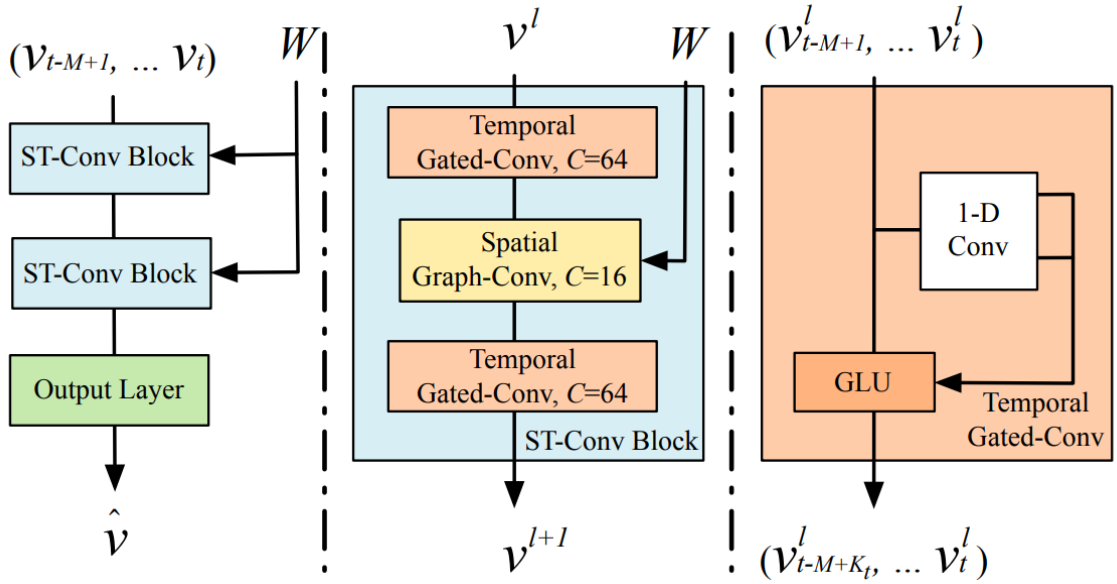
mindspore开发示例——STGCN

by czx 2022.07.27

论文链接：[1709.04875v4] Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting (arxiv.org)

STGCN 网络结构示意图



代码结构如下：

1.网络构建——stgcn_model.py

2.数据处理——stgcn_dataloader.py

3.相关计算函数——stgcn_utility.py

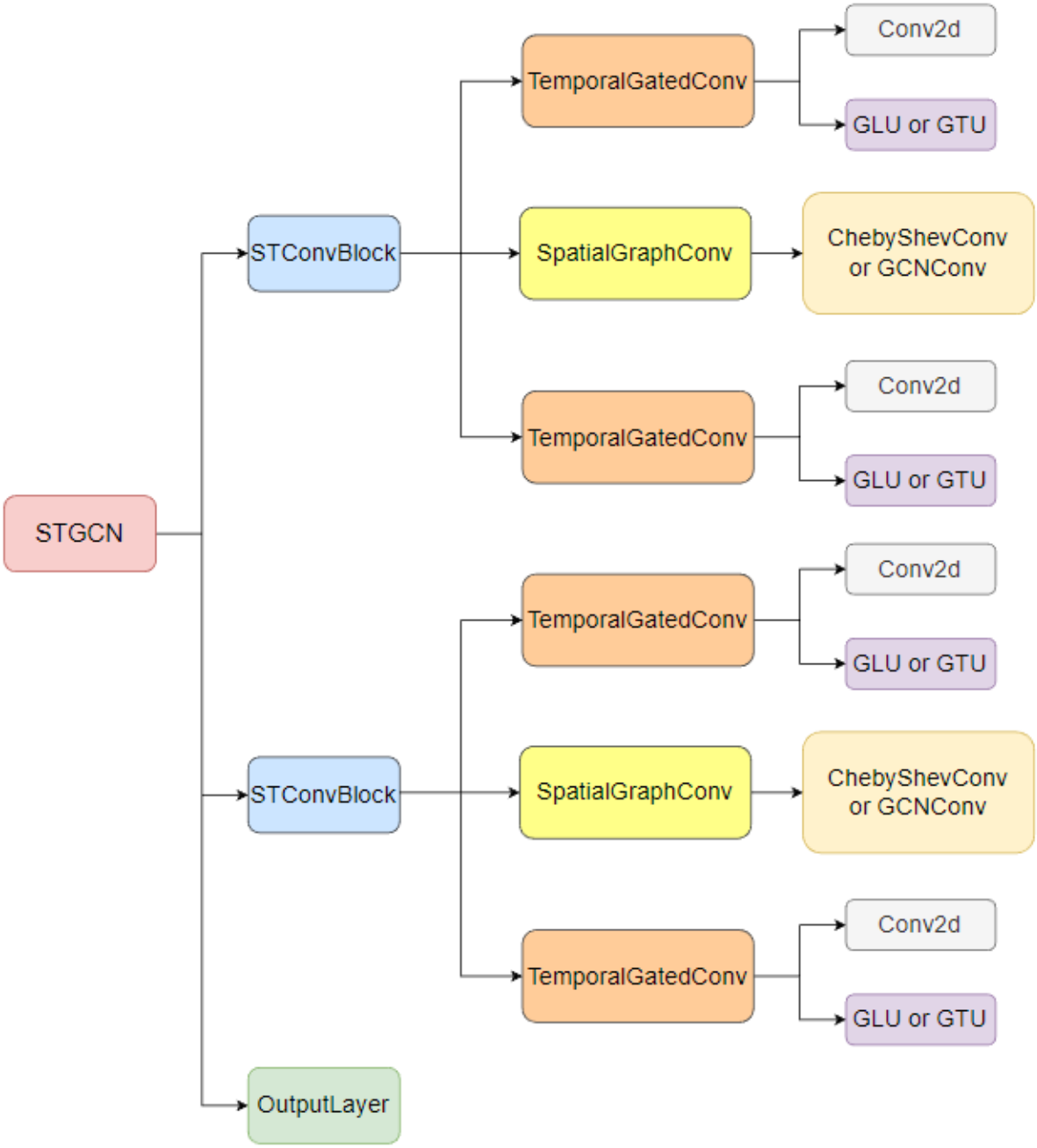
4.模型训练——stgcn_train.py

5.模型测试——stgcn_test.py

本文为大家提供一种可行的mindspore代码结构示例，复现其他模型时可以按照本文的代码结构，逐步实现所需的模型。

1.网络构建（stgcn_model.py）

网络构建的过程，本质上是**多个独立的神经网络层堆叠并串联**的过程；复杂的网络可拆分为多个独立的神经网络层并逐个实现。以本文实现的STGCN为例，按照原论文中的网络设计，其结构大致可如下拆分：



在mindspore的开发中，模型或神经网络层应当**继承mindspore.nn.Cell**（与Pytorch中的nn.Module类似）；mindspore.nn中也包含了许多**预定义的构建块**，如常见的卷积神经网络层、非线性激活函数层以及线性层等，可以在mindspore文档中查看相关用法。

以下为代码中STConvBlock的实现（**即上图中蓝色的STConvBlock**）：

```
class STConvBlock(nn.Cell):
    """
    # 1.TemporalConvLayer
    # 2.GraphConvLayer
    # 3.TemporalConvLayer
    # 4.Normalization
    # 5.Dropout
    """

    def __init__(self, t_kernel_size, cheb_k, vertex_num, last_block_channel,
channels, gate_type, graph_conv_type,
```

```
                graph_conv_matrix, drop_rate):
        super(STConvBlock, self).__init__()
        # 第一个TemporalConvLayer
        self.tmp_conv1 = TemporalConvLayer(t_kernel_size, last_block_channel,
channels[0],
                                           vertex_num, gate_type)
        # GraphConvLayer
        self.graph_conv = GraphConvLayer(cheb_k, channels[0], channels[1],
                                         graph_conv_type, graph_conv_matrix)
        # 第二个TemporalConvLayer
        self.tmp_conv2 = TemporalConvLayer(t_kernel_size, channels[1],
channels[2],
                                           vertex_num, gate_type)
        # 层归一化
        self.tc2_ln = nn.LayerNorm([vertex_num, channels[2]], begin_norm_axis=2,
                                   begin_params_axis=2, epsilon=1e-05)
        # ReLU激活
        self.relu = nn.ReLU()
        # 随机丢弃
        self.do = nn.Dropout(keep_prob=drop_rate)

    def construct(self, x):
        # 逐层传递计算结果，实现从输入到输出的串联
        x_tmp_conv1 = self.tmp_conv1(x)
        x_graph_conv = self.graph_conv(x_tmp_conv1)
        x_graph_conv_relu = self.relu(x_graph_conv)
        x_tmp_conv2 = self.tmp_conv2(x_graph_conv_relu)
        x_tc2_ln = ops.Transpose()(x_tmp_conv2, (0, 2, 3, 1))
        x_tc2_ln = self.tc2_ln(x_tc2_ln)
        x_tc2_ln = ops.Transpose()(x_tc2_ln, (0, 3, 1, 2))
        x_do = self.do(x_tc2_ln)
        return x_do
```

各个神经网络层的构建思路大致相同，即：**先实现其中较简单的层，随后串联多个神经网络层实现预期的网络结构。**

关于损失函数，mindspore中有多种定义方式，此处给出其一种可行的实现方法：

```
class LossCellWithNetwork(nn.Cell):
    """ STGCN loss """

    def __init__(self, network):
        super(LossCellWithNetwork, self).__init__()
        self.loss = nn.MSELoss()
        self.network = network
        self.reshape = ops.Reshape()

    def construct(self, x, label):
        # x为输入，label为预期结果
        x = self.network(x) # 得到网络输出
        # 计算loss并作为返回值
        x = self.reshape(x, (len(x), -1))
        label = self.reshape(label, (len(label), -1))
        loss = self.loss(x, label)
        return loss
```

此部分常见问题：

1.参数设置无从下手：可以先尽可能理解网络结构，随后阅读**Libcity**中Pytorch版的实现代码，找到相关层的参数设置方法；**此处需要注意，Pytorch中各个网络结构、计算函数所需的参数并不完全与mindspore的定义相同（如mindspore.nn.Conv2d与torch.nn.Conv2d），极有可能看似完全相同的代码会带来不一样的输出，因此一定要查阅相关文档并确定其使用方法。**

2.部分代码运行结果和预期不相符：可以在construct函数中**插入中间变量的输出**进行**debug**，例如输出中间计算结果x.shape来验证计算过程的正确性。

2.数据处理（stgcn_dataloader.py）

mindspore.dataset提供了加载和处理各种通用数据集的API，但我们往往用不上这些API，需要用自定义数据集来完成训练，此时可以用**mindspore.dataset.GeneratorDataset**来实现，其实现方法之一如下所示：

```
class STGCNDataset:

    def __init__(self, data_path, n_his, n_pred, zscore, mode=0):
        # 读取数据，预处理
        self.df = pd.read_csv(data_path, header=None)
        self.data_col = self.df.shape[0]
        # 设置train、val、test各占的比例
        self.val_and_test_rate = 0.15
        # 划分train、val、test数据集
        self.len_val = int(math.floor(self.data_col * self.val_and_test_rate))
        self.len_test = int(math.floor(self.data_col * self.val_and_test_rate))
        self.len_train = int(self.data_col - self.len_val - self.len_test)
        self.dataset_train = self.df[: self.len_train]
        self.dataset_val = self.df[self.len_train: self.len_train +
self.len_val]
        self.dataset_test = self.df[self.len_train + self.len_val:]

        # 利用zscore=preprocessing.StandardScaler()进行归一化
        self.dataset_train = zscore.fit_transform(self.dataset_train)
        self.dataset_val = zscore.transform(self.dataset_val)
        self.dataset_test = zscore.transform(self.dataset_test)
        # 根据mode决定返回的数据集种类
        if mode == 0:
            self.dataset = self.dataset_train
        elif mode == 1:
            self.dataset = self.dataset_val
        else:
            self.dataset = self.dataset_test
        # 与具体任务有关的数据处理
        self.n_his = n_his
        self.n_pred = n_pred
        self.n_vertex = self.dataset.shape[1]
        self.len_record = len(self.dataset)
        self.num = self.len_record - self.n_his - self.n_pred
        # 得到self.x和self.y，二者分别为数据data和对应的标签label
        self.x = np.zeros([self.num, 1, self.n_his, self.n_vertex], np.float32)
        self.y = np.zeros([self.num, self.n_vertex], np.float32)

        for i in range(self.num):
```

```
            head = i
            tail = i + self.n_his
            self.x[i, :, :, :] = self.dataset[head: tail].reshape(1, self.n_his,
self.n_vertex)
            self.y[i] = self.dataset[tail + self.n_pred - 1]

    def __getitem__(self, index):
        # 重要！使得我们定义的STGCNDataset为可迭代的对象，如此就可供mindspore.Model使用了
        return self.x[index], self.y[index]

    def __len__(self):
        return self.num


def create_dataset(data_path, batch_size, n_his, n_pred, zscore, mode=0):
    data = STGCNDataset(data_path, n_his, n_pred, zscore, mode=mode)
    shuffle = True # 是否打乱重排
    if mode != 0: # 是否为训练集，若为训练集则需要重排以便训练
        shuffle = False
    # 利用GeneratorDataset来构建可供mindspore.Model使用的数据集
    dataset = ds.GeneratorDataset(data, column_names=["inputs", "labels"],
shuffle=shuffle)
    dataset = dataset.batch(batch_size) # 设置batch_size
    return dataset
# 调用方法示例，dataset即为构建的数据集
dataset = stgcn_dataloader.create_dataset('./v.csv', settings['batch_size'],
settings['n_his'],              settings['n_pred'],zscore, mode=0)
```

简而言之，我们需要构建一个**可供mindspore.Model迭代的数据集**，并根据具体的任务进行对应的**数据预处理**；以上仅为一种可行的实现方式，其他定义方式可查阅mindspore文档。

3.相关计算函数（stgcn_utility.py）

此处可定义网络的**评价方式**（metric）等，以及与具体任务有关的**计算过程**，例如本文在这一部分还实现了拉普拉斯矩阵的计算。以下为实现评价指标的部分代码：

```
def evaluate_metric(model, dataset, scaler):
    # model为训练好的网络，dataset为数据集，scaler为前文提到过的归一化方法zscore，此处需要
反归一化
    mae, sum_y, mape, mse = [], [], [], []
    for data in dataset.create_dict_iterator():
        # 枚举数据集
        x = data['inputs']
        y = data['labels']
        y_pred = model(x)
        y_pred = ops.Reshape()(y_pred, (len(y_pred), -1))
        y_pred = scaler.inverse_transform(y_pred.asnumpy()).reshape(-1)
        y = scaler.inverse_transform(y.asnumpy()).reshape(-1)
        d = np.abs(y - y_pred)
        mae += d.tolist()
        sum_y += y.tolist()
        mape += (d / y).tolist()
        mse += (d ** 2).tolist()
    # 返回交通速度预测的评价指标MAE、MAPE、RMSE
    MAE = np.array(mae).mean()
```

```
    MAPE = np.array(mape).mean()
    RMSE = np.sqrt(np.array(mse).mean())

    return MAE, RMSE, MAPE
```

4.模型训练（stgcn_train.py）

定义具体的模型，除了神经网络层的设计，还需要相关的**超参数**，一般使用**argparse.ArgumentParser()**进行整理，便于使用者作为运行参数来调整，如下所示：

```
parser = argparse.ArgumentParser()
parser.add_argument('--learning_rate', type=float, default=0.003, help='learning
rate')
parser.add_argument('--epoch', type=int, default=100, help='training epoch')
args = parser.parse_args()
# 使用 args.epoch 即可得到对应超参数的值
# 输入时使用 python xxx.py --epoch=200 即可调整对应的参数
```

此处可以同时完成**路径**、**训练设备**等参数的定义。由于我的代码目前仅供我个人使用，因此暂时未使用标准的格式来编写这一部分，后续将统一至以上规格。

除此以外，实现训练过程的具体代码如下所示：

```
# 定义归一化函数
zscore = preprocessing.StandardScaler()

# 定义训练时的模式，以及设备等参数
context.set_context(mode=context.GRAPH_MODE, device_target="CPU",
save_graphs=False)

# 调用utility得到所需的数据
adj_mat = pd.read_csv('./w.csv', header=None)
vertex_num = adj_mat.shape[0]
conv_matrix = stgcn_utility.calculate_laplacian_matrix(adj_mat.to_numpy(),
mat_type)

# 根据参数定义网络
net = stgcn_model.STGCN(settings['t_kernel_size'], settings['cheb_k'], blocks,
settings['n_his'], vertex_num,settings['gated_act_func'],
settings['graph_conv_type'], conv_matrix, settings['drop_rate'])

if __name__ == "__main__":
    # 调用dataloader得到数据集
    dataset = stgcn_dataloader.create_dataset('./v.csv', settings['batch_size'],
settings['n_his'], settings['n_pred'],zscore, mode=0)
    data_len = dataset.get_dataset_size()
    # 定义优化器
    optimizer = nn.AdamWeightDecay(net.trainable_params(), learning_rate=0.003,
                                   weight_decay=settings['weight_decay_rate'])
    # 定义所需的callback，用来监控训练过程
    loss_cb = LossMonitor() # 监控每一步的loss
    time_cb = TimeMonitor(data_size=data_len) # 监控训练用时
    # 保存checkpoint
    config_ck = CheckpointConfig(save_checkpoint_steps=data_len,
```

```
                                    keep_checkpoint_max=settings['epochs'])
    ckpoint_cb = ModelCheckpoint(prefix='STGCN', directory=model_save_path,
config=config_ck)
    callbacks = [time_cb, loss_cb, ckpoint_cb]
    # 将loss模型和神经网络结合，得到用于训练的模型
    net = stgcn_model.LossCellWithNetwork(net)
    # 定义mindspore.Model，用于训练
    model = Model(net, optimizer=optimizer)
    # 开始训练并根据callbacks监视训练过程
    model.train(settings['epochs'], dataset, callbacks=callbacks)
```

在开始训练前，还需要定义用于训练的**优化器**（此处使用的是mindspore.nn.AdamWeightDecay）；为了能够实时监控训练过程并得到训练结果，需要定义相应的**callback**，例如**LossMonitor()**能帮助我们看到训练中每一步的loss，**checkpoint**则能够为我们保存对应时刻的模型，以便训练后测试训练的结果。以上训练方式仅为一种可能的实现，可以作为参考。

5.模型测试（stgcn_test.py）

与训练模型类似，需要先**定义相应的参数**；随后需要调用**dataloader**得到对应的数据集，并使用utility中定义的**评估方法**来完成测试。

```
# 调用dataloader加载数据集
dataset = stgcn_dataloader.create_dataset('./v.csv', settings['batch_size'],
settings['n_his'],
                                                settings['n_pred'], zscore,
mode=2)
# 读取对应的checkpoint
param_dict = load_checkpoint('./res/chebconv_15/STGCN_2-40_139.ckpt')
# 将checkpoint中保存的网络参数载入神经网络
load_param_into_net(net, param_dict)
# 调用utility中的评估函数，得到相应评价指标的值
test_MAE, test_RMSE, test_MAPE = stgcn_utility.evaluate_metric(net, dataset,
zscore)
# 打印测试结果
print(f'MAE {test_MAE:.2f} | MAPE {test_MAPE * 100:.2f} | RMSE {test_RMSE:.2f}')
```

代码使用方法

在目录下执行python stgcn_train.py即可开始训练，训练得到的checkpoints将保存在./res/中；测试时，**在stgcn_test.py中修改checkpoint的路径后**，即可使用python stgcn_test.py开始测试。

代码测试结果：以下为15min、30min、45min的预测情况，与论文中给出的结果差距不大。

```
MAE:2.25/2.91/3.30  MAPE:5.45/7.37/8.26 RMSE:4.05/5.39/6.09
```

Model	PeMSD7(M) (15/ 30/ 45 min)		
	MAE	MAPE (%)	RMSE
HA	4.01	10.61	7.20
LSVR	2.50/ 3.63/ 4.54	5.81/ 8.88/ 11.50	4.55/ 6.67/ 8.28
ARIMA	5.55/ 5.86/ 6.27	12.92/ 13.94/ 15.20	9.00/ 9.13/ 9.38
FNN	2.74/ 4.02/ 5.04	6.38/ 9.72/ 12.38	4.75/ 6.98/ 8.58
FC-LSTM	3.57/ 3.94/ 4.16	8.60/ 9.55/ 10.10	6.20/ 7.03/ 7.51
GCGRU	2.37/ 3.31/ 4.01	5.54/ 8.06/ 9.99	4.21/ 5.96/ 7.13
STGCN(Cheb)	**2.25/ 3.03/ 3.57**	5.26/ **7.33/ 8.69**	**4.04/ 5.70/ 6.77**
STGCN(1^{st})	2.26/ 3.09/ 3.79	**5.24**/ 7.39/ 9.12	4.07/ 5.77/ 7.03