# 《自然语言处理》
# -图神经网络实验


华为技术有限公司

# 目录

# 1 实验总览

## 1.1 实验背景

近年来，人们对深度学习方法在图上的扩展越来越感兴趣。在多方因素的成功推动下，研究人员借鉴了卷积网络、循环网络和深度自动编码器的思想，定义和设计了用于处理图数据的神经网络结构，由此一个新的研究热点——“图神经网络（Graph Neural Networks，GNN）”应运而生。

常见的图神经网络包括：图卷积网络（Graph Convolution Networks，GCN）、 图注意力网络（Graph Attention Networks）、图自编码器（ Graph Autoencoders）、图生成网络（Graph Generative Networks）和图时空网络（Graph Spatial-temporal Networks）。

## 1.2 实验目的

- 了解图神经网络相关知识；
- 了解如何使用 MindSpore 搭建并训练图神经网络。

## 1.3 实验清单

表格：实验、简述、难度、软件环境、硬件环境。

| 实验 | 简述 | 难度 | 软件环境 | 开发环境 |
| --- | --- | --- | --- | --- |
| GCN科学出版物分类 | 使用MindSpore实现图卷积神经网络GCN，用于科学出版物分类 | 中级 | Python3.7.5、MindSpore1.1 | ModelArts Ascend Notebook 环境 |

# 2 GCN 科学出版物分类

## 2.1 实验简介

图卷积网络（Graph Convolutional Network，GCN）是近年来逐渐流行的一种神经网络结构。不同于只能用于网格结构（grid-based）数据的传统网络模型 LSTM 和 CNN，图卷积网络能够处理具有广义拓扑图结构的数据，并深入发掘其特征和规律。

本实验主要介绍在下载的 Cora 和 Citeseer 数据集上使用 MindSpore 进行图卷积网络的训练。

## 2.2 实验环境

ModelArts Ascend Notebook 环境，MindSpore1.1.1

## 2.3 预备知识

### 2.3.1 图卷积神经网络概述

GCN 的本质目的就是用来提取拓扑图的空间特征。图卷积神经网络主要有两类，一类是基于空间域（spatial domain）或顶点域（vertex domain）的，另一类则是基于频域或谱域（spectral domain）的。GCN 属于频域图卷积神经网络。

空间域方法直接将卷积操作定义在每个结点的连接关系上，它跟传统的卷积神经网络中的卷积更相似一些。在这个类别中比较有代表性的方法有 Message Passing Neural Networks(MPNN), GraphSage, Diffusion Convolution Neural Networks(DCNN), PATCHY-SAN 等。

频域方法希望借助图谱的理论来实现拓扑图上的卷积操作。从整个研究的时间进程来看：首先研究 GSP（graph signal processing）的学者定义了 graph 上的傅里叶变化（Fourier Transformation），进而定义了 graph 上的卷积，最后与深度学习结合提出了 Graph Convolutional Network（GCN）。

### 2.3.2 GCN 网络过程

1、 定义 graph 上的 Fourier Transformation 傅里叶变换；

2、 定义 graph 上的 convolution 卷积。

## 2.3.3 图神经网络数据集

Cora 和 CiteSeer 是图神经网络常用的数据集，数据集官网 LINQS Datasets。

Cora 数据集包含 2708 个科学出版物，分为七个类别。 引用网络由 5429 个链接组成。 数据集中的每个出版物都用一个 0/1 值的词向量描述，0/1 指示词向量中是否出现字典中相应的词。 该词典包含 1433 个独特的单词。 数据集中的 README 文件提供了更多详细信息。

CiteSeer 数据集包含 3312 种科学出版物，分为六类。 引用网络由 4732 个链接组成。 数据集中的每个出版物都用一个 0/1 值的词向量描述，0/1 指示词向量中是否出现字典中相应的词。 该词典包含 3703 个独特的单词。 数据集中的 README 文件提供了更多详细信息。

本实验使用 Github 上 kimiyoung/planetoid 预处理和划分好的数据集。

将数据集放置在 data 目录下，该文件夹应包含以下文件：

data

├── ind.cora.allx

├── ind.cora.ally

├── ...

├── ind.cora.test.index

├── trans.citeseer.tx

├── trans.citeseer.ty

├── ...

└── trans.pubmed.y

模型的输入包含：

x，已标记的训练实例的特征向量，

y，已标记的训练实例的 one-hot 标签，

allx，标记的和未标记的训练实例（x 的超集）的特征向量，

graph，一个 dict，格式为{index: [index_of_neighbor_nodes]}.

令 n 为标记和未标记训练实例的数量。在 graph 中这 n 个实例的索引应从 0 到 n-1，其顺序与 allx 中的顺序相同。

除了 x，y，allx，和 graph 如上所述，预处理的数据集还包括：

1) tx，测试实例的特征向量，

2) ty，测试实例的 one-hot 标签，

3) test.index，graph 中测试实例的索引，

4） ally，是 allx 中实例的标签。

# 2.4 实验步骤

## 2.4.1 实验准备

步骤 1 OBS 创建项目文件夹

使用 OBS Brower+登录 OBS


**图2-1 OBS Brower+登录**

进入之前创建的用于挂载 notebook 的 obs 目录，此处我们是建了一个“nlp”的目录用于挂载昇腾环境下的 notebook：


**图2-2 进入课程主目录**

创建“gcn_mindspore_1.1”文件夹

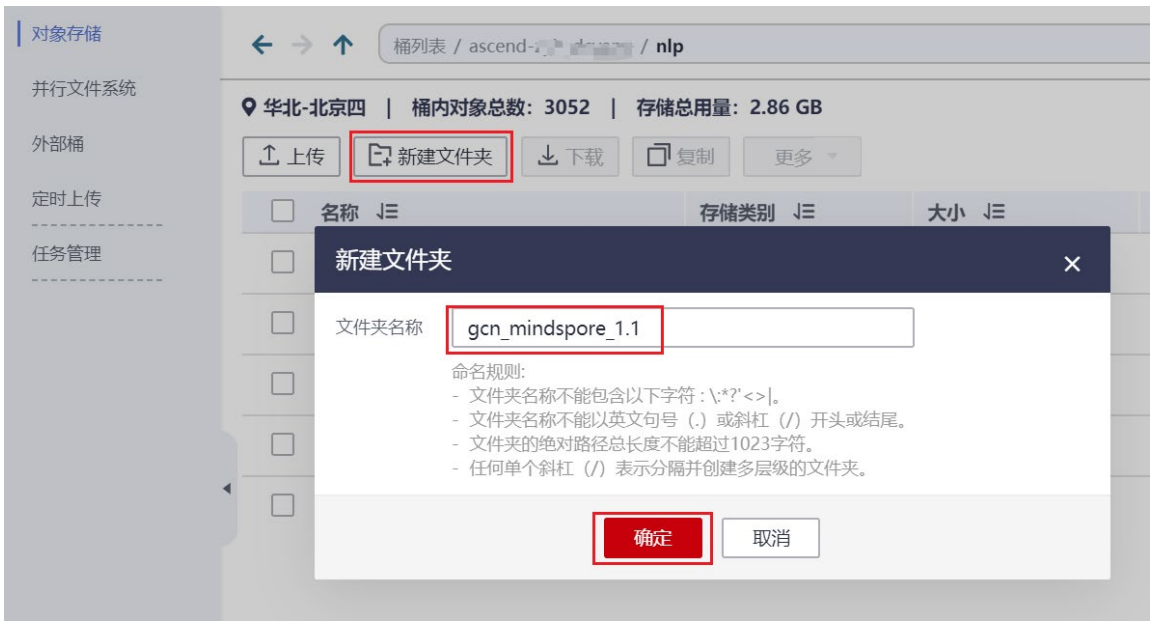


**图2-3 创建项目文件夹**

创建完如下所示：

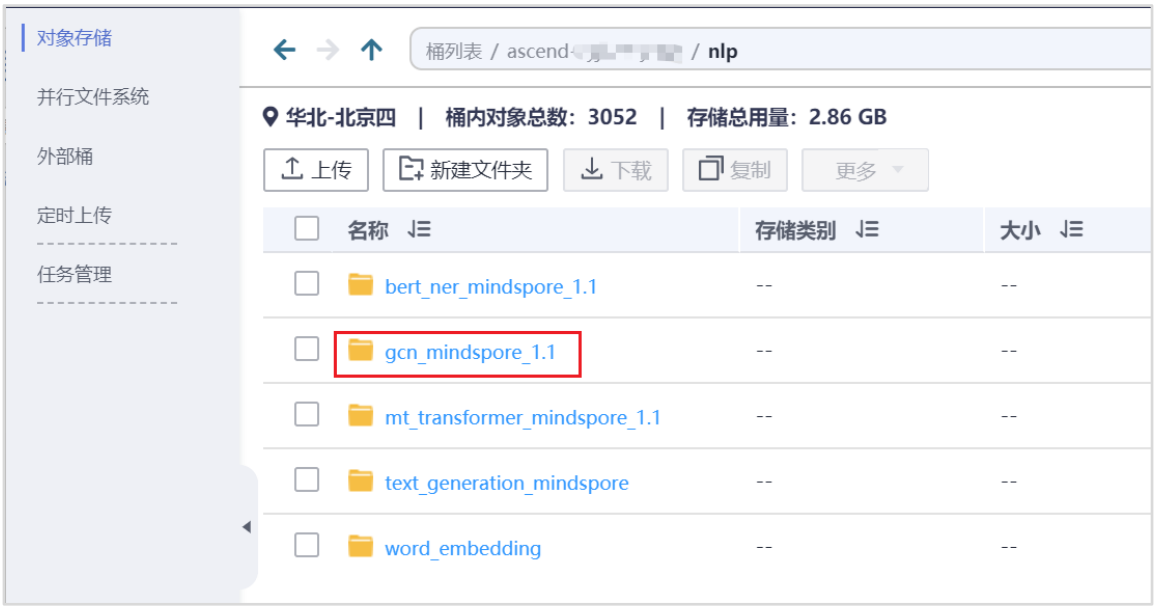


**图2-4 项目文件夹创建成功**

步骤 2　上传实验源码及数据

进入刚创建的“gcn_mindspore_1.1”文件夹，上传源码及数据至该目录下。

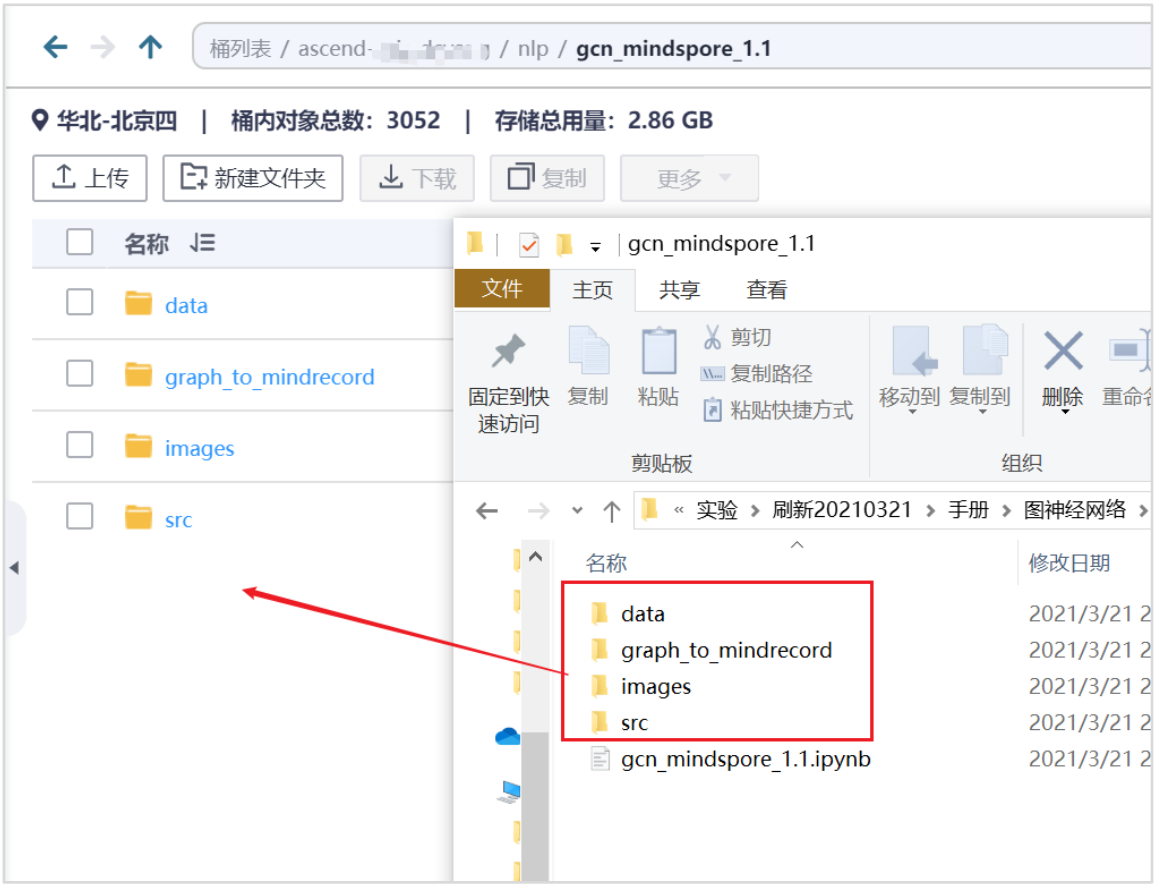

**图2-5 上传实验数据**

步骤 3　打开 JupyterLab

登录华为云，启动之前创建的 ModelArts Ascend notebook 环境


**图2-6 启动 notebook**

打开 JupyterLab，并进入之前创建的“gcn_mindspore_1.1”文件夹

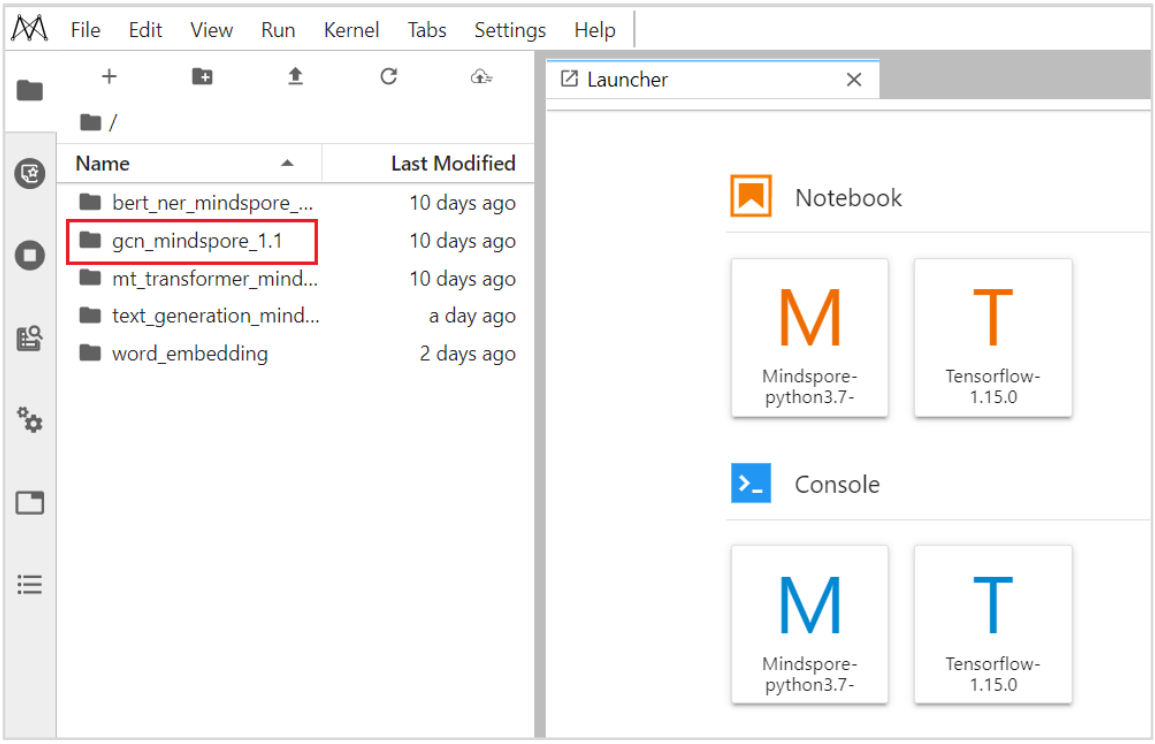


**图2-7 进入项目文件夹**

新建 notebook 文件：

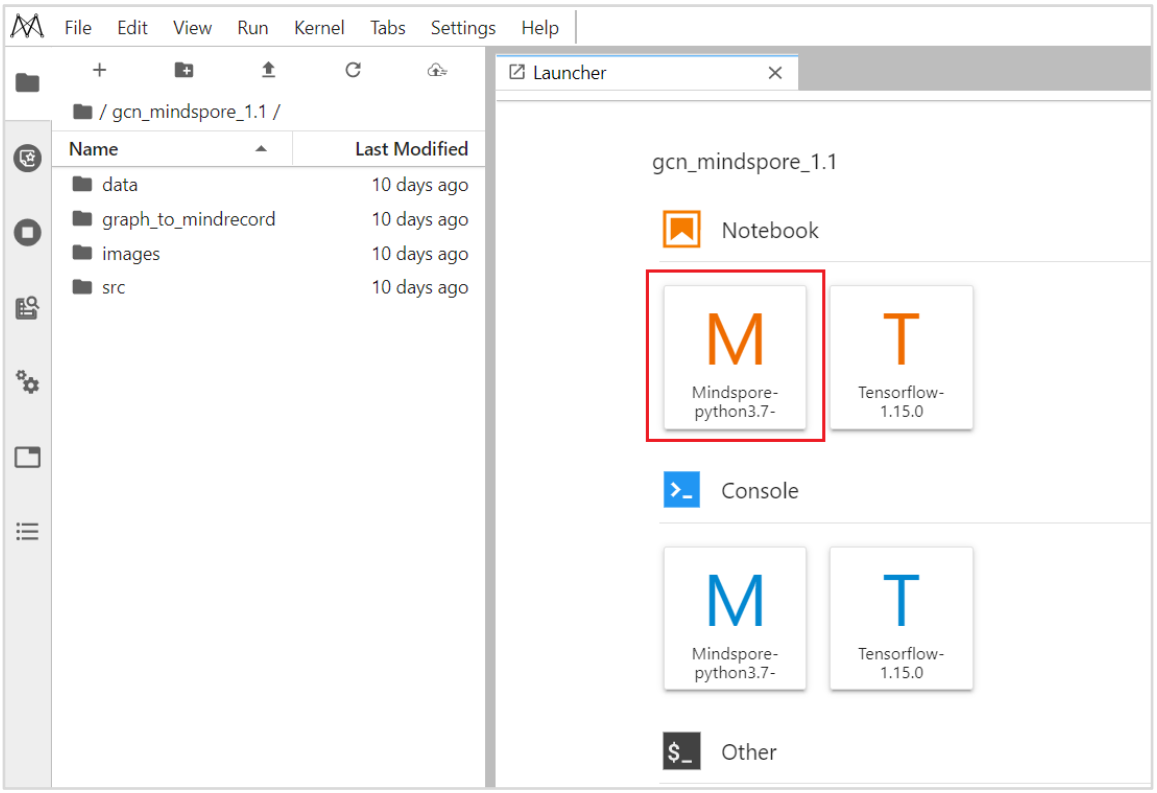


**图2-8 新建 notebook 文件**

重命名为“gcn_mindspore.ipynb”并打开，右侧显示代码编辑区

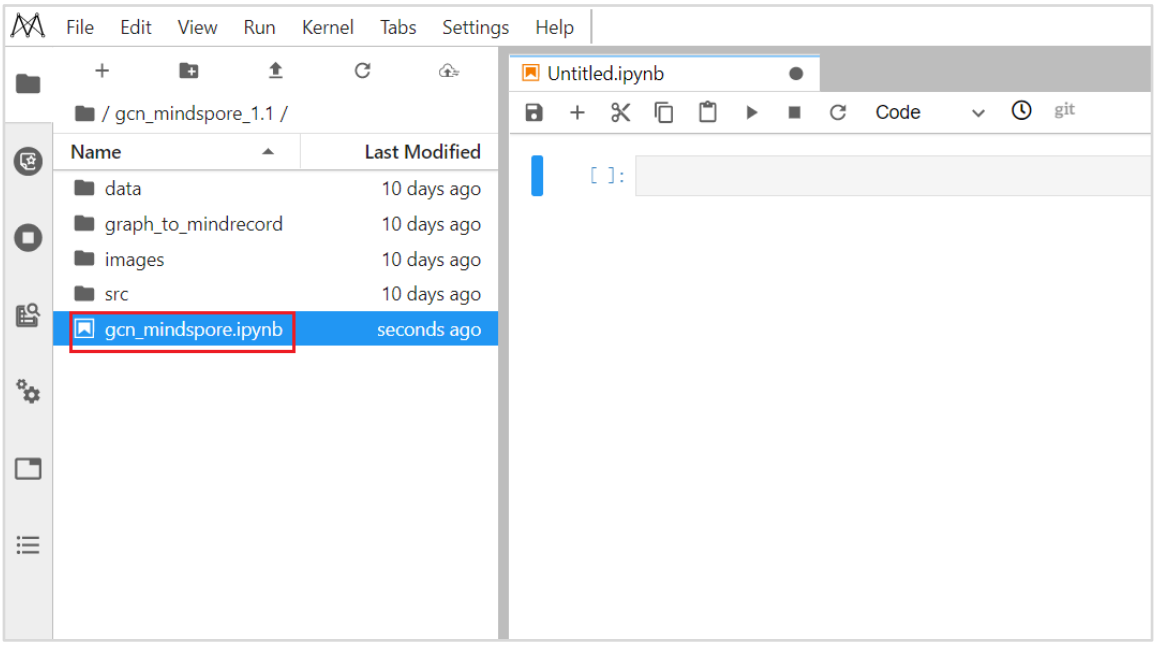

**图2-9 重命名并打开 notebook 文件**

## 2.4.2 实验过程

步骤 1　下载源码和数据至本地容器

因为 notebook 是挂载在 obs 上，运行的容器实例不能直接读取操作 obs 上的文件，需下载至容器本地环境中，请在 src_url 字段对应位置替换成自己创建的 obs 桶名字

输入：

```
import moxing as mox
mox.file.copy_parallel(src_url="s3://你的 obs 桶名称/nlp/gcn_mindspore_1.1/data/",
dst_url='./data/')  # 将 OBS 桶中数据拷贝到容器中
mox.file.copy_parallel(src_url="s3://你的 obs 桶名称/nlp/gcn_mindspore_1.1/src/", dst_url='./src/')
mox.file.copy_parallel(src_url="s3://你的 obs 桶名称
/nlp/gcn_mindspore_1.1/graph_to_mindrecord/", dst_url='./graph_to_mindrecord/')
```

步骤 2　导入依赖库

输入：

```
import os
# os.environ['DEVICE_ID']='7'

import time
import argparse
import numpy as np

from mindspore import nn
from mindspore import Tensor
from mindspore import context
from mindspore.ops import operations as P
from mindspore.nn.layer.activation import get_activation
from easydict import EasyDict as edict

from src.gcn import glorot, LossAccuracyWrapper, TrainNetWrapper
from src.dataset import get_adj_features_labels, get_mask
from graph_to_mindrecord.writer import run
```

步骤 3　设置运行环境

输入：

```
context.set_context(mode=context.GRAPH_MODE,device_target="Ascend", save_graphs=False)
```

步骤 4　定义参数配置

输入：

```
dataname = 'cora'
datadir_save = './data_mr'
datadir = os.path.join(datadir_save, dataname)
cfg = edict({
        'SRC_PATH': './data',
        'MINDRECORD_PATH': datadir_save,
        'DATASET_NAME': dataname,    # citeseer,cora
        'mindrecord_partitions':1,
        'mindrecord_header_size_by_bit' : 18,
        'mindrecord_page_size_by_bit' : 20,

        'data_dir': datadir,
        'seed' : 123,
        'train_nodes_num':140,
        'eval_nodes_num':500,
        'test_nodes_num':1000
})
```

步骤 5　转换数据格式为 mindrecord

输入：

```
#  转换数据格式
print("============== Graph To Mindrecord ==============")
run(cfg)
```

输出：

```
============== Graph To Mindrecord ==============
Init writer  ...
exec task 0, parallel: False ...
Node task is 0
transformed 512 record...
transformed 1024 record...
transformed 1536 record...
transformed 2048 record...
transformed 2560 record...
Processed 2708 lines for nodes.
transformed 2708 record...
exec task 0, parallel: False ...
Edge task is 0
transformed 512 record...
transformed 1024 record...
transformed 1536 record...
transformed 2048 record...
transformed 2560 record...
transformed 3072 record...
transformed 3584 record...
transformed 4096 record...
transformed 4608 record...
transformed 5120 record...
transformed 5632 record...
transformed 6144 record...
transformed 6656 record...
transformed 7168 record...
transformed 7680 record...
transformed 8192 record...
transformed 8704 record...
transformed 9216 record...
transformed 9728 record...
transformed 10240 record...
transformed 10752 record...
Processed 10858 lines for edges.
transformed 10858 record...
---------------------------------------------
END. Total time: 5.859450817108154
---------------------------------------------
```

步骤 6　定义 GCN 网络参数

输入：

```
class ConfigGCN():
    learning_rate = 0.01
    epochs = 200
    hidden1 = 16
    dropout = 0.5
    weight_decay = 5e-4
    early_stopping = 10
```

步骤 7　定义 GCN 网络结构

输入：

```
class GraphConvolution(nn.Cell):
    """
    GCN graph convolution layer.
```

```
    Args:
        feature_in_dim (int): The input feature dimension.
        feature_out_dim (int): The output feature dimension.
        dropout_ratio (float): Dropout ratio for the dropout layer. Default: None.
        activation (str): Activation function applied to the output of the layer, eg. 'relu'. Default:
None.

    Inputs:
        - **adj** (Tensor) - Tensor of shape :math:`(N, N)`.
        - **input_feature** (Tensor) - Tensor of shape :math:`(N, C)`.

    Outputs:
        Tensor, output tensor.
    """

    def __init__(self,
                 feature_in_dim,
                 feature_out_dim,
                 dropout_ratio=None,
                 activation=None):
        super(GraphConvolution, self).__init__()
        self.in_dim = feature_in_dim
        self.out_dim = feature_out_dim
        self.weight_init = glorot([self.out_dim, self.in_dim])
        self.fc = nn.Dense(self.in_dim,
                           self.out_dim,
                           weight_init=self.weight_init,
                           has_bias=False)
        self.dropout_ratio = dropout_ratio
        if self.dropout_ratio is not None:
            self.dropout = nn.Dropout(keep_prob=1-self.dropout_ratio)
        self.dropout_flag = self.dropout_ratio is not None
        self.activation = get_activation(activation)
        self.activation_flag = self.activation is not None
        self.matmul = P.MatMul()

    def construct(self, adj, input_feature):
        dropout = input_feature
        if self.dropout_flag:
            dropout = self.dropout(dropout)

        fc = self.fc(dropout)
        output_feature = self.matmul(adj, fc)

        if self.activation_flag:
```

```
                output_feature = self.activation(output_feature)
            return output_feature


class GCN(nn.Cell):
    """
    GCN architecture.

    Args:
        config (ConfigGCN): Configuration for GCN.
        adj (numpy.ndarray): Numbers of block in different layers.
        feature (numpy.ndarray): Input channel in each layer.
        output_dim (int): The number of output channels, equal to classes num.
    """

    def __init__(self, config, adj, feature, output_dim):
        super(GCN, self).__init__()
        self.adj = Tensor(adj)
        self.feature = Tensor(feature)
        input_dim = feature.shape[1]
        self.layer0 = GraphConvolution(input_dim, config.hidden1, activation="relu",
dropout_ratio=config.dropout)
        self.layer1 = GraphConvolution(config.hidden1, output_dim, dropout_ratio=None)

    def construct(self):
        output0 = self.layer0(self.adj, self.feature)
        output1 = self.layer1(self.adj, output0)
        return output1
```

步骤 8　定义训练、评估函数

输入：

```
def train_eval(args_opt):
    """Train model."""
    np.random.seed(args_opt.seed)
    config = ConfigGCN()
    adj, feature, label = get_adj_features_labels(args_opt.data_dir)

    nodes_num = label.shape[0]
    train_mask = get_mask(nodes_num, 0, args_opt.train_nodes_num)
    eval_mask = get_mask(nodes_num, args_opt.train_nodes_num, args_opt.train_nodes_num +
args_opt.eval_nodes_num)
    test_mask = get_mask(nodes_num, nodes_num - args_opt.test_nodes_num, nodes_num)

    class_num = label.shape[1]
    gcn_net = GCN(config, adj, feature, class_num)
```

```
    gcn_net.add_flags_recursive(fp16=True)

    eval_net = LossAccuracyWrapper(gcn_net, label, eval_mask, config.weight_decay)
    test_net = LossAccuracyWrapper(gcn_net, label, test_mask, config.weight_decay)
    train_net = TrainNetWrapper(gcn_net, label, train_mask, config)

    loss_list = []
    for epoch in range(config.epochs):
        t = time.time()

        train_net.set_train()
        train_result = train_net()
        train_loss = train_result[0].asnumpy()
        train_accuracy = train_result[1].asnumpy()

        eval_net.set_train(False)
        eval_result = eval_net()
        eval_loss = eval_result[0].asnumpy()
        eval_accuracy = eval_result[1].asnumpy()

        loss_list.append(eval_loss)
        if epoch%10==0:
            print("Epoch:", '%04d' % (epoch), "train_loss=", "{:.5f}".format(train_loss),
                  "train_acc=", "{:.5f}".format(train_accuracy), "val_loss=",
"{:.5f}".format(eval_loss),
                  "val_acc=", "{:.5f}".format(eval_accuracy), "time=", "{:.5f}".format(time.time() -
t))

        if epoch > config.early_stopping and loss_list[-1] > np.mean(loss_list[-
(config.early_stopping+1):-1]):
            print("Early stopping...")
            break

    t_test = time.time()
    test_net.set_train(False)
    test_result = test_net()
    test_loss = test_result[0].asnumpy()
    test_accuracy = test_result[1].asnumpy()
    print("Test set results:", "loss=", "{:.5f}".format(test_loss),
          "accuracy=", "{:.5f}".format(test_accuracy), "time=", "{:.5f}".format(time.time() - t_test))
```

步骤 9　启动训练、评估

输入：

```
#训练
```

```
print("============== Starting Training ==============")
train_eval(cfg)
```

输出：

```
============== Starting Training ==============
Epoch: 0000 train_loss= 1.95346 train_acc= 0.17857 val_loss= 1.94900 val_acc= 0.35000 time= 34.96806
Epoch: 0010 train_loss= 1.86234 train_acc= 0.85000 val_loss= 1.90394 val_acc= 0.52600 time= 0.00274
Epoch: 0020 train_loss= 1.76015 train_acc= 0.85714 val_loss= 1.86055 val_acc= 0.54600 time= 0.00281
Epoch: 0030 train_loss= 1.60570 train_acc= 0.86429 val_loss= 1.80513 val_acc= 0.57000 time= 0.00276
Epoch: 0040 train_loss= 1.45119 train_acc= 0.90714 val_loss= 1.74078 val_acc= 0.61400 time= 0.00276
Epoch: 0050 train_loss= 1.31144 train_acc= 0.94286 val_loss= 1.66661 val_acc= 0.70000 time= 0.00274
Epoch: 0060 train_loss= 1.17945 train_acc= 0.96429 val_loss= 1.59235 val_acc= 0.73800 time= 0.00276
Epoch: 0070 train_loss= 1.05087 train_acc= 0.95714 val_loss= 1.51406 val_acc= 0.77200 time= 0.00281
Epoch: 0080 train_loss= 0.94777 train_acc= 0.98571 val_loss= 1.44490 val_acc= 0.78200 time= 0.00275
Epoch: 0090 train_loss= 0.86569 train_acc= 0.95714 val_loss= 1.38161 val_acc= 0.77600 time= 0.00277
Epoch: 0100 train_loss= 0.81243 train_acc= 0.97143 val_loss= 1.32921 val_acc= 0.78200 time= 0.00276
Epoch: 0110 train_loss= 0.72469 train_acc= 0.97857 val_loss= 1.27977 val_acc= 0.78600 time= 0.00276
Epoch: 0120 train_loss= 0.71844 train_acc= 0.97857 val_loss= 1.23830 val_acc= 0.78600 time= 0.00277
Epoch: 0130 train_loss= 0.64569 train_acc= 0.98571 val_loss= 1.21176 val_acc= 0.78600 time= 0.00278
Epoch: 0140 train_loss= 0.62185 train_acc= 0.98571 val_loss= 1.17444 val_acc= 0.78000 time= 0.00283
Epoch: 0150 train_loss= 0.61790 train_acc= 0.98571 val_loss= 1.14649 val_acc= 0.78000 time= 0.00282
Epoch: 0160 train_loss= 0.58705 train_acc= 0.98571 val_loss= 1.12544 val_acc= 0.78400 time= 0.00279
Epoch: 0170 train_loss= 0.56079 train_acc= 0.99286 val_loss= 1.11025 val_acc= 0.78200 time= 0.00277
Epoch: 0180 train_loss= 0.52371 train_acc= 0.99286 val_loss= 1.09228 val_acc= 0.78400 time= 0.00278
Epoch: 0190 train_loss= 0.50555 train_acc= 0.99286 val_loss= 1.07423 val_acc= 0.78800 time= 0.00283
Test set results: loss= 1.00918 accuracy= 0.81800 time= 4.81428
```

## 2.5 实验小结

本节实验介绍了如何在 ModelArts 平台上使用 MindSpore 搭建图卷积网络用于科学出版物的分类，通过实验使学员了解图卷积网络的基本原理以及 MindSpore 搭建神经网络模型的基本方法。