题目 2 说明文档

方案设计

我们设计的混合量子经典神经网络是**以经典卷积神经网络（CNN）为基础架构，并在此基础上启发式地融入了变分量子线路（VQC）。**

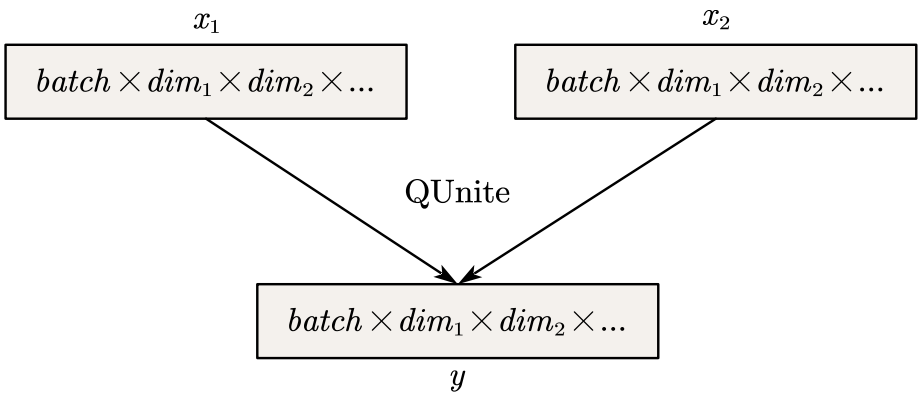
本题重在探索量子部分在图像领域的建模设计方案。然而，我们却在用经典 CPU 模拟量子芯片的行为，不是真正在量子芯片上运行程序，这致使我们运行较为参数较多的模型会极为缓慢（VQC 的 forward 和 backward 所致），难以进行机器学习。因此，我们最终给出的神经网络参数量很少（只有 1000 多个）。尽管如此，这充分展现了我们的想法。如果有需要，完全可以在此基础上推广（丰富网络结构和提高参数量），并提交到真正的量子芯片上运行。

我们的核心目的是想**利用量子计算的特性来增强经典神经网络的拟合能力**。

在 Quanvolution 进行图像分类的例子中，经典数据通过编码到量子卷积层最终变为量子线路的观测值，实现了**经典数据向“量子数据”的转化**。因为量子的各种特性，Quanvolution 可以生成高度复杂的内核，其计算至少在原则上是经典难处理的，所以“量子数据”相比于经典数据更加丰富，增强了整个神经网络的拟合能力。我们开始也想往这个方向做，也就是设计线路将经典数据映射到“量子数据”，以更加丰富抓取到的图片特征。但遗憾的是，直接对经典数据进行量子卷积，也就是 QConv，由于参数量较大，在 CPU 上模拟运行实在是太慢了，就算是调用 VQNET 库，也无济于事。另外，我们试图设计类似于经典全连接网络的量子变分全连接网络，以实现经典数据向“量子数据”的转化，但是由于其参数量不可避免地大，在 CPU 模拟的条件下我们甚至难以跑完一个 epoch，所以无从验证想法。我们最终抛弃了这些思路。

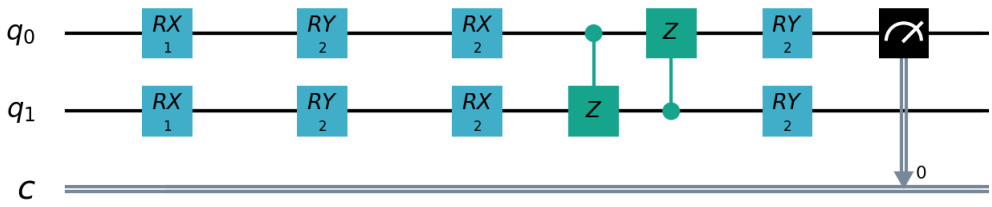
量子方面的可变参数量大小是 CPU 模拟的一大问题。如果我们目前想要设计混合量子经典神经网络，就不得不减少量子比特数、线路深度、总参数规模等，*这似乎与赛题的一些要求不谋而合了*。最终我们想到了这种思路：既然不能使用大规模的量子线路，我们就**设计简单的量子线路，并重复利用它来实现数据转化**。既然不能很好地实现前述的经典数据向“量子数据”的转变，我们就实现**经典数据向“半经典半量子”数据的转化**。所谓“半经典半量子”，就是前面说的重复利用简单量子线路对经典数据处理后的结果——因为量子参数很少，所以线路输出不能完全是量子成分的（因为变分参数都远远少于输出值了）。

最终引出我们设计的神经网络中的核心结构，我们把它称为“经典数据的量子合并层”，简称“合并层”（QUnite）。用这样的简图来表示：



其中 x_1, x_2 为合并层的输入， y 为输出。 x_1, x_2, y 具有完全相同的数据维度， y 的每一个元素都是 x_1, x_2 对应位置上元素经过同一个特定（变分）量子线路后的测量值。由于用的是同一个量子线路，名义上 y 只能说是 x_1, x_2 掺入了少量量子成分的组合。但是，我们要意识到，这种量子特性的掺和（本质上是一种经典难以做到的非线性成分）对神经网络整体的拟合能力的提升可能是不可估量的！

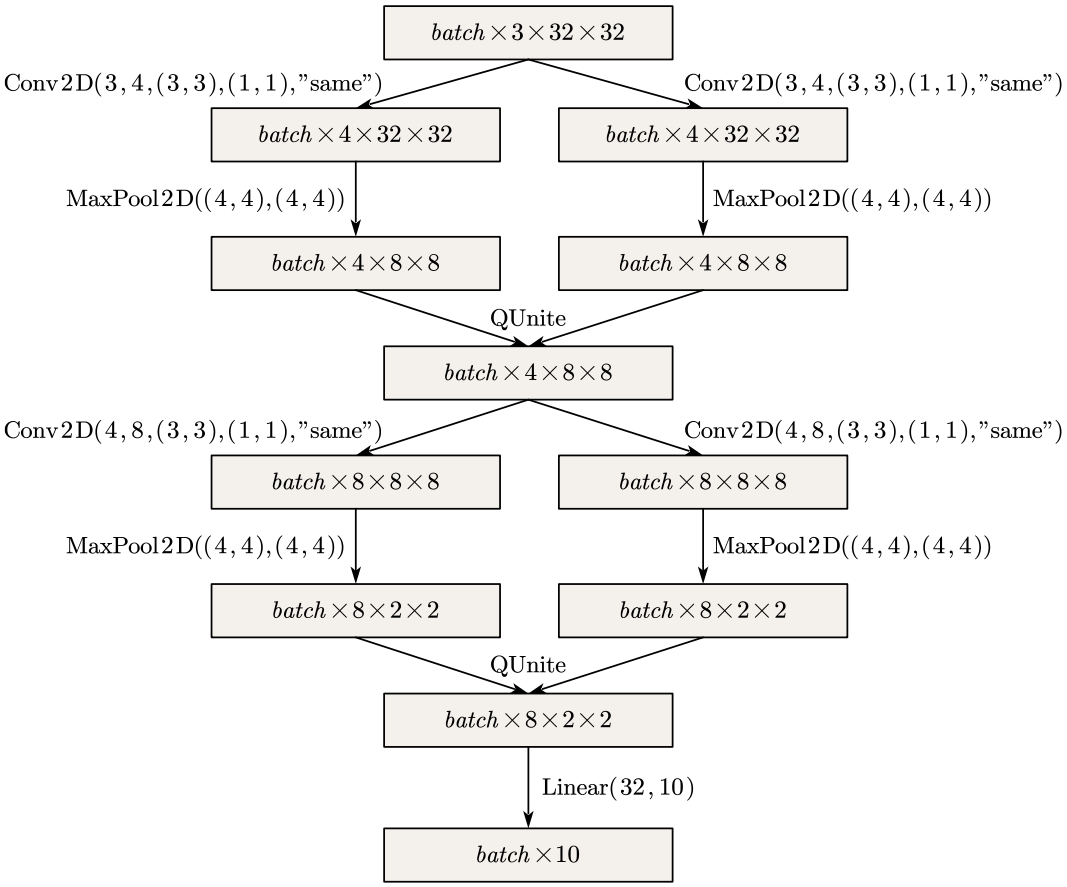
对于上面说到的简单量子线路，我们参考了 VSQL 模型的例子，给出以下线路，目的是尽可能让经典数据两两纠缠在一起：



其中参数1代表要编码经典数据到量子门上的位置，参数 2 代表变分参数。最终要得到的是量子比特 q_0 哈密顿量 Z 的期望值（也就是布洛赫球中的 $\cos\theta$ 值）。

当然，这个量子线路可以设计得很复杂，但是因为同样的原因：我们难以用 CPU 模拟它们。

除了这个合并层，我们设计的神经网络就只剩下经典 CNN 了，可以类似看做卷积层的“集成学习”。原本的 CNN 是单线的结构，即对数据进行卷积、池化一层一层单向地流下，最终全连接输出结果。我们设计的神经网络在**卷积部分有分支**，即进行两个参数不同但结构相同的卷积运算，接着做池化运算，最后经过合并层合并为单份数据，继续往下运算。整体结构如下（以 CIFAR-10 数据集为例）：



共经过了两层卷积运算和两次合并运算。

该神经网络的结构由 VQNET 相关函数可以清晰展现：

```
####################Model Summary#########################

classic layers: {'Conv2D': 4, 'MaxPool2D': 4, 'Linear': 1}
total classic parameters: 1146

========================================
qubits num: 2
gates: {'RX': 8, 'RY': 8, 'CZ': 4}
total quantum gates: 20
total quantum parameter gates: 16
total quantum parameters: 16
############################################################
```

参数量可以说已经太少太少了，相比于一般的深度学习模型。在这么少的参数下，我们很期待它的学习效果。

代码说明

utils.py 是数据处理的相关代码，包含数据解压，数据加载，数据增强等。其中数据增强 enhance()其实就是简单把每张图片水平镜像了一下，使训练集数据量从 50000 变为了 100000。

```
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import random


### 数据集
class Dataset:
    @staticmethod
    def unpickle(file):
        import pickle

        with open(file, "rb") as fo:
            dict = pickle.load(fo, encoding="bytes")
        return dict

    @staticmethod
    def getlabel(n):
        return Dataset.label_names[n].decode("utf-8")

    def __init__(self, *path):
        a = self.unpickle(path[0])
        self.x = a[b"data"].reshape((-1, 3, 32, 32))
        self.y = a[b"labels"]
        for i in range(1, len(path)):
            a = self.unpickle(path[i])
            self.x = np.concatenate((self.x, a[b"data"].reshape((-1, 3,
32, 32))))
            self.y = np.concatenate((self.y, a[b"labels"]))
        self.y = np.array(self.y, dtype="int64")
        self.len = self.y.shape[0]

    def getdata(self, index):
        return self.x[index] / 255, self.y[index]

    def getdatas(self):
```

```
        return self.x / 255, self.y

    def getimg(self, index):
        return self.x[index].transpose((1, 2, 0)), self.y[index]

    def showimg(self, index):
        img, label = self.getimg(index)
        plt.imshow(img)
        plt.axis("off")
        plt.show()
        return img, label

    def shuffle(self):
        r = np.random.permutation(self.len)
        x = self.x.copy()
        y = self.y.copy()
        for i in range(self.len):
            self.x[i] = x[r[i]]
            self.y[i] = y[r[i]]

    def enhance(self):
        self.x = np.concatenate((self.x, self.x))
        self.y = np.concatenate((self.y, self.y))
        for i in range(self.len):
            self.x[i] = self.x[i, :, :, ::-1]
        self.len *= 2
        self.shuffle()


Dataset.label_names = Dataset.unpickle("data/cifar-10-batches-
py/batches.meta")[
    b"label_names"
]

if __name__ == "__main__":
    import tarfile
    with tarfile.open("data/cifar-10-python.tar.gz","r") as f:
        f.extractall("data/")
```

model.py 是模型的相关代码。其中 class QM 是上面说过的简单量子线路，用到的是自动微分模拟。class Unite 是合并层。class Model 是神经网络的整体。

```python
import numpy as np
import pyqpanda as pq
import pyvqnet as pv
from functools import partial


class QM(pv.qnn.vqc.QModule):
    def __init__(self, name=""):
        super().__init__(name)
        self.device = pv.qnn.vqc.QMachine(2)
        self.RX1 = pv.qnn.vqc.qcircuit.RX(wires=0)
        self.RX2 = pv.qnn.vqc.qcircuit.RX(wires=1)
        self.RY1 = pv.qnn.vqc.qcircuit.RY(True, True, 0, wires=0)
        self.RY2 = pv.qnn.vqc.qcircuit.RY(True, True, 0, wires=1)
        self.RX3 = pv.qnn.vqc.qcircuit.RX(True, True, 0, wires=0)
        self.RX4 = pv.qnn.vqc.qcircuit.RX(True, True, 0, wires=1)
        self.CZ1 = pv.qnn.vqc.qcircuit.CZ(wires=(0, 1))
        self.CZ2 = pv.qnn.vqc.qcircuit.CZ(wires=(1, 0))
        self.RY3 = pv.qnn.vqc.qcircuit.RY(True, True, 0, wires=0)
        self.RY4 = pv.qnn.vqc.qcircuit.RY(True, True, 0, wires=1)
        self.measure = pv.qnn.vqc.MeasureAll(
            obs=[
                {
                    "wires": [0],
                    "observables": ["Z"],
                    "coefficient": [1],
                }
            ]
        )

    # @partial(pv.qnn.vqc.wrapper_compile)
    def forward(self, x, y):
        self.device.reset_states(x.shape[0])
        self.RX1(q_machine=self.device, params=x[:])
        self.RX2(q_machine=self.device, params=y[:])
        self.RY1(q_machine=self.device)
        self.RY2(q_machine=self.device)
        self.RX3(q_machine=self.device)
        self.RX4(q_machine=self.device)
        self.CZ1(q_machine=self.device)
        self.CZ2(q_machine=self.device)
        self.RY3(q_machine=self.device)
        self.RY4(q_machine=self.device)

        return self.measure(q_machine=self.device)
```

```
class QUnite(pv.nn.module.Module):
    def __init__(self, name=""):
        super().__init__(name)
        self.vqc = QM(name)

    def forward(self, x, y):
        X = pv.tensor.flatten(x, 1)
        Y = pv.tensor.flatten(y, 1)
        Z = pv.tensor.zeros_like(X)
        for i in range(X.shape[1]):
            Z[:, i] = self.vqc(X[:, i], Y[:, i])
        z = pv.tensor.reshape(Z, x.shape)
        return z


class Model(pv.nn.module.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = pv.nn.Conv2D(3, 4, (3, 3), (1, 1), "same")
        self.pool1 = pv.nn.MaxPool2D((4, 4), (4, 4))
        self.conv2 = pv.nn.Conv2D(3, 4, (3, 3), (1, 1), "same")
        self.pool2 = pv.nn.MaxPool2D((4, 4), (4, 4))
        self.vqc1 = QUnite("MyVQC1")
        self.conv3 = pv.nn.Conv2D(4, 8, (3, 3), (1, 1), "same")
        self.pool3 = pv.nn.MaxPool2D((4, 4), (4, 4))
        self.conv4 = pv.nn.Conv2D(4, 8, (3, 3), (1, 1), "same")
        self.pool4 = pv.nn.MaxPool2D((4, 4), (4, 4))
        self.vqc2 = QUnite("MyVQC2")
        self.fc = pv.nn.Linear(8 * 2 * 2, 10)

    def forward(self, x):
        x1 = 2 * pv.tensor.atan(self.pool1(self.conv1(x)))
        x2 = 2 * pv.tensor.atan(self.pool2(self.conv2(x)))
        x = self.vqc1(x1, x2)
        x1 = 2 * pv.tensor.atan(self.pool3(self.conv3(x)))
        x2 = 2 * pv.tensor.atan(self.pool4(self.conv4(x)))
        x = self.vqc2(x1, x2)
        x = pv.tensor.flatten(x, 1)
        x = self.fc(x)

        return x
```

train.py 对训练集数据训练，采用留出法，90000 个数据用作训练，10000 个用作跑完每一轮进行检验。设定 epoch=1000，batch=32（实际上跑完 1 个 epoch 要大约 4 小时，根本跑不了太多 epoch，只是设了一个值让它尽可能一直跑），使用交叉熵随损失函数和 Adam 优化器。（后期改成了 batch=128 和 SGD 优化器）训练时定时输出局部的详细信息。最后展示训练过程中的准确率变化曲线。

```
from utils import *
from model import *
import numpy as np
import matplotlib.pyplot as plt
import pyqpanda as pq
import pyvqnet as pv
import random
from functools import partial
import time


train_data = Dataset(
    "data/cifar-10-batches-py/data_batch_1",
    "data/cifar-10-batches-py/data_batch_2",
    "data/cifar-10-batches-py/data_batch_3",
    "data/cifar-10-batches-py/data_batch_4",
    "data/cifar-10-batches-py/data_batch_5",
)
train_data.enhance()


model = Model()
#
model.load_state_dict(pv.utils.storage.load_parameters("train.model"))
print(pv.model_summary(model))


epoch = 1000
batch = 32
holdout = 10000
e = 1
acc_train = []
acc_test = []

X, Y = train_data.getdatas()
los = pv.nn.loss.CrossEntropyLoss()
# opt = pv.optim.SGD(model.parameters())
```

```
opt = pv.optim.Adam(model.parameters())
start_time = time.time()

while e <= epoch:
    model.train()

    correct_tot = 0
    for I, (x, y) in enumerate(
        pv.data.data_generator(
            X[:-holdout], Y[:-holdout], batch_size=batch, shuffle=True
        )
    ):
        i = I + 1
        # print(i)
        opt.zero_grad()
        y_pred = model(x)
        loss = los(y, y_pred)
        correct = np.sum(y_pred.argmax(1, False).to_numpy() == y)
        acc = correct / y.shape[0]
        correct_tot += correct
        loss.backward()
        opt._step()
        if i % 10 == 0:
            end_time = time.time()
            print(
                f"epoch {e}/{epoch} \t batch {batch*i}/{Y.shape[0]-
holdout} \t loss {loss.item():.2f} \t accuracy {acc:.2f}    \t
{end_time-start_time:.2f} s"
            )
    e += 1

    model.eval()

    acc = correct_tot / (Y.shape[0] - holdout)
    print(f"********** train: epoch {e}/{epoch} \t accuracy
{100*acc:.2f}% **********")
    acc_train.append(acc)

    correct_tot = 0
    for x, y in pv.data.data_generator(
        X[-holdout:], Y[-holdout:], batch_size=128, shuffle=False
    ):
        y_pred = model(x)
        correct_tot += np.sum(y_pred.argmax(1, False).to_numpy() == y)
```

```
    acc = correct_tot / holdout
    print(f"********** test : epoch {e}/{epoch} \t accuracy
{100*acc:.2f}% **********")
    acc_test.append(acc)

    # opt = pv.optim.SGD(model.parameters(), 0.01 * (1 - acc))

pv.utils.storage.save_parameters(model.state_dict(), "train.model")


epochs = range(1, len(acc_train) + 1)
plt.plot(epochs, acc_train, label="Train", marker=".")
plt.plot(epochs, acc_test, label="Test", marker=".")
plt.title("Accuracy over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend()
# plt.ylim((0,1))
plt.tight_layout()
plt.savefig("raw.png")
plt.show()
```

eval.py 是在测试集上验证 10 分类准确率的代码。

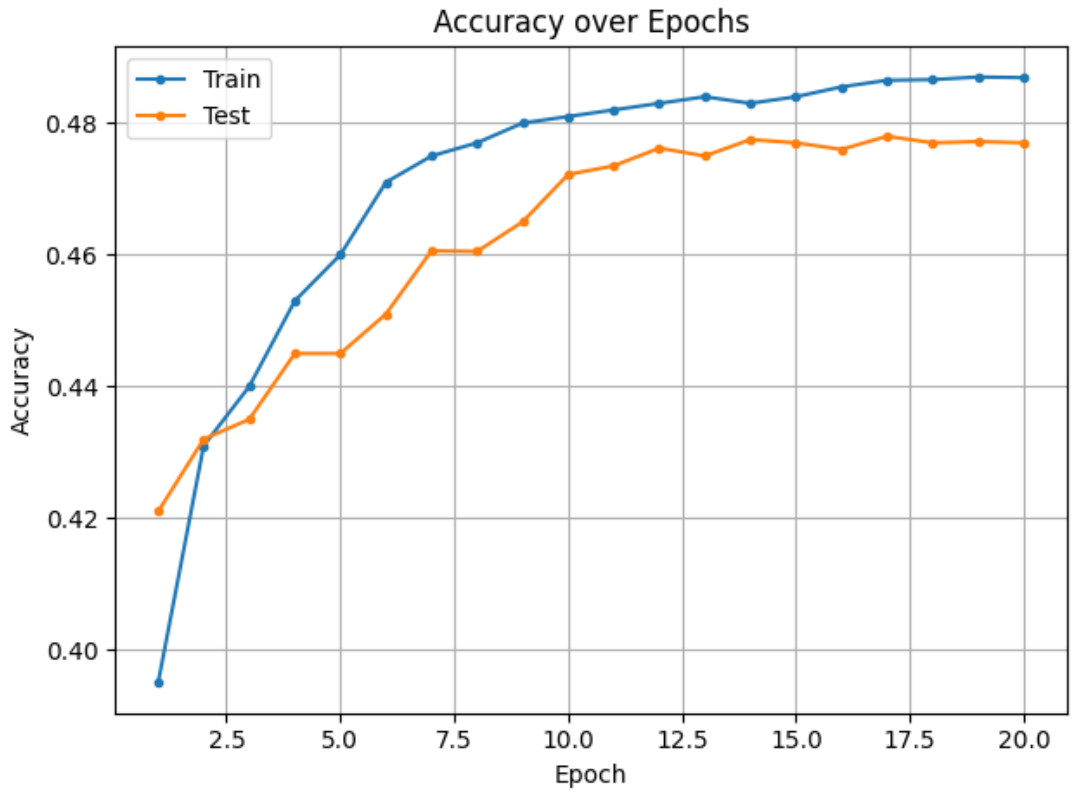
```
from utils import *
from model import *
import numpy as np
import matplotlib.pyplot as plt
import pyqpanda as pq
import pyvqnet as pv
import random
from functools import partial
import time


test_data = Dataset("data/cifar-10-batches-py/test_batch")


m = Model()
m.load_state_dict(pv.utils.storage.load_parameters("train.model"))
```

```
m.eval()
X, Y = test_data.getdatas()
correct_tot = 0
start_time = time.time()
for i, (x, y) in enumerate(pv.data.data_generator(X, Y, batch_size=128,
shuffle=False)):
    y_pred = m(x)
    correct_tot += np.sum(y_pred.argmax(1, False).to_numpy() == y)
    print(f"正在验证…… {min((i+1)*128,X.shape[0])}/{X.shape[0]}")
acc = correct_tot / Y.shape[0]
end_time = time.time()
print(f"********** eval : accuracy {100*acc:.2f}%    \t {end_time-
start_time:.2f} s **********")
```

运行结果



果然不出所料，准确率最后似乎上不去了，因为参数量太少了，学习能力较为低下。但即使是这么少的参数量，也跑了太多时间。前面 10 个 epoch 是我们取 batch=32

和 Adam 优化器慢慢训练的，大概 4 小时一轮，我们断断续续训了它 3 天的样子，达到了接近 50%的准确率。最后我们改了超参数 batch=128 以及 SGD 优化器，又训了一些时间，但准确率始终未能突破 50%。图上展现了 11~20 轮的情况，从图像上看几乎已经收敛了。（也有可能在缓步前进？）

最终在测试集上验证得到准确率如下：

由于本身该问题就是探索性量子机器学习问题，我们也就不强求什么了。在此也只是提出一种新的想法，至于其是否能够经得起检验，还是需要交给时间。