$

3.1.2 SmallMLP（小型多层感知机）

多层感知机是一种简单的神经网络结构，由多个全连接层组成，通常用于分类和回归任务。它的核心思想是通过逐层的线性变换和非线性激活函数来学习输入数据的复杂映射关系。

- 多层感知机（**MLP**）：

$$\begin{aligned} x_1 &= \text{ReLU}(\text{Linear}_1(x)) \\ x_2 &= \text{ReLU}(\text{Linear}_2(x_1)) \\ x_3 &= \text{Linear}_{\text{output}}(x_2) \\ \text{output} &= \text{Softmax}(x_3) \end{aligned}$$

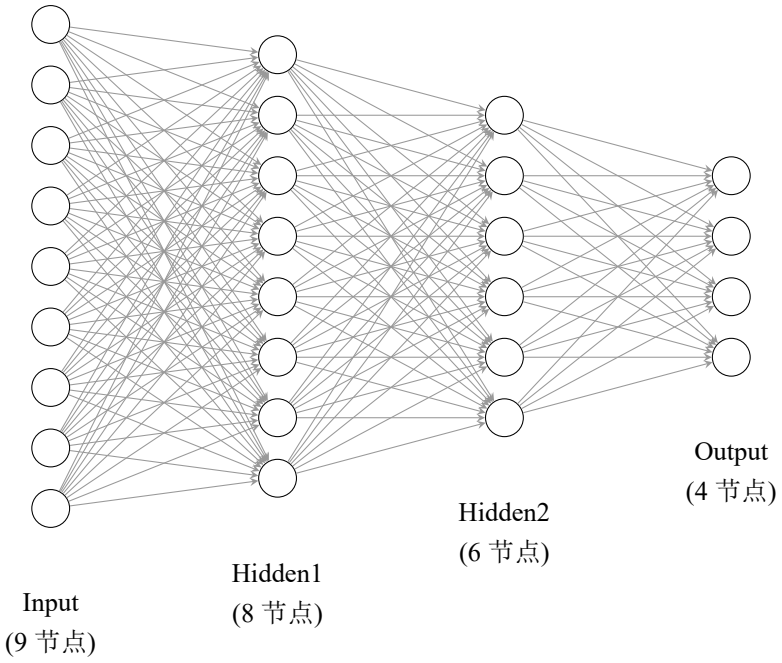

图 3.1 MLP 网络结构

3.2 实现细节

我们实现的一个 MLP 网络结构如图**??**所示。该网络包含一个输入层、两个隐藏层和一个输出层。输入层有 9 个节点，分别对应 9 个特征；第一个隐藏层有 8 个节点，第二个隐藏层有 6 个节点；输出层有 4 个节点，分别对应四类标签。每一层的神经元之间是全连接的。

3.3 性能评估

使用残差网络与多层感知机对数据集进行训练，得到如下所示实验结果

表 3.1 Training Results of Using Data Augmentation

Epoch	Scheduler	Train Accuracy	Valid Accuracy
50	SmallMLP	87.20%	85.60%
	SmallResNet	87.23%	85.60%

根据实验结果可知在固定参数大小下，经典的神经网络模型 ResNet 与 MLP 的实验精度基本一样，使用更复杂的 ResNet 模型对实验的结果精度并未产生较大提升。从模型理论上看，由于更复杂的模型 (ResNet) 在提出的时候往往是解决增加模型参数却不能极大提高准确率的问题，而在该任务中所用经典参数较少，并未达到 MLP 的模型性能极限，因此使用 ResNet 并未提升较大精度。

由以上实验分析，我们可以提出如下假设：在固定参数下，存在经典模型并未超出其性能极限，即在固定参数下更改模型并不能提升预测精度。在此假设下，我们希望存在某种模型能够在原有经典模型的基础上，能进一步较大提升模型的预测精度，从而减少模型的参数量，进而提高模型的计算效率。

4 量子机器学习模型

4.1 算法原理

4.1.1 量子数据编码

采用线性旋转编码将经典数据映射到量子态：

$$S(\mathbf{x}) = \bigotimes_{k=1}^{N} RX(x_k) = \exp\left(-i\sum_{k=1}^{N} x_k \sigma_x^{(k)}/2\right) \tag{4.1}$$

其中 $\sigma_x^{(k)}$ 为第 k 个量子比特的 Pauli-X 算符。

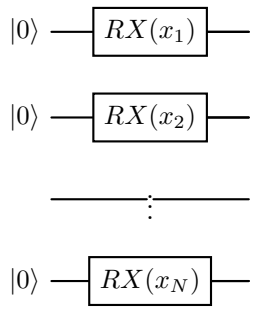

图 4.1 量子数据编码电路

4.1.2 变分量子电路

$$U(\theta) = \prod_{l=1}^{L} U_l(\theta_l) R_l(\mathbf{x}) \tag{4.2}$$

其中：

- $U_l(\theta_l)$：参数化量子门层
- $R_l(\mathbf{x})$：重上传单元

4.1.3 动态非线性机制

通过可调重上传单元 (RUU) 引入非线性：

$$R_l(\mathbf{x}) = \exp\left(-i f_l(\mathbf{x}) \sigma_j^{(k)}/2\right), \quad j \in \{x, y, z\} \tag{4.3}$$

典型配置包括：

- $RX(\mathrm{ReLU}(\mathbf{x}))$
- $RY(\mathbf{x}^2)$
- $RZ(\tanh(\mathbf{x}))$

4.1.4 参数化纠缠层

采用可训练的两比特门实现自适应纠缠：

$$CRX(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos\theta/2 & -i\sin\theta/2 \\ 0 & 0 & -i\sin\theta/2 & \cos\theta/2 \end{pmatrix} \tag{4.4}$$

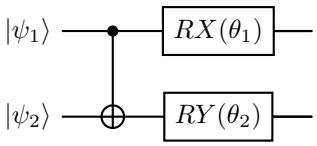

图 4.2 参数化纠缠层电路

图 4.3 QMLP 算法流程图

Algorithm 1 QMLP 训练流程

初始化参数 $\{\theta_l\}_{l=1}^{L}$ 和 $\{f_l\}_{l=1}^{L}$
for 每个训练 epoch **do**
    编码输入：$|\phi_0\rangle = S(\mathbf{x})\,|0\rangle^{\otimes N}$
    量子态演化：$|\phi_L\rangle = \prod_{l=1}^{L} U_l(\theta_l) R_l(f_l)\,|\phi_0\rangle$
    测量期望值：$\langle Z\rangle_i = \langle\phi_L|\, Z_i\,|\phi_L\rangle$
    计算损失：$\mathcal{L} = -\sum_c y_c \log p_c$
    通过参数偏移规则更新参数
end for

4.2 实现细节

4.3 性能评估

5 性能对比与分析

5.1 性能对比

5.2 差异分析

附录一：