인공지능개론 정리노트#1

201904022 김상옥

202284026 안정빈

(2장) **질문1. 다층 퍼셉트론의 한계는?**

안정빈 : 다층 퍼셉트론은 입력 데이터가 매우 크거나 고차원일 경우 계산량이 급격하게 증가하여 효율적으로 작동 하지 않을 수있습니다. 고차원의 데이터를 다루기에는 한계가 있다고 생각합니다.

김상옥 : 다층 퍼셉트론으로 컴퓨터를 "이론상" 만들 수 있다는건 분명 한계가 있다고 생각합니다.

컴퓨터는 수많은 양의 연산을 빠르고 정확하게 처리할 수 있는데 이와 관련된 문제를 처리하기 위해 퍼셉트론의 구조를 사용하지 않는다고 합니다.

데이터 처리에 있어서 속도나 복잡한 문제를 풀기 어렵기 때문이지 않을까?

또 퍼셉트론의 결과는 0 또는 1이 나타나는데 이 부분도 다양한 문제를 처리하고 학습하는데 있어서 부족한 성능을 보일 것 같습니다.

-> 질문 1-1. 극복하는 방법은?

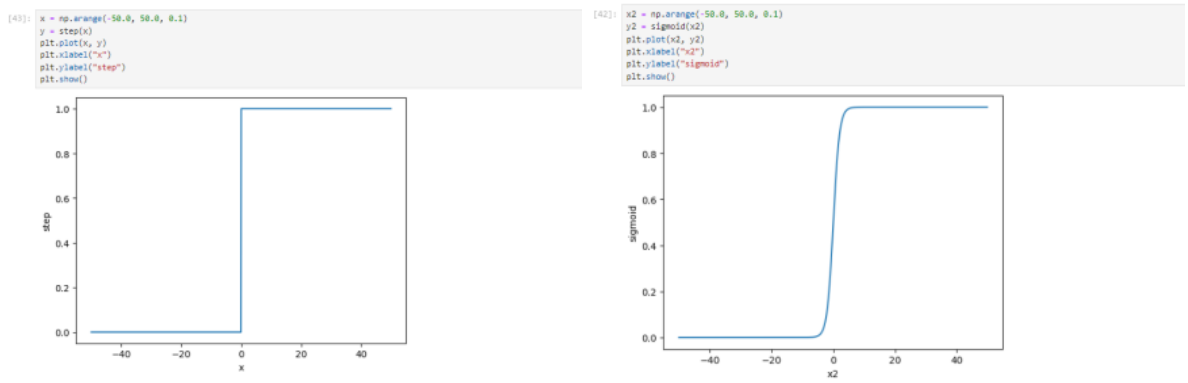
안정빈 : 은닉층을 더 많은 단계를 거쳐 학습하여 한계를 극복 하려고 노력하고, 모델의 단순화를 통해 모델의 구조를 간소화 하여 학습과 추론 속도를 향상시킬 것 입니다.

김상옥 : 더 좋은 하드웨어를 사용하거나 효율적인 구조를 설계한다면 성능이 좋아지겠지만

결국에는 한계가 있을 것이다. 그렇기에 인공 신경망을 주로 사용하는 것 같다.

결론 : 다층 퍼셉트론은 우리가 인공지능을 통해서 해결하고자 하는 문제들을 해결하기에는 성능이 부족하다고 생각합니다. 예를들어 복잡한 이미지나 숫자등을 제대로 구분하지 못하는 문제가 발생할 것 같습니다.

(3장)**질문2. 시그모이드 함수의 값의 편차를 크게 두면 계단 함수와 똑같은 그래프가 나타난다. 시그모이드 함수를 사용하는 이유는?**



안정빈 : 시그모이드 함수는 연속적인 함수이고, 학습과 최적화 과정을 부드럽게 만들어 줍니다. 반면에 계단 함수는 불연속적이며, 미분이 불가능하여 이는 역전파와 같은 알고리즘에서 문제를 야기할수 있을 것 입니다.
시그모이드는 계단함수에서는 표현할 수 없었던 내부값의 차이를 표현 할 것입니다. 계단 함수는 출력값이 0 과 1 로 구분되지만, 시그모이드는 더 상세하게 값을 출력해 줌으로써 어떠한 학습을 하여도 더 최적화 된 값을 표출해 줄것입니다. 그래서 같은 그래프의 모양이 나타나더라도 시그모이드 함수를 사용한다고 생각합니다.

김상옥 : 시그모이드 함수를 사용하는 이유를 생각해보자면 답은 수업시간에 교수님이 말씀하신 미분에 있을 것이다. 어떤 것을 학습한다는 것은 정답이 존재한다. 강아지와 고양이 이미지를 구분한다면 정답은 강아지 or 고양이로 나타난다.
이 신경망에 이미지를 넣었을 때 정답을 도출하기 위해서는 어떠한 기준으로 학습을 하지 않을까? 랜덤한 피라미터값을 설정한 후 강아지 이미지를 넣었을 때 나온 결과가 고양이라면 강아지에 적합한 피라미터값을 찾기 위해 미분을 사용하는 것이 올바른 학습을 위함일 것이다.
또 시그모이드의 그래프가 계단 함수와 비슷하지만 그렇게 보일 뿐 값들은 0.01 - 0.02 이런 식으로 분명한 차이가 있을 것이다. 완전히 0 과 1 로만 구성된 계단 함수와는 다르게 미분이 가능하기에 시그모이드 함수를 사용하는 것 같다.

결론 : 시그모이드 함수를 사용했을경우 더 최적화된 값을 출력함으로써 학습 결과 가 더 좋다고 생각했습니다.

질문 3 .사람의 신경망(구조)을/를 사용하는 이유는 그게 효율적이기 때문인가 아니면 말 그대로 기계를 사람처럼 인공 지능을 만들기 위함인가?

안정빈 : 사람과 비슷한 수준의 지능으로 학습하기 위해 사람의 신경망 구조를 사용 하는 것 같습니다. 사람의 신경망을 모방하여 학습하면 복잡한 정보처리와 학습을 수행하는데 효율적이게 됩니다. 학습을 점차 시키다 보면 사람이 개입하지 않아도 컴퓨터가 지능적인 결정을 내릴수 있게 되고, 데이터를 직접 이해하고 명시적인 훈련없이도 일반화, 추론 등이 가능해 지게 됩니다. 즉, 기계를 사람과 유사한 지능을 만들기 위해 사람의 신경망 구조를 사용한다고 생각합니다.

김상옥 : 동물들의 뇌가 사람의 뇌와 정확히 어떤 차이가 있는 지 잘 모르지만, 더 효율적이고 창의적인 뇌를 가졌기 때문에 지금의 인간이 있다고 생각한다. 이러한 이유로 인간의 뇌를 모방했을 수도 있을 것 같다. 다른 생각으로는 결국 인공지능을 사용하는 이유는 사람이 하는 일을 대신 처리할 기계를 만드는 것이기 때문이다.그렇기에 사람의 신경망 구조를 모방하여 구조를 만들지 않았을까 싶다. 이 아이디어가 나온 시대를 생각해보면 정확히 사람의 뇌가 효율적이라서 신경망을 만들었다기 보다 사람의 일을 처리하기 위해 사람의 뇌를 모방하여 만들었다는 점이 더 설득력있게 들린다.

결론 : 사람의 일을 처리하고, 유사한 지능을 만들기 위해 신경망을 사용한다고 생각합니다.

질문 4. 신경망과 퍼셉트론의 구체적인 차이점은?

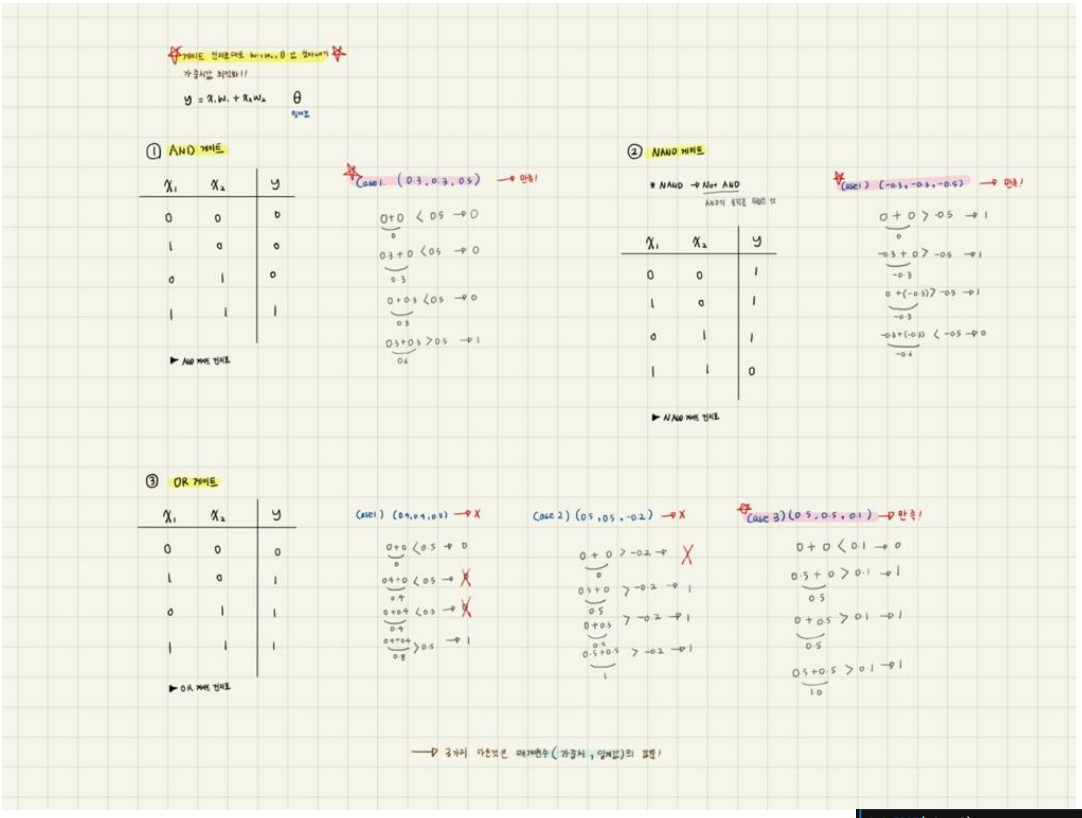
안정빈 : 퍼셉트론은 단순한 학습 알고리즘을 사용하여 가중치를 조정하고 입력 패턴에 대한 출력을 학습하며 계단함수와 같은 간단한 선형 형태의 활성화 함수를 사용하고, 신경망은 역전파 알고리즘 같은 더 복잡한 알고리즘을 사용하여 가중치와 편향을 조정하여 오차를 더 최소화 하여 학습하며 시그모이드, 렐루와 등 과 같은 비선형 활성 함수를 사용합니다. 이러한 차이점들은 퍼셉트론과 신경망이 서로 다른 수준의 복잡성과 기능을 가지고 있다는 것을 알 수 있으며 신경망은 더욱 복잡하고 강력한 모델을 구성 및 학습하는데 사용됩니다.

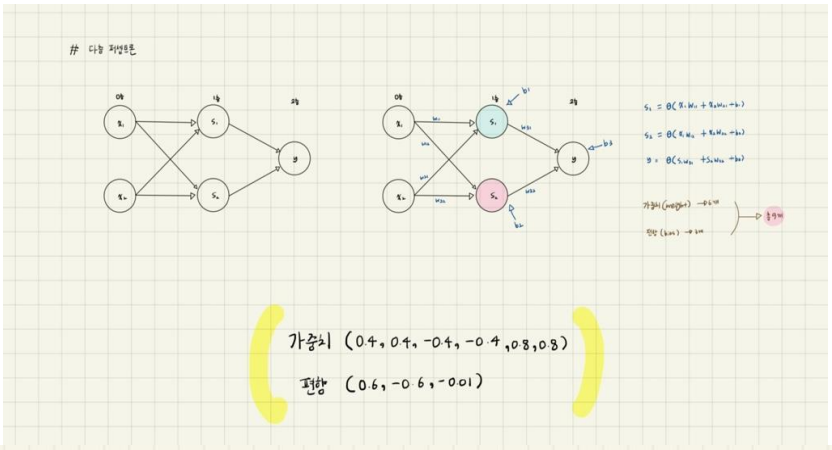
김상옥 : 퍼셉트론도 다층 퍼셉트론을 통해서 많은 은닉층을 구성할 수 있다.
이는 신경망과 비슷한 구조로 보이지만 퍼셉트론은 결과값이 0 과 1 로만 나타나며
회로를 구성하는 것 처럼 유연하지 않은 결과를 보여줄 것 같다.
신경망은 다양한 활성화 함수를 사용하고, 이게 맞게 다양한 출력값을 보여준다.
이러한 점이 더 창의적이고 복잡한 문제를 해결할 수 있다고 생각한다.

결론 : 활성화 함수의 차이 와 서로 다른 수준의 복잡한 문제를 해결 할 수 있다고 생각합니다.

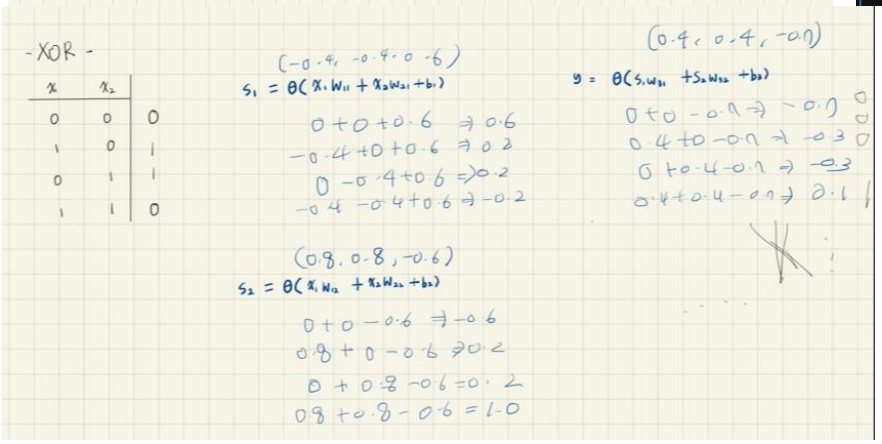
2 장

(안정빈)






```python
def TEST(x1, x2):
    w11 = 0.4
    w12 = 0.4

    w21 = -0.4
    w22 = -0.4

    w31 = 0.8
    w32 = 0.8

    b1 = 0.6
    b2 = -0.6
    b3 = -0.01

    s1 = x1 * w11 + x2 * w21 + b1
    s2 = x1 * w12 + x2 * w22 + b2

    y =  s1*w31 + s2* w32 + b3

    return 0 if y <= 0 else 1  # 반환값 조정


print(TEST(0, 0))  # 출력: 0
print(TEST(1, 0))  # 출력: 1
print(TEST(0, 1))  # 출력: 1
print(TEST(1, 1))  # 출력: 0
```



```python
def AND(x1,x2):
    x= np.array([x1,x2])
    w= np.array([0.4,0.4])
    b=-0.7
    tmp= np.sum(w*x)+b
    if tmp <=0:
        return 0
    else:
        return 1
```

```python
def NAND(x1,x2):
    x= np.array([x1,x2])
    w= np.array([-0.4,-0.4])
    b=0.6
    tmp= np.sum(w*x)+b
    if tmp <=0:
        return 0
    else:
        return 1
```

```python
def OR(x1,x2):
    x= np.array([x1,x2])
    w= np.array([0.8,0.8])
    b=-0.6
    tmp= np.sum(w*x)+b
    if tmp <=0:
        return 0
    else:
        return 1
```

```python
def XOR(x1,x2):
    s1=NAND(x1,x2)
    s2=OR(x1,x2)
    y= AND(s1,s2)
    return y
```

```python
print('(0,0)=',AND(0,0))
print('(1,0)=',AND(1,0))
print('(0,1)=',AND(0,1))
print('(1,1)=',AND(1,1))

print('--------------------')


print('(0,0)=',NAND(0,0))
print('(1,0)=',NAND(1,0))
print('(0,1)=',NAND(0,1))
print('(1,1)=',NAND(1,1))

print('--------------------')


print('(0,0)=',OR(0,0))
print('(1,0)=',OR(1,0))
print('(0,1)=',OR(0,1))
print('(1,1)=',OR(1,1))

print('--------------------')

print('(0,0)=',XOR(0,0))
print('(1,0)=',XOR(1,0))
print('(0,1)=',XOR(0,1))
print('(1,1)=',XOR(1,1))
```

```
(0,0)= 0
(1,0)= 0
(0,1)= 0
(1,1)= 1
--------------------
(0,0)= 1
(1,0)= 1
(0,1)= 1
(1,1)= 0
--------------------
(0,0)= 0
(1,0)= 1
(0,1)= 1
(1,1)= 1
--------------------
(0,0)= 0
(1,0)= 1
(0,1)= 1
(1,1)= 0
```

(김상욱)

```
[17]: # y = x1w1 + x2w2

[26]: x1 = float(input()) # 0 or 1
      x2 = float(input())

      1
      1

[27]: w1 = float(input())
      w2 = float(input())

      0.1
      0.2

[28]: y = x1*w1 + x2*w2
      print("y = ", y)
      print("결과 : ", 1 if y > 0 else 0)

      y =  0.30000000000000004
      결과 :  1

[ ]:
```

```
[17]: # y = x1w1 + x2w2

[26]: x1 = float(input()) # 0 or 1
      x2 = float(input())

      1
      1

[30]: w1 = float(input())
      w2 = float(input())

      0.3
      0.2

[31]: y1 = 0*w1 + 0*w2
      y2 = 0*w1 + 1*w2
      y3 = 1*w1 + 0*w2
      y4 = 1*w1 + 1*w2

[32]: print("결과1 : ", 1 if y1 > 0 else 0)
      print("결과2 : ", 1 if y2 > 0 else 0)
      print("결과3 : ", 1 if y3 > 0 else 0)
      print("결과4 : ", 1 if y4 > 0 else 0)

      결과1 :  0
      결과2 :  1
      결과3 :  1
      결과4 :  1

[ ]:
```

```
[17]: # y = x1w1 + x2w2

[26]: x1 = float(input()) # 0 or 1
      x2 = float(input())

      1
      1

[61]: w1 = float(input())
      w2 = float(input())
      theta = float(input())

      0.6
      0.6
      0.2

[62]: y1 = 0*w1 + 0*w2
      y2 = 0*w1 + 1*w2
      y3 = 1*w1 + 0*w2
      y4 = 1*w1 + 1*w2

[63]: print("결과1 : ", 1 if y1 > theta else 0)
      print("결과2 : ", 1 if y2 > theta else 0)
      print("결과3 : ", 1 if y3 > theta else 0)
      print("결과4 : ", 1 if y4 > theta else 0)

      결과1 :  0
      결과2 :  1
      결과3 :  1
      결과4 :  1

[ ]: # 0.5, 0.5, 0.7 -> 0 0 0 1
     #
```

```
[17]: # y = x1w1 + x2w2

[26]: x1 = float(input()) # 0 or 1
      x2 = float(input())

      1
      1

[76]: w1 = float(input())
      w2 = float(input())
      theta = float(input())

      0.3
      0.8
      0.5

[77]: y1 = 0*w1 + 0*w2
      y2 = 0*w1 + 1*w2
      y3 = 1*w1 + 0*w2
      y4 = 1*w1 + 1*w2

[78]: print("결과1 : ", 1 if y1 > theta else 0)
      print("결과2 : ", 1 if y2 > theta else 0)
      print("결과3 : ", 1 if y3 > theta else 0)
      print("결과4 : ", 1 if y4 > theta else 0)

      결과1 :  0
      결과2 :  1
      결과3 :  0
      결과4 :  1

[ ]: # 0.5, 0.5, 0.7 -> 0 0 0 1
     #
```

가중치와 편향 값에 따라 AND, OR, NAND -> 계산은 똑같음

[253]:

```
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.7
    tmp = np.sum(w*x) + b
    return 0 if tmp <= 0 else 1
```

[254]:

```
print("AND 0 0 | ", AND(0, 0))
print("AND 0 1 | ", AND(0, 1))
print("AND 1 0 | ", AND(1, 0))
print("AND 1 1 | ", AND(1, 1))
```

```
AND 0 0 |  0
AND 0 1 |  0
AND 1 0 |  0
AND 1 1 |  1
```

[ ]:

[255]:

```
def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5])
    b = 0.7
    tmp = np.sum(w*x) + b
    return 0 if tmp <= 0 else 1
```

[256]:

```
print("NAND 0 0 | ", NAND(0, 0))
print("NAND 0 1 | ", NAND(0, 1))
print("NAND 1 0 | ", NAND(1, 0))
print("NAND 1 1 | ", NAND(1, 1))
```

```
NAND 0 0 |  1
```

```
NOR 0 0 |  1
NOR 0 1 |  0
NOR 1 0 |  0
NOR 1 1 |  0
```

[ ]:

[307]:

```
def XOR(x1, x2):
    return AND(NAND(x1, x2), OR(x1, x2))
```

[308]:

```
print("XOR 0 0 | ", XOR(0, 0))
print("XOR 0 1 | ", XOR(0, 1))
print("XOR 1 0 | ", XOR(1, 0))
print("XOR 1 1 | ", XOR(1, 1))
```

```
XOR 0 0 |  0
XOR 0 1 |  1
XOR 1 0 |  1
XOR 1 1 |  0
```

[ ]:

[478]:

```
def XOR_test(x1, x2):
    w11 = 0.4
    w12 = 0.4
    w21 = 0.4
    w22 = 0.8
    w31 = 0.8
    w32 = 0.8

    b1 = 0.6
    b2 = -0.6
    b3 = -0.01

    s1 = x1*w11 + x2*w21 + b1
    s2 = x1*w12 + x2*w22 + b2

    y = x1 * s1 + x2 * s2 + b3
```

```
def XOR_test(x1, x2):
    w11 = 0.4
    w12 = 0.4
    w21 = 0.4
    w22 = 0.8
    w31 = 0.8
    w32 = 0.8

    b1 = 0.6
    b2 = -0.6
    b3 = -0.01

    s1 = x1*w11 + x2*w21 + b1
    s2 = x1*w12 + x2*w22 + b2

    y = x1 * s1 + x2 * s2 + b3

    return 0 if y <= 0 else 1
```

```
print("XOR_test 0 0 | ", XOR_test(0, 0))
print("XOR_test 0 1 | ", XOR_test(0, 1))
print("XOR_test 1 0 | ", XOR_test(1, 0))
print("XOR_test 1 1 | ", XOR_test(1, 1))
```

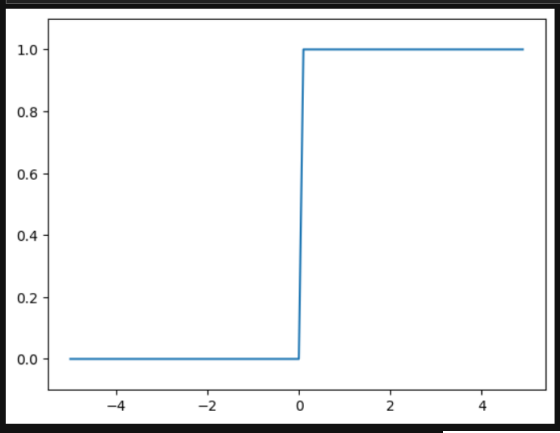
```
XOR_test 0 0 |  0
XOR_test 0 1 |  1
XOR_test 1 0 |  1
XOR_test 1 1 |  1
```

3 장

(안정빈)

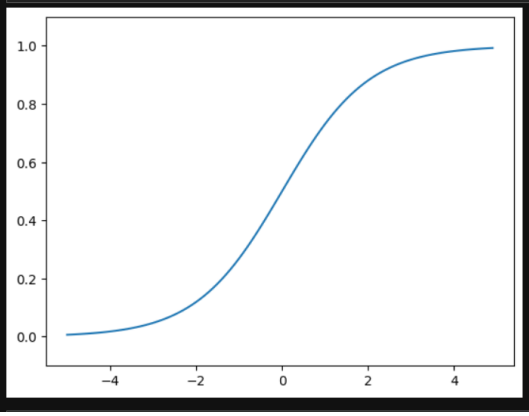
```
    return np.array(x>0, dtype=np.int32)
```

[6]:
```
X=np.arange(-5.0,5.0,0.1)
Y=step_function(X)
plt.plot(X,Y)
plt.ylim(-0.1,1.1) #y축의 범위 지정
plt.show()
```



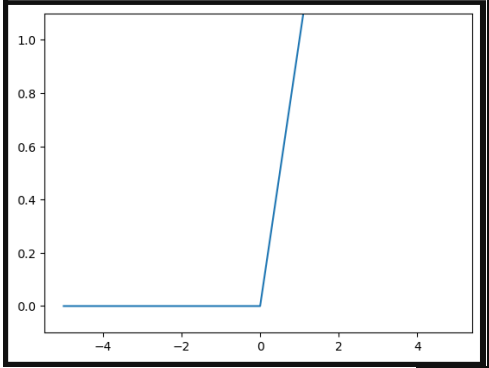
```
    return 1/(1+np.exp(-x))
```

[8]:
```
X=np.arange(-5.0,5.0,0.1)
Y=sigmoid(X)
plt.plot(X,Y)
plt.ylim(-0.1,1.1)
plt.show()
```



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

def relu(x):
    return np.maximum(0,x)
```

```
X=np.arange(-5.0, 5.0, 0.1)
Y=relu(X)
plt.plot(X,Y)
plt.ylim(-0.1,1.1)
plt.show()
```



```
import numpy as np

X=np.array([1.0, 0.5]) # 2개의 입계값
W1 = np.array([[0.1,0.3,0.5],[0.2,0.4,0.6]]) #가중치
B1 = np.array([0.1,0.2,0.3])
```

```
print(W1.shape)
print(X.shape)
print(B1.shape)
```

```
(2, 3)
(2,)
(3,)
```

```
A1 = np.dot(X,W1) + B1
print(A1)
```

```
[0.3 0.7 1.1]
```

```
import numpy as np

#활성화 함수
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
X= np.array([1.0, 0.5])
W1= np.array([[0.1, 0.3, 0.5],[0.2, 0.4, 0.6]])
B1= np.array([0.1, 0.2, 0.3])
```

```
print(W1.shape)
print(X.shape)
print(B1.shape)
```

```
(2, 3)
(2,)
(3,)
```

```
A1 = np.dot(X,W1) + B1
print(A1)
```

```
[0.3 0.7 1.1]
```

```
Z1 = sigmoid(A1) #활성화 처리 (sigmoid 값을 거쳐주었다)

print(A1)
print(Z1)
```

```
[0.3 0.7 1.1]
[0.57444252 0.66818777 0.75026011]
```

```
# 1층의 활성화 함수에서의 처리

X= np.array([1.0, 0.5])
W1= np.array([[0.1, 0.3, 0.5],[0.2, 0.4, 0.6]])
B1= np.array([0.1, 0.2, 0.3])
```

```
print(W1.shape)
print(X.shape)
print(B1.shape)
```

```
(2, 3)
(2,)
(3,)
```

```
A1 = np.dot(X,W1) + B1
print(A1)
```

```
[0.3 0.7 1.1]
```

```
Z1 = sigmoid(A1) #활성화 처리 (sigmoid 값을 거칠수있게)

print(A1)
print(Z1)
```

```
[0.3 0.7 1.1]
[0.57444252 0.66818777 0.75026011]
```

```
# 1층에서 2층으로 신호 전달

W2 = np.array([[0.1, 0.4],[0.2,0.5],[0.3,0.6]])
B2 = np.array([0.1,0.2])
```

```
print(Z1.shape)
print(W2.shape)
print(B2.shape)
```

```
(3,)
(3, 2)
(2,)
```

```
A2 = np.dot(Z1, W2) + B2
Z2 =sigmoid(A2)
```

```
#2층에서 출력층으로 신호 전달
def identity_function(x):
    return x
```

```
#3th layer -> output

W3 = np.array([[0.1, 0.3],[0.2,0.5]])
B3 = np.array([0.1,0.2])

A3 = np.dot(Z2, W3) + B3
Y = identity_function(A3)

print(Y)
print(A3)

# 항등함수라 Y,A3의 값은 동일하다.
```

```
[0.31682708 0.77338016]
[0.31682708 0.77338016]
```

```
##추가

def init_network():
    network ={}
    network['W1'] = np.array([[0.1, 0.3, 0.5],[0.2,0.4,0.6]])
    network['b1'] = np.array([0.1, 0.2, 0.3])
    network['W2'] = np.array([[0.1, 0.4],[0.2,0.5],[0.3,0.6]])
    network['b2'] = np.array([0.1, 0.2])
    network['W3'] = np.array([[0.1, 0.3],[0.2,0.4]])
    network['b3'] = np.array([0.1, 0.2])

    return network
```

```
def forward(network, x):
    W1,W2,W3 = network['W1'], network['W2'], network['W3']
    b1,b2,b3 = network['b1'], network['b2'], network['b3']

    a1= np.dot(x, W1)+b1
    z1=sigmoid(a1)
    a2= np.dot(z1,W2)+b2
    z2=sigmoid(a2)
    a3=np.dot(z2,W3)+b3
    y=identity_function(a3)

    return y #포워드 값
```

```
network = init_network()
x= np.array([1.0, 0.5])
y= forward(network, x)
print(y) # [0.31682708 0.69627909]
```

```
[0.31682708 0.69627909]
```

(김상욱)

- 계단함수
- 시그모이드 함수
- 렐루 함수
- 소프트맥스 함수

[22]:
```
import numpy as np
import matplotlib.pylab as plt
```

[23]:
```
def step(x):
    y = x > 0
    return np.array(x>0, dtype=int)
```

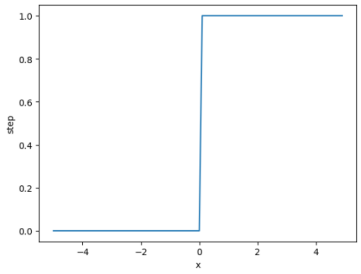
[24]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

[25]:
```
def relu(x):
    return np.maximum(0, x)
```

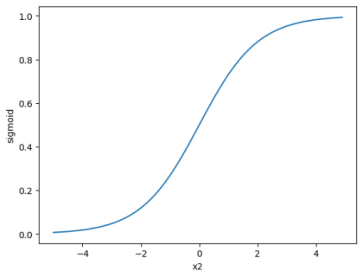
[26]:
```
def softmax(x):
    c = np.max(x)
    exp_x = np.exp(x - c)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x
    return y
```

그래프 구현 ▽

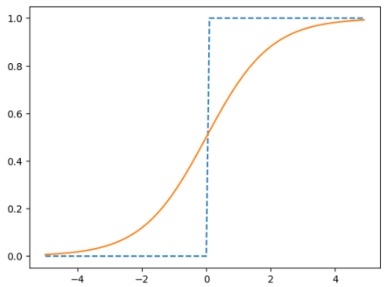
[27]:
```
x = np.arange(-5.0, 5.0, 0.1)
y = step(x)
plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("step")
plt.show()
```



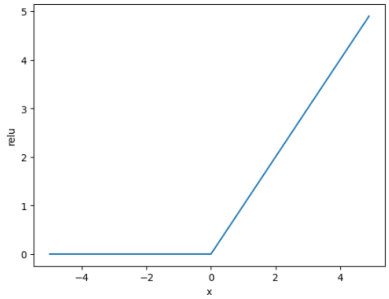
[28]:
```
x2 = np.arange(-5.0, 5.0, 0.1)
y2 = sigmoid(x2)
plt.plot(x2, y2)
plt.xlabel("x2")
plt.ylabel("sigmoid")
plt.show()
```



[29]:
```
plt.plot(x, y, "--")
plt.plot(x2, y2)
plt.show()
```



[30]:
```
relux = np.arange(-5.0, 5.0, 0.1)
reluy = relu(relux)
plt.plot(relux, reluy)
plt.xlabel("x")
plt.ylabel("relu")
plt.show()
```



[32]:
```
a = np.arange(1, 101, 1)
b = np.arange(101, 201, 1)
```

[33]:
```
c = np.array([a, b])
```

[34]:
```
a-100
```

[34]:
```
array([-99, -98, -97, -96, -95, -94, -93, -92, -91, -90, -89, -88, -87,
       -86, -85, -84, -83, -82, -81, -80, -79, -78, -77, -76, -75, -74,
       -73, -72, -71, -70, -69, -68, -67, -66, -65, -64, -63, -62, -61,
       -60, -59, -58, -57, -56, -55, -54, -53, -52, -51, -50, -49, -48,
       -47, -46, -45, -44, -43, -42, -41, -40, -39, -38, -37, -36, -35,
       -34, -33, -32, -31, -30, -29, -28, -27, -26, -25, -24, -23, -22,
       -21, -20, -19, -18, -17, -16, -15, -14, -13, -12, -11, -10,  -9,
        -8,  -7,  -6,  -5,  -4,  -3,  -2,  -1,   0])
```

[35]:
```
c
```

[35]:
```
array([[  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
         14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,
         27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,
         40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,
         53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,
         66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,
         79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,
         92,  93,  94,  95,  96,  97,  98,  99, 100],
       [101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113,
        114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126,
        127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
        140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152,
        153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,
        166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178,
        179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191,
        192, 193, 194, 195, 196, 197, 198, 199, 200]])
```

3층 신경망 ▽

[37]:
```
w = np.array([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])
w2 = np.array([[0.1, 0.4], [0.2, 0.5],  [0.3, 0.6]])
w3 = np.array([[0.1, 0.3], [0.2, 0.5]])

x = np.array([1, 0.5])

b = np.array([0.1, 0.2, 0.3])
b2 = np.array([0.1, 0.2])
b3 = np.array([0.1, 0.2])
```

[38]:
```
# 입력층
print(x)
print(x.shape)
```

```
[1.  0.5]
(2,)
```

[39]:
```
#은닉층 1
y = np.dot(x, w) + b
y = sigmoid(y)
y
```

[39]:
```
array([0.57444252, 0.66818777, 0.75026011])
```

[40]:
```
#은닉층 2
y2 = np.dot(y, w2) + b2
y2 = sigmoid(y2)
y2
```

[40]:
```
array([0.62624937, 0.7710107 ])
```

[41]:
```
# 출력층
y3 = np.dot(y2, w3) + b3
y3
```

[41]:
```
array([0.31682708, 0.77338016])
```