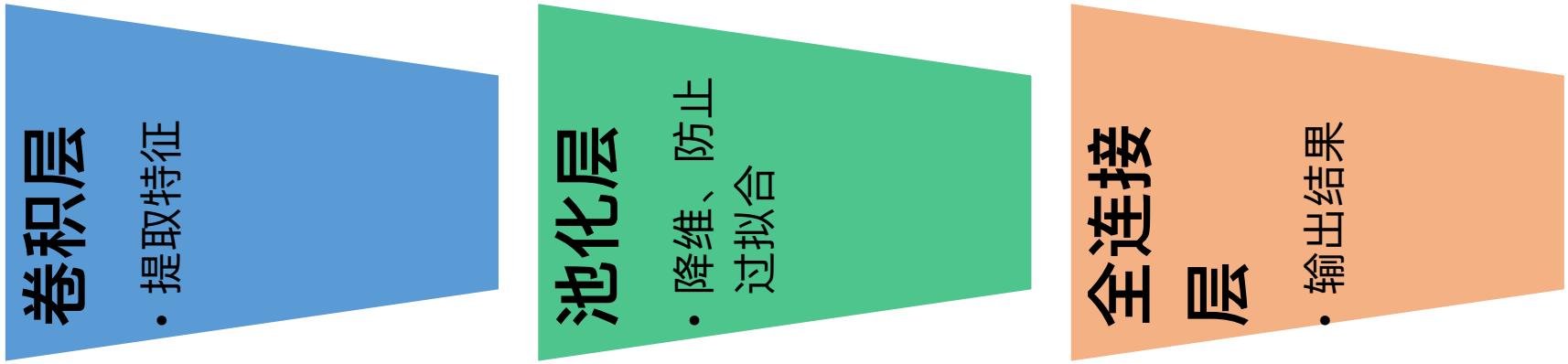
北京交通大学《时间序列数据分析挖掘》课程组

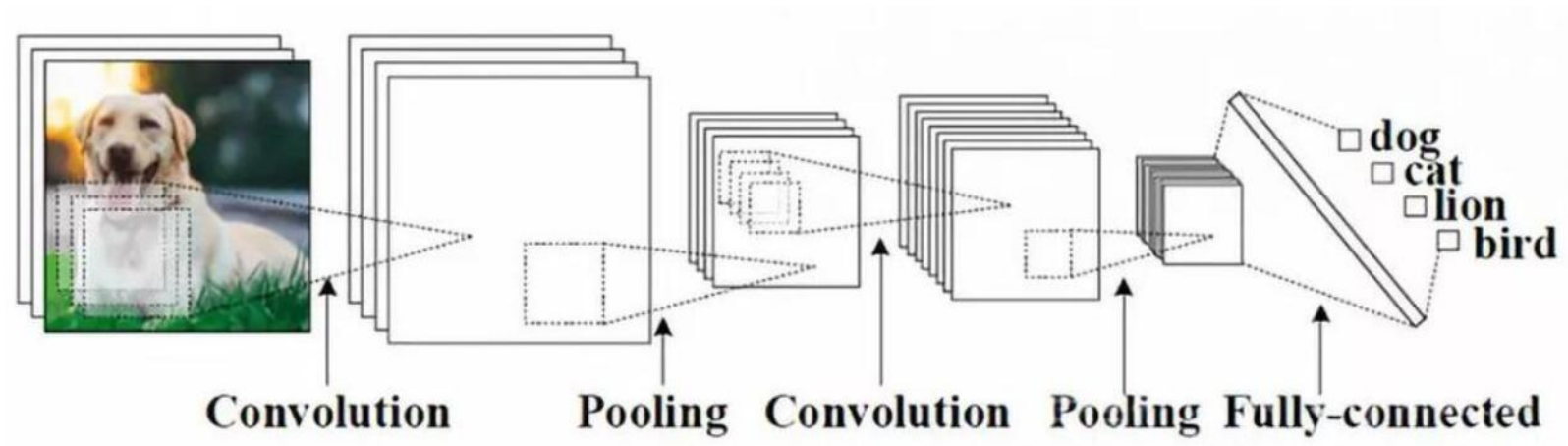
实验4 卷积神经网络

目 录

回顾-卷积神经网络

- 一般结构框架





一般流程

- 数据预处理，训练集、测试集划分；
- 设计损失函数；
- 构建神经网络，参数初始化；
- 在训练集上进行训练，反向传播，利用梯度下降法等更新网络参数，达到最小化损失函数的目的；
- 在测试集上进行评估网络性能

目 录

绘制序列

1. 导入所需要的包

```
import numpy as np
import torch.nn as nn
import torch
import matplotlib.pyplot as plt
import random
import torch.nn.functional as F
torch.set_default_tensor_type(torch.DoubleTensor)
```

- numpy用于数据处理；
- torch.nn用于构建网络；
- torch用于对tensor进行处理；
- 引入画图库matplotlib方便后续可视化；
- 将tensor的默认类型设置为双精度浮点类型，便于反向传播

2. 定义绘制序列函数

```
#绘制序列
def plot_series(time, series, format="-", start=0, end=None, label=None):
    #根据时间轴和对应数据列表绘制序列图像
    plt.plot(time[start:end], series[start:end], format, label=label)
    #设置横纵轴意义
    plt.xlabel("Time")
    plt.ylabel("Value")
    #设置图例说明字体大小
    if label:
        plt.legend(fontsize=14)
    #显示网格
    plt.grid(True)
```

- 以时间为横坐标、以序列值为纵坐标，设置曲线对应的label，将序列进行可视化；
- 设置横纵轴label；
- 画布显示网格，便于观察

趋势、噪声的生成函数

3. 趋势模式的生成函数

```
#趋势模式
def trend(time, slope=0):
    #序列与时间呈线性关系
    return slope * time
```

使序列的值与时间坐标呈线性关系来构造趋势；具体用参数slope来控制上升还是下降，陡峭还是平缓

4. 噪声模式的生成函数

```
#白噪声
def white_noise(time, noise_level=1, seed=None):
    #生成正态分布的伪随机数序列
    rnd = np.random.RandomState(seed)
    #noise_level控制噪声幅值大小
    return rnd.randn(len(time)) * noise_level
```

生成满足正态分布的序列，并用noise_level控制噪声幅值的大小

季节性的生成函数

5. 季节性模式的生成函数

```
#季节性（周期性）模式
def seasonal_pattern(season_time):
    """Just an arbitrary pattern, you can change it if you wish"""
    #分段函数(自变量取值[0,1])作为一个模式
    return np.where(season_time < 0.4,
                    np.cos(season_time * 2 * np.pi),
                    1 / np.exp(3 * season_time))

#将某个季节性（周期性）模式循环多次
def seasonality(time, period, amplitude=1, phase=0):
    """Repeats the same pattern at each period"""
    #将时间映射到0-1之间
    season_time = ((time + phase) % period) / period
    return amplitude * seasonal_pattern(season_time)
```

季节性的实现包括两步，第一步构造一个周期内的序列，第二步实现该序列的循环，用amplitude控制幅值大小

1. 划分训练集、测试集函数

```
#指定比例 划分训练集和测试集
def train_test_split(series,split_prop):
    train=series[:int(split_prop*int(series.size))]
    test=series[int(split_prop*int(series.size)):]
    return train,test
```

split_prop表示训练集所占的比例，一般取值为0.6、0.7或0.8等。

2. 划分batch函数

```
#将数据划分为指定大小的batch
def data_iter(batch_size, features, labels):
    num_examples= len(features)
    indices = list(range(num_examples))
    for i in range(0, num_examples, batch_size):
        # 最后一次可能不足一 个batch
        j = torch.LongTensor(indices[i: min(i+ batch_size, num_examples)])
        yield  features.index_select(0, j), labels.index_select(0, j)
```

3. 滑窗、划分特征/标签函数

```python
#滑窗、打乱
def data_process(train, test, window_size):  #滑窗 打乱等数据处理
    #将数据转为tensor并进行滑窗 得到短序列
    train_tensor=torch.from_numpy(train)
    train_window_split=train_tensor.unfold(0, window_size, 1)
    train_set=train_window_split.numpy()

    test_tensor=torch.from_numpy(test)
    test_window_split=test_tensor.unfold(0, window_size, 1)
    test_set=test_window_split.numpy()

    #将训练集短序列打乱
    train_temp1=train_set.tolist()
    random.shuffle(train_temp1)
    train_temp2=np.array(train_temp1)

    #将短序列划分为feature label
    train_feature_array=train_temp2[:,:window_size-1]
    train_label_array=train_temp2[:,window_size-1:]
    test_feature_array=test_temp2[:,:window_size-1]
    test_label_array=test_temp2[:,window_size-1:]

    #将ndarray转为tensor
    train_feature_tensor=torch.from_numpy(train_feature_array)
    train_label=torch.from_numpy(train_label_array)
    test_feature_tensor=torch.from_numpy(test_feature_array)
    test_label=torch.from_numpy(test_label_array)

    #扩展数据维度 符合CNN输入
    train_feature = train_feature_tensor.reshape(train_feature_tensor.shape[0], 1, train_feature_tensor.shape[1])
    test_feature = test_feature_tensor.reshape(test_feature_tensor.shape[0], 1, test_feature_tensor.shape[1])

    return train_feature, train_label, test_feature, test_label
```

扩展数据维度，使其符合1D-CNN的输入要求：

(batch_size, channel, length)

1. 生成混合模式的时序数据

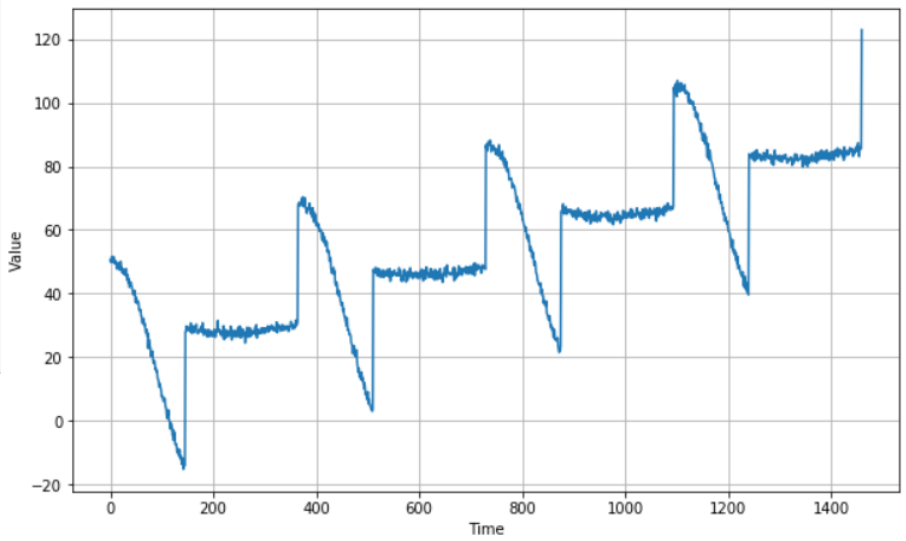
```
#生成序列并绘制图像
time = np.arange(4 * 365 + 1)
baseline = 10
slope = 0.05
amplitude = 40
noise_level = 1
series = baseline + trend(time, slope) + seasonality(time, period=365, amplitude=amplitude)+white_noise(time, noise_level, seed=42)

plt.figure(figsize=(10, 6))
plot_series(time, series)
plt.show()
```

2. 数据处理

```
split_prop=0.7  #设置划分比例
train,test=train_test_split(series,split_prop)  #按照划分比例划分训练集、测试集
window_size=6  #设置滑窗大小
#调用data_process进行数据处理
train_feature,train_label,test_feature,test_label=data_process(train,test,window_size)
#分别输出训练集、测试集的feature和label
print(train_feature.shape)
print(train_label.shape)
print(test_feature.shape)
print(test_label.shape)
```

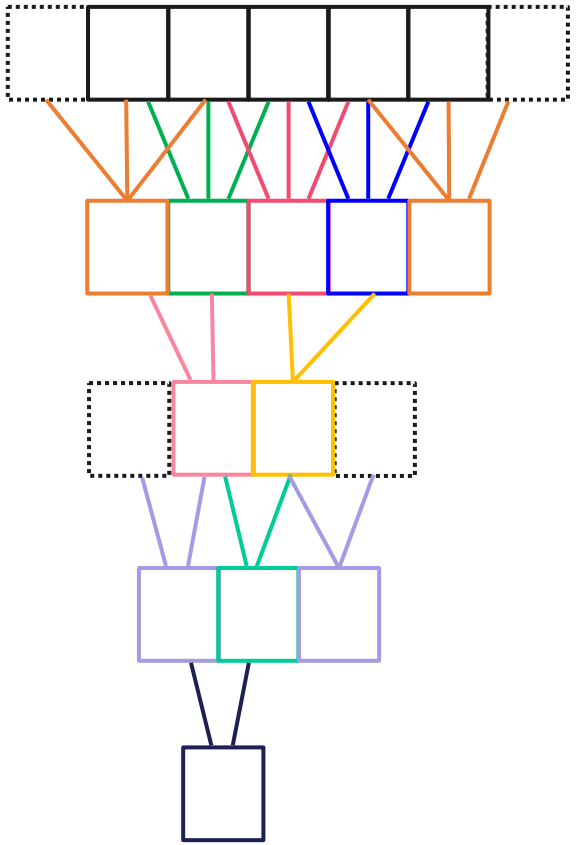
```
torch.Size([1017, 1, 5])
torch.Size([1017, 1])
torch.Size([434, 1, 5])
torch.Size([434, 1])
```



目 录

3. 构建一维卷积神经网络

```python
#定义卷积神经网络
class ConvModule1(nn.Module):
    def __init__(self):
        super(ConvModule1,self).__init__()
        #一层一维卷积
        self.conv1=nn.Sequential(
            nn.Conv1d(in_channels=1, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True)
        )
        self.conv2=nn.Sequential(
            nn.Conv1d(in_channels=32, out_channels=64, kernel_size=2, stride=1, padding=1),
            nn.ReLU(inplace=True)
        )
        #将输出通道变为单值
        self.fc1=nn.Linear(64,32)
        self.fc2=nn.Linear(32,1)
    def forward(self,X):
        out=self.conv1(X)  #一维卷积
        out=F.avg_pool1d(out,2)  #平均池化
        out=self.conv2(out)  #一维卷积
        out=F.avg_pool1d(out,2)  #平均池化
        out=out.squeeze()
        out=self.fc1(out)
        out=self.fc2(out)
        #out=self.fc3(out)
        #得到单值输出
        return out
#构建网络
net=ConvModule1()
```



1*5，padding：1

Conv kernel：1*3*32

stride：1

1*5*32

Avg_pool kernel：2

1*2*32，padding：1

Conv kernel：1*2*64

Stride：1

1*3*64

Avg_pool kernel：2

1*1*64

3. 构建一维卷积神经网络

损失函数-均方误差

```
def square_loss(feature, label):
    return (net(feature)-label)**2/2
```

$$loss = (\hat{y} - y)^2$$

参数初始化

```
#参数初始化
for params in net.parameters():
    torch.nn.init.normal_(params, mean=0, std=0.01)

lr= 0.0005   #学习率
num_epochs= 200   #训练轮数
batch_size=128   #batch_size大小
loss =square_loss   #损失函数
optimizer=torch.optim.Adam(net.parameters(), lr)   #设置优化器
```

4. 训练模型

```
train_loss=[]
test_loss=[]
#模型训练
for epoch in range(num_epochs):  #外循环训练一轮
    train_l,test_l=0.0,0.0
    for X,y in data_iter(batch_size,train_feature, train_label):  #内循环训练一个batch
        l=loss(X,y).sum()#计算模型输出与真实数据之间的差距
        #梯度清零
        if optimizer is not None:
            optimizer.zero_grad()
        elif params is not None and params[0].grad is not None:
            for param in params:
                param.grad.data.zero_()
        #反向传播
        l.backward()
        optimizer.step()
    #该轮训练结束后 计算目前网络再训练集、测试集上的损失 并输出
    train_l= loss((train_feature), train_label)
    test_l= loss((test_feature), test_label)
    train_loss.append(train_l.mean().item())
    test_loss.append(test_l.mean().item())
    print('epoch %d, train loss %f, test loss %f' % (epoch + 1, train_l.mean().item(),test_l.mean().item()))
```

完成网络初始化和训练参数的设置后，开始训练

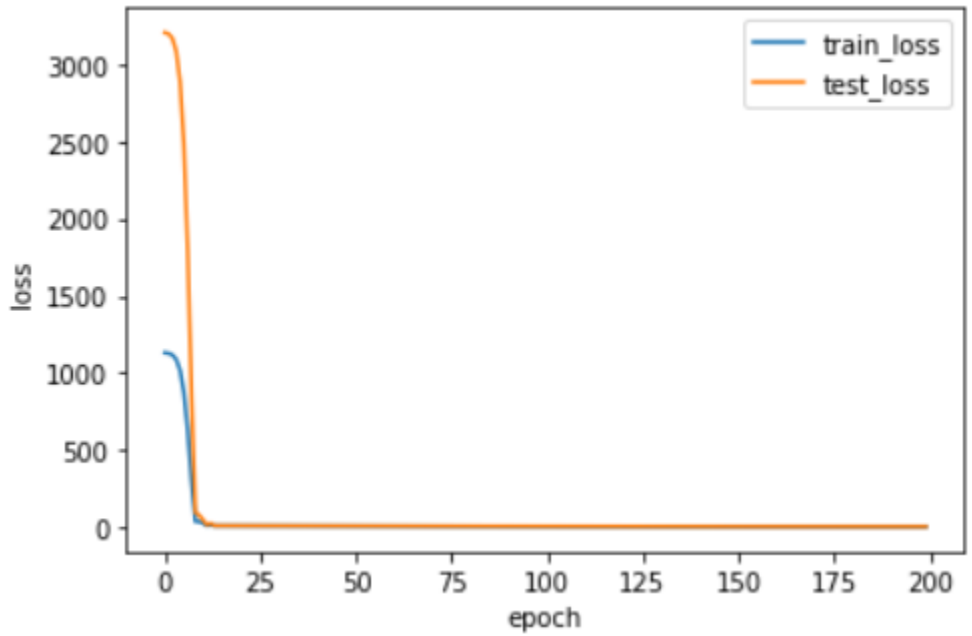
在每一轮训练中，对每一小批数据计算梯度，反向传播更新参数

每一轮训练后，计算并输出当前模型在训练集和

5. 绘制损失函数(loss)曲线

```
#绘制loss曲线
x=np.arange(num_epochs)
plt.plot(x, train_loss, label="train_loss", linewidth=1.5)
plt.plot(x, test_loss, label="test_loss", linewidth=1.5)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend()
plt.show()
```

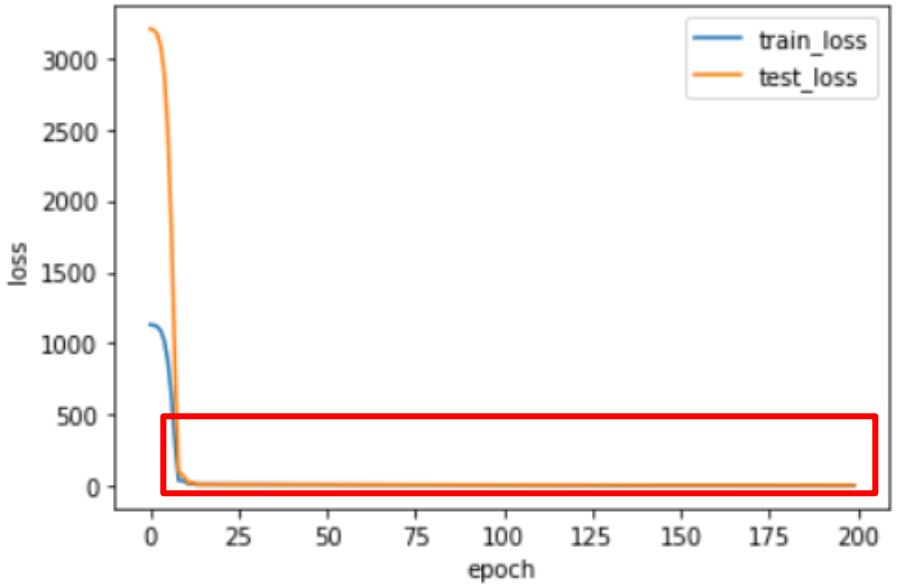
```
epoch 195, train loss 6.668962, test loss 7.911732
epoch 196, train loss 6.664293, test loss 7.905810
epoch 197, train loss 6.659668, test loss 7.899984
epoch 198, train loss 6.655089, test loss 7.894207
epoch 199, train loss 6.650558, test loss 7.888505
epoch 200, train loss 6.646086, test loss 7.882862
```

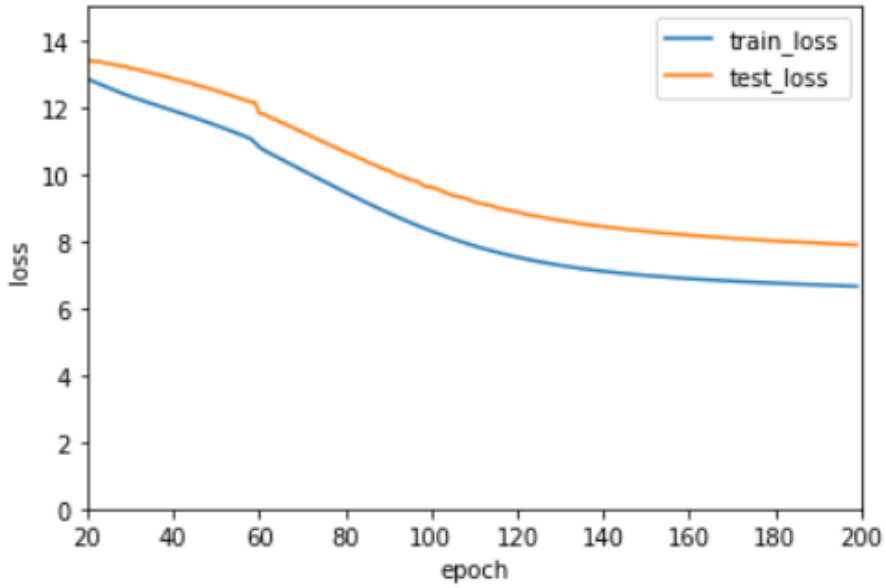


5. 绘制损失函数(loss)曲线

```
#绘制局部loss曲线
x=np.arange(num_epochs)
plt.plot(x, train_loss, label="train_loss", linewidth=1.5)
plt.plot(x, test_loss, label="test_loss", linewidth=1.5)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend()
plt.xlim(20, num_epochs)
plt.ylim(0, 15)
plt.show()
```

绘制局部曲线，观察变化情况





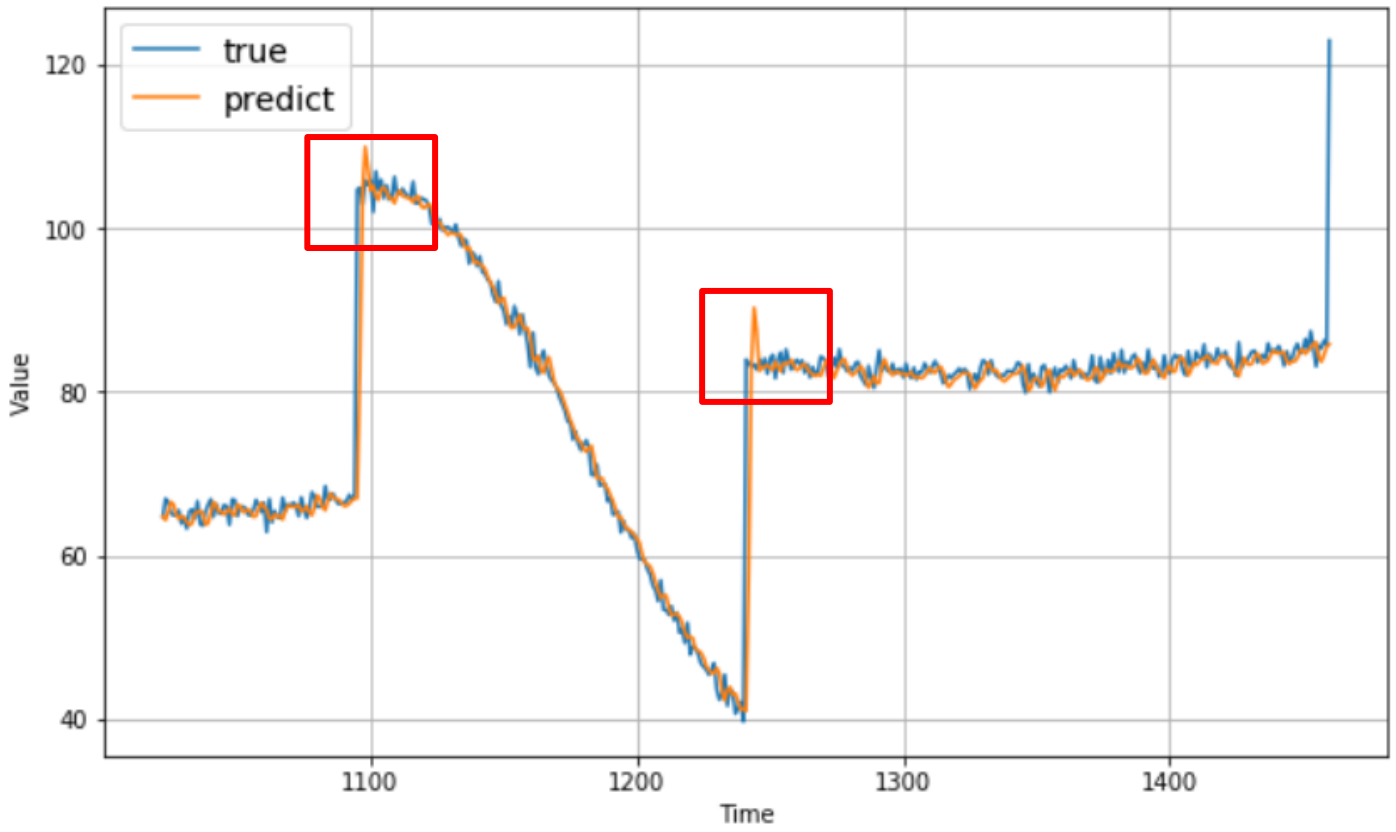
6. 预测并对比

```
def predict(x):
    temp = torch.from_numpy(x)
    x_tensor = temp.reshape(1, 1, window_size - 1)
    return net(x_tensor)
```

```
test_predict=[]
split_point=int(split_prop*int(series.size))
test_time=time[split_point+window_size-1:]
#测试集真实序列
test_true=series[split_point+window_size-1:]
#测试集预测序列
test_predict=net(test_feature).squeeze().tolist()
#用不同颜色画在一张图里 便于对比
plt.figure(figsize=(10, 6))
plot_series(test_time, test_true,label='true')
plot_series(test_time, test_predict,label='predict')
plt.show()
```

- 用训练好的模型在测试集上实现单值预测；
- 想要预测某时刻的值，将该时刻前面(window_size-1)个真实值作为网络输入，网络输出即为预测值；
- 将真实序列与预测序列用不同颜色绘制在同一张图里，评价网络性能

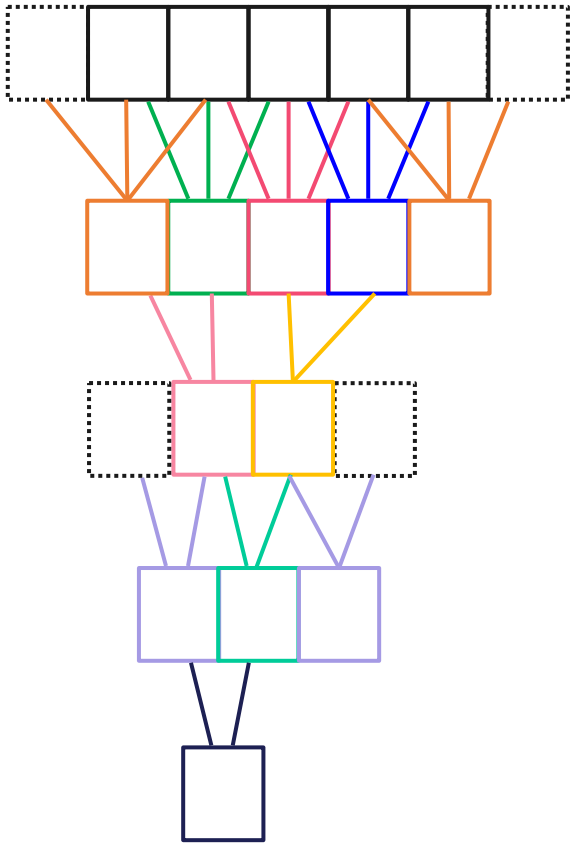
- ## 6. 预测并对比



- 其它模型

```python
#定义卷积神经网络
class ConvModule1(nn.Module):
    def __init__(self):
        super(ConvModule1,self).__init__()
        #一层一维卷积
        self.conv1=nn.Sequential(
            nn.Conv1d(in_channels=1, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True)
        )
        self.conv2=nn.Sequential(
            nn.Conv1d(in_channels=32, out_channels=64, kernel_size=2, stride=1, padding=1),
            nn.ReLU(inplace=True)
        )
        #将输出通道变为单值
        self.fc1=nn.Linear(64,32)
        self.fc2=nn.Linear(32,1)

    def forward(self,X):
        out=self.conv1(X)  #一维卷积
        out=F.max_pool1d(out,2)  #最大池化
        out=self.conv2(out)  #一维卷积
        out=F.max_pool1d(out,2)  #最大池化
        out=out.squeeze()
        out=self.fc1(out)
        out=self.fc2(out)
        #out=self.fc3(out)
        #得到单值输出
        return out
#构建网络
net=ConvModule1()
```



1*5，padding：1

Conv kernel：1*3*32

stride：1

1*5*32

max_pool kernel：2

1*2*32，padding：1

Conv kernel：1*2*64

Stride：1

1*3*64

max_pool kernel：2

1*1*64

CNN-模拟数据回归

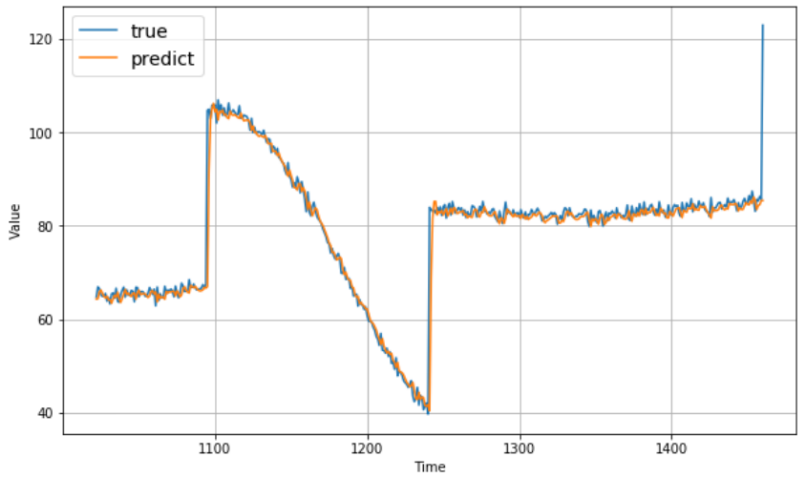
- **比较**

一层前馈神经网络

```
from sklearn.metrics import mean_absolute_error as mae
mae_nn=mae(test, test_predict)
print(mae_nn)
```

```
1.3744130469941527
```

```
epoch 195, train loss 5.089521, test loss 6.709623
epoch 196, train loss 5.086687, test loss 6.707826
epoch 197, train loss 5.083459, test loss 6.706043
epoch 198, train loss 5.080115, test loss 6.704247
epoch 199, train loss 5.077049, test loss 6.702497
epoch 200, train loss 5.073860, test loss 6.700808
```

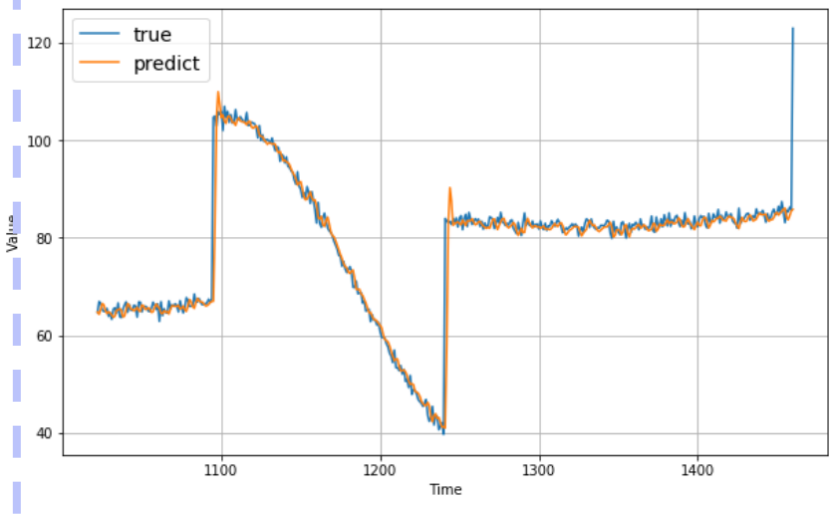


两层卷积神经网络（平均池

```
from sklearn.metrics import mean_absolute_error as mae
mae_nn=mae(test_true, test_predict)
print(mae_nn)
```

```
1.4410519066307985
```

```
epoch 195, train loss 6.668962, test loss 7.911732
epoch 196, train loss 6.664293, test loss 7.905810
epoch 197, train loss 6.659668, test loss 7.899984
epoch 198, train loss 6.655089, test loss 7.894207
epoch 199, train loss 6.650558, test loss 7.888505
epoch 200, train loss 6.646086, test loss 7.882862
```

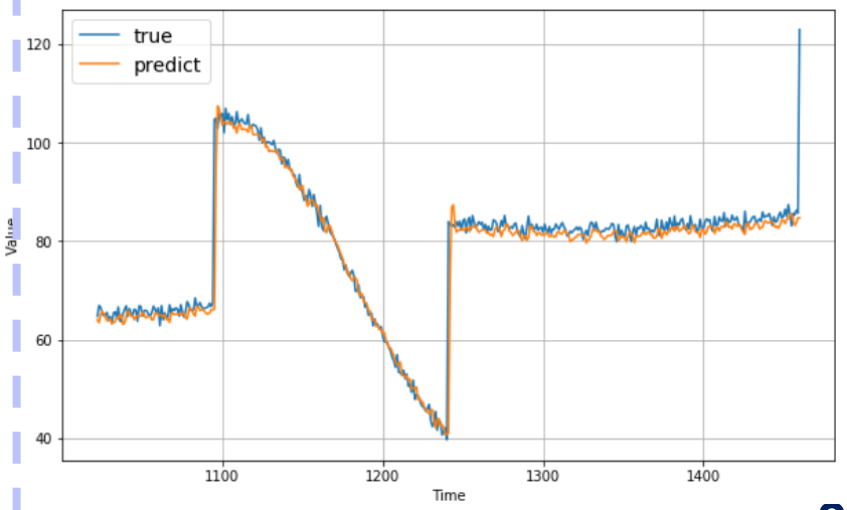


两层卷积神经网络（最大池

```
from sklearn.metrics import mean_absolute_error as mae
mae_nn=mae(test_true, test_predict)
print(mae_nn)
```

```
1.4921763508994446
```

```
epoch 195, train loss 5.702975, test loss 6.911098
epoch 196, train loss 5.696466, test loss 6.907323
epoch 197, train loss 5.689942, test loss 6.904433
epoch 198, train loss 5.683692, test loss 6.902232
epoch 199, train loss 5.677478, test loss 6.901237
epoch 200, train loss 5.671175, test loss 6.898108
```



目 录

1. 数据处理

从1956年至今的虹桥机场的气象数据，包括**气温**、干球温度、风速风向等等。

STATION	DATE	SOURCE	REPORT_	CALL_SIG	QUALITY_	AA1	AA2	AA3	AG1	AJ1	AL1	AY1	AY2	CALL_SIG	CIG	DEW	ED1	EQD	GA1	GA2	GA3	GA4	GE1
5.8E+10	1956-08-20T00:00:00	4	FM-12	99999	V020	06,0005,9	24,0050,9,1					8,1,99,9		99999	03600,1,C	+0250,1							
5.8E+10	1956-08-20T03:00:00	4	FM-12	99999	V020							6,1,99,9		99999	01230,1,C	+0250,1							
5.8E+10	1956-08-20T06:00:00	4	FM-12	99999	V020	06,0030,9,1						6,1,99,9		99999	00450,1,C	+0239,1							
5.8E+10	1956-08-20T09:00:00	4	FM-12	99999	V020							2,1,99,9		99999	00150,1,C	+0239,1							
5.8E+10	1956-08-20T12:00:00	4	FM-12	99999	V020	06,0006,9,1						4,1,99,9		99999	00270,1,C	+0228,1		Q01+000	08,1,+00270,9,07,9				
5.8E+10	1956-08-20T18:00:00	4	FM-12	99999	V020	06,0006,9,1						5,1,99,9		99999	22000,1,C	+0222,1		Q01+000002SCOTLCQ02+000042APCTENQ03+000042APC					
5.8E+10	1956-08-20T21:00:00	4	FM-12	99999	V020							2,1,99,9		99999	22000,1,C	+0222,1		Q01+000002SCOTLCQ02+000012APCTEN					
5.8E+10	1956-08-21T00:00:00	4	FM-12	99999	V020	24,0030,9,1						1,1,99,9		99999	22000,1,C	+0222,1		Q01+000042APCTENQ02+000152APC3					
5.8E+10	1956-08-21T03:00:00	4	FM-12	99999	V020							1,1,99,9		99999	00450,1,C	+0222,1							
5.8E+10	1956-08-21T06:00:00	4	FM-12	99999	V020							1,1,99,9		99999	00450,1,C	+0222,1							
5.8E+10	1956-08-21T09:00:00	4	FM-12	99999	V020							1,1,99,9		99999	22000,1,C	+0211,1							
5.8E+10	1956-08-21T12:00:00	4	FM-12	99999	V020							1,1,99,9		99999	22000,1,C	+0228,1		Q01+000042APCTENQ02+000112APC3					

GF1	HL1	IA1	IA2	KA1	KA2	MA1	MD1	ME1	MW1	MW2	MW3	OA1	OC1	QUALITY_	REM	REPORT_	SA1	SLP	SOURCE	TMP	UA1	UG1	VIS	WG1	WND
06,99,1,02,1,01,1,00450,1,04,1,02,1							6,1,005,1,	+999,9	01,1					V020		FM-12		10052,1	4	+0278,1			014000,1,N,9		200,1,N,0051,1
08,99,1,08,1,08,1,01250,1,00,1,00,1							0,1,003,1,	+999,9	60,1					V020		FM-12		10056,1	4	+0261,1			009000,1,N,9		250,1,N,0041,1
07,99,1,07,1,02,1,00450,1,07,1,07,1				999,N,+0250,1			9,9,015,1,	+999,9	21,1					V020		FM-12		10040,1	4	+0272,1			020000,1,N,9		250,1,N,0051,1
07,99,1,07,1,08,1,00150,1,03,1,07,1							6,1,002,1,	+999,9	02,1					V020		FM-12		10038,1	4	+0272,1			020000,1,N,9		270,1,N,0062,1
08,99,1,08,1,06,1,00250,1,00,1,00,1							2,1,024,1,	+999,9	50,1					V020		FM-12		10062,1	4	+0239,1			014000,1,N,9		270,1,N,0051,1
07,99,1,99,9,00,1,99999,9,05,1,00,1				999,M,+0278,1					03,1					V020		FM-12		10075,1	4	+0228,1			014000,1,N,9		290,1,N,0031,1
07,99,1,99,9,00,1,99999,9,07,1,00,1							4,1,000,1,	+999,9	02,1					V020		FM-12		10075,1	4	+0228,1			016000,1,N,9		320,1,N,0031,1
02,99,1,01,1,08,1,00450,1,04,1,00,1							2,1,015,1,	+999,9	03,1					V020		FM-12		10090,1	4	+0250,1			020000,1,N,9		320,1,N,0031,1
05,99,1,05,1,01,1,00450,1,04,1,00,1							0,1,004,1,	+999,9	02,1					V020		FM-12		10093,1	4	+0278,1			020000,1,N,9		290,1,N,0041,1
06,99,1,06,1,02,1,00450,1,00,1,00,1				999,N,+0228,1			6,1,011,1,	+999,9	02,1					V020		FM-12		10082,1	4	+0289,1			020000,1,N,9		270,1,N,0051,1
02,99,1,02,1,04,1,00450,1,00,1,00,1							6,1,002,1,	+999,9	01,1					V020		FM-12		10080,1	4	+0289,1			020000,1,N,9		320,1,N,0021,1
00,99,1,00,1,00,1,99999,9,00,1,00,1							2,1,012,1,	+999,9	04,1					V020		FM-12		10092,1	4	+0250,1			009000,1,N,9		090,1,N,0010,1

1. 数据处理

```
#读取原始文件 并将日期设置为索引
data=pd.read_csv("2362737.csv",index_col="DATE",na_values="+9999,9")
#选取气温（TMP）一列
data=data["TMP"]
#将data.index设置为时间格式
data.index=pd.to_datetime(data.index)
#设置开始及结束时间 选取一段数据
start_time=pd.to_datetime("2019-01-01 00:00:00")
end_time=pd.to_datetime("2019-06-30 23:00:00")
data=data[start_time:end_time]
#从data中剔除TMP列为NaN的数据
data=data.dropna()
#将TMP数据从str转为int 并获取正确的数值
data=data.str.split(",",expand=True)[0]
data=data.astype("int")/10
#将时间补全（前面有丢弃操作） 并将间隔设置为1h 重新设置data的索引
time=pd.date_range(start=start_time,end=end_time,freq="H")
data=data.reindex(time)
#进行插值（部分补全的时间没有对应的数据）
data=data.interpolate()
#将数据转为array类型 与前面实验相同
series=np.array(data)
```

1. 获取气温数据；

2. 将index设置为时间格式，

截取数据片段

3. 丢弃TMP的NaN数据；

4. 获取气温数值；

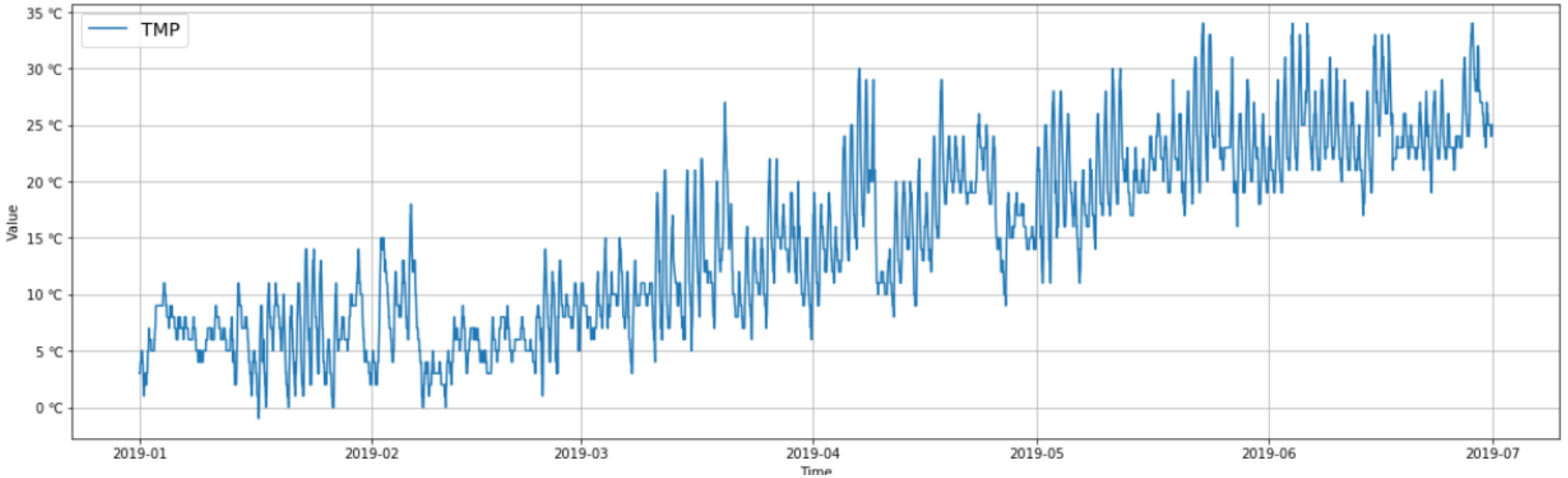
5. 补全DATE，设置间隔；

6. 对NaN插值；

1. 数据处理

调用绘制序列函数进行可视化

```
import matplotlib.ticker as mticker
#数据可视化
fig,ax=plt.subplots(figsize=(20, 6))
#设置纵轴单位
ax.yaxis.set_major_formatter(mticker.FormatStrFormatter('%d ℃'))
plot_series(time, series,label='TMP')
plt.show()
```



1. 数据处理

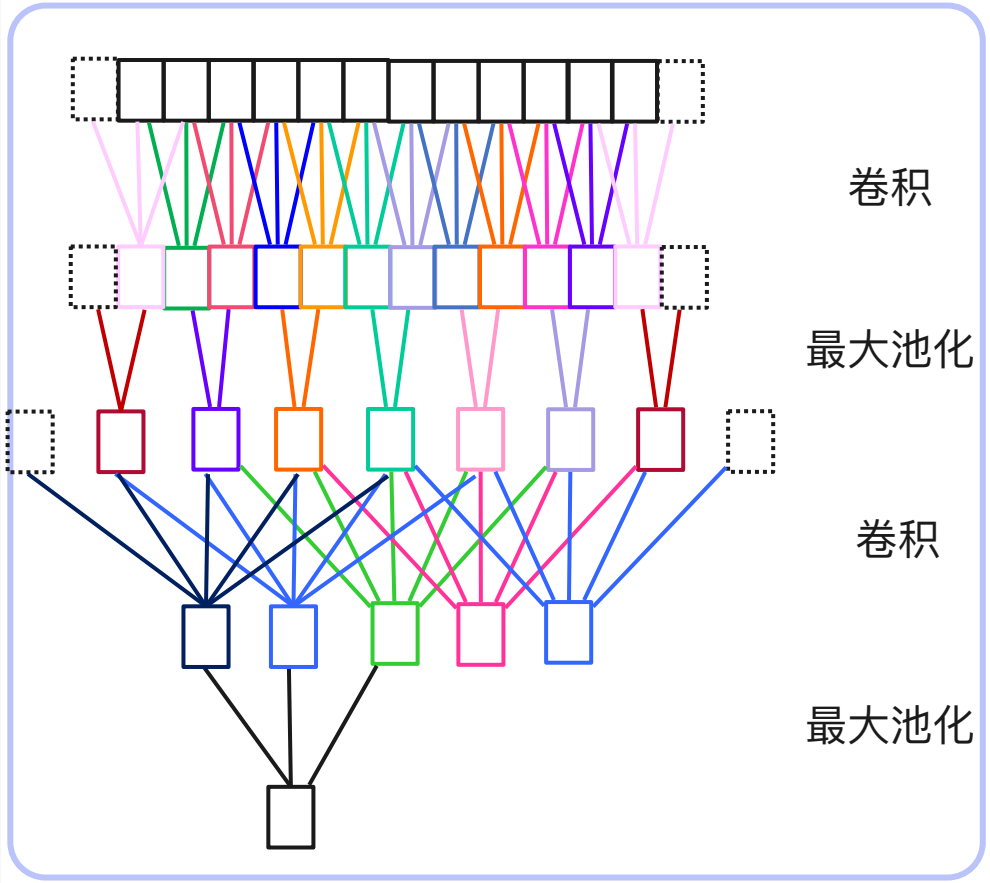
```
split_prop=0.7  #设置划分比例
train,test=train_test_split(series,split_prop)  #按照划分比例划分训练集、测试集
window_size=13  #设置滑窗大小
#调用data_process进行数据处理
train_feature,train_label,test_feature,test_label=data_process(train,test,window_size)
#分别输出训练集、测试集的feature和label
print(train_feature.shape)
print(train_label.shape)
print(test_feature.shape)
print(test_label.shape)
```

```
torch.Size([3028, 1, 12])
torch.Size([3028, 1])
torch.Size([1292, 1, 12])
torch.Size([1292, 1])
```

2. 构建一维卷积神经网络

```
#定义卷积神经网络
class ConvModule1(nn.Module):
    def __init__(self):
        super(ConvModule1,self).__init__()
        #两层一维卷积
        self.conv1=nn.Sequential(
            nn.Conv1d(in_channels=1, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True)
        )
        self.conv2=nn.Sequential(
            nn.Conv1d(in_channels=32,out_channels=64,kernel_size=5,stride=1,padding=1),
            nn.ReLU(inplace=True)
        )
        #线性层
        self.fc1=nn.Linear(64,32)
        self.fc2=nn.Linear(32,1)

    def forward(self,X):
        out=self.conv1(X)   #一维卷积
        out=F.max_pool1d(out,kernel_size=2,padding=1)   #最大池化
        out=self.conv2(out)   #一维卷积
        out=F.max_pool1d(out,3)   #最大池化
        out=out.squeeze()
        out=self.fc1(out)
        out=self.fc2(out)
        #得到单值输出
        return out
#构建网络
net=ConvModule1()
```



2. 构建一维卷积神经网络

损失函数-均方误差

```
def square_loss(feature, label):
    return (net(feature)-label)**2/2
```

$$loss = (\hat{y} - y)^2$$

参数初始化

```
#参数初始化
for params in net.parameters():
    torch.nn.init.normal_(params, mean=0, std=0.01)

lr= 0.001   #学习率
num_epochs= 100   #训练轮数
batch_size=128   #batch_size大小
loss =square_loss   #损失函数
optimizer=torch.optim.Adam(net.parameters(),lr)   #设置优化器
```

3. 训练模型

```
train_loss=[]
test_loss=[]
#模型训练
for epoch in range(num_epochs):  #外循环训练一轮
    train_l,test_l=0.0,0.0
    for X,y in data_iter(batch_size,train_feature, train_label):  #内循环训练一个batch
        l=loss(X,y).sum()  #计算模型输出与真实数据之间的差距
        #梯度清零
        if optimizer is not None:
            optimizer.zero_grad()
        elif params is not None and params[0].grad is not None:
            for param in params:
                param.grad.data.zero_()
        #反向传播
        l.backward()
        optimizer.step()
    #该轮训练结束后 计算目前网络在训练集、测试集上的损失 并输出
    train_l= loss((train_feature), train_label)
    test_l= loss((test_feature), test_label)
    train_loss.append(train_l.mean().item())
    test_loss.append(test_l.mean().item())
    print('epoch %d, train loss %f, test loss %f' % (epoch + 1, train_l.mean().item(),test_l.mean().item()))
```

完成网络初始化和训练参数的设置后，开始训练

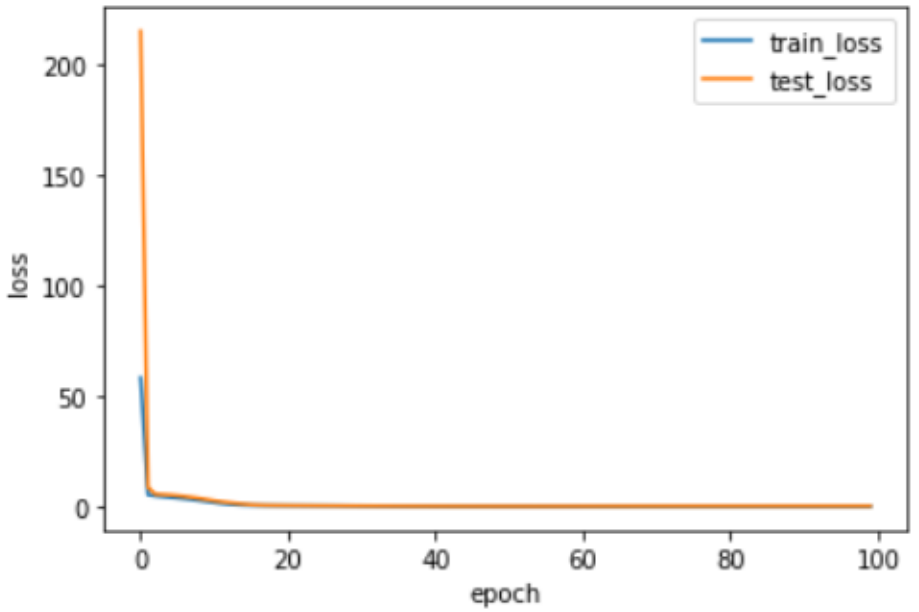
在每一轮训练中，对每一小批数据计算梯度，反向传播更新参数

每一轮训练后，计算并输出当前模型在训练集和

4. 绘制损失函数(loss)曲线

```
epoch 95, train loss 0.544907, test loss 0.662723
epoch 96, train loss 0.544843, test loss 0.663157
epoch 97, train loss 0.544681, test loss 0.663106
epoch 98, train loss 0.544343, test loss 0.662590
epoch 99, train loss 0.544268, test loss 0.663129
epoch 100, train loss 0.544192, test loss 0.663911
```

```
#绘制loss曲线
x=np.arange(num_epochs)
plt.plot(x,train_loss,label="train_loss",linewidth=1.5)
plt.plot(x,test_loss,label="test_loss",linewidth=1.5)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.legend()
plt.show()
```



5. 预测并对比

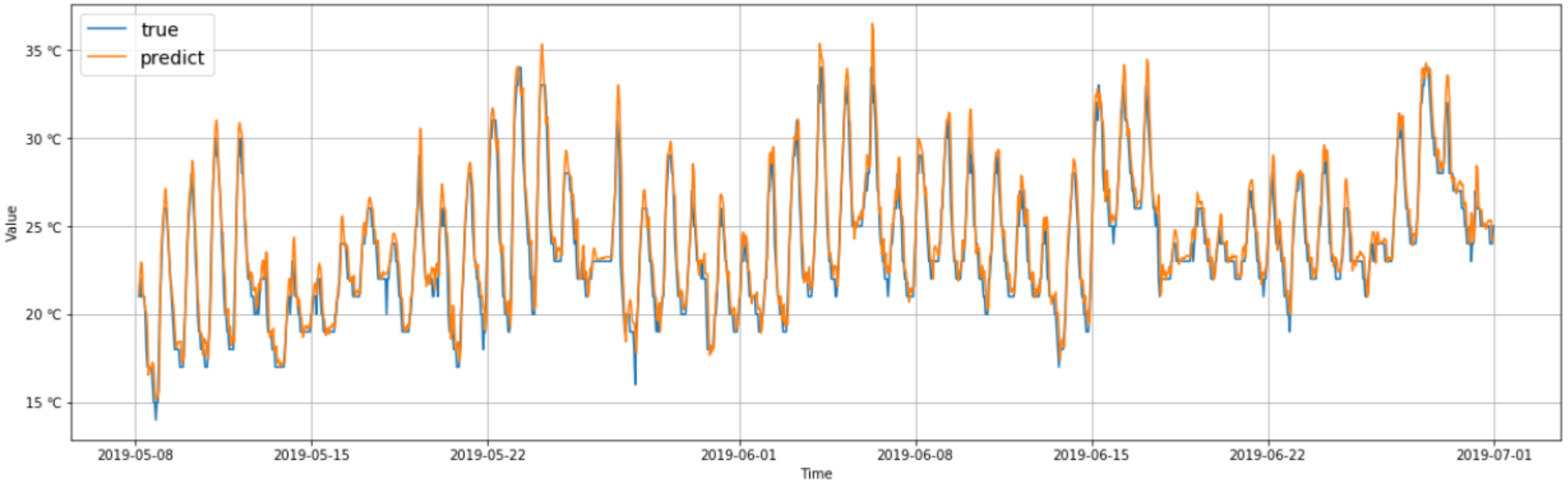
```
test_predict=[]
split_point=int(split_prop*int(series.size))
test_time=time[split_point+window_size-1:]
#测试集真实序列
test_true=series[split_point+window_size-1:]
#测试集预测序列
test_predict=net(test_feature).squeeze().tolist()
#用不同颜色画在一张图里 便于对比
fig,ax=plt.subplots(figsize=(20, 6))
#设置纵轴单位
ax.yaxis.set_major_formatter(mticker.FormatStrFormatter('%d ℃'))
plot_series(test_time, test_true,label='true')
plot_series(test_time, test_predict,label='predict')
plt.show()
```

用训练好的模型在测试集上单值预测；

想要预测某时刻的值，将该时刻前面(window_size-1)个真实值作为网络输入，网络输出即为预测值；

将真实序列与预测序列用不同颜色绘制在同一张图里，评价网络性能

- **5. 预测并对比**


true
predict
35 ℃
30 ℃
25 ℃
20 ℃
15 ℃
Value
2019-05-08
2019-05-15
2019-05-22
2019-06-01
2019-06-08
2019-06-15
2019-06-22
2019-07-01
Time

目 录

作业1：CNN-模拟数据回归复现

1. 生成多种模式叠加的时序数据，并处理为适合CNN的形式
2. 针对模拟数据，构建CNN进行训练，绘制loss曲线；用训练好的CNN进行预测，并与真实值进行对比

作业2：CNN-真实数据回归

1. 针对乙醇销售额数据，构建CNN（**复现**PPT中网络）进行训练，绘制loss曲线；用训练好的CNN进行预测，并与真实值对比；
2. 设计一个与PPT中不同的网络进行预测（预测效果优于PPT中的网络可加分），对优化的思路进行总结和分析。(要求：使用pytorch)
3. 与上次实验的结果进行对比，简要分析各个模型的优缺点。

作业要求

- 完成作业1+作业2
- **作业截至时间：10月18日16:00**
- 作业提交内容：实验报告jupyter notebook格式，要求写注释（注释单独算分）

北京交通大学《时间序列数据分析挖掘》课程组

赵守国： shgzhao@bjtu.edu.cn, http://faculty.bjtu.edu.cn/7563/

王　　晶： wj@bjtu.edu.cn, http://faculty.bjtu.edu.cn/9167/

夏佳楠： xiajn@bjtu.edu.cn , http://faculty.bjtu.edu.cn/9430/