# RNN 실습 및
# GAN(Generative Adversarial Network) 기초

임세진

https://youtu.be/r1wLfJSB1nI

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

# Contents



CryptoCraft LAB

# 01. RNN 실습 - Pytorch

# 01. RNN 실습 – Pytorch

- **Pytorch**

- 딥러닝 프레임워크 중 하나로, 최근 TensorFlow를 제치고 활발히 사용됨

- 장점
  - TensorFlow에 비해 간결한 코드
  - 일반적으로 Pytorch로 구현한 모델이 TensorFlow로 구현한 모델보다 성능(속도)이 좋음

- Tensor : Pytorch에서 기본 단위로 사용되는 Array (Numpy의 array와 비슷한 개념)

# 01. RNN 실습 – Pytorch

- **LSTM으로 NLP 실습하기**

- LSTM : RNN에서 시퀀스가 길어졌을 때의 문제를 해결하기 위해 사용

- NLP(Natural Language Processing) : [자연어 처리]
  - 자연어 : 일상 생활에서 사용하는 언어
  - 자연어 처리 : 자연어의 의미를 분석하여 컴퓨터가 처리할 수 있도록 하는 일

# 01. RNN 실습 – Pytorch

```python
#RNN의 입력 단위를 단어 단위로 사용하기.
#NLP(Natural Language Processing) 목적. 앞의 두 단어를 보고 뒤에 나올 단어를 예측.

import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim

sentences = ["i like dog", "i love coffee", "i hate milk", "you like cat", "you love milk", "you hate coffee"]
dtype = torch.float #사용할 tensor의 속성

"""
Word Processing
"""
word_list = list(set(" ".join(sentences).split()))
word_dict = {w: i for i, w in enumerate(word_list)} #enumerate를 사용하면 반복문의 인덱스 번호와 컬렉션의 원소를 tuple형태로 반환. 단어에 고유한 정수 인덱스 부여
number_dict = {i: w for i, w in enumerate(word_list)}
n_class = len(word_dict)
```

```
['like', 'cat', 'milk', 'dog', 'hate', 'coffee', 'you', 'love', 'i']
```

**Word_dict** : {'like': 0, 'cat': 1, 'milk': 2, 'dog': 3, 'hate': 4, 'coffee': 5, 'you': 6, 'love': 7, 'I': 8}

**Number_dict** : {0: 'like', 1: 'cat', 2: 'milk', 3: 'dog', 4: 'hate', 5: 'coffee', 6: 'you', 7: 'love', 8: 'i'}

# 01. RNN 실습 – Pytorch

```
batch_size = len(sentences)
n_step = 2  # 학습 하려고 하는 문장의 길이 - 1. input의 사이즈
n_hidden = 5  # 은닉층의 사이즈

def make_batch(sentences):
  input_batch = []
  target_batch = []

  for sen in sentences:
    word = sen.split()
    input = [word_dict[n] for n in word[:-1]]
    target = word_dict[word[-1]]

    input_batch.append(np.eye(n_class)[input])  # One-Hot Encoding
    target_batch.append(target)

  return input_batch, target_batch
```

```
input_batch, target_batch = make_batch(sentences)
input_batch = torch.tensor(input_batch, dtype=torch.float32, requires_grad=True)
target_batch = torch.tensor(target_batch, dtype=torch.int64)
```

```
{ 'like': 0, 'cat': 1, 'milk': 2, 'dog': 3 … }
```

milk를 ont-hot-encoding ➔ [0, 0, 1, 0, 0, 0, 0, 0, 0]

# 01. RNN 실습 – Pytorch

```
#TextLSTM 모델 설계
class TextLSTM(nn.Module):
  def __init__(self):
    super(TextLSTM, self).__init__()

    self.lstm = nn.LSTM(input_size=n_class, hidden_size=n_hidden, dropout=0.3) #dropout을 통해 현재 정보에서 기억할 %를 정할 수 있음
    self.W = nn.Parameter(torch.randn([n_hidden, n_class]).type(dtype))
    self.b = nn.Parameter(torch.randn([n_class]).type(dtype))
    self.Softmax = nn.Softmax(dim=1)

  def forward(self, hidden_and_cell, X):
    X = X.transpose(0, 1)
    outputs, hidden = self.lstm(X, hidden_and_cell)
    outputs = outputs[-1]  # 최종 예측 Hidden Layer
    model = torch.mm(outputs, self.W) + self.b  # 최종 예측 최종 출력 층
    return model
```

**순전파 (forward propagation)**

입력층부터 출력층까지 순서대로 변수들을 계산하고 저장하는 것

# 01. RNN 실습 – Pytorch

```
#Training
model = TextLSTM() #모델  생성
criterion = nn.CrossEntropyLoss() #손실함수 정의. 소프트맥스 함수 포함이며 실제 값은 원-핫 인코딩 안해도 됨
optimizer = optim.Adam(model.parameters(), lr=0.01) #옵티마이저 정의

for epoch in range(500):
  hidden = torch.zeros(1, batch_size, n_hidden, requires_grad=True)
  cell = torch.zeros(1, batch_size, n_hidden, requires_grad=True)
  output = model((hidden, cell), input_batch)
  loss = criterion(output, target_batch)

  if (epoch + 1) % 100 == 0: #기록
    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.6f}'.format(loss))

  optimizer.zero_grad() #이전 epoch에서 계산된 기울기를 0으로 초기화
  loss.backward() #역방향 전파. 각 layer에 대해 체인룰을 적용하여 기울기 계산 (이전 기울기에 누적하여 계산)
  optimizer.step() #매개변수 업데이트. 기울기 갱신

input = [sen.split()[:2] for sen in sentences]

hidden = torch.zeros(1, batch_size, n_hidden, requires_grad=True)
cell = torch.zeros(1, batch_size, n_hidden, requires_grad=True)
predict = model((hidden, cell), input_batch).data.max(1, keepdim=True)[1]
print([sen.split()[:2] for sen in sentences], '->', [number_dict[n.item()] for n in predict.squeeze()])
```

**역전파 (back propagation)**

뉴럴 네트워크의 파라미터들에 대한 그래디언트를 계산하는 법

중간 변수와 파라미터에 대한 그래디언트를 반대방향으로 계산하고 저장

```
Epoch: 0100 cost = 0.391738
Epoch: 0200 cost = 0.062609
Epoch: 0300 cost = 0.022932
Epoch: 0400 cost = 0.013043
Epoch: 0500 cost = 0.008647
[['i', 'like'], ['i', 'love'], ['i', 'hate'], ['you', 'like'], ['you', 'love'], ['you', 'hate']] -> ['dog', 'coffee', 'milk', 'cat', 'milk', 'coffee']
```

# 02. GAN의 개념

# 02. GAN의 개념

- **생성 모델 (Generative Models)**

- 주어진 학습 데이터를 학습 ➔ 학습 데이터의 분포를 따르는 유사한 데이터를 생성하는 모델

- 실존하지 않지만 있을 법한 데이터를 생성할 수 있는 모델

- 기존 데이터 분포를 근사하는 모델 G를 만드는 것이 생성 모델의 목표

- 2014년에 제안된 GAN이 대표적임

  이를 응용한 다양한 후속 연구 진행ing

<유명인 사진을 바탕으로 GAN이 만들어낸 허구의 인물>

# 02. GAN의 개념

- **연속확률분포**

- 확률분포 : 확률변수가 특정한 값을 가질 확률을 나타내는 함수 (이산확률분포, 연속확률분포)

- 연속확률분포 : 확률변수 X의 개수를 정확히 셀 수 없을 때 분포를 표현하는 방법

ex) 키, 몸무게 등

# 02. GAN의 개념

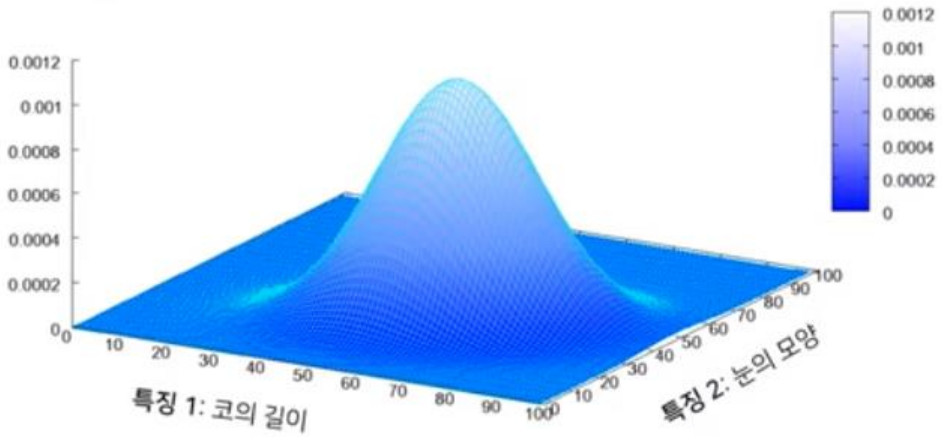
- **이미지 데이터에 대한 확률분포**

이미지는 많은 픽셀로 구성 + 각 픽셀이 RGB의 채널을 가지고 있음

- 이미지 데이터는 다차원 특징 공간의 한 점으로 표현됨 ➔ 이 분포를 학습할 수 있음
- 사람의 얼굴에는 통계적인 평균치가 존재할 수 있음 ➔ 모델은 이를 수치화 할 수 있음

신체적 특징

- 이미지의 다양한 특징들이 각각 확률 변수가 될 수 있음 (다변수 확률분포)



# 02. GAN의 개념

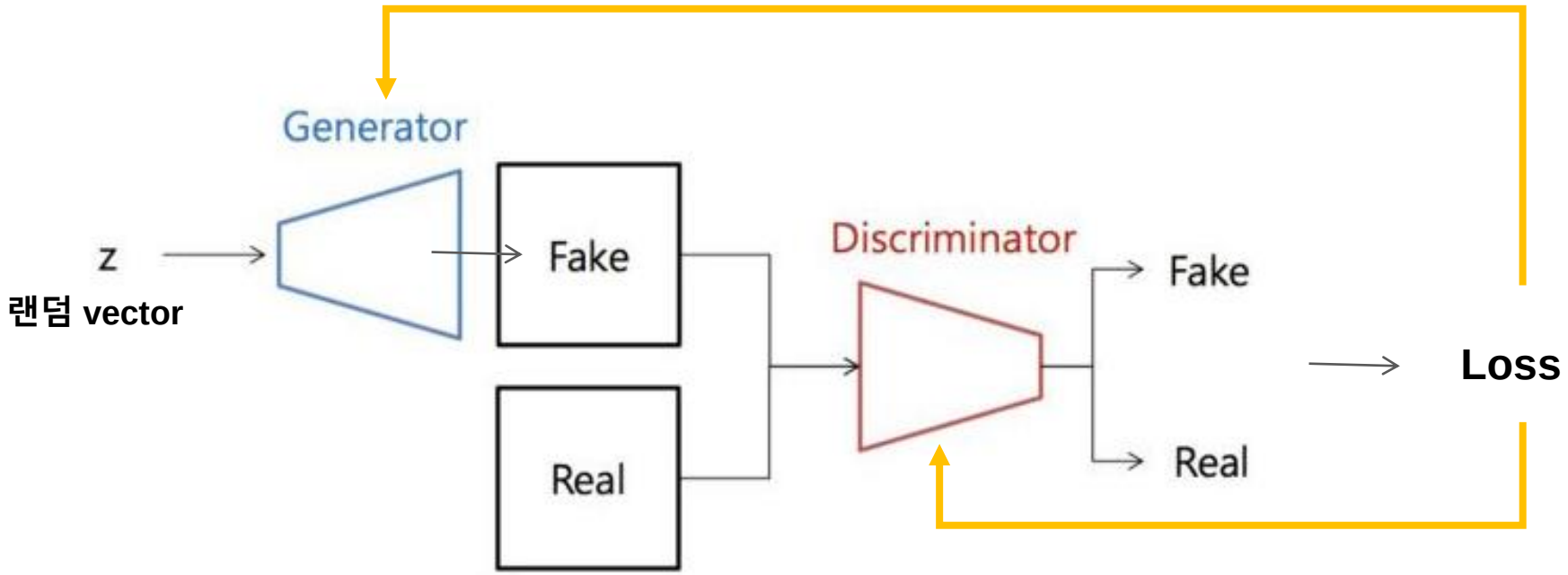
- **GAN (Generative Adversarial Network)**

- 생산적 적대 신경망
- 생성자와 판별자가 서로 **경쟁하면서** 데이터를 **생성하는 모델**
- 생성자(generator)와 판별자(discriminator) 두개의 네트워크를 활용한 생성 모델
- 경쟁을 통해 **서로의 성능을 점차 개선해 나가는 방향으로** 학습이 진행됨
- 주로, 학습이 끝난 후에 생성자(generator)가 많이 사용되고 판별자는 생성자의 학습을 돕는 역할을 함
- 이미지 생성 등 다양한 분야에 활용 가능

# 03. GAN의 구조와 원리

# 03. GAN의 구조와 원리

- GAN의 구조



# 03. GAN의 구조와 원리

- **학습에 사용되는 목적 함수 (objective function)**

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

원본 데이터 랜덤 vector

**랜덤 vector**

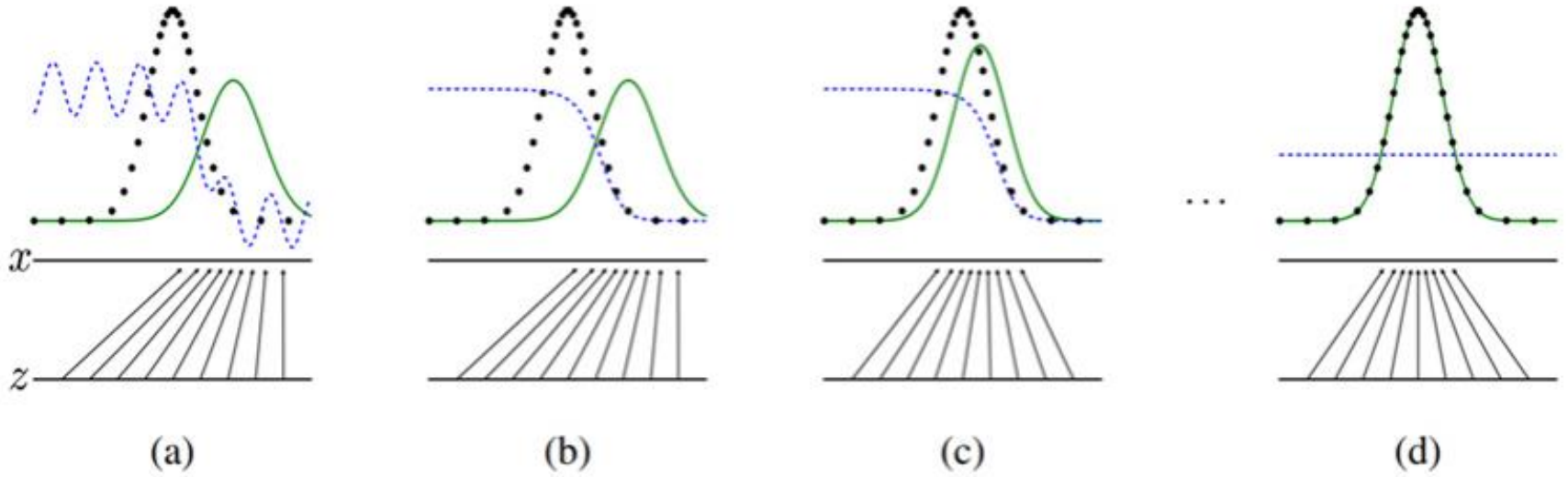
Generator : G(z) : 새로운 이미지 인스턴스 생성

Discriminator : D(input data) : 입력받은 이미지가 얼마나 진짜(Real) 같은지에 대한 확률 값 반환 [Real:1 ~ Fake:0]

# 03. GAN의 구조와 원리

- 공식의 목표

  $D(G(z))$ ➔ 1/2




※ 검은 점선: 원 데이터의 확률분포, 녹색 점선: GAN이 만들어 내는 확률분포, 파란 점선: 판별자의 확률분포

# 감사합니다