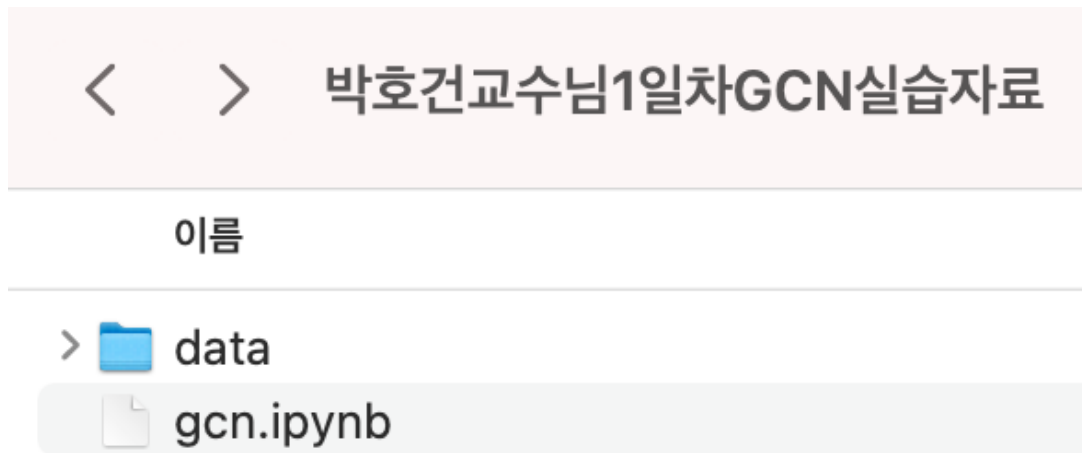


Graph Convolution networks

코드 리뷰 및 실습

성균관대학교 인공지능융합원
딥러닝 특강(심화) - 박호건 교수님
실습조교 - 정지원
수업 1일차 실습자료

1. 준비물



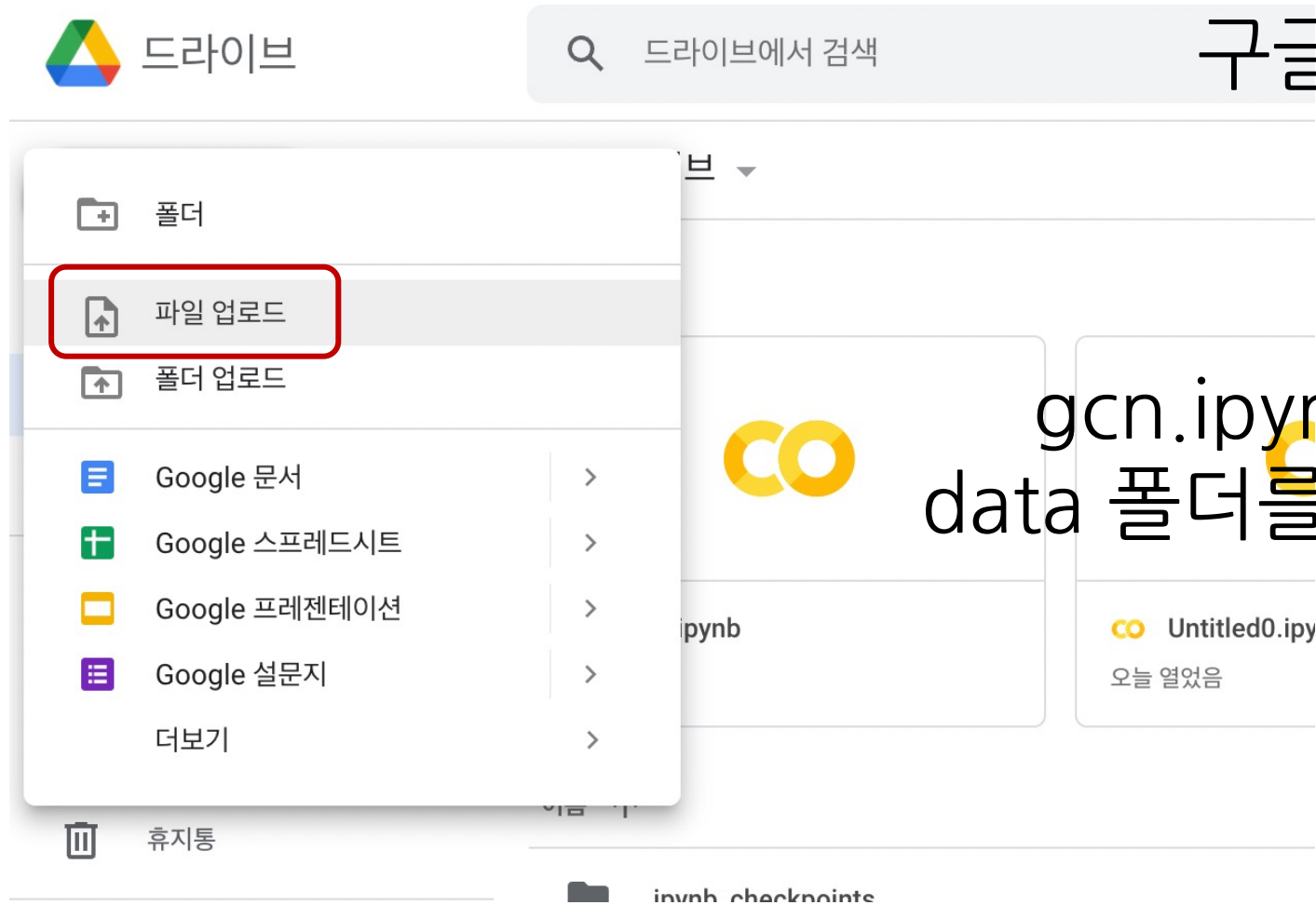
1. Colab(코랩) 실습 파일
gcn.ipynb

2. 데이터가 들어있는 폴더
data

2. 셋팅

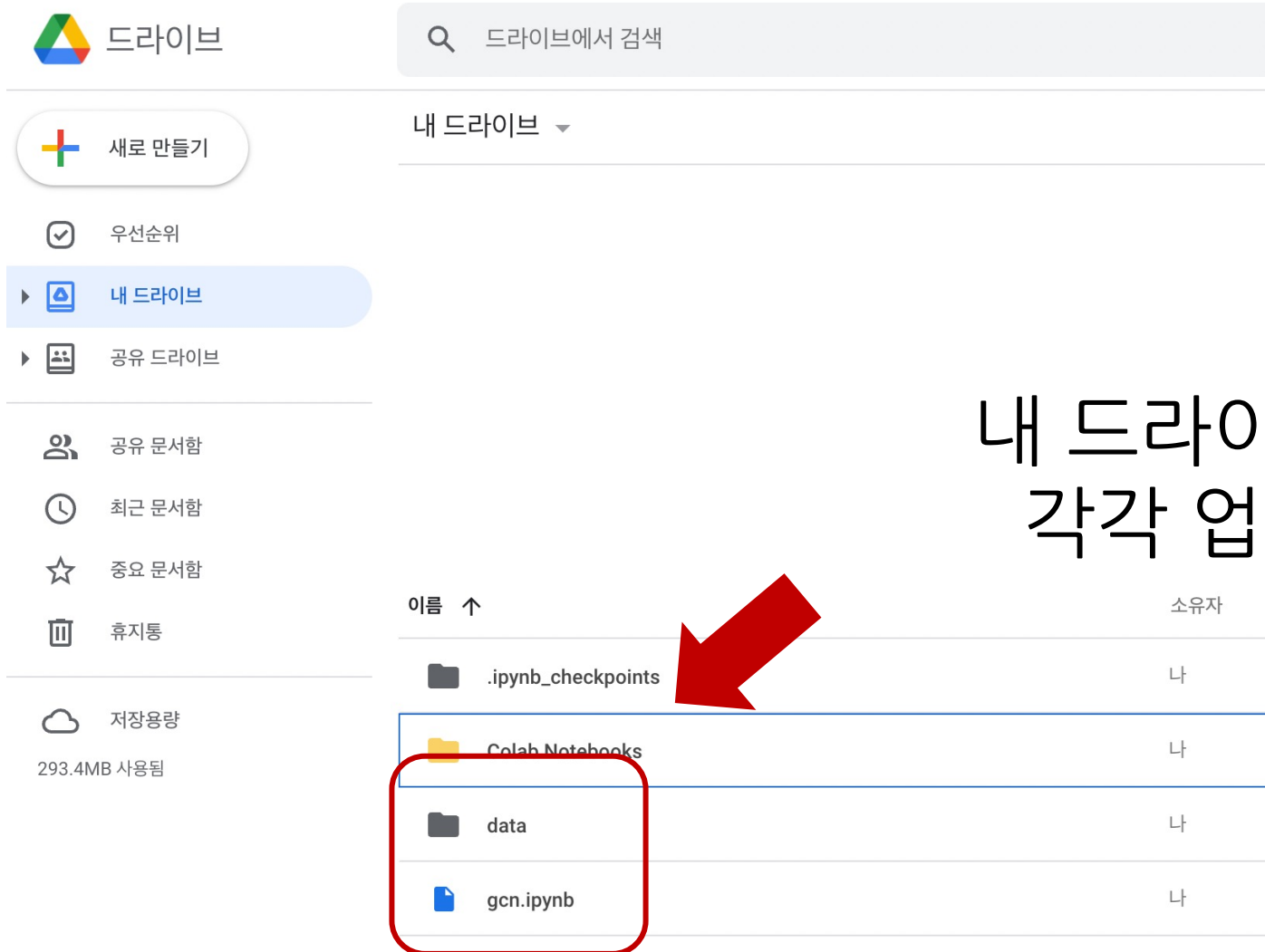
<https://drive.google.com/drive/my-drive>

구글 드라이브 접속



gcnn.ipynb 파일과
data 폴더를 각각 업로드

2. 셋팅



The image shows the Google Drive web interface. On the left, the sidebar contains navigation options: '새로 만들기' (Create), '우선순위' (Priority), '내 드라이브' (My Drive), '공유 드라이브' (Shared Drives), '공유 문서함' (Shared with me), '최근 문서함' (Recent), '중요 문서함' (Important), '휴지통' (Trash), and '저장용량' (Storage). The main area shows the '내 드라이브' (My Drive) view with a search bar and a dropdown menu. Below the search bar, there is a table of files and folders. A red arrow points to the '.ipynb_checkpoints' folder, and a red box highlights the 'data' folder and 'gcn.ipynb' file.

드라이브

드라이브에서 검색

내 드라이브 ▾

새로 만들기

우선순위

내 드라이브

공유 드라이브

공유 문서함

최근 문서함

중요 문서함

휴지통

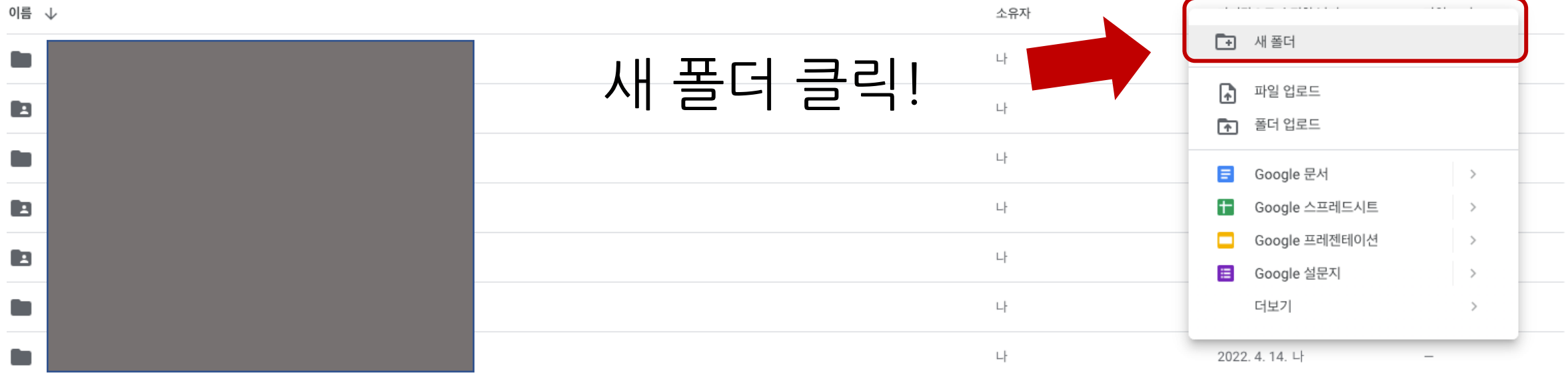
저장용량
293.4MB 사용됨

내 드라이브 내에
각각 업로드!!

이름 ↑	소유자
.ipynb_checkpoints	나
Colab Notebooks	나
data	나
gcn.ipynb	나

2. 셋팅

내 드라이브에서 마우스 우클릭 하기



GCN_exercise로 폴더명 지정하기!

2. 셋팅

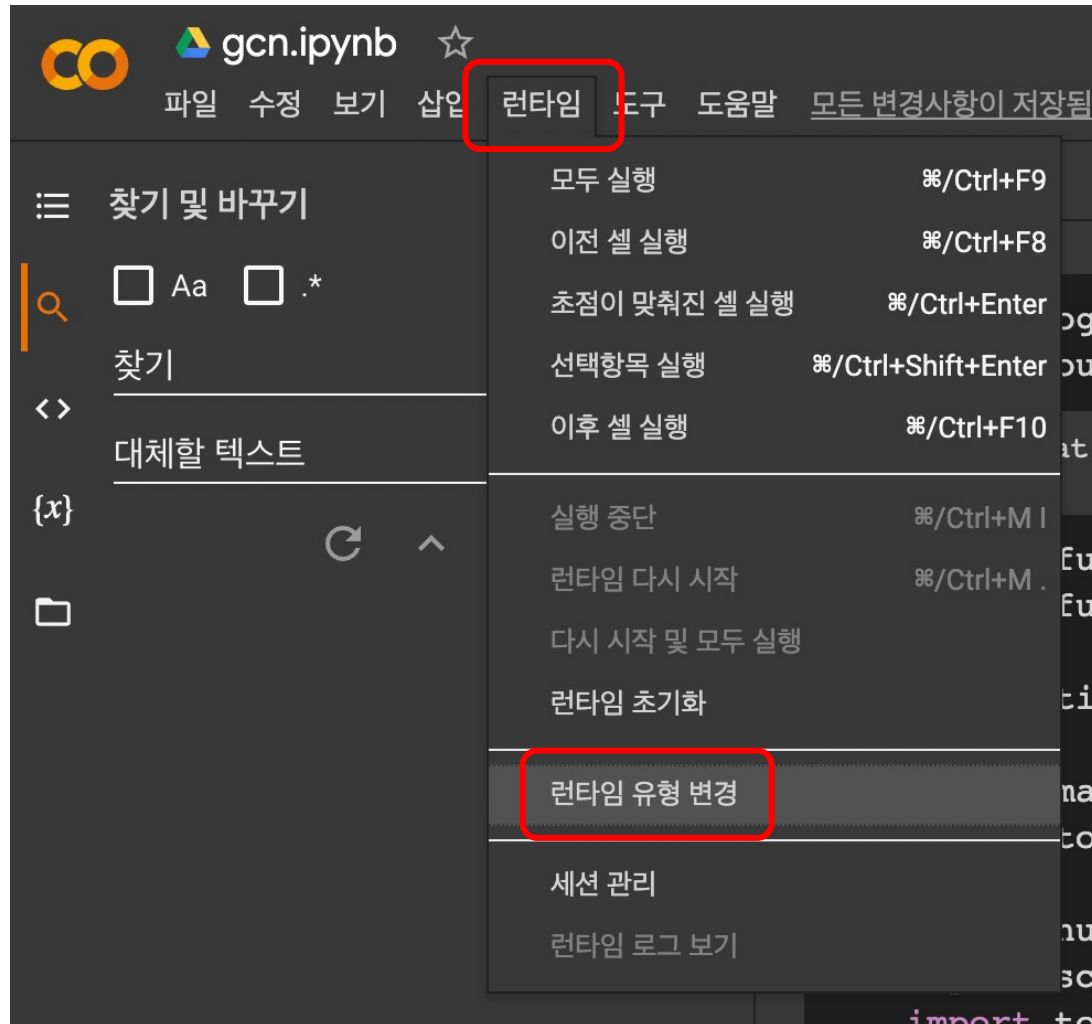
GCN_exercise 폴더에 내 드라이브에 있는
gcn.ipynb, data
각각 업로드 되면 완료!!

내 드라이브 > GCN_exercise

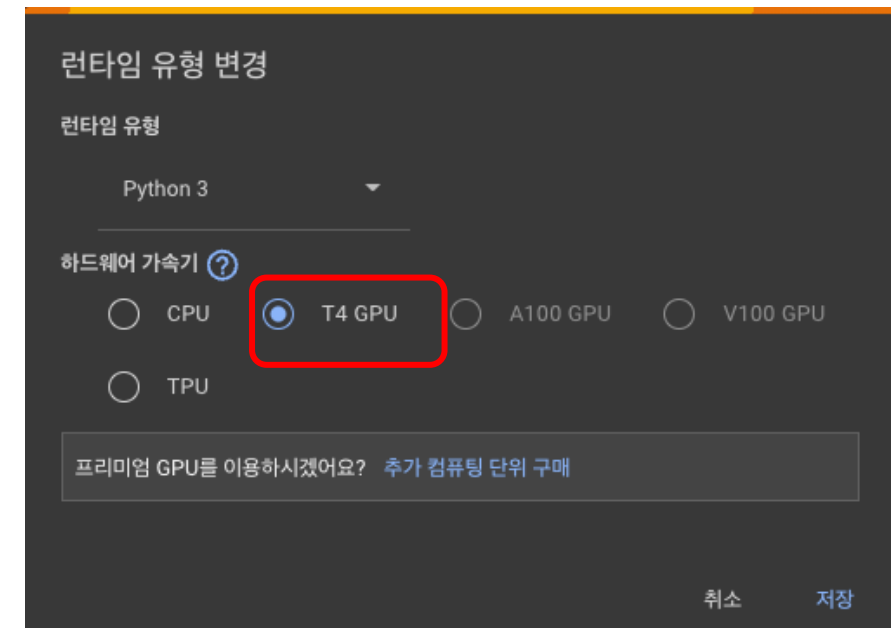
이름 ↓	소유자	마지막으로 수정한 날짜	파일 크기
data	나	오후 3:02 나	—
.ipynb_checkpoints	나	오후 3:51 나	—
gcn.ipynb	나	오후 6:54 나	33KB

gcn.ipynb 더블 클릭!!

2. 세팅

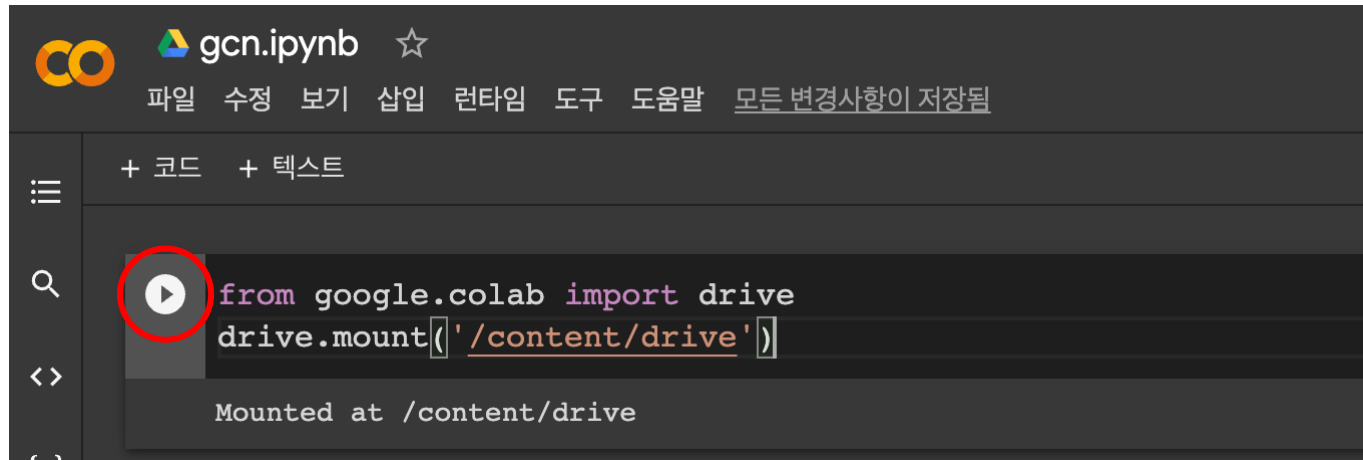


런타임 -> 런타임 유형변경 클릭!
하드웨어 가속기 GPU로
변경, T4라는 GPU 설정하기!!



2. 세팅

두번째 셀(shell) 실행!

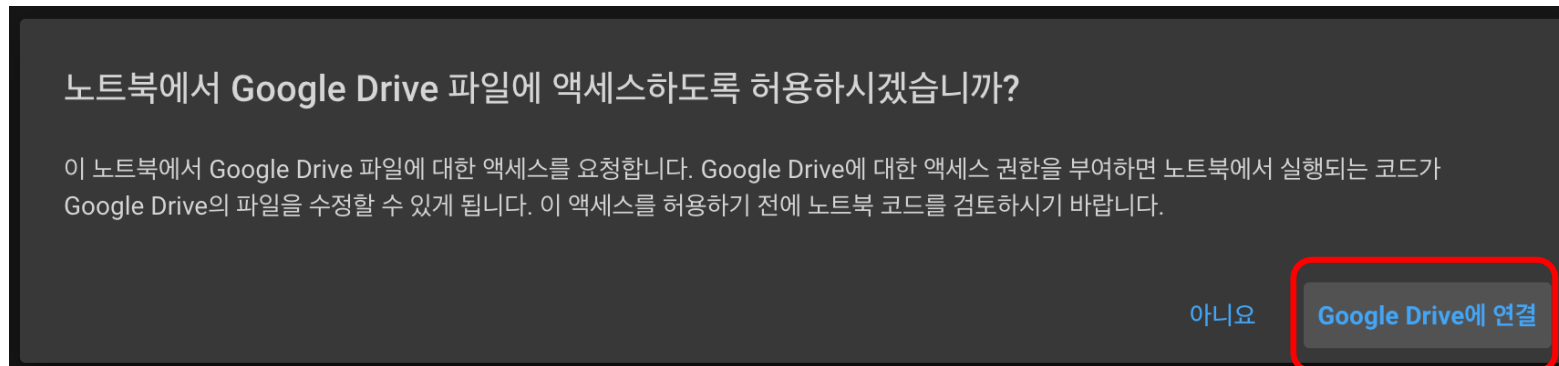


The screenshot shows a Google Colab notebook interface. At the top, the logo 'CO' and the text 'gcn.ipynb' are visible. Below the header, there are tabs for '파일', '수정', '보기', '삽입', '런타임', '도구', '도움말', and a link '모든 변경사항이 저장됨'. The main area shows a code cell with the following code:

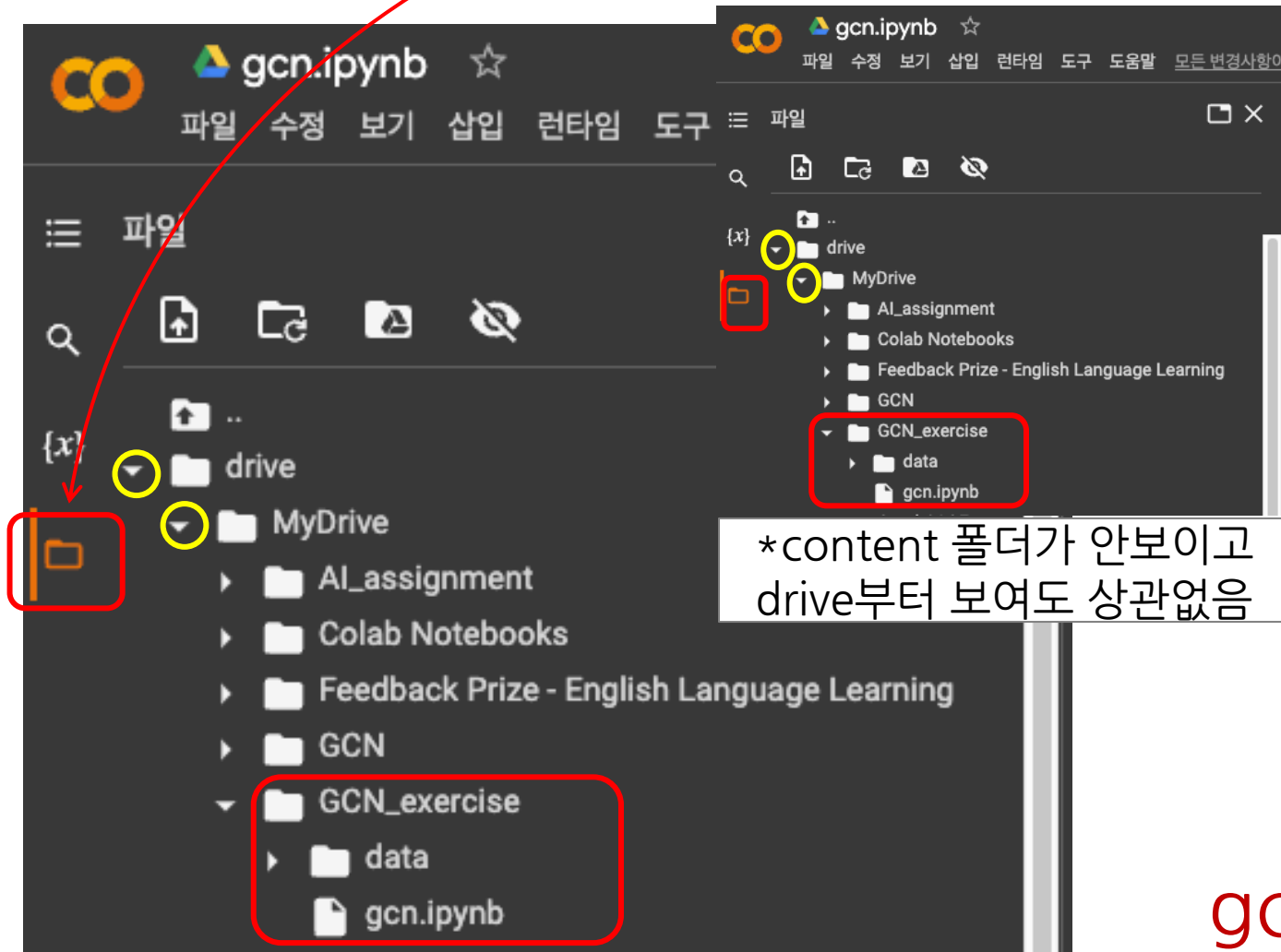
```
from google.colab import drive
drive.mount('/content/drive')
```

 A red circle highlights the play button icon on the left of the code cell. Below the code, the output 'Mounted at /content/drive' is displayed.

Google Drive에 연결 ->
로그인 -> 계속



2. 세팅



왼쪽 하단의 파일 창 클릭
파일 옆 세모 클릭

content

▸ drive

▸ MyDrive

...

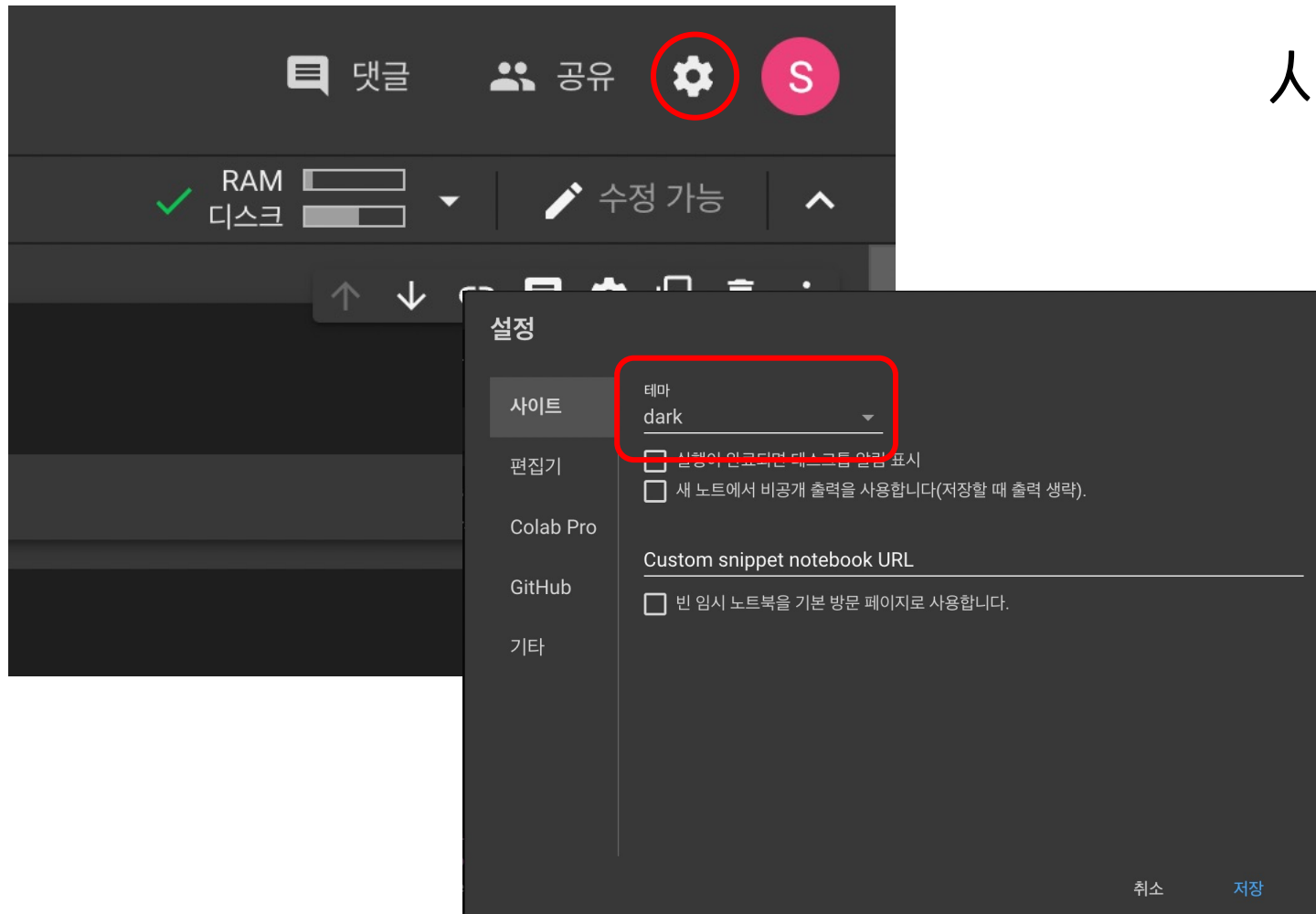
▸ GCN_exercise

▸ data

▸ gcn.ipynb

아까 업로드한 data와
gcn.ipynb 파일이 보인다면
연결 성공!

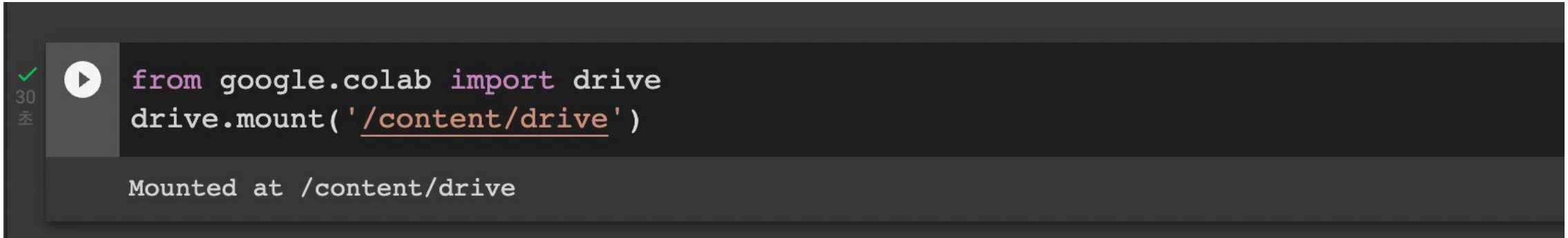
* 별첨 - 다크모드 사용법



설정
사이트 - *테마 dark* 선택
저장

하얀 화면을 오래보면
눈이 아프기때문에
또는 개인의 취향으로

2. 각 셀별 코드 구성소개



A screenshot of a Google Colab code cell. On the left, there is a green checkmark and the text '30 초' (30 seconds). The code cell contains two lines of Python code: `from google.colab import drive` and `drive.mount('/content/drive')`. Below the code, the output shows 'Mounted at /content/drive'.

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

두번째 셀 : 데이터 업로드를 위해 구글드라이브와 연동하는 코드

2. 각 셀별 코드 구성소개

```
[ ] from __future__ import division
    from __future__ import print_function

    import time

    import math
    import torch

    import numpy as np
    import scipy.sparse as sp
    import torch.optim as optim

    import torch.nn as nn
    import torch.nn.functional as F
    from torch.nn.parameter import Parameter
    from torch.nn.modules.module import Module
```

세번째 셀 : 여러 기능들이 들어있는 라이브러리 설치

2. 각 셀별 코드 구성소개

```
#layers
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output

    def __repr__(self):
        return self.__class__.__name__ + ' (' \
            + str(self.in_features) + ' -> ' \
            + str(self.out_features) + ')'
```

네번째 셀 :

#layers

Graph neural network의
각 층을 구현하는 코드

2. 각 셀별 코드 구성소개

```
[ ] #model

class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc2(x, adj)
        return F.log_softmax(x, dim=1)
```

다섯번째 셀 :

#model

layers에서 만든 층을 불러와
모델을 조립하는 코드

2. 각 셀별 코드 구성소개

```
#utils
def encode_onehot(labels):
    classes = set(labels)
    classes_dict = {c: np.identity(len(classes))[i, :] for i, c in
                    enumerate(classes)}
    labels_onehot = np.array(list(map(classes_dict.get, labels)),
                             dtype=np.int32)

    return labels_onehot

def load_data(path="/content/drive/MyDrive/GCN_exercise/data/cora/", dataset="cora"):
    print('Loading {} dataset...'.format(dataset))

    idx_features_labels = np.genfromtxt("{}{}.content".format(path, dataset),
                                         dtype=np.dtype(str))
    # print('idx_features_labels', idx_features_labels.shape) # (2708, 1435)
    # print('first', idx_features_labels[:, 0]) # id
    # print('last', idx_features_labels[:, -1]) # label, ['Neural_Networks']
    features = sp.csr_matrix(idx_features_labels[:, 1:-1], dtype=np.float32)
    labels = encode_onehot(idx_features_labels[:, -1])
    # print('labels', labels.shape) # (2708, 7)
    idx = np.array(idx_features_labels[:, 0], dtype=np.int32)

    idx_map = {j: i for i, j in enumerate(idx)}
    edges_unordered = np.genfromtxt("{}{}.cites".format(path, dataset),
                                     dtype=np.int32)

    edges = np.array(list(map(idx_map.get, edges_unordered.flatten())),
                     dtype=np.int32).reshape(edges_unordered.shape)
    # |edges| = (edges개수=5429, 2)

    adj = sp.coo_matrix((np.ones(edges.shape[0]), (edges[:, 0], edges[:, 1])),
                        shape=(labels.shape[0], labels.shape[0]),
                        dtype=np.float32) # |labels| =(2708,8), 2708 = # data

    # symmetric adjacency matrix
    adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)
    # print('adj', adj.shape) # (2708, 2708)

    features = normalize(features)
    adj = normalize(adj + sp.eye(adj.shape[0])) # Add self loop to adjacency matrix

    idx_train = range(140)
    idx_val = range(200, 500)
    idx_test = range(500, 1500)
```

여섯번째 셀 :

#utils

데이터를 불러와 가공하고
정확도 측정하는 등
학습과 테스트에 필요한
여러가지 기능들을 구현

2. 각 셀별 코드 구성소개

```
[ ] #Training
import easydict

args = easydict.EasyDict({"no-cuda":False, "fastmode":False, "seed":42, \
    "epochs":200, "lr":0.01, "weight_decay":5e-4, \
    "hidden":16, "dropout":0.5, "cuda":True})

np.random.seed(args.seed)
torch.manual_seed(args.seed)
if args.cuda:
    torch.cuda.manual_seed(args.seed)

# Load data
adj, features, labels, idx_train, idx_val, idx_test = load_data()

# Model and optimizer
model = GCN(nfeat=features.shape[1],
            nhid=args.hidden,
            nclass=labels.max().item() + 1,
            dropout=args.dropout)
optimizer = optim.Adam(model.parameters(),
                        lr=args.lr, weight_decay=args.weight_decay)

if args.cuda:
```

마지막 셀 :

#Training

모델을 초기화하고 학습 데이터를
넣어 파라미터들을 학습시키는
코드

#test

학습이 완료된 모델에 테스트
데이터를 넣어 정확도를 측정함

3. 코드 분석 - import

```
from __future__ import division
from __future__ import print_function

import time #시간측정 기능

import math #수학식 기능 (ex log, 삼각함수, 루트, ...)
import torch

import numpy as np
import scipy.sparse as sp
import torch.optim as optim

import torch.nn as nn
import torch.nn.functional as F
from torch.nn.parameter import Parameter
from torch.nn.modules.module import Module
```

__future__

파이썬 2와 3의 문법이 호환 가능하도록 도와줌

예시) 파이썬 2 - print "hello gcn"

파이썬 3 - print("hello gcn")

파이썬 2 - $5 / 2 = 2$

파이썬 3 - $5 / 2 = 2.5$

torch : 딥러닝을 도와주는 기능

numpy : 행렬연산을 도와주는 기능

scipy : 과학계산을 도와주는 기능 (선형대수, 확률분포 등)

torch.optim : 옵티마이저 패키지

torch.nn : 함수들이 담긴 클래스

torch.nn.functional : 활성화 함수 등이 담긴 패키지

torch.nn.parameter : 파라미터 관련 클래스

torch.nn.modules.module : 순전파, 역전파 자동화 패키지

3. 코드 분석 - layers 1

Table 1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

```
#layers
class GraphConvolution(Module):    Module을 상속하는 클래스를 만듦
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """
    클래스가 기본적으로 가지고 있을 변수들 선언해주는 초기화 메서드

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        # This is not a linear layer, but literally has only parameter values.
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters() # weight parameters initialization

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv) # uniform distribution U(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
```

super : 부모 클래스의 `__init__()` 메소드를 자식 클래스의 `__init__()` 메소드에서 실행한다.

*Super 하지 않을 경우, 상속받은 부모 클래스의 init 위로 새로 선언한 init이 덮어쓰여진다. (부모 init 사라짐)

in_feature : 층에 들어오는 입력값의 feature dimension

out_features : 층에서 나가는 출력값의 feature dimension

weight : 파라미터

bias : 절편(편향)

reset_parameters : 파라미터 초기화 함수

3. 코드 분석 - layers2

파라미터 초기화를 해주는 이유!
초기 파라미터가 0이나 1이면 학습이 치우치는 현상이 발생

```
def reset_parameters(self):
    stdv = 1. / math.sqrt(self.weight.size(1))
    self.weight.data.uniform_(-stdv, stdv) # uniform distribution U(-stdv, stdv)
    if self.bias is not None:
        self.bias.data.uniform_(-stdv, stdv)
```

모델 파라미터를 초기화해주는 메소드

$\frac{1}{\sqrt{\text{weight.size}(1)}}$ weight 크기 = in_feature * out_feature
의 첫번째 값 = out_feature = 1,433
tensor.data.uniform_ : 범위 사이에 균등하게 값이 분포

```
def forward(self, input, adj):
    # matrix multiplication
    support = torch.mm(input, self.weight)
    # matrix multiplication of the sparse matrix adj and the support matrix.
    output = torch.spmv(adj, support)
    if self.bias is not None:
        return output + self.bias
    else:
        return output
```

모델 통과 메소드

torch.mm 벡터곱
torch.spmv 벡터곱 *요소에 0이 있을 경우 더 빨리 계산해주는 함수
바이어스가 있으면? 더해준다

```
def __repr__(self):
    return self.__class__.__name__ + ' (' \
        + str(self.in_features) + ' -> ' \
        + str(self.out_features) + ')
```

객체를 사용자가 이해할 수 있는 문자열로 반환하는 메소드

이 클래스의 객체 gc1에 대해 (다음페이지에 나옵니다)
gc1.__repr__ 실행 시
➤ GraphConvolution (1433 -> 16)

3. 코드 분석 - model

```
#model

class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        # print('X1',x.shape)      # torch.Size([2708, 16])
        # As for the variable called training, whenever the mode is changed
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc2(x, adj)
        # print('X2',x.shape)      # torch.Size([2708, 7])

        return F.log_softmax(x, dim=1)
```

$$Z=f(X,A)=\text{softmax}(\tilde{A}\text{ReLU}(\tilde{A} XW(0))W(1))$$

GCN 클래스 생성(nn.module 상속)
초기화 메소드

nfeat, nhid를 인자로하는
GraphConvolution 클래스 객체 gc1 생성
(Cora data 에서 nfeat = 1,433, nhid = 16)

Nhid, nclass를 인자로하는
GraphConvolution 클래스 객체 gc1 생성
(nhid = 16, nclass= 7)

드롭아웃 : 모델의 일반화(보편성)를 위해 일정
비율의 인풋 값들을 없애는 기술

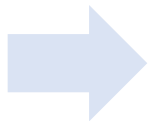
모델 순전파 진행 메소드

입력값 x가 위 초기화 메소드에서 생성된 gc1
층을 지나고 dropout 층을 지나고 gc2층을 지
나고 소프트맥스 층을 지나는 순서로 모델 설정

3. 코드 분석 - Utils

```
#utils
def encode_onehot(labels):
    classes = set(labels)
    classes_dict = {c: np.identity(len(classes))[i, :] for i, c in
                    enumerate(classes)}
    labels_onehot = np.array(list(map(classes_dict.get, labels)),
                             dtype=np.int32)
    return labels_onehot
```

사슴
다람쥐
토끼
양
말
강아지
고양이



1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	1

labels : ['Neural_Networks' 'Rule_Learning' 'Reinforcement_Learning', ...]

classes : {'Theory', 'Genetic_Algorithms', 'Probabilistic_Methods', 'Reinforcement_Learning', 'Rule_Learning', 'Neural_Networks', 'Case_Based'}

classes_dict : {'Theory': array([1., 0., 0., 0., 0., 0., 0.]), 'Genetic_Algorithms': array([0., 1., 0., 0., 0., 0., 0.]), ...}

원-핫 인코딩 함수 (각자 다른 위치에 1 하나를 가지는 벡터)

labels 모든 입력값의 정답들
set 집합, 중복되지않는 요소 추출 = 레이블의 종류 7개

np.identity(len(7)) = 대각값에 1을 가지는 7x7행렬
위 행렬의 각 줄과 각 레이블을 하나씩 매칭시킨다.

모든 레이블을 원핫인코딩 벡터로 변경한다.(map함수)

원핫인코딩으로 변경된 레이블벡터를 반환한다.

3. 코드 분석 - Utils

```
def load_data(path="/content/drive/MyDrive/GCN_exercise/data/cora/", dataset="cora"):
    print('Loading {} dataset...'.format(dataset))

    idx_features_labels = np.genfromtxt("{}{}.content".format(path, dataset),
                                         dtype=np.dtype(str))
    # print('idx_features_labels', idx_features_labels.shape) # (2708, 1435)
    # print('first', idx_features_labels[:,0])                # id
    # print('last', idx_features_labels[:, -1])                # label, e.g. ['Neural_Networks']
    features = sp.csr_matrix(idx_features_labels[:, 1:-1], dtype=np.float32)
    labels = encode_onehot(idx_features_labels[:, -1])
    # print('labels', labels.shape)                            # (2708, 7)
    idx = np.array(idx_features_labels[:, 0], dtype=np.int32)

    idx_map = {j: i for i, j in enumerate(idx)}
    edges_unordered = np.genfromtxt("{}{}.cites".format(path, dataset),
                                     dtype=np.int32)

    edges = np.array(list(map(idx_map.get, edges_unordered.flatten())),
                     dtype=np.int32).reshape(edges_unordered.shape)
    # |edges| = (edges개수=5429, 2)

    adj = sp.coo_matrix((np.ones(edges.shape[0]), (edges[:, 0], edges[:, 1])),
                        shape=(labels.shape[0], labels.shape[0]),
                        dtype=np.float32) # |labels| =(2708,8), 2708 = # data
```

노드(논문) : 2708개 논문에 각 논문에 사용하는 단어 1433개
 노드 레이블 : 논문의 레이블
 Edges : 인용관계

데이터 읽어오기 함수

데이터가 들어있는 위치와 이름을 인자로 넣어줌

위치에 들어있는 content.txt파일의 내용을 심표로 구분하여
 넘파이배열로 불러온다.

불러온 data의 첫 줄(id)과 끝 줄(label)만 빼고 features 변
 수에 넣는다.

마지막 줄은 label 변수에 넣는다.

첫번째 줄은 idx 변수에 넣는다.

입력값의 이름과 순번(인덱스)의 위치를 바꾼다.

(1380 : 0 이라고 저장)

cites.txt 파일에 들어있는 내용을 심표로 구분하여 넘파이배
 열로 불러온다.

엣지 데이터를 1차원으로 바꾸고 이름대신 순번으로 바꿔준
 다.

인접행렬생성 coo_matrix는 희소한 행렬을 인덱스만 표기함
 으로서 간단하게 저장하는 자료형태임

3. 코드 분석 -Utils

$$\text{adj} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{adj.T} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \text{Adj.T} > \text{adj} = \begin{bmatrix} 0 & 0 & 0 \\ \text{T} & 0 & 0 \\ \text{T} & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

```
# symmetric adjacency matrix
adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)
# print('adj',adj.shape) # (2708, 2708)

features = normalize(features)
adj = normalize(adj + sp.eye(adj.shape[0])) # Add self loop to adjacency matrix

idx_train = range(140)
idx_val = range(200, 500)
idx_test = range(500, 1500)

features = torch.FloatTensor(np.array(features.todense()))

labels = torch.LongTensor(np.where(labels)[1])

adj = sparse_mx_to_torch_sparse_tensor(adj)

# train, val, test에 index를 지정하여 semi-supervised(transductive)를 사용하고자 함
idx_train = torch.LongTensor(idx_train)
idx_val = torch.LongTensor(idx_val)
idx_test = torch.LongTensor(idx_test)

return adj, features, labels, idx_train, idx_val, idx_test
```

인접행렬이 아직 대각 위 부분만 있기때문에
전치행렬 구해서 더해줌

features 정규화(값들을 0과1사이로 맞춰줌)
인접행렬에 단위행렬(self-loop) 더해서 정규화

트레이닝 데이터 140개
벨리데이션 데이터 300개
테스트 데이터 1000개

희소행렬로 묶어둔 걸 원래행렬로 불러오면서 텐서로 변환
원핫인코딩으로 변환한 label에서 1이 들어있는 위치 반환
인접행렬 텐서로 변환

3. 코드 분석 - Utils

```
def normalize(mx):
    """Row-normalize sparse matrix"""
    rowsum = np.array(mx.sum(1))
    r_inv = np.power(rowsum, -1).flatten()
    r_inv[np.isinf(r_inv)] = 0.
    r_mat_inv = sp.diags(r_inv)
    mx = r_mat_inv.dot(mx)
    return mx

def accuracy(output, labels):
    preds = output.max(1)[1].type_as(labels)
    correct = preds.eq(labels).double()
    correct = correct.sum()
    return correct / len(labels)
```

정규화 함수

한 행이 하나의 데이터이기때문에 행을 기준으로 normalize

rowsum = 행의 총합

r_inv = 역수 취해준 뒤 1차원으로 변경

무한대 값 -> 0으로 바꿔줌 (0을 역수로 취해서 무한대가 나옴)

원래의 매트릭스와 곱친 값을 곱함

$$\begin{matrix}
 \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} & \begin{matrix} = 2 \\ = 3 \\ = 1 \end{matrix} & => & \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/3 & 0 \\ 0 & 0 & 1/1 \end{bmatrix} & \times & \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} & = & \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/3 & 1/3 & 1/3 \\ 0 & 1/1 & 0 \end{bmatrix} \\
 \text{mx} & \text{rowsum} & & \text{r_mat_inv} & & \text{mx} & & \text{최종값}
 \end{matrix}$$

정확도 구하는 함수

1차원의 최대값을 구하고 타입을 labels의 타입으로 변경

같은지 비교하고 64비트 float 타입으로 바꾼다.

정답을 맞춘 갯수를 세고

전체 값으로 나눈다.

3. 코드 분석 - Training

```
#Training
import easydict

args = easydict.EasyDict({"no-cuda":False, "fastmode":False, "seed":42, \
                          "epochs":200, "lr":0.01, "weight_decay":5e-4, \
                          "hidden":16, "dropout":0.5, "cuda":True})

np.random.seed(args.seed)
torch.manual_seed(args.seed)
if args.cuda:
    torch.cuda.manual_seed(args.seed)

# Load data
adj, features, labels, idx_train, idx_val, idx_test = load_data()
print('features', features.shape) # torch.Size([2708, 1433])
print('labels', labels.shape)    # torch.Size([2708])
print('adj', adj.shape)          # torch.Size([2708, 2708])

# Model and optimizer
model = GCN(nfeat=features.shape[1],          # 1433
            nhid=args.hidden,                 # 16
            nclass=labels.max().item() + 1,   # 7
            dropout=args.dropout)
optimizer = optim.Adam(model.parameters(),
                        lr=args.lr, weight_decay=args.weight_decay)
```

하이퍼 파라미터 조절해주는 부분
(일반적인 코드의 argument 인데 colab 특성상 커멘드라인으로 입력값을 줄 수 없어서 easydict 로 대체)

랜덤 시드 고정
재현이 가능하도록 랜덤시 나오는 숫자가 늘 같도록
시드값을 고정하는 부분

Utils에서 만든 load_data() 함수를 사용해 데이터를 불러오는 부분

Model에서 만든 GCN모델을 불러와 초기화 하는 부분

Import 한 최적화기 패키지를 이용하여 사용하고자 하는 Adam optimizer 불러오기

3. 코드 분석 - Training

```
def train(epoch):  
    t = time.time()  
    model.train()  
    optimizer.zero_grad()  
    output = model(features, adj)  
    loss_train = F.nll_loss(output[idx_train], labels[idx_train])  
    acc_train = accuracy(output[idx_train], labels[idx_train])  
    loss_train.backward()  
    optimizer.step()
```

```
loss_val = F.nll_loss(output[idx_val], labels[idx_val])  
acc_val = accuracy(output[idx_val], labels[idx_val])  
print('Epoch: {:04d}'.format(epoch+1),  
      'loss_train: {:.4f}'.format(loss_train.item()),  
      'acc_train: {:.4f}'.format(acc_train.item()),  
      'loss_val: {:.4f}'.format(loss_val.item()),  
      'acc_val: {:.4f}'.format(acc_val.item()),  
      'time: {:.4f}s'.format(time.time() - t))
```

학습 함수
현재 시간 저장
모델 학습모드로 바꿈
옵티마이저 0으로 초기화
모델에 데이터를 넣어주는 코드
import했던 Funtional에서 nll_loss 함수 불러와 로스 계산
Train 정확도 구하기
loss를 가지고 역전파 진행 (gradient 계산)
파라미터 갱신

벨리데이션 loss 계산
벨리데이션 정확도 계산
에포크에 따른 학습 로스, 학습 정확도, 벨리데이션 로스, 벨리데이션 정확도, 시간 출력

3. 코드 분석 - Training

```
def test():
    model.eval()
    output = model(features, adj)
    loss_test = F.nll_loss(output[idx_test], labels[idx_test])
    acc_test = accuracy(output[idx_test], labels[idx_test])
    print("Test set results:",
          "loss= {:.4f}".format(loss_test.item()),
          "accuracy= {:.4f}".format(acc_test.item()))

# Train model
t_total = time.time()
for epoch in range(args.epochs):
    train(epoch)
print("Optimization Finished!")
print("Total time elapsed: {:.4f}s".format(time.time() - t_total))

# Testing
test()
```

테스트 함수

모델 테스트모드로 전환
모델에 데이터 넣어주는 코드
결과에 대한 loss 계산
결과에 대한 정확도 계산
테스트 정확도 출력

Main

현재시간 저장
에포크 수만큼 train 함수 진행
총 걸린 시간 출력
test 함수 진행

4. 결과

```
Loading cora dataset...
Epoch: 0001 loss_train: 1.9098 acc_train: 0.2286 loss_val: 1.8914 acc_val: 0.3400 time: 2.9908s
Epoch: 0002 loss_train: 1.9010 acc_train: 0.3214 loss_val: 1.8820 acc_val: 0.3500 time: 0.0062s
Epoch: 0003 loss_train: 1.8890 acc_train: 0.3071 loss_val: 1.8734 acc_val: 0.3500 time: 0.0052s
Epoch: 0004 loss_train: 1.8801 acc_train: 0.2929 loss_val: 1.8649 acc_val: 0.3500 time: 0.0050s
Epoch: 0005 loss_train: 1.8678 acc_train: 0.2929 loss_val: 1.8566 acc_val: 0.3500 time: 0.0051s

Epoch: 0191 loss_train: 0.4143 acc_train: 0.9286 loss_val: 0.6915 acc_val: 0.8167 time: 0.0049s
Epoch: 0192 loss_train: 0.4045 acc_train: 0.9357 loss_val: 0.6902 acc_val: 0.8167 time: 0.0052s
Epoch: 0193 loss_train: 0.4020 acc_train: 0.9357 loss_val: 0.6906 acc_val: 0.8100 time: 0.0050s
Epoch: 0194 loss_train: 0.4071 acc_train: 0.9357 loss_val: 0.6920 acc_val: 0.8133 time: 0.0051s
Epoch: 0195 loss_train: 0.4041 acc_train: 0.9429 loss_val: 0.6946 acc_val: 0.8100 time: 0.0049s
Epoch: 0196 loss_train: 0.3292 acc_train: 0.9714 loss_val: 0.6956 acc_val: 0.8067 time: 0.0062s
Epoch: 0197 loss_train: 0.4257 acc_train: 0.9286 loss_val: 0.6944 acc_val: 0.8067 time: 0.0050s
Epoch: 0198 loss_train: 0.3766 acc_train: 0.9429 loss_val: 0.6929 acc_val: 0.8100 time: 0.0050s
Epoch: 0199 loss_train: 0.4016 acc_train: 0.9214 loss_val: 0.6912 acc_val: 0.8100 time: 0.0052s
Epoch: 0200 loss_train: 0.4313 acc_train: 0.9071 loss_val: 0.6882 acc_val: 0.8067 time: 0.0052s
Optimization Finished!
Total time elapsed: 4.5846s
Test set results: loss= 0.7123 accuracy= 0.8340
```

에포크에 따라 줄어드는 loss, 올라가는 정확도를 확인 할 수 있다.
전체 학습시간은 4.58초

학습 정확도 0.9071
벨리데이션 정확도 0.8067
테스트 정확도 0.8340

학습데이터셋에 과적합은 피하기 어렵다.
Val 정확도로 학습 중 test 정확도를 예측해볼 수 있다.

감사합니다.

질문함 : jwjw9603@g.skku.edu