8. 머신러닝과 딥러닝 이해하기 - III. Neural Nets(NN) for XOR

퍼셉트론

개요

1957년 프랑크 로젠블라트(Frank Rosenblat)가 고안한 알고리즘으로 복잡한 함수도 표현 가능

신경망(Neural Nets)의 기원이 되는 알고리즘

다수의 신호를 입력으로 받아 하나의 신호를 출력 (AND, OR, NAND)

세 가지 게이트에서 매개변수(가중치(weight)와 임계값(threshold))를 조정하여 계산 수행

- AND 게이트 : x1과 x2가 모두 1일때만 1
- OR 게이트 : x1과 x2 중 하나만 1이어도 1
- NAND 게이트 : AND게이트의 반대 (x1과 x2가 모두 1일때만 0)

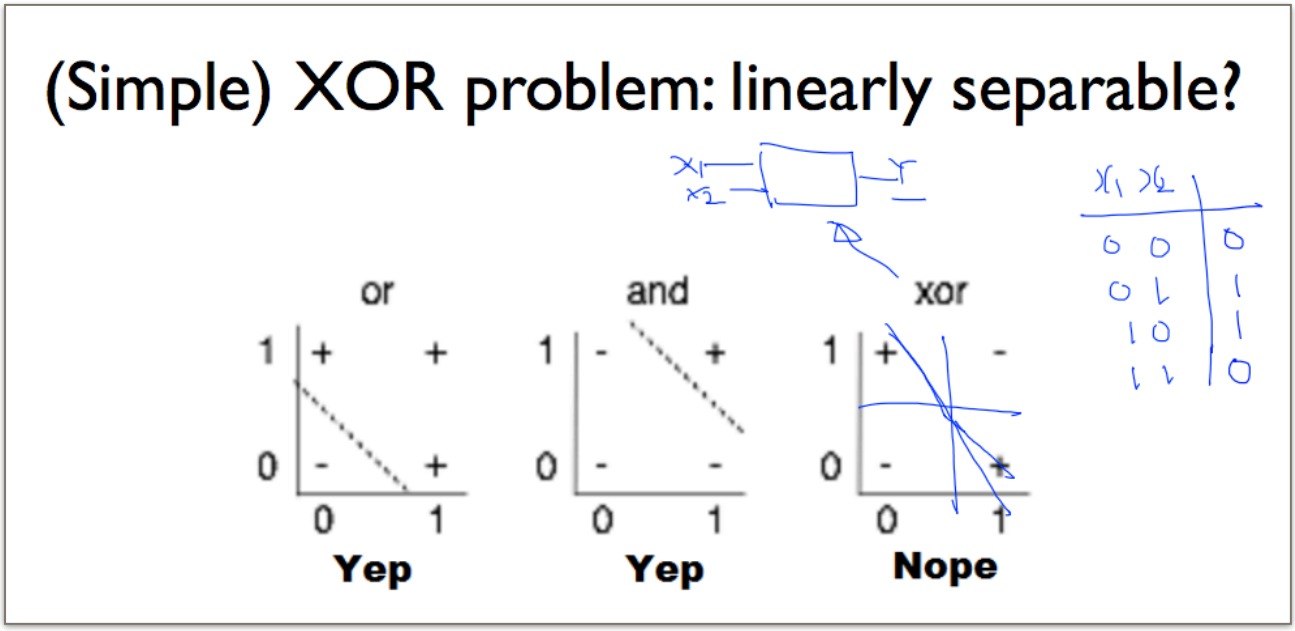
AND 연산 결과	
입력값	결과값
0 , 0	0
0 , 1	0
1 , 0	0
1 , 1	1

OR 연산 결과	
입력값	결과값
0 , 0	0
0 , 1	1
1 , 0	1
1 , 1	1

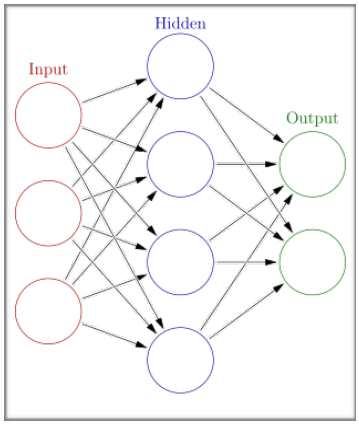
NAND 연산 결과	
입력값	결과값
0 , 0	1
0 , 1	1
1 , 0	1
1 , 1	0

한계

- XOR(배타적 논리합) 계산 불가능



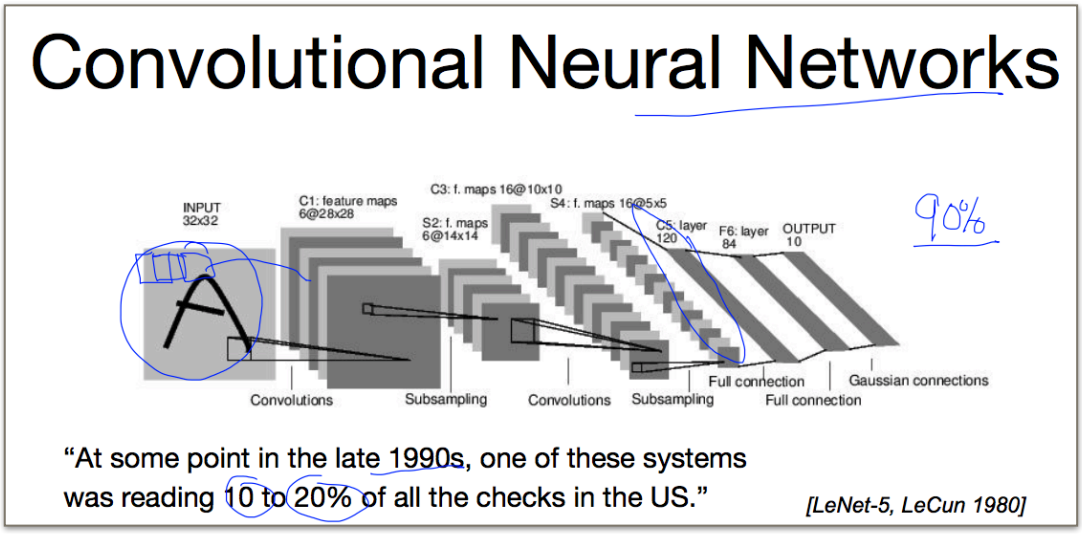
신경망(Neural Networks, NN)

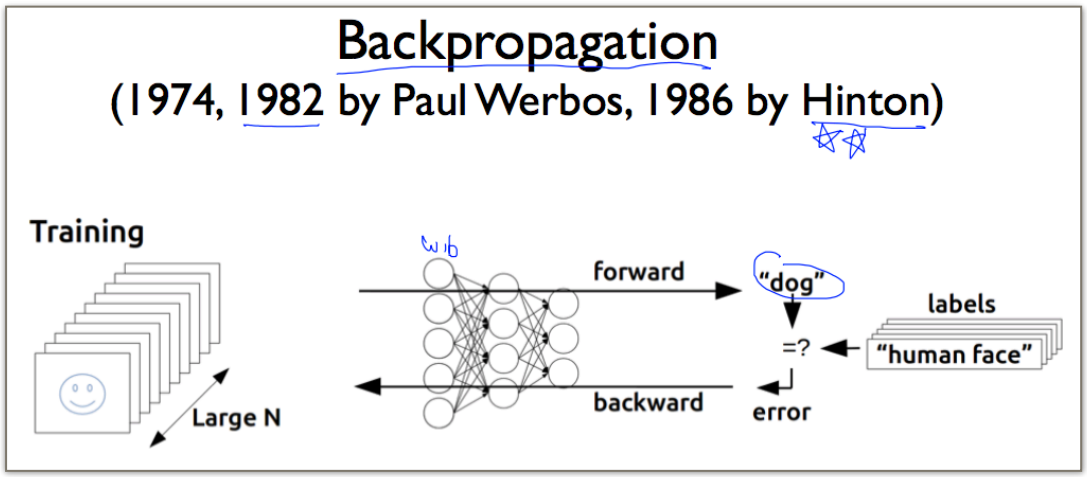


- 기계학습과 인지과학에서 생물학의 신경망에서 영감을 얻은 통계학적 학습 알고리즘 (인공신경망이라고도 함)
- 가중치 매개변수의 적절한 값을 데이터로부터 자동으로 학습하여 처리하는 알고리즘
- 각 층의 뉴런들이 다음 층의 뉴런으로 신호를 전달한다는 점에서 퍼셉트론과 비슷하지만 다음 뉴런으로 갈 때 신호를 변화시키는 활성화함수(Sigmoid function)에 큰 차이가 있음
- 딥러닝은 인공신경망으로부터 본격적으로 시작됨

합성곱 신경망(Convolutional Neural Networks, CNN)

- 최소한의 전처리(preprocess)를 사용하도록 설계된 다계층 퍼셉트론(multilayer perceptrons)의 한 종류
- CNN은 하나 또는 여러개의 합성곱 계층과 그 위에 올려진 일반적인 인공 신경망 계층들로 이루어져 있으며, 가중치와 통합 계층(pooling layer)들을 추가로 활용
- 이러한 구조 덕분에 CNN은 2차원 구조의 입력 데이터를 충분히 활용할 수 있음
- 다른 딥러닝 구조들과 비교해서, CNN은 영상, 음성 분야 모두에서 좋은 성능을 보임
- CNN은 또한 역전달(Backpropagation)을 통해 훈련되며 쉽게 훈련되고 적은 수의 매개변수를 사용함





딥러닝 실습1 : Linear Regression 알고리즘 구현하기

코드 : ch16_linear_regession.py

```
# 라이브러리 불러오기
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

# 데이터 포인트를 생성하고 모델 학습시킴. 먼저 생성할 데이터 포인트 수를 정의
num_points = 1200

# 데이터 생성에 사용할 매개변수 정의. 예제에서는 y = mx + c 라는 선형 모델을 사용
data = []
m = 0.2
c = 0.5
for i in range(num_points):
    # x 생성하기
    x = np.random.normal(0.0, 0.8)

    # 노이즈 생성(데이터에 약간의 변화를 주기 위함)
    noise = np.random.normal(0.0, 0.04)

    # 등식의 y 값 계산
    y = m*x + c + noise

    data.append([x, y])

# 반복문이 끝나면, 데이터를 입력변수(x)와 출력변수(y)로 나누기
x_data = [d[0] for d in data]
y_data = [d[1] for d in data]

# 생성된 데이터를 그래프로 그리기
plt.plot(x_data, y_data, 'ro')
plt.title('Input data')
plt.show()

# 퍼셉트론에 대한 가중치(W)와 바이어스(b) 생성하기
# 가중치는 균등난수발생기로 생성하고, 바이어스는 '0'으로 지정
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))

# 'y'에 대한 등식 정의하기
y = W * x_data + b
```

```
# 손실(loss) 계산방법 정의
loss = tf.reduce_mean(tf.square(y - y_data))

# Gradient descent 옵티마이저 정의하고, 손실함수 지정
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# 모든 변수 초기화하기
init = tf.initialize_all_variables()

# 텐서플로우 세션 생성 및 실행
sess = tf.Session()
sess.run(init)

# 학습과정 시작
# 반복문 시작
num_iterations = 10
for step in range(num_iterations):
    # 세션 실행하기
    sess.run(train)

    # 진행상태를 화면에 출력하기
    print('\nITERATION', step+1)
    print('W =', sess.run(W)[0])
    print('b =', sess.run(b)[0])
    print('loss =', sess.run(loss))

    # 입렉 데이터를 그래프로 그리기
    plt.plot(x_data, y_data, 'ro')

    # 예측 결과에 대한 직선 그래프 그리기
    plt.plot(x_data, sess.run(W) * x_data + sess.run(b))

    # 그래프에 대한 매개변수 설정하기
    plt.xlabel('Dimension 0')
    plt.ylabel('Dimension 1')
    plt.title('Iteration ' + str(step+1) + ' of ' + str(num_iterations))
    plt.show()
```

딥러닝 실습2 : XOR문제 딥러닝으로 풀기

코드 :

lab-09-1-xor.py

lab-09-2-xor-nn.py

 # lab-09-1-xor.py

```
import tensorflow as tf
import numpy as np

tf.set_random_seed(777)  # for reproducibility
learning_rate = 0.1

x_data = [[0, 0],
          [0, 1],
          [1, 0],
          [1, 1]]
y_data = [[0],
          [1],
          [1],
          [0]]
x_data = np.array(x_data, dtype=np.float32)
y_data = np.array(y_data, dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid: tf.div(1., 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                       tf.log(1 - hypothesis))

train = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
```

```
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, sess.run(cost, feed_dict={
                  X: x_data, Y: y_data}), sess.run(W))

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy],
                       feed_dict={X: x_data, Y: y_data})
    print("\nHypothesis: ", h, "\nCorrect: ", c, "\nAccuracy: ", a)
```

실행 결과

```
'''
Hypothesis:  [[ 0.5]
 [ 0.5]
 [ 0.5]
 [ 0.5]]
Correct:  [[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]
Accuracy:  0.5
'''
```

```
# lab-09-2-xor-nn.py

import tensorflow as tf
import numpy as np

tf.set_random_seed(777)  # for reproducibility
learning_rate = 0.1

x_data = [[0, 0],
          [0, 1],
          [1, 0],
          [1, 1]]
y_data = [[0],
          [1],
          [1],
          [0]]
x_data = np.array(x_data, dtype=np.float32)
y_data = np.array(y_data, dtype=np.float32)

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])

W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) *
                       tf.log(1 - hypothesis))

train = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
```

```
# Initialize TensorFlow variables
sess.run(tf.global_variables_initializer())

for step in range(10001):
    sess.run(train, feed_dict={X: x_data, Y: y_data})
    if step % 100 == 0:
        print(step, sess.run(cost, feed_dict={
              X: x_data, Y: y_data}), sess.run([W1, W2]))

# Accuracy report
h, c, a = sess.run([hypothesis, predicted, accuracy],
                   feed_dict={X: x_data, Y: y_data})
print("\nHypothesis: ", h, "\nCorrect: ", c, "\nAccuracy: ", a)
```

실행 결과

```
'''
Hypothesis:  [[ 0.01338218]
 [ 0.98166394]
 [ 0.98809403]
 [ 0.01135799]]
Correct:  [[ 0.]
 [ 1.]
 [ 1.]
 [ 0.]]
Accuracy:  1.0
'''
```

딥러닝 실습3 : MNIST 손글씨 인식하기

코드 : lab-07-4-mnist_introduction.py

```
# Lab 7 Learning rate and Evaluation
import tensorflow as tf
import random
import matplotlib.pyplot as plt
tf.set_random_seed(777)  # for reproducibility

from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

nb_classes = 10

# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])

W = tf.Variable(tf.random_normal([784, nb_classes]))
b = tf.Variable(tf.random_normal([nb_classes]))

# Hypothesis (using softmax)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Test model
is_correct = tf.equal(tf.arg_max(hypothesis, 1), tf.arg_max(Y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
training_epochs = 15
batch_size = 100

with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
```

```
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        c, _ = sess.run([cost, optimizer], feed_dict={
                        X: batch_xs, Y: batch_ys})
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1),
          'cost =', '{:.9f}'.format(avg_cost))

print("Learning finished")
```

```
# Test the model using test sets
print("Accuracy: ", accuracy.eval(session=sess, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

plt.imshow(
    mnist.test.images[r:r + 1].reshape(28, 28),
    cmap='Greys',
    interpolation='nearest')
plt.show()
```

실행결과

```
'''
Epoch: 0001 cost = 2.868104637
Epoch: 0002 cost = 1.134684615
Epoch: 0003 cost = 0.908220728
Epoch: 0004 cost = 0.794199896
Epoch: 0005 cost = 0.721815854
Epoch: 0006 cost = 0.670184430
Epoch: 0007 cost = 0.630576546
Epoch: 0008 cost = 0.598888191
Epoch: 0009 cost = 0.573027079
Epoch: 0010 cost = 0.550497213
Epoch: 0011 cost = 0.532001859
Epoch: 0012 cost = 0.515517795
Epoch: 0013 cost = 0.501175288
Epoch: 0014 cost = 0.488425370
Epoch: 0015 cost = 0.476968593
Learning finished
Accuracy:  0.888
'''
```