变分量子算法

林元莘

2024 年 6 月 19 日

摘要

本文探讨了变分量子算法（VQAs）在量子计算领域的应用及其在 NISQ（Noisy Intermediate-Scale Quantum）设备上实现量子优势的潜力。VQAs 是一种基于优化或学习的方法，通过参数化的量子电路来近似未知量子态，并利用经典优化算法进行参数优化。这种方法的优势在于能够保持量子电路的深度较浅，从而减轻噪声影响。VQAs 已被考虑用于多种应用，包括质数分解、量子系统模拟和线性系统方程求解等。尽管 VQAs 在实现近期量子优势方面具有潜力，但它们仍面临可训练性、准确性和效率方面的挑战。文章还介绍了变分量子算法奇异值分解（VQSVD）的概念，这是一种通过训练参数化量子线路来近似矩阵奇异值分解的方法，并提供了使用华为 Mindspore 的 Mindquantum 库实现 VQSVD 算法的示例。最后，文章讨论了量子机器学习在分类任务中的应用，展示了如何将分类任务转化为优化问题，并使用量子神经网络（QNN）进行建模和训练。

关键词：变分量子算法；奇异值分解；量子神经网络

目录

1 引言

量子计算对许多应用都显示出强大的潜力，这些应用已经激发了数十年来构建必要物理硬件的追求。例如，随着量子算法对经典算法的指数级别加速，能够进行质数分解、模拟量子系统、求解线性系统方程等等。

然而，量子计算机真正的承诺——对实际应用的加速，通常被称为量子优势——尚未实现。此外，具有容错能力的量子计算机的可用性看起来仍然需要许多年甚至几十年的时间。因此，关键的技术问题是如何最好地利用当今的 NISQ 设备来实现量子优势。任何此类策略都必须考虑到量子比特数量有限、量子比特之间的连接有限，以及限制量子电路深度的相干和非相干错误。

变分量子算法（VQAs）[1] 已成为在 NISQ 设备上获得量子优势的领先策略。要用单一策略解决 NISQ 计算机施加的所有限制，需要一种基于优化或基于学习的方法，这正是 VQAs 所使用的。可以说，VQAs 是非常成功的机器学习方法（如神经网络）的量子类比。此外，VQAs 利用经典优化的工具箱，因为它们使用参数化的量子电路在量子计算机上运行，然后将参数优化外包给经典优化器。这种方法的额外优势在于保持量子电路的深度较浅，从而减轻噪声，这与为容错时代开发的量子算法形成对比。

VQAs 已经被考虑用于众多应用，基本涵盖了研究人员曾为量子计算机设想的所有应用。尽管它们可能是获得近期量子优势的关键，VQAs 仍面临重要挑战，包括其可训练性、准确性和效率。在这篇文章中，我们讨论了 VQAs 的激动人心的前景，并强调了为实现量子优势的终极目标必须克服的挑战。

2 基本概念

变分量子算法的一大好处就是，他们可以为一系列问题提供普适的框架。变分量子算法类似于深度学习，它把深度学习中的神经网络用被参数化的量子线路 $U(\theta)$ 代替。对于使用变分量子算法来解决问题，我们第一步要做的是定义一个损失函数 L 使得我们可以通过最小化 C 来找到我们想要的问题的解。于是，我们可以通过一种量子电路-经典电路循环交替的方法来解决一个优化任务

$$\theta^* = \arg\min_{\theta} L(\theta)$$

变分量子算法的特点便是，用一个参数化的量子线路来近似一个未知的量子态，用量子计算机来估计损失函数，然后通过经典优化算法来优化这个参数化的量子线路。

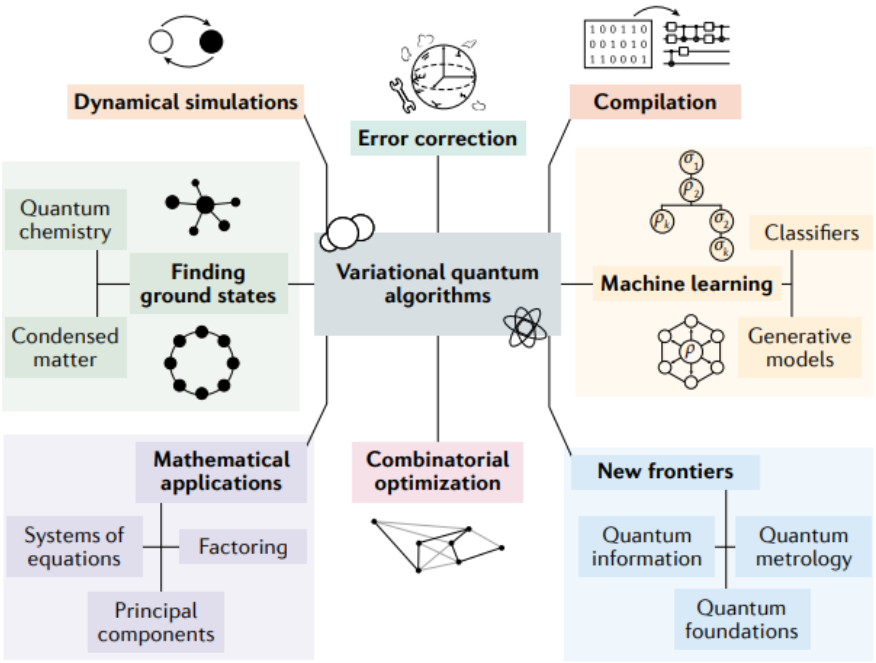

图 1: 变分量子算法的应用 [1]

2.1 损失函数

在变分量子算法中，我们需要定义一个损失函数 L，来编码我们想要解决的问题。类似于量子机器学习，损失函数将可训练的参数 θ 映射到一个实数。一般情况下，可以将损失函数表达为如下：

$$L(\theta) = f(\rho_k, O_k, U(\theta))$$

其中，ρ_k 是一个从训练集中输入进的状态，O_k 是一个观测量，$U(\theta)$ 是一个参数化的量子线路。

更多时候，我们会使用如下形式来表达损失函数：（假设参数化电路输出的波函数为 $|\psi>=U(\theta)|0>$）

$$L(\theta) = \sum_k f_k(Tr[O_k U(\theta)\rho_k U(\theta)^\dagger])$$

在具体的问题中，一般遇到以下情况：

- 考虑特定系统的基态能量，若系统的厄米哈密顿量矩阵为 H，则损失函数为

$$L(\theta) = \langle\psi|H|\psi\rangle$$

- 考虑特定生成的目标态，可以考虑内积损失函数

$$L(\theta) = \langle\psi|\psi\rangle$$

- 考虑有监督的量子机器学习问题，若数据集为 $\{(|\psi\rangle, y_i)\}$，损失函数可取为：

$$L(\theta) = \sum_i ||y_i - \langle\psi_i|U(\theta)^\dagger H U(\theta)|\psi_i\rangle||^2$$

2.2 Trotter 分解

在 n 个量子比特的系统里，哈密顿量 H 是一个 2^n 维的矩阵，因此在量子计算机上模拟哈密顿量的演化是一个非常困难的问题。

但是，如果哈密顿量有一个特殊的结构，量子计算机可以将其高效计算。设哈密顿量

$$H = \sum_{k=1}^{K} H_k$$

在 Trotter 一阶近似下，

$$e^{-iHt} = (\prod_{k=1}^{K} e^{-\frac{itH_k}{p}})^p + O(\frac{(mt)^2}{p})$$

用量子电路模拟 $U(\theta)$ 模拟 $(\prod_{k=1}^{K} e^{-\frac{itH_k}{p}})^p$ 需要 p 层,在每 j 层中用 $U_j(\theta_j) = \prod_{k=1}^{K} e^{-i\theta_{jk}H_k}$ 来模拟 $\prod_{k=1}^{K} e^{-\frac{itH_k}{p}}$，其中 θ_{jk} 为第 j 层的第 k 个变分系数。

$$U(\theta) = U_p(\theta_p)\ldots U_1(\theta_1)$$

显然，这是一种酉算子的嵌套，可以将其类比为深度学习的每个神经元的嵌套。

2.3 变分量子算法流程

Algorithm 1: 变分量子算法 (VQA) 过程

Input: 参数 θ 与参数化量子电路 $U(\theta)$，训练集 $\{\psi\}$，损失函数 $L(\theta)$。

Output: 问题解的估计；输出类型取决于具体任务。

初始化参数 θ，优化器及其参数;

while 未满足终止条件 **do**

  在量子计算机上执行 $U(\theta)$;

  测量输出以估算 $L(\theta)$ 或其梯度;

  在经典计算机上接收量子计算机的输出;

  使用优化器基于接收到的数据更新 θ 以最小化 $L(\theta)$;

end

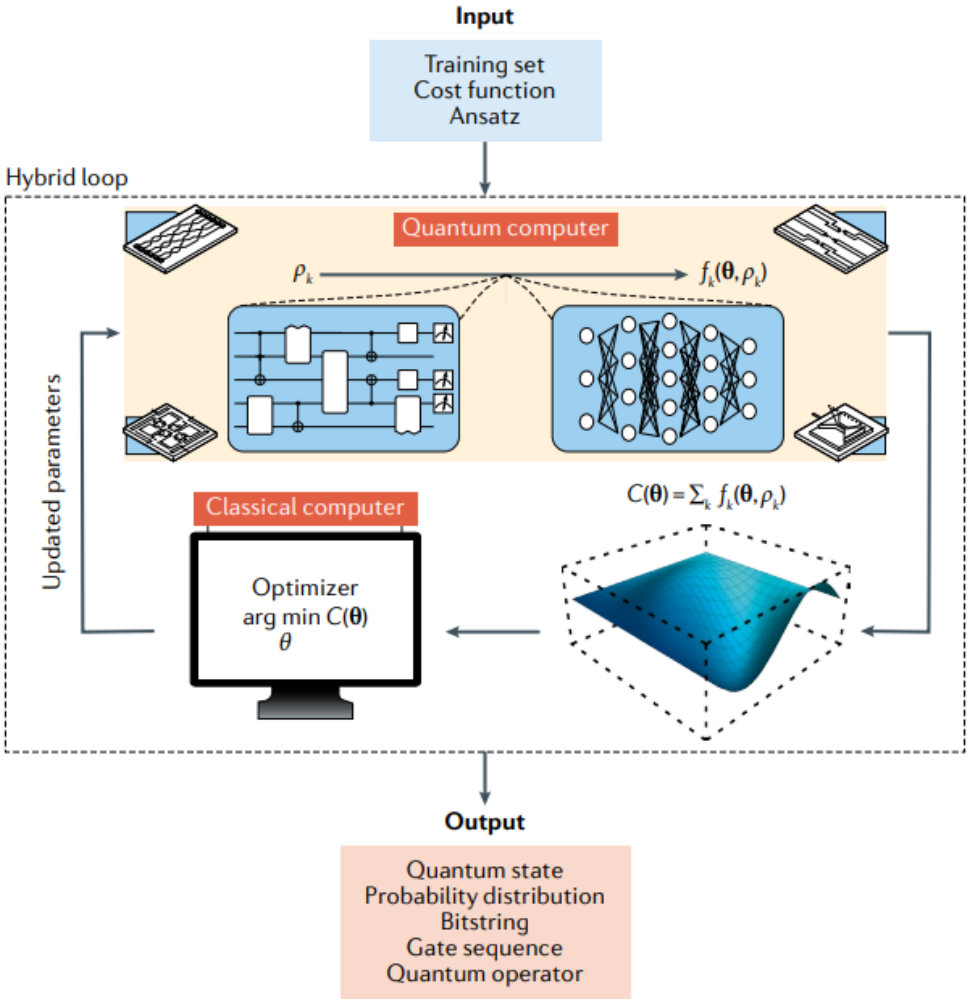

图 2: 变分量子算法的流程 [1]

3 变分量子算法奇异值分解 (VQSVD)

3.1 基本原理

给定复数矩阵 $M \in \mathbb{C}^{n\times n}$，矩阵 M 的奇异值分解为 $M = UDV^\dagger$，其中 U, V 是酉矩阵，D 是由奇异值组成的对角矩阵。不妨记为 $D = diag\{d_1, \ldots, d_r, 0, \ldots, 0\}$。

VQSVD [2] 的思路是通过训练两个参数化的量子线路 $U(\alpha)$ 和 $V(\beta)$ 来近似矩阵 M 的奇异值分解。

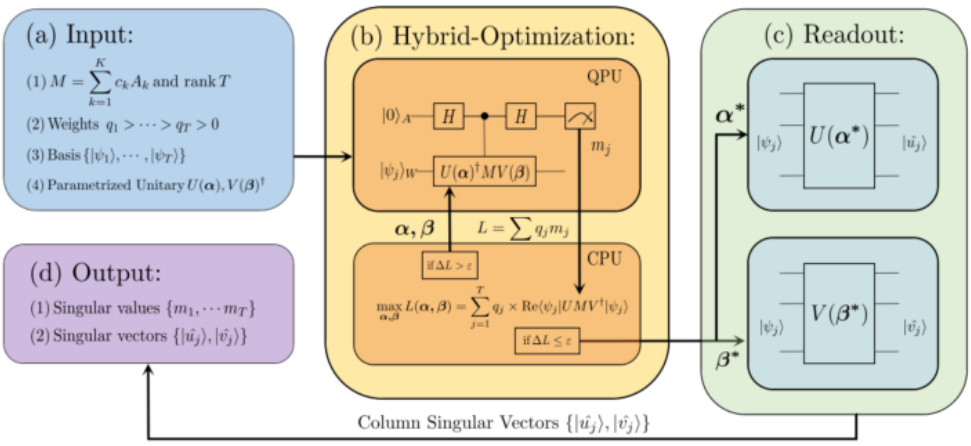

图 3: 变分量子奇异值分解 (VQSVD) [2]

易知，S 为纯态集合（标准化向量），最大的奇异值可以表示如下：

$$d_1 = \max_{|u\rangle,|v\rangle} \frac{\langle u|M|v\rangle}{\|u\|\|v\|} = \max_{|u\rangle,|v\rangle\in S} Re\langle u|M|v\rangle$$

其他的奇异值也可以表示如下：

$$\begin{aligned} d_k = &\max Re(\langle u|M|v\rangle) \\ &\text{s.t.} \quad |u\rangle, |v\rangle \in S, \\ &|u\rangle \perp \mathrm{span}\{|u_1\rangle, \ldots, |u_{k-1}\rangle\}, \\ &|v\rangle \perp \mathrm{span}\{|v_1\rangle, \ldots, |v_{k-1}\rangle\}. \end{aligned}$$

又有如下定理

$$\sum_{j=1}^{T} d_j = \max_{\substack{\{u_j\},\{v_j\} \\ \text{orthonormal}}} \sum_{j=1}^{T} \langle u_j|M|v_j\rangle.$$

那么我们的损失函数即可定义为

$$L(\alpha,\beta)=\sum_{j=1}^{T}q_j\langle\psi_j|U(\alpha)^\dagger MV(\beta)|\psi_j\rangle$$

其中 $q_1>\cdots>q_T>0$ 是实数权重，$\{\psi_j\}_{j=1}^T$ 为一组正交状态。对于给定的矩阵 M，该损失函数取最大值时，当且仅当对于每个 $1\le j\le T$，有

$$\langle\psi_j|U(\alpha)^\dagger MV(\beta)|\psi_j\rangle=d_j$$

即可得到我们需要的最大的 T 个奇异值。

在求解过程中我们需要计算参数的梯度，若 $U=U_{l_1}\dots U_1$, $V=V_{l_2}\dots V_1$，其中 $U_i=e^{-i\alpha_i H_i/2}$，$V_j=e^{-i\beta_j Q_j/2}$，则有公式

$$\frac{\partial L}{\partial\alpha_i}=\frac{1}{2}L(\alpha^{(i)},\beta)$$
$$\frac{\partial L}{\partial\beta_j}=\frac{1}{2}L(\alpha,\beta^{(j)})$$

其中，$\alpha^{(i)}=\alpha+\pi e_i$，$\beta^{(j)}=\beta-\pi e_j$，$e_i$ 和 e_j 是第 i 和第 j 个基向量。

3.2 算法流程

Algorithm 2: 变分量子奇异值分解 (VQSVD)

Input: 压缩因子 $\{c_k,A_k\}_{k=1}^K$，所需的秩 T，参数化电路 $U(\alpha)$ 和 $V(\beta)$ 以及初始参数 α,β，容忍度 ε

Output: 最大的 T 个奇异值，对应的酉算符 $U(\alpha')$ 和 $V(\beta')$（左右奇异向量 $\{|v_j(\alpha')\rangle\}$ 和 $\{|v_j(\beta')\rangle\}$）

1 准备正数 $\gamma_1>\gamma_2>\ldots>\gamma_T>0$;
2 选择计算基态 $\{|\psi_j\rangle\}_{j=1}^r$;
3 **for** $j=1$ **to** r **do**
4 　应用 $U(\alpha)$ 到状态 $\{|\psi_j\rangle\}$，得到 $\{|u_j\rangle=U(\alpha)|\psi_j\rangle\}$;
5 　应用 $V(\beta)$ 到状态 $\{|\psi_j\rangle\}$，得到 $\{|v_j\rangle=V(\beta)|\psi_j\rangle\}$;
6 　通过哈达马测试计算 $m_j=\mathrm{Re}\langle u_j|M|v_j\rangle$;
7 **end**
8 计算损失函数 $L(\alpha,\beta)=\sum_{j=1}^T\gamma_j m_j$;
9 执行优化以最大化 $L(\alpha,\beta)$，更新 α 和 β 的参数;
10 重复步骤 4-6，直到损失函数 $L(\alpha,\beta)$ 以容忍度 ε 收敛;

3.3 代码实现

我们利用华为 Mindspore 的 Mindquantum 库来实现 VQSVD 算法，下面是一个简单的例子：

```
import mindspore as ms
from mindquantum import Simulator, MQAnsatzOnlyLayer, add_prefix
from mindquantum import Hamiltonian, Circuit, RY, RZ, X

import numpy as np
from scipy.sparse import csr_matrix
from scipy.linalg import norm
from matplotlib import pyplot
import tqdm

#定义常数和设置权重
n_qubits = 3  # 量子比特数
cir_depth = 20  # 电路深度
N = 2**n_qubits
rank = 8  # 秩
step = 3
ITR = 200  # 迭代步数
LR = 0.02  # 学习率

#设置学习权重
if step == 0:
    weight = ms.Tensor(np.ones(rank))
else:
    weight = ms.Tensor(np.arange(rank * step, 0, -step))

#随机生成8*8的复数矩阵
np.random.seed(42)


def mat_generator():
    matrix = np.random.randint(
        10, size=(N, N)) + 1j * np.random.randint(10, size=(N, N))
    return matrix
```

```
M = mat_generator()
m_copy = np.copy(M)

print('Random matrix M is: ')
print(M)

#通过传统方法进行奇异值分解
U, D, v_dagger = np.linalg.svd(M, full_matrices=True)
```

矩阵 M 如下

```
Random matrix M is:
[[6.+1.j 3.+9.j 7.+3.j 4.+7.j 6.+6.j 9.+8.j 2.+7.j 6.+4.j]
    [7.+1.j 4.+4.j 3.+7.j 7.+9.j 7.+8.j 2.+8.j 5.+0.j 4.+8.j]
    [1.+6.j 7.+8.j 5.+7.j 1.+0.j 4.+7.j 0.+7.j 9.+2.j 5.+0.j]
    [8.+7.j 0.+2.j 9.+2.j 2.+0.j 6.+4.j 3.+9.j 8.+6.j 2.+9.j]
    [4.+8.j 2.+6.j 6.+8.j 4.+7.j 8.+1.j 6.+0.j 1.+6.j 3.+6.j]
    [8.+7.j 1.+4.j 9.+2.j 8.+7.j 9.+5.j 4.+2.j 1.+0.j 3.+2.j]
    [6.+4.j 7.+2.j 2.+0.j 0.+4.j 3.+9.j 1.+6.j 7.+6.j 3.+8.j]
    [1.+9.j 5.+9.j 5.+2.j 9.+6.j 3.+0.j 5.+3.j 1.+3.j 9.+4.j]]
```

定义 Ansatz 类、量子网络层以及其他需要的函数

```
class Ansatz:
'''
定义ansatz
'''

def __init__(self, n, depth):
    self.circ = Circuit()
    num = 0
    for _ in range(depth):

        for i in range(n):
            self.circ += RY('theta' + str(num)).on(i)
            num += 1

        for i in range(n):
            self.circ += RZ('theta' + str(num)).on(i)
            num += 1
```

```
18
19          for i in range(n - 1):
20              self.circ += X.on(i + 1, i)
21
22          self.circ += X.on(0, n - 1)
23
24  def loss_plot(loss):
25      '''
26      画出学习曲线
27      '''
28      pyplot.plot(list(range(1, len(loss) + 1)), loss)
29      pyplot.xlabel('iteration')
30      pyplot.ylabel('loss')
31      pyplot.title('Loss Over Iteration')
32      pyplot.suptitle('step = ' + str(step))
33      pyplot.show()
34
35  def quantnet(qubits_num, hams, circ_right, circ_left=None, base=None):
36      '''
37      生成量子网络
38      '''
39      sim = Simulator('mqvector', qubits_num)
40
41      if base is None:
42          pass
43      else:
44          sim.set_qs(base)
45      grad_ops = sim.get_expectation_with_grad(hams, circ_right, circ_left)
46
47      ms.context.set_context(mode=ms.context.PYNATIVE_MODE, device_target="CPU")
48
49      quantumnet = MQAnsatzOnlyLayer(grad_ops, 'ones')
50
51      return quantumnet
```

实例化 Ansatz 类，量子网络层以及定义损失函数

```
1  u_ansatz = add_prefix(Ansatz(n_qubits, cir_depth).circ, 'u')
2  v_ansatz = add_prefix(Ansatz(n_qubits, cir_depth).circ, 'v')
```

```
v_ansatz.svg()
ham = Hamiltonian(csr_matrix(M))

i_matrix = np.identity(N)
quantum_models = dict()
quantum_models['net_0'] = quantnet(n_qubits, ham, v_ansatz, u_ansatz,
                                   i_matrix[0])
for s in range(1, rank):
    quantum_models["net_" + str(s)] = quantnet(n_qubits, ham, v_ansatz,
                                               u_ansatz, i_matrix[s])
    quantum_models["net_" + str(s)].weight = quantum_models['net_0'].weight
class MyNet(ms.nn.Cell):
    '''
    生成经典量子网络
    '''

    def __init__(self):
        super(MyNet, self).__init__()

        self.build_block = ms.nn.CellList()
        for j in range(rank):
            self.build_block.append(quantum_models["net_" + str(j)])

    def construct(self):
        x = self.build_block[0]() * weight[0]
        k = 1

        for layer in self.build_block[1:]:
            x += layer() * weight[k]
            k += 1

        return -x

net = MyNet()

opt = ms.nn.Adam(net.trainable_params(), learning_rate=LR)

```

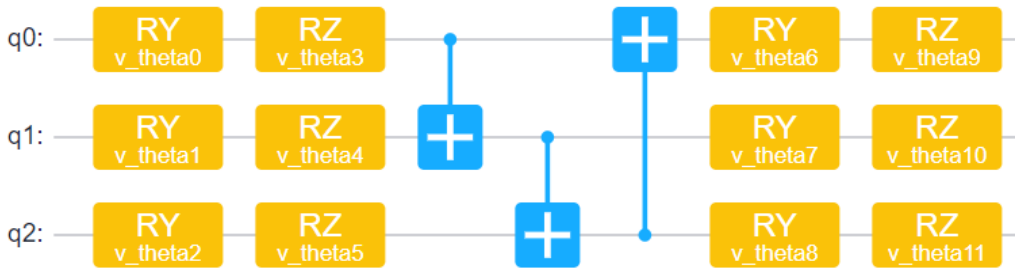

图 4: V 的部分 Ansatz

```
train_net = ms.nn.TrainOneStepCell(net, opt)

loss_list = list()
for itr in tqdm.tqdm(range(ITR)):
    res = train_net()
    loss_list.append(res.asnumpy().tolist())
```

V 的部分 Ansatz 示例：读取训练结果与学习曲线，与传统算法解进行对比

```
singular_value = list()

for _, qnet in quantum_models.items():
    singular_value.append(qnet().asnumpy()[0])

loss_plot(loss_list)
print('预测的奇异值:', singular_value)
print("真实的奇异值:", D)
```

损失函数的学习曲线如下：

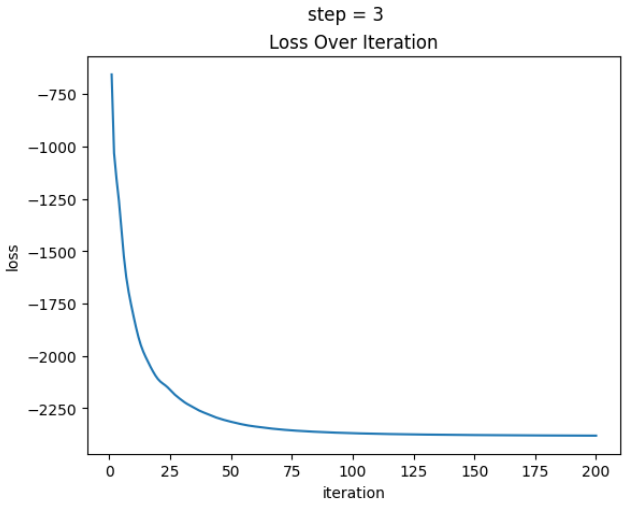

图 5: 损失函数的学习曲线

结果对比如下：

```
预测的奇异值: [54.83174, 19.169168, 14.88653, 11.093878, 10.533753, 7.648352,
    5.5560594, -0.3320913]
真实的奇异值: [54.83484985 19.18141073 14.98866247 11.61419557 10.15927045
    7.60223249
5.81040539  3.30116001]
```

可以得到前几个相差不大，但是后面的奇异值相差较大，这需要更好地调整参数以使得能够更好地逼近较小的奇异值。

若使用该方法进行矩阵压缩，该方法和经典方法误差相差不大

```
value = quantum_models['net_0'].weight.asnumpy()
v_value = value[:120]
u_value = value[120:]

# Calculate U and V
u_learned = u_ansatz.matrix(u_value)
v_learned = v_ansatz.matrix(v_value)

v_dagger_learned = np.conj(v_learned.T)
d_learned = np.array(singular_value)
err_subfull, err_local, err_svd = [], [], []
U, D, v_dagger = np.linalg.svd(M, full_matrices=True)

```

```
# Calculate Frobenius-norm error
for t in range(rank):
    lowrank_mat = np.matrix(U[:, :t]) * np.diag(D[:t]) * np.matrix(
        v_dagger[:t, :])
    recons_mat = np.matrix(u_learned[:, :t]) * np.diag(
        d_learned[:t]) * np.matrix(v_dagger_learned[:t, :])
    err_local.append(norm(lowrank_mat - recons_mat))
    err_subfull.append(norm(m_copy - recons_mat))
    err_svd.append(norm(m_copy - lowrank_mat))

# Plot SVD error and VQSVD error
fig, ax = pyplot.subplots()
ax.plot(list(range(1, rank + 1)),
        err_subfull,
        "o-.",
        label='Reconstruction via VQSVD')
ax.plot(list(range(1, rank + 1)),
        err_svd,
        "^--",
        label='Reconstruction via SVD')
# ax.plot(list(range(1, rank + 1)), err_local, "*--", label='SVD V/S QSVD')
pyplot.xlabel('Singular Value Used (Rank)', fontsize=14)
pyplot.ylabel('Norm Distance', fontsize=14)
leg = pyplot.legend(frameon=True)
leg.get_frame().set_edgecolor('k')
```

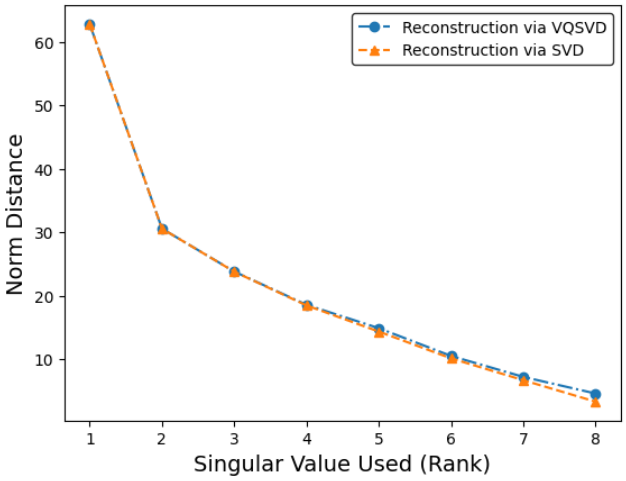

图 6: SVD 误差和 VQSVD 压缩误差

可知，量子神经网络分解出的奇异值和传统奇异值分解的奇异值相差不大，因此可以用于数据主成分分析或用于矩阵压缩。

4 量子神经网络 (QNN)

4.1 基本原理

当我们有一个量子神经网络模型时，我们希望将其应用于分类任务。在大多数情况下，我们首先需要将任务形式化为一个优化问题。如何将分类任务适配到基于优化的 QNN 模型可以通过以下简单示例来说明：

在数据准备和数据编码过程之后，QNN 之后的输出状态经过最终的测量 M。对于分类任务，出于简化考虑，我们可以考虑一个二元分类任务，其中两种分类标签分别为 “cat” 和 “dog”，并假设最终测量是在某个特定比特的计算基础上进行的。如果输入是 “cat”，我们的目标是最大化测量到 σ_z 输出 “1” 的概率，即最大化 $P(0)$。相反，如果输入是 “dog”，我们的目标是最大化测量到 σ_z 输出 “-1” 的概率，即最大化 $P(1)$。在预测阶段，我们简单地比较不同测量结果的概率，并将标签分配给对应的最高概率。

为了实现这一目标，我们需要优化可训练参数以获得理想的预测。首先，我们通常需要定义一个成本函数来衡量当前输出与目标输出之间的距离。常用的成本函数包括均方误差（MSE）和交叉熵（CE）：

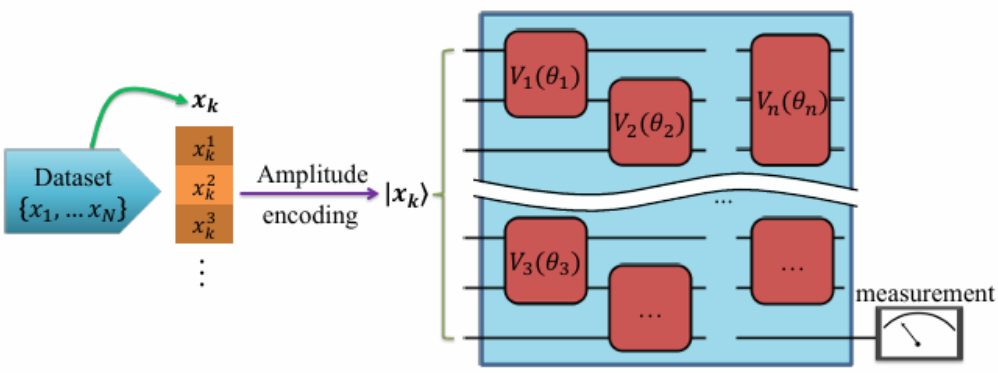

图 7: 基于幅度编码的量子神经网络 [3]

$$L_{MSE}(h(\xi;\theta\ ,a) = \sum_k (\alpha_k - \beta_k)^2,$$

$$L_{CE}(h(\xi;\theta), a) = -\sum_k \alpha_k \log \beta_k,$$

其中 $a = (a_1, \ldots, a_m)$ 是输入数据 $\hat{\xi}$ 的标签，以 one-hot 编码的形式给出，h 表示由量子神经网络（参数统一用 Θ 表示）确定的假设函数，$g = (g_1, \ldots, g_m) = \mathrm{diag}(P_{\mathrm{out}})$ 展示了输出状态 P_{out} 中相应输出类别的所有概率。这里，g 可以测量一些量子位来获得。在我们专注于二元分类的教程中，我们选择在 Z-基上重复测量一个量子位以评估 $g = (g_1, g_2)$，其中这个量子位的索引可以灵活选择。

对于幅度编码的量子神经网络 [3]，通过给出输入状态 $|psi_x\rangle$，一个 QNN 电路 $U(\Theta)$，和一个观测值 O_k，概率函数可以写为

$$g_k = \langle\psi_x|U(\Theta)^\dagger O_k U(\Theta)|\psi_x\rangle$$

4.2 算法流程

Algorithm 3: 基于幅度编码的量子神经网络分类器

Input: 未训练的模型 h，变分参数 Θ，损失函数 L，训练集 $\{(|x_m\rangle, a_m)\}_{m=1}^{n}$，批量大小 n_b，迭代次数 T，学习率 e，Adam 优化器 f_{Adam}

Output: 训练后的模型

初始化：为 Θ 生成随机初始参数;

for $i \leftarrow 1$ **to** T **do**

  随机选择 n_b 个样本 $\{|x_{i,1}\rangle, |x_{i,2}\rangle, \ldots, |x_{i,n_b}\rangle\}$ 从训练集中;

  计算损失函数 L 相对于参数 Θ 的梯度，并对训练批次取平均值 $G \leftarrow \frac{1}{n_b}\sum_{k=1}^{n_b} \nabla L(h(|x_{i,k}\rangle), \Theta); a_{i,k});$

  更新参数：$\Theta \leftarrow f_{\text{Adam}}(\Theta, e, G);$

end

输出训练后的模型;

4.3 代码实现

我们使用 FashionMNIST 数据集来进行量子神经网络的分类任务，进行靴子与 T 恤的二分类任务。

```
import h5py
import scipy.io
import numpy as np

train_num = 1000
test_num = 200

dataset = h5py.File('./Dataset/FashionMNIST_1_2_wk.mat')
train_data = np.transpose(dataset['x_train'])
train_label = np.transpose(dataset['y_train'])
test_data = np.transpose(dataset['x_test'])
test_label = np.transpose(dataset['y_test'])

train_pixels = np.array(train_data[:,:train_num].tolist())[:,:,0].transpose() # [:,:,0]取实部
test_pixels = np.array(test_data[:,:test_num].tolist())[:,:,0].transpose()
train_index = train_label[:train_num,0].astype(int) # 0-> 靴子 1->T恤
```

```
test_index = test_label[:test_num,0].astype(int)
import matplotlib.pyplot as plt
from matplotlib import gridspec
plt.matshow(np.reshape(train_pixels[0,:],[16,16]))
plt.matshow(np.reshape(train_pixels[1,:],[16,16]))
print(f'前2个训练集标签为 {train_index[:2]}')
```

Listing 1: Python example

目前训练数据为像素数据，不能直接作为输入态进入量子电路，我们需要用幅度编码将像素数据转换为线路参量

```
from mindquantum.algorithm.library import amplitude_encoder
def amplitude_param(pixels):
    param_rd = []
    _, parameterResolver = amplitude_encoder(pixels, 8)
    for _, param in parameterResolver.items():
        param_rd.append(param)
    param_rd = np.array(param_rd)
    return param_rd

# 将幅度转为编码线路参数, 幅度shape(256,), 参数shape(255,)
train_param = np.array([amplitude_param(i) for i in train_pixels ])
test_param = np.array([amplitude_param(i) for i in test_pixels ])

import mindspore as ms
BATCH_SIZE = 100
train_loader = ms.dataset.NumpySlicesDataset(
    {'features': train_param, 'labels': train_index}, shuffle=True).batch(
   BATCH_SIZE)
test_loader = ms.dataset.NumpySlicesDataset(
    {'features': test_param, 'labels': test_index}).batch(BATCH_SIZE)
```

定义含参 Ansatz 线路

```
from mindquantum.core.circuit import Circuit
import mindquantum.core.gates as Gate

def Classifying_circuit(qubit_num, block_num):
    num = qubit_num
```

```
    depth = block_num
    circ = Circuit()
    for i in range(depth):
        circ = Para_circuit(circ, num)
        for i in range(0,qubit_num-1,2):
            circ += Gate.Z.on(i+1,i)
        for i in range(1,qubit_num-2,2):
            circ += Gate.Z.on(i+1,i)
    return circ

def Para_circuit(circuit,qubit_num):
    for i in range(qubit_num):
        circuit += Gate.RX(f'Xtheta{i}').on(i)
        circuit += Gate.RZ(f'Ztheta{i}').on(i)
        circuit += Gate.RX(f'Xtheta2{i}').on(i)
    return circuit

QUBIT_NUM = 8
BLOCK_NUM = 2
ansatz = Classifying_circuit(QUBIT_NUM,BLOCK_NUM).as_ansatz()
ansatz.svg()
```

含参 Ansatz 线路如下：

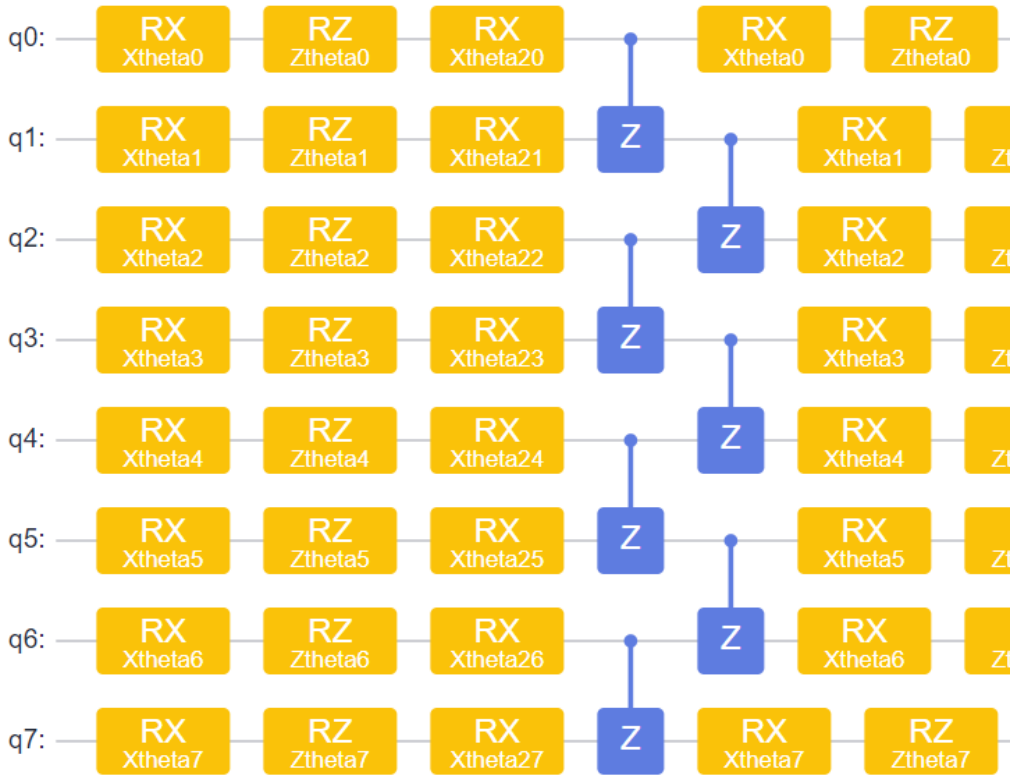

图 8: 含参 Ansatz 线路部分图

传入初态以及通过 Pauli-Z 测量来进行二分类任务

```
encoder = amplitude_encoder([0], QUBIT_NUM)[0].as_encoder()
from mindquantum.framework import MQLayer
from mindquantum.core.operators import Hamiltonian, QubitOperator
from mindquantum.simulator import Simulator
import mindspore as ms
WORKER = 4
ms.set_context(mode=ms.PYNATIVE_MODE, device_target="CPU")
ms.set_seed(1)
circ = encoder.as_encoder() + ansatz
meas = [Hamiltonian(QubitOperator(f'Z{i}')) for i in [QUBIT_NUM-2, QUBIT_NUM
    -1]]
sim = Simulator('mqvector', circ.n_qubits)
grad_ops = sim.get_expectation_with_grad(meas, circ, parallel_worker=WORKER)
Qnet = MQLayer(grad_ops)
```

进行模型训练

```
class ForwardAndLoss(ms.nn.Cell):
```

```
def __init__(self, backbone, loss_fn):
    super(ForwardAndLoss, self).__init__(auto_prefix=False)
    self.backbone = backbone
    self.loss_fn = loss_fn

def construct(self, data, label):
    output = self.backbone(data)
    return self.loss_fn(output, label)

def backbone_network(self):
    return self.backbone


class TrainOneStep(ms.nn.TrainOneStepCell):

def __init__(self, network, optimizer):
    super(TrainOneStep, self).__init__(network, optimizer)
    self.grad = ms.ops.GradOperation(get_by_list=True)

def construct(self, data, label):
    weights = self.weights
    loss = self.network(data, label)
    grads = self.grad(self.network, weights)(data, label)
    return loss, self.optimizer(grads)

LR = 0.05
loss = ms.nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
opt = ms.nn.Adam(Qnet.trainable_params(), learning_rate=LR)
net_with_loss = ForwardAndLoss(Qnet, loss)
train_one_step = TrainOneStep(net_with_loss, opt)

class EpochLoss(ms.nn.Metric):
    def __init__(self):
        super(EpochLoss, self).__init__()
        self.clear()

    def clear(self):
```

```
        self.loss = 0
        self.counter = 0

    def update(self, *loss):
        loss = loss[0].asnumpy()
        self.loss += loss
        self.counter += 1

    def eval(self):
        return self.loss / self.counter

class EpochAcc(ms.nn.Metric):
    def __init__(self):
        super(EpochAcc, self).__init__()
        self.clear()

    def clear(self):
        self.correct_num = 0
        self.total_num = 0

    def update(self, *inputs):
        y_output = inputs[0].asnumpy()
        y = inputs[1].asnumpy()
        y_pred = np.zeros_like(y)
        for i in range(y_pred.shape[0]):
            yi = y_output[i]
            if yi[0] >= yi[1]:
                y_pred[i] = 0
            else:
                y_pred[i] = 1
        self.correct_num += np.sum(y == y_pred)
        self.total_num += y.shape[0]

    def eval(self):
        return self.correct_num / self.total_num

acc_epoch = EpochAcc()
```

```
loss_epoch = EpochLoss()

train_loss_epoch = []
train_acc_epoch = []
test_loss_epoch = []
test_acc_epoch = []

STEP_NUM = 30

for epoch in range(STEP_NUM):
    loss_epoch.clear()
    acc_epoch.clear()
    loss_epoch.clear()
    acc_epoch.clear()

    for data in train_loader:
        train_one_step(data[0], data[1])   # 执行训练，并更新权重，data[0]参数，data[1]为标签
        loss = net_with_loss(data[0], data[1])
        loss_epoch.update(loss)
    train_loss = loss_epoch.eval()
    train_loss_epoch.append(train_loss)

    # training accuracy
    for data in train_loader:
        logits = Qnet(data[0]) # 向前传播得到预测值
        acc_epoch.update(logits, data[1]) # 计算预测准确率
    train_acc = acc_epoch.eval()
    train_acc_epoch.append(train_acc)

    # testing loss
    for data in test_loader:
        loss = net_with_loss(data[0], data[1])  # 计算损失值
        loss_epoch.update(loss)
    test_loss = loss_epoch.eval()
    test_loss_epoch.append(test_loss)

```

```
    # testing accuracy
    for data in test_loader:
        logits = Qnet(data[0])
        acc_epoch.update(logits, data[1])
    test_acc = acc_epoch.eval()
    test_acc_epoch.append(test_acc)

    print(f"epoch: {epoch+1}, training loss: {train_loss}, training acc: {
    train_acc}, testing loss: {test_loss}, testing acc: {test_acc}")
```

可以看到，量子神经网络在该数据集上进行二分类任务表现不错

```
epoch: 30, training loss: 0.4774589389562607, training acc: 0.958, testing loss
    : 0.4789623940984408, testing acc: 0.9575
```

5 总结

本文总结了变分量子算法（VQAs）在量子计算领域的应用前景和面临的挑战。VQAs 作为一种有潜力在 NISQ 设备上实现量子优势的策略，已经吸引了广泛的研究兴趣。通过变分量子算法奇异值分解（VQSVD）的实例，展示了 VQAs 在解决特定问题上的应用方法。此外，文章还探讨了量子机器学习在分类任务中的应用，说明了如何将传统机器学习任务转化为量子优化问题，并使用量子神经网络进行有效训练。尽管 VQAs 在实际应用中仍存在一些挑战，但它们的发展为量子计算的实际应用提供了新的思路和方法。随着量子硬件和算法的不断进步，VQAs 有望在未来量子技术的发展中扮演重要角色。

参考文献

[1] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

[2] Xin Wang, Zhixin Song, and Youle Wang. Variational quantum singular value decomposition. *Quantum*, 5:483, 2021.

[3] Weikang Li, Zhide Lu, and Dong-Ling Deng. Quantum Neural Network Classifiers: A Tutorial. *SciPost Phys. Lect. Notes*, page 61, 2022.