
Introduction to AI for Image Processing



황성태

ye0077@naver.com

https://github.com/HST0077/AI_intro_for_Imae

Python Basics



개발환경

Colab을 이용하여 쉽게 GPU를 이용한 파이썬 코딩을 시작할 수 있고, 전문적 사용을 위해서는 Amazon Web Service를 사용하면 된다.

The screenshot shows the Google Colab interface. At the top, there's a toolbar with icons for file, edit, view, insert, run, tools, and help. Below the toolbar, there are two tabs: '+ 코드' (Code) and '+ 텍스트' (Text). The code tab is active, showing a code cell with the following content:

```
[4] # 행렬의 표현 1
import numpy as np

A=np.array([[1,2],[3,4]])
A

[5] # 행렬의 표현 2
import sympy as sp

sp.Matrix(A)
```

On the left side, there are several icons: a CO icon, a search icon, a copy/paste icon, a file icon, and a refresh icon. A sidebar on the right shows a progress bar at 0% completion.



필요한 라이브러리 설치

파이썬은 여러가지 유용한 함수들의 집합인 라이브러리(패키지) 등을 무료로 설치하고, 쉽게 불러와서 사용할 수 있다.

```
# 필요한 라이브러리 설치  
!pip install sympy
```

```
Requirement already satisfied:  
Requirement already satisfied:
```

```
# 미분하기  
from sympy import diff, symbols  
  
x=symbols('x')  
diff(x**2,x)
```

```
2x
```

```
# 적분하기  
from sympy import integrate, symbols  
  
x=symbols('x')  
integrate(x**2,(x,0,3))
```

```
9
```

자료형: list, dictionary, tuple, set

```
# 리스트(list)
```

```
L=[1,2,3]
```

```
L
```

```
L[0],L[2]
```

```
(1, 3)
```

```
# 사전형(dictionary)
```

```
D={'A':'사과', 'B':'배'}
```

```
D['A']
```

```
'사과'
```

```
# 튜플형(tuple)
```

```
T=(1,2,3)
```

```
T[-1]
```

```
3
```

```
# 집합형(Set)
```

```
S1,S2={1,2},{2,3,4}
```

```
S1.intersection(S2)
```

```
{2}
```

행렬과 텐서

텐서(tensor)는 행렬을 확장한 개념이라고 보면 된다. 0차원 텐서는 스칼라, 1차원 텐서는 벡터, 2차원 텐서는 행렬이 된다.

```
# 행렬의 표현 1
import numpy as np
# numpy는 array라는 자료형 기반
A=np.array([[1,2],[3,4]])
A

array([[1, 2],
       [3, 4]])
```

```
# 행렬의 표현 2
import sympy as sp

sp.Matrix(A)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
# 3차원 텐서
import numpy as np

A=np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
A

array([[[1, 2],
       [3, 4]],
      [[5, 6],
       [7, 8]]])
```

자료형의 슬라이싱

```
# 슬라이싱이란 자료의 일부를 추출하는 것
# 인덱싱(indexing)은 특정 원소를 추출하는 것
L=[1,2,3,4,5]
print('처음 2개:',L[:2])
print('마지막 2개:',L[-2:])

처음 2개: [1, 2]
마지막 2개: [4, 5]
```

```
A=np.array([[1,2],[3,4],[5,6],[7,8]])
# 첫번째 행렬
print('첫번째 행렬:',A[0,:,:])
```

첫번째 행렬: [[1 2]
 [3 4]]

```
# (3,5) 텐서의 행,열의 합 구하기
import numpy as np
A=np.random.randint(1,10,(3,5))
print(A)
A.sum(axis=0) # 열의 합
```

[[7 8 1 8 8]
 [8 1 5 3 8]
 [7 3 6 8 4]]
array([22, 12, 12, 19, 20])

```
A.sum(axis=1) # 행의 합
```

array([32, 25, 28])

```
# (2,3,4) 텐서의 (2,3) 행렬 합 구하기
import numpy as np
A=np.random.randint(1,10,(2,3,4))
print(A)
A.sum(axis=2)
```

[[[4 2 4 3]
 [9 1 8 5]
 [8 5 4 8]]

[[1 9 4 1]
 [2 1 9 6]
 [7 7 4 9]]]
array([[13, 23, 25],
 [15, 18, 27]])

```
# 마지막 축의 합 구하기
A.sum(axis=(0,1))
```

array([31, 25, 33, 32])

반복문과 제어문

```
# 1부터 10까지 더하기  
sum=0  
for i in range(1,11):  
    sum+=i  
print(sum)
```

55

```
# 1부터 20까지의 짝수 더하기: 방법1  
sum=0  
for i in range(1,21):  
    if i%2==0: # 2로 나눈 나머지가 0이면  
        sum+=i  
print(sum)
```

110

```
# 1부터 20까지의 짝수 더하기: 방법2  
sum=0  
for i in range(0,21,2):  
    sum+=i  
print(sum)
```

사용자 정의 함수

```
# 'def' 를 사용하여 정의
def hst_sum(a,b):
    return a+b
```

```
hst_sum(10,1000)
```

```
1010
```

```
# 인라인(inline) 함수
# 'lambda'를 사용하여 정의
```

```
f=lambda a,b:a+b
f(10,1000)
```

```
1010
```

```
# 재귀함수(Recursive function)
def hst_factorial(n):
    if n==0:
        return 1
    else:
        return n*hst_factorial(n-1)
```

```
hst_factorial(5)
```

```
120
```

```
import sympy as sp
sp.factorial(5)
```

```
120
```

AI Basics

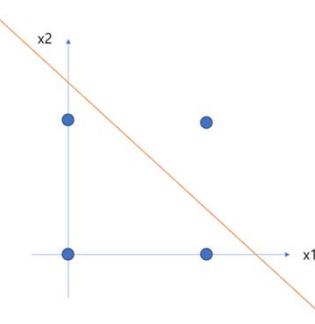


Perceptron

Q. (Classification 예제) 임의의 가중치 벡터 w_1, w_2 에 대해서 두 입력변수 x_1, x_2 를 값에 따라 y 의 값을 아래와 같이 출력하는 함수를 만들어 보아라.

x1	x2	y
0	0	0
1	0	0
0	1	0
1	1	1

$$y = \begin{cases} 0, & w^T x \leq \theta \\ 1, & w^T x \geq \theta \end{cases}$$



```
# AND 게이트 함수
def AND(x1,x2):
    w1,w2,theta = 0.7,0.7,1
    tmp = x1*w1 + x2*w2
    if tmp <= theta:
        return 0
    elif tmp > theta:
        return 1

AND(0,0),AND(1,0),AND(0,1),AND(1,1)

(0, 0, 0, 1)
```

퍼셉트론은 입력값에 따라 출력값을 리턴해 주는 하나의 선형함수라고 할 수 있음. 뇌신경세포인 뉴런(neuron)을 모방한 구조

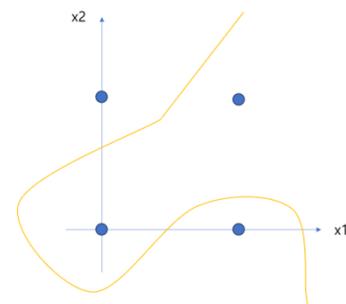
Multilayer Perceptron

Q. 아래에서 x_1, x_2 는 입력값이며, y 는 출력값이다.
입력값에 해당하는 출력값을 만들어내는 함수를
작성해 보아라.

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

```
# OR 게이트 함수
def OR(x1,x2):
    w1,w2,theta = 1.5,1.5,1
    tmp = x1*w1 + x2*w2
    if tmp <= theta:
        return 0
    else:
        return 1
```

```
# NAND 게이트 함수
def NAND(x1,x2):
    w1,w2,theta = 0.7,0.7,1
    tmp = x1*w1 + x2*w2
    if tmp <= theta:
        return 1
    else:
        return 0
```



```
# XOR 게이트 함수
# NAND, OR, AND 의 합성
def XOR(x1,x2):
    s1 = NAND(x1,x2)
    s2 = OR(x1,x2)
    y = AND(s1,s2)
    return y

XOR(0,0),XOR(1,0),XOR(0,1),XOR(1,1)

(0, 1, 1, 0)
```

여러 퍼셉트론을 조합하면 다양한 비선형함수를 표현하기 용이해짐. Multi Perceptron은 Neural Net이라고 부르기도 함.

TensorFlow를 활용한 머신러닝 모델 구축

```
# XOR 게이트 입력력 만들기
import numpy as np
x=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([[0],[1],[1],[0]])

# TensorFlow는 Google이 개발한 오픈소스 머신러닝 프레임워크
# Keras는 신경망 API로 tensorflow에 전용으로 통합됨
# Keras는 복잡한 TensorFlow 코드를 더 직관적으로 사용할 수 있도록 설계됨
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer

# 신경망모델 구축
model=Sequential()

# Layer 등록
# 케라스 Layer는 일종의 멀티퍼셉트론으로 여러가지 Layer들이 설계되어 있음.
# 사용자는 입력값, 출력값 등을 연결해 주면 됨
# 활성화함수 중 하나인 sigmoid 함수의 결과는 확률값으로 이진분류 문제와 함께 쓰임
model.add(InputLayer(shape=(2,))) # 입력값의 모양 지정, 입력값 개수는 불필요
model.add(Dense(units=2,activation='sigmoid')) # units으로 출력 뉴런의 개수 지정
model.add(Dense(units=1,activation='sigmoid')) # 마지막 출력은 자료의 차원으로 설정

# 신경망 모델 컴파일
# binary_crossentropy 는 이진 분류에 사용됨
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Comments

- Layer 내의 활성화함수(activation): Layer의 입력뉴런을 출력뉴런으로 변환하는데 사용되는 함수로 ‘sigmoid’, ‘relu’ 등이 있음.
- 학습(learning)이란: Layer내에 존재하는 가중치를 변경하여, 주어진 손실함수(‘binary_crossentropy’)의 최소값을 찾아가는 과정을 말함.
- 최소값을 찾아가는 대표적인 방법으로 gradient를 활용한 경사하강법이 많이 사용되며, ‘adam’은 경사하강법 알고리즘의 하나라고 보면 됨.

TensorFlow를 활용한 XOR 게이트 문제 풀기

```
# 모델학습
# epoch: 반복학습 회수 지정
# verbose: 0이면 학습과정을 보이지 않음, 1이면 학습과정을 보여줌
# epochs=10000 인 경우 약 7분 소요
model.fit(x,y,epochs=20000,verbose=0)

<keras.src.callbacks.history.History at 0x7ff350970950>

# 모델 예측
predictions=model.predict(x)
print(predictions)

1/1 ━━━━━━━━━━━━ 0s 92ms/step
[[6.7207543e-04]
 [9.9905628e-01]
 [9.9907708e-01]
 [1.1821819e-03]]

# 결과값 반올림
rounded_predictions=np.round(predictions)
print(rounded_predictions)

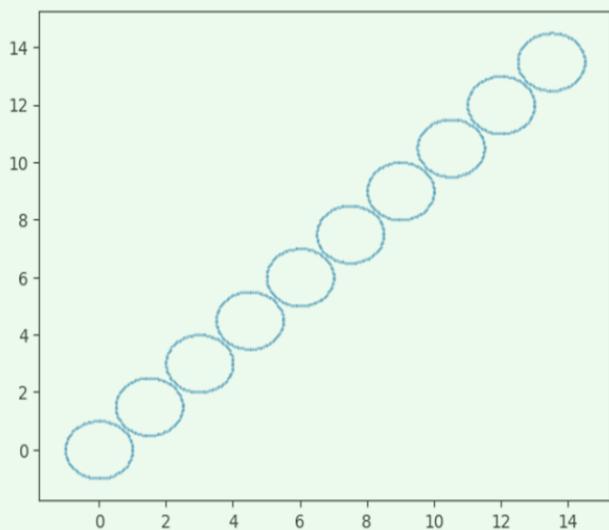
[[0.]
 [1.]
 [1.]
 [0.]]
```

Comments

- ‘fit’을 이용해 간단히 학습하고, ‘predict’를 이용해 예측해 볼 수 있음.
- 텐서플로우를 이용한 학습과정
 - ① 자료의 입력값과 출력값의 차원을 살펴보기
 - ② 모델 설정: Sequential, Functional API, Subclassing 중 하나 선택
 - ③ InputLayer를 사용하여 입력값의 차원 설정하기
 - ④ 적절한 Layer를 원하는 만큼 연결하고, 마지막 Layer의 출력은 자료의 출력값의 차원과 일치시키기
 - ⑤ ‘compile’을 활용하여 손실함수, 최적화방법, 모델성과지표 등 특정하기
 - ⑥ ‘fit’을 활용하여 학습시키기

Regression: 예제 만들기

Q. 중심이 $y=x$ 직선위에 있으며 반지름의 크기가 1인 원들을 10개 생성하고 각 원에서 100개의 점의 좌표를 바탕으로 점들의 좌표를 한데 모아 아래와 같은 산점도(scatter diagram)를 그려보아라.



```
import matplotlib.pyplot as plt
import numpy as np

# 원 생성 함수
# 원의 중심 좌표와 반지름을 인자로 주면
# 원주상의 100개의 점의 좌표를 return 해 줌
def create_circle(center, radius):
    # linspace(a,b,n)은 a부터 b까지 n개의 균등값 생성
    theta = np.linspace(0, 2*np.pi, 100000) # 0부터 2pi까지 100000개의 점 생성
    x = center[0] + radius * np.cos(theta) # x좌표
    y = center[1] + radius * np.sin(theta) # y좌표
    return x, y
```

```
X,Y=[],[] # 점들을 저장할 빈 리스트 생성

# 10개의 원 생성
for i in range(10): # i=0부터 9까지 1씩 증가
    center = [i*1.5,i*1.5] # 원의 중심생성
    x,y = create_circle(center, radius=1)
    X.append(x) # X 리스트에 추가
    Y.append(y) # Y 리스트에 추가
```

```
# numpy array로 자료변환
# reshape 은 array의 형태를 변형
X=np.array(X)
X=X.reshape(1000000,1)
Y=np.array(Y)
Y=Y.reshape(1000000,1)
plt.scatter(X,Y,s=0.2)
```

Regression: 선형회귀식

```
# 선형회귀식 만들기
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score # 결정계수 계산

# 모델 객체 생성
LR=LinearRegression()

# 모델 구축
X=X.reshape(-1,1)
LR.fit(X,Y)

# 예측치 구하기
Yp=LR.predict(X)
print(f'결정계수: {r2_score(Y,Yp)}')

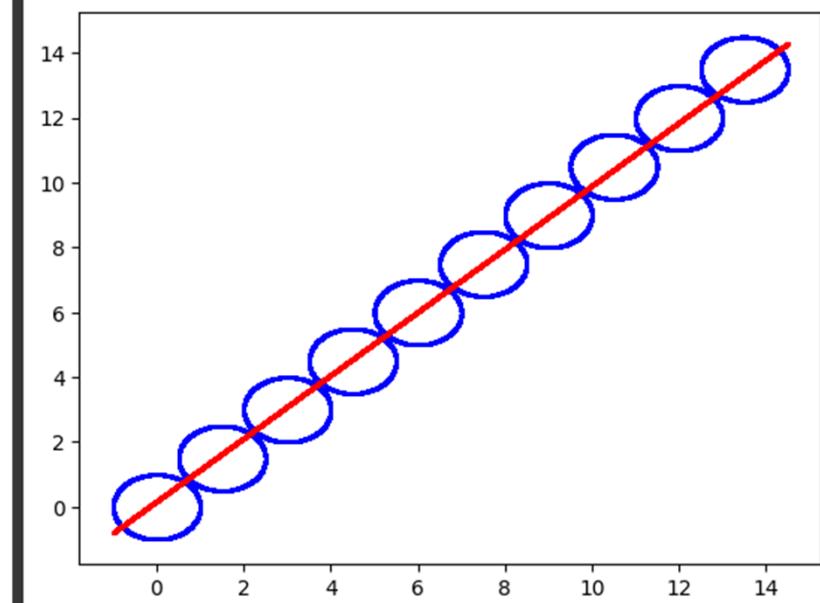
# 회귀 계수 (가중치) 확인
coefficients = LR.coef_ # 기울기 (회귀 계수)
intercept = LR.intercept_ # 절편

print("회귀 계수 (기울기):", coefficients)
print("절편:", intercept)

결정계수: 0.9482289707118388
회귀 계수 (기울기): [[0.97377024]]
절편: [0.17704117]
```

```
# Scatter plot 그리기
plt.scatter(X, Y, color='blue', label='Set 1', s=0.2)
plt.scatter(X, Yp, color='red', label='Set 2', s=0.2)
```

```
<matplotlib.collections.PathCollection at 0x7ff32c472150>
```



Regression: TensorFlow 학습

```
from tensorflow.keras import layers, models

# 모델 정의
model = models.Sequential()

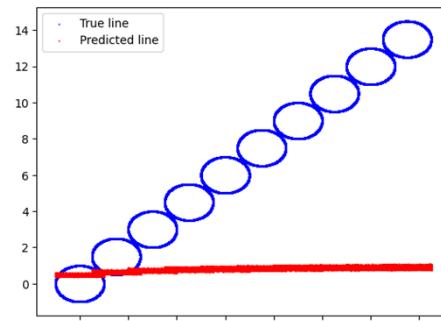
# 100은 필터수로 필터수를 늘리면 모델의 표현력이 증가함
# 그러나 계산비용 증가, overfitting의 위험 존재
model.add(layers.InputLayer(shape=(100000, 1)))
model.add(layers.SimpleRNN(100))
model.add(layers.Dense(100000))

# 모델 컴파일
model.compile(optimizer='adam', loss='mean_squared_error')

# 모델 훈련
model.fit(X, Y, epochs=100, verbose=0)

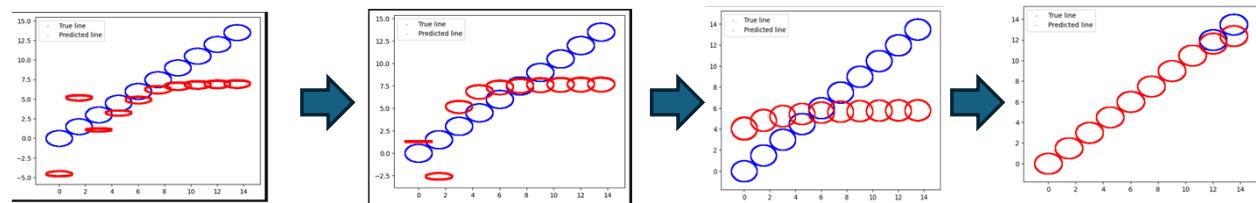
# 테스트 세트에서의 평균 제곱 오차(MSE) 평가
Yp = model.predict(X)

# Scatter plot 그리기
plt.scatter(X, Y, color='blue', label='True line', s=0.2)
plt.scatter(X, Yp, color='red', label='Predicted line', s=0.2)
plt.legend()
```



Comments

- SimpleRNN 레이어는 기본적인 순환신경망(Recurrent Neural Network)으로 입력 데이터가 시간 순서로 존재할 때, 과거 정보를 기억하여 학습시키기 위해 만들어짐
- RNN 학습을 위해서는 GPU나 TPU 활용을 해야함
- 'epoch'를 반복 수행하면서 결과를 살펴보면 아래와 같이 점점 예측력이 좋아지는 것을 확인할 수 있음.



원래 noise가 많은 데이터였다면, noise까지 학습시키는 것
이 정답일까? → 과적합(overfitting) 이슈

AI for Image Processing



Recognizing a Color: RGB 3채널

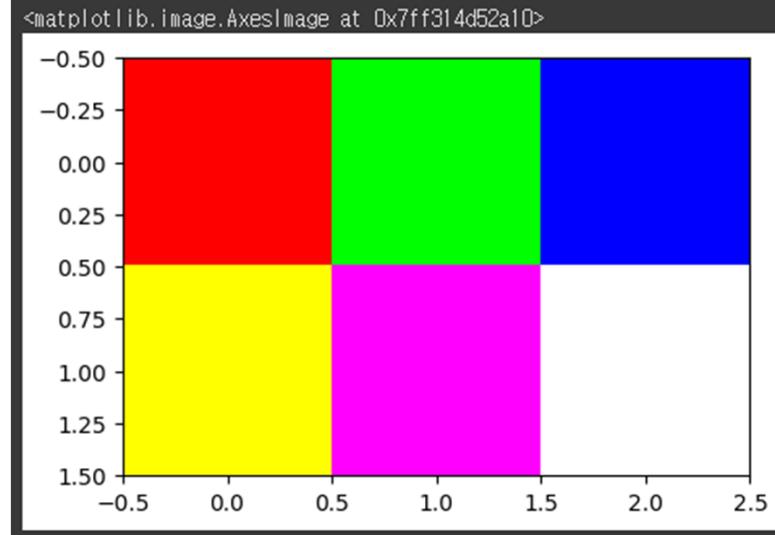
Q. 사각형 6개를 만들고 첫 줄에는 빨강, 녹색, 파랑을, 두번째 줄에는 노랑색, 핑크색, 흰색을 표현해 보아라.

```
# 빨강색의 표현
# (R,G,B)

import matplotlib.pyplot as plt
import numpy as np

C=np.array([255,0,0]) #빨강색 점 지정
# X축,Y축을 정하고, 각 pixel에 들어갈 3차원 공간 설정
image=np.zeros([100,100,3])
image[:, :, ...]=C # 모든 pixel을 C로
# imshow()는 (0,1)사이의 값으로 정규화 필요
plt.figure(figsize=(2, 2)) # 그래프 크기 설정
plt.imshow(image/255)
```

```
CM=np.zeros((2,3,3))
CM[0,0,:]=(255,0,0) # Red
CM[0,1,:]=(0,255,0) # Green
CM[0,2,:]=(0,0,255) # Blue
CM[1,0,:]=(255,255,0) # Yellow
CM[1,1,:]=(255,0,255) # Pink
CM[1,2,:]=(255,255,255) # white
plt.figure(figsize=(5, 5))
plt.imshow(CM/255)
```

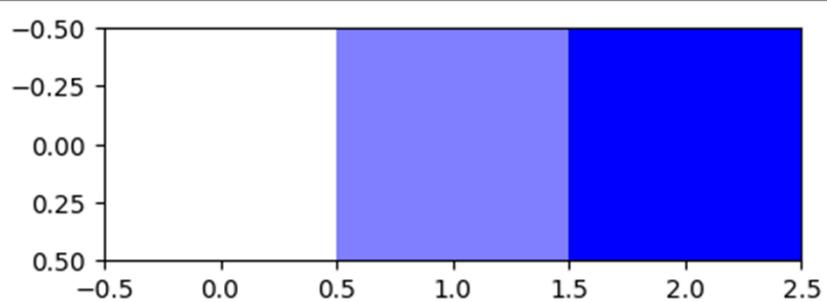


Recognizing a Color: RGBA와 흑백

```
# RGBA 4채널 색상표현
```

```
CM=np.zeros((1,3,4))
CM[0,0,:]=(0,0,255,0) # 투명도 100%
CM[0,1,:]=(0,0,255,128) # 투명도 50%
CM[0,2,:]=(0,0,255,255) # 투명도 0%
plt.figure(figsize=(5, 5))
plt.imshow(CM/255)
```

```
<matplotlib.image.AxesImage at 0x7a16f0fdaed0>
```



```
# 흑백 표현
```

```
# 'cmap=gray'로 설정하지 않으면 기본 컬러맵으로 설정되어
# 흑백이미지가 예상치 못한 색상으로 보일 수 있음
CM = np.zeros([1,2]) # 2차원 배열로 생성
CM[0,0]=0 # 검정색
CM[0,1]=255 # 흰색
```

```
# 흑백 이미지 시각화
```

```
# cmap='gray'로 흑백 표현
plt.imshow(CM/255, cmap='gray')
plt.title("1-Channel Grayscale")
plt.axis('off') # 축 제거
plt.show()
```

1-Channel Grayscale

Recognizing an Image

Q. 자유의 여신상 링크를 바탕으로 사진을 열고,
데이터의 차원을 확인해 보아라.

```
import requests
from google.colab import files
import matplotlib.pyplot as plt

# 다운로드할 이미지 URL 입력
image_url = "https://upload.wikimedia.org/wikipedia/commons/0/0d/Statue_of_Liberty_in_NY.jpg"

# 이미지 다운로드
response = requests.get(image_url)

# 이미지 파일 저장
image_filename = "downloaded_image.jpg"
with open(image_filename, "wb") as file:
    file.write(response.content)

# Colab에서 내 PC로 다운로드
files.download(image_filename)

# 이미지 출력하기
plt.imread(image_filename)
```



```
ndarray (550, 250, 3) show data
array([[206, 219, 236],
       [206, 219, 236],
       [206, 219, 236],
       ...,
       [203, 218, 239],
       [203, 218, 239],
       [203, 218, 239]],
      [[206, 219, 236],
       [206, 219, 236],
       [206, 219, 236],
       ...,
```

```
# 또 다른 방법
import requests
from PIL import Image
from io import BytesIO
import IPython.display as display

# 이미지 다운로드
response = requests.get(image_url)

# 이미지 열기 및 표시
image = Image.open(BytesIO(response.content))
display.display(image)
```

손글씨 숫자 이미지 학습: mnist데이터

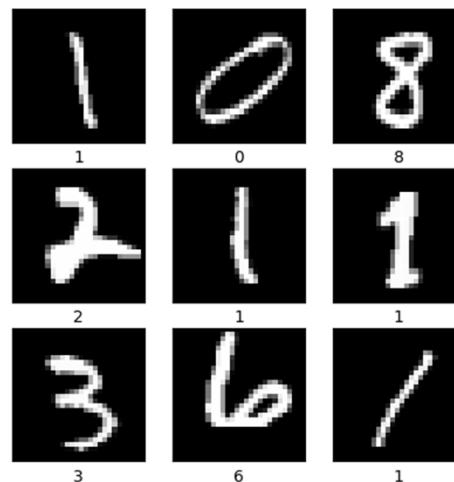
```
from tensorflow.keras.datasets import mnist
```

```
# MNIST 데이터셋은 아래와 같이 load_data()를 이용해  
# 학습용, 테스트용을 분리하여 읽어오게 된다.
```

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()  
train_images.shape, test_images.shape
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz  
11490434/11490434 1s 0us/step  
(60000, 28, 28), (10000, 28, 28)
```

```
# 9개 랜덤 이미지 출력  
import matplotlib.pyplot as plt  
import numpy as np  
  
N=np.random.randint(0,60000,size=9) # 랜덤 인덱스 추출  
plt.figure(figsize=(5,5)) # 그림판의 가로, 세로 크기  
k=1  
for i in N:  
    plt.subplot(3,3,k) # 9개의 subplot  
    plt.imshow(train_images[i],cmap='gray')  
    plt.xlabel(train_labels[i]) # 해당 레이블에 대한 class_names  
    plt.xticks([]) # x축 표기 숫자 제거  
    plt.yticks([]) # y축 표기 숫자 제거  
    k+=1 # k=k+1
```



Comments

- ‘load_data’를 통해 학습용 데이터 60,000개와 테스트용 데이터 10,000개를 분리하여 다운받게 됨
- 각 이미지의 채널은 (28,28)로서 흑백 이미지임을 예상할 수 있음.
- ‘train_images’에는 0부터 9까지 60,000개의 손글씨 숫자가 있고, 해당 손글씨 숫자에 대한 정답이 ‘train_labels’에 있음
- 학습용 데이터로 학습한 후, 테스트용 데이터를 통해 얼마나 정확도가 있는지 알아보게 될 것임

손글씨 숫자 이미지 학습: ANN* 학습 모델 구축

```
# 텐서플로우 신경망 학습
from tensorflow.keras.layers import InputLayer,Dense,Flatten
from tensorflow.keras.models import Sequential
from sklearn.metrics import accuracy_score,confusion_matrix

# 신경망 모델 구축
model = Sequential()

# 레이어 등록
model.add(InputLayer(shape=(28, 28)))
model.add(Flatten()) # 이미지를 1차원 배열로 평탄화
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=10, activation='softmax'))

# 모델 컴파일
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 학습
history=model.fit(train_images, train_labels, epochs=50,
                   verbose=1,validation_split=0.2)
```

Comments

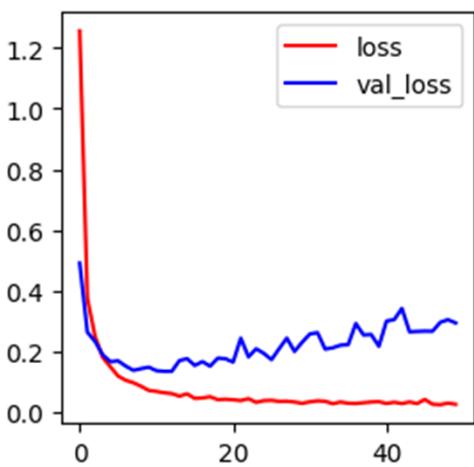
- Flatten() 레이어를 사용해 데이터를 1차원으로 변환
- 중간 레이어에는 ‘relu’라는 활성화함수 사용
- 마지막 레이어는 결과를 10개가 나오도록 설정하며, ‘softmax’라는 활성화 함수 사용함. ‘softmax’는 10개의 값 각각에 대한 확률값을 출력해 줌
- 손실함수로는 ‘sparse_categorical_crossentropy’를 사용하였음.
- ‘validation_split’ 매개변수를 통해 학습용 데이터 자체를 가지고, 20%를 예측 정확도를 측정하는 용도로 사용하였음
- 모델 학습 과정을 history라는 변수에 할당해 줌

*ANN: Artificial Neural Network, Dense 레이어로만 구성된 모형

손글씨 숫자 이미지 학습: 예측과 평가

```
import matplotlib.pyplot as plt
# 'loss'는 훈련 데이터의 손실값, epochs에 따라 감소
# 'val_loss'는 검증 데이터의 손실로,
# epochs에 따라 감소하지 않을 수 있음(과적합 문제)
loss=history.history['loss']
val_loss=history.history['val_loss']
plt.figure(figsize=(3,3))
plt.plot(loss,'r',label='loss')
plt.plot(val_loss,'b',label='val_loss')
plt.legend()

<matplotlib.legend.Legend at 0x7a165bf7a150>
```

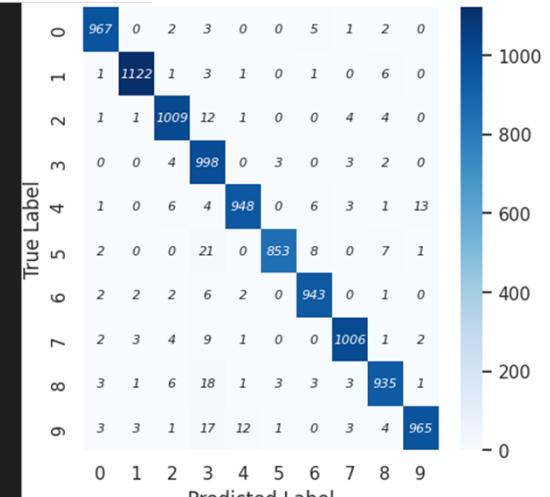


```
# 'test_images'를 사용해 얼마나 잘 학습되었는지 예측
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

pred=model.predict(test_images) # 예측
# softmax 함수의 결과가 10개의 확률값으로,
# argmax를 이용해 최대값을 갖는 인덱스 추출
pred_labels=np.argmax(pred,axis=1)
# confusion_matrix는 실제값을 어떻게 예측했는지 행렬로 보여줌
conf_matrix=confusion_matrix( test_labels,pred_labels)

sns.set(style='white')

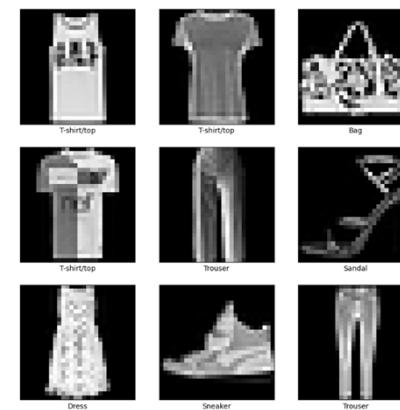
plt.figure(figsize=(5,5))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', cbar=True,
            annot_kws={"size": 8, "weight": "normal", "style": "italic"})
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```



옷 이미지 학습: fashion_mnist 데이터

```
from tensorflow.keras.datasets import fashion_mnist  
  
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()  
train_images.shape, test_images.shape  
  
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz  
29515/29515 - 0s 0us/step  
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz  
26421880/26421880 - 0s 0us/step  
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz  
5148/5148 - 0s 0us/step  
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz  
4422102/4422102 - 0s 0us/step  
((60000, 28, 28), (10000, 28, 28))
```

```
# 9개 랜덤 이미지 출력  
import matplotlib.pyplot as plt  
import numpy as np  
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',  
               'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']  
  
N=np.random.randint(0,60000,size=9) # 랜덤 인덱스 추출  
plt.figure(figsize=(10,10)) # 그림판의 가로, 세로 크기  
k=1  
for i in N:  
    plt.subplot(3,3,k) # 25개의 subplot  
    plt.imshow(train_images[i],cmap='gray')  
    plt.xlabel(class_names[train_labels[i]]) # 해당 레이블에 대한 class_names  
    plt.xticks([]) # x축 표기 숫자 제거  
    plt.yticks([]) # y축 표기 숫자 제거  
    k+=1 # k=k+1
```



Comments

- 각 이미지의 채널은 (28,28)로서 흑백 이미지라고 할 수 있음
- 'train_images'에는 10가지 옷 이미지 60,000개가 있음
- 해당 이미지에 대한 대한 정답이 'train_labels'에 0부터 9까지의 label로 변환되어 있음. AI 학습 자료는 모두 숫자형이어야 함
- 참고로 0부터 9 숫자 label에 해당하는 옷 이름은 class_names에 정의되어 있음

옷 이미지 학습: ANN 학습

```
# 신경망 모델 구축
model = Sequential()

# 레이어 등록
model.add(InputLayer(shape=(28, 28)))
model.add(Flatten()) # 이미지를 1차원 배열로 평탄화
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=10, activation='softmax'))

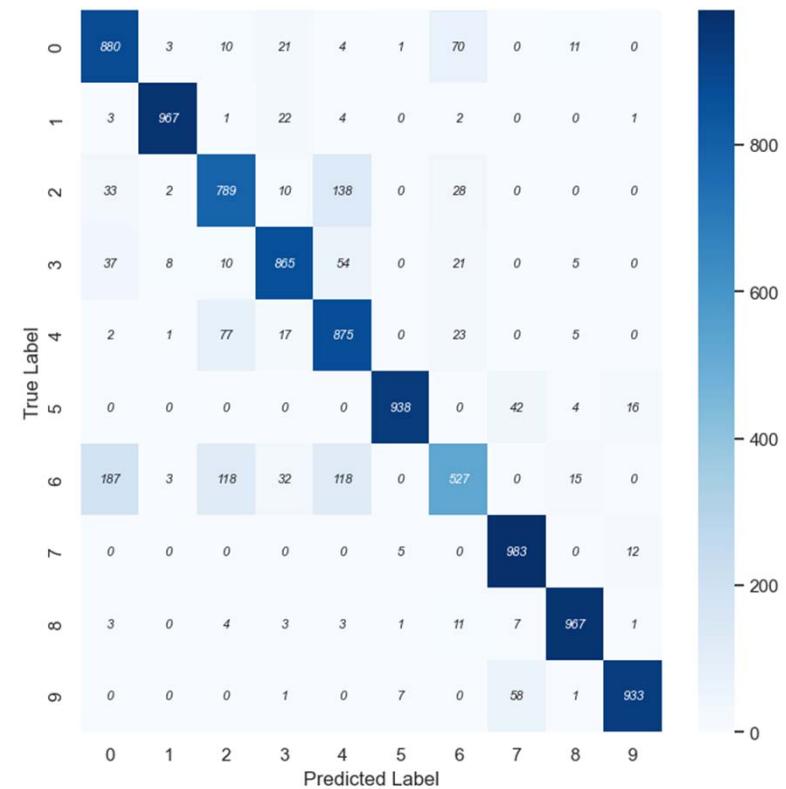
# 모델 컴파일
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 학습
history=model.fit(train_images, train_labels, epochs=50,
                   verbose=1, validation_split=0.2)

# 모델 예측과 혼동행렬
pred=model.predict(test_images)
conf_matrix=confusion_matrix( test_labels,np.argmax(pred, axis=1))

sns.set(style='white')

plt.figure(figsize=(8,8))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', cbar=True,
            annot_kws={"size": 8, "weight": "normal", "style": "italic"})
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```



사물 이미지 학습: CIFAR10 데이터

```
# 데이터 불러오기
from tensorflow.keras.datasets import cifar10

(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
train_images.shape, test_images.shape
((50000, 32, 32, 3), (10000, 32, 32, 3))
```

```
# 9개 랜덤 이미지 출력
import matplotlib.pyplot as plt
import numpy as np
class_names = ['airplane', 'automobile', 'bird', 'cat',
               'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

N=np.random.randint(0,50000, size=9) # 랜덤 인덱스 추출
plt.figure(figsize=(5,5)) # 그림판의 가로, 세로 크기
k=1
for i in N:
    plt.subplot(3,3,k) # 25개의 subplot
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
    plt.xticks([]) # x축 표기 수자 제거
    plt.yticks([]) # y축 표기 수자 제거
    k+=1 # k=k+1
```



Comments

- 각 이미지의 채널은 (32,32,3)로서 RGB 컬러 이미지라고 할 수 있음
- 'train_images'에는 10가지 사물 이미지 50,000개가 있음
- 해당 이미지에 대한 대한 정답이 'train_labels'에 0부터 9까지의 label로 변환되어 있음
- 참고로 0부터 9 숫자 label에 해당하는 사물 이름은 class_names에 정의되어 있음

사물 이미지 학습: ANN 학습

```
# 신경망 모델 구축
model = Sequential()

# 레이어 등록
model.add(InputLayer(shape=(32,32,3)))
model.add(Flatten()) # 이미지를 1차원 배열로 평탄화
model.add(Dense(units=128, activation='relu'))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=10, activation='softmax'))

# 모델 컴파일
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 학습
history=model.fit(train_images, train_labels, epochs=5,
                   verbose=1, validation_split=0.2)

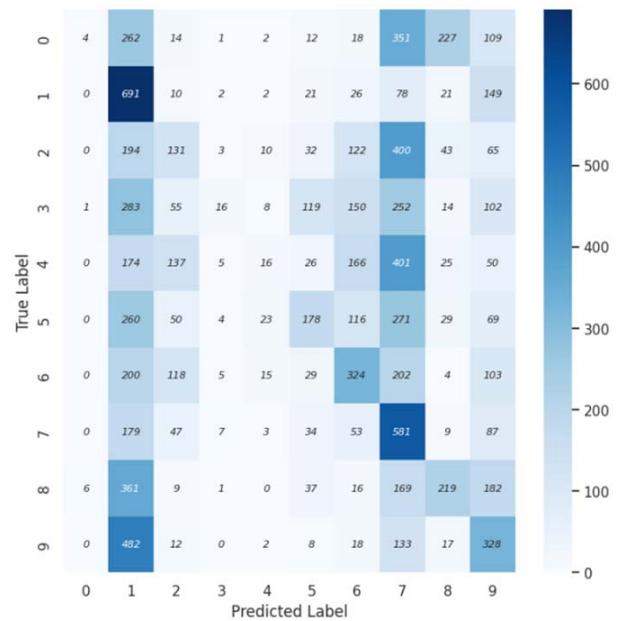
# 모델 예측과 혼동행렬
pred=model.predict(test_images)
conf_matrix=confusion_matrix( test_labels,np.argmax(pred,axis=1))

sns.set(style='white')

plt.figure(figsize=(8,8))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', cbar=True,
            annot_kws={"size": 8, "weight": "normal", "style": "italic"})
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```
# 정확도 출력
from sklearn.metrics import accuracy_score
pred_labels=np.argmax(pred, axis=1)
print('예측 정확도:', accuracy_score(test_labels,pred_labels))

예측 정확도: 0.2488
```



Advanced Topics

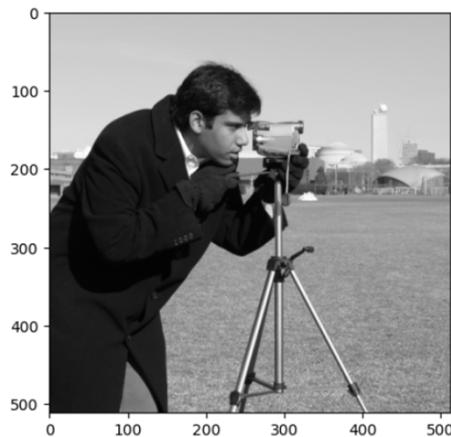


Convolution?

```
import matplotlib.pyplot as plt
from skimage import data

# 이미지 로드 (예: 카메라 이미지)
image = data.camera()

# 이미지 보기
plt.imshow(image, cmap='gray')
```

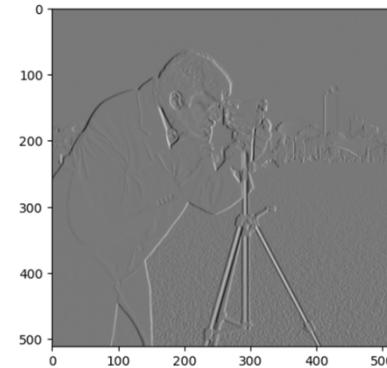


```
import numpy as np
from scipy.signal import convolve2d

# 커널 생성
kernel = np.array([[1, 0, -1],
                   [2, 0, -2],
                   [1, 0, -1]])

# 컨볼루션 연산 수행
result = convolve2d(image, kernel, mode='same')

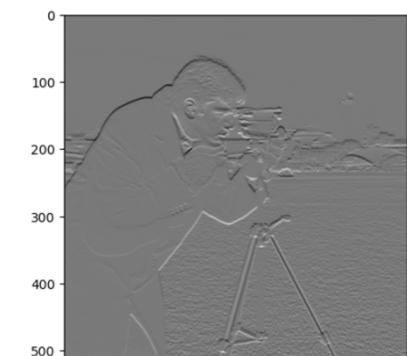
# 결과 시각화
plt.imshow(result, cmap='gray')
```



```
# 세로방향 컨볼루션
# 커널 생성
kernel = np.array([[1, 2, 1],
                   [0, 0, 0],
                   [-1, -2, -1]])

# 컨볼루션 연산 수행
result = convolve2d(image, kernel, mode='same', boundary='symm')

# 결과 시각화
plt.imshow(result, cmap='gray')
```



사물 이미지 학습: CNN * 모형 구축

```
# CNN 모형
# Conv2D, MaxPool2D 레이어는 CNN 모형에 사용
from tensorflow.keras.layers import InputLayer,Dense,Flatten,Conv2D,MaxPool2D
from tensorflow.keras.models import Sequential

# 신경망 모델 구축
model = Sequential()

# 레이어 등록
model.add(InputLayer(shape=(32,32,3)))
model.add(Conv2D(filters=32,kernel_size=(3,3),padding='valid',activation='relu'))
model.add(MaxPool2D(2,2)) # 필터링 된 결과를 또 압축

model.add(Conv2D(64,(3,3),padding='valid',activation='relu'))
model.add(MaxPool2D(2,2))

model.add(Flatten())
model.add(Dense(units=10, activation='softmax'))

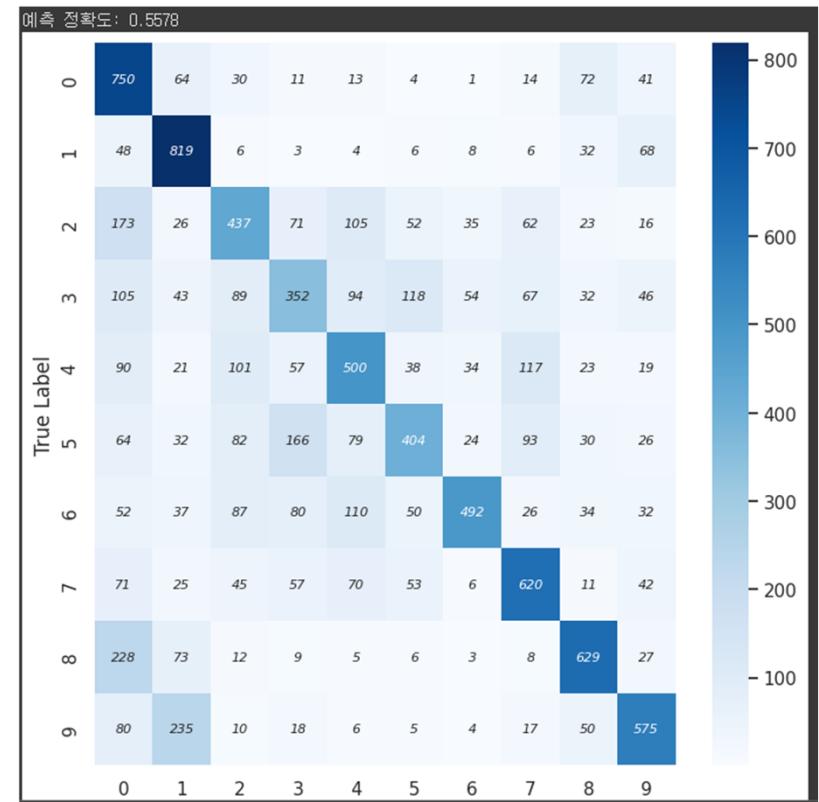
model.summary() # 모델 확인
```

Model: "sequential_3"		
Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_4 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_5 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_3 (Flatten)	(None, 2304)	0
dense_8 (Dense)	(None, 10)	23,050

Total params: 42,442 (165.79 KB)
Trainable params: 42,442 (165.79 KB)
Non-trainable params: 0 (0.00 B)

사물 이미지 학습: CNN 학습

```
# 모델 컴파일  
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])  
  
# 모델 학습  
history=model.fit(train_images, train_labels, epochs=50,  
                   verbose=1, validation_data=(test_images,test_labels))  
  
# 모델 예측과 혼동행렬  
pred=model.predict(test_images)  
pred_labels=np.argmax(pred, axis=1)  
conf_matrix=confusion_matrix(test_labels,pred_labels)  
  
sns.set(style='white')  
  
plt.figure(figsize=(8,8))  
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', cbar=True,  
            annot_kws={"size": 8, "weight": "normal", "style": "italic"})  
plt.xlabel('Predicted Label')  
plt.ylabel('True Label')  
print('예측 정확도:', accuracy_score(test_labels,pred_labels))
```



Embedding?

임베딩(embedding)이란 단어, 문장, 이미지 등의 다양한 유형의 데이터를 벡터 공간에 매핑하는 기술들을 의미한다.

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

# 문장 데이터
sentences = [
    "한국은 아름다운 나라입니다.",
    "오늘 날씨가 좋네요.",
    "인공지능은 빠르게 발전하고 있습니다."
]

# 태그가 지정된 문서로 변환
tagged_data = [TaggedDocument(words=sentence.split(), tags=[str(i)]) for i, sentence in enumerate(sentences)]

# Doc2Vec 모델 학습
model = Doc2Vec(vector_size=100, alpha=0.025, min_alpha=0.00025, min_count=1, dm=1, epochs=100)
model.build_vocab(tagged_data)
model.train(tagged_data, total_examples=model.corpus_count, epochs=model.epochs)

# 각 문장의 벡터 추출
sentence_vectors = [model.infer_vector(sentence.split()) for sentence in sentences]
```

```
sentence_vectors[0]
array([ 2.4499847e-03,  2.7034997e-03,  2.1970592e-03,  4.4220462e-04,
       -1.9106810e-03,  1.2104870e-03,  1.3904123e-03,  -5.0817677e-03,
      -3.7231506e-04,  1.8070794e-03,  -4.3188538e-03,  5.0406661e-03,
      -1.0319690e-03,  3.7238502e-03,  5.6448841e-04,  3.0137801e-03,
      -1.7543872e-03,  -3.4062867e-04,  -3.2703718e-03,  -3.4487753e-03,
      9.0562156e-05,  -3.5689378e-03,  1.9029301e-03,  5.3596469e-03,
     -3.9880881e-03,  1.2158056e-03,  5.0912279e-04,  -4.0187021e-03,
      3.9556320e-03,  3.7663924e-03,  2.5242547e-04,  -3.6578465e-03,
     -2.9199251e-03,  3.8707284e-03,  6.7377521e-04,  2.5817850e-03,
     -4.3739844e-03,  2.5762042e-03,  -2.5353772e-03,  -4.9546993e-03,
     -3.5744412e-03,  2.6390944e-03,  4.2264992e-03,  3.5796862e-03,
     -4.4756751e-03,  -5.1706962e-03,  1.6601145e-03,  -1.9344211e-03,
     -3.3587180e-03,  -1.4402145e-03,  -4.7761970e-03,  -9.6618751e-03,
      4.3715993e-03,  -2.2955195e-03,  -5.3979985e-04,  2.4035687e-03,
     -4.8316843e-03,  3.0679945e-04,  3.7636035e-04,  1.8665777e-03,
     -2.5691548e-03,  3.2743767e-03,  1.9143241e-03,  -2.1641953e-03,
      2.935049e-03,  -8.8662311e-04,  -2.1640590e-04,  -4.0783819e-03,
     -1.9940464e-03,  1.0982229e-03,  -5.6148944e-03,  1.7419764e-03,
      2.1767814e-03,  -3.7238738e-03,  -3.5156044e-03,  3.7305249e-03,
     -9.7470183e-04,  5.9322437e-04,  3.2831051e-03,  -2.0428395e-03,
      2.7160107e-03,  -3.5705226e-03,  2.7633039e-04,  -3.7312829e-03,
     -1.8501710e-04,  3.3051323e-03,  4.1603632e-03,  2.8783272e-03,
     -1.0796399e-03,  -4.2855609e-03,  2.8297275e-03,  3.9516808e-03,
      1.1110901e-03,  3.6923934e-03,  -3.4740185e-03,  4.9812226e-03,
      1.9972334e-03,  4.1470975e-03,  3.2541393e-03,  4.1054417e-03],  
dtype=float32)
```

Transformer?

Attention Is All You Need

Ashish Vaswani* Noam Shazeer* Niki Parmar* Jakob Uszkoreit*
Google Brain Google Brain Google Research Google Research
avaswani@google.com noam@google.com nikip@google.com usz@google.com

Llion Jones* Aidan N. Gomez* † Lukasz Kaiser*
Google Research University of Toronto Google Brain
llion@google.com aidan@cs.toronto.edu lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been actively involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation. Lukasz was the chief person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

Self-Attention: 입력 토큰들의 관계학습

문장:

"나는 밥을 먹는다."

단어	Query(Q)	Key(K)	유사도 계산
"나는"	현재 보고 있는 단어	비교할 단어	"나는" vs. "밥을", "먹는다"
"밥을"	현재 보고 있는 단어	비교할 단어	"밥을" vs. "나는", "먹는다"
"먹는다"	현재 보고 있는 단어	비교할 단어	"먹는다" vs. "나는", "밥을"

➤ 예상되는 Self-Attention 결과

Query (현재 단어)	Key (비교 단어)	Attention Score (유사도)
"나는"	"나는"	0.8 (높음)
"나는"	"밥을"	0.3 (낮음)
"나는"	"먹는다"	0.5 (중간)
"밥을"	"나는"	0.3 (낮음)
"밥을"	"밥을"	0.9 (높음)
"밥을"	"먹는다"	0.7 (중간)
"먹는다"	"나는"	0.5 (중간)
"먹는다"	"밥을"	0.7 (중간)
"먹는다"	"먹는다"	0.9 (높음)

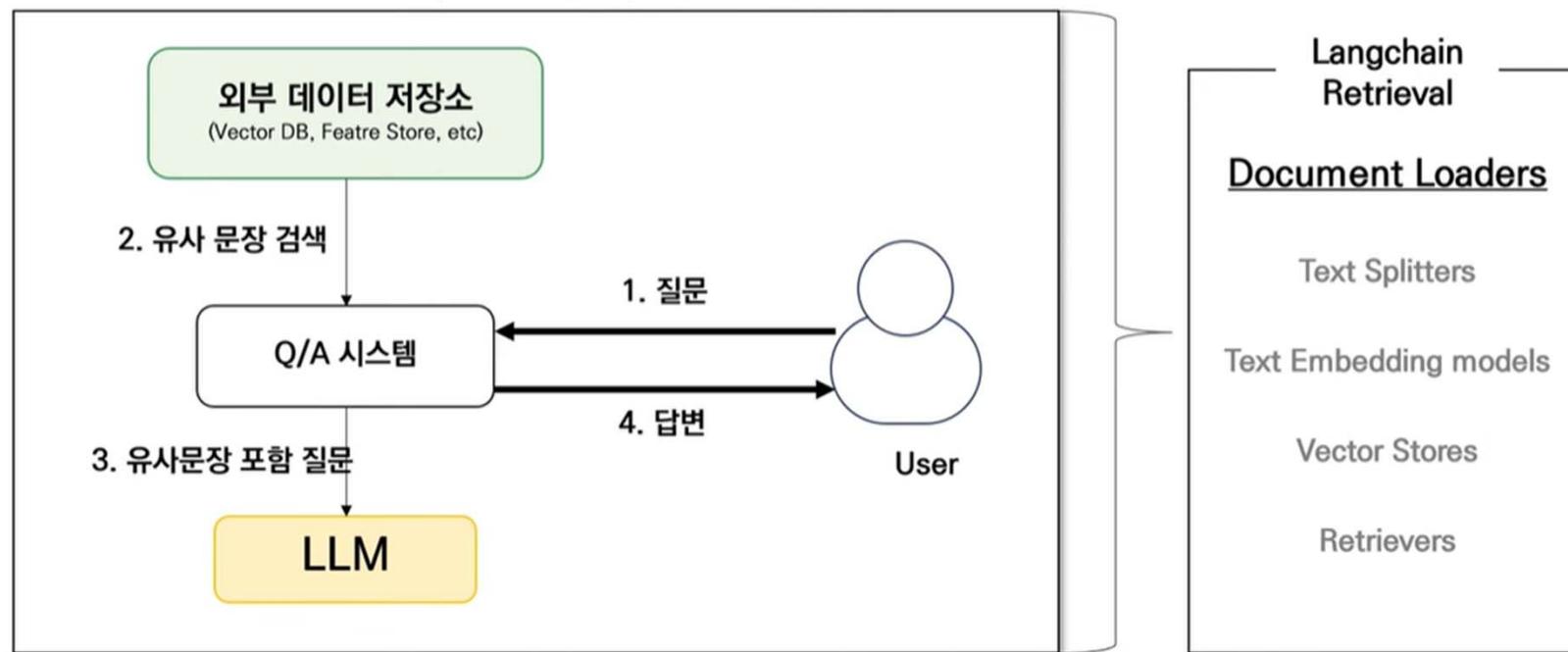
Comments

- 2017년 논문에 의해 LLM의 혁신을 이루어 냈으며, 최근의 multimodal 모형은 모두 transformer 모형에 의존하고 있음
- Transformer는 인코더(encoder)와 디코더(decoder)로 구분할 수 있음.
- 인코더: 어텐션 메커니즘을 이용하여 단어간의 관계성을 학습하면서 문장의 의미를 효율적으로 압축하는 역할
- 디코더: 인코더의 정보를 활용하여 단어를 순차적으로 생성하는 역할을 수행

RAG?

RAG(Retrieval Augmented Generation)은 외부 데이터를 참조하여 LLM이 답변할 수 있도록 해주는 프레임워크

〈RAG 구조〉



Transfer Learning

Q. 'huggingface/cats-image'에는 여러 배경하에 있는 고양이 이미지들이 들어 있다. 첫번째 이미지를 불러와서 고양이가 현재 어떤 배경하에 있는지 설명하도록 추론해 보아라.

```
import matplotlib.pyplot as plt
from datasets import load_dataset

dataset=load_dataset("huggingface/cats-image")
image=dataset["test"]["image"][0]
plt.imshow(image)
plt.show()
```



```
# BLIP(Bootstrapping Language-Image Pre-training)은 이미지와 텍스트 간의 관계를
# 효과적으로 학습해 우수한 이미지 캡셔닝 성능을 보이는 최신 모델
from transformers import Blip2Processor, Blip2ForConditionalGeneration
import torch
```

```
# 모델 이름
model_name = "Salesforce/blip2-opt-2.7b"
```

```
# Processor와 모델 로드
processor = Blip2Processor.from_pretrained(model_name)
model = Blip2ForConditionalGeneration.from_pretrained(
    model_name,
    torch_dtype=torch.float16, # 16-bit 부동소수점 연산 사용
    device_map="auto"          # 자동으로 GPU 또는 CPU에 할당
)
```

```
prompt='Question: Where are the cats? Answer:'
# processor로 이미지, prompt를 모두 변환시킴
inputs=processor(text=prompt,images=image,return_tensors="pt").to(model.device,torch.float16)
generated_ids=model.generate(**inputs,max_length=100) # 문장 생성
# 토큰 ID를 텍스트로 디코딩
# skip_special_tokens=True로 특수 토큰 제외
generated_text=processor.batch_decode(generated_ids,skip_special_tokens=True)[0].strip()
print(generated_text)
```

```
Question: Where are the cats? Answer: On the couch, sleeping
```