

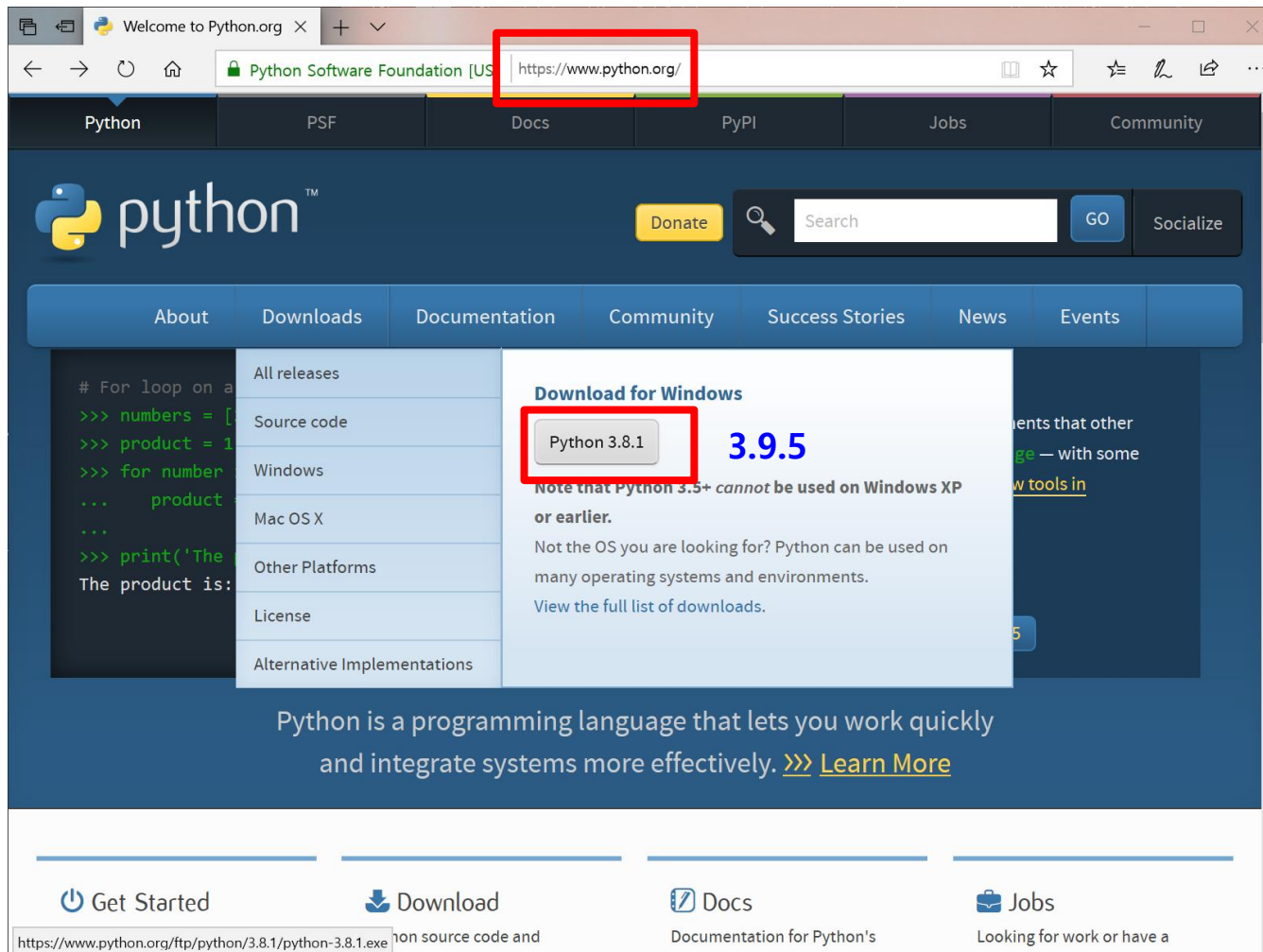
---

# Prerequisite for ML-based NLP

---

## Practical Exercise

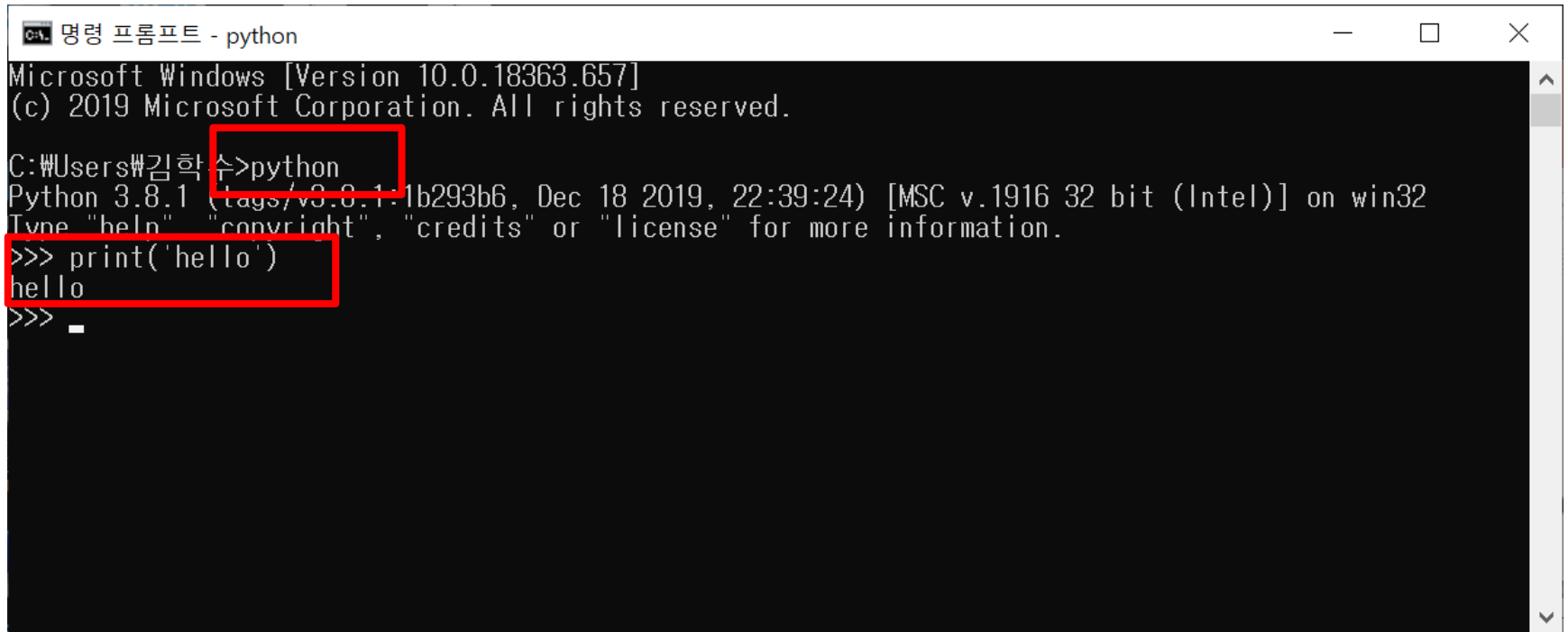
# Python 설치



# Python 설치



# Python 설치 확인



```
명령 프롬프트 - python
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\김학수>python
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('hello')
hello
>>> _
```



# PIP (Python Package Index)

- PIP: 파이썬으로 작성된 패키지 소프트웨어를 관리하는 패키지 관리 시스템

```
C:\Users\김학수>python -m pip install --upgrade pip
Collecting pip
  Downloading https://files.pythonhosted.org/packages/54/0c/d01aa759fdc501a58f431eb594a17495f15b88da142ce14b5845662c13f3/pip-20.0.2-py2.py3-none-any.whl (1.4MB)
    |#####| 1.4MB 233kB/s
Installing collected packages: pip
  Found existing installation: pip 19.2.3
    Uninstalling pip-19.2.3:
      Successfully uninstalled pip-19.2.3
  Successfully installed pip-20.0.2
```



# Numpy, Scipy, Matplot 설치

- Numpy: 행렬이나 다차원 배열을 쉽게 처리 할 수 있도록 지원하는 라이브러리
- Scipy: 과학 컴퓨팅과 기술 컴퓨팅에 사용되는 라이브러리
- Matplot: 유사한 그래프 표시를 가능케 하는 라이브러리

명령 프롬프트

```
(c) 2019 Microsoft Corporation. All rights reserved.  
C:\Users\김학수>python -m pip install numpy scipy matplotlib  
Collecting numpy  
  Downloading https://files.pythonhosted.org/packages/0e/c3/be53614c4e3490778050e1df48fd463837297d5dd402dae3b500f2050eba/numpy-1.18.1-cp38-cp38-win32.whl (10.8MB)  
    |#####| 10.8MB 1.1MB/s  
Collecting scipy  
  Downloading https://files.pythonhosted.org/packages/db/9e/465a416eb04114e3722b17b0f4fa5235bab8a76961de51db0e5850183fb1/scipy-1.4.1-cp38-cp38-win32.whl (27.9MB)  
    |#####| 27.9MB 3.3MB/s  
Collecting matplotlib  
  Downloading https://files.pythonhosted.org/packages/86/e4/1ef1cb7f2c52345a7e3c8efd3de7ec943818c0011c839b4880c0ba0bb7b1/matplotlib-3.1.3-cp38-cp38-win32.whl (8.9MB)  
    |#####| 8.9MB 6.8MB/s  
Collecting kiwisolver>=1.0.1 (from matplotlib)  
  Downloading https://files.pythonhosted.org/packages/b9/b1/118f3d5dee660bbe4548f06dcd0e1a10e45458326c3d0efad7dbbf28be24/kiwisolver-1.1.0-cp38-none-win32.whl (43kB)  
    |#####| 51kB ...
```



# Numpy, Scipy, Matplot 설치

---

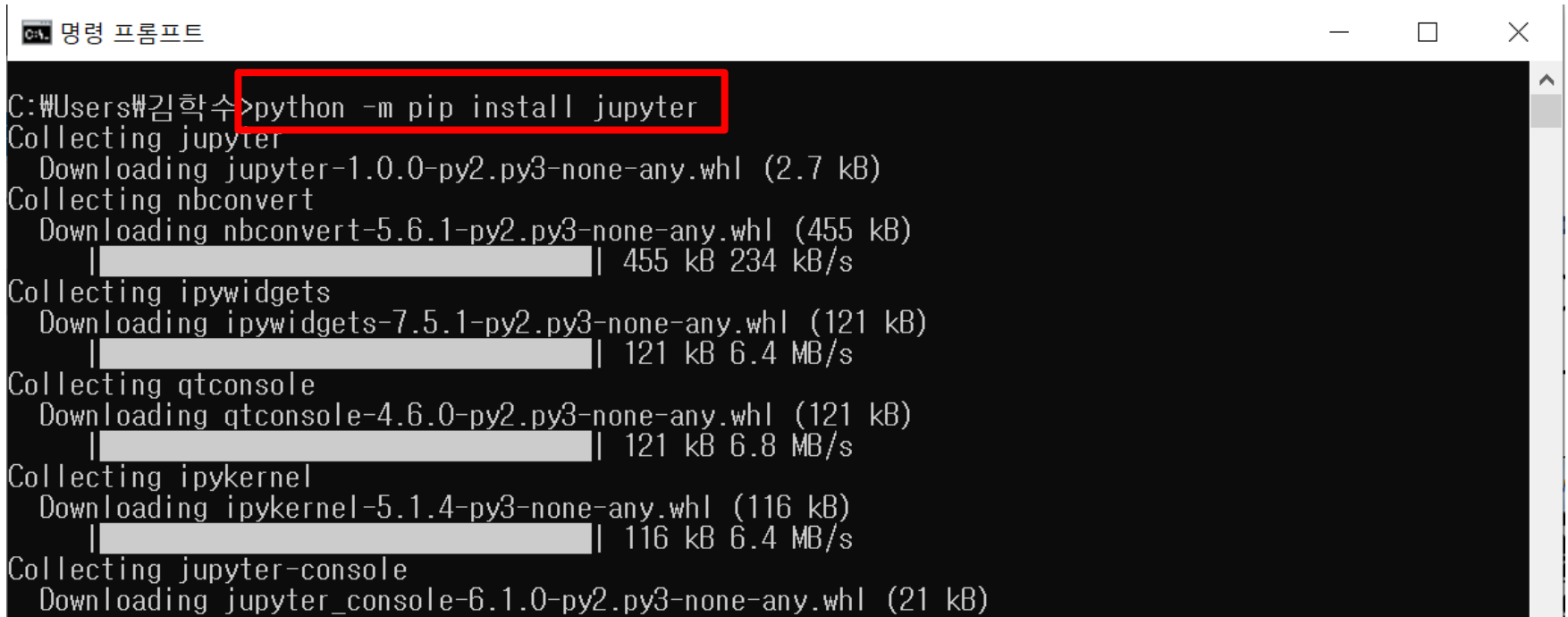
```
C:\Users\김학수>python
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information
>>> import numpy
>>> import scipy
>>> import matplotlib
>>>
```

설치확인!



# Jupyter Notebook 설치

- Jupyter Notebook: 웹 브라우저에서 파이썬 코드를 작성하고 실행해 볼 수 있는 개발도구



```
명령 프롬프트
C:\Users\김학수>python -m pip install jupyter
Collecting jupyter
  Downloading jupyter-1.0.0-py2.py3-none-any.whl (2.7 kB)
Collecting nbconvert
  Downloading nbconvert-5.6.1-py2.py3-none-any.whl (455 kB)
    | 455 kB 234 kB/s
Collecting ipywidgets
  Downloading ipywidgets-7.5.1-py2.py3-none-any.whl (121 kB)
    | 121 kB 6.4 MB/s
Collecting qtconsole
  Downloading qtconsole-4.6.0-py2.py3-none-any.whl (121 kB)
    | 121 kB 6.8 MB/s
Collecting ipykernel
  Downloading ipykernel-5.1.4-py3-none-any.whl (116 kB)
    | 116 kB 6.4 MB/s
Collecting jupyter-console
  Downloading jupyter_console-6.1.0-py2.py3-none-any.whl (21 kB)
```





# Jupyter 실행

```
명령 프롬프트 - jupyter notebook

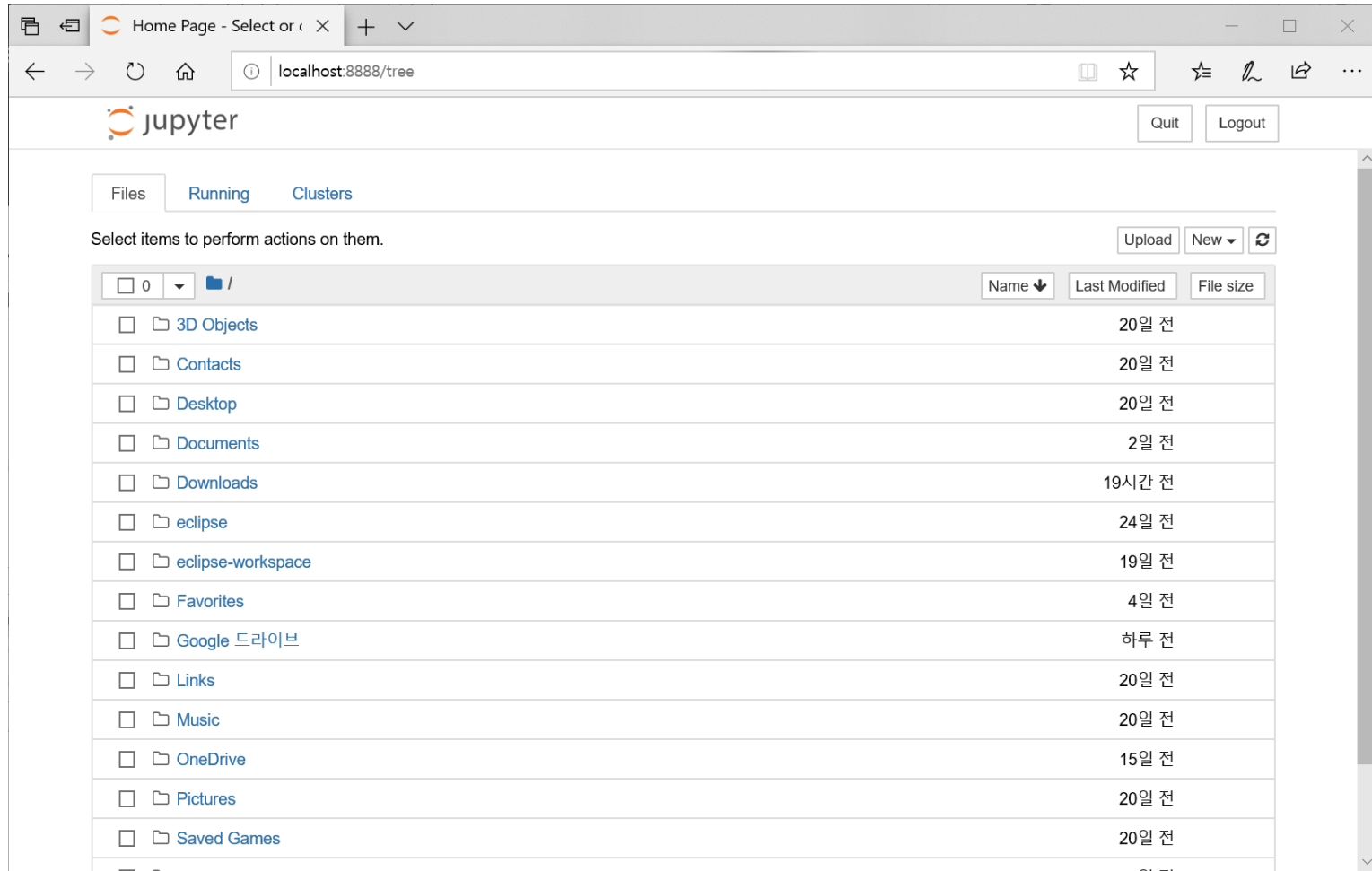
C:\Users\김학수>
C:\Users\김학수>jupyter notebook
[I 10:24:19.840 NotebookApp] Writing notebook server cookie secret to C:\Users\김학수\AppData\Roaming\jupyter\runtime\notebook_cookie_secret
[W 10:24:20.300 NotebookApp] Terminals not available (error was No module named 'winpty.cythonpty')
[I 10:24:20.300 NotebookApp] Serving notebooks from local directory: C:\Users\김학수
[I 10:24:20.300 NotebookApp] The Jupyter Notebook is running at:
[I 10:24:20.300 NotebookApp] http://localhost:8888/?token=a3bf7025eebf7610c5973563a16494be8c39ce8082947215
or http://127.0.0.1:8888/?token=a3bf7025eebf7610c5973563a16494be8c39ce8082947215
[I 10:24:20.300 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 10:24:20.331 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/%EA%B9%80%ED%95%99%EC%88%98/AppData/Roaming/jupyter/runtime/nbserver-14056-open.html
0 copy and paste one of these URLs:
http://localhost:8888/?token=a3bf7025eebf7610c5973563a16494be8c39ce8082947215
or http://127.0.0.1:8888/?token=a3bf7025eebf7610c5973563a16494be8c39ce8082947215
[I 10:25:28.675 NotebookApp] 302 GET /?token=a3bf7025eebf7610c5973563a16494be8c39ce8082947215 (:::1) 0.00ms
```

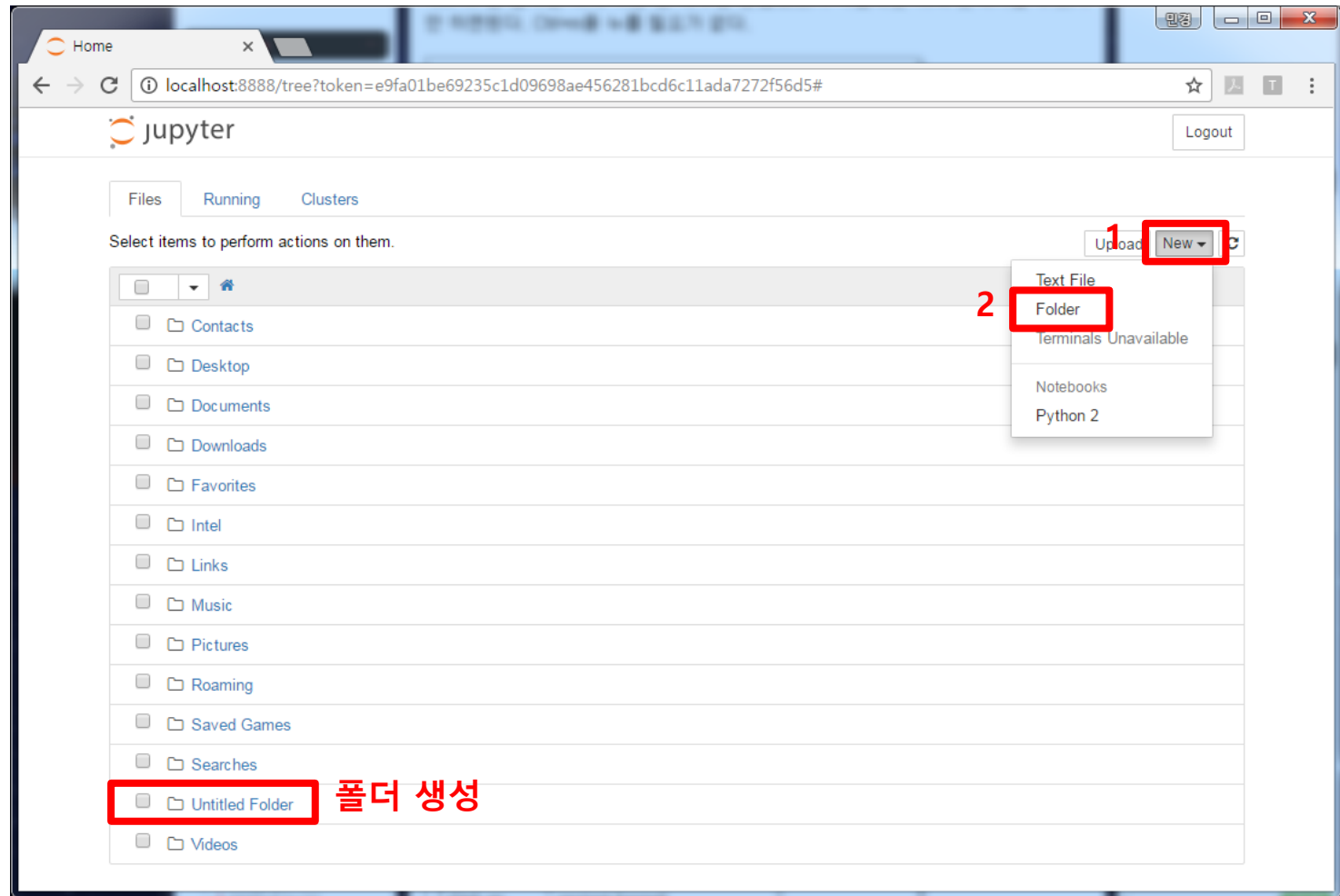
웹브라우저에 복사하여 붙여넣기



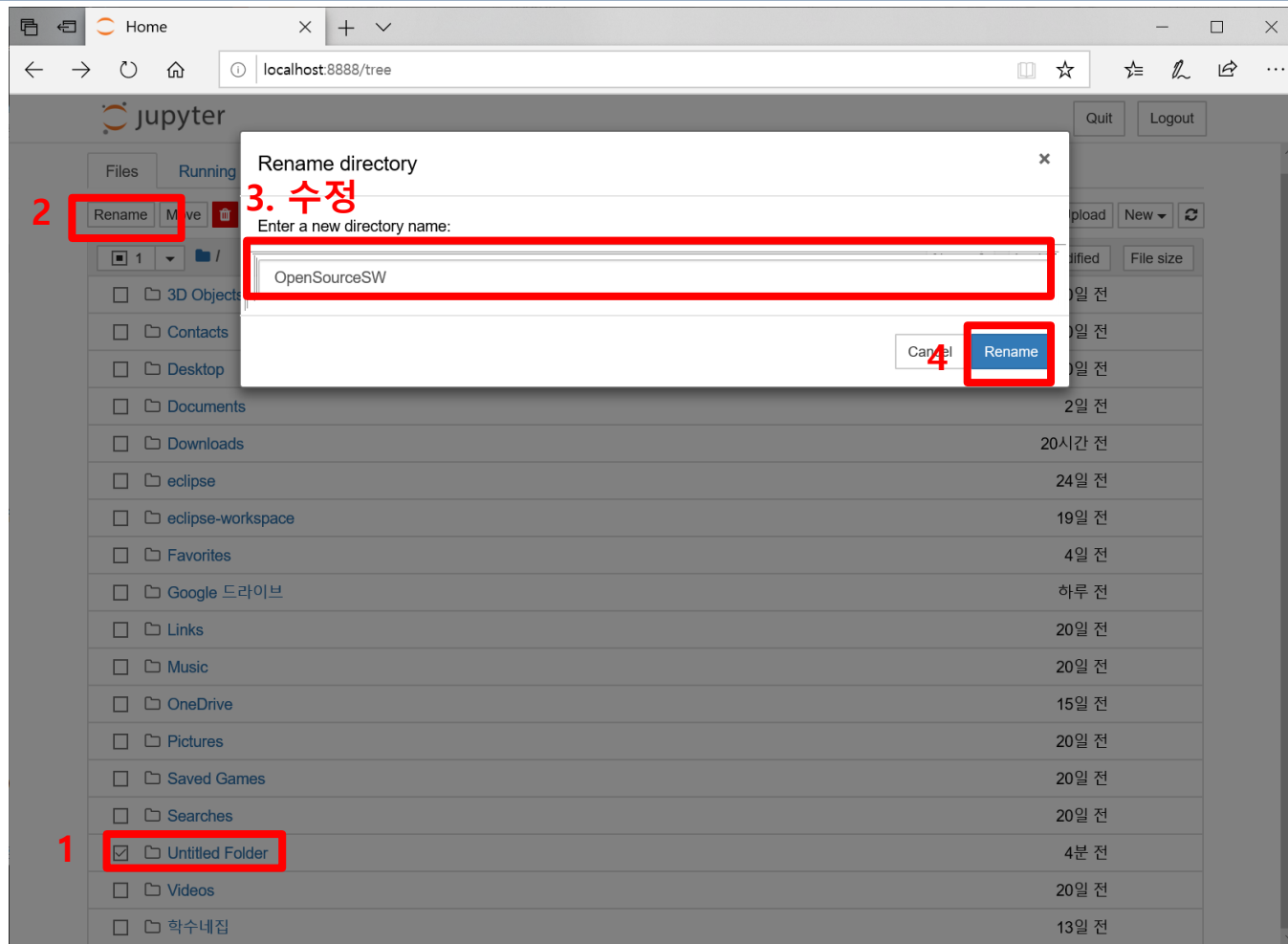
# Jupyter 실행 화면



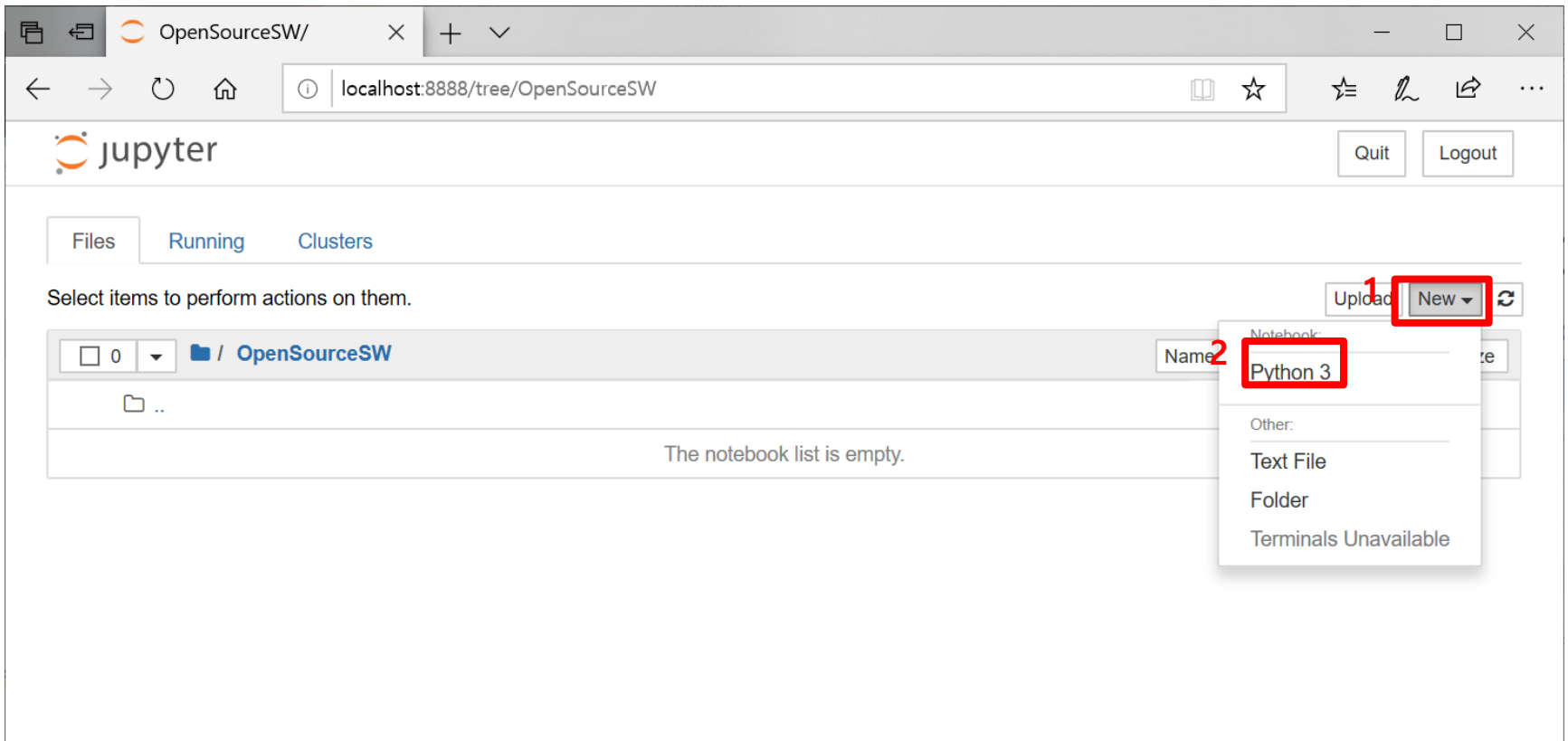
# Jupyter 폴더 생성



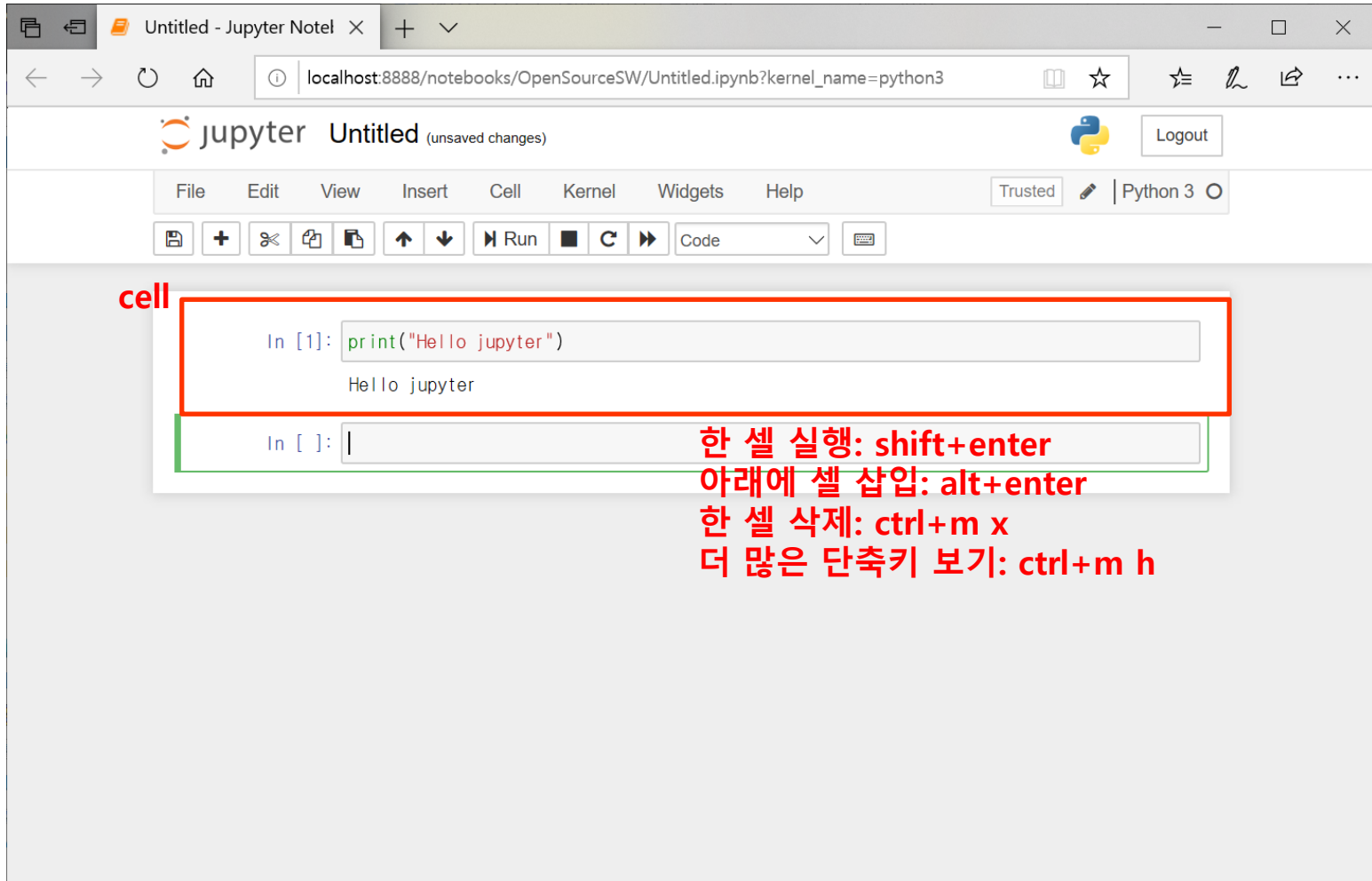
# Jupyter 폴더 이름 수정



# Jupyter로 python 실행



# Jupyter로 python 코딩하기



The screenshot shows the Jupyter Notebook interface in a web browser. The browser address bar shows the URL: `localhost:8888/notebooks/OpenSourceSW/Untitled.ipynb?kernel_name=python3`. The Jupyter interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, adding cells, and running code. The main area displays a code cell with the following content:

```
In [1]: print("Hello jupyter")
```

The output of the cell is displayed below the code:

```
Hello jupyter
```

Below the code cell, there is a text box for the next input:

```
In [ ]: |
```

Red text annotations provide keyboard shortcuts for Jupyter Notebook:

- 한 셀 실행: **shift+enter**
- 아래에 셀 삽입: **alt+enter**
- 한 셀 삭제: **ctrl+m x**
- 더 많은 단축키 보기: **ctrl+m h**



# Scikit-learn

- scikit-learn: 파이썬으로 작성된 데이터 분석을 위한 범용 오픈 소스 라이브러리

```
C:\> 명령 프롬프트
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\김학수>python -m pip install scikit-learn
Collecting scikit-learn
  Downloading scikit_learn-0.22.1-cp38-cp38-win32.whl (5.6 MB)
    | 5.6 MB 6.8 MB/s
Requirement already satisfied: numpy>=1.11.0 in c:\Users\김학수\AppData\Local\Programs\Python\Python38-32\Lib\site-packages (from scikit-learn) (1.18.1)
Requirement already satisfied: scipy>=0.17.0 in c:\Users\김학수\AppData\Local\Programs\Python\Python38-32\Lib\site-packages (from scikit-learn) (1.4.1)
Collecting joblib>=0.11
  Downloading joblib-0.14.1-py2.py3-none-any.whl (294 kB)
    | 294 kB 89 kB/s
Installing collected packages: joblib, scikit-learn
Successfully installed joblib-0.14.1 scikit-learn-0.22.1
```

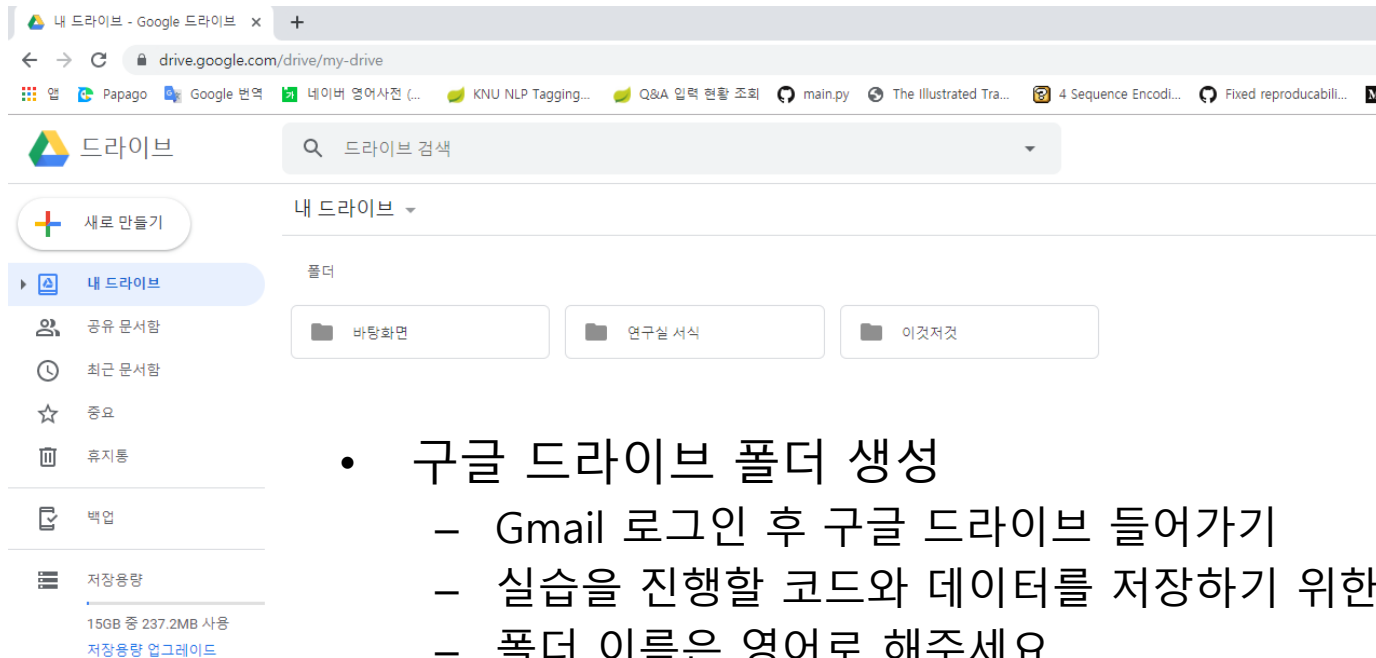
```
C:\Users\김학수>python
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help()" "copyright()" "credits()" or "license()" for more information.
>>> import sklearn
>>>
```

설치 확인



# Google Colab

- Google Colab: AI 개발자들을 위해 구글에서 제공하는 무료 클라우드 서비스



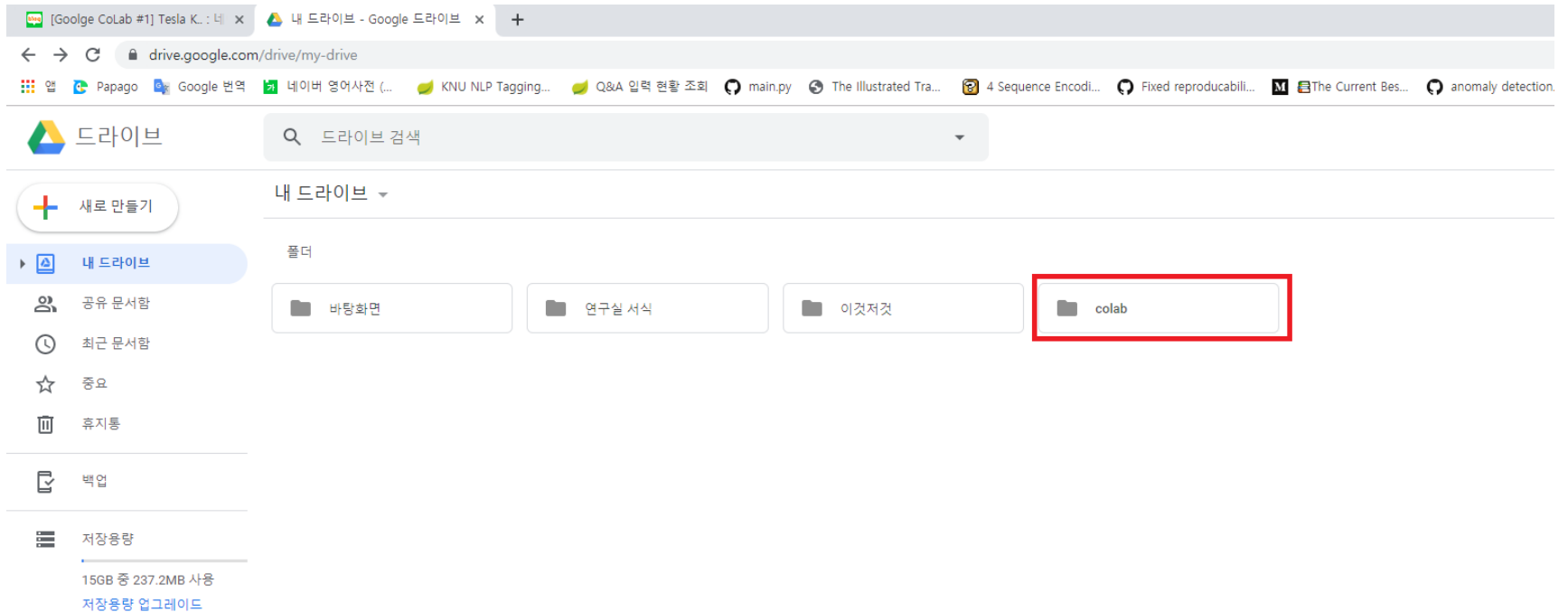
The screenshot shows the Google Drive web interface. The top navigation bar includes the Google Drive logo and a search bar. Below the navigation bar, the '내 드라이브' (My Drive) section is visible, showing a list of folders: '바탕화면' (Desktop), '연구실 서식' (Lab Format), and '이것저것' (Miscellaneous). The left sidebar contains various icons for file management, including '새로 만들기' (New), '공유 문서함' (Shared with me), '최근 문서함' (Recent), '중요' (Important), '휴지통' (Trash), '백업' (Backup), and '저장용량' (Storage).

- 구글 드라이브 폴더 생성
  - Gmail 로그인 후 구글 드라이브 들어가기
  - 실습을 진행할 코드와 데이터를 저장하기 위한 폴더 생성
  - 폴더 이름은 영어로 해주세요.





# Google Colab 설치



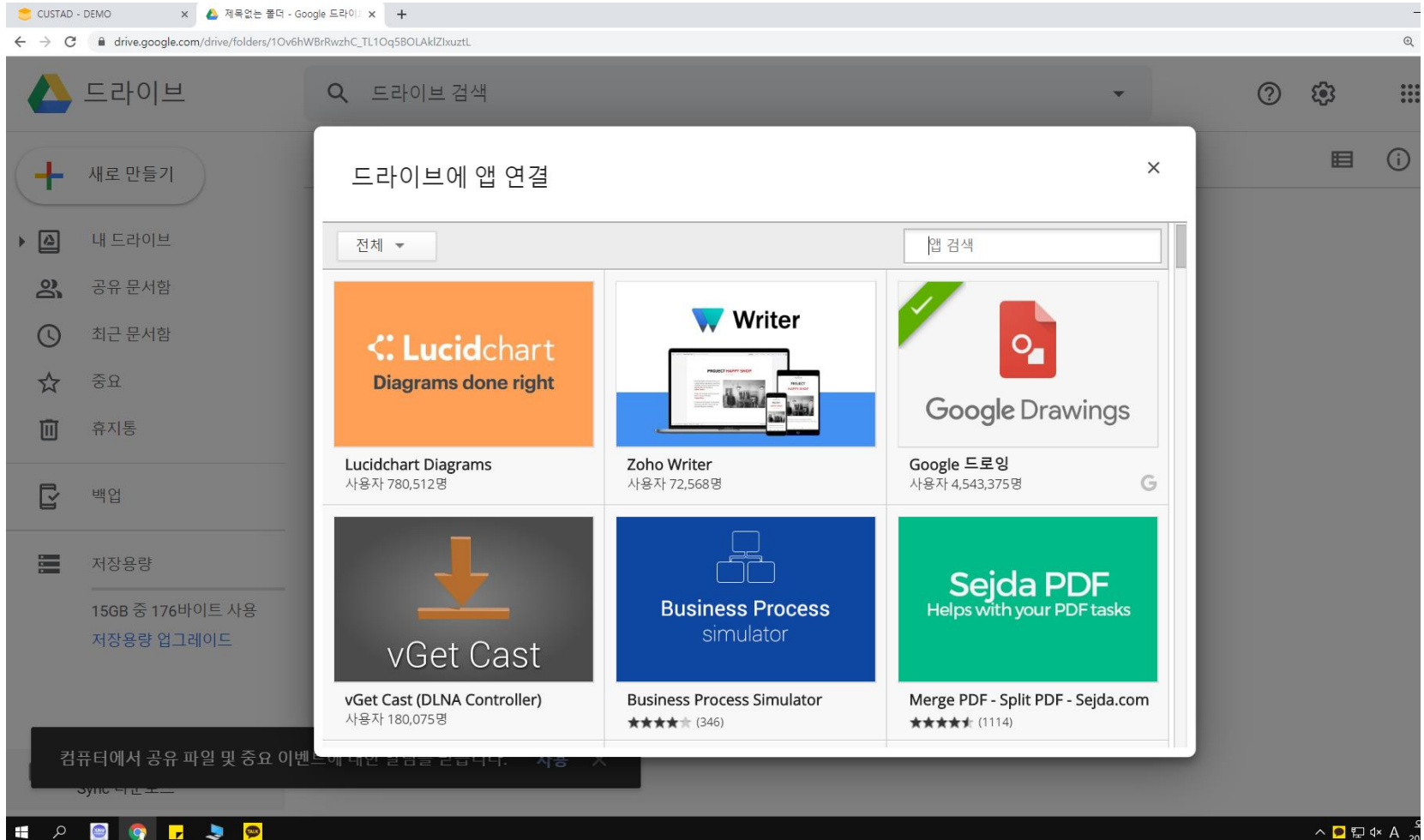
# Google Colab 설치

The screenshot shows the Google Drive web interface. The browser tabs include '[Google CoLab #1] Tesla K...', 'colab - Google 드라이브', and '+'. The address bar shows 'drive.google.com/drive/folders/1Q2y6BpFTii1CLJBmowg7APkHpqCRp1h'. The left sidebar shows '내 드라이브' (My Drive) and '공유 문서함' (Shared with me). The main area shows '내 드라이브 > colab' with a dropdown menu open. The dropdown menu includes '새 폴더' (New folder), '파일 업로드' (Upload file), '폴더 업로드' (Upload folder), 'Google 문서' (Google Docs), 'Google 스프레드시트' (Google Sheets), 'Google 프레젠테이션' (Google Slides), and '더보기' (More). The '더보기' menu is open, showing 'Google 설문지' (Google Forms), 'Google 드로잉' (Google Drawings), 'Google 내 지도' (Google My Maps), 'Google 사이트 도구' (Google Site Tools), 'Google Colaboratory' (highlighted with a red box), 'Google Jamboard', and '+ 연결할 앱 더보기' (See more apps to connect).

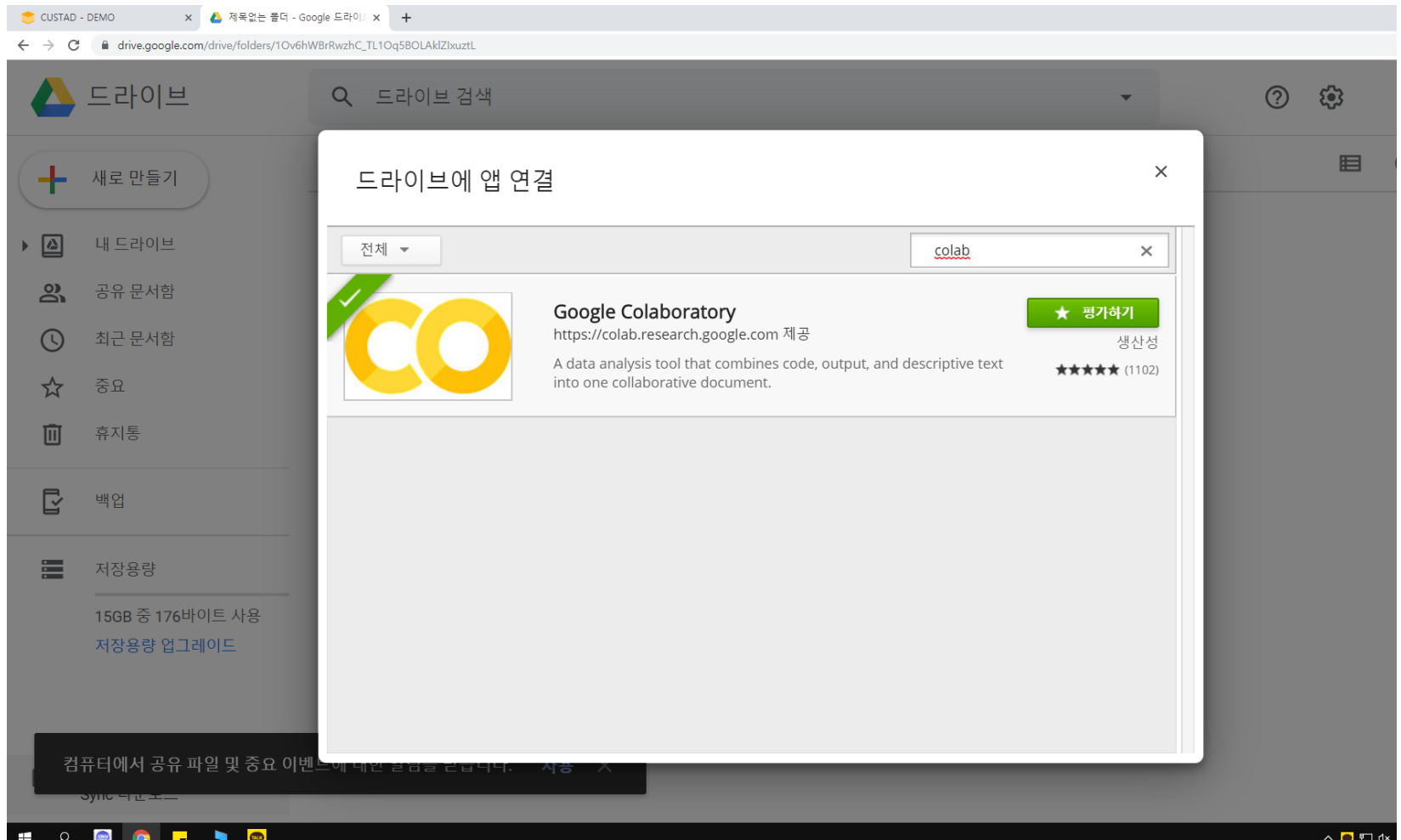
1. 실습 폴더 클릭
2. 마우스 우클릭 > 더보기 > Google Colaboratory 클릭
3. Google Colaboratory가 없는 경우
4. 연결할 앱 더 보기 > colab 검색 > 연결



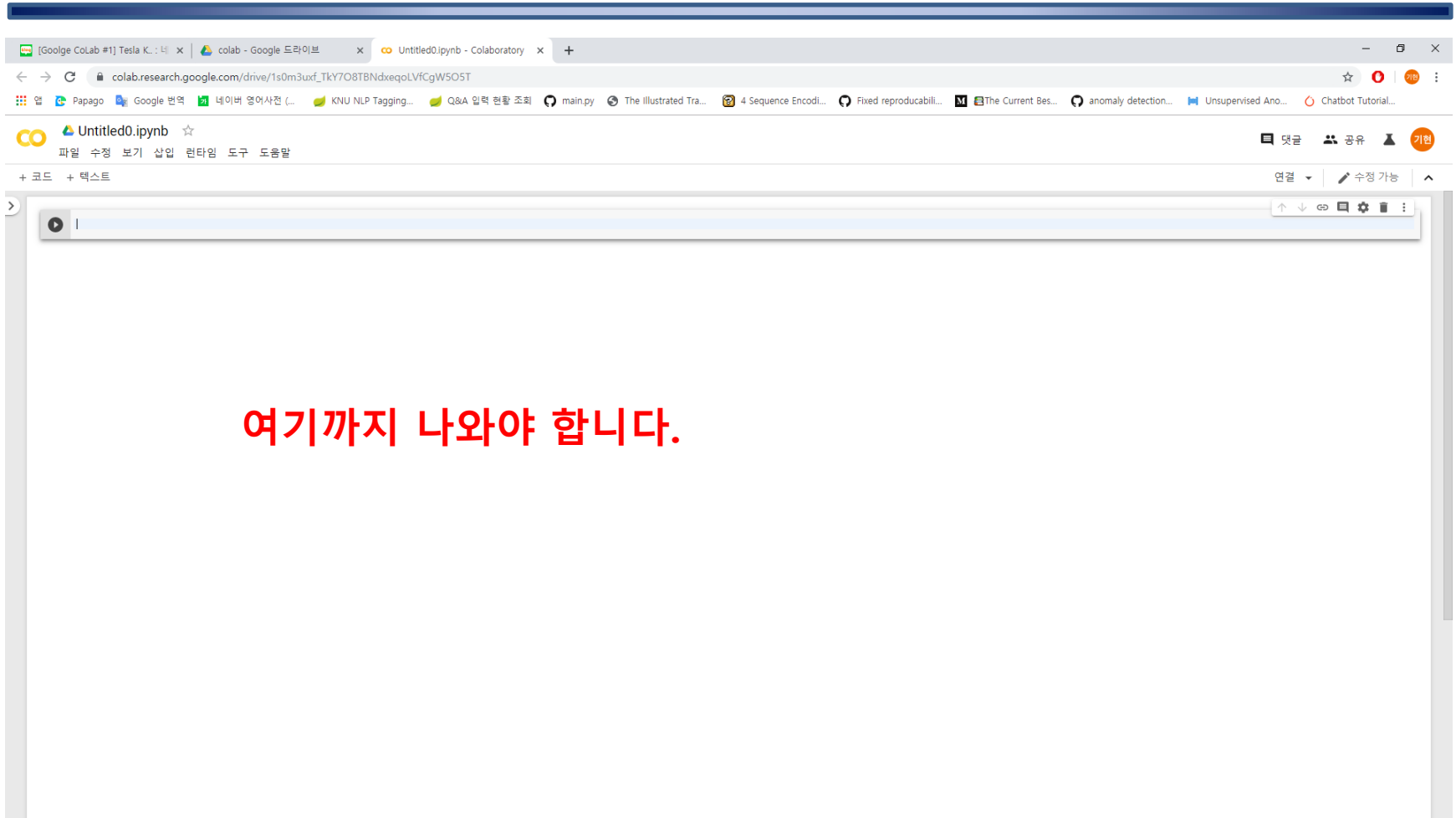
# Google Colab 설치



# Google Colab 설치



# Google Colab 설치



# 구글 드라이브 연동

---

- 개인의 구글 드라이브와 코랩 클라우드를 연동
  - 코랩 클라우드에 바로 파일을 업로드하면 일정 시간 후에 해당 파일이 삭제됨
  - 따라서 구글 드라이브에 파일을 업로드 하고 이를 코랩 클라우드와 연동하여 사용



# 구글 드라이브 연동

```
from google.colab import drive
drive.mount('/gdrive', force_remount=True)
```

위의 코드 입력 후 shift+enter 을 눌러 코드 실행

```
from google.colab import drive
drive.mount('/gdrive', force_remount=True)
```

... Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=947318989803](https://accounts.google.com/o/oauth2/auth?client_id=947318989803)

Enter your authorization code:

링크 선택 > 자신의 계정 선택 > 액세스 허용 > 인증 코드 복사하여 입력



# 구글 드라이브 연동

```
▶ from google.colab import drive
  drive.mount('/gdrive', force_remount=True)
```

Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=947318989803-6bn6gk8qdc](https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6gk8qdc)

Enter your authorization code:  
.....

Mounted at /gdrive

**“Mounted at /gdrive” 라는 메시지가 뜨면 연동 완료**





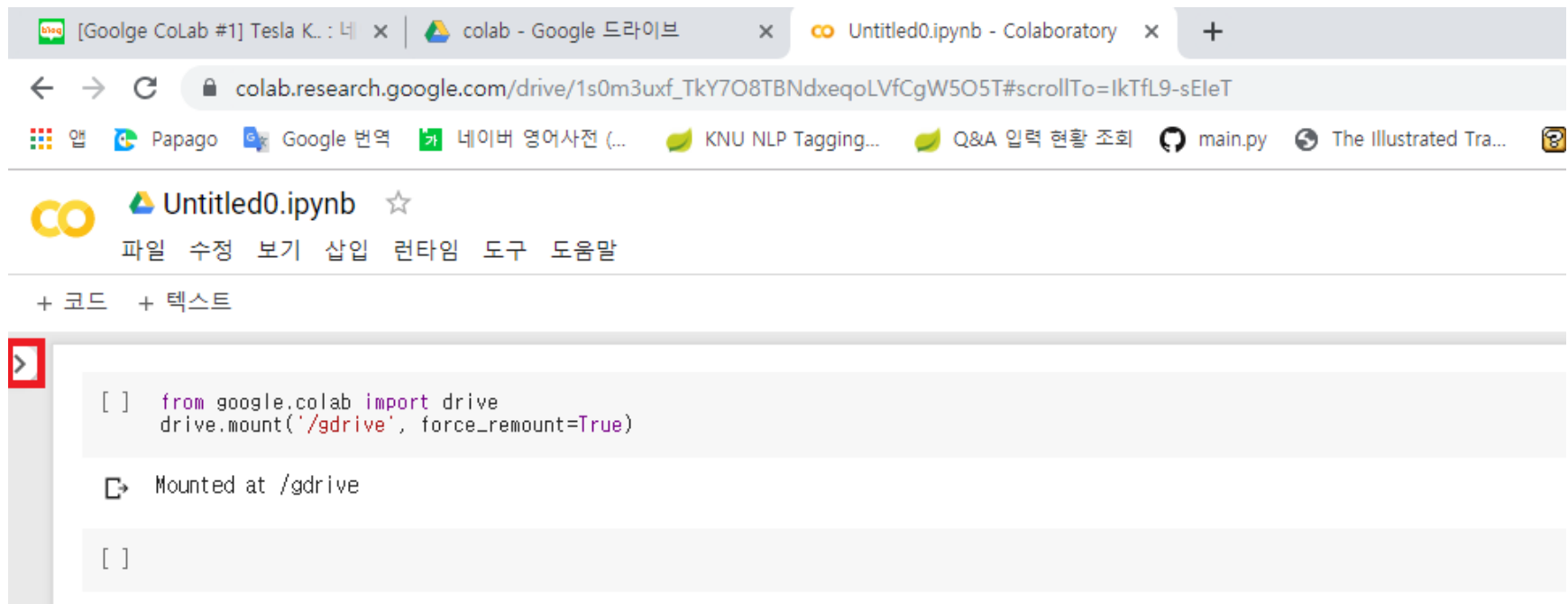
# 파일 업로드

The screenshot shows a Google Drive web interface. The browser tabs include '[Google CoLab #1] Tesla K...', 'colab - Google 드라이브', and '+'. The address bar shows 'drive.google.com/drive/folders/1Q2y6BpFTii1ICUBmowg7APkHpqCRp1h'. The left sidebar contains navigation options: '새로 만들기' (New), '내 드라이브' (My Drive), '공유 문서함' (Shared with me), '최근 문서함' (Recent), '중요' (Important), '휴지통' (Trash), '백업' (Backup), and '저장용량' (Storage) with a progress bar showing '15GB 중 237.3MB 사용' and a link to '저장용량 업그레이드' (Upgrade storage). The main area shows the 'colab' folder with a '파일' (Files) section. Two files are visible: '실습예제1.txt' (highlighted with a red box) and 'Untitled0.ipynb'.

실습 폴더에 “실습예제1.txt” 파일 업로드



# 파일 업로드



The screenshot shows a Google Colab notebook titled 'Untitled0.ipynb'. The browser address bar displays the URL: `colab.research.google.com/drive/1s0m3uxf_TkY7O8TBNdxepoLVfCgW5O5T#scrollTo=lkTfL9-sEleT`. The notebook interface includes a toolbar with options like '파일' (File), '수정' (Edit), '보기' (View), '삽입' (Insert), '런타임' (Runtime), '도구' (Tools), and '도움말' (Help). Below the toolbar, there are tabs for '+ 코드' (Code) and '+ 텍스트' (Text). The code cell contains the following Python code:

```
[ ] from google.colab import drive
    drive.mount('/gdrive', force_remount=True)
```

Below the code, a message indicates the successful mounting of the drive:

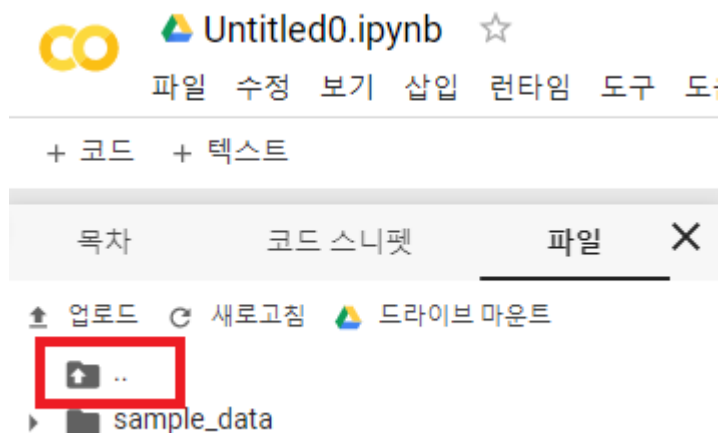
```
Mounted at /gdrive
```

The code cell is currently empty, showing only the prompt `[ ]`.

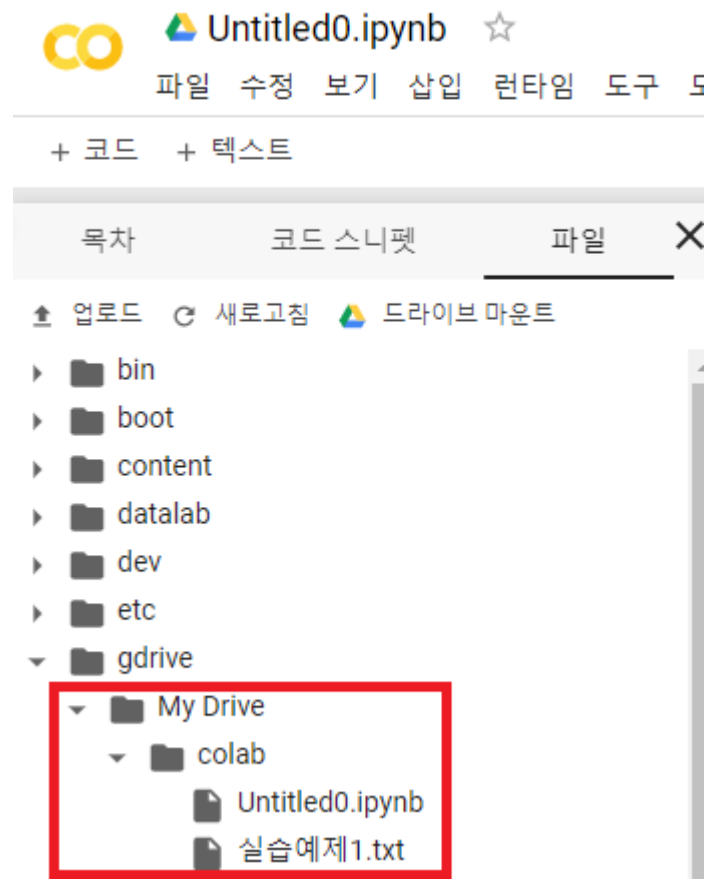
왼쪽 상단 화살표 클릭



# 파일 업로드



“..” 클릭



실습예제1.txt이 보이면 업로드 완료



# 업로드 한 파일 출력

```
[9] from google.colab import drive  
drive.mount('/gdrive', force_remount=True)
```

Mounted at /gdrive

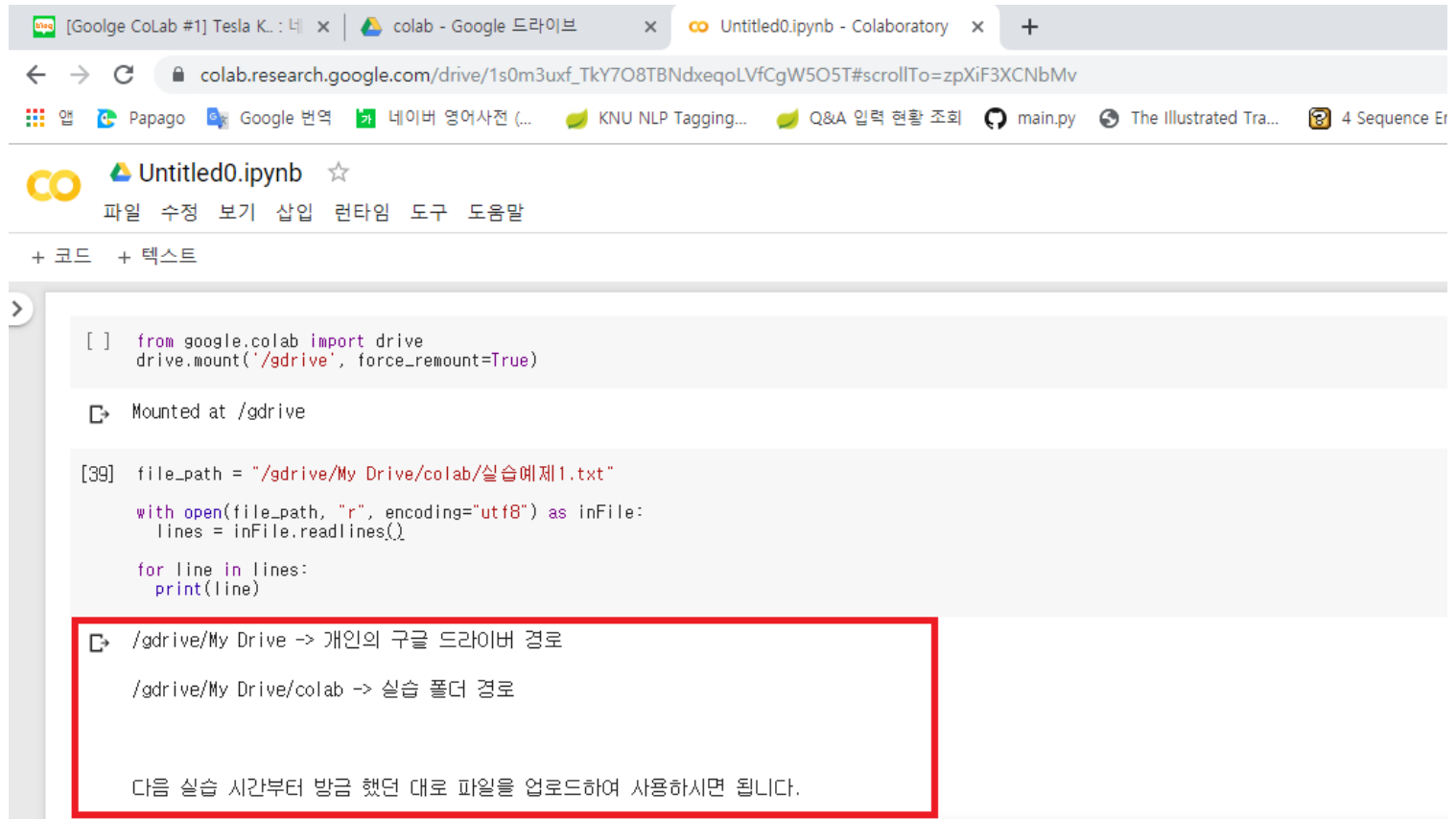
```
▶ file_path = "/gdrive/My Drive/colab/실습예제1.txt"  
with open(file_path, "r", encoding="utf8") as inFile:  
    lines = inFile.readlines()  
    |  
    for line in lines:  
        print(line)
```

본인이 생성한 실습 폴더 이름 입력

위의 코드 입력 후 코드 실행



# 업로드한 파일 출력



The screenshot shows a Google Colab notebook interface. The browser tabs at the top include '[Google CoLab #1] Tesla K...', 'colab - Google 드라이브', and 'Untitled0.ipynb - Colaboratory'. The address bar shows the URL 'colab.research.google.com/drive/1s0m3uxf\_TkY7O8TBNdxeqoLVfCgW5O5T#scrollTo=zxXiF3XCnbMv'. The notebook title is 'Untitled0.ipynb'. Below the title are tabs for '+ 코드' and '+ 텍스트'. The code cell contains the following Python code:

```
[ ] from google.colab import drive
drive.mount('/gdrive', force_remount=True)
```

Below the code, the output shows 'Mounted at /gdrive'. The next code cell contains:

```
[39] file_path = "/gdrive/My Drive/colab/실습예제1.txt"

with open(file_path, "r", encoding="utf8") as inFile:
    lines = inFile.readlines()

for line in lines:
    print(line)
```

The output of this cell is highlighted with a red box and contains the following text:

```
↳ /gdrive/My Drive -> 개인의 구글 드라이브 경로
/gdrive/My Drive/colab -> 실습 폴더 경로

다음 실습 시간부터 방금 했던 대로 파일을 업로드하여 사용하시면 됩니다.
```

**위 내용이 출력되면 성공!**



# Deep Neural Network

컴퓨터공학부 / 인공지능학과(대학원)

김학수

# XOR by PyTorch

```
# GPU 사용 가능 여부 확인
if torch.cuda.is_available():
    device = 'cuda'
else:
    device = 'cpu'

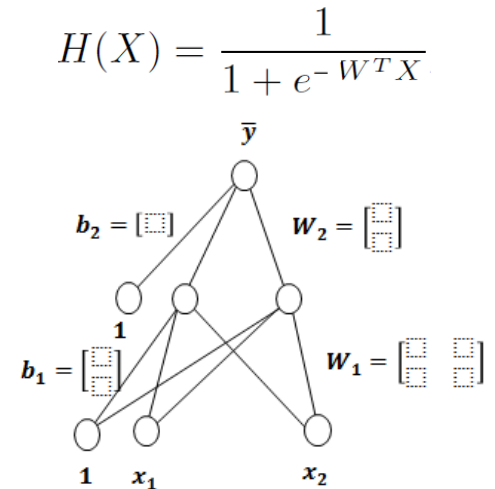
input_features, labels = load_dataset("/gdrive/My Drive/colab/ann/xor/train.txt", device)

# NN 모델 만들기
model = nn.Sequential(
    nn.Linear(2, 2, bias=True), nn.Sigmoid(),
    nn.Linear(2, 1, bias=True), nn.Sigmoid()).to(device)

# 이진분류 크로스엔트로피 비용 함수
loss_func = torch.nn.BCELoss().to(device)

# 옵티마이저 함수 (역전파 알고리즘을 수행할 함수)
optimizer = torch.optim.SGD(model.parameters(), lr=1)

# 학습 모드 셋팅
model.train()
```



$$\begin{aligned} \text{Cost}(w) \\ = -\frac{1}{m} \sum y \log(H(x)) + (1 - y) \log(1 - H(x)) \end{aligned}$$

$$W := W - \alpha \frac{\partial}{\partial W} \text{cost}(W)$$



# XOR by PyTorch

학습 실행 및 코스트 출력

```
# 모델 학습
for epoch in range(1001):

    # 기울기 계산한 것들 초기화
    optimizer.zero_grad()

    # H(X) 계산: forward 연산
    hypothesis = model(input_features)

    # 비용 계산
    cost = loss_func(hypothesis, labels)
    # 역전파 수행
    cost.backward()
    optimizer.step()

    # 100 에폭마다 비용 출력
    if epoch % 100 == 0:
        print(epoch, cost.item())
```

예측(0과 1로 변환) 및  
정밀도 계산

```
# 평가 모드 셋팅 (학습 시에 적용했던 드랍 아웃 여부 등을 비적용)
model.eval()

# 역전파를 적용하지 않도록 context manager 설정
with torch.no_grad():
    hypothesis = model(input_features)
    logits = (hypothesis > 0.5).float()
    predicts = tensor2list(logits)
    golds = tensor2list(labels)
    print("PRED=", predicts)
    print("GOLD=", golds)
    print("Accuracy : {0:f}".format(accuracy_score(golds, predicts)))
```

```
0 0.6988072395324707
100 0.693162202835083
200 0.6930413246154785
300 0.692793071269989
400 0.6919294595718384
500 0.6865977048873901
600 0.6376820802688599
700 0.5430010557174683
800 0.3161814212799072
900 0.0973285436630249
1000 0.051614370197057724
PRED= [[0.0], [1.0], [1.0], [0.0]]
GOLD= [[0.0], [1.0], [1.0], [0.0]]
Accuracy : 1.000000
```





# Wide ANN

Hidden layer를 2\*2에서 2\*10으로 변경

# NN 모델 만들기

?

Widening은 선의  
개수를 늘리는 효과!

더 빨리 수렴함 (학습 속도는 느려짐)

```
0 0.6988072395324707
100 0.693162202835083
200 0.6930413246154785
300 0.692793071269989
400 0.6919294595718384
500 0.6865977048873901
600 0.6376820802688599
700 0.5430010557174683
800 0.3161814212799072
900 0.0973285436630249
1000 0.051614370197057724
PRED= [[0.0], [1.0], [1.0], [0.0]]
GOLD= [[0.0], [1.0], [1.0], [0.0]]
Accuracy : 1.000000
```



```
0 0.7083522081375122
100 0.6920320987701416
200 0.6887232065200806
300 0.6461161971092224
400 0.3004415035247803
500 0.09012892842292786
600 0.04456526041030884
700 0.028334952890872955
800 0.020417045801877975
900 0.01582087203860283
1000 0.01284930482506752
PRED= [[0.0], [1.0], [1.0], [0.0]]
GOLD= [[0.0], [1.0], [1.0], [0.0]]
Accuracy : 1.000000
```



# Shallow ANN

Hidden layer를 없애고 Single-layer Perceptron으로 변경

```
# NN 모델 만들기  
model = nn.Sequential(  
    nn.Linear(2, 1, bias=True), nn.Sigmoid()), to(device)
```

Non-linear  
Separable Problem

학습 속도는 빠르지만 10,000 epoch를 수행해도 문제를 풀지 못함

```
0 0.7842277884483337  
1000 0.6931471824645996  
2000 0.6931471824645996  
3000 0.6931471824645996  
4000 0.6931471824645996  
5000 0.6931471824645996  
6000 0.6931471824645996  
7000 0.6931471824645996  
8000 0.6931471824645996  
9000 0.6931471824645996  
10000 0.6931471824645996  
PRED= [[0.0], [0.0], [0.0], [0.0]]  
GOLD= [[0.0], [1.0], [1.0], [0.0]]  
Accuracy : 0.500000
```



# Deep ANN

Hidden layer 층을 1개에서 2개로 변경

# NN 모델 만들기

?

Deeping은 선을  
구부리는 효과!

오래 걸리지만 학습됨

```
0 0.6953558921813965
100 0.6930767893791199
200 0.6930528283119202
300 0.693021297454834
400 0.6929775476455688
500 0.6929132342338562
600 0.6928118467330933
700 0.692637026309967
800 0.6922969818115234
900 0.6915085315704346
1000 0.6891056299209595
PRED= [[1.0], [0.0], [1.0], [0.0]]
GOLD= [[0.0], [1.0], [1.0], [0.0]]
Accuracy : 0.500000
```



```
0 0.7725627422332764
300 0.693056046962738
600 0.6929765939712524
900 0.69272780418396
1200 0.6910351514816284
1500 0.5805514454841614
1800 0.085045225918293
2100 0.014176958240568638
2400 0.0076338378712534904
2700 0.0052098375745117664
3000 0.003949393518269062
PRED= [[0.0], [1.0], [1.0], [0.0]]
GOLD= [[0.0], [1.0], [1.0], [0.0]]
Accuracy : 1.000000
```



# Deeper ANN

Hidden layer 층을 1개에서 7개로 변경

# NN 모델 만들기

```
model = nn.Sequential(  
    nn.Linear(2, 10, bias=True), nn.Sigmoid(),  
    nn.Linear(10, 10, bias=True), nn.Sigmoid(),  
    nn.Linear(10, 10, bias=True), nn.Sigmoid(),  
    nn.Linear(10, 10, bias=True), nn.Sigmoid(),  
    nn.Linear(10, 10, bias=True), nn.Sigmoid(),  
    nn.Linear(10, 10, bias=True), nn.Sigmoid(),  
    nn.Linear(10, 1, bias=True), nn.Sigmoid()).to(device)
```

WHY?

10,000 epoch를 돌려도 학습이 안됨!

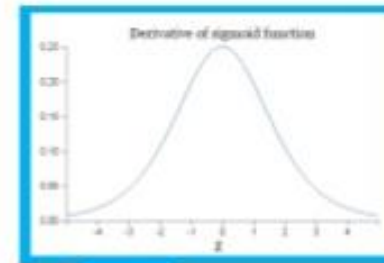
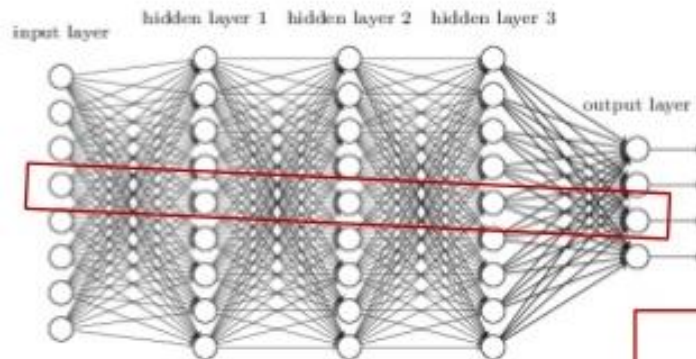
```
0 0.7086373567581177  
100 0.6931471824645996  
200 0.6931471824645996  
300 0.6931471824645996  
400 0.6931471824645996  
500 0.6931471824645996  
600 0.6931471824645996  
700 0.6931471824645996  
800 0.6931471824645996  
900 0.6931471824645996  
1000 0.6931472420692444  
PRED= [[1.0], [0.0], [0.0], [1.0]]  
GOLD= [[0.0], [1.0], [1.0], [0.0]]  
Accuracy : 0.000000
```



```
0 0.6933478713035583  
1000 0.6931472420692444  
2000 0.6931472420692444  
3000 0.6931471824645996  
4000 0.6931471824645996  
5000 0.6931471824645996  
6000 0.6931471824645996  
7000 0.6931471824645996  
8000 0.6931471824645996  
9000 0.6931471824645996  
10000 0.6931471824645996  
PRED= [[0.0], [1.0], [0.0], [0.0]]  
GOLD= [[0.0], [1.0], [1.0], [0.0]]  
Accuracy : 0.750000
```

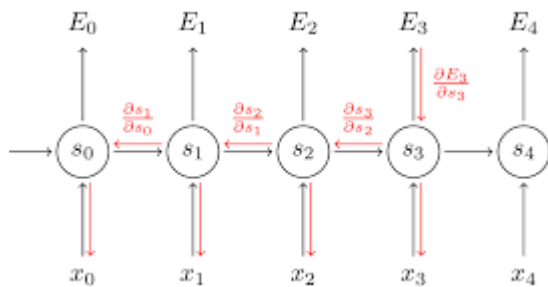


# Vanishing Gradient

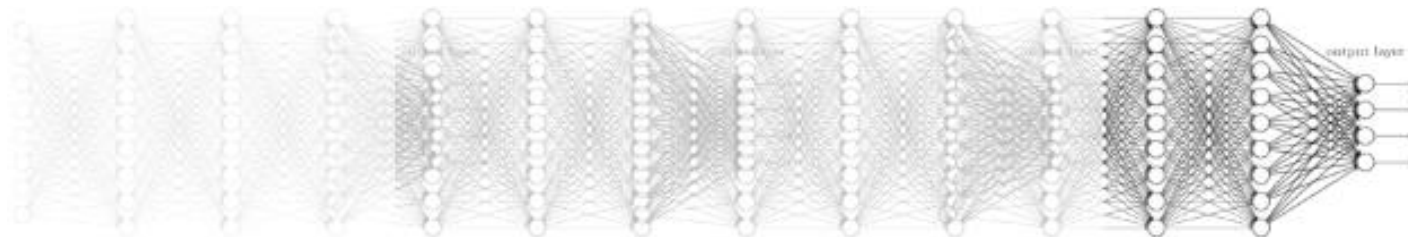


Sigmoid 함수에 의한  
back propagation

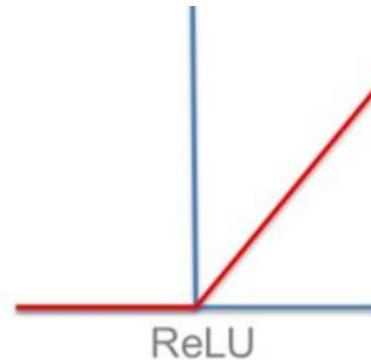
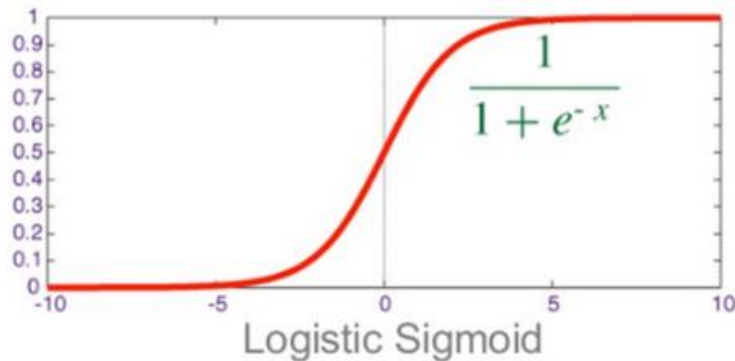
$$\frac{\partial C}{\partial w_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



NN Winter 2: 1985-2006



# Sigmoid to ReLU



Rectified Linear Unit

# NN 모델 만들기

?

Sigmoid를 ReLU로 변경

마지막 layer는 0과 1사이 값을 출력 하도록 하기 위해서 Sigmoid 유지

# Sigmoid to ReLU

Sigmoid (10,000 epoch)

```
0 0.6933478713035583
1000 0.6931472420692444
2000 0.6931472420692444
3000 0.6931471824645996
4000 0.6931471824645996
5000 0.6931471824645996
6000 0.6931471824645996
7000 0.6931471824645996
8000 0.6931471824645996
9000 0.6931471824645996
10000 0.6931471824645996
PRED= [[0.0], [1.0], [0.0], [0.0]]
GOLD= [[0.0], [1.0], [1.0], [0.0]]
Accuracy : 0.750000
```

vs.

ReLU (3,000 epoch)

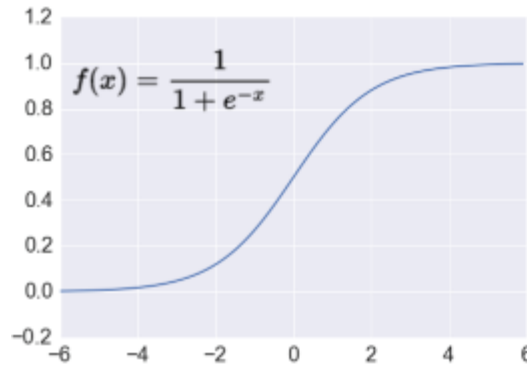
```
0 0.6946406960487366
300 0.6931304931640625
600 0.6931031942367554
900 0.6930624842643738
1200 0.6929516196250916
1500 0.6924918293952942
1800 0.6677674055099487
2100 0.0012112632393836975
2400 0.0003007405321113765
2700 0.00014935494982637465
3000 9.424101881450042e-05
PRED= [[0.0], [1.0], [1.0], [0.0]]
GOLD= [[0.0], [1.0], [1.0], [0.0]]
Accuracy : 1.000000
```

Vanishing  
gradient 문제가  
사라짐

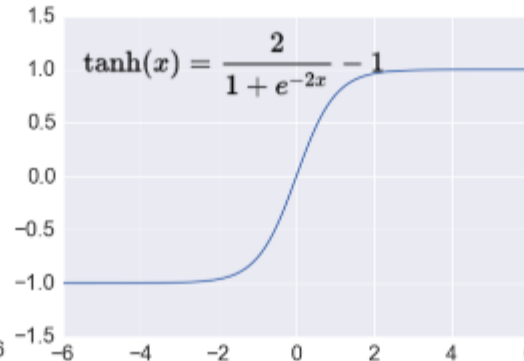


# Activation Functions

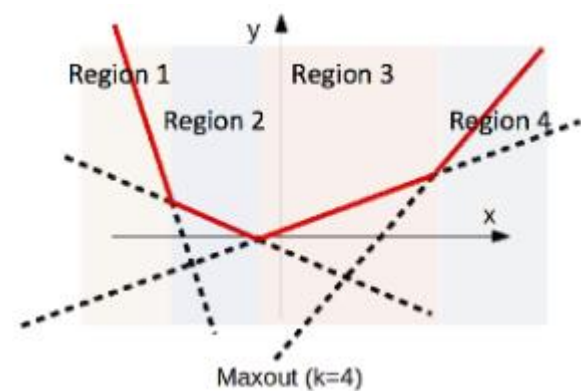
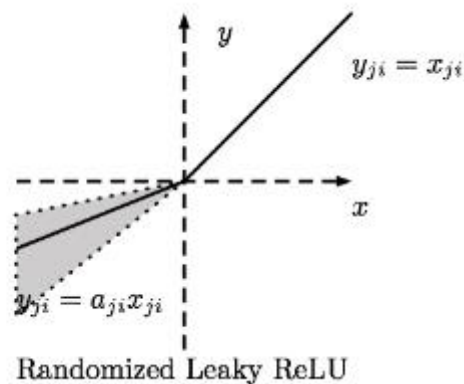
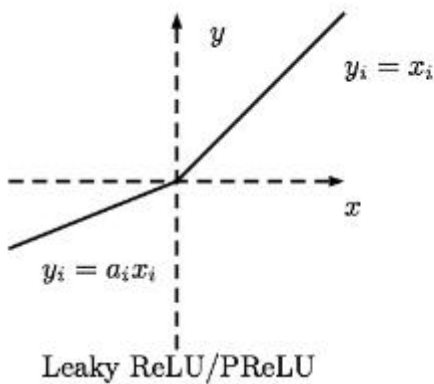
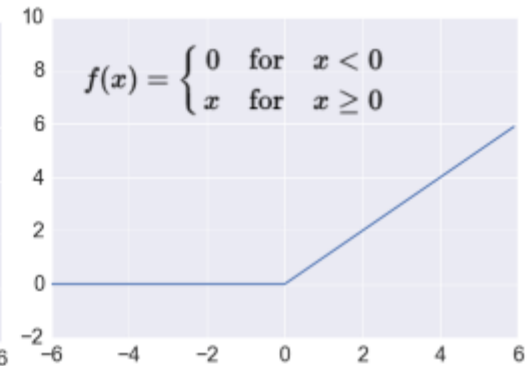
Sigmoid



Tanh



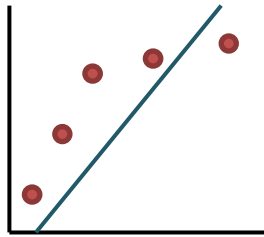
ReLU





# Fitting

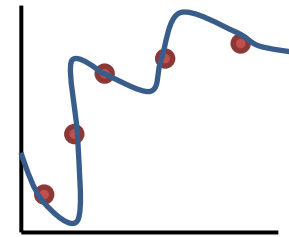
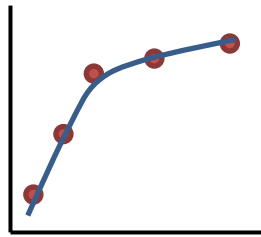
- 적합



Underfitting  
(high bias)



More training!



Overfitting  
(high variance)



More training data  
Reduce the number of features  
Regularization

# Regularization

- Regularization (정규화, 일반화)

- Cost function 값이 작아지는 방향으로 학습하는 과정에서 특정 가중치가 너무 커져서 일반화 성능이 떨어지는 것을 방지하기 위한 방법

- L1 regularization

$$D(S, L) = -\frac{1}{N} \sum_i L_i \log(S_i) + \lambda \sum |W|$$

Feature selection →  
Sparse Model에 적합

$$a = [0.1, 0.5, 0.2] \rightarrow \|a\| = 0.8$$

$$b = [0.3, 0.5, 0.0] \rightarrow \|b\| = 0.8$$

- L2 Regularization

$$D(S, L) = -\frac{1}{N} \sum_i L_i \log(S_i) + \lambda \sum W^2$$

Weight decay

$$a = [0.1, 0.5, 0.2] \rightarrow \|a\|_2 = 0.55$$

$$b = [0.3, 0.5, 0.0] \rightarrow \|b\|_2 = 0.58$$

- Early Stopping

- Dev set에서 성능이 더 이상 증가하지 않을 때 지정 횟수보다 학습을 일찍 끝마치는 것

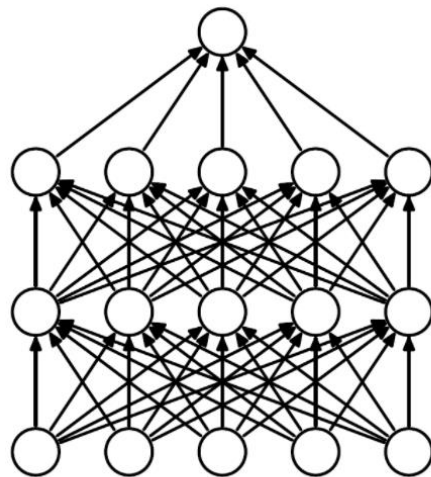
- Dropout



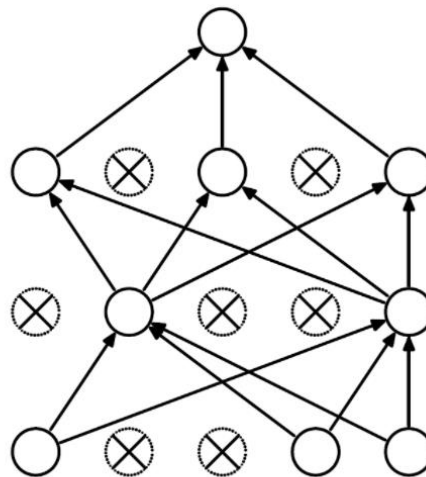
# Dropout

- 드롭아웃

- 학습 과정 중에 지정된 비율로 임의의 연결을 끊음으로써 일반화 성능을 개선하는 방법



(a) Standard Neural Net



(b) After applying dropout.

그림 출처: <https://medium.com/@gopalkalpande/biological-inspiration-of-convolutional-neural-network-cnn-9419668898ac>

# Dropout by PyTorch

# NN 모델 만들기

```
model = nn.Sequential(  
    nn.Linear(2, 10, bias=True), nn.ReLU(), nn.Dropout(0.1),  
    nn.Linear(10, 10, bias=True), nn.ReLU(), nn.Dropout(0.1),  
    nn.Linear(10, 10, bias=True), nn.ReLU(), nn.Dropout(0.1),  
    nn.Linear(10, 10, bias=True), nn.ReLU(), nn.Dropout(0.1),  
    nn.Linear(10, 10, bias=True), nn.ReLU(), nn.Dropout(0.1),  
    nn.Linear(10, 10, bias=True), nn.ReLU(), nn.Dropout(0.1),  
    nn.Linear(10, 1, bias=True), nn.Sigmoid()).to(device)
```

# 이진분류 크로스엔트로피 비용 함수

```
loss_func = torch.nn.BCELoss().to(device)
```

# 옵티마이저 함수 (역전파 알고리즘을 수행할 함수)

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.2)
```

# 학습 모드 셋팅

```
model.train()
```

# 평가 모드 셋팅 (학습 시에 적용했던 드랍 아웃 여부 등을 비적용)

```
model.eval()
```

# 역전파를 적용하지 않도록 context manager 설정

```
with torch.no_grad():
```

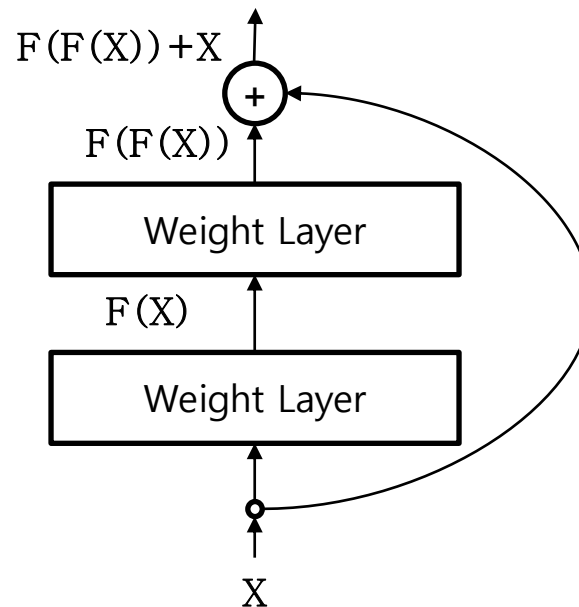
```
    hypothesis = model(input_features)
```

```
DATA= [[0. 0. 0.]  
 [0. 1. 1.]  
 [1. 0. 1.]  
 [1. 1. 0.]]  
INPUT_FEATURES= [[0. 0.]  
 [0. 1.]  
 [1. 0.]  
 [1. 1.]]  
LABELS= [[0.]  
 [1.]  
 [1.]  
 [0.]]  
0 0.7056415677070618  
300 0.6939449310302734  
600 0.6798115968704224  
900 0.6760240197181702  
1200 0.7288740873336792  
1500 0.5377387404441833  
1800 0.4900955557823181  
2100 0.00430224509909749  
2400 0.2083825021982193  
2700 0.6571221351623535  
3000 0.0671871230006218  
PRED= [[0.0], [1.0], [1.0], [0.0]]  
GOLD= [[0.0], [1.0], [1.0], [0.0]]  
Accuracy : 1.000000
```



# Residual Connection

- 추가 연결
  - 가중치층을 우회하여 상위 층으로 직접 연결하는 것
  - 추상화 정도(낮은 수준 추상화와 높은 수준 추상화)를 적절히 섞어주는 효과 → 앙상블 효과를 통해 성능 개선



# ANN Programming with Class

```
import os
import numpy as np
from sklearn.metrics import accuracy_score
import torch
import torch.nn as nn
from torch.utils.data import (DataLoader, RandomSampler, TensorDataset)
```

## Configure in main

```
config = {"mode": "test",
          "model_name": "epoch_{0:d}.pt".format(1000),
          "output_dir": output_dir,
          "input_data": input_data,
          "input_node": 2,
          "hidden_node": 10,
          "output_node": 1,
          "learn_rate": 1,
          "batch_size": 4,
          "epoch": 1000,
          }
```

Hypothesis  
만들기

```
class XOR(nn.Module):
```

```
def __init__(self, config):
    super(XOR, self).__init__()
```

```
# 입력층 노드 수
self.inode = config["input_node"]
# 은닉층 데이터 크기
self.hnode = config["hidden_node"]
# 출력층 노드 수: 분류해야 하는 레이블 수
self.onode = config["output_node"]
```

```
# 활성화 함수로 Sigmoid 사용
self.activation = nn.Sigmoid()
```

```
# 신경망 설계
self.linear1 = nn.Linear(self.inode, self.hnode, bias=True)
self.linear2 = nn.Linear(self.hnode, self.onode, bias=True)
```

```
def forward(self, input_features):
```

```
    output1 = self.linear1(input_features)
    hypothesis1 = self.activation(output1)
```

```
    output2 = self.linear2(hypothesis1)
    hypothesis2 = self.activation(output2)
```

```
    return hypothesis2
```

XOR Class



# ANN Programming with Class

## Training 함수

```
# 모델 학습 함수
def train(config):
```

```
    # 모델 생성
```

```
    model = XOR(config).cuda()
```

```
    # 데이터 읽기
```

```
    (input_features, labels) = load_dataset(config["input_data"])
```

```
    # TensorDataset/DataLoader를 통해 배치(batch) 단위로 데이터를 나누고 셔플(shuffle)
```

```
    train_features = TensorDataset(input_features, labels)
```

```
    train_dataloader = DataLoader(train_features, shuffle=True, batch_size=config["batch_size"])
```

```
    # 이진분류 크로스엔트로피 비용 함수
```

```
    loss_func = nn.BCELoss()
```

```
    # 옵티마이저 함수 (역전파 알고리즘을 수행할 함수)
```

```
    optimizer = torch.optim.SGD(model.parameters(), lr=config["learn_rate"])
```

```
# 데이터 읽기 함수
```

```
def load_dataset(file):
```

```
    data = np.loadtxt(file)
```

```
    print("DATA=", data)
```

```
    input_features = data[:,0:-1]
```

```
    print("INPUT_FEATURES=", input_features)
```

```
    labels = np.reshape(data[:, -1], (4, 1))
```

```
    print("LABELS=", labels)
```

```
    input_features = torch.tensor(input_features, dtype=torch.float)
```

```
    labels = torch.tensor(labels, dtype=torch.float)
```

```
    return (input_features, labels)
```



# ANN Programming with Class

```
for epoch in range(config["epoch"]+1):

    # 학습 모드 셋팅
    model.train()

    # epoch 마다 평균 비용을 저장하기 위한 리스트
    costs = []

    for (step, batch) in enumerate(train_dataloader):

        # batch = (input_features[step], labels[step])*batch_size
        # .cuda()를 통해 메모리에 업로드
        batch = tuple(t.cuda() for t in batch)

        # 각 feature 저장
        input_features, labels = batch

        # 역전파 변화도 초기화
        # .backward() 호출 시, 변화도 버퍼에 데이터가 계속 누적한 것을 초기화
        optimizer.zero_grad()

        # H(X) 계산: forward 연산
        hypothesis = model(input_features)
        # 비용 계산
        cost = loss_func(hypothesis, labels)
        # 역전파 수행
        cost.backward()
        optimizer.step()

    # 현재 batch의 스텝 별 loss 저장
    costs.append(cost.data.item())

    # 100 에폭마다 평균 loss 출력하고 모델을 저장
    if epoch%100 == 0:
        print("Average Loss= {0:f}".format(np.mean(costs)))
        torch.save(model.state_dict(), os.path.join(config["output_dir"], "epoch_{0:d}.pt".format(epoch)))
        do_test(model, train_dataloader)
```





# ANN Programming with Class

## Test 함수

```
# 모델 평가 함수
def test(config):

    model = XOR(config).cuda()

    # 저장된 모델 가중치 로드
    model.load_state_dict(torch.load(os.path.join(config["output_dir"], config["model_name"])))

    # 데이터 로드
    (features, labels) = load_dataset(config["input_data"])

    test_features = TensorDataset(features, labels)
    test_dataloader = DataLoader(test_features, shuffle=True, batch_size=config["batch_size"])

    do_test(model, test_dataloader)
```

```
# 모델 평가 결과 계산을 위해 텐서를 리스트로 변환하는 함수
def tensor2list(input_tensor):
    return input_tensor.cpu().detach().numpy().tolist()

# 평가 수행 함수
def do_test(model, test_dataloader):

    # 평가 모드 셋팅
    model.eval()

    # Batch 별로 예측값과 정답을 저장할 리스트 초기화
    predicts, golds = [], []

    with torch.no_grad():

        for step, batch in enumerate(test_dataloader):

            # .cuda()를 통해 메모리에 업로드
            batch = tuple(t.cuda() for t in batch)

            input_features, labels = batch
            hypothesis = model(input_features)
            logits = (hypothesis > 0.5).float()
            x = tensor2list(logits)
            y = tensor2list(labels)

            # 예측값과 정답을 리스트에 추가
            predicts.extend(x)
            golds.extend(y)

    print("PRED=", predicts)
    print("GOLD=", golds)
    print("Accuracy= {0:f}%".format(accuracy_score(golds, predicts)))
```



# ANN Programming with Class

## Training in Main

```
if(__name__=="__main__"):

    root_dir = "/gdrive/My Drive/colab/ann/xor"
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    input_data = "{0:s}/{1:s}".format(root_dir,"train.txt")

    config = {"mode": "train",
              "model_name": "epoch_{0:d}.pt".format(1000),
              "output_dir": output_dir,
              "input_data": input_data,
              "input_node": 2,
              "hidden_node": 10,
              "output_node": 1,
              "learn_rate": 1,
              "batch_size": 4,
              "epoch": 1000,
              }

    if(config["mode"] == "train"):
        train(config)
    else:
        test(config)
```

```
DATA= [[0. 0. 0.]
 [0. 1. 1.]
 [1. 0. 1.]
 [1. 1. 0.]]
INPUT_FEATURES= [[0. 0.]
 [0. 1.]
 [1. 0.]
 [1. 1.]]
LABELS= [[0.]
 [1.]
 [1.]
 [0.]]
Average Loss= 0.693318
PRED= [[0.0], [0.0], [1.0], [1.0]]
GOLD= [[1.0], [0.0], [0.0], [1.0]]
Accuracy= 0.500000

Average Loss= 0.690562
PRED= [[1.0], [1.0], [0.0], [0.0]]
GOLD= [[1.0], [0.0], [1.0], [0.0]]
Accuracy= 0.500000

Average Loss= 0.667773
PRED= [[0.0], [1.0], [0.0], [1.0]]
GOLD= [[1.0], [1.0], [0.0], [0.0]]
Accuracy= 0.500000

Average Loss= 0.433278
PRED= [[1.0], [0.0], [0.0], [1.0]]
GOLD= [[1.0], [0.0], [0.0], [1.0]]
Accuracy= 1.000000

Average Loss= 0.133441
PRED= [[0.0], [1.0], [0.0], [1.0]]
GOLD= [[0.0], [1.0], [1.0], [0.0]]
Accuracy= 1.000000
```

내 드라이브 > colab > ann > xor ▾

이름 ↑

📁	output
📄	train.txt
📄	xor.ipynb



내 드라이브 > ... > xor > output ▾

이름 ↑

📄	epoch_0.pt
📄	epoch_100.pt
📄	epoch_200.pt
📄	epoch_300.pt
📄	epoch_400.pt



# ANN Programming with Class

## Test in Main

```
if(__name__=="__main__"):

    root_dir = "/gdrive/My Drive/colab/ann/xor"
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    input_data = "{0:s}/{1:s}".format(root_dir, "train.txt")

    config = {"mode": "test",
              "model_name": "epoch_{0:d}.pt".format(1000),
              "output_dir": output_dir,
              "input_data": input_data,
              "input_node": 2,
              "hidden_node": 10,
              "output_node": 1,
              "learn_rate": 1,
              "batch_size": 4,
              "epoch": 1000,
              }

    if(config["mode"] == "train"):
        train(config)
    else:
        test(config)
```

1,000 epoch 모델 결과를  
읽어 들여 테스트 수행

```
DATA= [[0. 0. 0.]
        [0. 1. 1.]
        [1. 0. 1.]
        [1. 1. 0.]]
INPUT_FEATURES= [[0. 0.]
                  [0. 1.]
                  [1. 0.]
                  [1. 1.]]
LABELS= [[0.]
          [1.]
          [1.]
          [0.]]
PRED= [[0.0], [1.0], [1.0], [0.0]]
GOLD= [[0.0], [1.0], [1.0], [0.0]]
Accuracy= 1.000000
```



---

# Simple ML Programming

---

Practical Exercise

# 실습: Spam Mail Filter

- SMSSpamCollection 데이터를 입력으로 하는 SVM 기반 스팸 메일 필터링 프로그램을 작성하시오.
  - SMSSpamCollection 데이터 형식
    - ham/spam ₩t 문장

예제 코드 다운로드:  
<https://github.com/KUNLP/KTAI-Practice>

```
ham Go until jurong point, crazy.. Available only in bugis
ham Ok lar... Joking wif u oni...
spam    Free entry in 2 a wkly comp to win FA Cup final tkt
ham U dun say so early hor... U c already then say...
ham Nah I don't think he goes to usf, he lives around here
spam    FreeMsg Hey there darling it's been 3 week's now an
ham Even my brother is not like to speak with me. They trea
ham As per your request 'Melle Melle (Oru Minnaminunginte N
spam    WINNER!! As a valued network customer you have been
spam    Had your mobile 11 months or more? U R entitled to
ham I'm gonna be home soon and i don't want to talk about t
spam    SIX chances to win CASH! From 100 to 20,000 pounds
```



# 실습: Spam Mail Filter

## 구글 colab 연결

```
from google.colab import drive
drive.mount("/gdrive", force_remount=True)
```

## 데이터 읽기 (처음부터 100개)

```
import numpy as np

file_path = "/gdrive/My Drive/colab/svm/SMSSpamCollection"

# 파일 읽기
x_data, y_data = [], []
with open(file_path, 'r', encoding='utf8') as inFile:
    lines = inFile.readlines()

lines = lines[:100]

for line in lines:
    line = line.strip().split(' ')
    sentence, label = line[1], line[0]
    x_data.append(sentence)
    y_data.append(label)

print("x_data의 개수 : " + str(len(x_data)))
print("y_data의 개수 : " + str(len(y_data)))
```

```
x_data의 개수 : 100
y_data의 개수 : 100
```



# 실습: Spam Mail Filter

## 데이터 변환 (문자열 → 숫자)

```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer()

# spam, ham 라벨을 대응하는 index로 치환하기 위한 딕셔너리
label2index_dict = {'spam':0, 'ham':1}

# indexing 한 데이터를 넣을 리스트 선언
indexing_x_data, indexing_y_data = [], []

for label in y_data:
    indexing_y_data.append(label2index_dict[label])

# x_data를 사용하여 딕셔너리 생성
tokenizer.fit_on_texts(x_data)

# x_data에 있는 각 문장의 단어들을 대응하는 index로 치환하고 그 결과값을 indexing_x_data에 저장
indexing_x_data = tokenizer.texts_to_sequences(x_data)

print("x_data indexing 하기 전 : " + str(x_data[0]))
print("x_data indexing 하기 후 : " + str(indexing_x_data[0]))
print("y_data indexing 하기 전 : " + str(y_data[0]))
print("y_data indexing 하기 후 : " + str(indexing_y_data[0]))
```

```
x_data indexing 하기 전 : Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
x_data indexing 하기 후 : [38, 93, 239, 240, 241, 242, 53, 11, 243, 72, 94, 244, 245, 126, 246, 247, 73, 74, 248, 127]
y_data indexing 하기 전 : ham
y_data indexing 하기 후 : 1
```



# 실습: Spam Mail Filter

## SVM 학습

```
from sklearn.svm import SVC

# 문자의 길이를 max_length로 맞춰 변환
max_length = 60
for index in range(len(indexing_x_data)):
    length = len(indexing_x_data[index])

    if(length > max_length):
        indexing_x_data[index] = indexing_x_data[index][:max_length]
    elif(length < max_length):
        indexing_x_data[index] = indexing_x_data[index] + [0]*(max_length-length)

# 전체 데이터를 9:1의 비율로 나누어 학습 및 평가 데이터로 사용
number_of_train = int(len(indexing_x_data)*0.9)

train_x = indexing_x_data[:number_of_train]
train_y = indexing_y_data[:number_of_train]
test_x = indexing_x_data[number_of_train:]
test_y = indexing_y_data[number_of_train:]

print("train_x의 개수 : " + str(len(train_x)))
print("train_y의 개수 : " + str(len(train_y)))
print("test_x의 개수 : " + str(len(test_x)))
print("test_y의 개수 : " + str(len(test_y)))

svm = SVC(kernel='linear', C=1e10)
svm.fit(train_x, train_y)
```

Soft Margin 슬랙의 C 값!

$$\text{minimize: } Q_1(w, b, \xi_i) = \frac{1}{2} \|w\|^2 + C \sum_i \xi_i$$

$$\text{subject to: } y_i(w \cdot x_i - b) \geq 1 - \xi_i, \quad \forall (x_i, y_i) \in D$$
$$\xi_i \geq 0$$

```
train_x의 개수 : 90
train_y의 개수 : 90
test_x의 개수 : 10
test_y의 개수 : 10
SVC(C=10000000000.0, break_ties=False, cache_size=200, class_weight=None,
    coef0=0.0, decision_function_shape='ovr', degree=3, gamma='scale',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```





# 실습: Spam Mail Filter

## SVM 평가

```
predict = svm.predict(test_x)

correct_count = 0
for index in range(len(predict)):
    if(test_y[index] == predict[index]):
        correct_count += 1

accuracy = 100.0*correct_count/len(test_y)

print("Accuracy: " + str(accuracy))

index2label = {0:"spam", 1:"ham"}

test_x_word = tokenizer.sequences_to_texts(test_x)

for index in range(len(test_x_word)):
    print()
    print("문장 : ", test_x_word[index])
    print("정답 : ", index2label[test_y[index]])
    print("모델 출력 : ", index2label[predict[index]])
```

Accuracy: 80.0

문장 : yeah do don 't stand to close tho you 'll catch so  
정답 : ham  
모델 출력 : spam

문장 : sorry to be a pain is it ok if we meet another nig  
정답 : ham  
모델 출력 : spam

문장 : smile in pleasure smile in pain smile when trouble  
정답 : ham  
모델 출력 : ham

문장 : please call our customer service representative on  
정답 : spam  
모델 출력 : spam

문장 : havent planning to buy later i check already lido  
정답 : ham  
모델 출력 : ham

문장 : your free ringtone is waiting to be collected simp  
정답 : spam  
모델 출력 : spam



# 과제

- 영어 불용어 리스트를 참고하여 입력 문장에서 불용어를 제거한 SVM 기반 스팸 메일 필터를 구현하시오. (10점)
  - 불용어 리스트를 얻는 방법

```
import nltk
```

```
print('영어 불용어 갯수:', len(nltk.corpus.stopwords.words('english')))  
print(nltk.corpus.stopwords.words('english')[:10])
```

영어 불용어 갯수: 179

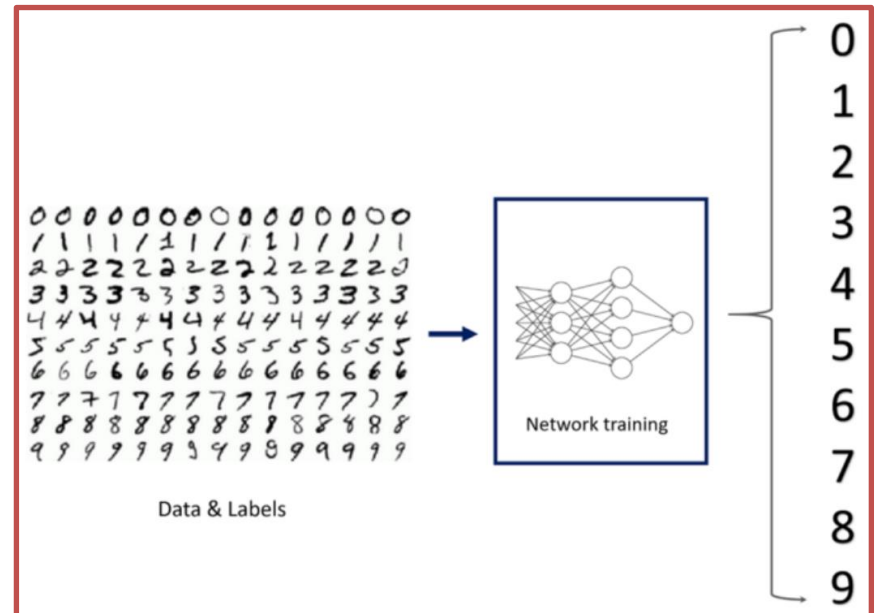
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]



# 실습: MNIST

예제 코드 다운로드:  
<https://github.com/KUNLP/KTAI-Practice>

- XOR 문제에 사용한 “ANN Class in PyTorch” 프로그램을 수정하여 손 글씨 숫자를 판독하는 프로그램을 작성하시오.
  - 입력 데이터셋
    - MNIST dataset
    - 0~9 손 글씨 이미지에 대한 픽셀 값 데이터
  - 문제
    - 이미지의 픽셀 값을 입력으로 하여 해당 이미지가 0~9 중에 어떤 숫자인지 분류



# 실습: MNIST

```
class MNIST(nn.Module):
```

## MNIST Class

```
def __init__(self, config):
```

?

```
# 입력층 노드 수
self.inode = config["input_node"]
# 은닉층 데이터 크기
self.hnode = config["hidden_node"]
# 출력층 노드 수: 분류해야 하는 레이블 수
self.onode = config["output_node"]
```

```
# 활성화 함수로 Sigmoid 사용
self.activation = nn.Sigmoid()
```

```
# 신경망 설계
self.linear1 = nn.Linear(self.inode, self.hnode, bias=True)
self.linear2 = nn.Linear(self.hnode, self.onode, bias=True)
```

```
def forward(self, input_features):
```

```
output1 = self.linear1(input_features)
hypothesis1 = self.activation(output1)
```

```
output2 = self.linear2(hypothesis1)
hypothesis2 = self.activation(output2)
```

```
return hypothesis2
```

```
import os
import numpy as np
from sklearn.metrics import accuracy_score
import torch
import torch.nn as nn
from torch.utils.data import (DataLoader, RandomSampler, TensorDataset)
from keras.datasets import mnist
```

# 데이터 읽기 함수

```
def load_dataset():
```

데이터 읽기

?(keras.datasets)

```
train_X = train_X.reshape(-1, 28*28)
print(train_X.shape)
test_X = test_X.reshape(-1, 28*28)
```

```
train_X = torch.tensor(train_X, dtype=torch.float)
train_y = torch.tensor(train_y, dtype=torch.long)
test_X = torch.tensor(test_X, dtype=torch.float)
test_y = torch.tensor(test_y, dtype=torch.long)
```

```
return (train_X, train_y), (test_X, test_y)
```



# 실습: MNIST

# 모델 평가 함수

def test(config):

Test 함수

?

# 저장된 모델 가중치 로드

model.load\_state\_dict(torch.load(os.path.join(config["output\_dir"], config["model\_name"])))

?

test\_features = TensorDataset(features, labels)

test\_dataloader = DataLoader(test\_features, shuffle=True, batch\_size=config["batch\_size"])

do\_test(model, test\_dataloader)



# 실습: MNIST

# 모델 학습 함수  
def train(config):

Training 함수

?

# TensorDataset/DataLoader를 통해 배치(batch) 단위로 데이터를 나누고 셔플(shuffle)  
train\_features = TensorDataset(input\_features, labels)  
train\_dataloader = DataLoader(train\_features, shuffle=True, batch\_size=config["batch\_size"])

?

# 현재 batch의 스텝 별 loss 저장  
costs.append(cost.data.item())

# 에폭마다 평균 비용s 출력하고 모델을 저장  
print("Average Loss= {0:f}".format(np.mean(costs)))  
torch.save(model.state\_dict(), os.path.join(config["output\_dir"], "epoch\_{0:d}.pt".format(epoch)))  
do\_test(model, train\_dataloader)



# 실습: MNIST

## Main

```
if(__name__=="__main__"):

    root_dir = "/gdrive/My Drive/colab/ann/mnist"
    output_dir = os.path.join(root_dir, "output")
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    config = {"mode": "train",
              "model_name": "epoch_{0:d}.pt".format(10),
              "output_dir": output_dir,
              "input_node": 784,
              "hidden_node": 512,
              "output_node": 10,
              "learn_rate": 0.001,
              "batch_size": 32,
              "epoch": 10,
              }

    if(config["mode"] == "train"):
        train(config)
    else:
        test(config)
```

## 과제

빈칸 채우기: 10점

```
Downloading data from https://storage.googleapis.com/tensorflow
11493376/11490434 [=====] - 0s 0us/steps
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
(60000, 784)
Average Loss= 1.620250
PRED= [3, 2, 1, 0, 4, 1, 4, 1, 9, 6, 1, 4, 5, 7, 5, 4, 7, 8, 4,
GOLD= [3, 2, 1, 0, 4, 1, 4, 1, 9, 6, 1, 4, 5, 7, 5, 4, 7, 2, 4,
Accuracy= 0.915850

Average Loss= 1.563620
PRED= [0, 8, 7, 9, 1, 7, 7, 6, 3, 6, 9, 3, 1, 6, 8, 7, 2, 1, 9,
GOLD= [0, 8, 7, 3, 1, 7, 7, 6, 3, 6, 9, 3, 1, 6, 8, 7, 2, 1, 9,
Accuracy= 0.918350

Average Loss= 1.551876
PRED= [9, 8, 8, 4, 3, 0, 5, 9, 0, 2, 1, 8, 7, 1, 1, 1, 8, 2, 9,
GOLD= [9, 8, 8, 4, 3, 6, 5, 9, 0, 2, 1, 8, 7, 1, 1, 1, 5, 2, 9,
Accuracy= 0.922050
```



# 질의응답

---

Q&A

Homepage: <http://nlp.konkuk.ac.kr>  
E-mail: [nlpdrkim@konkuk.ac.kr](mailto:nlpdrkim@konkuk.ac.kr)

