Project 2 项目报告

我选择的任务是 task1，在实现过程中，一开始使用的是用多项式加上回归模型尝试寻找一个可能的函数，计算得到的 nmse 误差在 0.6 左右。后来在网上搜索以后尝试用 TensorFlow 中的神经网络训练，得到的 nmse 值到 -1.46 左右。

问题定义

我选择的是 task1，这个任务是要求我们根据输入x中的一段连续的值，来估计输出y中的一个值，所取的x中连续的值的个数是m。换句话说，就是要确定从一段输入到一个输出的一个函数，使得他与实际的情况符合较好。

因此，我们要做的是从训练集中获取到一个函数，并在测试集上看看他的具体表现如何。

数据预处理

```
trainData = scipy.io.loadmat('E:\\SJTU-
Classes\\csMath\\lab2\\dataset\\task1\\PA_data_train.mat')

trainInput = trainData['paInput'].flatten()
trainOutput = trainData['paOutput'].flatten()

m = 2  # 记忆的深度

def create_memory_matrix(x, m):
    n = len(x)
    X = np.zeros((n - m, m + 1), dtype = complex)
    for i in range(m, n):
        X[i - m] = x[i - m:i + 1]
    return X

X_train = create_memory_matrix(trainInput, m)
y_train = trainOutput[m:]
scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_train_real = scaler_X.fit_transform(X_train.real)
X_train_imag = scaler_X.fit_transform(X_train.imag)
y_train_real = scaler_y.fit_transform(y_train.real.reshape(-1, 1)).flatten()
y_train_imag = scaler_y.fit_transform(y_train.imag.reshape(-1, 1)).flatten()
```

其中第一部分是在从文件中读取训练数据，函数是在从训练数据中根据所需的记忆长度拿取对每一个输出真正的可能输入，第三部分是在分离实部和虚部，以便后面将两部分分开来做函数的确定。

细节解释、使用的方法及实现

一开始尝试使用多项式进行拟合，发现在选择的 degree 为 2、3时比较好，最后使用了 2。

```
## 推测函数
# 创建多项式特征
poly = PolynomialFeatures(degree=2)
X_train_real_poly = poly.fit_transform(X_train_real)
X_train_imag_poly = poly.fit_transform(X_train_imag)
# 构建和训练多项式回归模型
model_real = LinearRegression()
model_imag = LinearRegression()

model_real.fit(X_train_real_poly, y_train_real)
model_imag.fit(X_train_imag_poly, y_train_imag)
```

后来使用神经网络的代码如下：

```
# 创建神经网络
def create_nn_model(input_shape):
    model = Sequential([
        Dense(64, activation='relu', input_shape=(input_shape,)),
        Dense(64, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

input_shape = X_train_real.shape[1]

model_real = create_nn_model(input_shape)
model_imag = create_nn_model(input_shape)

# 训练模型
model_real.fit(X_train_real, y_train_real, epochs=100, batch_size=32, verbose=0)
model_imag.fit(X_train_imag, y_train_imag, epochs=100, batch_size=32, verbose=0)
```

在获取到训练得到的函数（模型）后，用测试数据的输入和这个模型计算得到预测的结果，并且和真实结果进行对比，并计算 NMSE 的值。

```
## 测试部分
# 预测测试数据的输出
testData = scipy.io.loadmat('E:\\SJTU-
Classes\\csMath\\lab2\\dataset\\task1\\PA_data_test.mat')

testInput = testData['paInput'].flatten()
testOutput = testData['paOutput'].flatten()
X_test = create_memory_matrix(testInput, m)
y_test = testOutput[m:]
X_test_real = scaler_X.transform(X_test.real)
X_test_imag = scaler_X.transform(X_test.imag)
y_test_real = y_test.real
y_test_imag = y_test.imag
X_test_real_poly = poly.transform(X_test_real)
X_test_imag_poly = poly.transform(X_test_imag)

y_pred_real = model_real.predict(X_test_real_poly)
y_pred_imag = model_imag.predict(X_test_imag_poly)
```

```
y_pred = y_pred_real + 1j * y_pred_imag

# 计算NMSE
def calculate_nmse(true_values, predicted_values):
    # 分离实部和虚部
    I_out = true_values.real
    Q_out = true_values.imag
    I_pred = predicted_values.real
    Q_pred = predicted_values.imag

    # 计算NMSE
    numerator = np.sum((I_out - I_pred)**2 + (Q_out - Q_pred)**2)
    denominator = np.sum(I_out**2 + Q_out**2)

    nmse = 10 * np.log10(numerator / denominator)
    return nmse

nmse_value = calculate_nmse(y_test, y_pred)
print(f'NMSE: {nmse_value:.6f}')
```

测试结果

在使用多项式回归的方法时，在 m = 2，degree = 2 时得到了较小的 NMSE值，为 0.597359。

在使用神经网络进行训练时，同样在 m = 2 时得到了更小的 NMSE值，为 -1.459520。

感觉多项式回归的方法结果误差比较大的可能原因是这个函数不好用多项式进行拟合，而且由给我们的图中可以看出数据在输入的二范数不断增大时输出是有上界的，这也可能导致这种拟合的方法不太好。当然，在上面的编码与测试的过程中，也可以感受到训练一个神经网络所需要的时间与多项式degree较小的比较，花费的时间是要多的多的。