



浙江工业大学

《人工智能导论》 自主实验项目报告

题 目	基于图神经网络的交通预测
院 系	计算机科学与技术学院、软件学院
专 业	计算机科学与技术专业
班 级	计算机科学与技术 2201 班
小组成员	

团队分工说明

成员姓名	学号	主要分工	备注
		1. 文档：第二部分程序运行环境，第四部分的程序实现和实验结果展示，损失函数 2. 论文代码的整体复现 3. 文献的查找 4. 成果汇报	组长
		1. 文档：时空同步图卷积模块,时空同步图卷积层,局部时空图创建，时空嵌入 2. 国内外实验研究 3. 文献的阅读 4. PPT 制作	组员
		1. 文档：第三部分的增强模型性能的相关操作,实验背景,实验意义 2. 可视化代码实现 3. 文献的阅读 4. PPT 制作	组员

一、实验背景和意义

本次自主实验，小组主要任务为复现 AAAI-20 论文《Spatial-Temporal Synchronous Graph Convolutional Networks: A New Framework for Spatial-Temporal Network Data Forecasting》。本小组选用近年来较火的图神经网络来对交通流量的时间和空间进行建模，从而预测未来的交通流量。

1.1 实验背景

随着城市化的加剧和交通拥堵问题的日益突出，交通预测成为解决城市交通问题中的关键环节。交通预测的准确性直接影响城市交通系统的效率和可持续性，因此对交通流量、交通状况和交通模式的预测成为学术界和产业界关注的焦点之一，具体如下图 1.1 所示：

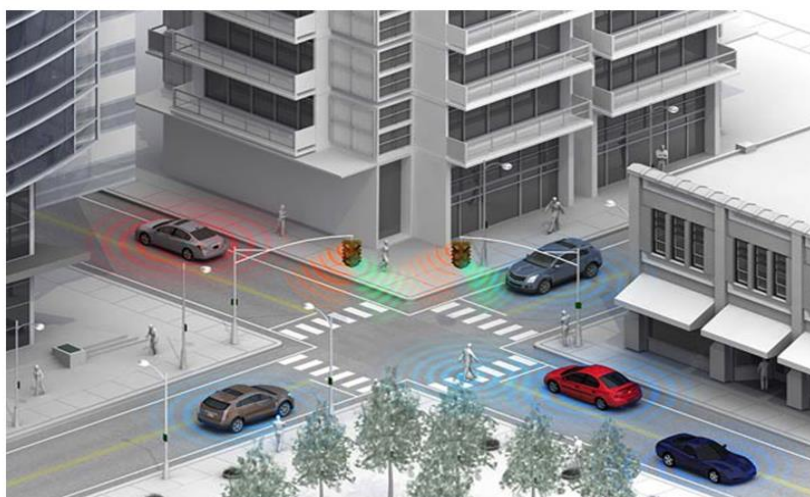


图 1.1：交通预测

交通预测也成为了最近几年较火的一个研究课题。对于交通预测而言，它不仅包括每一时刻的交通流量，即时间维度的信息；也存在某一时段的交通结构信息。因此针对于交通预测问题，当然少不了的空间—时间网络的运用。

空间-时间网络数据预测是空间-时间数据挖掘中的一个基本研究问题。空间-时间网络是一种典型的数据结构，可以描述许多实际应用中的数据，如交通网络、移动基站网络、城市水系统等。对空间-时间网络数据的准确预测可以显著提高这些应用的服务质量。随着深度学习在图上的发展，强大的方法如图卷积网络及

其变种已广泛应用于这些空间-时间网络数据预测任务，并达到令人满意的性能。然而，仍然缺乏有效的方法来建模空间和时间方面的相关性和异质性。本文将专注于设计一个模型，以同步捕捉复杂的空间-时间相关性并考虑异质性，以提高空间-时间网络数据预测的准确性。具体模型如下图 1.2 所示：

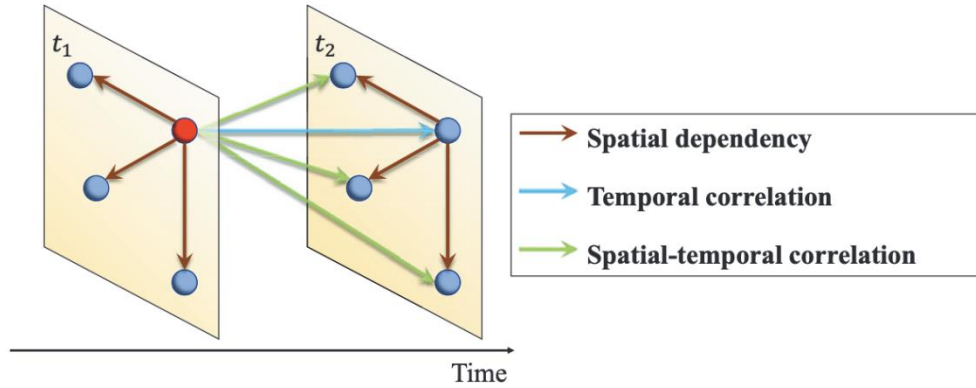


图 1.2：空间-时间网络中红色节点的影响

棕色箭头表示网络的边缘，同时表示在空间维度上的影响。蓝色箭头表示红色节点对自身在下一个时间步的影响。绿色箭头表示跨越空间和时间维度的影响，在红色节点和其相邻节点之间在下一个时间步。 t_1 和 t_2 表示两个连续的时间步。

以图 1.2 中显示的空间-时间网络为例，该网络中存在三种不同的影响。空间-时间图中的每个节点都可以在同一时间步直接影响其相邻节点，这种影响来自实际的空间依赖关系。同时，由于时间序列中的时间相关性，每个节点还可以在下一个时间步直接影响自身。此外，由于同步的空间-时间相关性，每个节点甚至可以在下一个时间步直接影响其相邻节点，如图 1.2 所示。存在这三种不同类型影响的原因是空间-时间网络中的信息传播同时发生在空间和时间维度上。由于节点之间的空间距离和时间序列的时间范围的限制，这些复杂的空间-时间相关性通常是局部的。我们将这些复杂的影响称为局部空间-时间相关性。建模这样的相关性对于空间-时间网络数据预测至关重要。先前的研究，如 DCRNN (Li 等人, 2017)^[1]、STGCN (Yu、Yin 和 Zhu, 2018)^[2]和 ASTGCN (Guo 等人, 2019a)^[3]，使用两个分离的组件分别捕捉时间和空间依赖关系。这些方法只直接捕捉了上述前两种影响，即空间依赖关系和时间相关性。它们将空间表示输入到时间建模模块中，间接捕捉第三种影响。然而，如果能够同时捕捉这些复杂的局部空间-时间相关性，将对空间-时间数据预测非常有效，因为这种建模方法揭示了空间

-时间网络数据生成的基本方式。

此外，空间-时间网络数据通常在空间和时间维度上表现出异质性。例如，在城市道路网络中，住宅区和商业区交通监测站记录的观测往往在不同时间呈现不同的模式。然而，许多先前的研究对不同时间段使用共享模块，无法有效捕捉空间-时间网络的异质性。

1.2 实验意义

为了捕捉复杂的局部空间-时间相关性和空间-时间数据的异质性，该论文提出了一种名为空间-时间同步图卷积网络（STSGCN）的模型。与许多先前的工作不同，STSGCN 模型可以直接同时捕捉局部空间-时间相关性，而不是使用不同类型的深度神经网络分别建模空间依赖关系和时间相关性。具体而言，该论文构建了局部空间-时间图，将相邻时间步的个体空间图连接成一个图。然后，其构建了一个空间-时间同步图卷积模块（STSGCM）来捕捉这些局部空间-时间图中的复杂局部空间-时间相关性。同时，为了捕捉长距离空间-时间网络数据中的异质性，该论文中设计了一个空间-时间同步图卷积层（STSGCL），在不同时间段上部署多个独立的 STSGCM。最后，其堆叠多个 STSGCL 以聚合长距离空间-时间相关性和异质性进行预测。

而在本次自主实验中，本小组便是通过阅读这篇论文，和查阅其他相关的同一领域的研究，去理解这篇论文的核心算法思想。最后本小组将通过将论文中原有的 MXNet 框架改用 Pytorch 这一深度学习框架来进行复现这篇论文的代码，并且与原文的相关的指标进行相应的比较。

二、程序运行环境

本次实现的基于图神经网络的交通预测是在 Pycharm 平台下开发而成的，并且通过 Xshell 远程连接服务器，进行训练该模型。

程序运行的操作系统：Linux

2.1 硬件环境

由于小组成员的电脑硬件设备有限，所以我们在具有 3090 的 Linux 服务器上训练模型。具体的显卡配置如下图 2.1 所示：

```
Last login: Thu Dec 19 21:16:41 2024 from 10.12.41.61
(base) star@star-G7466-X5:~$ nvidia-smi
Fri Dec 20 20:44:09 2024
```

NVIDIA-SMI 550.135			Driver Version: 550.135			CUDA Version: 12.4		
GPU	Name	Perf	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr.	ECC
Fan	Temp		Pwr:Usage/Cap		Memory-Usage	GPU-Util	Compute	M.
							MIG	M.
0	NVIDIA GeForce RTX 3090		Off	00000000:17:00.0	Off			N/A
42%	55C	P2	306W / 350W	6484MiB / 24576MiB		78%	Default	N/A
1	NVIDIA GeForce RTX 3090		Off	00000000:31:00.0	Off			N/A
41%	58C	P2	312W / 350W	3772MiB / 24576MiB		86%	Default	N/A
2	NVIDIA GeForce RTX 3090		Off	00000000:65:00.0	Off			N/A
42%	26C	P8	24W / 350W	10MiB / 24576MiB		0%	Default	N/A
3	NVIDIA GeForce RTX 3090		Off	00000000:B1:00.0	Off			N/A
36%	28C	P8	29W / 350W	10MiB / 24576MiB		0%	Default	N/A

Processes:							
GPU	GI	CI	PID	Type	Process name	GPU Memory	Usage
	ID	ID					
0	N/A	N/A	2097	G	/usr/lib/xorg/Xorg	4MiB	
0	N/A	N/A	863734	C	python	6468MiB	
1	N/A	N/A	2097	G	/usr/lib/xorg/Xorg	4MiB	
1	N/A	N/A	863734	C	python	3756MiB	
2	N/A	N/A	2097	G	/usr/lib/xorg/Xorg	4MiB	
3	N/A	N/A	2097	G	/usr/lib/xorg/Xorg	4MiB	

```
(base) star@star-G7466-X5:~$
```

图 2.1: 服务器显卡的相关配置

具体的 CPU 信息如下图 2.2 所示：

```
(base) star@star-G7466-X5:~$ free -g
```

	total	used	free	shared	buff/cache	available
Mem:	125	7	113	0	4	116
Swap:	1	0	1			

```
(base) star@star-G7466-X5:~$
```

图 2.2: 服务器 CPU 的相关配置

因此本小组认为我们的硬件资源丰富，有一定的算力去完成后续的训练任务。

2.2 相关工具

- 编程语言：Python。因为 Python 中具有大量的深度学习框架，所以这对于小组后续的复现代码提供了极大的便利。
- 编程软件：Pycharm，小组本次安装的 Pycharm 是 2023.2.1 版本。PyCharm 是由 JetBrains 开发的一款专业的 Python 集成开发环境（IDE）。它被广泛用于 Python 开发，提供了许多功能和工具，使得编写、调试和部署 Python 代码变得更加高效。一些 PyCharm 的特性包括如下：

1. 智能代码完成：PyCharm 能为代码提供智能的建议，包括变量、方法和类名。高级调试器：内置调试器允许你在代码中设置断点、逐行执行代码，并检查变量的值，以便更轻松地调试程序。

2. 强大的重构：PyCharm 具有许多重构工具，可以帮助更改代码结构，改名或提取方法。

3. 集成版本控制：PyCharm 提供了与各种版本控制系统（如 Git、SVN 等）集成的功能，方便管理代码版本。

4. Web 开发支持：除了 Python，PyCharm 还支持 HTML、CSS 和 Javascript 等前端语言，提供了专业的 Web 开发环境。插件生态系统：PyCharm 拥有丰富的插件生态系统，允许用户根据自己的需求扩展 IDE 的功能。

下面就具体介绍 Pycharm 软件的安装：

1. 通过点击网址 [Download PyCharm: Python IDE for Professional Developers by JetBrains](#)，进入到 Pycharm 下载的主页进行点击自己对应想要的版本进行下载，本小组采用的是专业版。具体如下图 2.3 所示：

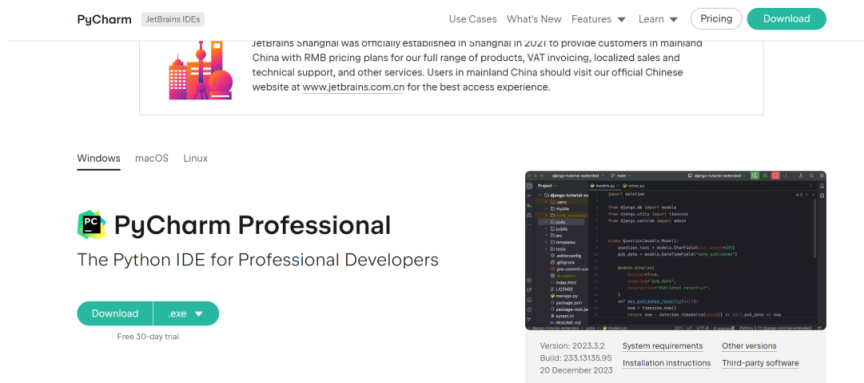


图 2.3：Pycharm 的下载官方主页

2. 在官网下载后，会得到对应的 exe 文件，之后点击对应的 exe 文件，会弹出对应的安装选项。具体如下图 2.4 所示：

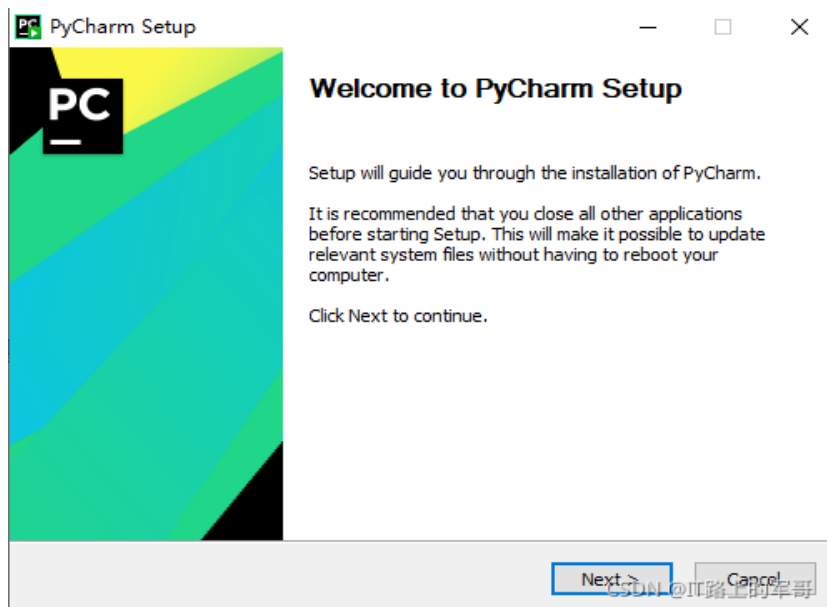


图 2.4: Pycharm 的安装

3. 修改对应的安装路径，具体如下图 2.5 所示：

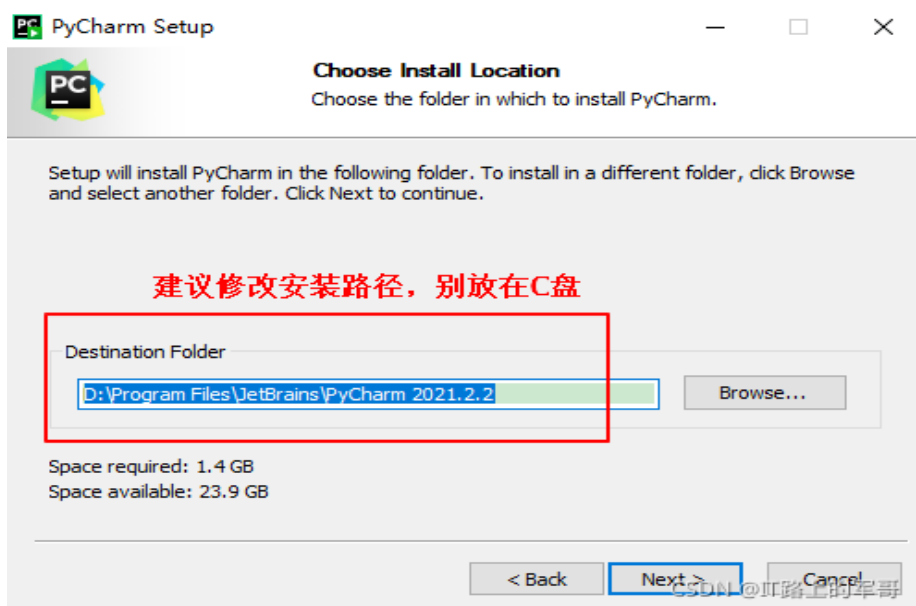


图 2.5: Pycharm 的安装路径

4. 点击 next 和 install 的选项，就可以成功安装好对应的 Pycharm，具体的 Pycharm 中的项目界面如下图 2.6 所示：

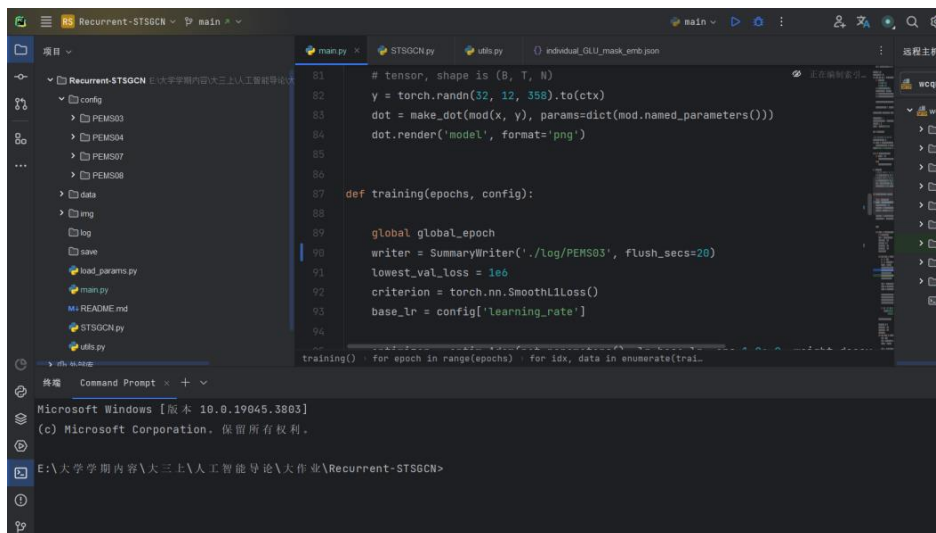


图 2.6: Pycharm 打开项目主页

- 其他软件：XShell 和 XFTP，小组将用该软件去远程连接服务器。它的具体介绍如下所示：Xshell 是一个强大的 SSH（安全外壳协议）客户端，用于在 Windows 系统下连接远程计算机或服务器。它是一个由南韩公司 NetSarang 开发的商业级软件，旨在为用户提供可靠的远程连接体验和广泛的功能。一些 Xshell 的特性包括：
 1. SSH 安全性：Xshell 支持 SSH 协议，这意味着它能够提供安全的加密通信，确保数据在网络上传输时不会被窃取或篡改。
 2. 会话管理：Xshell 允许用户轻松地管理多个远程连接会话，包括创建、保存和加载会话配置。
 3. 高级终端仿真：Xshell 提供了类似于 Linux 终端的界面，支持颜色化、多标签页等功能，让用户感觉仿佛在本地使用远程主机一样。
 4. 文件传输：用户可以通过使用内置的 SFTP 客户端在本地计算机和远程主机之间安全地传输文件。
 5. 便捷的用户界面：Xshell 的用户界面直观而且易于使用，同时还提供了丰富的自定义选项以满足不同用户的需求。
 6. 脚本支持：使用 Xshell, 用户可以编写和执行脚本以自动化一些重复的任务。XShell 的具体安装如下所示：
 1. 首先点击 <https://www.xshell.com/zh/free-for-home-school/> 链接就可以进入对应的 Xshell 的官网，点击获取 Xshell 和 XFTP 的安装包，具体如下图 2.7 所

示:

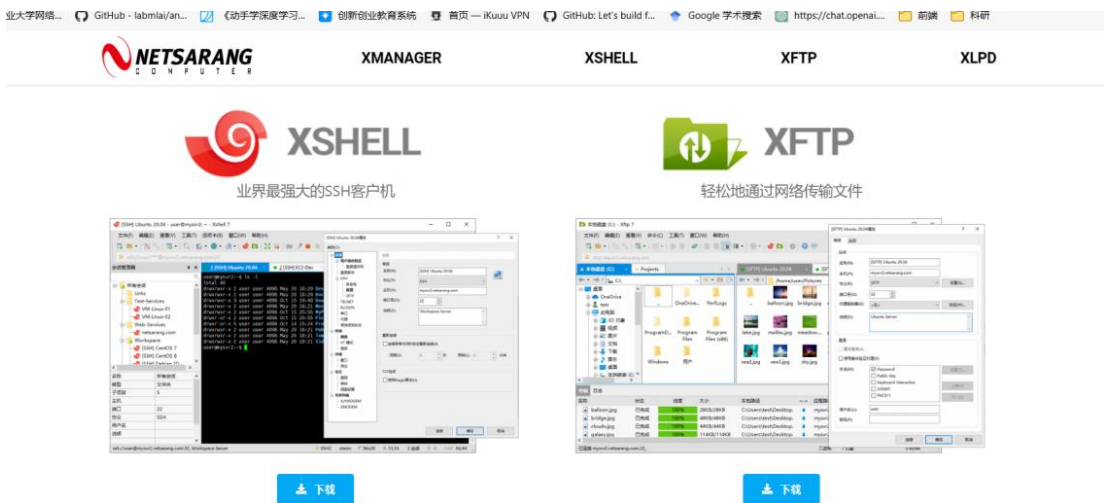


图 2.7: Xshell 和 XFTP 的下载主页

2. 得到对应的安装包之后, 与上文安装 Pycharm 的顺序一致, 设置好对应的安装路径和其他选项。这里就不再过多的赘述。最后通过账号和密码, 远程连接服务器, XShell 呈现的页面如下图 2.8 所示:

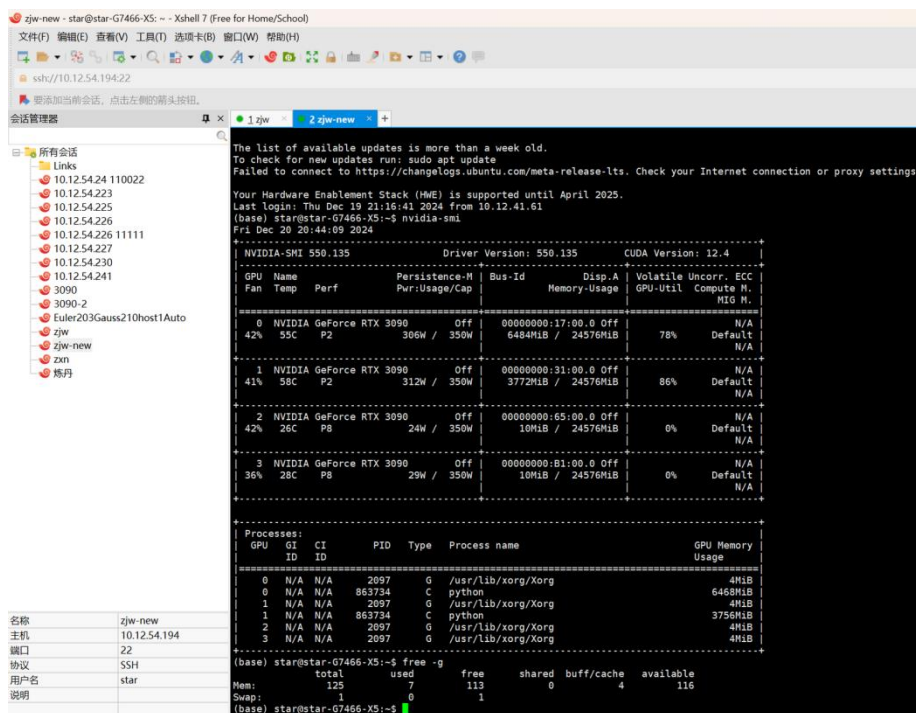


图 2.8: 成功配置好 Xshell

对于 XFTP 而言, 它的安装顺序与 XShell 一致, 所以这里不再过多的介绍。

2.3 环境配置

这一部分将会详细介绍本次编程环境的配置。主要以如何创建对应的虚拟环境，安装深度学习框架和其他相关的可视化库这两个方面进行详细的说明，具体如下所示：

- 安装虚拟环境：首先，小组为本次的项目进行创建对应的虚拟环境。在原本的 Linux 环境中，已经存在对应的 Aconda。所以首先调用以下命令去创建对应的虚拟环境，具体如下图 2.9 所示：

```
(AI) (base) wcq@amax:~$ conda create -n AI python=3.8
CondaValueError: prefix already exists: /data/wcq/anaconda3/envs/AI
(AI) (base) wcq@amax:~$
```

图 2.9: 创建虚拟环境

接下来通过相应的命令，转化到对应的虚拟环境中，具体如下图 2.10 所示：

```
(base) wcq@amax:~$ source activate AI
(AI) (base) wcq@amax:~$
```

图 2.10: 进入对应的虚拟环境

从图可知，在(base)前面出现了(AI)，这表明已经到了对应的虚拟环境中。

由于已经安装好了对应的虚拟环境，所以系统会提示已经安装对应的虚拟环境，无法再次创建。在接下来的深度学习框架和其他包安装中，就在该虚拟环境下进行配置和安装。

- 深度学习框架：目前存在的深度学习框架很多，比如 Tensflow、Keras、Caffe 等。近年来，论文中开源代码所采用的深度学习框架的对比图如下图 2.11 所示：

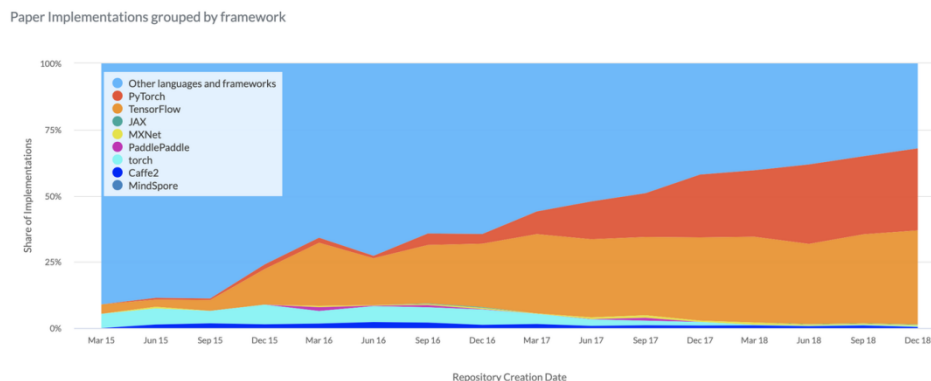


图 2.11: 各个学习框架进行比较

本次小组选择目前最为推崇和流行的 Pytorch 作为这次复现论文代码的深度学习框架。下面具体介绍 Pytorch 的安装。

1. 点击 Pytorch 的官网 <https://pytorch.org/>进行相关的下载，具体的下载界面如下图 2.12 所示，本次选择的是 Pytorch2.0 版本：

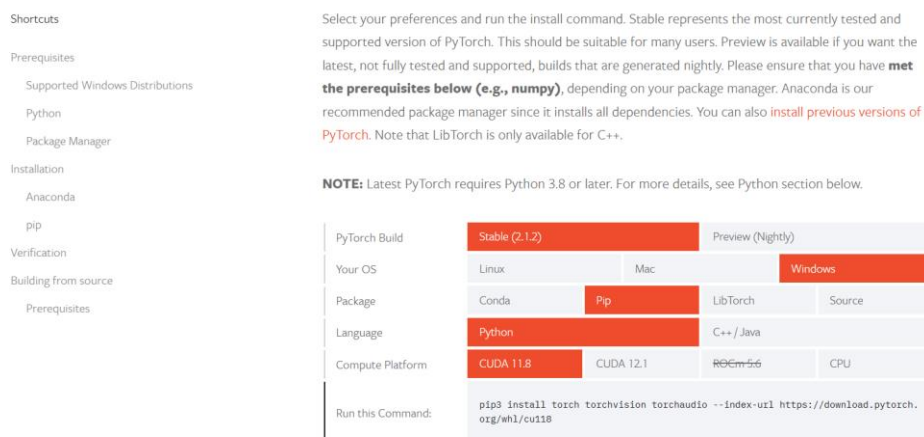


图 2.12: Pytorch 下载主页

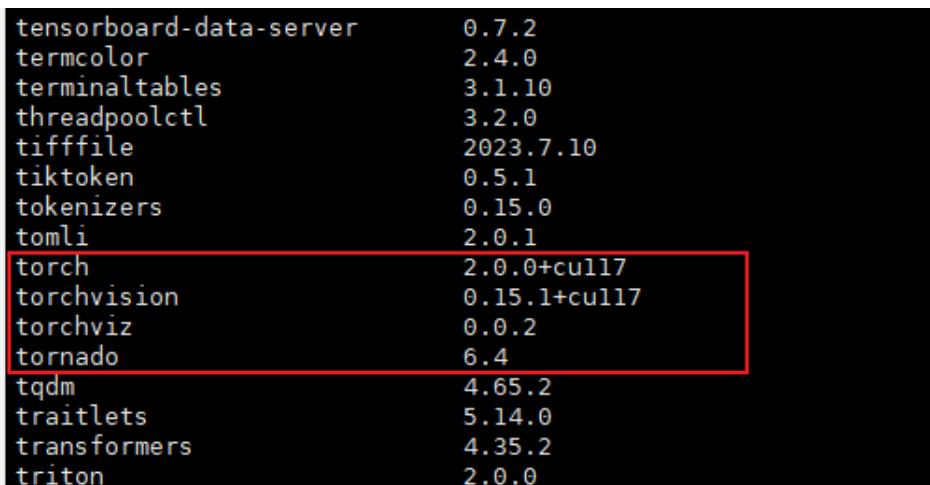
2. 通过查阅相关的资料，采用了 Pip 进行相关的安装，因为采用 Pip 安装相对稳定。所以采用选择 Linux, Pip, Python, CUDA11.8 的版本进行安装 GPU 版本的 Pytorch，最终将对应的底部链接复制，进行相关的配置。具体命令行如下图 2.13 所示：

```
(base) wq@amax:~$ pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
```

图 2.13: 下载 Pytorch

3. 安装完后，通过 `pip list` 命令就可以查看对应 Pytorch 版本。具体如下图 2.14

所示：



tensorboard-data-server	0.7.2
termcolor	2.4.0
terminaltables	3.1.10
threadpoolctl	3.2.0
tifffile	2023.7.10
tiktoken	0.5.1
tokenizers	0.15.0
tomli	2.0.1
torch	2.0.0+cu117
torchvision	0.15.1+cu117
torchviz	0.0.2
tornado	6.4
tqdm	4.65.2
traitlets	5.14.0
transformers	4.35.2
triton	2.0.0

图 2.14: 具体的 Pytorch 版本

- 可视化库：Python 中存在着大量的可视化库，比如 matplotlib, Seaborn 等。但是对于 Pytorch 而言，通过查阅了大量的相关资料可以得知，Tensflow 中有一款 TensorBoard 的包，可以很好地呈现数据。而对应于 Pytorch，TensorBoardX 则是它的可视化工具。它的优点如下所示：
 1. 可视化神经网络图：TensorBoard 可以可视化绘制神经网络的计算图，帮助用户直观地理解整个模型的结构和数据流动。
 2. 实时监控训练过程：通过 TensorBoard 可以实时监控模型在训练过程中的性能指标，如损失函数、准确率等，帮助用户更好地了解模型的训练情况。
 3. 可视化嵌入向量：TensorBoard 可以可视化嵌入向量，这对于理解高维数据（如 Word Embeddings 或图片特征）非常有帮助。
 4. 可视化训练过程中的分布：TensorBoard 可以展示训练过程中张量数值的分布情况，帮助用户发现潜在的数值偏差或异常情况。
 5. 可视化模型参数直方图：TensorBoard 可以绘制模型参数随时间的变化情况，有助于用户了解参数的更新情况。
 6. 生成实验报告：TensorBoard 可以生成富有信息的实验报告，并通过网络进行共享，从而方便团队间的交流和共享成果。而具体的安装也十分简单，即在终端中输入 `pip install tensorboard` 即可。具体如下图 2.15 所示：

```

[AI] (base) wcq@max:~$ pip install tensorboard
Requirement already satisfied: tensorboard in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (2.14.0)
Requirement already satisfied: absl-py>=0.4 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (2.0.0)
Requirement already satisfied: grpcio>=1.48.2 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (1.60.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (2.25.2)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (3.5.1)
Requirement already satisfied: numpy>=1.12.0 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (1.24.4)
Requirement already satisfied: protobuf>=3.19.6 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (4.25.1)
Requirement already satisfied: requests<3,>=2.21.0 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (2.28.2)
Requirement already satisfied: setuptools>=41.0.0 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (68.2.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (3.0.1)
Requirement already satisfied: wheel>=0.26 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from tensorboard) (0.41.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from google-auth<3,>=1.6.3->tensorboard) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from google-auth<3,>=1.6.3->tensorboard) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from google-auth<3,>=1.6.3->tensorboard) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from google-auth-oauthlib<1.1,>=0.5->tensorboard) (1.3.1)
Requirement already satisfied: importlib-metadata>=4.4 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from markdown>=2.6.8->tensorboard) (6.8.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensorboard) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensorboard) (3.6)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensorboard) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from requests<3,>=2.21.0->tensorboard) (2023.11.17)
Requirement already satisfied: MarkupSafe>=2.1.1 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from werkzeug>=1.0.1->tensorboard) (2.1.3)
Requirement already satisfied: zipp>=0.5 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensorboard) (3.17.0)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard) (0.5.1)
Requirement already satisfied: oauthlib>=3.0.0 in /data/wcq/anaconda3/envs/AI/lib/python3.8/site-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<1.1,>=0.5->ten
[AI] (base) wcq@max:~$

```

图 2.15: 安装 TensorBoard 可视化工具

通过 pip list 命令，最终的安装包版本如下图 2.16 所示：

```

tensorboard                2.14.0
tensorboard-data-server    0.7.2
termcolor                  2.4.0
terminaltables             3.1.10
threadpoolctl              3.2.0
tiffio                     2023.7.10
tiktoken                   0.5.1
tokenizers                 0.15.0
tomli                      2.0.1
torch                     2.0.0+cu117
torchvision                0.15.1+cu117
torchviz                   0.0.2
tornado                   6.4
tqdm                      4.65.2

```

图 2.16: 最终的包版本

之后为了显示出整张图的网络信息，小组还使用了 networkx 和 matplotlib 的库，具体的安装步骤较为简单，主要是利用 pip networkx 的操作就可以直接实现安装，因此在这里不做过多赘述。

三、主要技术

本小组通过阅读论文之后，尝试自己将整个论文流程图给画出来，具体如图 3.1 所示：

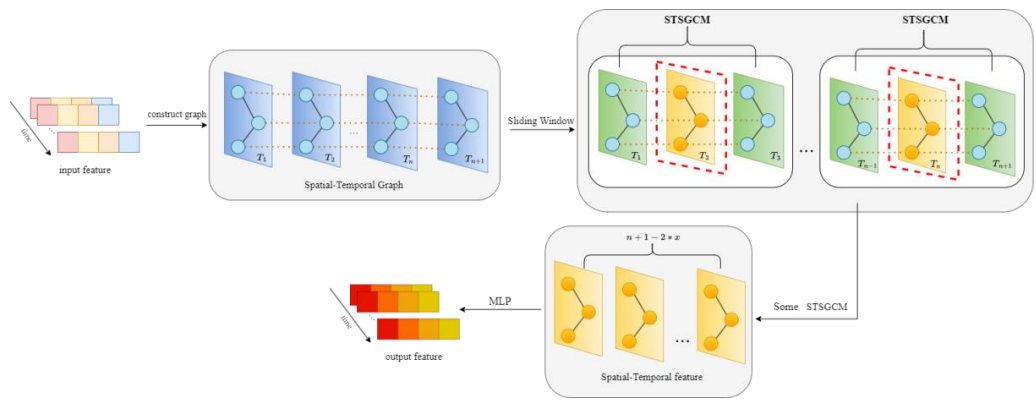


图 3.1: 时空网络流程图

具体的方法介绍如下所示：

3.1 局部时空图的创建

在交通流量预测等任务中，交通网络中的节点之间存在着复杂的时空依赖关系，即某个节点的状态不仅受到其相邻节点的影响，还受到时间因素的影响。因此，在处理这些数据时，需要考虑空间和时间两个维度的因素，并且对于不同时间段的数据，它们之间的关联性也可能会有所不同。图 3.2 展示的是一个空间-时间网络中红色节点的影响。具体来说，图中的每个节点代表一个传感器，节点之间的连接表示它们之间存在相互作用。通过观察红色节点与其周围节点的关系，可以了解到该节点对整个网络的影响程度以及它与其他节点之间的关联性。蓝色箭头表示网络中的边，并且也指示了在空间维度上的影响。棕色箭头表示红色节点对自身的下一时间步长的影响。绿色箭头表示跨越时间和空间维度，在下一个时间步长中，红色节点与其邻居之间的影响。 t_1 和 t_2 表示两个连续的时间步长。

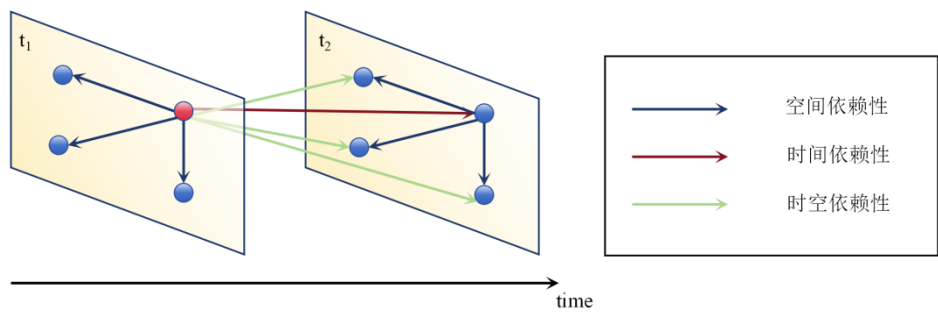


图 3.2: 时空网络中节点的影响

理解了空间-时间网络中节点的相互作用模式后,将这个模式应用于实际的局部时空图构建中,图 3.3 展示了如何基于空间和时间维度的相互作用来构建局部时空图结构。局部时空图是指在一个特定的时间段内,针对一个局部区域内的多个传感器数据所构成的一个图形化表示。通过对每个时刻的数据进行分组,构建出不同的局部时空图来表示不同时间段内的数据特征。为了直接捕获每个节点对其同时属于当前和相邻时间步长的相邻节点的影响,在相邻的时间步长上将所有节点与自己连接起来。通过在前一个时刻和下一个时刻将所有节点与它们自己连接起来,就可以得到一个局部的时空图。根据局部时空图的拓扑结构,可以直接捕获每个节点与其时空邻居之间的相关性。

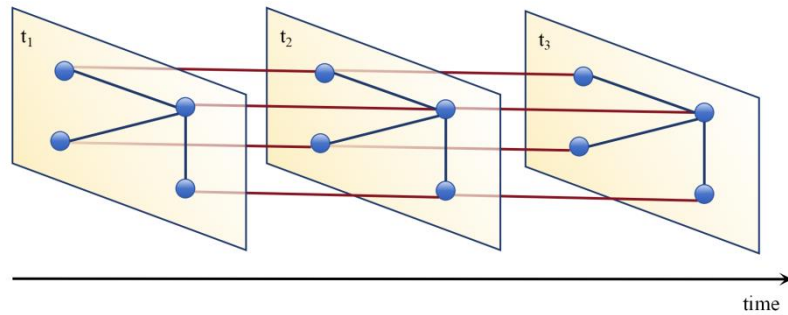


图 3.3: 局部空间-时间图的构建

局部时空图的具体创建步骤如下:

Step1: 时间节点自连接构建

从空间网络中获取物理实体或位置,然后连接每个节点在前一时刻和后一时刻的状态,形成局部化的空间网络。

Step2: 确定节点索引

对于空间图中的节点 i , 可以通过计算其在时序方向上的位置来确定其在局部化时空图中的新索引。其中, t 表示局部化时空图的时间步数 (即从当前时刻向前或向后连续的三个时刻), N 表示空间图中的总节点数, 因此通过计算公式

$$index = (t - 1)N + i$$

其中 $index$ 代表索引, 可以得出节点 i 在局部化时空图中的新索引。例如, 当 $t = 1$ 时, 节点 i 在局部化时空图中的新索引就是 i , 因为此时它对应的是前一时刻的空间图; 而当 $t = 2$ 时, 节点 i 在局部化时空图中的新索引就是 $N + i$, 因为它同时出现在了当前时刻和后一时刻的空间图中。

Step3: 构建邻接矩阵

构建描述每个节点状态的邻接矩阵。这里使用 $A \in \mathbb{W}^{N \times N}$ 来表示空间邻接矩阵。 $A \in \mathbb{W}^{3N \times 3N}$ 则表示在三个连续空间图上构造的局部时空图的邻接矩阵。如果在这个局部时空图中，两个节点相互连接，在邻接矩阵中对应的值被设为 1。局部化的邻接矩阵时空图可表述为：

$$A'_{i,j} = \begin{cases} 1, & v_i \rightarrow v_j \\ 0, & \text{other} \end{cases}$$

其中 v_i 表示局部时空图中的节点 i 。最终构建的邻接矩阵如图 3.4 所示，邻接矩阵的对角线是三个连续时间步长的空间网络的邻接矩阵。对角线的两侧表示每个节点与属于相邻时间步长的自身的连通性。

$A^{(t_1)}$	$A^{(t_1 \rightarrow t_2)}$	
$A^{(t_2 \rightarrow t_1)}$	$A^{(t_2)}$	$A^{(t_2 \rightarrow t_3)}$
	$A^{(t_3 \rightarrow t_2)}$	$A^{(t_3)}$

图 3.4: 邻接矩阵

3.2 时空嵌入

为了进一步提高模型的表现能力，论文中使用了时空嵌入技术。时空嵌入（spatial-temporal embedding）是一种用于处理时空数据的技术，它能够将空间和时间维度的信息映射到低维向量空间中，并在此基础上进行数据分析和建模。时空嵌入可以将时间和空间信息编码成低维向量表示，这样就可以更好地捕捉时间和空间之间的相互关系以及它们对数据的影响。同时，在加入时空嵌入后，模型可以更好地处理不同尺度的时间和空间信息，从而更好地适应真实世界中的时空数据分布。因此，在局部时空图的基础上加上时空嵌入可以使模型更加全面地考虑时间和空间因素，提高模型的预测能力和泛化性能。

在 3.1 中已经构建好的局部时空图下，时空嵌入具体步骤如下：

Step1: 创建两个嵌入矩阵

对于时空序列 $X_g \in \mathbb{R}^{N \times C \times T}$ ，创建一个可学习的时间嵌入矩阵 $T_{emb} \in \mathbb{R}^{C \times T}$ 和一个空间嵌入矩阵 $S_{emb} \in \mathbb{R}^{N \times C}$ 。这些矩阵包含必要的时间和空间信息，将用帮助模型捕捉空间-时间相关性。

Step2: 将嵌入矩阵加入到空间-时间网络序列

使用广播操作将时间和空间嵌入矩阵添加到空间-时间网络序列 X_g 中。则新的网络序列可以表示为

$$X_{g+T_{emb}+S_{emb}} = X_g + T_{emb} + S_{emb} \in \mathbb{R}^{N \times C \times T}$$

3.3 时空同步图卷积模块

具体模块的方法如下图所示：

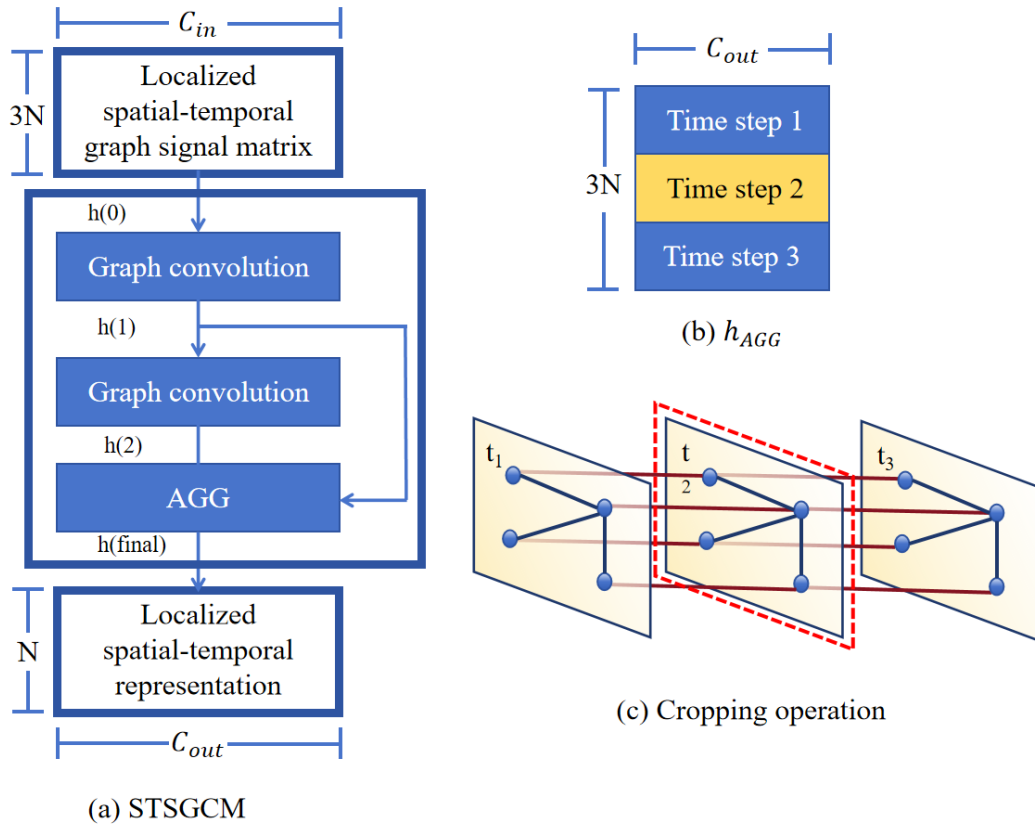


图 3.5: (a)是具有两个图卷积操作的时空同步图卷积模块的架构示例。(b)表示聚合操作的输出。(c)是聚合层中裁剪操作的示例。

由于经典卷积神经网络不能处理图等非结构化数据，因此考虑将卷积操作拓展到图结构数据中，主流的实现思路为谱域图卷积和空域图卷积。因为谱域

图卷积假定图结构固定，不能改变节点之间的权重，而对于城市交通的空间时间网络数据来说，图结构可能会变化；同时，空域图卷积不依靠图谱卷积理论，直接在空间上定义卷积操作，不仅可以应用于无向图，也可以应用于有向图，具备较强的灵活性。因此采用空域图卷积作为实现思路。

论文中提出了时空同步图卷积模块（STSGCM），通过构建该模块来捕获局部时空的相关性。

Step1: 输入图信号矩阵的特征数

将局部时空图构建的邻接矩阵作为模块的输入。

Step2: 进行图卷积操作

STSGCM 由一组基于空域的图卷积操作组成。通过在顶点域中定义图卷积运算来聚合时空网络中的局部时空特征。

图卷积运算可以在相邻的时间步长上将每个节点的特征与其相邻节点进行聚合。聚合函数是线性的，其权值等于节点与其相邻节点之间的边的权值。

其中，每一层的图卷积的运算公式如下所示：

$$GCN(h^{(l-1)}) = h^{(l)} = \sigma(A'h^{(l-1)}W + b) \in \mathbb{R}^{3N \times C'}$$

其中， $A' \in \mathbb{R}^{3N \times 3N}$ 表示局部时空图构建的邻接矩阵， $h^{(l-1)} \in \mathbb{R}^{3N \times C}$ 表示是第 1 个图卷积层的输入， $W \in \mathbb{R}^{C \times C'}$ 和 $b \in \mathbb{R}^{C'}$ 是可学习参数， σ 指的是激活函数。论文选择使用 GLU 作为图卷积层的激活函数。

GLU 的一般形式如下所示：

$$GCN(x, W, V, b, c) = (xW + b) \otimes \text{sigmoid}(xV + c)$$

即 x 的两个线性映射逐点相乘，并且其中一个部分先进行 sigmoid 激活函数的运算。

将 GLU 门控线性单元作为激活函数，每一层的图卷积的运算公式如下所示：

$$GCN(h^{(l-1)}) = h^{(l)} = (A'h^{(l-1)}W_1 + b_1) \otimes \text{sigmoid}(A'h^{(l-1)}W_2 + b_2)$$

其中， $W_1 \in \mathbb{R}^{C \times C'}$, $W_2 \in \mathbb{R}^{C \times C'}$, $b_1 \in \mathbb{R}^{C'}$, $b_2 \in \mathbb{R}^{C'}$ 是可学习参数， sigmoid 表示采用了 $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$ 作为激活函数。因此，GLU 控制了哪个节点的信息可以传递到下一层。

Step3: 部署带有激活函数的聚合层

然后，部署一个带有激活函数的聚合层，将节点的特征转化为一个新的空间。在网络中，将输入直接连接到输出，以允许信息在不同层之间跳跃传递，即跳跃连接（Skip Connection），也被称为残差连接（Residual Connection）。在有跳跃连接的情况下，允许低级特征直接传递到最终的全连接层，对于捕获细节和纹理信息非常有用，使得网络可以更好地利用低级特征，从而提高性能和训练速度。

对于具有 L 个图卷积操作的 STSGCM，每个图卷积操作的输出将被扩展到聚合层(AGG)，以增加图卷积操作的接受域，来捕获局部时空相关性。并且在局域化时空图的每个节点上添加了自循环，以便图卷积运算在聚合特征时考虑到自身的特征。

其中 AGG 聚合层包括了两个部分，聚合和裁剪。

聚合操作选择了最大池化来实现。对于 STSGCM 中所有的图卷积的输出，最大聚合操作可表示为：

$$h_{AGG} = \max(h^{(1)}, h^{(2)}, \dots, h^{(L)}) \in \mathbb{R}^{3N \times C_{out}}$$

其中， C_{out} 表示图卷积运算的核数。这也就解释了上文两个图卷积操作的时空同步图卷积模块 $h^{(1)}$ 到 AGG 层存在跳跃连接的原因。

裁剪操作是对于聚合操作的结果，去除前一时刻和后一时刻节点的所有特征，只保留中间时刻的节点。因为图卷积操作已经聚合了来自前一个和下一个时间步骤的信息，即使裁剪了前后两个时间步的信息，每个节点也已经包含了局部的时空相关性。如果保留所有相邻时间步长的特征，会导致模型中存在大量冗余信息，严重影响模型的性能。

综上所述，具有至少两个叠加图卷积操作的 STSGCM 可以直接对时空网络中节点三种不同类型的相关性进行建模。

3.4 时空同步图卷积层

为了捕捉整个网络序列的长期时空相关性：使用滑动窗口，来剪切不同的时间段；并且由于时空数据的异质性，使用多个 STSGCM 来模拟不同时期，而不是所有时期共享一个。因此论文中部署了一组 STSGCM 作为时空同步图卷积层

(STSGCL)来提取远程时空特征,

Step1: 输入数据矩阵

将 STSGCL 的输入矩阵表示为 $X \in \mathbb{R}^{T \times N \times C}$, 其中 T 表示时间步, N 表示节点个数, C 表示特征数。首先为每个 STSGCL 添加时空嵌入, 每个时空网络序列可表示为 $X' \in \mathbb{R}^{3 \times N \times C}$, 再将其重塑为 $X'_{reshape} \in \mathbb{R}^{3N \times C}$, 可以直接与局部时空图一起输入 STSGCM 中。

Step2: 进行滑动窗口操作以裁切数据

原始时间序列数据有 T 个时间段, 以 3 个时间片作为裁切单元, 滑动窗口将得到 T-2 个裁切结果, 以此作为时空网络的输入。STSGCL 在 T-2 个局部时空图上部署 T-2 个 STSGCM, 以捕获这些时空网络序列中的局域时空相关性。

Step3: 结果输出的连接

将所有 STSGCM 的输出串联成矩阵, 作为 STSGCL 的输出, 可以表示为:

$$M = [M_1, M_2, \dots, M_{T-2}] \in \mathbb{R}^{(T-2) \times N \times C_{out}}$$

其中, $M_i \in \mathbb{R}^{N \times C_{out}}$ 表示第 i 个 STSGCM 的输出。

综上所述, 通过叠加多个 STSGCL 经过多次时空同步图卷积层运算后, 每个节点将包含以自身为中心的局域时空关联, 因此可以建立一个能够捕获复杂时空相关性和时空异质性的分层模型。

3.5 损失函数介绍

在论文中, 是直接采用简单的 Huber Loss 进行设计对应的损失函数, 此函数广泛应用于回归问题, 尤其是对那些异常值比较敏感的情况下。具体的公式如下所示:

$$L(Y, \check{Y}) = \begin{cases} \frac{1}{2}(Y - \check{Y})^2 & |Y - \check{Y}| \leq \delta \\ \delta|Y - \check{Y}| - \frac{1}{2}\delta^2 & otherwise \end{cases}$$

其中 Y 是预测值, 而 \check{Y} 是真实值, δ 是对应的参数, 需要认为的调整。

对于 Huber Loss 损失函数, 需要对 δ 进行相关调参, 才能去找到对应的最佳值。但是在实际中, 为了让 Loss 对离群值更加鲁棒, 小组采用 SmoothL1Loss 作为损失函数, 它的本质就是让 Huber Loss 中的 δ 变为 1。

3.6 增强模型性能的相关操作

在整个模型训练时候，论文采用了以下几种方法及逆行增强模型的性能。具体的介绍如下所示：

1. 对于 STSGCN 中的图卷积操作，邻接矩阵 A 决定聚合的权重。然而，每个节点对其邻居节点具有不同的影响幅度。如果邻接矩阵仅包含 0 和 1，则聚合操作可能受到限制。如果局部时空图中的两个节点是连通的，即使它们在某个时间段没有相关性，它们的特征也会被聚合。因此在 STSGCN 中添加了一个可学习的掩码矩阵 W_{mask} 来调整聚合权重，使聚合更加合理。具体的公式如下所示：

$$A_{adjusted} = W_{mask} \times A' \in R^{3N \times 3N}$$

其中 W_{mask} 是可学习的掩码矩阵，其中的 \times 表示矩阵的各个元素相乘。

2. 在网络的输入层中，该网络添加一个全连接层，将输入转换到高维空间，这可以提高网络的表示能力。

四、程序实现和实验结果展示

在本节中，将会去展示整个复现代码的流程、复现代码时候所遇到的问题和相关训练数据集的介绍。

4.1 数据集介绍

本次主要采用了 PEMS03、PEMS04、PEMS07、PEMS08 这四个数据集。具体的数据集介绍如下所示：

PEMS 系列数据集的数据采集方式：由分布在不同地点的若干个探测器每隔 5 分钟采集一次，连续采集若干天。例如 PEMS04 是由 307 个探测器每隔 5 分钟采集一次数据，共采集 59 天产生的交通流量数据；PEMS08 是由 170 个探测器每隔 5 分钟采集一次，共采集 62 天产生的数据。

并且，每个探测器每次采集的数据包含三个维度的特征，分别为：流量、平均速度和平均占有率。

这里对平均占有率进行说明：占有率分为时间占有率和空间占有率，是衡量道路被利用程度的重要指标。下面对这两个占有率进行详细介绍：

1. 空间占有率是指在观测时间内，观测路段中各车辆所占道路面积总量与区域道路面积总量的比值；
2. 时间占有率是指在观测时间内通过道路某断面的累计时间与该段时间的比值。

针对于这四个数据集，详细介绍如下表 4.1 所示：

表 4.1：数据集介绍

数据集	探测器数量	时间范围
PEMS03	358	9/1/2018 - 11/30/2018
PEMS04	307	1/1/2018 - 2/28/2018
PEMS07	883	5/1/2017 - 8/31/2017
PEMS08	170	7/1/2016 - 8/31/2016

基于上述表格，这里主要以 PEMS04 进行说明，小组获取 PEMS04 数据集的维度是 (16992, 307, 3)。其中 16992 代表的是时间 ($59 \times 24 \times 12 = 16992$)，而 307 代表的是探测器的数量，而 3 就是代表对应的特征维度数量，也就是上文提及到的特征维度（流量、平均速度、平均占有率）。通过 np.load 进行加载对应的 npz 文件，具体如下图 4.1 所示，可以观察到 data 的维度（因为这里是训练集的大小，所以是 10195，具体的训练集，验证集，测试集的划分在后续进行介绍）：

```
> data = {ndarray: (10195, 307, 3)} [[6.20e+01 7.70e-03 6.79e+01], [5.60e+01 1.12e-02 6.84e+01], [9.00e+01 1.43e-02 6.84e+01], ...]
10 pred_length = (int) 12
10 train_length = (int) 12
```

图 4.1：PEMS04 数据集呈现

4.2 程序实现过程

对于整个项目的整体构思，我们小组分为了以下步骤进行：数据集的加载，算法的实现，训练模型，测试模型。具体流程图如下图 4.2 所示：

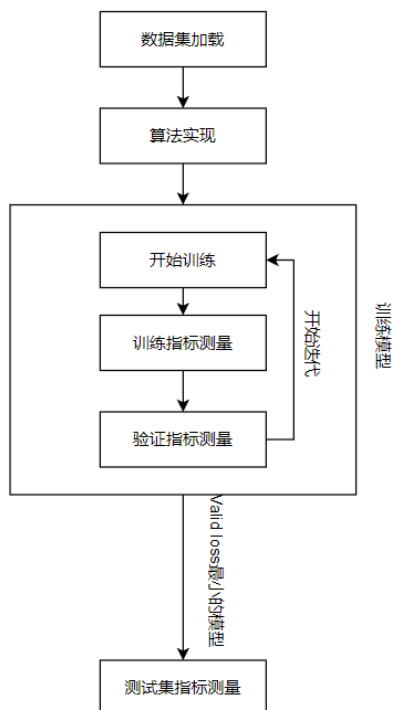


图 4.2: 算法流程图呈现

在整个程序实现过程，将分为以下几个步骤进行详细的阐述和说明，分别是数据集加载，STSGCN 算法的实现，训练过程的实现。由于验证集过程和测试集进行测试的过程与训练过程代码类似，因此不做过多的介绍和说明。

● 数据集加载:

这里主要以 PEMS03 的数据集为例，详细去介绍对应的数据集加载的函数说明。具体 PEMS03 数据集的文件如下图 4.3 所示:

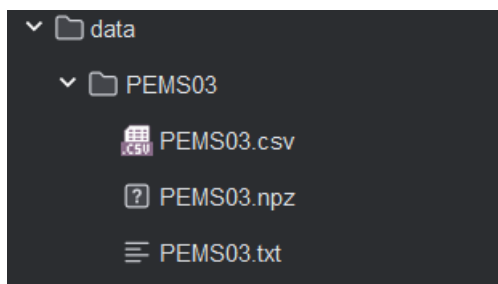


图 4.3: 数据集文件呈现

其中 txt 文件中代表的是所有节点的编号，而 csv 文件代表的是各个探测器（节点）之间的距离和关联关系，而 npz 文件中储存的是交通流量的信息，我们需要通过 `np.load()` 函数进行加载其中的数据。

1. 首先是构造整个数据集的关联矩阵，主要是根据 csv 文件中，各个节点之间的关联关系进行相关的构建，如果存在 txt 文件（只有 PEMS03 存在 txt 文件）就构建对应的探测器编号的映射，再进行构造对应的邻接矩阵。如果不存在，就直接从 csv 中构造数据集对应的临界矩阵。具体的代码如下所示：

```
1. def get_adjacency_matrix(distance_df_filename, num_of_vertices,
2.                             type_='connectivity', id_filename=None):
3.     """
4.
5.     :param distance_df_filename: str, path of the csv file contains edges information
6.     :param num_of_vertices: int, the number of vertices
7.     :param type_: str, {connectivity, distance}
8.     :param id_filename: str
9.     :return: np.ndarray, adjacency matrix
10.    """
11.    A = np.zeros((int(num_of_vertices), int(num_of_vertices)),
12.                  dtype=np.float32)
13.
14.    if id_filename:
15.        with open(id_filename, 'r') as f:
16.            id_dict = {int(i): idx
17.                        for idx, i in enumerate(f.read().strip().split("\n"))}
18.        with open(distance_df_filename, 'r') as f:
19.            f.readline()
20.            reader = csv.reader(f)
21.            for row in reader:
22.                if len(row) != 3 or row[0] == "" or row[1] == "" or row[2] == "":
23.                    continue
24.                i, j, distance = int(row[0]), int(row[1]), float(row[2])
25.                A[id_dict[i], id_dict[j]] = 1
26.                A[id_dict[j], id_dict[i]] = 1
27.        return A
28.
29.    # Fills cells in the matrix with distances.
30.    with open(distance_df_filename, 'r') as f:
31.        f.readline()
32.        reader = csv.reader(f)
33.        for row in reader:
34.            if len(row) != 3:
35.                continue
36.            i, j, distance = int(row[0]), int(row[1]), float(row[2])
37.            if type_ == 'connectivity':
38.                A[i, j] = 1
```

```

39.         A[j, i] = 1
40.         elif type == 'distance':
41.             A[i, j] = 1 / distance
42.             A[j, i] = 1 / distance
43.         else:
44.             raise ValueError("type_ error, must be "
45.                               "connectivity or distance!")
46.     return A

```

而通过这方法构建出来的仅仅只是一个时间段的邻接矩阵,它是属于 $N \times N$ 维度的。在这之后,需要将它拓展为 $3N \times 3N$ 的邻接矩阵,具体的算法思想其实就是相应的矩阵扩充和对应矩阵值的变化。具体的代码如下所示:

```

1.  Def construct_adj(A, steps):
2.     ""
3.     construct a bigger adjacency matrix using the given matrix
4.     :param A: np.ndarray, adjacency matrix, shape is (N, N)
5.     :param steps: how many times of the does the new adj mx bigger than A
6.     :return: new adjacency matrix: csr_matrix, shape is (N * steps, N * steps)
7.     ""
8.     N = len(A)
9.     adj = np.zeros([N * steps] * 2)
10.
11.    for i in range(steps):
12.        adj[i * N: (i + 1) * N, i * N: (i + 1) * N] = A
13.
14.    for i in range(N):
15.        for k in range(steps - 1):
16.            adj[k * N + i, (k + 1) * N + i] = 1
17.            adj[(k + 1) * N + i, k * N + i] = 1
18.
19.    for i in range(len(adj)):
20.        adj[i, i] = 1
21.
22.    # shape is (N * steps, N * steps)
23.    return adj

```

2. 进行切分后续的训练集、验证集、测试集,按照 6:2:2 的方式进行切分对应的数据集。并且在一开始对相应的数据集进行相关的归一化操作,具体如下所示:

$$\frac{X - \text{mean}(X)}{\text{std}(X)}$$

具体的代码如下所示:

```

1.  def generate_from_data(data, length, transformer):

```

- ```

2. mean = None
3. std = None
4. train_line, val_line = int(length * 0.6), int(length * 0.8)
5. for line1, line2 in ((0, train_line),
6. (train_line, val_line),
7. (val_line, length)):
8. x, y = generate_seq(data['data'][line1: line2], 12, 12)
9. if transformer:
10. x = transformer(x)
11. y = transformer(y)
12. if mean is None:
13. mean = x.mean()
14. if std is None:
15. std = x.std()
16. yield (x - mean) / std, y

```
3. 加载得到对应的数据之后，需要将对应的变量 (B,N,C) 转化为 (K, T, N, C) 的格式。其中 T 固定为 12，代表的是 1 小时 (5min\*12=60min=1h)，具体如下所示：

```

1. def generate_seq(data, train_length, pred_length):
2. """
3.
4. :param data: shape is (K,N,C)
5. :param train_length: int T
6. :param pred_length: int T'
7. :return:
8. """
9. # shape is (K - T - T' + 1, T + T', N, C)
10. seq = np.concatenate([np.expand_dims(
11. data[i: i + train_length + pred_length], 0)
12. for i in range(data.shape[0] - train_length - pred_length + 1)],
13. axis=0)[:, :, 0: 1]
14. return np.split(seq, 2, axis=1)

```

4. 最终的获取数据集的函数如下所示：

```

1. def generate_data(graph_signal_matrix_filename, transformer=None):
2. """
3.
4. :param graph_signal_matrix_filename: str
5. :param transformer: str
6. :return: shape is (num_of_samples, 12, num_of_vertices, 1)
7. """
8. data = np.load(graph_signal_matrix_filename)
9. keys = data.keys()

```

```

10. if 'train' in keys and 'val' in keys and 'test' in keys:
11. for i in generate_from_train_val_test(data, transformer):
12. yield i
13. elif 'data' in keys:
14. length = data['data'].shape[0]
15. for i in generate_from_data(data, length, transformer):
16. yield i
17. else:
18. raise KeyError("neither data nor train, val, test is in the data")

```

相关的 Dataloader 加载具体如下所示，该行代码能够成功返回 Dataloader。每次通过迭代器获取 Dataloader 的值，会得到前 1 小时的值和后 1 小时的值（对应于 x 和 y）。

```

1. loaders.append(torch.utils.data.DataLoader(torch.utils.data.TensorDataset(x, y),
2. batch_size=batch_size, shuffle=True))

```

### ● STSGCN 算法实现：

该部分将详细去介绍 STSGCN 的算法流程和相关的代码编写。具体介绍如下所示：

1. 位置嵌入：这里的 embedding 主要包括 spatial embedding 和 temporal embedding，之后输入特征初始化时候，需要将输入特征加上 spatial embedding 和 temporal embedding。这里主要是对 embedding 中的参数进行相应的初始化，保证信息在正向传播和反向传播都能够保持稳定。具体的代码如下所示：

```

1. def position_embedding(data,
2. input_length, num_of_vertices, embedding_size,
3. temporal=True, spatial=True):
4. """
5. 位置编码
6. :param data: tensor shape is (B,T,N,C)
7. :param input_length: int, length of time series, T
8. :param num_of_vertices: int, N
9. :param embedding_size: int, C
10. :param temporal: bool, whether equip this type of embeddings
11. :param spatial: bool, whether equip this type of embeddings
12. :return:
13. data: output shape is (B, T, N, C)
14. """
15. temporal_emb = None
16. spatial_emb = None
17.
18. if temporal:
19. # shape is (1, T, 1, C)

```

```

20. temporal_emb = torch.empty(1, input_length, 1, embedding_size).to(torch.device("cuda:0"))
21. torch.nn.init.xavier_uniform_(temporal_emb, gain=0.0003)
22. if spatial:
23. # shape is (1, 1, N, C)
24. spatial_emb = torch.empty(1, 1, num_of_vertices, embedding_size).to(torch.device("cuda:0"))
25. torch.nn.init.xavier_uniform_(spatial_emb, gain=0.0003)
26.
27. if temporal_emb is not None:
28. data = torch.add(data, temporal_emb)
29. if spatial_emb is not None:
30. data = torch.add(data, spatial_emb)
31.
32. return data

```

2. GCN 操作：此处没有直接调用相应的图神经网络的框架（例如 Pyg 等），而是通过对应的矩阵运算，来实现相应图神经网络的消息传递机制。并且根据所设置的激活函数（GLU 和 RELU 这两种激活函数）的不同，输出不同的结果。具体代码如下所示：

```

1. class gcn_operation(nn.Module):
2. def __init__(self, adj, num_of_filter, num_of_features, num_of_vertices,
3. activation):
4. """
5. GCN 操作
6. :param adj: tensor shape is (3N, 3N)
7. :param num_of_filter: int, C'
8. :param num_of_features: int, C
9. :param num_of_vertices: int, N
10. :param activation: str, {'GLU','relu'}
11. """
12. super().__init__()
13. assert activation in {'GLU', 'relu'}
14.
15. self.activation = activation
16. self.num_of_features = num_of_features
17. self.num_of_filter = num_of_filter
18. self.adj = adj
19.
20. if activation == 'GLU':
21. self.Liner = nn.Linear(num_of_features, 2 * num_of_filter)
22.
23.

```

```

24. elif activation == 'relu':
25. self.Liner = nn.Linear(num_of_features, num_of_filter)
26.
27. def forward(self, data):
28. '''
29.
30. :param data: tensor, shape is(3N, B, C)
31. :return:
32. output shape is (3N, B, C')
33. '''
34. data=torch.einsum('nm, mbc->nbc', self.adj.to(data.device), data)
35.
36.
37. if self.activation == 'GLU':
38. # shape is (3N, B, 2C')
39. data = self.Liner(data)
40.
41. # shape is (3N, B, C') , (3N, B, C')
42. lhs, rhs = torch.split(data, split_size_or_sections=self.num_of_filter, dim=-1)
43.
44. # shape is (3N, B, C')
45. return lhs * torch.sigmoid(rhs)
46.
47. elif self.activation == 'relu':
48. # shape is (3N, B, C')
49. data = self.Liner(data)
50.
51. # shape is (3N, B, C')
52. return torch.relu(data)

```

3. STSGCM 模块：其中该模块的核心步骤主要如下图 4.4 所示：

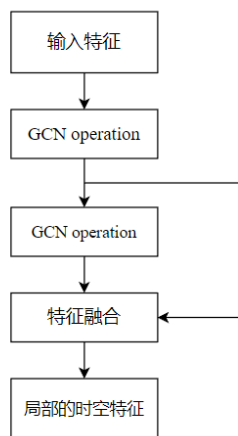


图 4.4: STSGCM 算法呈现

根据第二步所实现的 GCN Operation，成功搭建 STSGCM 网络。输入特征首

先经过了多层的 GCN 网络，之后将这些得到的特征进行相关的融合操作，在融合操作之后，还进行裁剪的工作，从而只提取到中间时间片的特征信息。具体代码如下所示：

```

1. class stsgcm(nn.Module):
2. def __init__(self, adj, filters, num_of_features, num_of_vertices,
3. activation):
4. """
5. STSGCM, multiple stacked gcn layers with cropping and max operation
6. :param adj: tensor, shape is (3N, 3N)
7. :param filters: list[int], list of C'
8. :param num_of_features: int, C
9. :param num_of_vertices: int, N
10. :param activation: str, {'GLU', 'relu'}
11. """
12. super().__init__()
13. self.adj = adj
14. self.filters = filters
15. self.num_of_vertices = num_of_vertices
16. feture = [filters[i] for i in range(len(filters) - 1)]
17. feture.insert(0, num_of_features)
18. self.gcn_operation = nn.ModuleList([gcn_operation(adj, filters[i], feture[i], num_of_vertices, activation)
19. for i in range(len(filters))])
20.
21. def forward(self, data):
22. """
23. :param data: tensor, shape is (3N, B, C)
24. :return: output shape is (3N, B, C')
25. """
26. need_concat = []
27. for layer in self.gcn_operation:
28. data = layer(data)
29. need_concat.append(data)
30.
31. # shape of each element is (1, N, B, C')
32. need_concat = [
33. torch.unsqueeze(i[self.num_of_vertices:2 * self.num_of_vertices, :, :],
34. dim=0)
35. for i in need_concat
36.]
37.
38. # shape is (N, B, C')
39. result = torch.max(torch.cat(need_concat, dim=0), dim=0).values

```

40.

41. `return result`

4. STSGCL 操作：该网络的输入特征的大小是(B,T,N,C)，所以采用滑动窗口的策略，将每三个时间片段，作为一个特征向量，并且将它输入到之前定义好的 STSGCM。而之后，将这些特征向量进行相应的融合（这里主要采用的是简单的拼接），最后得到的特征大小是(B,T-2,N,C)。具体的代码如下所示：

```

1. class stsgcl(nn.Module):
2. def __init__(self, adj, T, num_of_vertices, num_of_features, filters,
3. module_type, activation, temporal_emb=True, spatial_emb=True):
4. """
5. :param adj: tensor, shape is (3N, 3N)
6. :param T: int, length of time series, T
7. :param num_of_vertices: int, N
8. :param num_of_features: int, C
9. :param filters: list[int], list of C'
10. :param module_type: str, {'sharing', 'individual'}
11. :param activation: str, {'GLU', 'relu'}
12. :param temporal_emb: bool
13. :param spatial_emb: bool
14. """
15. super(stsgcl, self).__init__()
16.
17. assert module_type in {'sharing', 'individual'}
18. self.adj = adj
19. if module_type == 'sharing':
20. self.layer = sthgcg_layer_sharing(
21. adj, T, num_of_vertices, num_of_features, filters,
22. activation, temporal_emb, spatial_emb
23.)
24. elif module_type == 'individual':
25. self.layer = sthgcg_layer_individual(
26. adj, T, num_of_vertices, num_of_features, filters,
27. activation, temporal_emb, spatial_emb
28.)
29.
30. def forward(self, data):
31. """
32. :param data: tensor, shape is (B, T, N, C)
33. :param adj: tensor, shape is (3N, 3N)
34. :return: output shape is (B, T-2, N, C')
35. """
36. return self.layer(data)

```



5. 函数整合：将上述这些函数进行相关的整合，最后形成了最后的 STSGCN 模型，具体如下所示：

```

1. class stsgcn(nn.Module):
2. def __init__(self, adj, input_length, num_of_vertices, num_of_features,
3. filter_list, module_type, activation,
4. use_mask=True, mask_init_value=None,
5. temporal_emb=True, spatial_emb=True, rho=1, predict_length=12):
6. """
7. stsgcn
8. :param adj: tensor, shape is (3N, 3N)
9. :param input_length: int T
10. :param num_of_vertices: int, N
11. :param num_of_features: int, C
12. :param filter_list: list[int][int], list of C'
13. :param module_type: str, {'sharing', 'individual'}
14. :param activation: str, {'GLU', 'relu'}
15. :param use_mask: bool
16. :param mask_init_value: float
17. :param temporal_emb: bool
18. :param spatial_emb: bool
19. :param rho: float
20. :param predict_length: int T'
21. """
22. super(stsgcn, self).__init__()
23.
24. self.adj=adj
25. self.filter_list = filter_list
26. self.predict_length = predict_length
27. self.rho = rho
28. T = [input_length - 2 * i for i in range(len(filter_list))]
29. features = [filter_list[i][-1] for i in range(len(filter_list) - 1)]
30. features.insert(0, num_of_features)
31.
32. self.stsgcl = nn.ModuleList([stsgcl(adj, T[i], num_of_vertices, features[i],
33. filter_list[i], module_type, activation, temporal_emb, spatial_em
34. b) for i in
35. range(len(filter_list))])
36.
37. self.output = output_layer(num_of_vertices, input_length-
38. 2*len(filter_list), num_of_features, 128, 1)
39.
40. # mask shape is (3N, 3N)
41. if use_mask:
42. if mask_init_value is None:

```

```

40. raise ValueError("mask init value is None!")
41. self.mask = torch.empty(3 * num_of_vertices, 3 * num_of_vertices)
42. self.mask = mask_init_value
43.
44. def forward(self, data):
45. """
46. :param data: tensor, shape is (B, T, N, C)
47. :param label: shape is (B, T, N)
48. :return: output, loss
49. """
50.
51. # shape is (3N, 3N)
52. if self.mask is not None:
53. self.adj = self.adj * self.mask
54.
55. for layer in self.stsgcl:
56. data = layer(data)
57.
58. # shape is (B, 1, N)
59. need_concat = []
60. for i in range(self.predict_length):
61. need_concat.append(self.output(data))
62.
63. # shape is (B, T, N)
64. data = torch.cat(need_concat, dim=1)
65.
66. return data

```

### ● Train 函数实现:

最终实现相应的 Train 函数，具体的 loss 函数定义，优化器的定义，和指标的测量，相关的训练过程如下所示：

```

1. def training(epochs, config):
2.
3. global global_epoch
4. writer = SummaryWriter('./log/PEMS03', flush_secs=20)
5. lowest_val_loss = 1e6
6. criterion = torch.nn.SmoothL1Loss()
7. base_lr = config['learning_rate']
8.
9. optimizer = optim.Adam(net.parameters(), lr=base_lr, eps=1.0e-
10. 8, weight_decay=0, amsgrad=False)
11. lr_scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer=optimizer,

```

```
11. milestones=[15,40,70,105,145],
12. gamma=0.3)
13. exp=1
14.
15. for epoch in range(epochs):
16. t = time.time()
17. info = [global_epoch]
18. train_loss = []
19. train_mae = []
20. train_mape = []
21. train_rmse = []
22. net.train()
23.
24. for idx, data in enumerate(train_loader):
25. x, y = data
26. x, y = x.to(ctx), y.to(ctx)
27. output = net(x)
28. loss=criterion(output, y)
29. optimizer.zero_grad()
30. loss.backward()
31. torch.nn.utils.clip_grad_norm_(net.parameters(), 5.0)
32. optimizer.step()
33.
34. tmae = utils.masked_mae(output, y).item()
35. tmape = utils.masked_mape(output, y , 0.0).item()
36. trmse = utils.masked_rmse(output, y , 0.0).item()
37.
38. train_loss.append(loss.item())
39. train_mae.append(tmae)
40. train_mape.append(tmape)
41. train_rmse.append(trmse)
42.
43. if idx % 200 == 0:
44. print(f'Epoch: {idx}, Loss: {train_loss[-1]}, MAE: {train_mae[-1]}, "
45. f'MAPE: {train_mape[-1]}, RMSE: {train_rmse[-1]}')
46.
47. lr_scheduler.step()
48.
49. valid_mae = []
50. valid_mape = []
51. valid_rmse = []
52. net.eval()
53. for idx, data in enumerate(val_loader):
54. x,y = data
```

```
55. x,y = x.to(ctx), y.to(ctx)
56. output = net(x)
57.
58. tmae = utils.masked_mae(output, y).item()
59. tmape = utils.masked_mape(output, y , 0.0).item()
60. trmse = utils.masked_rmse(output, y , 0.0).item()
61.
62. valid_mae.append(tmae)
63. valid_mape.append(tmape)
64. valid_rmse.append(trmse)
65.
66. mtrain_loss = np.mean(train_loss)
67. mtrain_mae = np.mean(train_mae)
68. mtrain_mape = np.mean(train_mape)
69. mtrain_rmse = np.mean(train_rmse)
70.
71. mvalid_loss = np.mean(valid_mae)
72. mvalid_mape = np.mean(valid_mape)
73. mvalid_rmse = np.mean(valid_rmse)
74.
75. print(f'Epoch: {epoch}, Train Loss: {mtrain_loss}, Train MAE: {mtrain_mae}, "
76. f'Train MAPE: {mtrain_mape}, Train RMSE: {mtrain_rmse}, \n"
77. f'Valid MAE: {mvalid_loss}, "
78. f'Valid MAPE: {mvalid_mape}, Valid RMSE: {mvalid_rmse}')
79.
80. writer.add_scalar('train/Loss', mtrain_loss, epoch)
81. writer.add_scalar('train/MAE', mtrain_mae, epoch)
82. writer.add_scalar('train/MAPE', mtrain_mape, epoch)
83. writer.add_scalar('train/RMSE', mtrain_rmse, epoch)
84.
85. writer.add_scalar('valid/MAE', mvalid_loss, epoch)
86. writer.add_scalar('valid/MAPE', mvalid_mape, epoch)
87. writer.add_scalar('valid/RMSE', mvalid_rmse, epoch)
88.
89. if mvalid_loss <= lowest_val_loss:
90. lowest_val_loss = mvalid_loss
91. torch.save(net.state_dict(), 'save/exp_{}_{}.pth'.format(exp, str(round(lowest_val_loss, 2))))
92. exp+=1
93.
94. global_epoch += 1
```

## 4.3 实验结果展示

### ● 数据展示:

首先利用 `networkx` 进行相关的可视化操作, 以 PEMS08 的数据集为例, 具体如下图 4.5 所示:

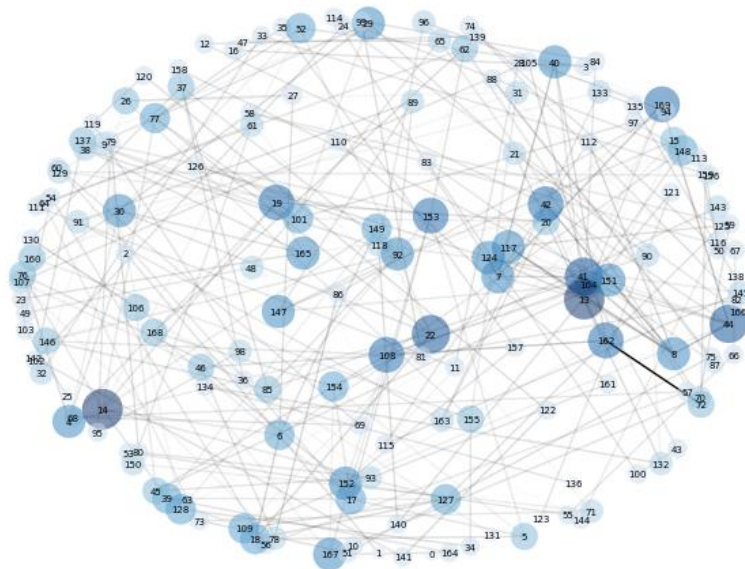


图 4.5: PEMS08 数据集交通节点可视化呈现

从该图密集的连接, 可以看到该数据集所呈现出来交通网络的复杂性。

之后采用相应的可视化的操作, 将原始的交通的流量相关变化显示出来, 具体如下图 4.6 所示:

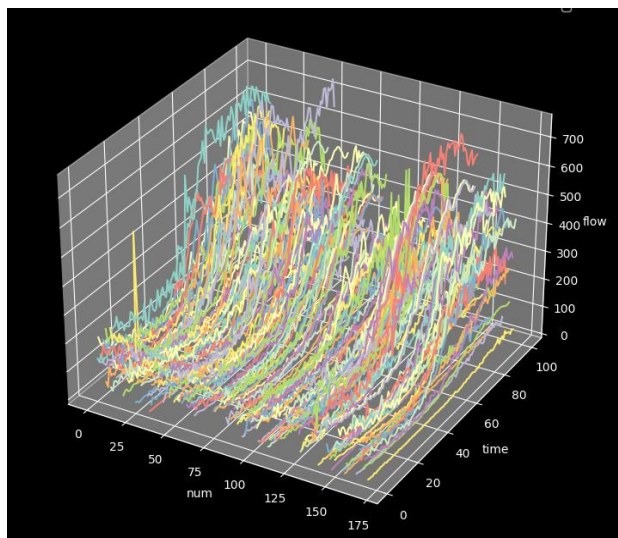


图 4.6: PEMS08 数据集流量可视化

上图代表的是所有探测点的流量和时间的变化趋势。由于探测点过多，不太明显，所以我们小组专门可视化了第一个探测点的流量变化趋势，具体如下图 4.7 所示：

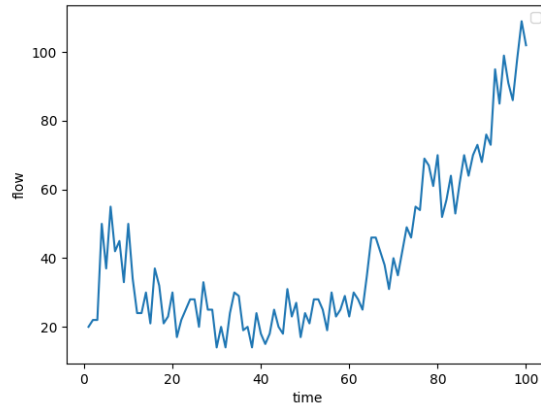


图 4.7: PEMS08 数据集第一个传感器呈现

#### ● 训练过程分析：

在真实训练中，模型所占 GPU 显存为 7G 左右，具体如下图 4.8 所示：

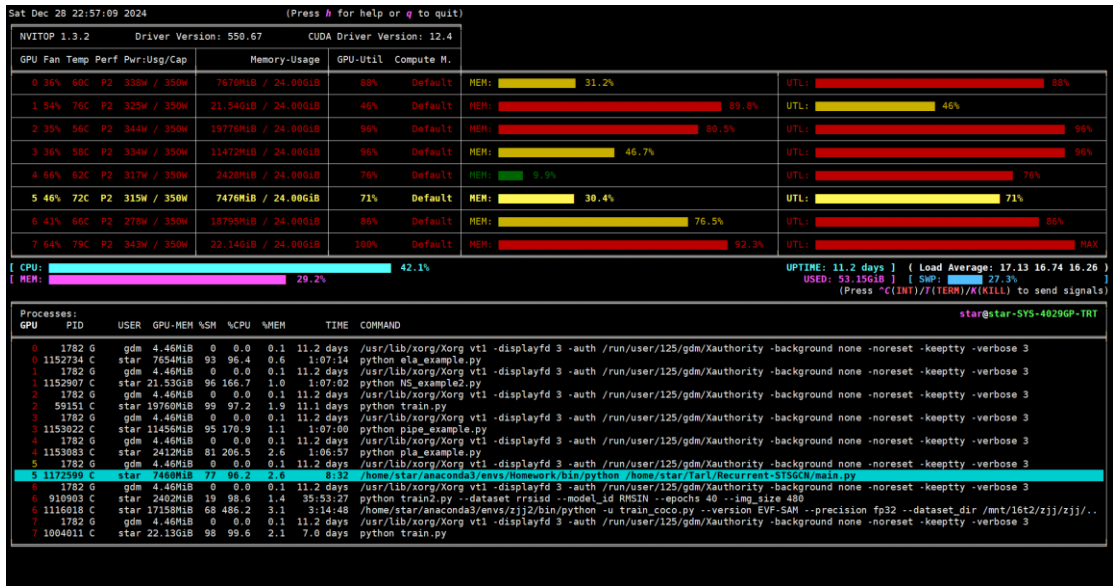


图 4.8: 训练所占资源呈现

在训练过程中，同时在 tensorboard 中实时观察 loss 和各个指标的相关变化，具体如下图 4.9 所示：

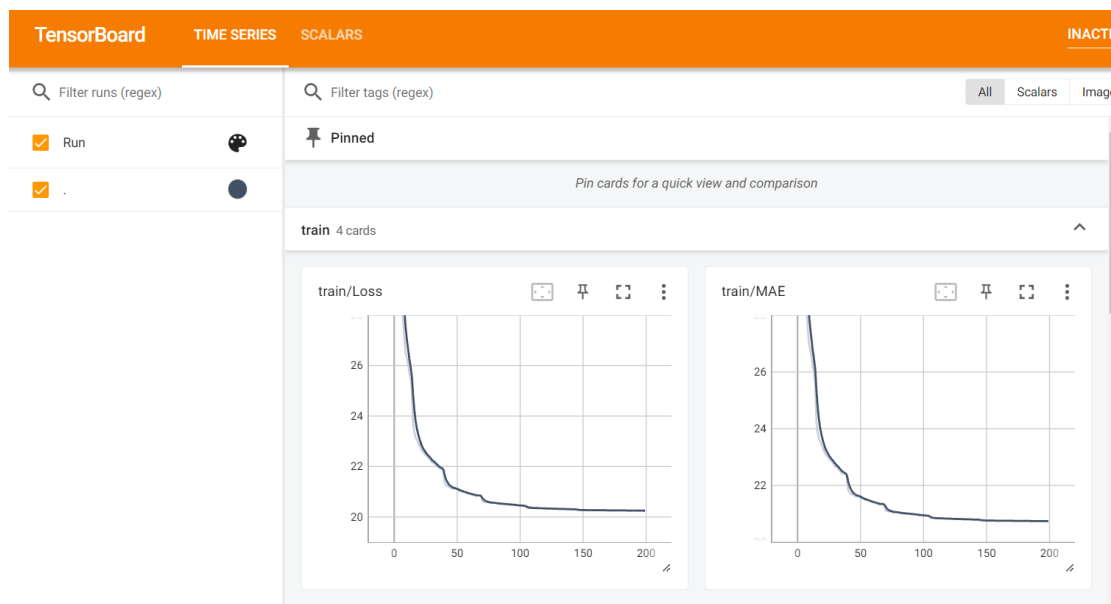


图 4.9: tensorboard 实时观察 loss 变化

最终的训练图像的 Loss 和验证集上各个指标的变化趋势如下图所示：

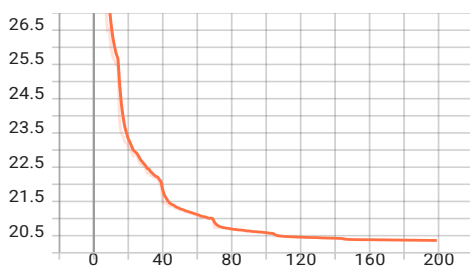


图 4.10: Loss

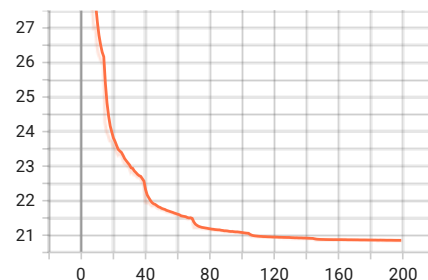


图 4.11: MAE

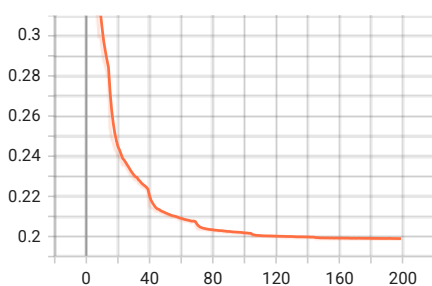


图 4.12: MAPE

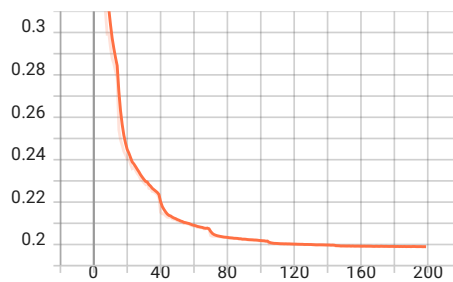


图 4.13: RMSE

从各个趋势图，我们可以观察，到 120 个 epoch 之后，各个指标和 Loss 都趋于稳定的状态。

最终在测试集的实验结果如下表 4.2 所示：



表 4.2: 测试集结果

|        | MAE   | MAPE   | RMSE  |
|--------|-------|--------|-------|
| PEMS03 | 20.61 | 22.45% | 31.22 |
| PEMS04 | 25.78 | 19.50% | 37.20 |
| PEMS07 | 27.17 | 13.38% | 43.85 |
| PEMS08 | 21.03 | 14.47% | 30.29 |

论文的相关的指标结果如下图 4.14 所示:

| Baseline methods |          | VAR    | SVR          | LSTM         | DCRNN               | STGCN        | ASTGCN(r)    | STG2Seq      | Graph WaveNet | STSGCN              |
|------------------|----------|--------|--------------|--------------|---------------------|--------------|--------------|--------------|---------------|---------------------|
| Datasets         | Metrics  |        |              |              |                     |              |              |              |               |                     |
| PEMS03           | MAE      | 23.65  | 21.97 ± 0.00 | 21.33 ± 0.24 | 18.18 ± 0.15        | 17.49 ± 0.46 | 17.69 ± 1.43 | 19.03 ± 0.51 | 19.85 ± 0.03  | <b>17.48 ± 0.15</b> |
|                  | MAPE (%) | 24.51  | 21.51 ± 0.46 | 23.33 ± 4.23 | 18.91 ± 0.82        | 17.15 ± 0.45 | 19.40 ± 2.24 | 21.55 ± 1.68 | 19.31 ± 0.49  | <b>16.78 ± 0.20</b> |
|                  | RMSE     | 38.26  | 35.29 ± 0.02 | 35.11 ± 0.50 | 30.31 ± 0.25        | 30.12 ± 0.70 | 29.66 ± 1.68 | 29.73 ± 0.52 | 32.94 ± 0.18  | <b>29.21 ± 0.56</b> |
| PEMS04           | MAE      | 23.75  | 28.70 ± 0.01 | 27.14 ± 0.20 | 24.70 ± 0.22        | 22.70 ± 0.64 | 22.93 ± 1.29 | 25.20 ± 0.45 | 25.45 ± 0.03  | <b>21.19 ± 0.10</b> |
|                  | MAPE (%) | 18.09  | 19.20 ± 0.01 | 18.20 ± 0.40 | 17.12 ± 0.37        | 14.59 ± 0.21 | 16.56 ± 1.36 | 18.77 ± 0.85 | 17.29 ± 0.24  | <b>13.90 ± 0.05</b> |
|                  | RMSE     | 36.66  | 44.56 ± 0.01 | 41.59 ± 0.21 | 38.12 ± 0.26        | 35.55 ± 0.75 | 35.22 ± 1.90 | 38.48 ± 0.50 | 39.70 ± 0.04  | <b>33.65 ± 0.20</b> |
| PEMS07           | MAE      | 75.63  | 32.49 ± 0.00 | 29.98 ± 0.42 | 25.30 ± 0.52        | 25.38 ± 0.49 | 28.05 ± 2.34 | 32.77 ± 3.21 | 26.85 ± 0.05  | <b>24.26 ± 0.14</b> |
|                  | MAPE (%) | 32.22  | 14.26 ± 0.03 | 13.20 ± 0.53 | 11.66 ± 0.33        | 11.08 ± 0.18 | 13.92 ± 1.65 | 20.16 ± 4.36 | 12.12 ± 0.41  | <b>10.21 ± 0.05</b> |
|                  | RMSE     | 115.24 | 50.22 ± 0.01 | 45.84 ± 0.57 | <b>38.58 ± 0.70</b> | 38.78 ± 0.58 | 42.57 ± 3.31 | 47.16 ± 3.66 | 42.78 ± 0.07  | 39.03 ± 0.27        |
| PEMS08           | MAE      | 23.46  | 23.25 ± 0.01 | 22.20 ± 0.18 | 17.86 ± 0.03        | 18.02 ± 0.14 | 18.61 ± 0.40 | 20.17 ± 0.49 | 19.13 ± 0.08  | <b>17.13 ± 0.09</b> |
|                  | MAPE (%) | 15.42  | 14.64 ± 0.11 | 14.20 ± 0.59 | 11.45 ± 0.03        | 11.40 ± 0.10 | 13.08 ± 1.00 | 17.32 ± 1.14 | 12.68 ± 0.57  | <b>10.96 ± 0.07</b> |
|                  | RMSE     | 36.33  | 36.16 ± 0.02 | 34.06 ± 0.32 | 27.83 ± 0.05        | 27.83 ± 0.20 | 28.16 ± 0.48 | 30.71 ± 0.61 | 31.05 ± 0.07  | <b>26.80 ± 0.18</b> |

图 4.14: 论文指标

我们得到的三个指标的值与原论文相比, 相差了 5 左右, 可能由于参数的设置的不同, 导致与原论文产生了一定的偏差, 但是也在可接受范围之内。

## 4.4 实验遇到的困难

在本次实验中, 我们小组也遇到了许多的问题, 相关的介绍如下所示:

1. 在训练的过程中, 会一直出现 loss 突然上升的情况, 具体如下图 4.15 所示:

```
Valid MAE: 26.079380221483184, Valid MAPE: 0.25052210788537815, Valid RMSE: 31.22
Epoch: 0, Loss: 22.977012634277344, MAE: 23.469194412231445, MAPE: 0.2451
Epoch: 200, Loss: 31.011817932128906, MAE: 31.50639533996582, MAPE: 0.2451
Epoch: 400, Loss: 43.69353103637695, MAE: 44.188716888427734, MAPE: 0.61
Epoch: 27, Train Loss: 50.94748028135591, Train MAE: 51.441702957308706
Valid MAE: 178.5111685031798, Valid MAPE: 0.9700621809901261, Valid RMSE: 178.51
Epoch: 0, Loss: 191.2989501953125, MAE: 191.7974395751953, MAPE: 0.9749
Epoch: 200, Loss: 156.40084838867188, MAE: 156.8983917236328, MAPE: 0.9749
Epoch: 400, Loss: 199.98397827148438, MAE: 200.48263549804688, MAPE: 0.9749
```

图 4.15: 梯度爆炸

分析原因, 可能是在 28 个 epochs 的时候, 发生了梯度爆炸的情况, 从而使 loss 突然飙升。针对于梯度爆炸的情况, 可能是优化器设置有误, 原先的优化器



只是用了一个 Adam 的优化器，经过多次实验后发现，在接近 30 个 epochs 的时候，它都会出现相应的梯度爆炸情况，主要是学习率过大导致的。因此对优化器进行进一步的改进，具体改进如下所示：

```
1. optimizer = optim.Adam(net.parameters(), lr=base_lr, eps=1.0e-8, weight_decay=0, amsgrad=False)
2. lr_scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer=optimizer, milestones=[15, 40, 70, 105, 145], gamma=0.3)
```

采用多步骤的学习率调整策略，使用了一个学习调度器。这使得在训练的特定时刻（milestones）会对学习率进行调整。在这个例子中，milestones 指定了在哪些 epoch（第 15, 40, 70, 105, 145 个 epoch）会有学习率的调整，而 gamma=0.3 则表示学习率的衰减因子。

通过这样的动态学习率改变，学习率能更好适应网络的训练，从而不会出现梯度爆炸的情况。

2. 在训练 PEMS04 数据集的过程中，又出现了梯度爆炸的问题。在第一步训练 PEMS03 的时候已经成功调整了学习率策略，但是由于一些超参数（gamma 或者 milestones）的原因，所以不能够适应于第二个数据集使用。所以我们小组继续上网搜索相关的资料，最终得知可以进行裁剪相关的梯度，具体的 Python 语句如下所示：

```
torch.nn.utils.clip_grad_norm_(net.parameters(), max_norm=5.0)
```

具体来说，这行代码的作用是对 STSGCN 的参数 parameters() 计算梯度，并对这些梯度进行修剪，限制其整体的范数不超过 5.0。

通过这段代码，成功控制住了训练过程中的梯度变化，避免出现梯度爆炸的情况发生。

通过上述的两个问题，我们小组从根源上成功解决了梯度爆炸的问题。在之前的神经网络训练中，也从来没有遇到过梯度爆炸的情况发生，而这次也给我们小组对类似问题积累了许多宝贵的经验。

## 五、总结和展望

在本次小组实验中，我们遇到了许多的问题，除了在第四部分中编程遇到的困难与挑战，我们在小组讨论中也遇到了相当大的问题。比如一开始选题的时候，每个人都有自己的想法，没有一个固定的选题。但之后，通过每个人的商讨与一次次的头脑风暴和讨论，我们选定了去复现这篇顶会的论文。一开始我们每人会呈现出对新知识的惧怕，比如小组成员们从未接触过图神经网络的相关知识。但是我们还是去选择继续坚持完成这篇论文的代码工作。在一次次探索论文方法中，我们终于明白了其中的算法原理。

而对于我们选择的交通预测，国内外也有相关的研究，具体如下所示：

**国内：**中国交通预测受益于广泛的数据采集和人工智能技术的快速发展。城市交通管理部门、互联网企业和新兴科技公司正在利用大数据分析和机器学习技术来预测交通流量，优化交通信号配时，以及提供智能交通管理解决方案。中国在智能交通基础设施建设也取得极大的进步，中国政府积极推进智能交通基础设施的建设，包括交通信号灯的智能化、车辆识别技术、交通监控摄像头等设施的部署。这些举措为交通预测提供了更丰富的数据来源，同时也为智能交通系统的建设奠定了基础。

**国外：**国外目前具有先进的交通数据采集技术，国外在交通数据采集方面表现出色，包括使用 GPS 数据、移动应用程序数据、传感器网络数据以及社交媒体数据等。这些数据的充分利用对于建立准确的交通预测模型至关重要。国外能够进行实时的交通预测，国外市场对于实时交通预测的需求日益增长。国外一些研究采用基于历史数据和实时数据的融合分析，从而帮助交通管理部门和个人用户更好地规划路线和避开拥堵。

**交通网络优化：**在一些发达国家，交通预测不仅仅关注交通流量，还着重于交通网络优化和运输规划，以及推动可持续的城市交通发展。

而针对于我们本次小组实验所做交通预测，它具有一定的现实意义，比如我们所做的交通流量预测能够进行控制道路上的红绿灯的变化；能够更好地进行道路系统规划；通过交通流量预测，可以为人们提供更好的导航应用。并且我们相信，我们目前所做的工作在未来一定能够落到实处。

而通过这次小组的课外实验，我们小组每个人感悟深刻，下面是我们小组的感想与收获。

通过本次课外大作业，我们拓展了个人的视野。我们这次选择的是一篇 2020 年的 AAAI 论文，虽然时间有点久远，但是这篇论文在“图神经运用于交通预测”的领域中具有极大的含金量。从小组讨论论文方法，到个人完成相应的代码工作，和训练模型，这中间花费了我们很多的时间和精力，但是我们的收获也是很多。我们通过最传统的邻接矩阵和相关的矩阵的变化来进行复现论文中的方法，而不是采用现在主流的图神经框架 Pyg 等，这让我们更加捕捉到了 GCN 的本质，也为我们后续的科研之路打下很好的基础！

## 六、参考文献

- [1] Li Y, Yu R, Shahabi C, et al. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting[J]. arXiv preprint arXiv:1707.01926, 2017.
- [2] Yu B, Yin H T, Zhu Z X. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting[J]. arXiv:1709.04875, 2017.
- [3] Guo S N, Lin Y F, Feng N, et al. Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting [C]// Proc of the 33rd AAAI Conference on Artificial Intelligence, Palo Alto, CA: AAAI Press, 2019, 33 (1): 922-929.
- [4] Song C, Lin Y, Guo S, et al. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting[C]//Proceedings of the AAAI conference on artificial intelligence. 2020, 34(01): 914-921.
- [5] Li Y, Yu R, Shahabi C, et al. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting[J]. arXiv preprint arXiv:1707.01926, 2017.
- [6] Wu Z, Pan S, Long G, et al. Graph wavenet for deep spatial-temporal graph modeling[J]. arXiv preprint arXiv:1906.00121, 2019.