9. Autoencoder

심층 학습(Deep Learning)에 대한 개념이 정립되기 전에는 신경망이 다층인 경우 학습이 제대로 이루어지지 않는 문제가 있었다. 그러던 와중, 2006년에 제프리 힌튼[Geoffrey Hinton] 교수가 오토인코더(AutoEncoder)라는 혁명적인 기계 학습 방식을 제안하였다. 오토인코더는 신경망의 각 층을 단계적으로 학습해나가, 최종 출력이 최초 입력을 재현하도록 하는 것이 주된 특징이다.

입력과 출력 층의 차원(노드의 개수)은 동일하되, 은닉 층은 입력/출력 층보다 차원이 낮아야 한다. 다시 말해, 은닉 층의 노드 개수가 입력/출력 층보다 적다는 것이다. (은닉 층의 차원이 입력/출력 층과 동일하다면, 입력받은 데이터를 그대로 출력해버리면 그만이기 때문에, 아무런 의미가 없는 신경망이 되어버리고 말 것임.) 은닉 층의 차원이 더 낮기 때문에, 신경망은 입력 데이터들을 압축하여 이들로부터 특징을 추출하고, 추출한 특징을 기반으로 입력을 최대한으로 재현한 출력 데이터를 내놓게 된다.

사람이 컴퓨터에게 입력 데이터에 대한 정보를 주지 않는 오토인코더는 오늘날 대표적인 비지도 학습(Unsupervised Learning) 방식으로, 많은 사람들에 의해 계속해서 연구되고 있으며, 학습에 있어서 제일 번거로운 과정인 지도가 필요 없다는 점에서, 오토인코더의 잠재력은 무궁무진 하다고 할 수 있다.

가. Autoencoder란?

오토인코더(Autoencoder)는 아래의 그림 9.1과 같이 단순히 입력을 출력으로 복사하는 신경망이다. 어떻게 보면 간단한 신경망처럼 보이지만 네트워크에 여러 가지 방법으로 제약을 줌으로써 어려운 신경망으로 만든다. 예를 들어 아래 그림처럼 hidden layer의 뉴런 수를 input layer(입력층) 보다 작게 해서 데이터를 압축(차원을 축소)한다거나, 입력 데이터에 노이즈(noise)를 추가한 후 원본 입력을 복원할 수 있도록 네트워크를 학습시키는 등 다양한 오토인코더가 있다. 이러한 제약들은 오토인코더가 단순히 입력을 바로 출력으로 복사하지 못하도록 방지하며, 데이터를 효율적으로 표현(representation)하는 방법을 학습하도록 제어한다.

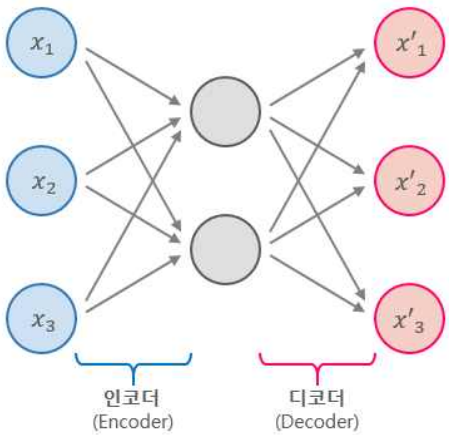

그림 9.1 Autoencoder의 기본 개념

나. Uncomplete Autoencoder

오토인코더는 위의 그림 9.1에서 볼 수 있듯이 항상 인코더(encoder)와 디코더(decoder), 두 부분으로 구성되어 있다.

- 인코더(encoder) : 인지 네트워크(recognition network)라고도 하며, 입력을 내부 표현으로 변환한다.
- 디코더(decoder) : 생성 네트워크(generative nework)라고도 하며, 내부 표현을 출력으로 변환한다.

오토인코더는 위의 그림 9.1에서 처럼, 입력과 출력층의 뉴런 수가 동일하다는 것만 제외하면 일반적인 MLP(Multi-Layer Perceptron)과 동일한 구조이다. 오토인코더는 입력을 재구성하기 때문에 출력을 재구성(reconstruction)이라고도 하며, 손실함수는 입력과 재구성(출력)의 차이를 가지고 계산한다.

위 그림 9.1의 오토인토더는 히든 레이어의 뉴런(노드, 유닛)이 입력층보다 작으므로 입력이 저차원으로 표현되는데, 이러한 오토인코더를 Undercomplete Autoencoder라고 한다. undercomplete 오토인코더는 저차원을 가지는 히든 레이어에 의해 입력을 그대로 출력으로 복사할 수 없기 때문에, 출력이 입력과 같은 것을 출력하기 위해 학습해야 한다. 이러한 학습을 통해 undercomplete 오토인코더는 입력 데이터에서 가장 중요한 특성(feature)을 학습하도록 만든다.

다. Stacked Autoencoder

Stacked 오토인코더 또는 deep 오토인코더는 여러개의 히든 레이어를 가지는 오토인코더이며, 레이어를 추가할수록 오토인코더가 더 복잡한 코딩(부호화)을 학습할 수 있다. stacked 오토인코더의 구조는 아래의 그림과 같이 가운데 히든레이어(코딩층)을 기준으로 대칭인 구조를 가진다.

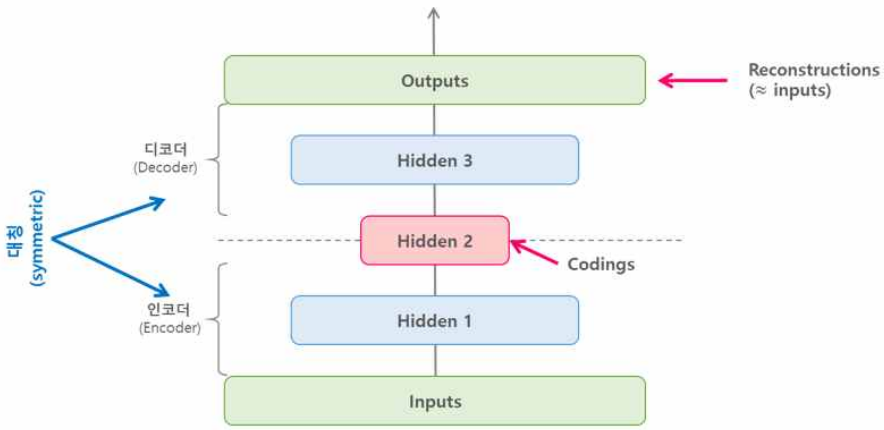

그림 9.2 Stacked Autoencoder의 기본 개념

라. Variational Autoencoder(VAE)

VAE(Variational AutoEncoder)는 2014년 D.Kingma와 M.Welling이 Auto-Encoding Variational Bayes 논문에서 제안한 오토인코더의 한 종류이다. VAE는 위에서 살펴본 오터인코더와는 다음과 같은 다른점이 있다.

- VAE는 확률적 오토인코더(probabilistic autoencoder)다. 즉, 학습이 끝난 후에도 출력이 부분적으로 우연에 의해 결정된다.
- VAE는 생성 오토인코더(generatie autoencoder)이며, 학습 데이터셋에서 샘플링된 것과 같은 새로운 샘플을 생성할 수 있다.

VAE의 구조는 아래의 그림과 같다.

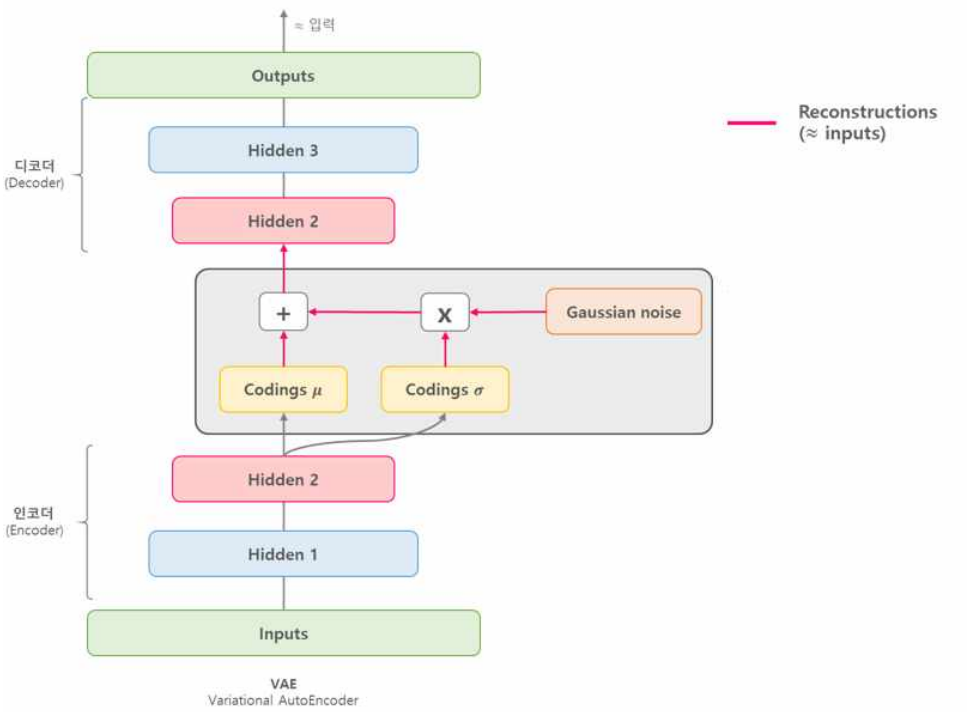

그림 9.3 Variational Autoencoder의 기본 개념

VAE의 코딩층은 다른 오토인코더와는 다른 부분이 있는데 주어진 입력에 대해 바로 코딩을 만드는 것이 아니라, 인코더(encoder)는 평균 코딩 μ와 표준편차 코딩 σ 을 만든다. 실제 코딩은 평균이 μ이고 표준편차가 σ인 가우시안 분포(gaussian distribution)에서 랜덤하게 샘플링되며, 이렇게 샘플링된 코딩을 디코더(decoder)가 원본 입력으로 재구성하게 된다.

VAE는 마치 가우시안 분포에서 샘플링된 것처럼 보이는 코딩을 만드는 경향이 있는데, 학습하는 동안 손실함수가 코딩(coding)을 가우시안 샘플들의 집합처럼 보이는 형태를 가진 코딩 공간(coding space) 또는 잠재 변수 공간(latent space)로 이동시키기 때문이다. 이러한 이유로 VAE는 학습이 끝난 후에 새로운 샘플을 가우시안 분포로 부터 랜덤한 코딩을 샘플링해 디코딩해서 생성할 수 있다.

마. Autoencode 구현

1) 데이터 load

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)
```

2) 하이퍼파라미터 정의

```
learning_rate = 0.01
learning_rate = 0.01
training_epoch = 20
batch_size = 100
# 신경망 레이어 구성 옵션
n_hidden = 256  # 히든 레이어의 뉴런 갯수
n_input = 28*28   # 입력값 크기 - 이미지 픽셀수
```

3) 신경망 모델 구성

```
# Y 가 없습니다. 입력값을 Y로 사용하기 때문입니다.
X = tf.placeholder(tf.float32, [None, n_input])
# 인코더 레이어와 디코더 레이어의 가중치와 편향 변수를 설정합니다.
# 다음과 같이 이어지는 레이어를 구성하기 위한 값들 입니다.
# input -> encode -> decode -> output
W_encode = tf.Variable(tf.random_normal([n_input, n_hidden]))
b_encode = tf.Variable(tf.random_normal([n_hidden]))
# sigmoid 함수를 이용해 신경망 레이어를 구성합니다.
# sigmoid(X * W + b)
# 인코더 레이어 구성
encoder = tf.nn.sigmoid(tf.add(tf.matmul(X, W_encode), b_encode))
# encode 의 아웃풋 크기를 입력값보다 작은 크기로 만들어 정보를 압축하여 특
# 성을 뽑아내고, decode 의 출력을 입력값과 동일한 크기를 갖도록하여 입력과
# 똑같은 아웃풋을 만들어 내도록 합니다. 히든 레이어의 구성과 특성치을 뽑아내
# 는 알고리즘을 변경하여 다양한 오토인코더를 만들 수 있다.
W_decode = tf.Variable(tf.random_normal([n_hidden, n_input]))
b_decode = tf.Variable(tf.random_normal([n_input]))
# 디코더 레이어 구성, 이 디코더가 최종 모델
decoder = tf.nn.sigmoid(tf.add(tf.matmul(encoder, W_decode), b_decode))
```

4) 손실함수 정의 및 최적화

```
# 디코더는 인풋과 최대한 같은 결과를 내야 하므로, 디코딩한 결과를 평가하기
# 위해 입력 값인 X 값을 평가를 위한 실측 결과 값으로하여 decoder 와의 차이
# 를 손실값으로 설정
cost = tf.reduce_mean(tf.pow(X - decoder, 2))
optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(cost)
```

5) 신경망 모델 학습

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

total_batch = int(mnist.train.num_examples/batch_size)
for epoch in range(training_epoch):
    total_cost = 0
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        _, cost_val = sess.run([optimizer, cost], feed_dict={X: batch_xs})
        total_cost += cost_val
    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.4f}'.format(total_cost / total_batch))

print('최적화 완료!')
```

6) 결과확인

```
# 입력값(위쪽)과 모델이 생성한 값(아래쪽)을 시각적으로 비교
sample_size = 10
samples = sess.run(decoder,
                   feed_dict={X: mnist.test.images[:sample_size]})

fig, ax = plt.subplots(2, sample_size, figsize=(sample_size, 2))

for i in range(sample_size):
    ax[0][i].set_axis_off()
    ax[1][i].set_axis_off()
    ax[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
    ax[1][i].imshow(np.reshape(samples[i], (28, 28)))
plt.show()
```

< Example 9-1 > Autoencoder 모델을 이용한 MNIST 숫자 합성

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

learning_rate = 0.01
training_epoch = 20
batch_size = 100
n_hidden = 256  # 히든 레이어의 뉴런 갯수
n_input = 28*28   # 입력값 크기 - 이미지 픽셀수

#신경망 모델 구성
X = tf.placeholder(tf.float32, [None, n_input])
W_encode = tf.Variable(tf.random_normal([n_input, n_hidden]))
b_encode = tf.Variable(tf.random_normal([n_hidden]))
encoder = tf.nn.sigmoid(tf.add(tf.matmul(X, W_encode), b_encode))
W_decode = tf.Variable(tf.random_normal([n_hidden, n_input]))
b_decode = tf.Variable(tf.random_normal([n_input]))
decoder = tf.nn.sigmoid(tf.add(tf.matmul(encoder, W_decode), b_decode))

#손실함수 및 최적화
cost = tf.reduce_mean(tf.pow(X - decoder, 2))
optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(cost)

#신경망 모델 학습
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
total_batch = int(mnist.train.num_examples/batch_size)

for epoch in range(training_epoch):
    total_cost = 0
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        _, cost_val = sess.run([optimizer, cost], feed_dict={X: batch_xs})
        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.4f}'.format(total_cost / total_batch))
print('최적화 완료!')

#결과 확인
sample_size = 10
samples = sess.run(decoder,
                   feed_dict={X: mnist.test.images[:sample_size]})
fig, ax = plt.subplots(2, sample_size, figsize=(sample_size, 2))
for i in range(sample_size):
    ax[0][i].set_axis_off()
    ax[1][i].set_axis_off()
    ax[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
    ax[1][i].imshow(np.reshape(samples[i], (28, 28)))
plt.show()
```

< Example 9-2 > tf.layers API를 이용한 Autoencoder MNIST 숫자 합성

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

learning_rate = 0.01
training_epoch = 20
batch_size = 100
n_input = 28*28   # 입력값 크기 - 이미지 픽셀수
n_output = 50*50
#신경망 모델 구성
X = tf.placeholder(tf.float32, [None, n_input])
L1_encoder = tf.layers.dense(inputs=X, units=512, activation=tf.nn.relu)
L2_encoder = tf.layers.dense(inputs=L1_encoder, units=256, activation=tf.nn.relu)
L3_encoder = tf.layers.dense(inputs=L2_encoder, units=128, activation=tf.nn.relu)
Latent     = tf.layers.dense(inputs=L3_encoder, units=64, activation=tf.nn.relu)
L3_decoder = tf.layers.dense(inputs=Latent, units=128, activation=tf.nn.relu)
L2_decoder = tf.layers.dense(inputs=L3_decoder, units=256, activation=tf.nn.relu)
L1_decoder = tf.layers.dense(inputs=L2_decoder, units=512, activation=tf.nn.relu)
Output = tf.layers.dense(inputs=L1_decoder, units=n_input, activation=tf.nn.sigmoid)

#cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=Output))
cost = tf.reduce_mean(tf.pow(X - Output, 2))
optimizer = tf.train.AdamOptimizer(0.001).minimize(cost)

#신경망 모델 학습
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
total_batch = int(mnist.train.num_examples / batch_size)

for epoch in range(training_epoch):
    total_cost = 0
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        _, cost_val = sess.run([optimizer, cost], feed_dict={X: batch_xs})
        total_cost += cost_val

    print('Epoch:', '%04d' % (epoch + 1),
          'Avg. cost =', '{:.4f}'.format(total_cost / total_batch))
print('최적화 완료!')

# 결과 확인 : 입력값(위쪽)과 모델이 생성한 값(아래쪽)을 시각적으로 비교
sample_size = 10
samples = sess.run(Output, feed_dict={X: mnist.test.images[:sample_size]})
fig, ax = plt.subplots(2, sample_size, figsize=(sample_size, 2))
for i in range(sample_size):
    ax[0][i].set_axis_off()
    ax[1][i].set_axis_off()
    ax[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
    ax[1][i].imshow(np.reshape(samples[i], (28, 28)))
plt.show()
```

< Example 9-3 > Convolutional Autoencoder MNIST 숫자 합성

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)

learning_rate = 0.001
#신경망 모델 구성
inputs_ = tf.placeholder(tf.float32, (None, 28, 28, 1), name='inputs')
targets_ = tf.placeholder(tf.float32, (None, 28, 28, 1), name='targets')
### Encoder
conv1 = tf.layers.conv2d(inputs=inputs_, filters=32, kernel_size=(3,3),
padding='same', activation=tf.nn.relu) # Now 28x28x32
maxpool1 = tf.layers.max_pooling2d(conv1, pool_size=(2,2), strides=(2,2),
padding='same')  # Now 14x14x32
conv2 = tf.layers.conv2d(inputs=maxpool1, filters=32, kernel_size=(3,3),
padding='same', activation=tf.nn.relu) # Now 14x14x32
maxpool2 = tf.layers.max_pooling2d(conv2, pool_size=(2,2), strides=(2,2),
padding='same') # Now 7x7x32
conv3 = tf.layers.conv2d(inputs=maxpool2, filters=16, kernel_size=(3,3),
padding='same', activation=tf.nn.relu) # Now 7x7x16
encoded = tf.layers.max_pooling2d(conv3, pool_size=(2,2), strides=(2,2),
padding='same') # Now 4x4x16
### Decoder
upsample1 = tf.image.resize_images(encoded, size=(7,7),
method=tf.image.ResizeMethod.NEAREST_NEIGHBOR) # Now 7x7x16
conv4 = tf.layers.conv2d(inputs=upsample1, filters=16, kernel_size=(3,3),
padding='same', activation=tf.nn.relu) # Now 7x7x16
upsample2 = tf.image.resize_images(conv4, size=(14,14),
method=tf.image.ResizeMethod.NEAREST_NEIGHBOR) # Now 14x14x16
conv5 = tf.layers.conv2d(inputs=upsample2, filters=32, kernel_size=(3,3),
padding='same', activation=tf.nn.relu) # Now 14x14x32
upsample3 = tf.image.resize_images(conv5, size=(28,28),
method=tf.image.ResizeMethod.NEAREST_NEIGHBOR) # Now 28x28x32
conv6 = tf.layers.conv2d(inputs=upsample3, filters=32, kernel_size=(3,3),
padding='same', activation=tf.nn.relu) # Now 28x28x32
logits = tf.layers.conv2d(inputs=conv6, filters=1, kernel_size=(3,3), padding='same',
activation=None) #Now 28x28x1
# Pass logits through sigmoid to get reconstructed image
decoded = tf.nn.sigmoid(logits)
# Pass logits through sigmoid and calculate the cross-entropy loss
loss = tf.nn.sigmoid_cross_entropy_with_logits(labels=targets_, logits=logits)
# Get cost and define the optimizer
cost = tf.reduce_mean(loss)
opt = tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

< Example 9-3 > Convolutional Autoencoder MNIST 숫자 합성(계속)

```
sess = tf.Session()
epochs = 20
batch_size = 100
# Set's how much noise we're adding to the MNIST images
noise_factor = 0.5
sess.run(tf.global_variables_initializer())
for e in range(epochs):
    print(e)
    for ii in range(mnist.train.num_examples // batch_size):
        batch = mnist.train.next_batch(batch_size)
        # Get images from the batch
        imgs = batch[0].reshape((-1, 28, 28, 1))

        # Add random noise to the input images
        noisy_imgs = imgs + noise_factor * np.random.randn(*imgs.shape)
        # Clip the images to be between 0 and 1
        noisy_imgs = np.clip(noisy_imgs, 0., 1.)

        # Noisy images as inputs, original images as targets
        batch_cost, _ = sess.run([cost, opt], feed_dict={inputs_: noisy_imgs,
                                                         targets_: imgs})
print("Epoch: {}/{}...".format(e + 1, epochs),
      "Training loss: {:.4f}".format(batch_cost))

sample_size = 10
tmp = mnist.test.images[:sample_size]
images = tmp.reshape((-1, 28, 28, 1))
result = sess.run(decoded, feed_dict={inputs_: images})

fig, ax = plt.subplots(2, sample_size, figsize=(sample_size, 2))
for i in range(sample_size):
    ax[0][i].set_axis_off()
    ax[1][i].set_axis_off()
    ax[0][i].imshow(np.reshape(images[i], [28,28]))
    ax[1][i].imshow(np.reshape(result[i], [28,28]))

plt.show()
```