项目分析（2023-12-13）

对模型建立的初步尝试-利用LSTM的神经网络来做时间序列值的预测

1 选题背景

本项目旨在立足中南大学的风洞实验室，通过分析车辆在桥面风环境的影响下的动力学特性，以及对桥面风自身特性的分析，建立一种“基于风速分析的车辆安全性评估方法”，结合风洞试验与数值模拟所得数据与从大数据中获取的数据样本，形成数据库，最后训练神经网络模型，形成基于深度学习的风险预警系统，对安全行车和交通合理控流提出合理建议。本项目有助于提高桥梁交通安全性和效率，减少因强风引起的车辆事故和拥堵，对于大跨度桥上安全行车具有重要的参考价值。

通过分析选题要求，结合实际安全预警需要，模型的建立关键点在于：（1）**通过实地调查与试验获取真实可靠的，特定区域的风力变化数据集；（2）选定合适的神经网络算法框架；（3）不断调校参数，迭代训练，形成具有鲁棒性的，可以经受得住实际检验的模型；**

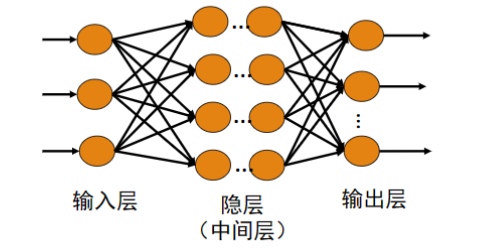
我们最终需要获得的，是一个可以不断增补数据，具有持续学习而不遗忘和实时预测能力的模型，究其本质，就是要建立在特定区域的“**时间—风速映射关系”。**

（可以讲一下数据获取的渠道，比如实验，气象站数据等）

目前主要流行的神经网络框架有很多，下面进行简要的对比和介绍：

（1）传统BP神经网络

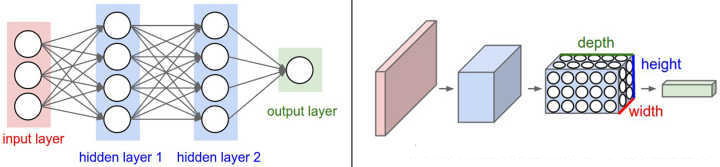
BP神经网络特征是“信号正向传播，而误差反向传播”，采用的是“结果反馈机制”，简而言之，BP网络主要特点是使用了BP（Back Propogation）优化策略，结构一般比较简单；示意图如下：



其优点是，得益于非线性激活函数，BP网络具有一定的非线性映射能力，除此之外，还具有自学习和自适应能力（通过每一次迭代的梯度来修正网络各个参数，使之不断接近真实模型），以及一定的泛化能力和容错能力；

但与此同时，BP网络也具有一些缺点：容易陷入局部最优，收敛速度较慢，神经网络的结构选择没有成熟的理论指导，以及网络预测能力和训练结果的矛盾（泛化能力和拟合能力在超过一定范围后此消彼长，需要取舍，找到合适的“平衡点”）；

（2）CNN（Convolutional Neural Networks，卷积神经网络）



（BP神经网络与卷积神经网络的对比图）

卷积神经网络是一类包含卷积运算目具有深度结构的前馈神经网络，相比早期的BP神经网络，卷积神经网络最重要的特性在于“局部感知”与“参数共享，整体架构包含”输入层——卷积层——池化层——全连接层——输出层”。**CNN具有一些传统方法，比如BP网络所没有的优点:** 良好的容错能力、并行处理能力和自学习能力；运行速度快，自适应性能好；允许样本有较大的缺损、畸变等；

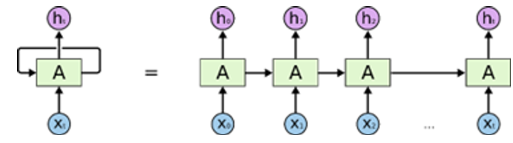
当然，CNN同时也存在着一定的局限：CNN同样容易收敛到局部最优而非全局最优；池化层会丢失一定的有价值信息，忽略了局部与整体之间的关联性；特征提取的物理含义不是十分明确，导致可解释性一般等。

（3）RNN与LSTM

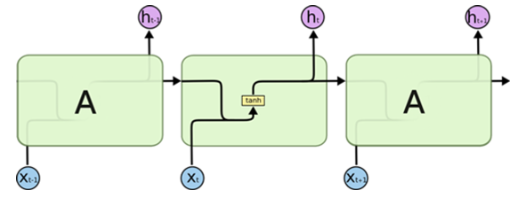
（3.1）RNN简介

**循环神经网络**（recurrent neural network，简称**RNN**）是指在全连接神经网络的基础上增加了**“前后时序上的关系”**，可以更好地处理比如机器翻译,时序数据预测等的与时序相关的问题，是一种对**序列数据有较强的处理能力**的网络。在网络模型中不同部分进行**权值共享**使得模型可以扩展到不同样式的样本，意味着它有很好的泛化性能；

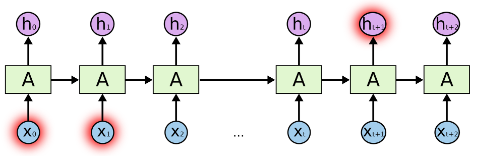
（RNN按时间序列展开的结构图）



（公式化的RNN结构）



RNN存在着的问题：**长期依赖（Long-TermDependencies）**



RNN的关键点之一就是他们可以用来连接先前的信息到当前的任务上，但还有很多依赖因素；如果预测信息和相关先验信息之间的距离过大，RNN就会丧失良好的学习能力，表现就会不好；

RNN 会受到**短时记忆的影响（影响较大）**。如果一条序列足够长，那它们将很难将信息从较早的时间步传送到后面的时间步。

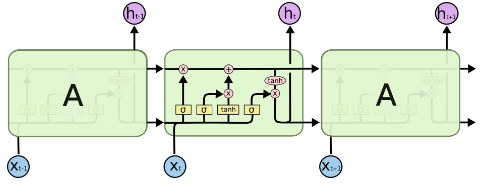
存在长期依赖问题的原因：**梯度消失**

RNN在训练的过程中，会采用反向传播过程（使用梯度下降算法）来进行优化，而梯度是用于更新神经网络的权重值（新的权值 = 旧权值 - 学习率\*梯度），梯度会随着时间的推移不断下降减少，而当梯度值变得非常小时，就不会继续学习。

换言之，在递归神经网络中，获得小梯度更新的层会停止学习—— 那些通常是较早的层。 由于这些层不学习，**RNN会忘记它在较长序列中以前看到的内容，因此RNN只具有短时记忆**。

（3.2）LSTM

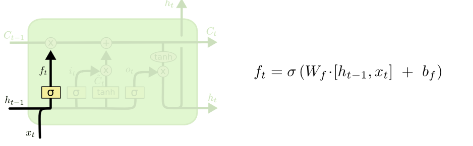
LSTM(长短时记忆网络)属于RNN的变体，相比于RNN，LSTM的重复的模块拥有一个不同的结构，包含四个交互的层，**三个Sigmoid 和一个tanh层，并以一种非常特殊的方式进行交互。（LSTM网络结构示意图）**

****

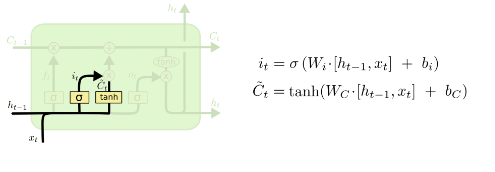
LSTM网络的核心思想在于：**通过细胞状态和门结构（忘记门，输入门和输出门），来在有限的记忆能力下实现关键信息的保留和不相关数据的剔除**；（**因记忆能力有限，记住重要的，忘记无关紧要的**）在一定程度上**解决了RNN梯度消失和梯度爆炸这两个问题**；

现在将各种门结构的示意图和功能分析展示如下：

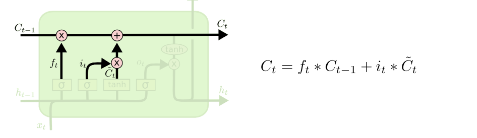
（1）‘忘记门’（规定：丢弃旧信息）



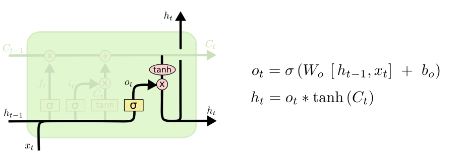
（2）输入门（规定：添加新信息）



（3）细胞状态（实践前两步）



（4）输出门（过滤信息，决定每一个时刻的输出值）



2 实验过程

**前言：**随着笔者对于各种算法原理了解的不断深入，以及对各个算法的应用场景进行分析，认识到了CNN的应用场景主要在图像的识别和视频处理，同时传统的BP网络容易陷入局部最优，并且适合项目选题的网络结构难以选定，因此不再选用CNN和BP结构作为本选题的适用算法，而选择适合于时间序列数据分析的RNN网络，但是一般的RNN网络伴随着长期依赖的问题，时间跨度过长会影响学习效果，而准确的风速预测又是需要建立在长期观测的基础之上的。因此，经过对比，决定选择RNN网络的一种变体—LSTM作为网络模型建立的框架，这不仅可以较好的适合于时序数据（时间-风速）的分析，也一定程度上缓解了传统RNN梯度消失和梯度爆炸的问题；

**实验过程：**

**（1）实验环境：**

操作系统：Windows11

IDE（集成开发环境）：Pycharm Community

解释器：Python3.8

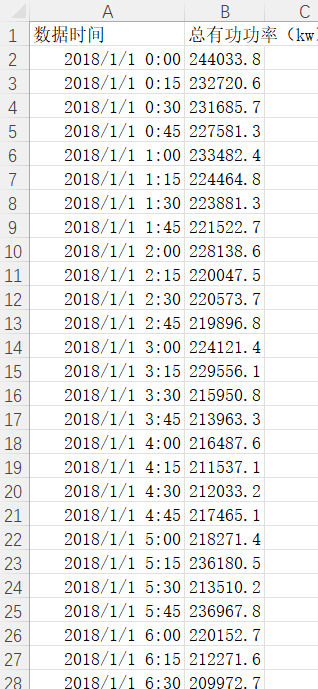
Libs（包和库）：csv numpy pandas matplotlib seaborn sklearn tnsorflow keras

神经网络框架：TensorFlow

**（2）数据集介绍**

由于本实验所需的风力数据集还需要进一步获取，因此利用了一个相似的数据集—时间—风力发电功率数据集作为测试数据集，用来检验模型性能；

数据集格式如下：



**（3）代码组成分析（只提及关键步骤,具体的代码作用已经在代码中进行了注释）**

**（3.1）数据预处理：具体包括数据集导入，训练集和测试集划分，数据异常和缺失值处理，格式规整，归一化处理(将原始数据压缩到0-1范围内，有益于模型处理数据)等，目的是为了确保数据格式规范，数据范围正常；**

"""数据获取"""  
#获取数据集 订正数据格式  
plt.rcParams['font.sans-serif'] = ['SimHei'] # 显示中文标签  
plt.rcParams['axes.unicode\_minus'] = False#解决负数问题  
x\_Seq\_len=16  
#修改csv文件路径  
dataset = pd.read\_excel("D:/pythoncode/WindProject/风力发电功率数据集.xlsx")  
dataset['数据时间']=pd.to\_datetime(dataset['数据时间'],format="%Y-%m-%d %H:%M:%S")  
dataset.index=dataset.数据时间#将其索引变为时间  
dataset.drop(columns='数据时间',axis=1,inplace=True)  
plt.figure()  
plt.plot(dataset)  
plt.show()  
"""  
数据清洗  
"""  
#缺失值处理  
#查看是否有缺失值  
print(dataset.info())#无缺失值  
# print(dataset[dataset.isnull()==False])#无  
# dataset['总有功功率（kw）']=dataset['总有功功率（kw）'].fillna(0) 对缺失值填值处理  
# dataset1=dataset[dataset['总有功功率（kw）'].notnull()] 剔除存在缺失值的数据，自己选择一直缺失值处理的方法  
#异常值处理  
"""  
箱型图查看 可视化数据分布  
"""  
f, ax = plt.subplots()  
sns.boxplot(y='总有功功率（kw）', data=dataset, ax=ax)  
plt.show()  
s = dataset.describe()  
# 基本统计量，存在异常值的将其筛选出来进行处理，可以用中位数填值或者众数填值，方法任选，这里没有异常值就没有处理  
q1 = s.loc['25%']  
q3 = s.loc['75%']  
iqr = q3 - q1#分位差  
mi = q1 - 1.5 \* iqr#下限，低于这个为异常值  
ma = q3 + 1.5 \* iqr#上限，高于这个为异常值  
#无异常值  
"""  
 数据归一化处理，均值为0，方差为1  
"""  
scaler = MinMaxScaler()  
dataset['总有功功率（kw）'] = scaler.fit\_transform(dataset['总有功功率（kw）'].values.reshape(-1, 1))  
#将归一化的数据保持  
with open('data.csv','w',encoding='utf-8',newline='')as f:  
 w=csv.writer(f)  
 w.writerow(dataset['总有功功率（kw）'])  
#归一化后的绘图  
dataset['总有功功率（kw）'].plot()  
plt.show()

"""

数据集划分（X\_train, X\_test, y\_train, y\_test）

"""

dataset\_new = dataset  
# X为特征数据集，y为标签数据集  
X, y = create\_new\_dataset(dataset\_new.values, seq\_len=x\_Seq\_len)  
# X\_train为数据训练集，X\_test为数据测试集,y\_train为标签训练集,y\_test为标签测试集合  
X\_train, X\_test, y\_train, y\_test = split\_dataset(X, y)  
# 基于新的X\_train, X\_test, y\_train, y\_test创建批数据(batch dataset)  
# 测试批数据  
test\_batch\_dataset = create\_batch\_data(X\_test, y\_test, batch\_size=24, data\_type=1)  
# 训练批数据  
train\_batch\_dataset = create\_batch\_data(X\_train, y\_train, batch\_size=24, data\_type=2)

**（3.2）构建模型：搭建网络结构，这里使用到了Tensorflow框架，定义了LSTM网络层，是整个工程的核心**

"""  
构建模型  
"""  
model = Sequential([  
 layers.LSTM(8, input\_shape=(x\_Seq\_len, 1)),  
 layers.Dense(1)  
])  
# 定义 checkpoint，保存权重文件  
file\_path = "best\_checkpoint.hdf5"#将数据加载到内存  
checkpoint\_callback = tf.keras.callbacks.ModelCheckpoint(filepath=file\_path,  
 monitor='loss',  
 mode='min',  
 save\_best\_only=True,  
 save\_weights\_only=True)

**（3.2）模型训练：利用已有的数据集，设定模型超参数，指定优化器和损失函数，迭代次数，进行训练；**

"""  
编译运行，预测，指标检验  
"""  
# 模型编译  
model.compile(optimizer='adam', loss="mae")  
# 模型训练（次数200）  
history = model.fit(train\_batch\_dataset,  
 epochs=100,  
 validation\_data=test\_batch\_dataset,  
 callbacks=[checkpoint\_callback])  
# 显示 train loss 和 val loss  
plt.figure()  
plt.plot(history.history['loss'], label='train loss')  
plt.plot(history.history['val\_loss'], label='val loss')  
plt.title("LOSS")  
plt.xlabel("Epochs")  
plt.ylabel("Loss")  
plt.legend(loc='best')  
plt.show()  
# 模型验证  
test\_pred = model.predict(X\_test, verbose=1)  
plt.figure()  
d1=plt.plot(y\_test, label='True')  
d2=plt.plot(test\_pred, label='pred')  
plt.legend([d1,d2],labels=['True','pred'])  
plt.show()  
# 计算r2  
score = r2\_score(y\_test, test\_pred)  
print("r^2 的值： ", score)  
# 绘制test中前100个点的真值与预测值  
y\_true = y\_test # 真实值  
y\_pred = test\_pred # 预测值  
  
fig, axes = plt.subplots(2, 1)  
ax0=axes[0].plot(y\_true, marker='o', color='red',label='true')  
ax1=axes[1].plot(y\_pred, marker='\*', color='blue',label='pred')  
plt.show()

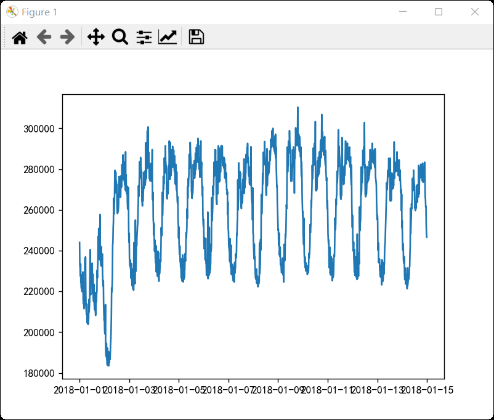
**（3.2）模型测试和评估：利用测试集数据，对模型的拟合性能进行评估，计算模型精确度和R2分数来评价模型性能；**

"""  
模型测试，评估  
"""  
# 选择test中的最后一个样本  
sample = X\_test[-1]  
sample = sample.reshape(1, sample.shape[0], 1)  
# 模型预测  
sample\_pred = model.predict(sample)#predict()预测标签值  
ture\_data = X\_test[-1] # 真实test的最后20个数据点  
# 预测后48个点  
preds=predict\_next(model,ture\_data,48)  
# 绘图  
plt.figure()  
plt.plot(preds, color='yellow', label='Prediction')  
plt.plot(ture\_data, color='blue', label='Truth')  
plt.xlabel("Epochs")  
plt.ylabel("Value")  
plt.legend(loc='best')  
plt.show()  
relative\_error = 0  
"""模型精确度计算"""  
for i in range(len(y\_pred)):  
 relative\_error += (abs(y\_pred[i] - y\_true[i]) / y\_true[i]) \*\* 2  
acc = 1- np.sqrt(relative\_error / len(y\_pred))  
print(f'模型的测试准确率为：{acc}')

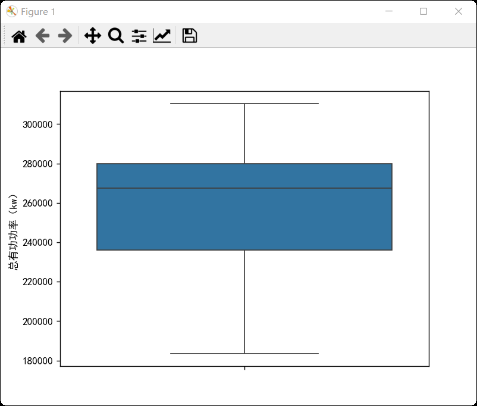
3 实验结果分析

（1）处理后的数据结果展示（也就是原始数据集的分布情况）

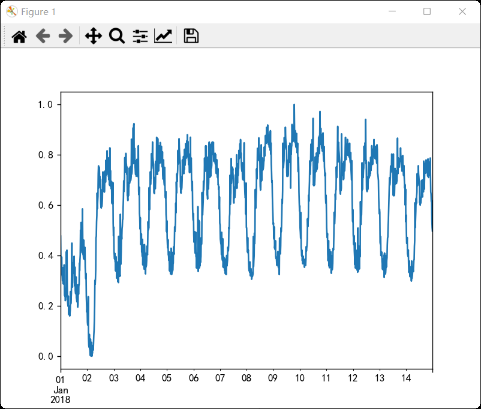
原始数据分布图



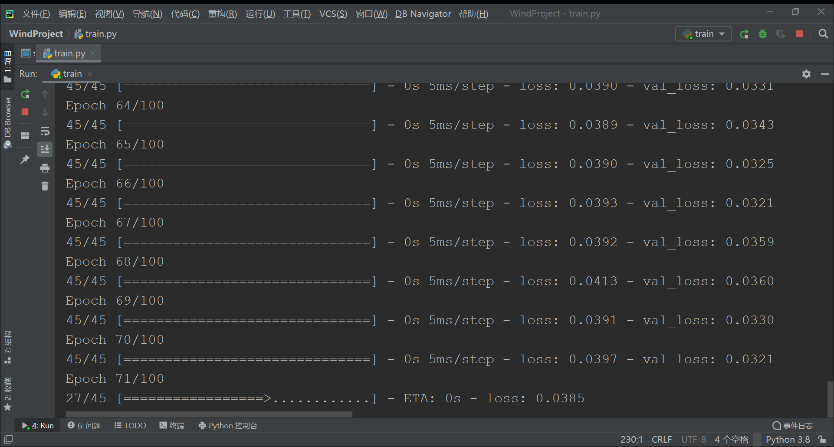
原始数据分布箱型图

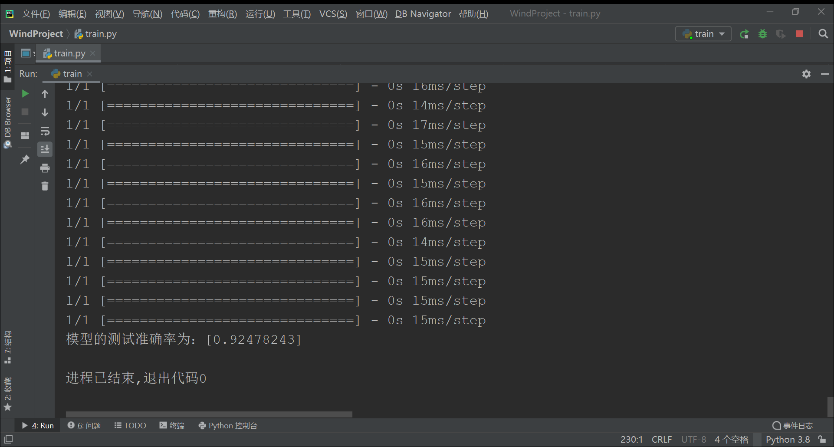


归一化处理（将标签值归一化到0-1的范围内）之后的数据



（2）训练过程



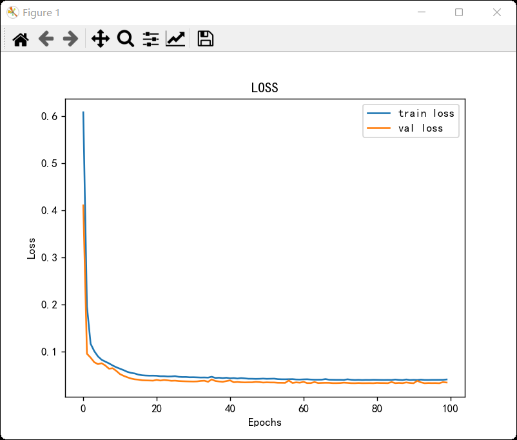


模型准确率：0.93

（3）LOSS函数在整个训练过程中的收敛性

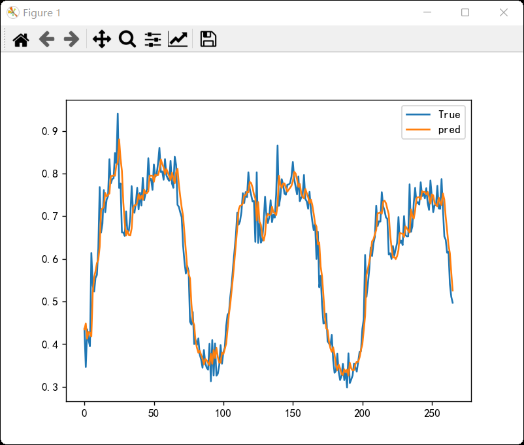
LOSS函数在仅仅经过仅仅40个轮次的训练之后就达到了收敛效果，进一步论证了LSTM算法性能的优越；

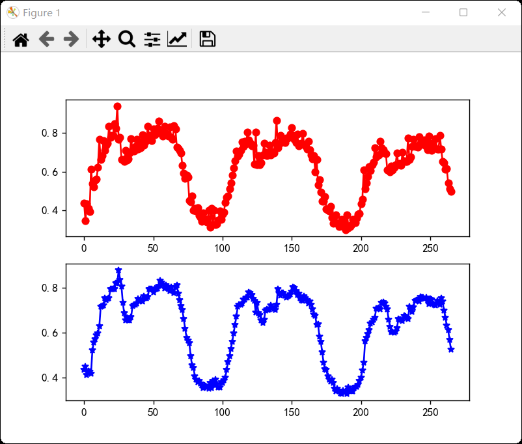
LOSS函数随迭代次数的变化曲线



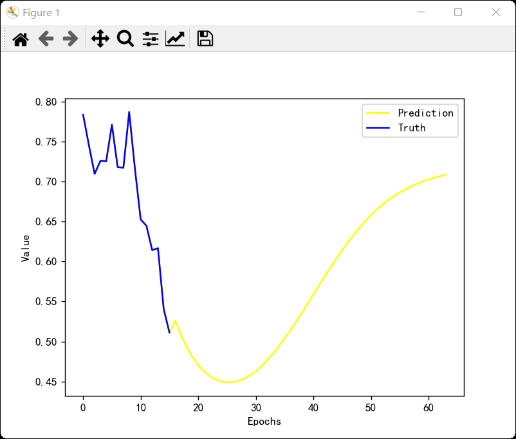
（4）测试数据集预测结果

下面是测试集中前100个点的真值与预测值的对比图（数据经过归一化处理）





选取测试集中最后一个样本进行预测，预测值随迭代次数的变化图



4 总结与展望：

本次模型测试是建立在LSTM网络和风力发电数据集的基础之上，模型的预测准确率达到了90%以上，LOSS函数很快的取得了收敛，取得了良好的预测效果，这得益于LSTM对长时间序列数据的良好处理能力。但是，由于真实的风力数据随时间的变化模型可能与本次实验所采取的数据分布情况并不一致，因此还需要再获取到大量风力数据集的基础上再次进行测试，完善，这也是后期项目开展需要进行的工作；