

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，去掉五组气压相关的数据，对其余数据进行归一化处理，十四维数据作为特征，以以为一维数据(极大风速)为标签；

```
1 def __init__(self, predict_step=3, label=1):
2     self.raw_data =
3     np.array(pd.read_excel("weather.xlsx")).astype(float)
4     # 数据标准化
5     self.predict_step = predict_step
6     self.label = label
7     self.min = np.amin(self.raw_data)
8     self.max = np.amax(self.raw_data)
9     self.data = (self.raw_data - self.min) / (self.max - self.min)
```

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

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

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；

```
1     def construct_set(self, train_por=0.65, val_por=0.2, test_por=0.15,
2     window_size=50):
3         .....
4         for i in range(train_seqs.shape[0] - window_size -
5             self.predict_step+1):
6             train_seq = train_seqs[i:i+window_size+self.predict_step]
7             train_x.append(train_seq[0:window_size,:])
8
8             train_y.append(train_seq[window_size:window_size+self.predict_step, self.
9             label])
10
10            for i in range(val_seqs.shape[0] - window_size -
11                self.predict_step+1):
12                val_seq = val_seqs[i:i+window_size+self.predict_step]
13                val_x.append(val_seq[0:window_size,:])
```

```

11     val_y.append(val_seq[window_size:window_size+self.predict_step,self.label])
12
13         for i in range(test_seqs.shape[0] - window_size-
14             self.predict_step+1):
15             test_seq = test_seqs[i:i+window_size+self.predict_step]
16             test_x.append(test_seq[0:window_size,:])
17
18             test_y.append(test_seq[window_size:window_size+self.predict_step,self.label])
19
20     .....

```

## 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；

```

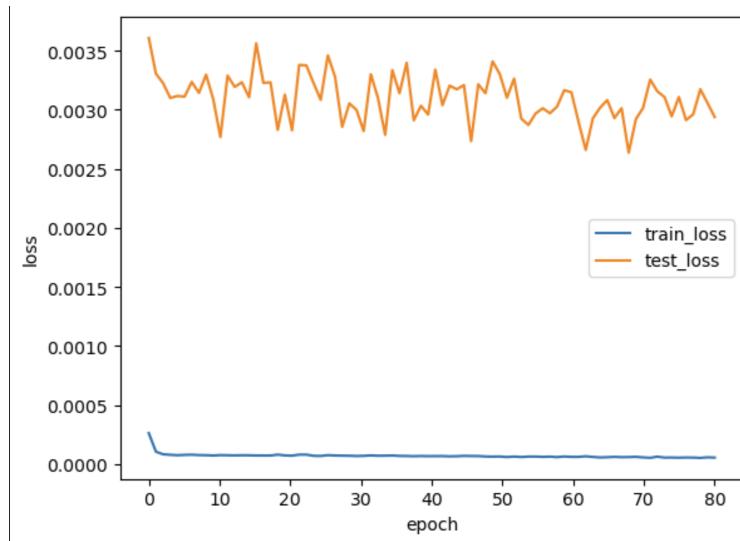
1 class LSTMNet(nn.Module):
2     def
3         __init__(self,batch_size,input_size,hidden_size,output_size,num_layers,se
4             q_len,device="cuda"):
5                 super().__init__()
6                 self.batch_size = batch_size
7                 self.input_size = input_size
8                 self.hidden_size = hidden_size
9                 self.output_size = output_size
10                self.num_layers = num_layers
11                self.seq_len = seq_len
12                self.lstm =
13                    nn.LSTM(input_size=input_size,hidden_size=hidden_size,num_layers=num_laye
14                        rs,batch_first=False)
15                    self.outlinear = nn.Sequential(
16
17                        nn.Linear(in_features=self.hidden_size,out_features=hidden_size//2),
18                            nn.LeakyReLU(),
19
20                        nn.Linear(in_features=hidden_size//2,out_features=self.output_size)
21
22                            )
23                            for param in self.parameters():
24                                if param.dim() > 1:
25                                    nn.init.xavier_uniform_(param)
26
27    def forward(self,x):

```

```
20     h, c =
21     (torch.zeros(self.num_layers, self.seq_len, self.hidden_size).to(x.device)
22      for _ in range(2))
23     H, (h, c) = self.lstm(x, (h, c))
24     out = H[:, -1, :].squeeze()
25     out = self.outlinear(out)
26     return out
```

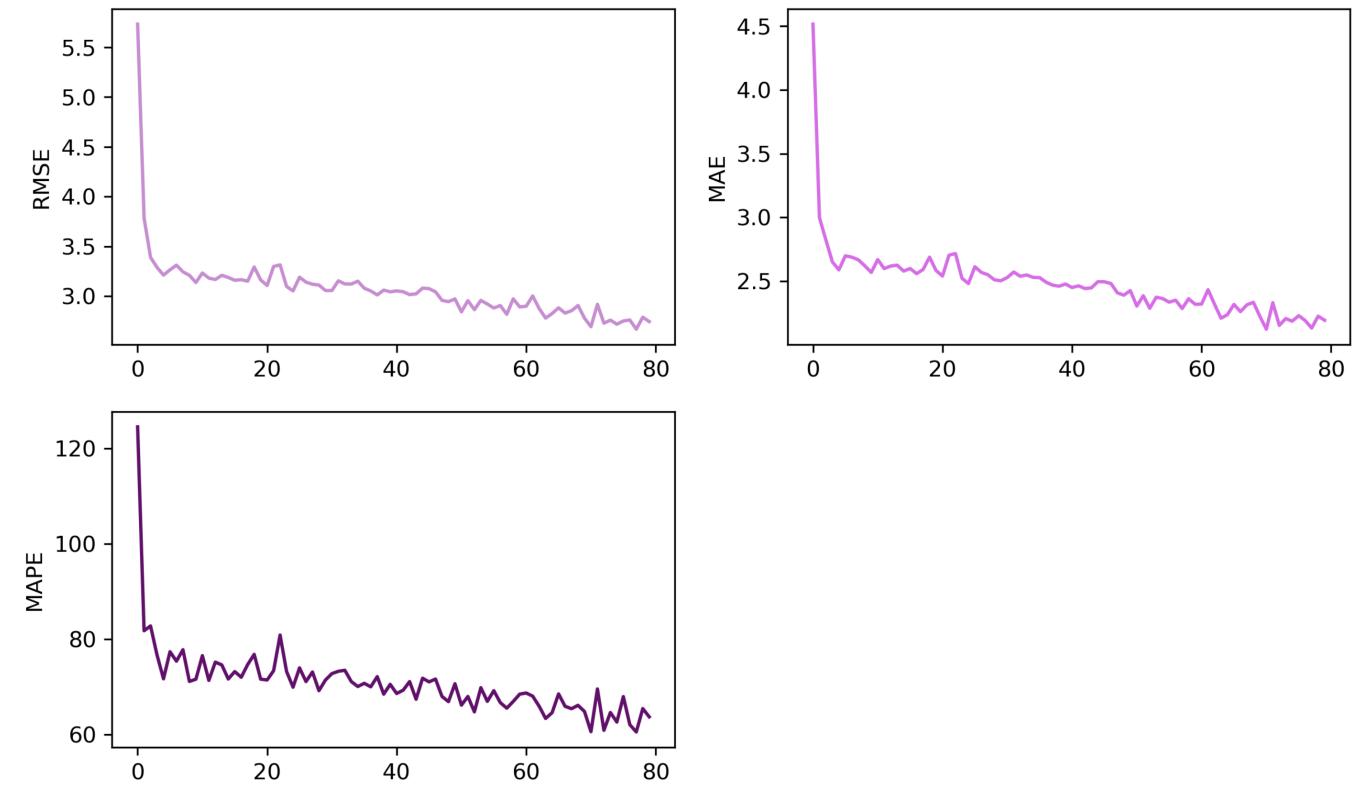
## 模型训练

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



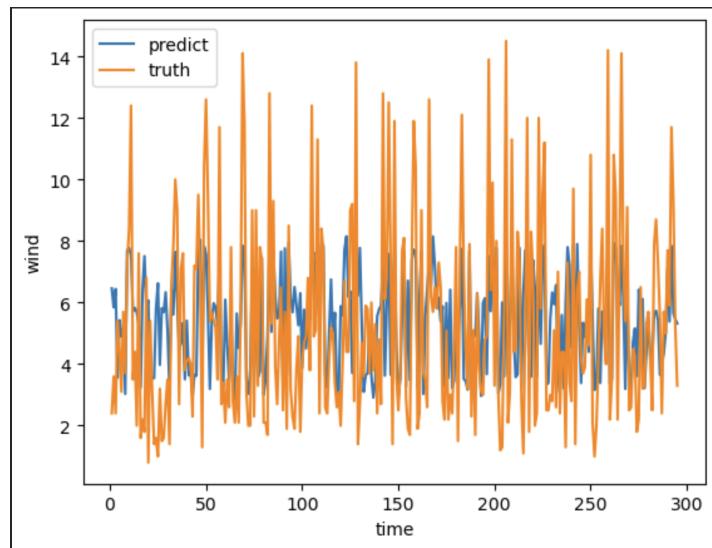
## 模型评估

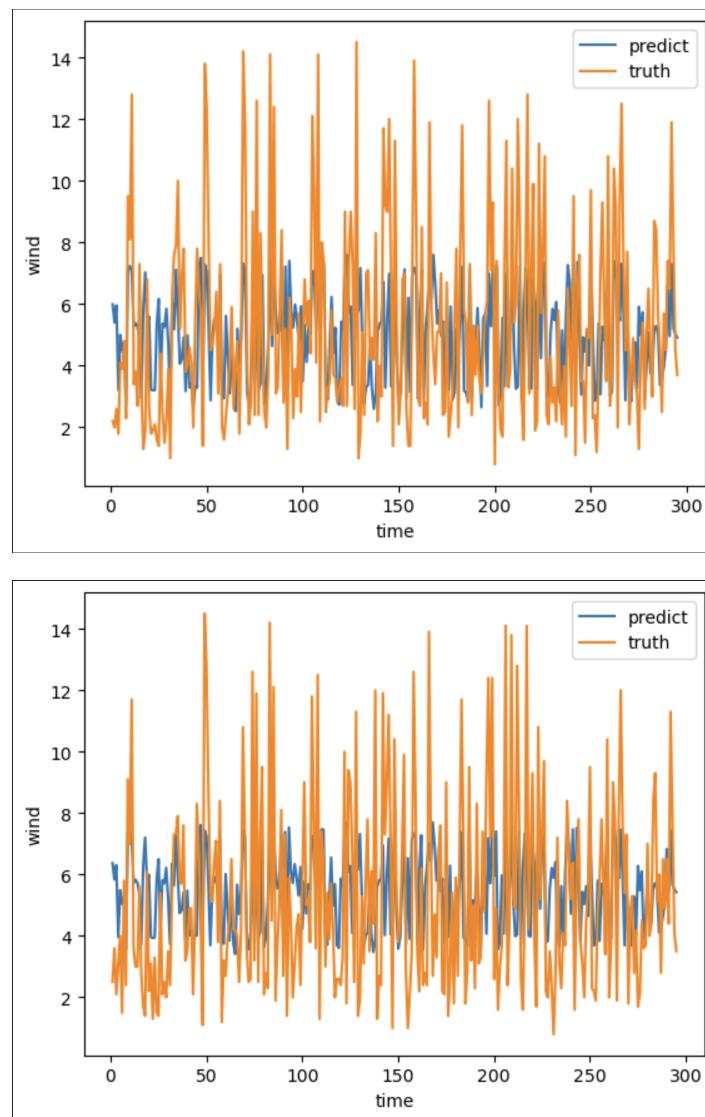
使用平均均方根误差RMSE，平均绝对误差MAE，平均绝对百分比误差MAPE来评估模型：



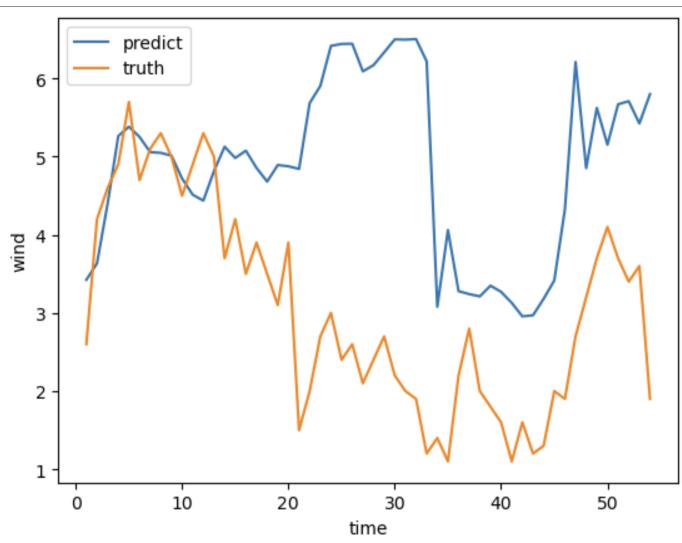
## 数据可视化

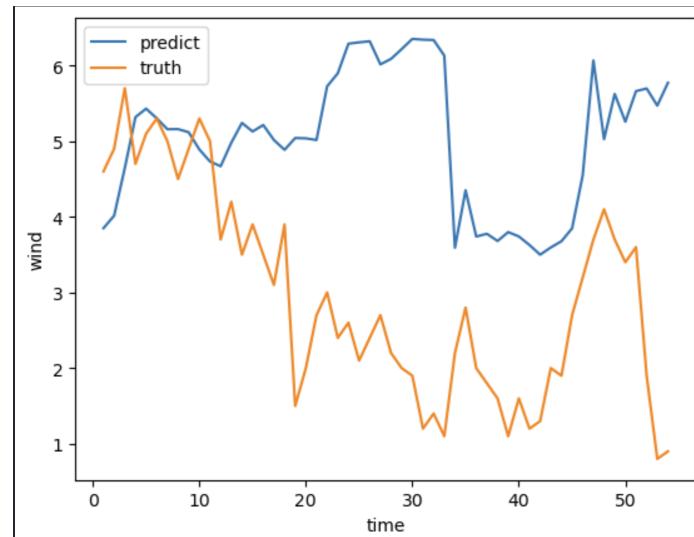
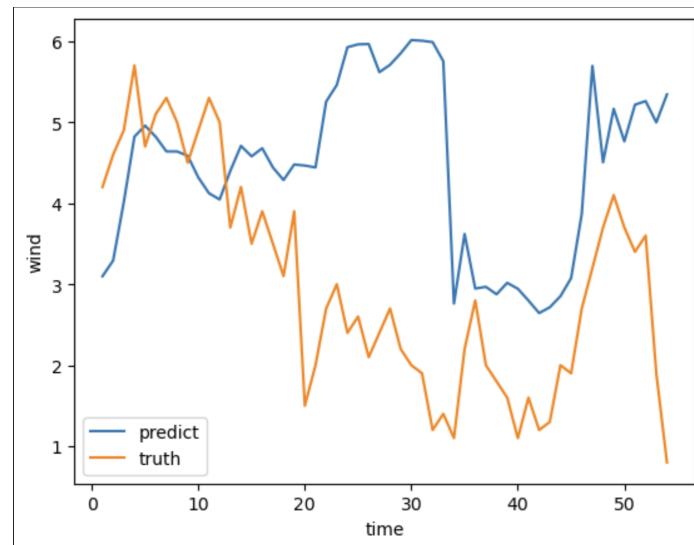
1h,2h,3h的预测值与真实值在训练集比较如下图所示



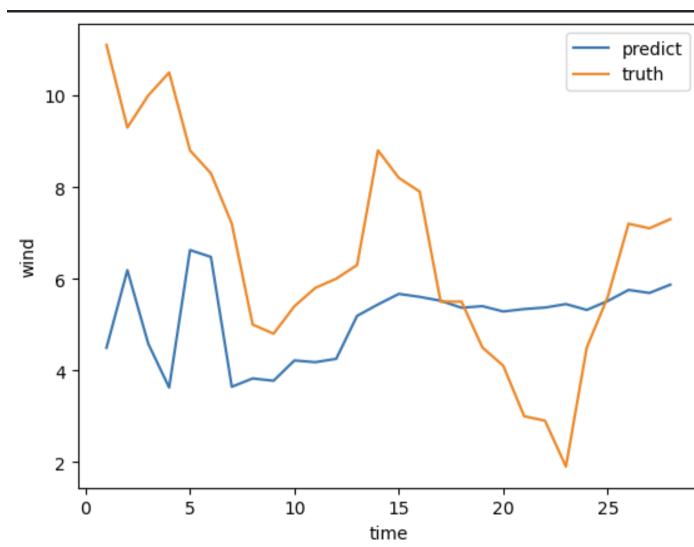


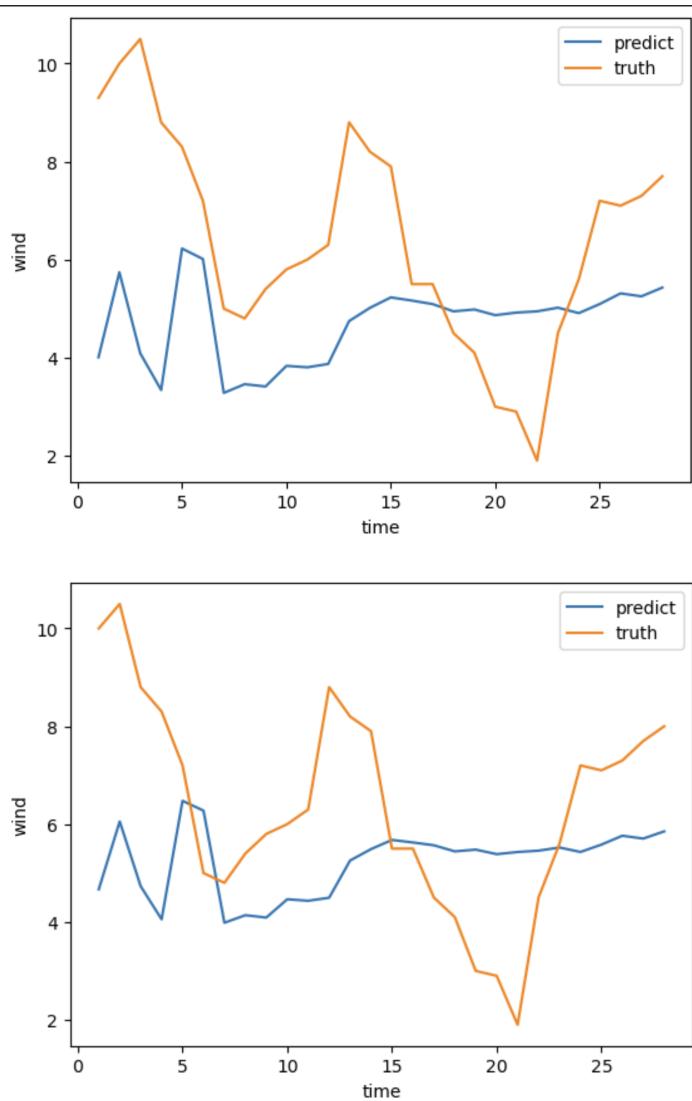
1h,2h,3h的预测值与真实值在验证集上的比较如下图所示





1h,2h,3h的预测值与真实值在测试集上的比较如下图所示





## 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。

```

1 | class Chomp1d(nn.Module):
2 |     def __init__(self, chomp_size):
3 |         super(Chomp1d, self).__init__()
4 |         self.chomp_size = int(chomp_size)
5 |     def forward(self, x):
6 |         return x[:, :, 0:-self.chomp_size].contiguous()
7 | #在padding与chomp浪费了大量时间为了使输出序列相等
8 | class TCNResidualBlock(nn.Module):
9 |     def __init__(self, inchannel, outchannel, kernelsize, dilation):

```

```

10     super(TCNResidualBlock, self).__init__()
11     self.ke = int(kernelsize+(kernelsize-1)*(dilation-1))
12     self.conv = nn.Sequential(
13
14         weight_norm(nn.Conv1d(inchannel,outchannel,kernel_size=kernelsize,dilati
15         on=dilation,padding=self.ke-1)),
16             Chomp1d(self.ke-1),
17             nn.ReLU(inplace=True),
18             nn.Dropout(p=0.2),
19
20         weight_norm(nn.Conv1d(outchannel,outchannel,kernel_size=kernelsize,dilat
21         ion=dilation,padding=self.ke-1)),
22             Chomp1d(self.ke-1),
23             nn.ReLU(inplace=True),
24             nn.Dropout(p=0.2),
25
26     self.conv1x1 = nn.Sequential(
27         weight_norm(nn.Conv1d(inchannel,outchannel,kernel_size=1)),
28     )
29
30     def forward(self,X):
31         Y = self.conv(X)
32         X = self.conv1x1(X)
33         out = F.relu(X+Y)
34
35         return out

```

```

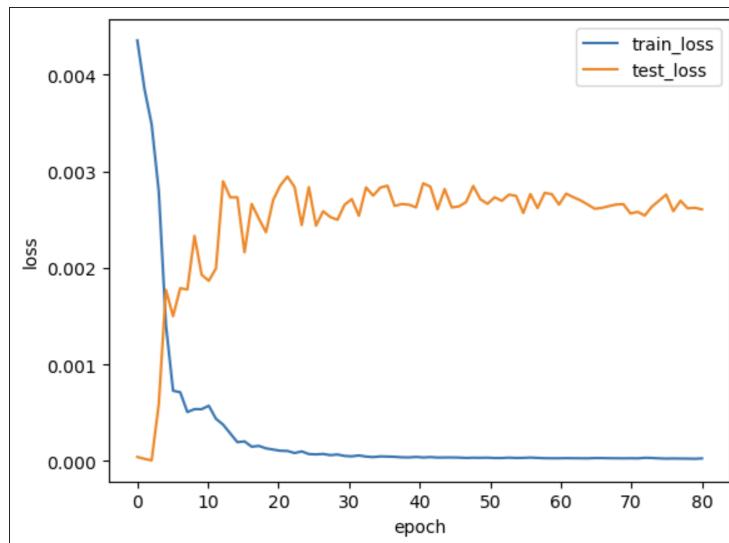
1 #通道数是特征的意思
2 #input: batch_size * features * seq_length但数据集的输入是batch_size *
3 #seq_length * features
4 block1 = TCNResidualBlock(14,32,5,1)
5 block2 = TCNResidualBlock(32,16,5,2)
6 block3 = TCNResidualBlock(16,8,3,4)
7
8 class TCNet(nn.Module):
9     def __init__(self,hidden_size,predict_step):
10         super(TCNet,self).__init__()
11         self.features = nn.Sequential(
12             block1,block2,block3
13         )
14         self.flatten = nn.Flatten()
15         self.regression = nn.Sequential(
16             nn.Linear(hidden_size, hidden_size//2),
17             nn.Dropout(p=0.2),
18             nn.ReLU(),
19             nn.Linear(hidden_size//2, predict_step),
20             nn.LeakyReLU()

```

```
20 )
21     for param in self.parameters():
22         if param.dim() > 1:
23             nn.init.xavier_uniform_(param)
24     def forward(self,X):
25         X = X.permute(0,2,1).contiguous()
26         features = self.features(X)
27         out = self.flatten(features)
28         out = self.regression(out).squeeze()
29         return out
```

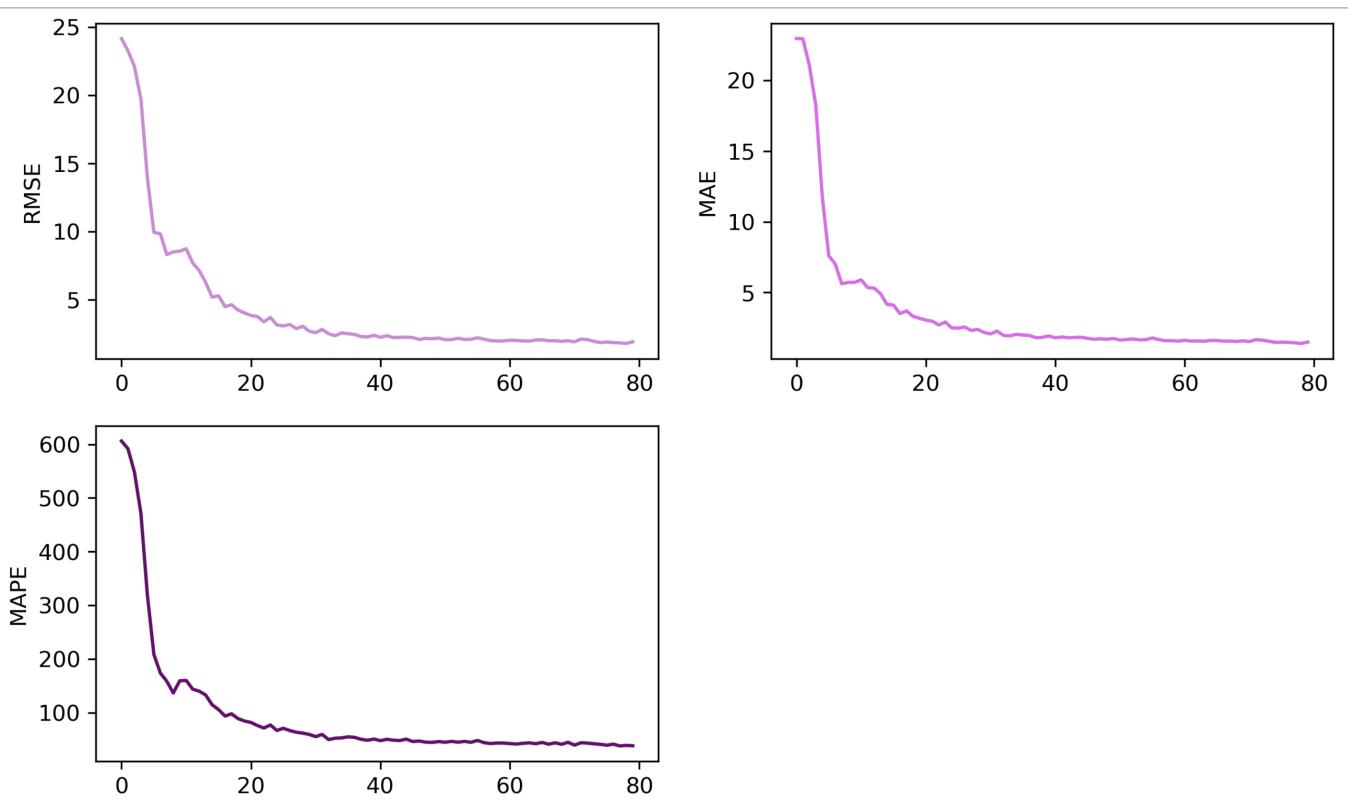
## 模型训练

取学习率为0.001，优化器选择Adam，训练轮次为80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



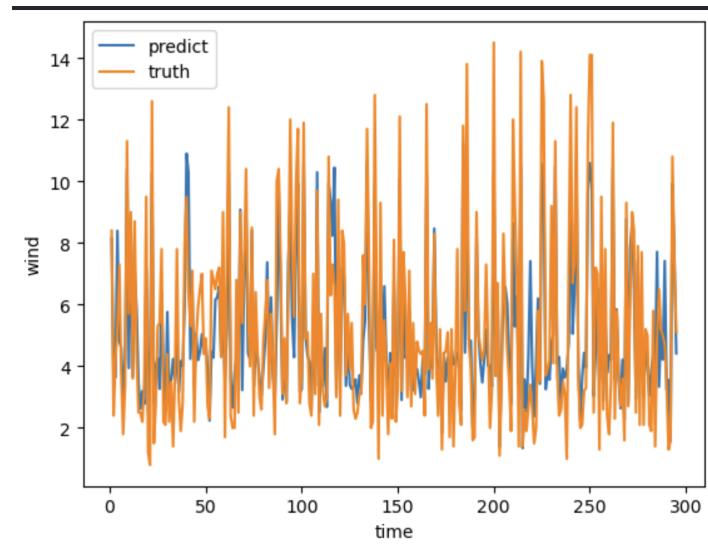
## 模型评估

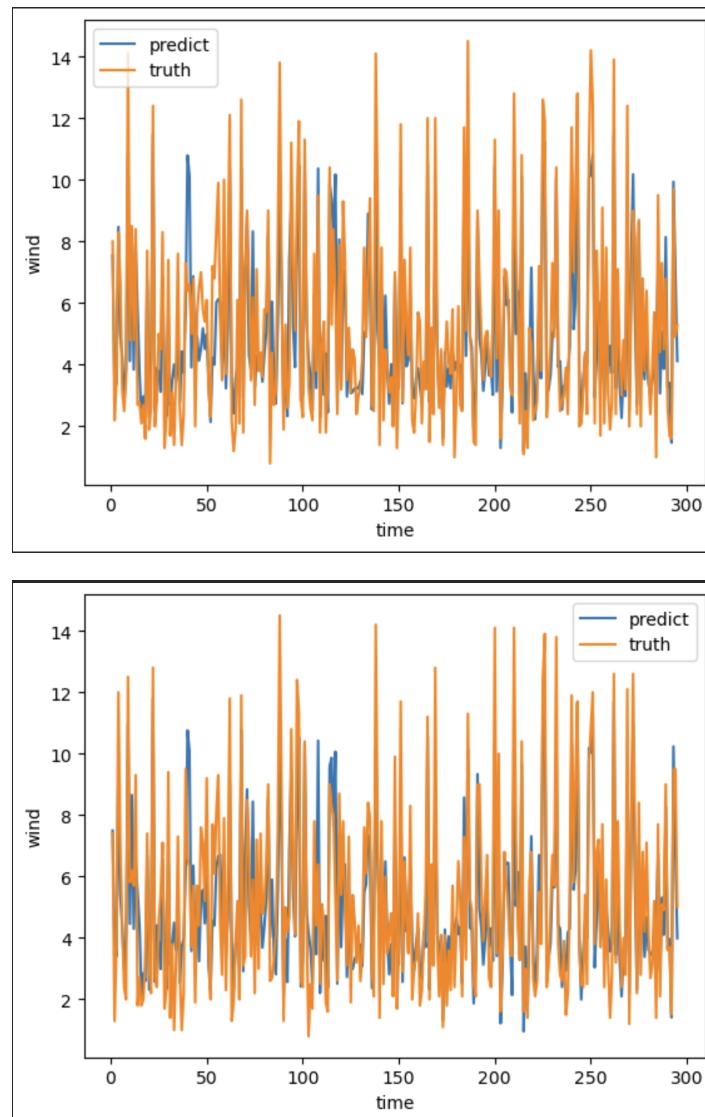
使用平均均方根误差RMSE，平均绝对误差MAE，平均绝对百分比误差MAPE来评估模型：



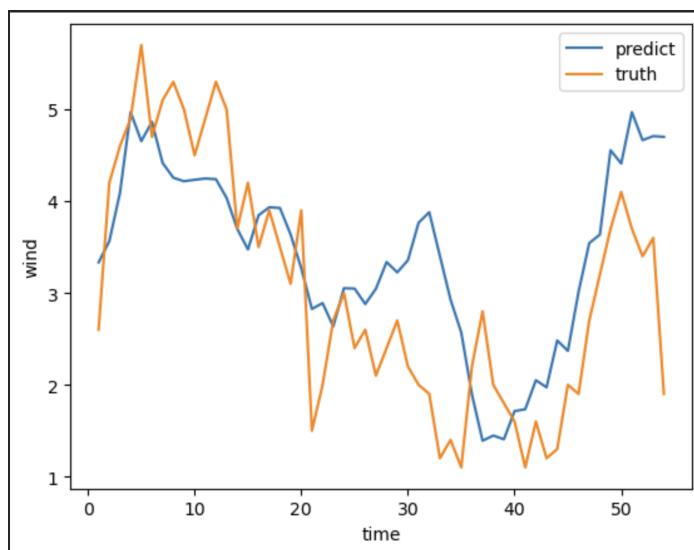
## 数据可视化

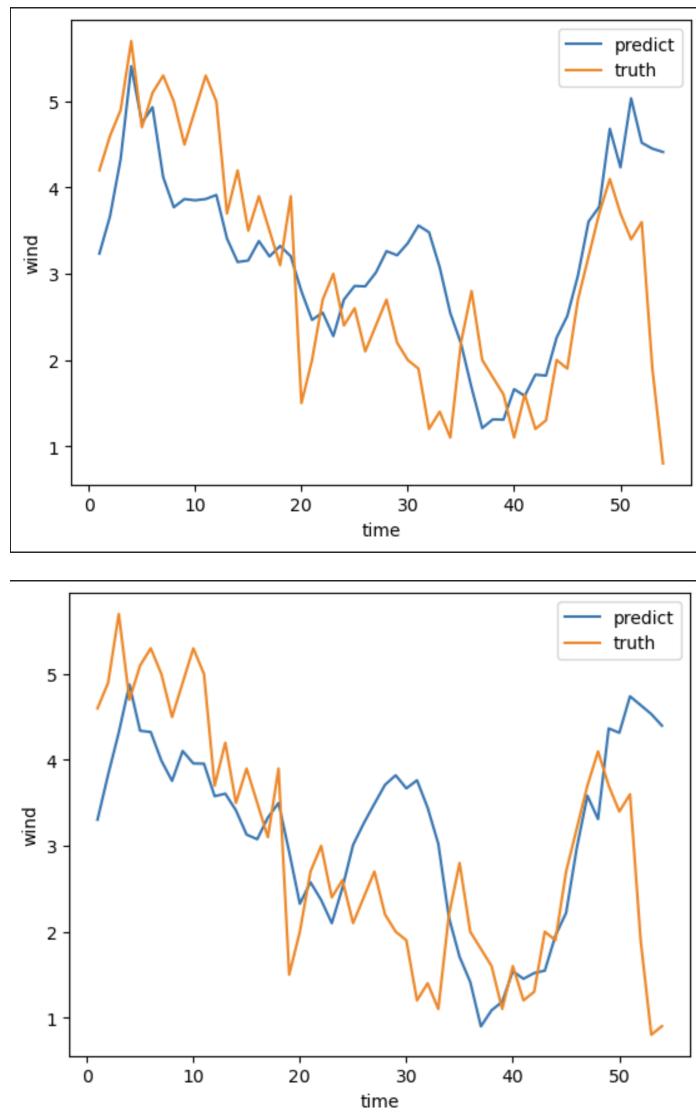
1h,2h,3h的预测值与真实值在训练集比较如下图所示



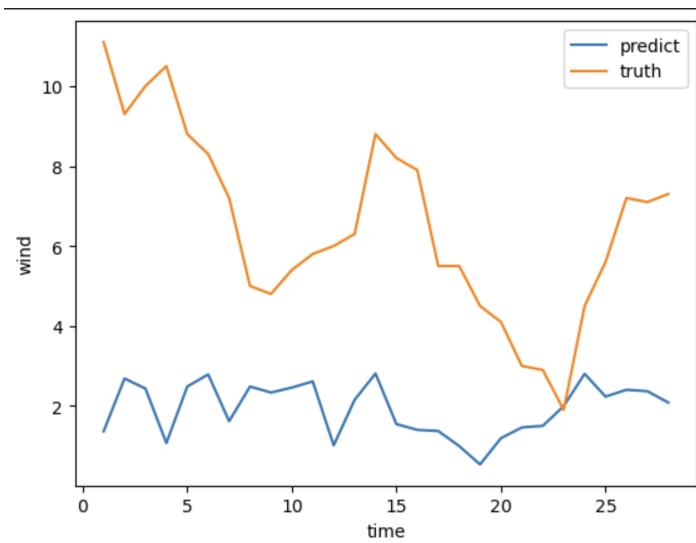


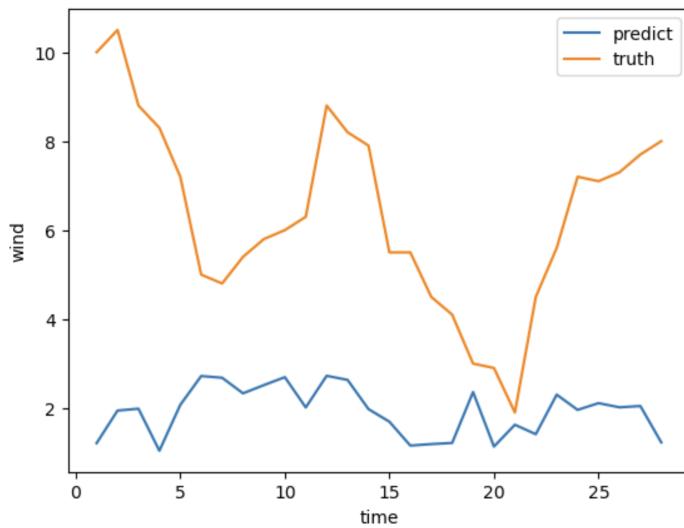
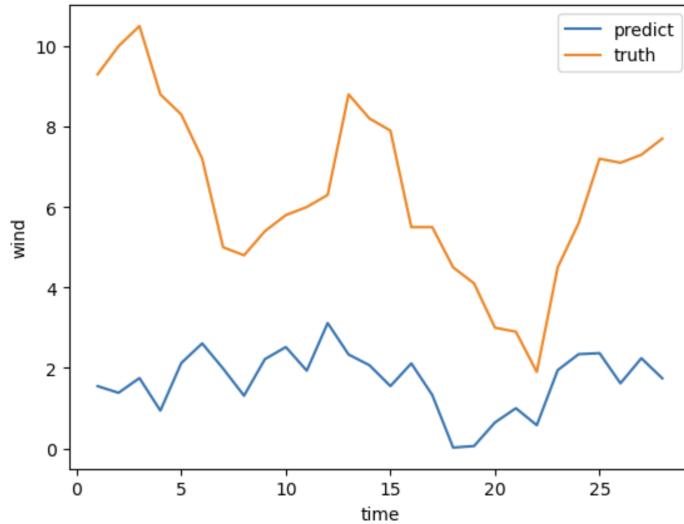
1h,2h,3h的预测值与真实值在验证集比较如下图所示





1h,2h,3h的预测值与真实值在测试集比较如下图所示





## SVM: 支持向量机

核函数选择径向基函数：

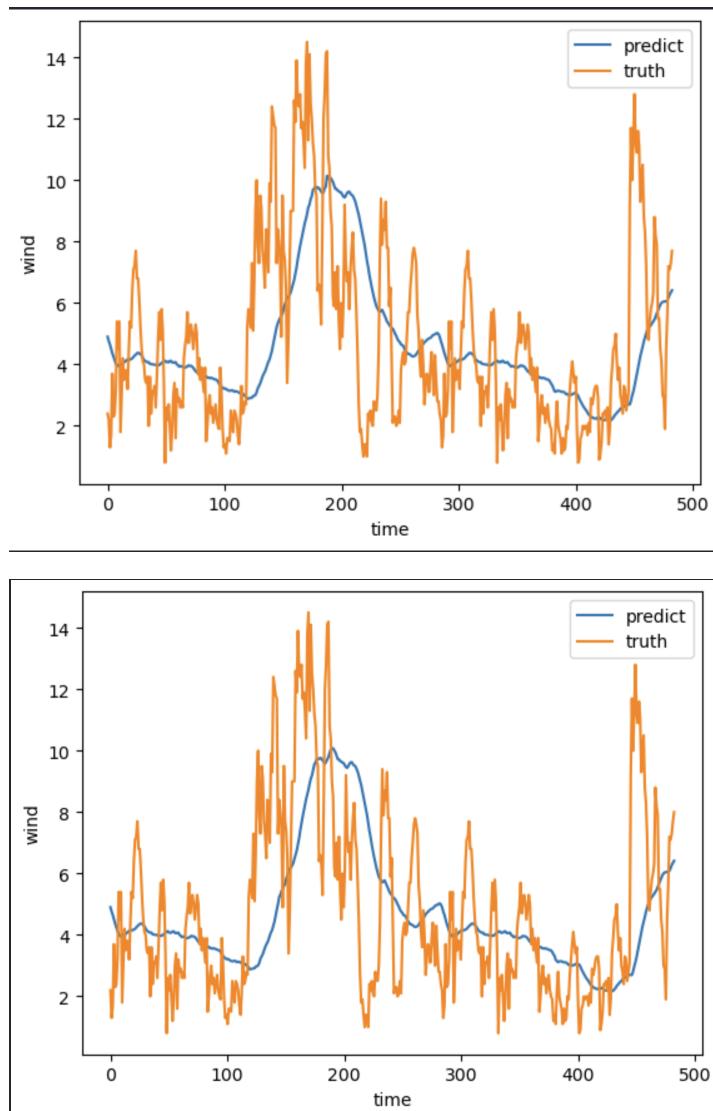
```
1 | svr_rbf = SVR(kernel="rbf", C=100, gamma=0.1, epsilon=0.1)
```

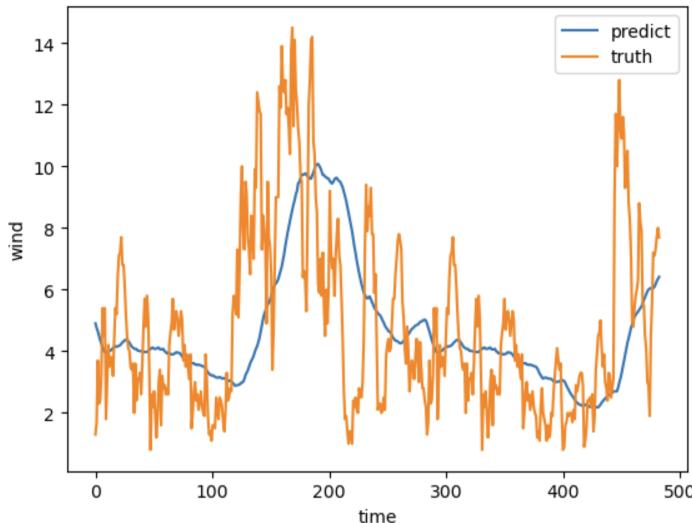
预测函数如下所示,依然使用滑动窗口法, 窗口大小为50, 步长为1, 原数据、预测步长、评估方法、可视化策略等都与神经网络相同。

```
1 | def predict(x,y>window_size,predict_step,func,eval,visulization):
2 |     predict = []
3 |     test = []
4 |     for i in range(x.shape[0]-window_size-predict_step+1):
5 |         temp_train_x = raw_data[i:i+window_size,:]
6 |         temp_train_y = y[i:i+window_size]
7 |         temp_test_x =
8 |         raw_data[i+window_size:i+window_size+predict_step,:]
9 |         temp_test_y = y[i+window_size:i+window_size+predict_step]
```

```
9     temp_predict =
10    func.fit(temp_train_x,temp_train_y.reshape(-1)).predict(temp_test_x)
11    predict.append(temp_predict)
12    test.append(temp_test_y)
13    predict = np.array(predict).squeeze()
14    test = np.array(test).squeeze()
15    print(predict.shape)
16    print(test.shape)
17    [rmse, mae, mape] = eval(test,predict)
18    print("rmse:",rmse,"mae:",mae,"mape:",mape)
19    visulization(predict,test)
predict(x,y,50,3,svr_rbf,eval,visulization)
```

结果如图：





三种方法对比

	<b>RMSE</b>	<b>MAE</b>	<b>MAPE</b>
LSTM	3.402497770937961	2.921602964401245	<b>36.95100545883179%</b>
TCN	5.746902294830505	5.355382919311523	70.02002596855164%
SVM	<b>2.6913743661902703</b>	<b>2.00855275156032</b>	56.84138900980481%

神经网络存在很大问题，虽然在测试集上能反映出大致的变换趋势，但在具体数值上存在较大误差；

TCN在验证集上的表现强于LSTM，但在测试集上不如LSTM，可能出现了过拟合。

SVM纯粹从数学角度进行拟合，效果稳定。