생성 모델
오토인코더

생성 모델

1 생성 모델이란

2 변형 오토인코더

생성 모델이란

13.1 생성 모델이란

생성 모델이란

- 생성 모델(generative model)은 주어진 데이터를 학습하여 데이터 분포를 따르는 유사한 데이터를 생성하는 모델
- **생성모델에서 가장 중요한 것은 학습 데이터의 분포를 배우는 것이 제일 중요**하다고 말할 수 있다. 학습 데이터의 분포와의 차이가 적을수록 실제 데이터와 비슷한 데이터를 생성할 수 있을 것이다.

[참고:https://wikidocs.net/151937]

13.1 생성 모델이란

생성 모델 개념

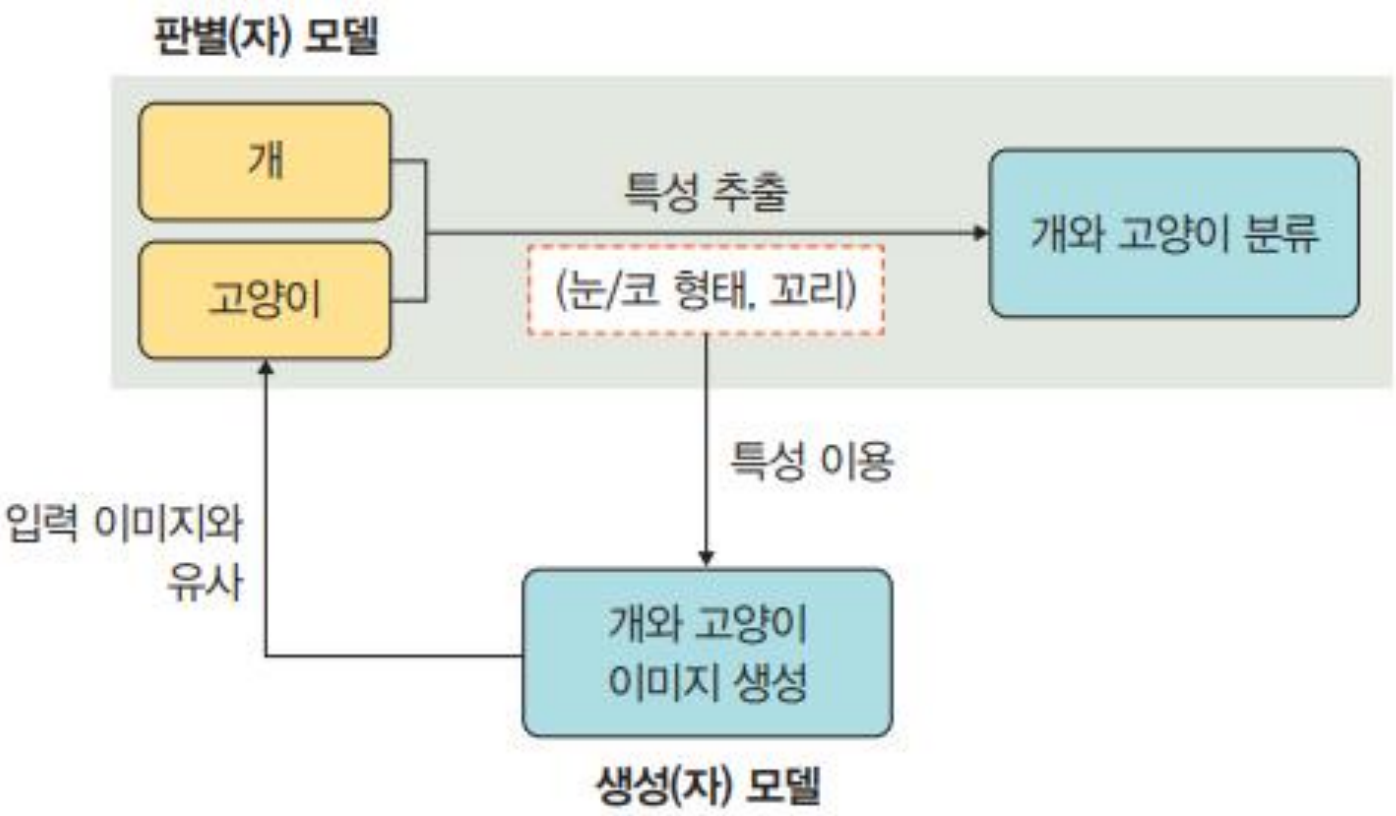
- 기존 합성곱 신경망에서 다룬 이미지 분류, 이미지 검출 등은 입력 이미지(x)가 있을 때 그에 따른 정답(y)을 찾는 것
- 이렇게 이미지를 분류하는 것을 '판별(자) 모델(discriminative model)'이라고 함
- 일반적으로 판별자 모델에서는 이미지를 정확히 분류(구별)하고자 해당 이미지를 대표하는 특성들을 잘 찾는 것을 목표로 함
- 예를 들어 개와 고양이를 구별하려면 개의 귀, 꼬리 등 특성을 찾는 것이 중요함

13.1 생성 모델이란

- **생성 모델 개념**
  - 판별자 모델에서 추출한 특성들의 조합을 이용하여 새로운 개와 고양이 이미지를 생성할 수 있는데, 이것을 '생성(자) 모델(generative model)'이라고 함
  - 즉, 생성 모델은 입력 이미지에 대한 데이터 분포 p(x)를 학습하여 새로운 이미지(새로운 이미지이면서 기존 이미지에서 특성을 추출했기 때문에 최대한 입력 이미지와 유사한 이미지)를 생성하는 것을 목표로 함

13.1 생성 모델이란

▼ 그림 13-1 생성 모델

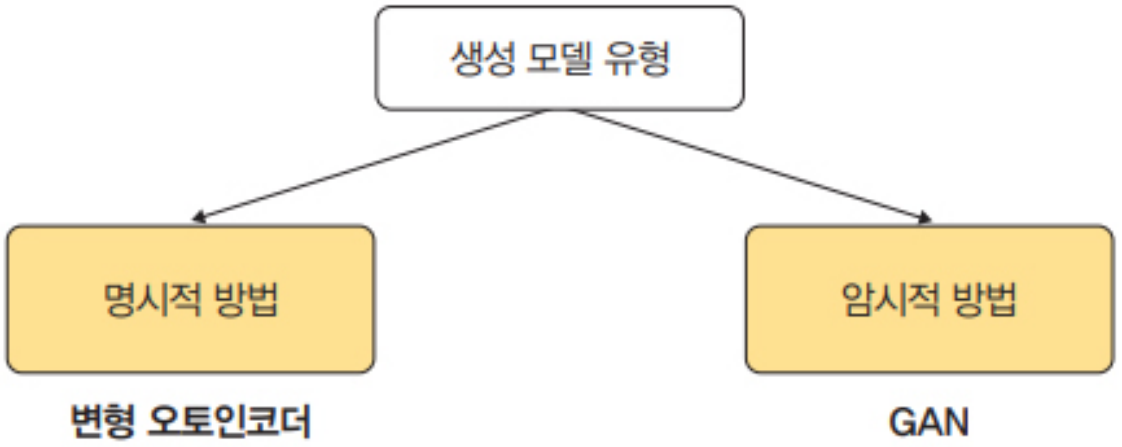


13.1 생성 모델이란

●생성 모델의 유형

- 생성 모델의 유형에는 다음 그림과 같이 모델의 확률 변수를 구하는 '변형 오토인코더 모델'과 확률 변수를 이용하지 않는 'GAN 모델'이 있음

▼ **그림 13-2 생성 모델의 유형**



13.1 생성 모델이란

- **생성 모델의 유형**
  - 생성 모델은 크게 명시적 방법(explicit density)과 암시적 방법(implicit density)으로 분류할 수 있음
  - 명시적 방법은 확률 변수 p(x)를 정의하여 사용
  - 대표적인 모델로 변형 오토인코더(variational autoencoder)가 있음
  - 암시적 방법은 확률 변수 p(x)에 대한 정의 없이 p(x)를 샘플링하여 사용
  - 대표적인 모델로 GAN(Generative Adversarial Network)이 있음

13.1 생성 모델이란

- **생성 모델의 유형**
  - 변형 오토인코더는 이미지의 잠재 공간(latent space)에서 샘플링하여 완전히 새로운 이미지나 기존 이미지를 변형하는 방식으로 학습을 진행
  - GAN은 생성자와 판별자가 서로 경쟁하면서 가짜 이미지를 진짜 이미지와 최대한 비슷하게 만들도록 학습을 진행

13.2 변형 오토인코더

13.2 변형 오토인코더

- **변형 오토인코더**
  - 변형 오토인코더는 오토인코더의 확장
  - 오토인코더가 무엇인지 확인한 후 변형 오토인코더 학습

EdgesCats Demo pix2pix-tensorflow

https://affinelayer.com/pixsrv/

13.2 변형 오토인코더

- **오토인코더란**
  - 오토인코더는 단순히 입력을 출력으로 복사하는 신경망으로 은닉층(혹은 병목층이라고도 함)의 노드 수가 입력 값보다 적은 것이 특징
  - 입력과 출력이 동일한 이미지라고 예상할 수 있음
  - 왜 입력을 출력으로 복사하는 방법을 사용할까?
  - 바로 은닉층 때문임
  - 오토인코더의 병목층은 입력과 출력의 뉴런보다 훨씬 적음
  - 즉, 적은 수의 병목층 뉴런으로 데이터를 가장 잘 표현할 수 있는 방법이 오토인코더

13.2 변형 오토인코더

● 오토인코더란

- 오토인코더는 네 가지 주요 부분으로 구성

1. **인코더**: 인지 네트워크(recognition network)라고도 하며, 특성에 대한 학습을 수행하는 부분
2. **병목층(은닉층)**: 모델의 뉴런 개수가 최소인 계층

   이 계층에서는 차원이 가장 낮은 입력 데이터의 압축 표현이 포함
3. **디코더**: 생성 네트워크(generative network)라고도 하며, 이 부분은 병목층에서 압축된 데이터를 원래대로 재구성(reconstruction)하는 역할을 함

   즉, 최대한 입력에 가까운 출력을 생성하도록 함

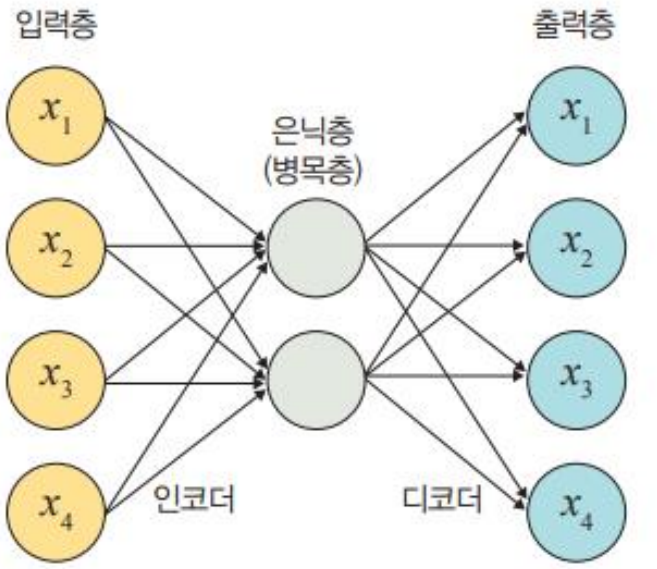
13.2 변형 오토인코더

●오토인코더란

4. 손실 재구성: 오토인코더는 다음 그림과 같이 입력층과 출력층의 뉴런 개수가 동일하다는것만 제외하면 일반적인 다층 퍼셉트론(Multi-Layer Perceptron, MLP)과 구조가 동일

오토인코더는 압축된 입력을 출력층에서 재구성하며, 손실 함수는 입력과 출력(인코더와 디코더)의 차이를 가지고 계산

▼ **그림 13-3 오토인코더**



오토인코더 (스택 오토인코더)

```
import torch
import torchvision
from torchvision import transforms
import torch.nn.functional as F
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
```

오토인코더 (스택 오토인코더)

```
# CPU/GPU
device = torch.device("cuda:0" if
torch.cuda.is_available() else "cpu")
print(f'{device} is available.’)


dataset =
torchvision.datasets.MNIST('./data/',
download=True, train=True,
transform=transforms.ToTensor())
trainloader =
torch.utils.data.DataLoader(dataset,batch_s
ize=50,shuffle=True)
```

```
dataset = torchvision.datasets.MNIST('./data/', download=True, train=True, transform=transforms.ToTensor())
trainloader = torch.utils.data.DataLoader(dataset,batch_size=50,shuffle=True)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 81198437.67it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 102570443.54it/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 24152388.02it/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 18768993.86it/s]Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

[16]
```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
                        nn.Linear(784, 128),
                        nn.ReLU(),
                        nn.Linear(128, 32),
                        nn.ReLU(),
                        nn.Linear(32, 10),
                        nn.ReLU())
        self.decoder = nn.Sequential(
                        nn.Linear(10, 32),
                        nn.ReLU(),
                        nn.Linear(32, 128),
                        nn.ReLU(),
                        nn.Linear(128, 28*28)
                        #nn.Sigmoid()
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

```
model = Autoencoder().to(device)
def normalize_output(img):
    img = (img - img.min())/(img.max()-img.min())
    return img
def check_plot():
    with torch.no_grad():
        for data in trainloader:

            inputs = data[0].to(device)
            outputs = model(inputs.view(-1,28*28))
            outputs = outputs.view(-1,1,28,28)

            input_samples = inputs.permute(0,2,3,1).cpu().numpy()
# 원래 이미지
            reconstructed_samples = outputs.permute(0,2,3,1).cpu().numpy()
# 생성 이미지
            break
```

```
columns = 10 # 시각화 전체 너비
rows = 5 # 시각화 전체 높이

fig=plt.figure(figsize=(columns, rows)) # figure 선언

for i in range(1, columns*rows+1):
    img = input_samples[i-1]
    fig.add_subplot(rows, columns, i)
    plt.imshow(img.squeeze())
    plt.axis('off')
plt.show()
plt.close()

fig=plt.figure(figsize=(columns, rows))

for i in range(1, columns*rows+1):
    img = reconstructed_samples[i-1]
    fig.add_subplot(rows, columns, i)
    plt.imshow(img.squeeze())
    plt.axis('off')
plt.show()
```

```
criterion = nn.MSELoss() # MSE 사용
optimizer = optim.Adam(model.parameters(),
lr=1e-4)
```

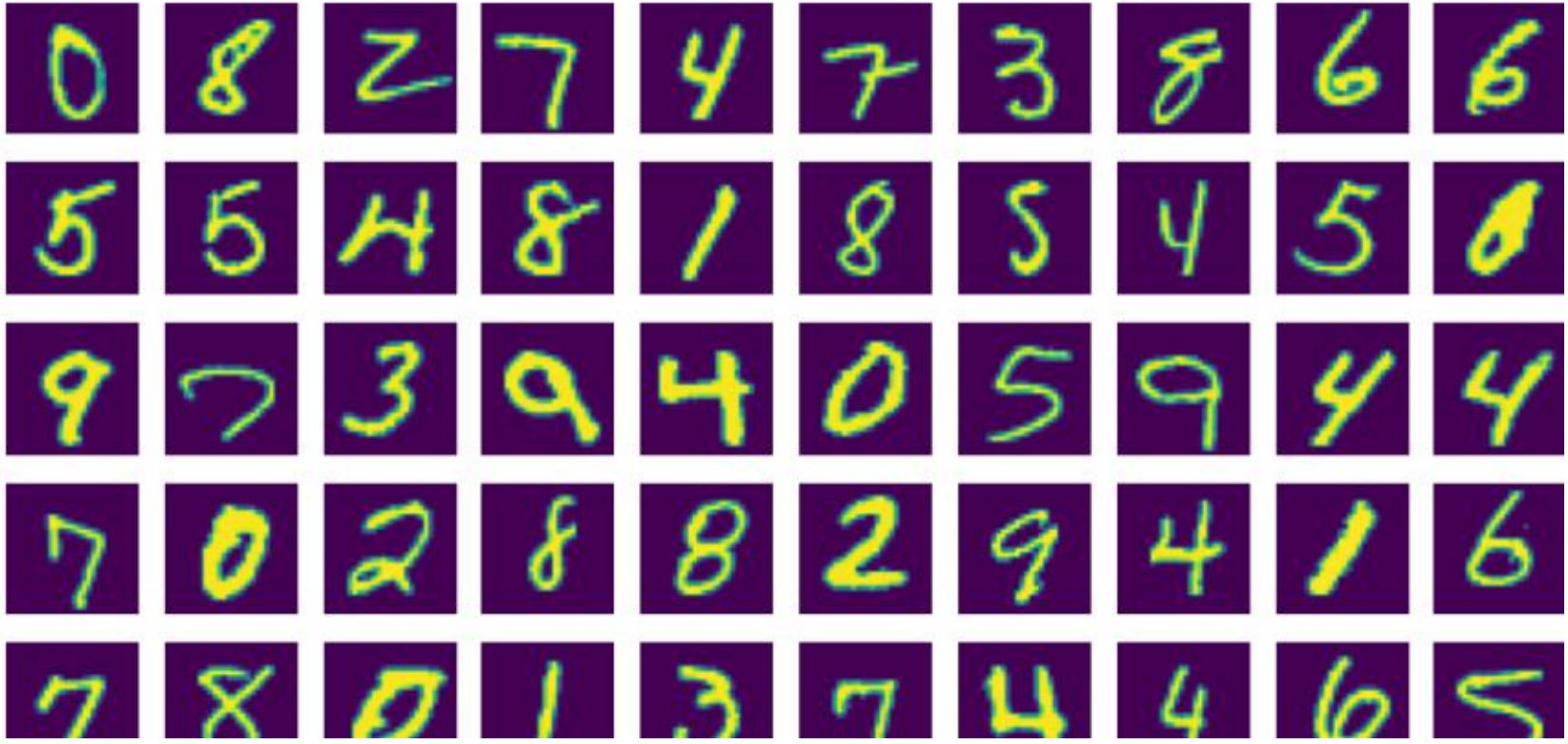
```
for epoch in range(51):

    running_loss = 0.0
    for data in trainloader:
        inputs = data[0].to(device)
        optimizer.zero_grad()
        outputs = model(inputs.view(-1,28*28))
        outputs = outputs.view(-1,1,28,28)
        loss = criterion(inputs, outputs)
# 라벨 대신 입력 이미지와 출력 이미지를 비교
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    cost = running_loss / len(trainloader)
    print('[%d] loss: %.3f' %(epoch + 1, cost))
```

9분 [17]
```
[43] loss: 0.023
[44] loss: 0.023
[45] loss: 0.023
[46] loss: 0.023
[47] loss: 0.023
[48] loss: 0.023
[49] loss: 0.023
[50] loss: 0.023
[51] loss: 0.023
```

3초
```
check_plot()
```



감사합니다.