实验九报告

姓名：宋卓伦 孔祥昊 刘孙延　　组号：11

一、实验目的

基于 AI 框架编写人工智能程序，掌握 AI 框架的使用方法，能够在已有代码基础上修改模型结构。

二、实验环境

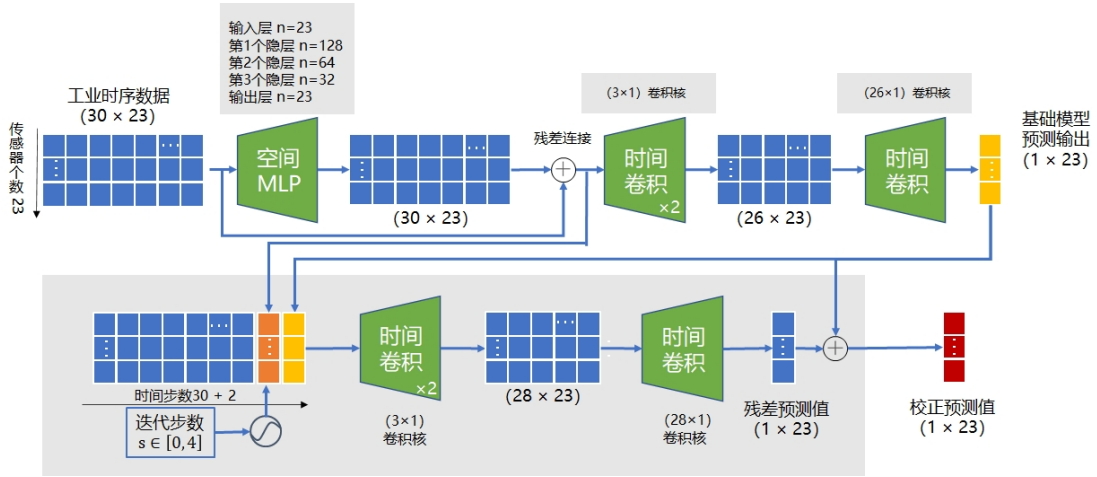
Python 版本：Python 3.12

使用的模型框架：MindSpore

IDE：Jupyter Notebook

三、实验步骤

根据第 10 章：人工智能应用案例课件，以流程工业控制系统时序数据预测案例为基础，修改 Step_Aware_TCN_MLP 类的代码，以实现如下图所示的具有更复杂结构的带偏差块的 TCN_MLP_with_Bias_Block 类，并编写相应的模型训练和测试代码。改进模型如下：



四、Python 基于偏差块的模型优化修改

定义 TCN_MLP_with_Bias_Block 类：

构造方法：

```
def __init__(self):  # 构造方法
    super().__init__()  # 调用父类的构造方法
    # 比 MULTI_STEP_TCN_MLP 类增加一个偏差块
    self.bias_block = nn.SequentialCell(

        # 改掉之前原有的空间 MLP 结构，将其换成两次时间卷积（3*1），得到 28*23 矩
阵
        # 再对后面进行一次 28*1 卷积，得到残差预测值

        nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(3, 1),
         pad_mode='valid'),
        nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(3, 1),
         pad_mode='valid'),
        nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(28, 1),
                  pad_mode='valid')
    )


# 对预测时间步数据的嵌入编码操作
    self.step_embedding = nn.Embedding(horizon, sensor_num)

# 对不同传感器的数据做融合（提取传感器数据间的关联特征）
    self.spatial_mlp = nn.SequentialCell(
        nn.Dense(sensor_num, 128),
        nn.ReLU(),
        nn.Dense(128, 64),
        nn.ReLU(),
        nn.Dense(64, 32),
        nn.ReLU(),
        nn.Dense(32, sensor_num)
    )
# 对时间序列做卷积（提取时间点数据间的关联特征）
    self.tcn = nn.SequentialCell(
        nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(3, 1),
             pad_mode='valid'),
        nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(3, 1),
             pad_mode='valid'),
```

```
    )
# 通过一个卷积层得到最后的预测结果
    self.final_conv = nn.Conv2d(in_channels=1, out_channels=1,
                          kernel_size=(26, 1), pad_mode='valid')  # 使用
26*1 卷积核，不补边
```

construst 方法：

```
def construct(self, x, iter_step):  # construct 方法
    h = self.spatial_mlp(x)  # 经过 spatial_mlp 空间处理后，得到的数据 h 的形
状：[batch_size, 30, 23]
    # 输入数据 x 的形状：[batch_size, 30, 23]
    h = x + h  # 残差连接，将 x 和 h 对应元素相加，得到的数据 h 的形状：
[batch_size, 30, 23]
    h0 = h.unsqueeze(1)  # 根据卷积操作需要，将 3 维数据升为 4 维数据：
[batch_size, 1, 30, 23]
    h0 = self.tcn(h0)  # 经过 tcn 时间卷积后，得到的数据 x 的形状：
[batch_size, 1, 26, 23]
    y = self.final_conv(h0)  # 通过 26*1 的卷积操作后，得到的数据 y 的形状：
[batch_size, 1, 1, 23]
    y = y.squeeze(1)  # 将前面增加的维度去掉，得到的数据 y 的形状：
[batch_size, 1, 23]

# 计算时间步数据的嵌入编码
    iter_step_tensor = mindspore.numpy.full((x.shape[0], 1),
iter_step,
              dtype=mindspore.int32)
    step_embedding = self.step_embedding(iter_step_tensor)

#step_embedding 的形状：[batch_size,1,23]

# 将输入数据 x 与时间步数据的嵌入编码拼接
    concat_op = mindspore.ops.Concat(axis=1)
# bias_input = concat_op((x, step_embedding,y))
# #bias_input 的形状：[batch_size,32,23]
    bias_input = concat_op((h, step_embedding, y))
# bias_input 的形状：[batch_size,32,23]

    bias_output = self.bias_block(bias_input.unsqueeze(1))
# [batch_size, 1, 1, 23]
    bias_output = bias_output.squeeze(1)
# [batch_size, 1, 23]

# 加上偏差块的预测结果
    y = y + bias_output
```

```
# y 的形状：[batch_size, 1, 23]

    return y  # 返回计算结果
```

训练代码：

```
tcn_mlp_bias = TCN_MLP_with_Bias_Block() #创建 TCN_MLP_with_Bias_Block 类对象 tcn_mlp_bias
loss_fn = nn.MAELoss() #定义损失函数
multi_step_optimizer = nn.Adam(tcn_mlp_bias.trainable_params(), 1e-3) #使用 Adam 优化器
def multi_step_forward_fn(data, label):
#定义多步预测前向计算的 multi_step_forward_fn 方法
    muti_step_pred =
        mindspore.numpy.zeros_like(label[:,:,PV_index+DV_index])
    x = data
    for step in range(horizon):
        pred = tcn_mlp_bias(x, step) #使用 tcn_mlp_bias 模型进行预测
        muti_step_pred[:,step:step+1,:] = pred[:,:,PV_index+DV_index]
#将当前时间步的预测结果保存到 multi_step_pred 中
        concat_op = mindspore.ops.Concat(axis=1)
        x = concat_op((x[:,1:,:], pred)) #将预测结果加到输入中
        x[:,-1:,OP_index] = label[:,step:step+1,OP_index] #OP 控制变量无法预测、始终使用真实值
    loss = loss_fn(muti_step_pred, label[:,:,PV_index+DV_index]) #根据损失函数计算 PV 和 DV 变量的损失值
    return loss, muti_step_pred #返回损失值和预测结果
multi_step_grad_fn = mindspore.value_and_grad(multi_step_forward_fn, None, multi_step_optimizer.parameters, has_aux=True) #获取用于计算梯度的函数
multi_step_model_run = MULTI_STEP_MODEL_RUN(tcn_mlp_bias, loss_fn, multi_step_optimizer, multi_step_grad_fn) #创建 MODEL_RUN 类对象 model_run
multi_step_model_run.train(train_dataset_t2, val_dataset_t2, 10, 'tcn_mlp_bias.ckpt') #调用 model_run.train 方法完成训练
```

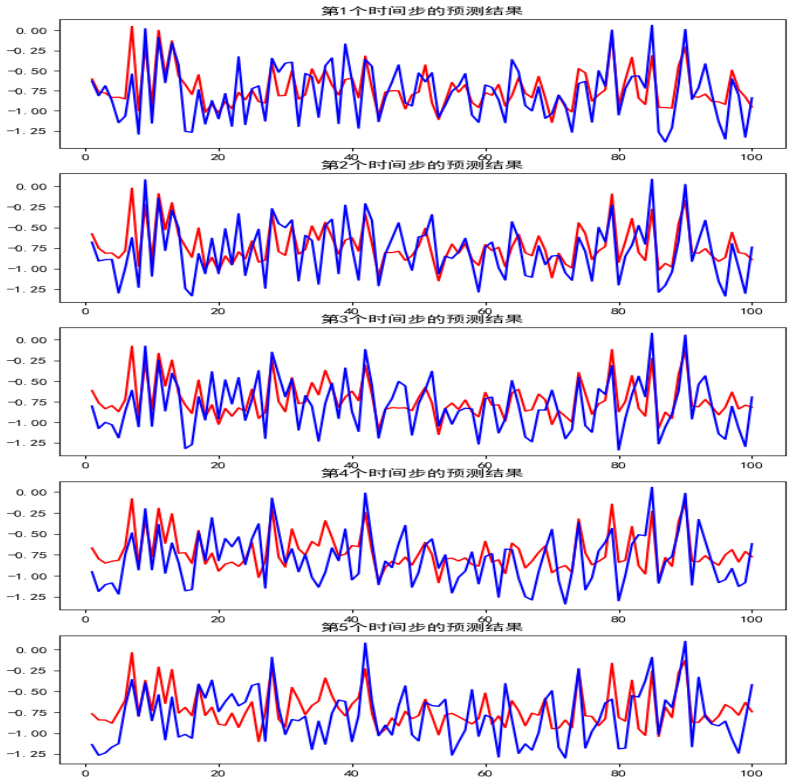
绘图代码：

```
tcn_mlp_bias = TCN_MLP_with_Bias_Block() #创建 TCN_MLP_with_Bias_Block 类对象 tcn_mlp_bias
loss_fn = nn.MAELoss() #定义损失函数
multi_step_model_run = MULTI_STEP_MODEL_RUN(tcn_mlp_bias, loss_fn) #创建 MULTI_STEP_MODEL_RUN 类对象 multi_step_model_run
train_loss,_,_ = multi_step_model_run.test(train_dataset_t2, 'tcn_mlp_bias.ckpt') #计算训练集损失
val_loss,_,_ = multi_step_model_run.test(val_dataset_t2,
```

```
'tcn_mlp_bias.ckpt') #计算验证集损失
test_loss,preds,labels = multi_step_model_run.test(test_dataset_t2,
'tcn_mlp_bias.ckpt') #计算测试集损失
print('训练集损失：{0}，验证集损失：{1}，测试集损失：
{2}'.format(train_loss,val_loss,test_loss))
plt.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False
_,axes = plt.subplots(5,1,figsize=(8, 16))
interval = int(horizon/5)
for step in range(5):
    axes[step].set_title('第%d 个时间步的预测结果'%(step*interval+1))
    axes[step].plot(range(1,101), preds[:100,step*interval,0],
        color='Red') # 绘制第 1 个传感器的前 100 条数据的预测结果
    axes[step].plot(range(1,101), labels[:100,step*interval,0],
        color='Blue') # 绘制第 1 个传感器的前 100 条数据的标签
```

修改后的运行结果：

```
开始训练......
第 1/10 轮
训练集损失：0.2462623，验证集损失：0.28171587
第 2/10 轮
训练集损失：0.22445497，验证集损失：0.26102737
第 3/10 轮
训练集损失：0.2092233，验证集损失：0.27171445
第 4/10 轮
训练集损失：0.20435657，验证集损失：0.2918072
第 5/10 轮
训练集损失：0.194821，验证集损失：0.30782187
第 6/10 轮
训练集损失：0.18917839，验证集损失：0.32789937
第 7/10 轮
训练集损失：0.18191803，验证集损失：0.35044008
第 8/10 轮
训练集损失：0.17806095，验证集损失：0.36858776
第 9/10 轮
训练集损失：0.17494999，验证集损失：0.39032102
第 10/10 轮
训练集损失：0.1725559，验证集损失：0.40841547
训练完成！
```



根据上面结果发现，由于出现过拟合的情况，验证集损失是逐步上升的。而且经过多个时间步的预测，结果越来越不太贴合实际情况。

解决过拟合的几种方法：

1、使用数据增强和扩充：可以在时间维度上做变换，或者对于特征维度的数据进行数据扰动；

2、使用正则化方法：正则化权重或者使用 dropout 随机丢弃一部分神经元；

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n}|\theta_i|$$

正则化可以降低模型复杂性，该惩罚的目的是优化权重绝对值的总和。它生成一个简单且可解释的模型，且对于异常值是鲁棒的。

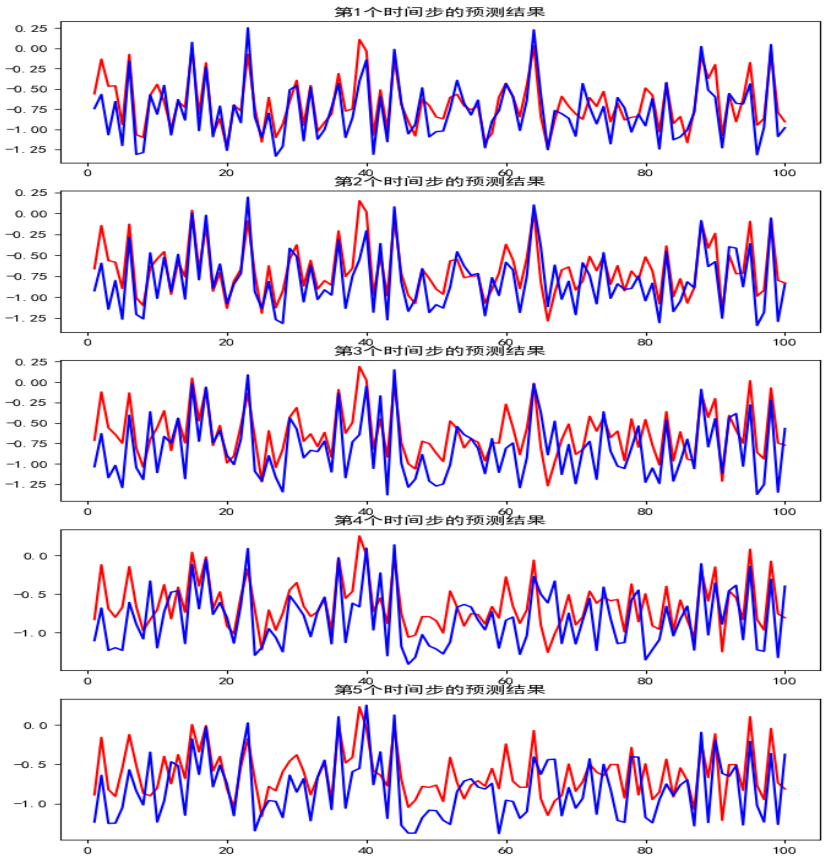
3、早停法（early-stopping）：避免模型在训练后期过度拟合训练数据，让模型在泛化性能最佳的阶段停止训练，从而提高模型在未见过的数据上的表现。

4、训练多个模型并集成或者优化模型结构（减少复杂度或增加模型深度、宽度权衡）。

我们提出了一种思考，在每次时间卷积操作后添加激活函数，不断激活，验

证这对于拟合的影响（存在一定风险）。

```
开始训练......
第 1/10 轮
训练集损失：0.2580407，验证集损失：0.3170104
第 2/10 轮
训练集损失：0.23319764，验证集损失：0.29752958
第 3/10 轮
训练集损失：0.21716091，验证集损失：0.28403884
第 4/10 轮
训练集损失：0.20879263，验证集损失：0.28892258
第 5/10 轮
训练集损失：0.20217681，验证集损失：0.2944351
第 6/10 轮
训练集损失：0.1963473，验证集损失：0.30331892
第 7/10 轮
训练集损失：0.19384888，验证集损失：0.32249138
第 8/10 轮
训练集损失：0.18788119，验证集损失：0.32862937
第 9/10 轮
训练集损失：0.18414183，验证集损失：0.34599605
第 10/10 轮
训练集损失：0.18146639，验证集损失：0.35864156
训练完成！
```



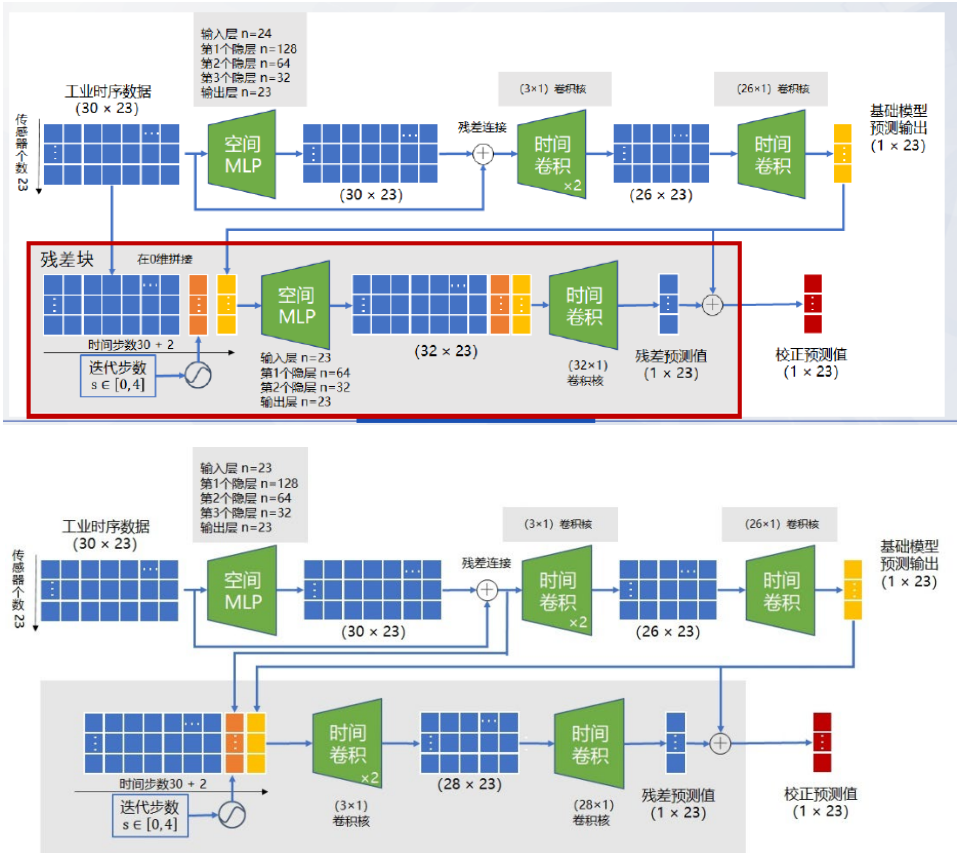
修改的部分如下：

```
self.bias_block = nn.SequentialCell(
     nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(3,1),
                                              pad_mode='valid'),
     nn.ReLU(),
     nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(3,1),
                                              pad_mode='valid'),
     nn.ReLU(),
     nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(28,1),
                                              pad_mode='valid')
)
```

可以看到，仍然存在一定过拟合，但是比上次好了一点点（涨幅较小）。

五、实验思考

原有的 ipynb 文件存在如下图所示的残差块函数，我们对于里面存在的函数分析其逻辑过程，发现了可以修改的过程。



如两幅图所示，要求的流程图将原有的流程图中残差块内空间 MLP 替换为两次时间卷积，再一次获得残差预测值。输入过程中原有数据经过空间 MLP 并与

之前的结果进行残差连接获得新的矩阵，与时间编码和基础模型预测输出一起输入到残差块拼接中进行后续操作。得到校正预测值。注意的是图片里有一点错误，残差连接后的矩阵应当作为属于于时间编码和基础模型预测输出一起拼接。