

# Graph Convolution networks

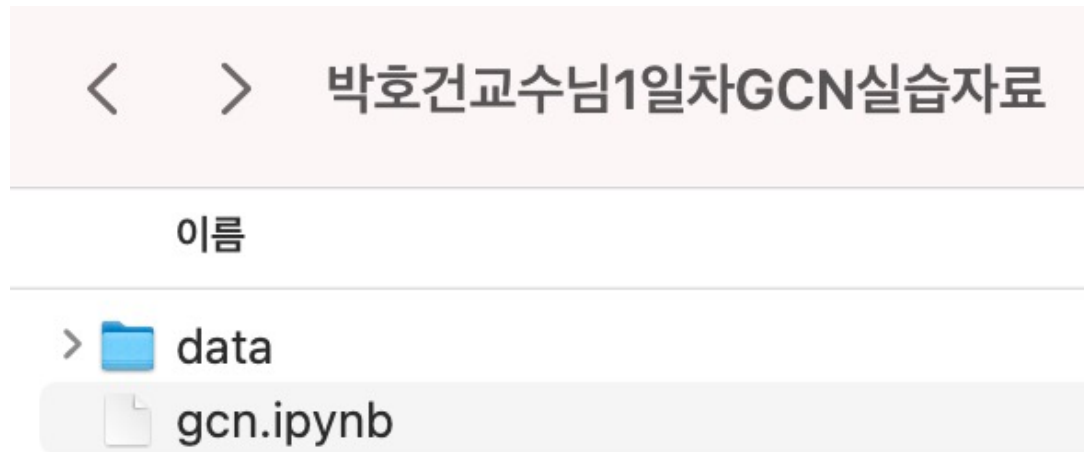
## 코드 리뷰 및 실습

성균관대학교 인공지능연구소  
딥러닝 특강(심화) - 박호건교수님  
수업 1일차 실습자료

## 목 차

- 0. 준비물
- 1. 세팅
- 2. 셸(shell)별 코드 소개
- 3. 세부 코드 분석
- 4. 결과

## 0. 준비물



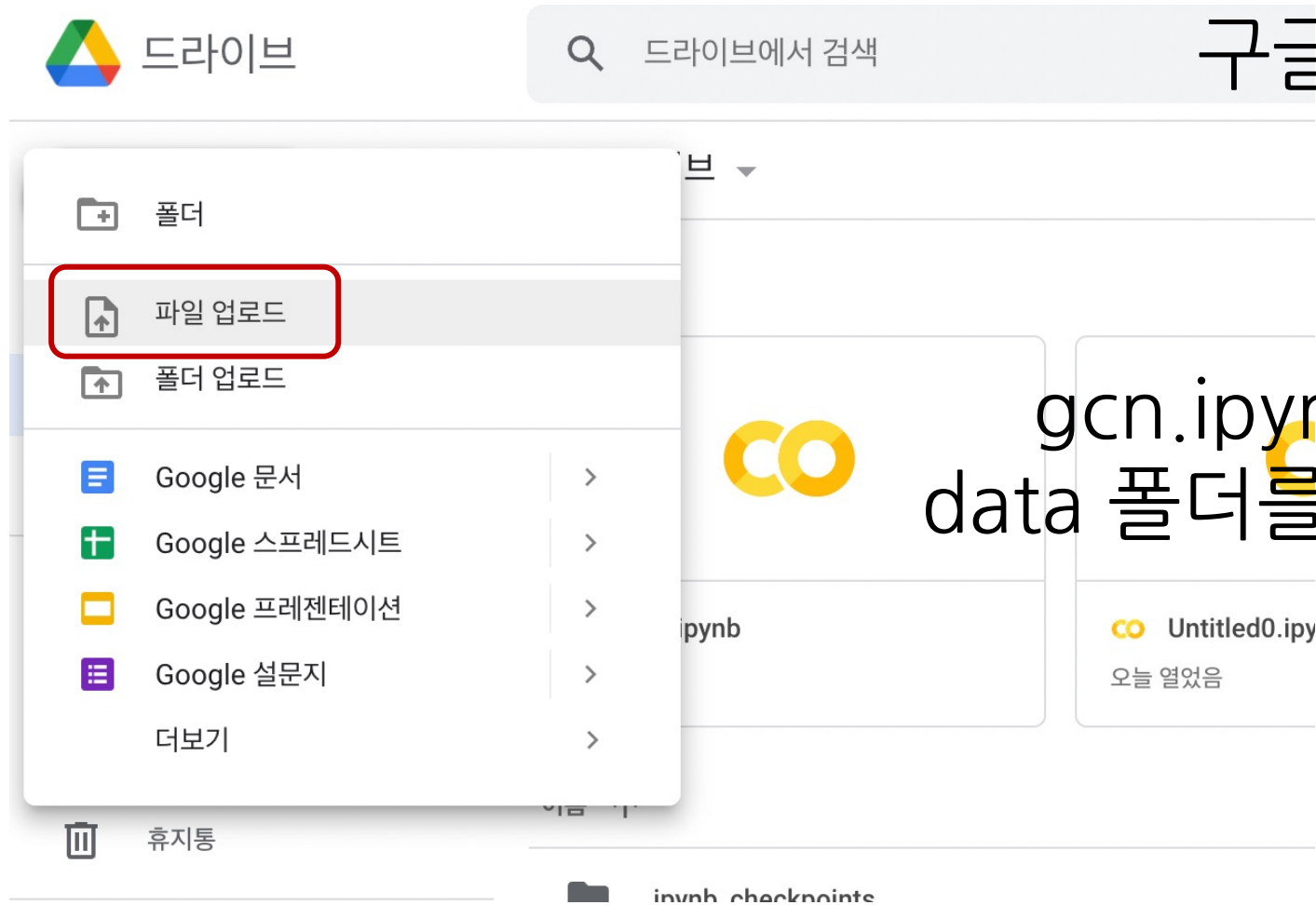
1. Colab(코랩) 실습 파일  
gcn.ipynb

2. 데이터가 들어있는 폴더  
data

## 1. 셋팅

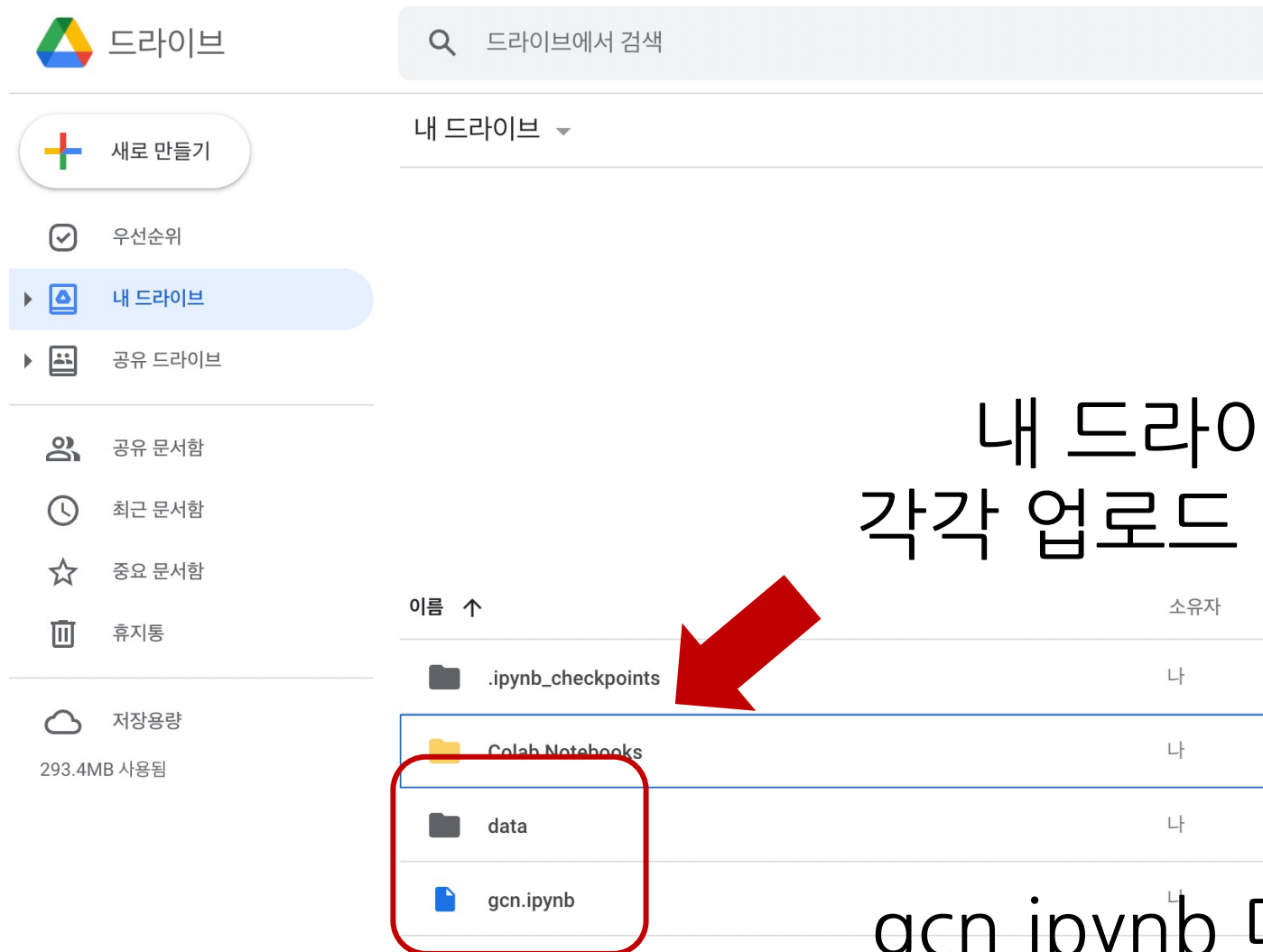
<https://drive.google.com/drive/my-drive>

구글 드라이브 접속



gcnn.ipynb 파일과  
data 폴더를 각각 업로드

# 1. 셋팅



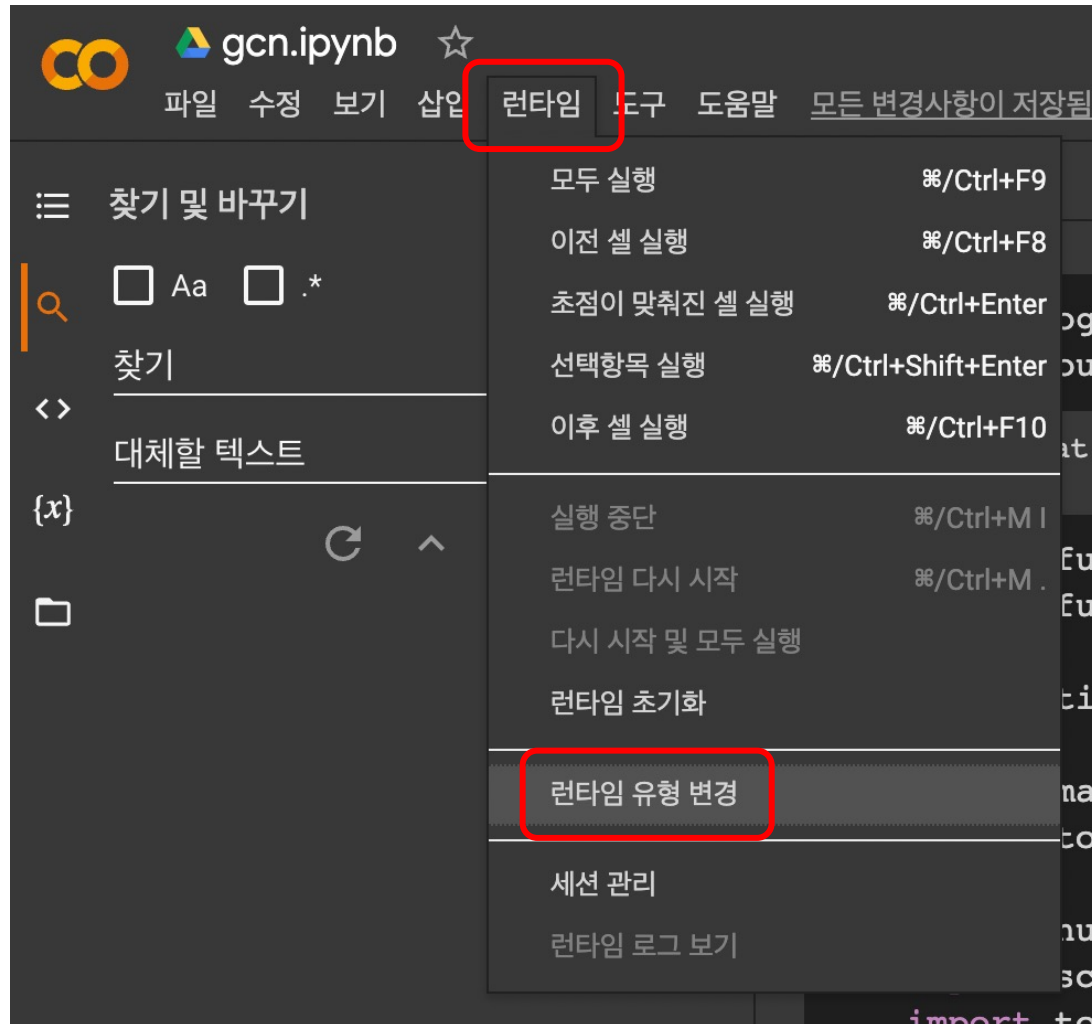
The screenshot shows the Google Drive web interface. On the left sidebar, the '내 드라이브' (My Drive) option is selected. The main area displays a list of files and folders. A red arrow points to the '.ipynb\_checkpoints' folder. Below it, the 'Colab Notebooks' folder is highlighted with a red box. Inside this folder, the 'data' folder and the 'gcn.ipynb' file are also highlighted with a red box. The file 'gcn.ipynb' is a Jupyter Notebook file, indicated by its blue icon.

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

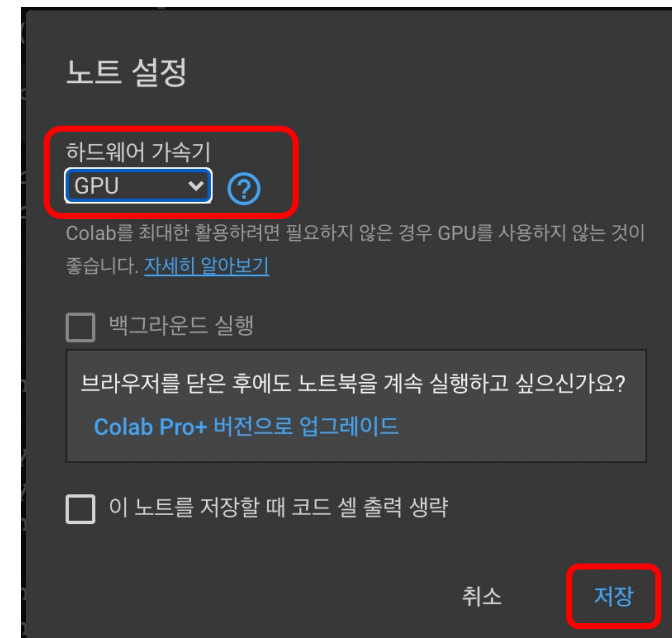
내 드라이브 내에  
각각 업로드 되면 완료!!

gcn.ipynb 더블클릭!

# 1. 세팅

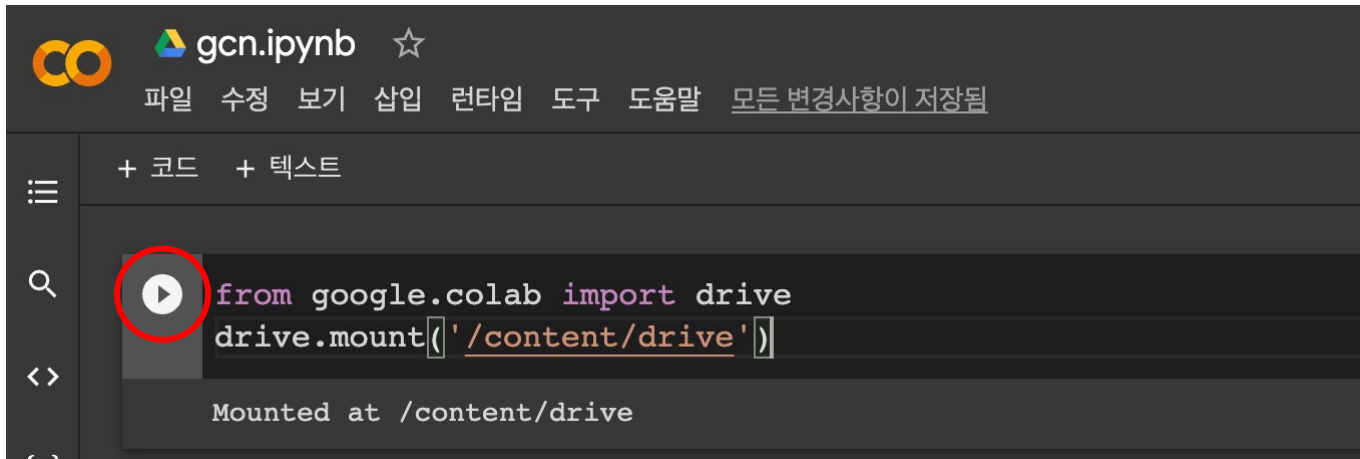


런타임 -> 런타임 유형변경 클릭!  
하드웨어 가속기 GPU로  
변경 후 저장



# 1. 세팅

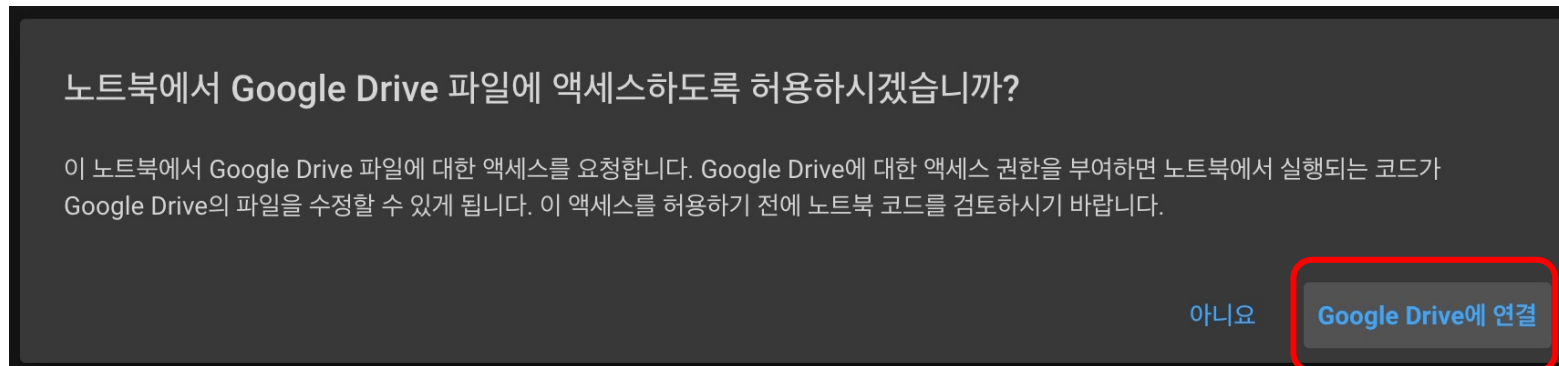
첫번째 셀(shell) 실행!



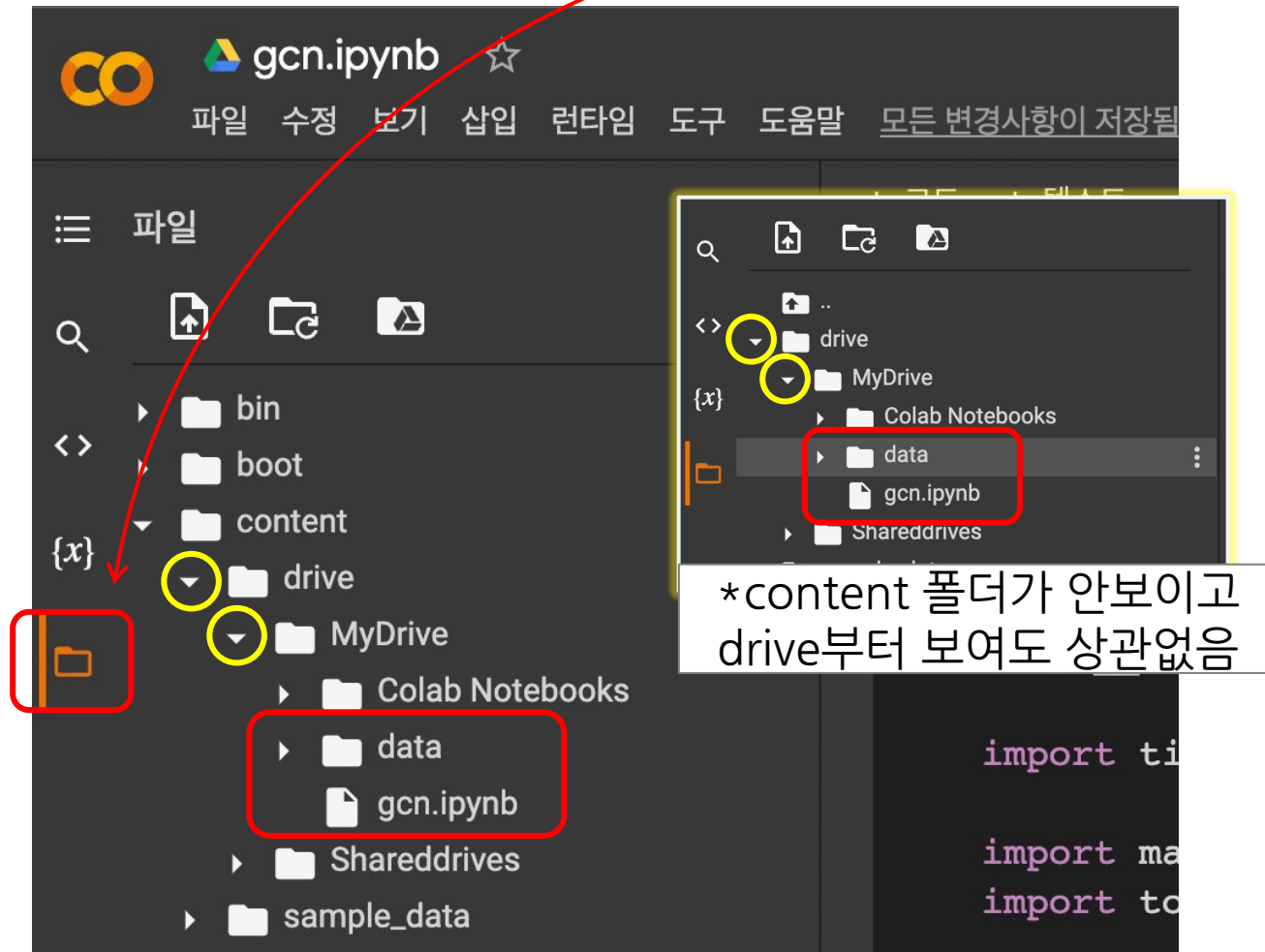
```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

Google Drive에 연결 ->  
로그인 -> 계속



## 1. 세팅



왼쪽 하단의 파일 창 클릭  
파일 옆 세모 클릭

content

▸ drive

▸ MyDrive

...

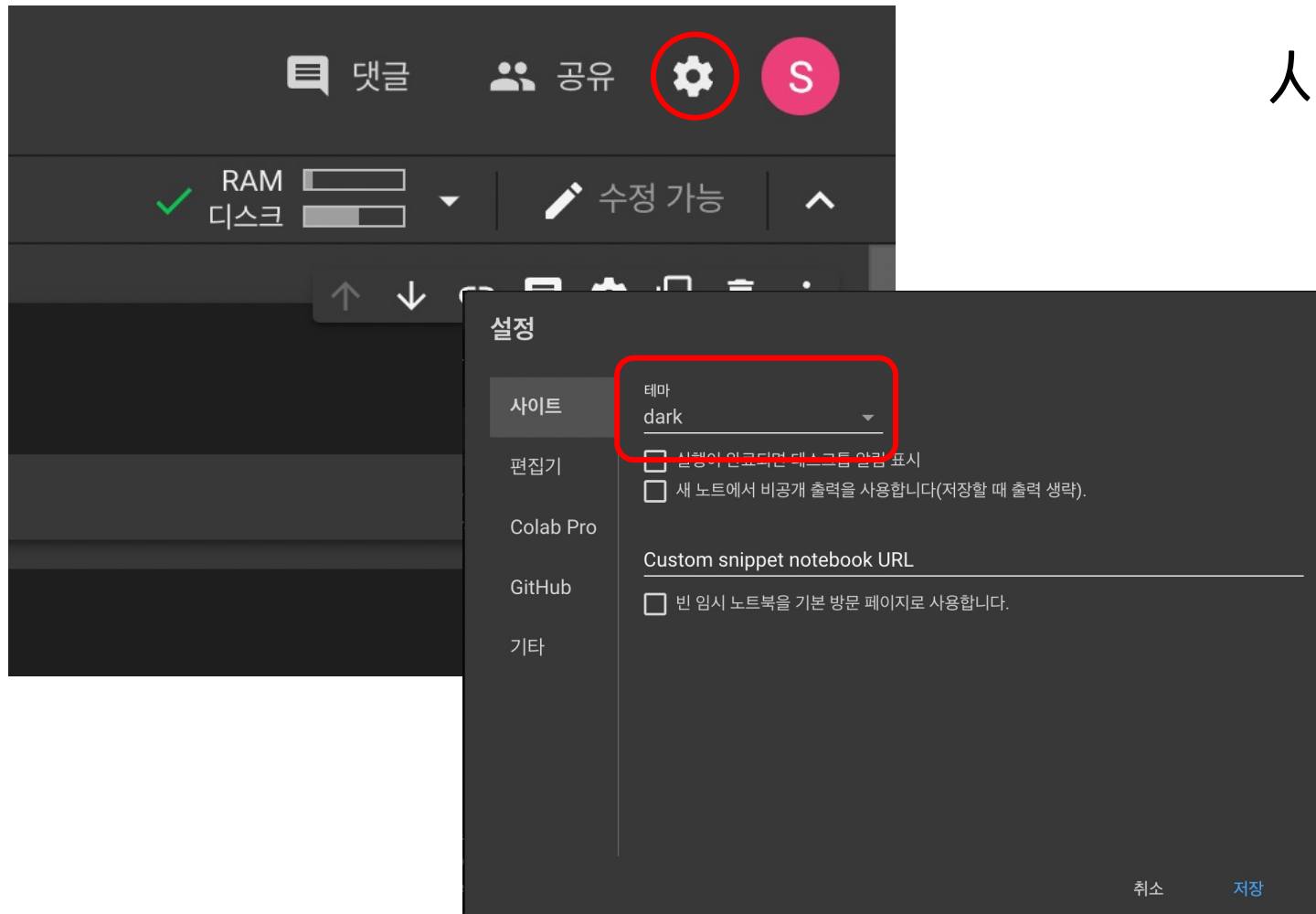
▸ data

▸ gcn.ipynb

아까 업로드한 data와  
gcn.ipynb 파일이 보인다면  
연결 성공!



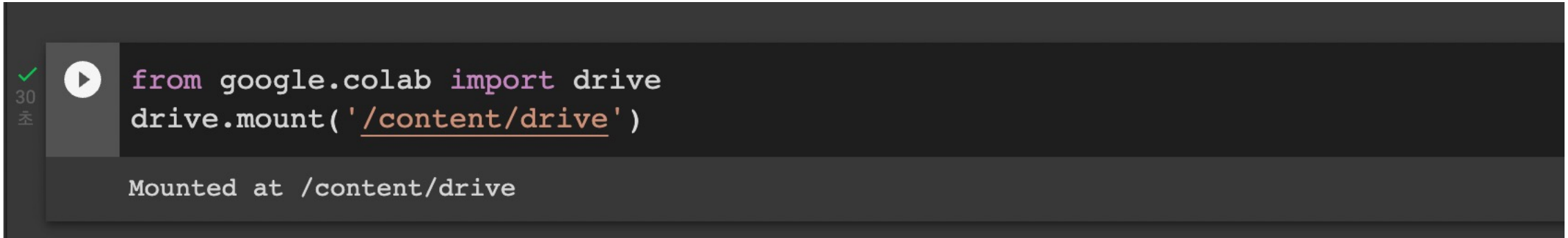
## \* 별첨 - 다크모드 사용법



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

첫번째 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
```

두번째 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) + ')'
```

세번째 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)
```

네번째 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/data/cora/", dataset="cora"):
    print('Loading {} dataset...'.format(dataset))

    idx_features_labels = np.genfromtxt("{}{}.content".format(path, dataset),
                                         dtype=np.dtype(str))
    features = sp.csr_matrix(idx_features_labels[:, 1:-1], dtype=np.float32)
    labels = encode_onehot(idx_features_labels[:, -1])

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

    adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)

    features = normalize(features)
    adj = normalize(adj + sp.eye(adj.shape[0]))

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

다섯번째 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 : 머신러닝을 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):
    """
    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))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
```

Module을 상속하는 클래스를 만듦

**super** : 부모 클래스의 init도 상속받음  
인자는 상속받는 자식의 이름 (python2)  
\*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)
    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.spmv(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) + ')
```

모델 파라미터를 초기화해주는 메소드

$$\frac{1}{\sqrt{\text{weight.size}(1)}}$$
 weight 크기 = in\_feature \* out\_feature  
의 첫번째 값 = in\_feature = 1,433

tensor.data.uniform\_ : 범위 사이에 균등하게 값이 분포  
입력 개수의 표준편차로 나누어 주는 것이 일반적

모델 통과 메소드

torch.mm 벡터곱

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

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))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc2(x, adj)
        return F.log_softmax(x, dim=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)

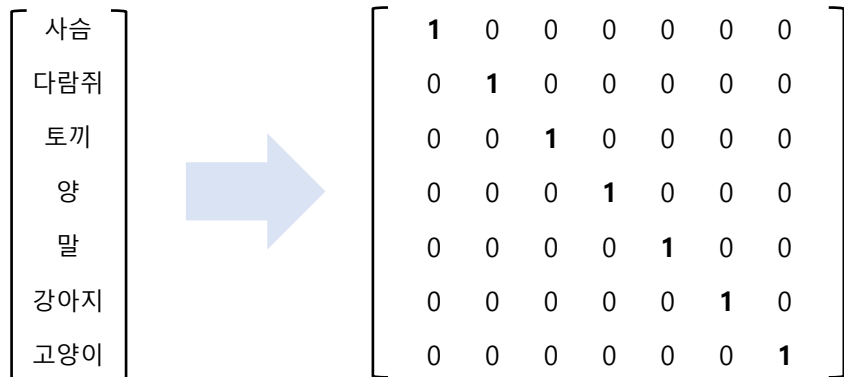
드롭아웃 : 모델의 일반화(보편성)를 위해 일정  
비율의 인풋 값들을 없애는 기술

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
```



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.

원핫인코딩으로 변경된 레이블벡터를 반환한다.

### 3. 코드 분석 - Utils

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

    idx_features_labels = np.genfromtxt("{}{}.content".format(path, dataset),
                                         dtype=np.dtype(str))

    features = sp.csr_matrix(idx_features_labels[:, 1:-1], dtype=np.float32)
    labels = encode_onehot(idx_features_labels[:, -1])

    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)

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

데이터 읽어오기 함수

데이터가 들어있는 위치와 이름을 인자로 넣어줌

위치에 들어있는 content.txt파일의 내용을 심표로 구분하여 넘파이배열로 불러온다.

불러온 data의 첫줄과 끝줄만 빼고 features 변수에 넣는다.  
마지막 줄은 label 변수에 넣는다.  
첫번째 줄은 idx 변수에 넣는다.

입력값의 이름과 순번(인덱스)의 위치를 바꾼다.

(1380 = 1번 이라고 저장된 것을 1번 = 1380으로 바꿈)

cites.txt 파일에 들어있는 내용을 심표로 구분하여 넘파이배열로 불러온다.

엣지 데이터를 1차원으로 바꾸고 이름대신 순번으로 바꿔준다.

인접행렬생성 coo\_matrix는 희소한 행렬을 인덱스만 표기함으로써 간단하게 저장하는 자료형태임

### 3. 코드 분석 -Utils

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

```
adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)

features = normalize(features)
adj = normalize(adj + sp.eye(adj.shape[0]))

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)

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 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()

# 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)
```

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 역전파진행 ( gradient 계산 )  
파라미터 갱신

벨리데이션 loss 계산  
벨리데이션 정확도 계산  
에폭에 따른 학습 로스, 학습 정확도, 벨리데이션 로스, 벨리  
데이션 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  
에폭 수만큼 train 함수 진행  
총 걸린 시간 출력  
test 함수 진행

## 4. 결과

```
Loading cora dataset...
Epoch: 0001 loss_train: 1.9418 acc_train: 0.2000 loss_val: 1.9047 acc_val: 0.3500 time: 0.0077s
Epoch: 0002 loss_train: 1.9225 acc_train: 0.2929 loss_val: 1.8947 acc_val: 0.3500 time: 0.0108s
Epoch: 0003 loss_train: 1.9123 acc_train: 0.2929 loss_val: 1.8855 acc_val: 0.3500 time: 0.0076s
Epoch: 0004 loss_train: 1.9040 acc_train: 0.2929 loss_val: 1.8767 acc_val: 0.3500 time: 0.0084s
Epoch: 0005 loss_train: 1.8931 acc_train: 0.2929 loss_val: 1.8681 acc_val: 0.3500 time: 0.0100s
Epoch: 0191 loss_train: 0.4459 acc_train: 0.9571 loss_val: 0.6923 acc_val: 0.8267 time: 0.0096s
Epoch: 0192 loss_train: 0.4488 acc_train: 0.9429 loss_val: 0.6899 acc_val: 0.8300 time: 0.0105s
Epoch: 0193 loss_train: 0.4289 acc_train: 0.9286 loss_val: 0.6884 acc_val: 0.8300 time: 0.0097s
Epoch: 0194 loss_train: 0.4162 acc_train: 0.9500 loss_val: 0.6870 acc_val: 0.8300 time: 0.0118s
Epoch: 0195 loss_train: 0.4145 acc_train: 0.9500 loss_val: 0.6862 acc_val: 0.8267 time: 0.0109s
Epoch: 0196 loss_train: 0.3705 acc_train: 0.9429 loss_val: 0.6855 acc_val: 0.8233 time: 0.0102s
Epoch: 0197 loss_train: 0.4360 acc_train: 0.9357 loss_val: 0.6851 acc_val: 0.8200 time: 0.0098s
Epoch: 0198 loss_train: 0.4056 acc_train: 0.9571 loss_val: 0.6847 acc_val: 0.8233 time: 0.0100s
Epoch: 0199 loss_train: 0.4111 acc_train: 0.9357 loss_val: 0.6836 acc_val: 0.8267 time: 0.0099s
Epoch: 0200 loss_train: 0.4426 acc_train: 0.9143 loss_val: 0.6825 acc_val: 0.8267 time: 0.0105s
Optimization Finished!
Total time elapsed: 2.2469s
Test set results: loss= 0.7285 accuracy= 0.8390
```

에폭에 따라 줄어드는 loss, 올라가는 정확도를 확인 할 수 있다.  
전체 학습시간은 2.24초

학습 정확도 0.9143  
벨리데이션 정확도 0.8267  
테스트 정확도 0.83

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

감사합니다.

질문함 : [lsy7451@g.skku.edu](mailto:lsy7451@g.skku.edu)