> *范峻铭22121286*
>
> *https://github.com/HatcherRobotics/intelligent-information-processing*
>
> *代码均为原创，雷同即为抄袭*

数据清洗与数据加载器

使用Pandas读取Excel中的数据，对数据进行归一化或标准化处理，十九维数据作为特征，以一维数据(极大风速)为标签；

```
    def __init__(self,predict_step=3,label=5):
        standard_scaler = StandardScaler()
        raw_data = pd.read_excel("weather.xlsx")
        self.predict_step = predict_step
        self.label = label
        #标准化
        self.data = standard_scaler.fit_transform(raw_data)
        self.mean = standard_scaler.mean_[label]
        self.sd = math.sqrt(standard_scaler.var_[label])

```

```
    def __init__(self,predict_step=3,label=5):
        minmax_scaler = MinMaxScaler()
        raw_data = pd.read_excel("weather.xlsx")
        self.predict_step = predict_step
        self.label = label
        #归一化
        self.data = minmax_scaler.fit_transform(raw_data)
        self.min = minmax_scaler.data_min_[label]
        self.max = minmax_scaler.data_max_[label]
```

反归一化/标准化函数在评估模型以及数据可视化中用到，将归一化/标准化数据变换为原始数据的尺度；

```
    def denormalize(self, x):
        x = x*(self.max-self.min)+self.min
        return x
```

```
    def denormalize(self, x):
        x = x*self.sd+self.mean
        return x
```

采用滑动窗口的方式建立数据集，步长为1，序列长度为50，预测布长为3；

```
    def construct_set(self, train_por=0.65,val_por=0.2,test_por=0.15, window_size=50):
        ......
        for i in range(train_seqs.shape[0] - window_size-self.predict_step+1):
            train_seq = train_seqs[i:i+window_size+self.predict_step]
            train_x.append(train_seq[0:window_size,:])

 train_y.append(train_seq[window_size:window_size+self.predict_step,self.label])

        for i in range(val_seqs.shape[0] - window_size-self.predict_step+1):
            val_seq = val_seqs[i:i+window_size+self.predict_step]
            val_x.append(val_seq[0:window_size,:])

 val_y.append(val_seq[window_size:window_size+self.predict_step,self.label])

        for i in range(test_seqs.shape[0] - window_size-self.predict_step+1):
            test_seq = test_seqs[i:i+window_size+self.predict_step]
```

```
            test_x.append(test_seq[0:window_size,:])

 test_y.append(test_seq[window_size:window_size+self.predict_step,self.label])
        ......
```

LSTM:长短期记忆网络

搭建神经网络

调库实现LSTM，取其最后一步的输出，将其通过两层全连接以及激活函数得到3维输出，对应三个小时的预测值；

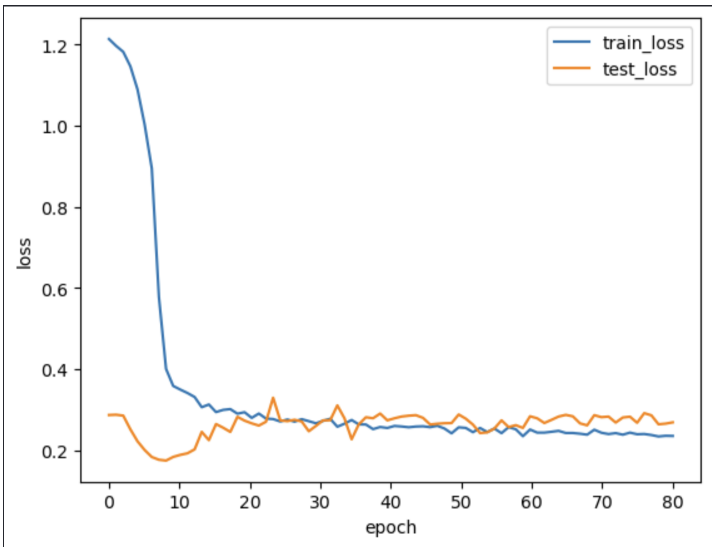
```
class LSTMNet(nn.Module):
    def __init__(self,batch_size,input_size,hidden_size,output_size,num_layers,seq_len,device="cuda"):
        super().__init__()
        self.batch_size = batch_size
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.num_layers = num_layers
        self.seq_len = seq_len
        self.lstm = nn.LSTM(input_size=input_size,hidden_size=hidden_size,num_layers=num_layers,batch_first=False)
        self.outlinear = nn.Sequential(

 nn.Linear(in_features=self.hidden_size,out_features=hidden_size//2),
            nn.LeakyReLU(),

 nn.Linear(in_features=hidden_size//2,out_features=self.output_size)
        )
        for param in self.parameters():
            if param.dim() > 1:
                nn.init.xavier_uniform_(param)
    def forward(self,x):
        h,c = (torch.zeros(self.num_layers,self.seq_len,self.hidden_size).to(x.device) for _ in range(2))
        H,(h,c) = self.lstm(x,(h,c))
        out = H[:,-1,:].squeeze()
        out = self.outlinear(out)
        return out
```

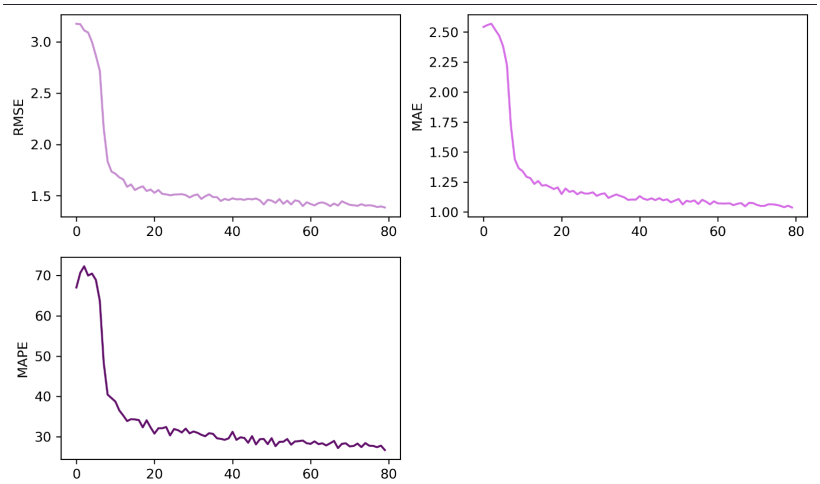
模型训练

取学习率为0.0001，优化器选择Adam，训练轮次为80轮，批量大小为16，LSTM的层数为3层。训练结果如下图所示



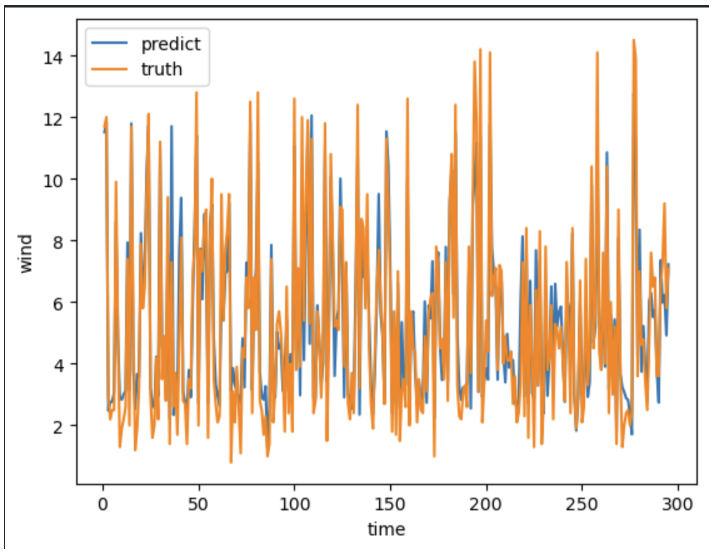
模型评估

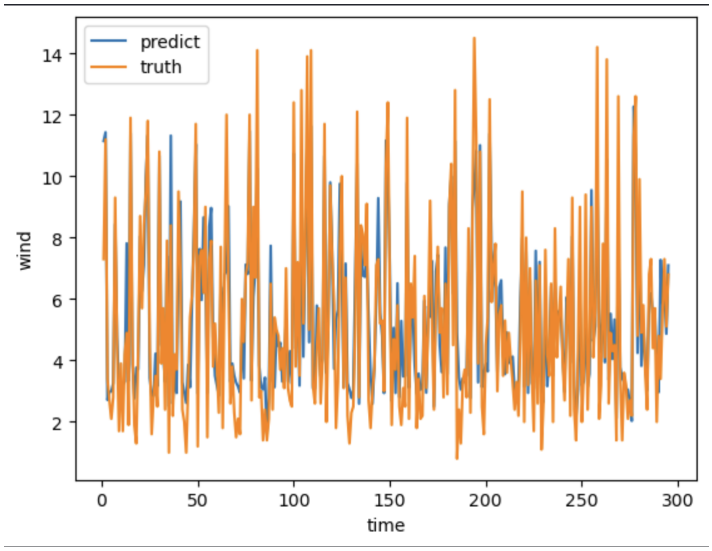
使用平均均方根误差RMSE，平均绝对误差MAE，平均绝对百分比误差MAPE来评估模型：

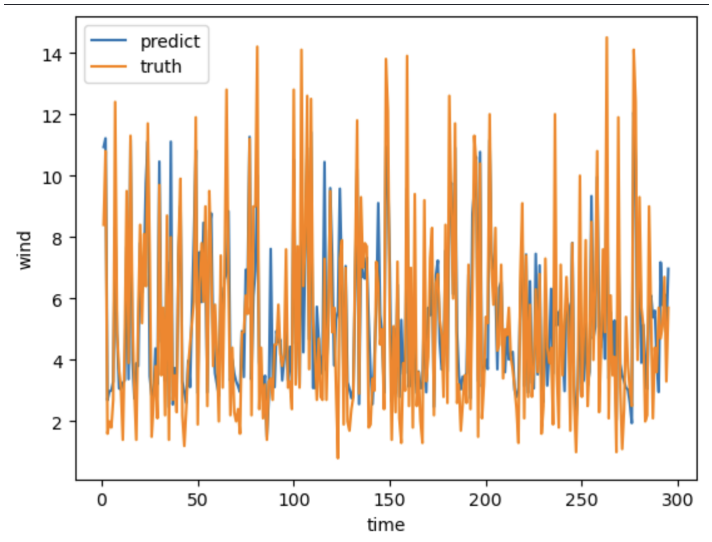


数据可视化

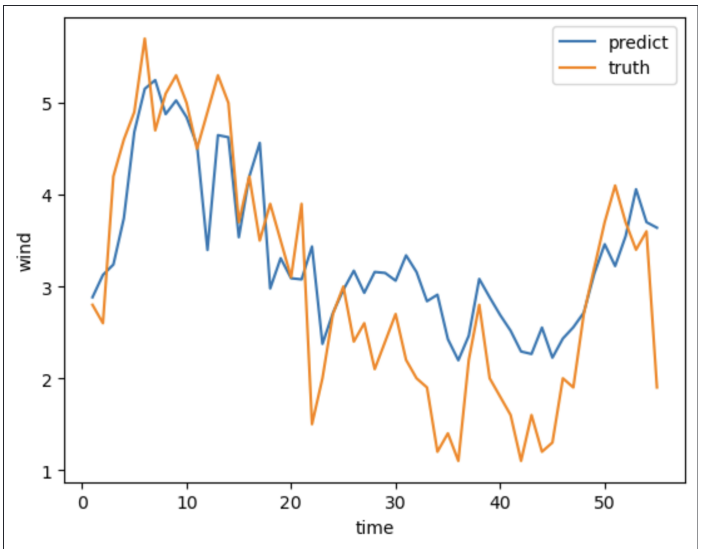
1h,2h,3h的预测值与真实值在训练集比较如下图所示

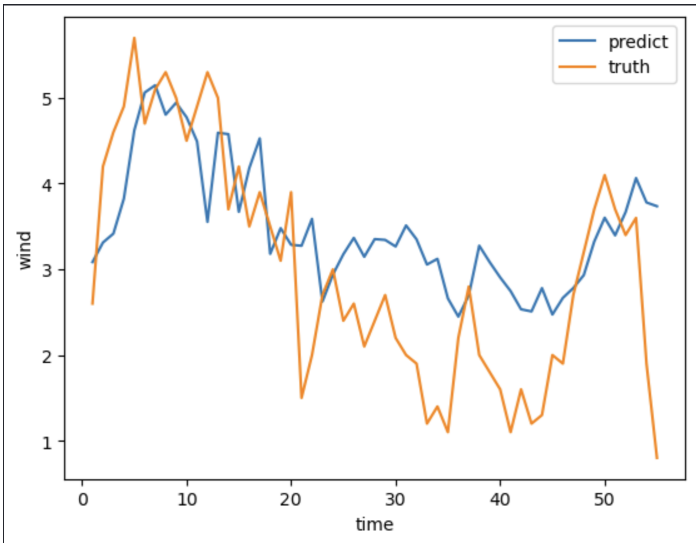


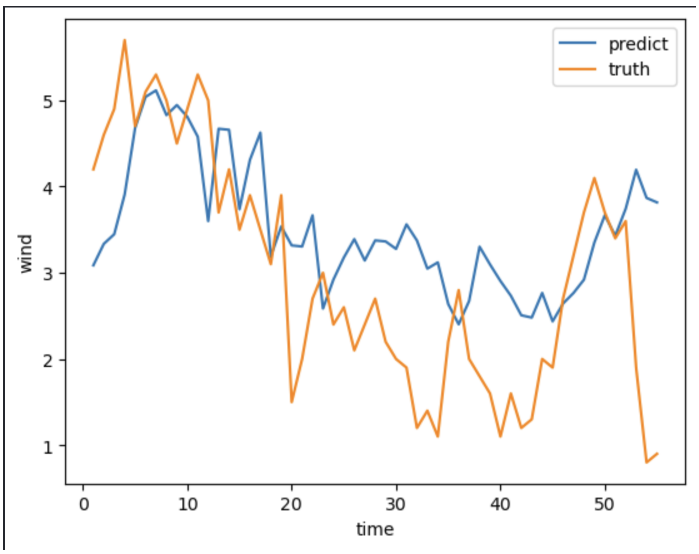




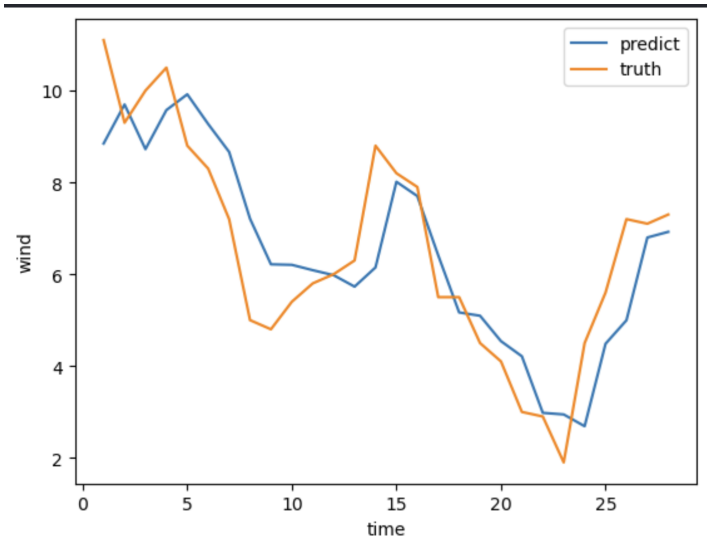
1h,2h,3h的预测值与真实值在验证集上的比较如下图所示

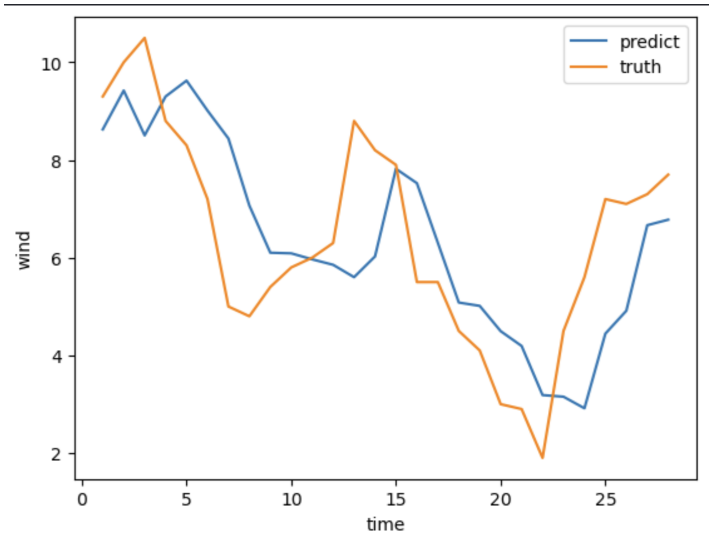


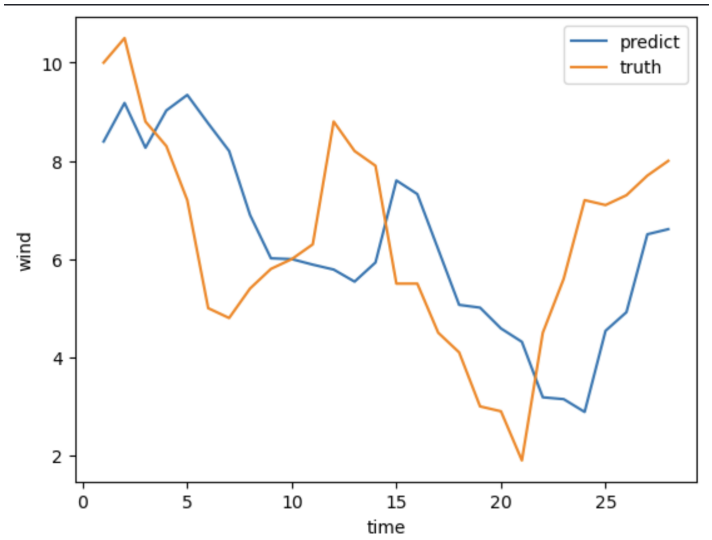




1h,2h,3h的预测值与真实值在测试集上的比较如下图所示







总结

LSTM较好的完成了预测任务，但其预测具有滞后性，且预测长度越长情况越严重。

TCN：时间卷积网络

考虑到非因果的普通一维卷积会涉及到未来信息，选择以空洞因果卷积为基础的时间卷积网络。

搭建神经网络

原作者已开源代码，但代码为本人阅读论文后原创；

padding为了使输入序列与输出序列相等，使用Chomp1d去掉由最右端填充元素参与卷积的生成元素，确保卷积不使用未来特征。

```
class Chomp1d(nn.Module):
    def __init__(self,chomp_size):
        super(Chomp1d, self).__init__()
        self.chomp_size = int(chomp_size)
    def forward(self,x):
        return x[:,:,0:-self.chomp_size].contiguous()
#在padding与chomp浪费了大量时间为了使输出序列相等
class TCNResidualBlock(nn.Module):
    def __init__(self,inchannel,outchannel,kernelsize,dilation):
        super(TCNResidualBlock,self).__init__()
        self.ke = int(kernelsize+(kernelsize-1)*(dilation-1))
        self.conv = nn.Sequential(

 weight_norm(nn.Conv1d(inchannel,outchannel,kernel_size=kernelsize,dilati
on=dilation,padding=self.ke-1)),
            Chomp1d(self.ke-1),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.2),

 weight_norm(nn.Conv1d(outchannel,outchannel,kernel_size=kernelsize,dilat
ion=dilation,padding=self.ke-1)),
            Chomp1d(self.ke-1),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.2),
        )
        self.conv1x1 = nn.Sequential(
            weight_norm(nn.Conv1d(inchannel,outchannel,kernel_size=1)),
            )

    def forward(self,X):
        Y = self.conv(X)
        X = self.conv1x1(X)
        out = F.relu(X+Y)
        return out
```

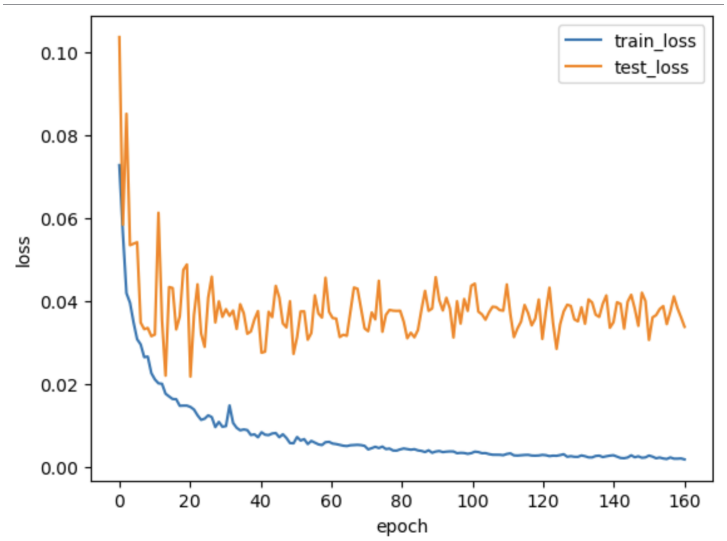
```
#通道数是特征的意思
```

```
#input: batch_size * features * seq_length但数据集的输入是batch_size * seq_length * features
block1 = TCNResidualBlock(14,32,5,1)
block2 = TCNResidualBlock(32,16,5,2)
block3 = TCNResidualBlock(16,8,3,4)

class TCNet(nn.Module):
    def __init__(self,hidden_size,predict_step):
        super(TCNet,self).__init__()
        self.features = nn.Sequential(
            block1,block2,block3
        )
        self.flatten = nn.Flatten()
        self.regression = nn.Sequential(
            nn.Linear(hidden_size, hidden_size//2),
            nn.Dropout(p=0.2),
            nn.ReLU(),
            nn.Linear(hidden_size//2, predict_step),
            nn.LeakyReLU()
        )
        for param in self.parameters():
            if param.dim() > 1:
                nn.init.xavier_uniform_(param)
    def forward(self,X):
        X = X.permute(0,2,1).contiguous()
        features = self.features(X)
        out = self.flatten(features)
        out = self.regression(out).squeeze()
        return out
```

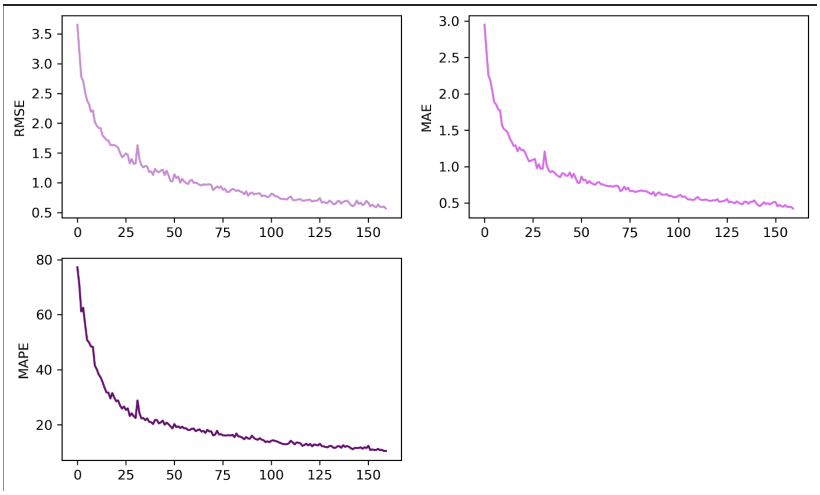
模型训练

TCN的收敛速度要明显慢于LSTM，取学习率为0.001，优化器选择Adam，训练轮次为160轮，批量大小为16，损失函数选择均方损失，堆叠三个TCN块，其空洞系数分别为1，2，4，卷积核的大小分别为5，5，3，输出通道数分别为32，16，8，训练结果如下图所示



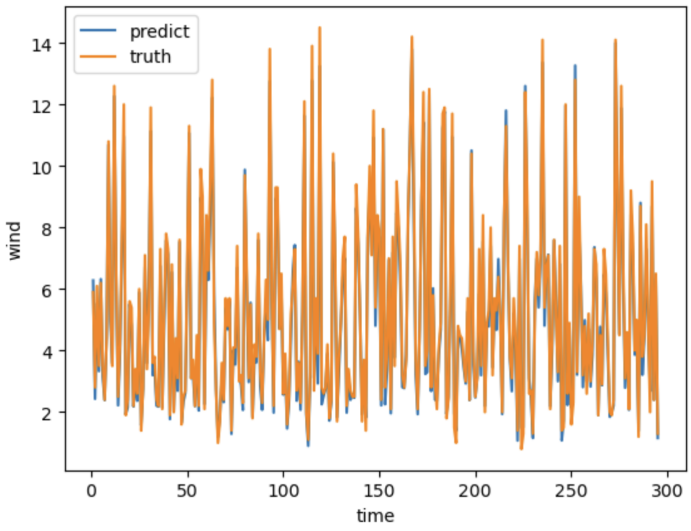
模型评估

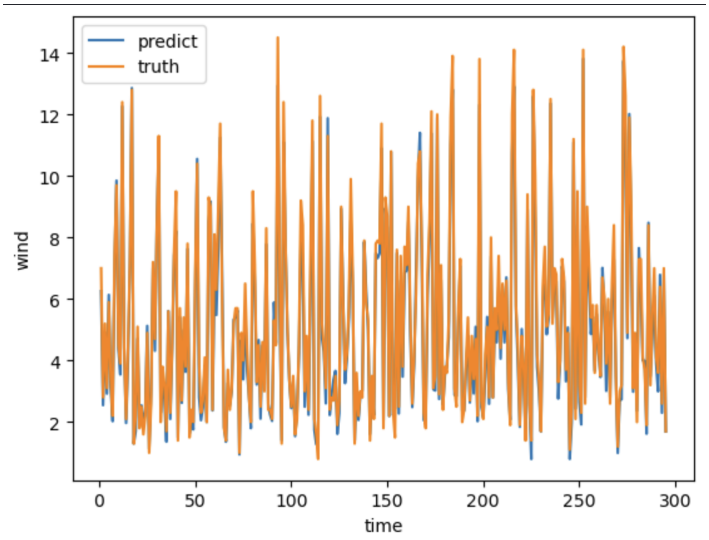
使用平均均方根误差RMSE，平均绝对误差MAE，平均绝对百分比误差MAPE来评估模型：

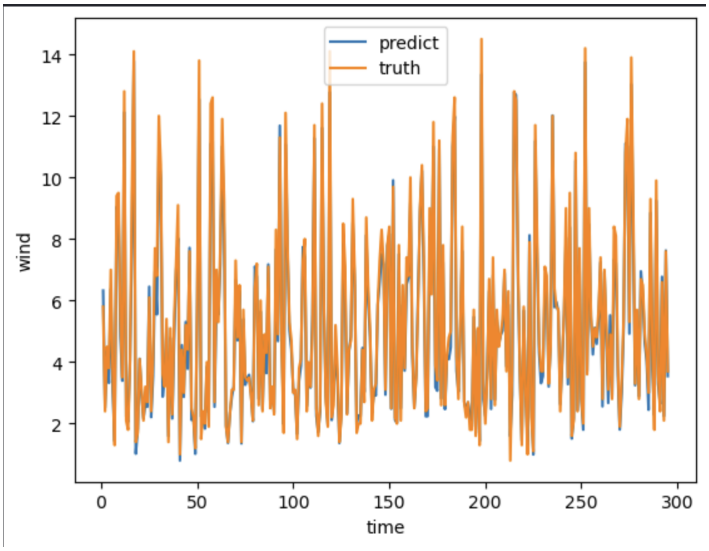


数据可视化

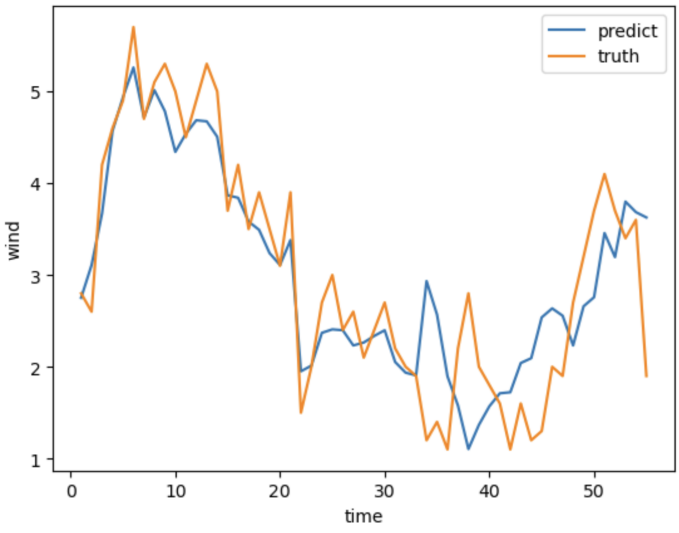
1h,2h,3h的预测值与真实值在训练集比较如下图所示

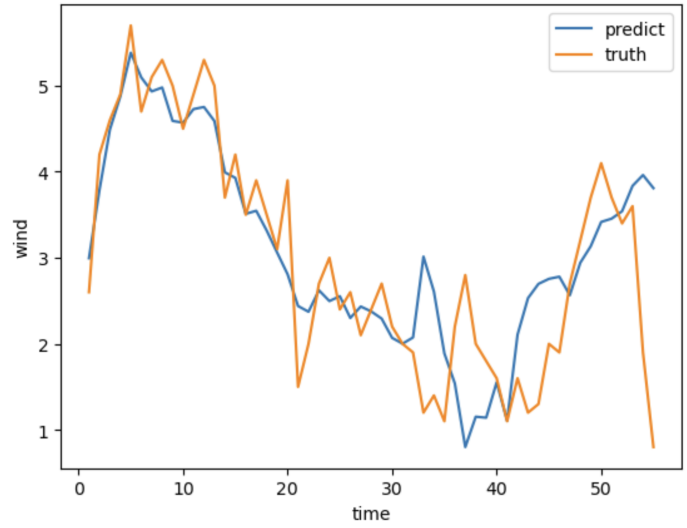


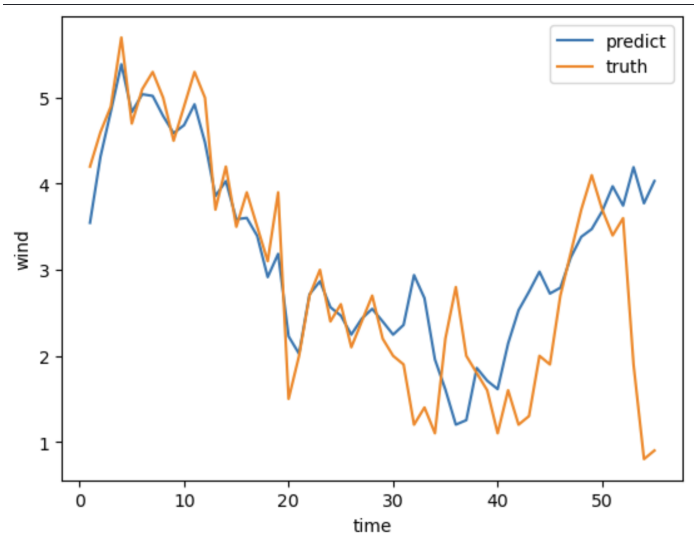




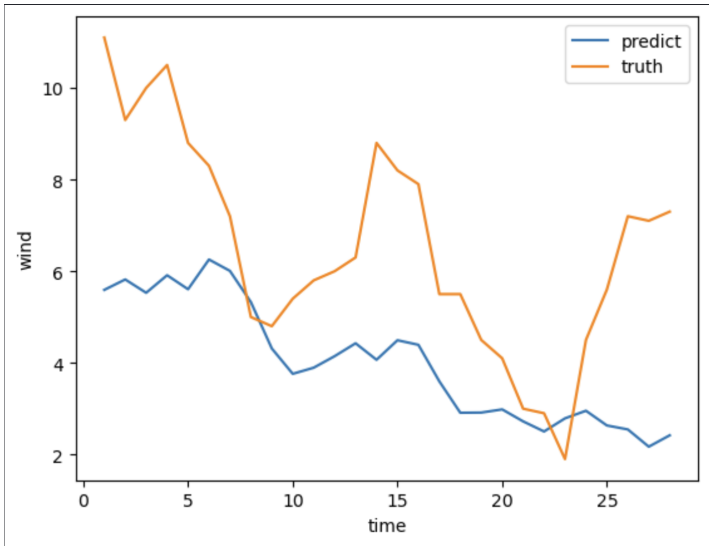
1h,2h,3h的预测值与真实值在验证集比较如下图所示

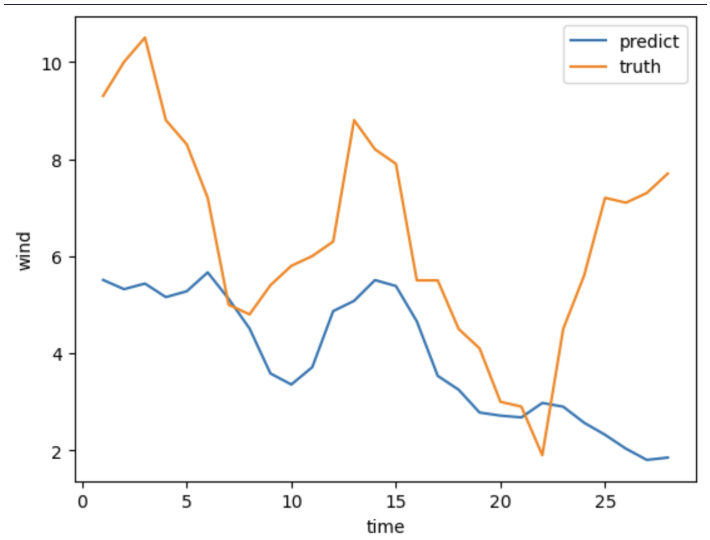


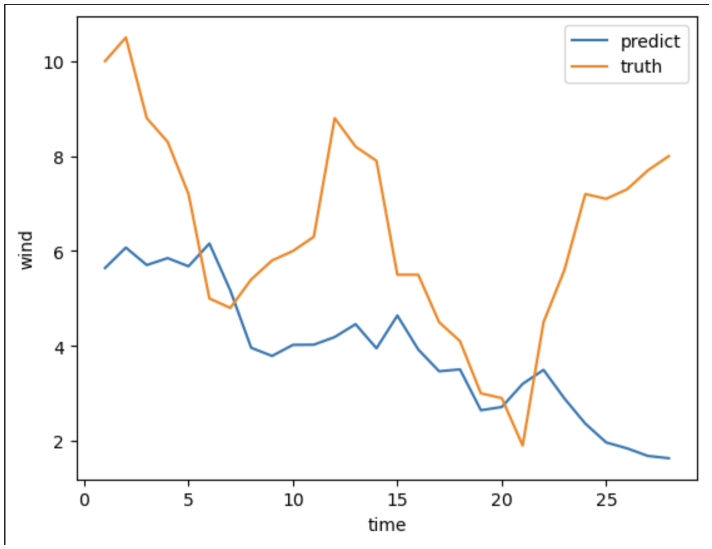




1h,2h,3h的预测值与真实值在测试集比较如下图所示







总结

TCN在测试集上的表现远差于其在验证集上的表现，虽然其在验证集上表现更好，但与LSTM相比其训练时间长且训练效果差。

SVM：支持向量机

核函数选择径向基函数：

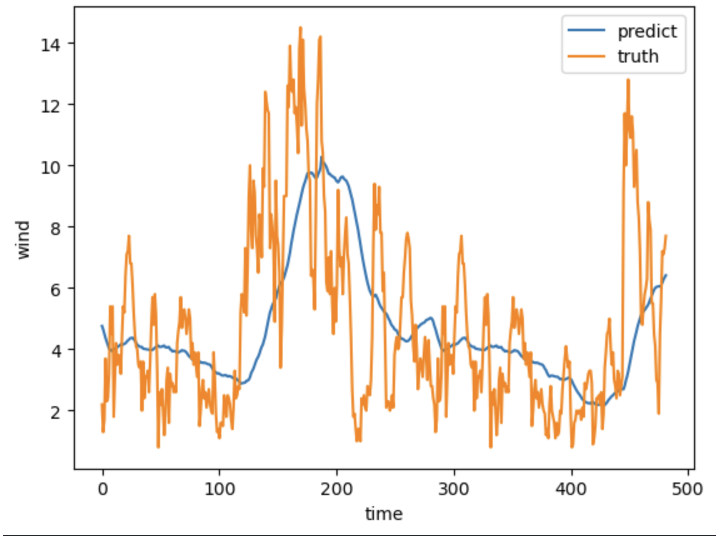
```
svr_rbf = SVR(kernel="rbf", C=100, gamma=0.1, epsilon=0.1)
```

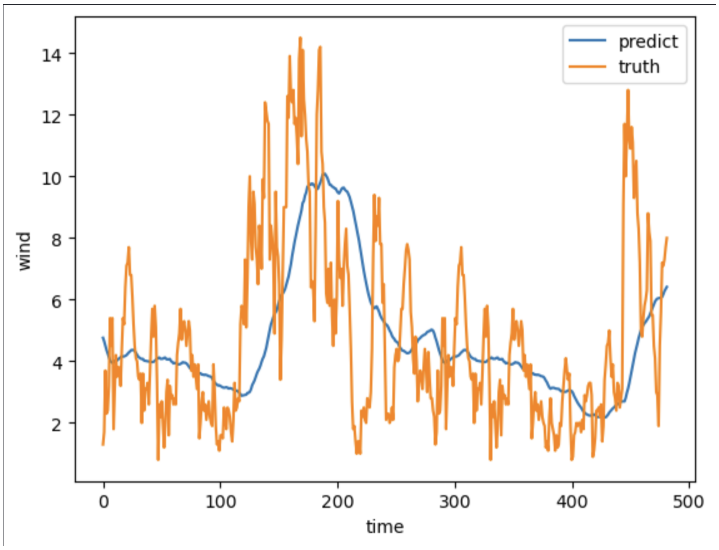
预测函数如下所示,依然使用滑动窗口法，窗口大小为50，步长为1，原数据、预测步长、评估方法、可视化策略等都与神经网络相同。

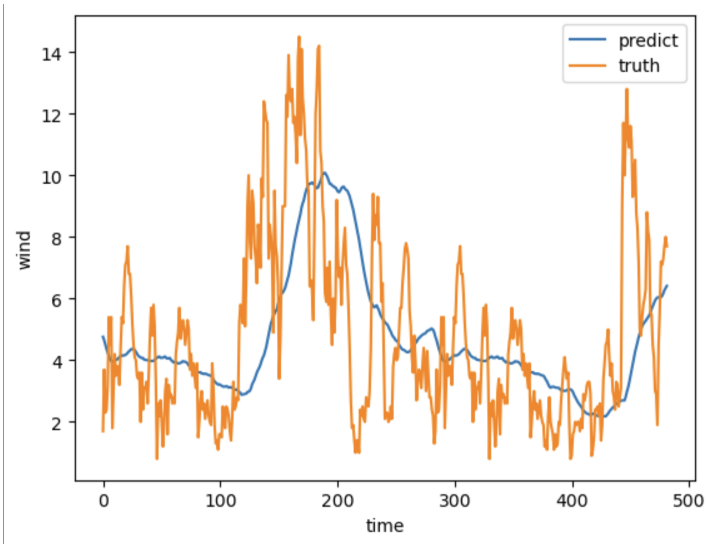
```
def predict(x,y,window_size,predict_step,func,eval,visulization):
    predict = []
    test = []
    for i in range(x.shape[0]-window_size-predict_step+1):
        temp_train_x = raw_data[i:i+window_size,:]
```

```
        temp_train_y = y[i:i+window_size]
        temp_test_x = raw_data[i+window_size:i+window_size+predict_step,:]
        temp_test_y = y[i+window_size:i+window_size+predict_step]
        temp_predict = func.fit(temp_train_x,temp_train_y.reshape(-1)).predict(temp_test_x)
        predict.append(temp_predict)
        test.append(temp_test_y)
    predict = np.array(predict).squeeze()
    test = np.array(test).squeeze()
    print(predict.shape)
    print(test.shape)
    [rmse, mae, mape] = eval(test,predict)
    print("rmse:",rmse,"mae:",mae,"mape:",mape)
    visulization(predict,test)
predict(x,y,50,3,svr_rbf,eval,visulization)
```

结果如图:







三种算法对比

	RMSE	MAE	MAPE
LSTM_MinMaxScale	2.1156738050232304	1.7285808324813843	25.489583611488342%
LSTM_StandardScale	**1.9344022659574036**	**1.549852967262268**	**23.301874101161957%**
TCN_MinMaxScale	2.9587482824026456	2.588508367538452	32.5937956571579%
TCN_StandardScale	6.710886341117575	6.45556640625	86.65335774421692%
SVM	2.692101691892707	2.007541401442317	56.64045941429213%

实验证明，用标准化预处理数据后送入LSTM的预测精度最高；

TCN过拟合情况严重；

SVM纯粹从数学角度进行拟合，效果稳定，但是不如LSTM。