Mnist Tensorflow 예제 및 소스

2018.07.31

작성자	검토자	승인자

목 차

1. Mnist 데이터

- 손으로 필기한 숫자 각각의 이미지가 정수로 표시되는 데이터베이스
- 머신러닝 알고리즘의 성능을 벤치마킹하는 데 사용되며 99.7% 이상의 정확도 달성
- 55000개의 학습 데이터와 10000개의 테스트 데이터, 5000개의 검증 데이터
- 학습 데이터는 28 X 28 사이즈에 총 784개의 픽셀로 이루어진 흑백 이미지
- 컴퓨터는 흑색 이미지를 표현할 때 픽셀의 숫자가 0에 가까울수록 검정색으로 255에 가까울수록 하얀색으로 표현

1.1 Mnist 데이터 확인

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

# Training Data 로드
data_file = open('./MNIST_data/mnist_train.csv', 'r')

# Training Data 파일의 내용을 한줄씩 불러와서 문자열 리스트로 반환
training_data = data_file.readlines()

# Training Data 의 두번째 데이터 확인
print(training_data[1])

# Training Data 의 두번째 데이터를 ','로 분리
training_data_array = np.asfarray(training_data[1].split(","))

# 일렬로 늘어진 784 개의 픽셀 정보를 28X28 행렬로 변환
matrix = training_data_array[1:].reshape(28,28)

# 회색으로 Training Data 의 두번째 데이터 숫자 확인
plt.imshow(matrix, cmap='gray')
plt.show()
```

<결과>

Training Data의 두번째 데이터 확인

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,51,159,253,159,50,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,48,238,252,252,252,237,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,54,227,253,252,239,233,252,57,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,10,60,224,252,253,252,202,84,252,253,122,0

,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,163,252,252,252,253,252,252,96,189,253,167,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,51,238,253,

253,190,114,253,228,47,79,255,168,0,0,0,0,0,0,0,0,0,0,0,0,0,0,48,238,252,252,179,12,75,121,21,0,0,253,243,50,0,0,

0,0,0,0,0,0,0,0,0,0,38,165,253,233,208,84,0,0,0,0,0,0,253,252,165,0,0,0,0,0,0,0,0,0,0,0,0,7,178,252,240,71,19,28,0,0,

0,0,0,0,253,252,195,0,0,0,0,0,0,0,0,0,0,0,0,57,252,252,63,0,0,0,0,0,0,0,0,0,0,253,252,195,0,0,0,0,0,0,0,0,0,0,0,0,198,253,

190,0,0,0,0,0,0,0,0,0,0,255,253,196,0,0,0,0,0,0,0,0,0,0,0,76,246,252,112,0,0,0,0,0,0,0,0,0,0,253,252,148,0,0,0,0,0,0,0,0,0

,0,0,0,85,252,230,25,0,0,0,0,0,0,0,0,7,135,253,186,12,0,0,0,0,0,0,0,0,0,0,0,85,252,223,0,0,0,0,0,0,0,0,7,131,252,225,71

,0,0,0,0,0,0,0,0,0,0,0,0,85,252,145,0,0,0,0,0,0,0,48,165,252,173,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,86,253,225,0,0,0,0,0,0,114,2

38,253,162,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,85,252,249,146,48,29,85,178,225,253,223,167,56,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

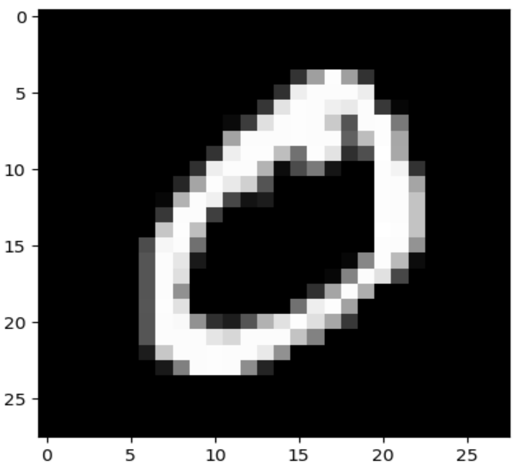
85,252,252,252,229,215,252,252,252,196,130,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,28,199,252,252,253,252,252,233,145,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,25,128,252,253,252,141,37,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Training Data의 두번째 데이터 숫자 확인



2. Mnist Tensorflow 예제

2.1 Softmax Logistic Regression

- 단일 Layer의 Logistic Regression에다가 Softmax를 붙여서 0~9사이의 숫자로 분류
- Softmax Regression : Classfication 알고리즘 중 하나로, 들어온 값이 어떤 분류인지 구분해주는 알고리즘으로 로지스틱 회귀는 두 가지만 분류가 가능하지만 소프트맥스 회귀는 n 개의 분류로 구분이 가능

<코드>

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

# 데이터 로드
mnist = input_data.read_data_sets('/tmp/tensorflow/mnist/input_data', one_hot=True)

# 모델 정의
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)

# 크로스 엔트로피와 옵티마이저 정의
learning_rate = 0.5
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step =
tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cross_entropy)

# 세션 초기화
sess = tf.Session()
init = tf.global_variables_initializer()    # .run()
sess.run(init)
```

```
# 트레이닝
print("Training....")
for i in range(1000):
    # 1000 번씩, 전체 데이타에서 100 개씩 뽑아서 트레이닝을 함.
    batch_x, batch_y = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_x, y_: batch_y})

# 모델 검증
print("Testing Model")
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('accuracy', sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
print('Finish')
```

<설명>

1) 모델 정의

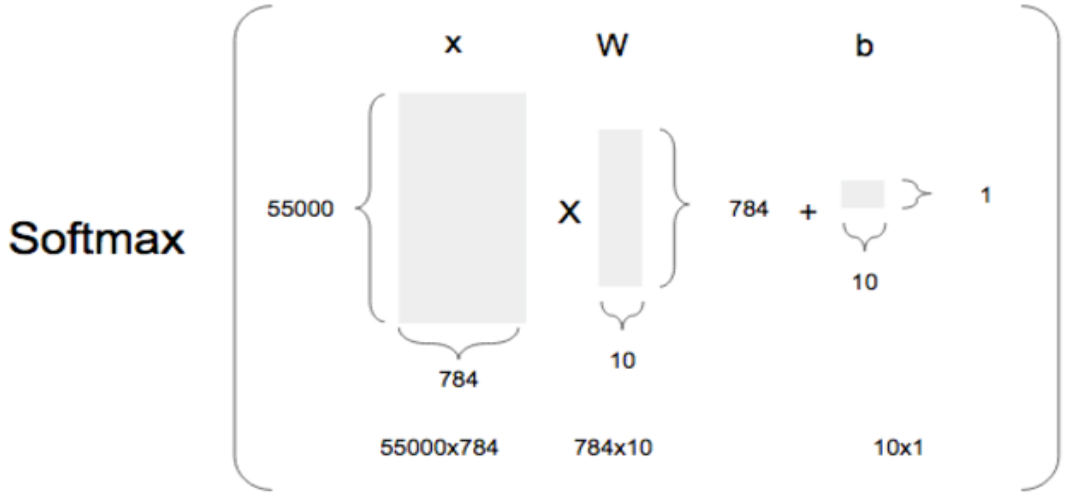
Placeholder는 실행할 때 우리가 값을 넣을 수 있음

x는 input image, 784 = 28*28

y_는 class label, mnist가 0~9까지 이미지이르모 10개, one-hot 벡터

w는 가중치, b는 bias

y는 output nodes의 엑티베이션 값, wx+b 값을 소프트맥스 함수에 넣음



2) 크로스 엔트로피와 옵티마이저 정의

learning_rate는 학습률로 0.5

cross_entropy는 트레이닝 과정에서 최소화해야 하는 값으로

Gradient Desent 기법을 사용하여 최적화 한다

3) 트레이닝

100 크기의 mini-batch로 1000번 학습

4) 모델 검증

argmax는 tensor에서 max 값을 찾는 함수

correct_prediction 은 예측한 라벨과 정답 라벨을 비교

accuracy 는 비교한 결과의 평균을 낸다

<출력>

Training....

Testing Model

accuracy 0.918

Finish

2.2 ANN(Artificial Neural Network)

<코드>

```
import tensorflow as tf

# MNIST 데이터 로드
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

# 학습을 위한 설정값 정의.
learning_rate = 0.001   # 학습률
num_epochs = 30         # 학습횟수
batch_size = 256        # 배치개수
display_step = 1        # 손실함수 출력 주기
input_size = 784        # 28 * 28
hidden1_size = 256      # 첫 번째 은닉층 크기
hidden2_size = 256      # 두 번째 은닉층 크기
output_size = 10        # 출력 개수(0~9 10 개)

# 입력값과 출력값을 받기 위한 플레이스홀더를 정의
x = tf.placeholder(tf.float32, shape=[None, input_size])
y = tf.placeholder(tf.float32, shape=[None, output_size])

# ANN 모델 정의
def build_ANN(x):
    W1 = tf.Variable(tf.random_normal(shape=[input_size, hidden1_size]))
    b1 = tf.Variable(tf.random_normal(shape=[hidden1_size]))
    H1_output = tf.nn.relu(tf.matmul(x,W1) + b1)
    W2 = tf.Variable(tf.random_normal(shape=[hidden1_size, hidden2_size]))
    b2 = tf.Variable(tf.random_normal(shape=[hidden2_size]))
    H2_output = tf.nn.relu(tf.matmul(H1_output,W2) + b2)
    W_output = tf.Variable(tf.random_normal(shape=[hidden2_size, output_size]))
    b_output = tf.Variable(tf.random_normal(shape=[output_size]))
    logits = tf.matmul(H2_output,W_output) + b_output
    return logits
```

```
# ANN 모델을 선언
predicted_value = build_ANN(x)

# 손실함수와 옵티마이저를 정의
# tf.nn.softmax_cross_entropy_with_logits 함수를 이용하여 활성함수를 적용하지 않은 output layer 의 결과값(logits)에 softmax 함수를 적용합니다.
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=predicted_value, labels=y))
train_step = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

# 세션을 열고 그래프를 실행합니다.
with tf.Session() as sess:
    # 변수들에 초기값을 할당합니다.
    sess.run(tf.global_variables_initializer())

    # 지정된 횟수만큼 최적화를 수행합니다.
    for epoch in range(num_epochs):
        average_loss = 0.
        # 전체 배치를 불러옵니다.
        total_batch = int(mnist.train.num_examples/batch_size)
        # 모든 배치들에 대해서 최적화를 수행합니다.
        for i in range(total_batch):
            batch_x, batch_y = mnist.train.next_batch(batch_size)
            # 옵티마이저를 실행해서 파라마터들을 업데이트합니다.
            _, current_loss = sess.run([train_step, loss], feed_dict={x: batch_x, y: batch_y})
            # 평균 손실을 측정합니다.
            average_loss += current_loss / total_batch
        # 지정된 epoch 마다 학습결과를 출력합니다.
        if epoch % display_step == 0:
            print("반복(Epoch): %d, 손실 함수(Loss): %f" % ((epoch+1), average_loss))
```

```
  # 테스트 데이터를 이용해서 학습된 모델이 얼마나 정확한지 정확도를
출력합니다.
  correct_prediction = tf.equal(tf.argmax(predicted_value, 1), tf.argmax(y, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
  print("정확도(Accuracy): %f" % (accuracy.eval(feed_dict={x: mnist.test.images, y:
mnist.test.labels}))) # 정확도: 약 94%
```

<설명>

1) 학습을 위한 설정값 정의

```
learning_rate = 0.001   # 학습률
num_epochs = 30         # 학습횟수
batch_size = 256        # 배치개수
display_step = 1        # 손실함수 출력 주기
input_size = 784        # 28 * 28
hidden1_size = 256      # 첫 번째 은닉층 크기
hidden2_size = 256      # 두 번째 은닉층 크기
output_size = 10        # 출력 개수(0~9 10개)
```

2) ANN 모델 정의

- 두 개의 Hidden Layer(은닉층)로 이루어진 모델
- Weight 초기는 random_normal 이용해서 랜덤으로 설정으로 정규 분포에 의해서 0~1 사이의 값으로 만들어 낸다.
- 첫 번째, 두 번째 Hidden Layer는 각각 256개의 노드를 갖는다
- 활성화 함수는 Relu 함수 사용
- Optimizer 는 Adam 사용

3) 학습

- 학습 횟수 만큼 배치 사이즈 만큼 최적화 수행
- Optimizer를 실행해서 파라미터들을 업데이트 하면서 평균 손실 값을 구한다

<출력>

```
반복(Epoch): 1, 손실 함수(Loss): 324.921943
반복(Epoch): 2, 손실 함수(Loss): 73.615544
반복(Epoch): 3, 손실 함수(Loss): 48.066992
반복(Epoch): 4, 손실 함수(Loss): 36.026038
반복(Epoch): 5, 손실 함수(Loss): 27.964136
반복(Epoch): 6, 손실 함수(Loss): 22.508947
반복(Epoch): 7, 손실 함수(Loss): 18.460410
반복(Epoch): 8, 손실 함수(Loss): 15.387058
반복(Epoch): 9, 손실 함수(Loss): 12.646134
반복(Epoch): 10, 손실 함수(Loss): 10.399010
반복(Epoch): 11, 손실 함수(Loss): 8.880302
반복(Epoch): 12, 손실 함수(Loss): 7.538867
반복(Epoch): 13, 손실 함수(Loss): 6.405457
반복(Epoch): 14, 손실 함수(Loss): 5.228646
반복(Epoch): 15, 손실 함수(Loss): 4.443401
반복(Epoch): 16, 손실 함수(Loss): 3.598247
반복(Epoch): 17, 손실 함수(Loss): 2.978423
반복(Epoch): 18, 손실 함수(Loss): 2.553977
반복(Epoch): 19, 손실 함수(Loss): 1.949236
반복(Epoch): 20, 손실 함수(Loss): 1.697232
반복(Epoch): 21, 손실 함수(Loss): 1.341644
반복(Epoch): 22, 손실 함수(Loss): 1.044634
반복(Epoch): 23, 손실 함수(Loss): 0.873269
반복(Epoch): 24, 손실 함수(Loss): 0.734212
반복(Epoch): 25, 손실 함수(Loss): 0.615406
반복(Epoch): 26, 손실 함수(Loss): 0.509251
반복(Epoch): 27, 손실 함수(Loss): 0.417816
반복(Epoch): 28, 손실 함수(Loss): 0.359362
반복(Epoch): 29, 손실 함수(Loss): 0.313582
반복(Epoch): 30, 손실 함수(Loss): 0.286042
정확도(Accuracy): 0.938700
```

2.3 CNN(Convolutional Neural Network)

- 두 개의 합성곱 층(Convolution Layer)을 가진 CNN 모델

<코드>

```
import tensorflow as tf

# MNIST 데이터를 다운로드 합니다.
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# 인풋, 아웃풋 데이터를 받기위한 플레이스홀더를 정의합니다.
x = tf.placeholder(tf.float32, shape=[None, 784])
y = tf.placeholder(tf.float32, shape=[None, 10])

# 학습을 위한 설정값들을 정의합니다.
learning_rate = 0.001 # 학습률
num_epochs = 100      # 학습횟수
batch_size = 10       # 배치개수


# CNN 모델을 정의합니다.
def build_CNN_classifier(x):
    # MNIST 데이터를 3 차원 형태로 reshape 합니다. MNIST 데이터는 grayscale 이미지기 때문에 3 번째차원(컬러채널)의 값은 1 입니다.
    x_image = tf.reshape(x, [-1, 28, 28, 1])

    # 첫번째 Convolution Layer
    # 5x5 Kernel Size 를 가진 32 개의 Filter 를 적용합니다.
    # 28x28x1 -> 28x28x32
    W_conv1 = tf.Variable(tf.truncated_normal(shape=[5, 5, 1, 32], stddev=5e-2))
    b_conv1 = tf.Variable(tf.constant(0.1, shape=[32]))
    h_conv1 = tf.nn.relu(tf.nn.conv2d(x_image, W_conv1, strides=[1, 1, 1, 1], padding='SAME') + b_conv1)
```

```
    # 첫번째 Pooling Layer
    # Max Pooling 을 이용해서 이미지의 크기를 1/2 로 downsample 합니다.
    # 28x28x32 -> 14x14x32
    h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME')

    # 두번째 Convolutional Layer
    # 5x5 Kernel Size 를 가진 64 개의 Filter 를 적용합니다.
    # 14x14x32 -> 14x14x64
    W_conv2 = tf.Variable(tf.truncated_normal(shape=[5, 5, 32, 64], stddev=5e-2))
    b_conv2 = tf.Variable(tf.constant(0.1, shape=[64]))
    h_conv2 = tf.nn.relu(tf.nn.conv2d(h_pool1, W_conv2, strides=[1, 1, 1, 1],
padding='SAME') + b_conv2)

    # 두번째 Pooling Layer
    # Max Pooling 을 이용해서 이미지의 크기를 1/2 로 downsample 합니다.
    # 14x14x64 -> 7x7x64
    h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME')

    # Fully Connected Layer
    # 7x7 크기를 가진 64 개의 activation map 을 1024 개의 특징들로 변환합니다.
    # 7x7x64(3136) -> 1024
    W_fc1 = tf.Variable(tf.truncated_normal(shape=[7 * 7 * 64, 1024], stddev=5e-2))
    b_fc1 = tf.Variable(tf.constant(0.1, shape=[1024]))
    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

    # Output Layer
    # 1024 개의 특징들(feature)을 10 개의 클래스-one-hot encoding 으로 표현된 숫자
0~9-로 변환합니다.
    # 1024 -> 10
    W_output = tf.Variable(tf.truncated_normal(shape=[1024, 10], stddev=5e-2))
    b_output = tf.Variable(tf.constant(0.1, shape=[10]))
```

```
    logits = tf.matmul(h_fc1, W_output) + b_output
    y_pred = tf.nn.softmax(logits)

    return y_pred, logits


# Convolutional Neural Networks(CNN)을 선언합니다.
y_pred, logits = build_CNN_classifier(x)

# Cross Entropy 를 손실 함수(loss function)으로 정의하고 옵티마이저를
정의합니다.
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=logits))
train_step = tf.train.AdamOptimizer(learning_rate).minimize(loss)

# 정확도를 계산하는 연산을 추가합니다.
correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# 세션을 열어 실제 학습을 진행합니다.
with tf.Session() as sess:
    # 모든 변수들을 초기화합니다.
    sess.run(tf.global_variables_initializer())

    # 10000 Step 만큼 최적화를 수행합니다.
    for i in range(num_epochs):
        # 50 개씩 MNIST 데이터를 불러옵니다.
        batch = mnist.train.next_batch(batch_size)
        # 100 Step 마다 training 데이터셋에 대한 정확도를 출력합니다.
        if i % batch_size == 0:
            train_accuracy = accuracy.eval(feed_dict={x: batch[0], y: batch[1]})
            print("반복(Epoch): %d, 트레이닝 데이터 정확도: %f" % (i, train_accuracy))
        # 옵티마이저를 실행해 파라미터를 한스텝 업데이트합니다.
        sess.run([train_step], feed_dict={x: batch[0], y: batch[1]})

    # 학습이 끝나면 테스트 데이터에 대한 정확도를 출력합니다.
```

```
print("테스트 데이터 정확도: %f" % accuracy.eval(feed_dict={x: mnist.test.images, y: mnist.test.labels}))
```

<설명>

1) 학습을 위한 설정 값들 정의

```
learning_rate = 0.001 # 학습률
num_epochs = 100      # 학습횟수
batch_size = 10       # 배치개수
```

2) CNN 모델 정의

- CNN 모델(순서)
  첫 번째 합성곱 -> 활성화 함수 -> 맥스 풀링 -> 두 번째 합성곱 -> 활성화 함수 -> 맥스 풀링 -> 완전 연결 -> 활성화 함수 -> 출력 -> 소프트 맥스 -> 라벨
- 두 개의 Convolution Layer
- 활성화 함수는 relu 사용
- Max pooling 사용
- Optimizer는 Adam 사용

<출력>

```
반복(Epoch): 0, 트레이닝 데이터 정확도: 0.000000
반복(Epoch): 10, 트레이닝 데이터 정확도: 0.000000
반복(Epoch): 20, 트레이닝 데이터 정확도: 0.300000
반복(Epoch): 30, 트레이닝 데이터 정확도: 0.100000
반복(Epoch): 40, 트레이닝 데이터 정확도: 0.900000
반복(Epoch): 50, 트레이닝 데이터 정확도: 0.800000
반복(Epoch): 60, 트레이닝 데이터 정확도: 0.800000
반복(Epoch): 70, 트레이닝 데이터 정확도: 0.900000
반복(Epoch): 80, 트레이닝 데이터 정확도: 0.900000
반복(Epoch): 90, 트레이닝 데이터 정확도: 0.800000
테스트 데이터 정확도: 0.869200
```