In [41]:

```
def AND_gate(x1, x2):
    w1 = 0.5
    w2 = 0.5
    b = -0.7

    result = x1 * w1 + x2 * w2 + b

    if result <= 0:
        return 0

    else :
        return 1
```

In [42]:

```
print("AND gate에 0, 0 입력 결과", AND_gate(0, 0))
print("AND gate에 0, 1 입력 결과", AND_gate(0, 1))
print("AND gate에 1, 0 입력 결과", AND_gate(1, 0))
print("AND gate에 1, 1 입력 결과", AND_gate(1, 1))
```

```
AND gate에 0, 0 입력 결과 0
AND gate에 0, 1 입력 결과 0
AND gate에 1, 0 입력 결과 0
AND gate에 1, 1 입력 결과 1
```

In [43]:

```
def OR_gate(x1, x2):
    w1 = 0.6
    w2 = 0.6
    b = -0.5

    result = x1 * w1 + x2 * w2 + b

    if result <= 0:
        return 0

    else :
        return 1
```

In [44]:

```
print("OR gate에 0, 0 입력 결과", OR_gate(0, 0))
print("OR gate에 0, 1 입력 결과", OR_gate(0, 1))
print("OR gate에 1, 0 입력 결과", OR_gate(1, 0))
print("OR gate에 1, 1 입력 결과", OR_gate(1, 1))
```

```
OR gate에 0, 0 입력 결과 0
OR gate에 0, 1 입력 결과 1
OR gate에 1, 0 입력 결과 1
OR gate에 1, 1 입력 결과 1
```

In [45]:

```
def NAND_gate(x1, x2):
    w1 = -0.5
    w2 = -0.5
    b = 0.7

    result = x1 * w1 + x2 * w2 + b

    if result <= 0:
        return 0

    else :
        return 1
```

In [46]:

```
print("NAND gate에 0, 0 입력 결과", NAND_gate(0, 0))
print("NAND gate에 0, 1 입력 결과", NAND_gate(0, 1))
print("NAND gate에 1, 0 입력 결과", NAND_gate(1, 0))
print("NAND gate에 1, 1 입력 결과", NAND_gate(1, 1))
```

```
NAND gate에 0, 0 입력 결과 1
NAND gate에 0, 1 입력 결과 1
NAND gate에 1, 0 입력 결과 1
NAND gate에 1, 1 입력 결과 0
```

In [47]:

```
def XOR_gate(x1, x2):
    s1 = NAND_gate(x1, x2)
    s2 = OR_gate(x1, x2)
    y = AND_gate(s1, s2)

    return y
```

In [48]:

```
print("XOR gate에 0, 0 입력 결과", XOR_gate(0, 0))
print("XOR gate에 0, 1 입력 결과", XOR_gate(0, 1))
print("XOR gate에 1, 0 입력 결과", XOR_gate(1, 0))
print("XOR gate에 1, 1 입력 결과", XOR_gate(1, 1))
```

```
XOR gate에 0, 0 입력 결과 0
XOR gate에 0, 1 입력 결과 1
XOR gate에 1, 0 입력 결과 1
XOR gate에 1, 1 입력 결과 0
```

In [49]:

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

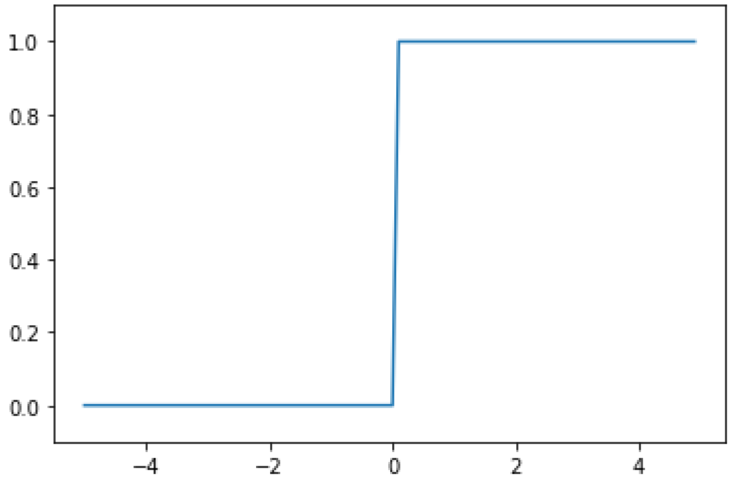
In [50]:

```
def step_function(x):
    y = x > 0

    return y.astype(np.int)
```

In [51]:

```
x = np.arange(-5, 5, 0.1)
y = step_function(x)

plt.plot(x, y)
plt.ylim(-0.1, 1.1)
plt.show()
```

```
C:₩Users₩wogml₩AppData₩Local₩Temp₩ipykernel_2596₩4154412597.py:4: DeprecationWarnin
g: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, us
e `int` by itself. Doing this will not modify any behavior and is safe. When replaci
ng `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precis
ion. If you wish to review your current use, check the release note link for additio
nal information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/r
elease/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/1.20.0-note
s.html#deprecations)
  return y.astype(np.int)
```

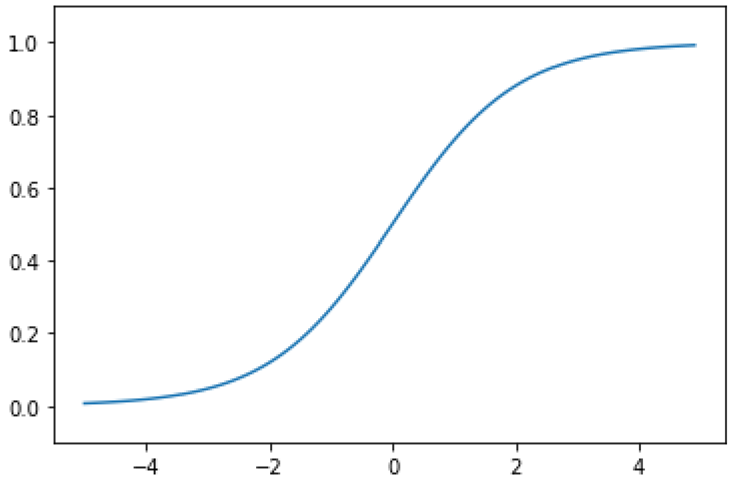


In [52]:

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

In [53]:

```
x = np.arange(-5, 5, 0.1)
y = logistic_function(x)

plt.plot(x, y)
plt.ylim(-0.1, 1.1)
plt.show()
```

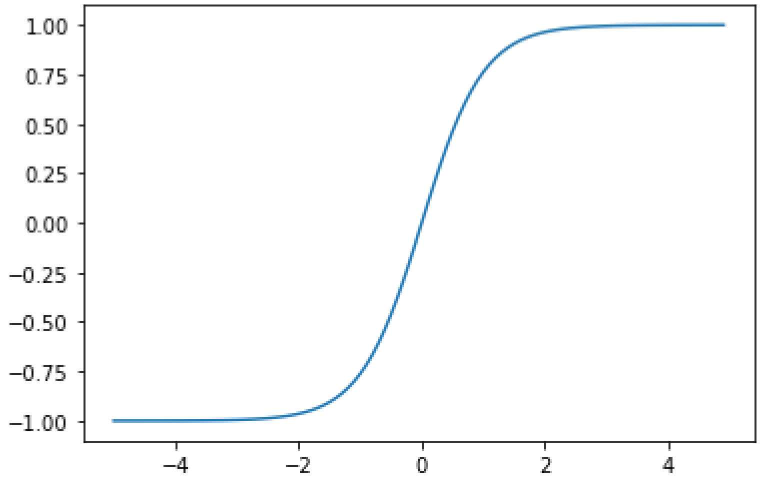


In [54]:

```
def tanh_function(x):
    return ((np.exp(x) - np.exp(-x)) / ((np.exp(x) + np.exp(-x)) ))
```

In [55]:

```
x = np.arange(-5, 5, 0.1)
y = tanh_function(x)

plt.plot(x, y)
plt.ylim(-1.1, 1.1)
plt.show()
```

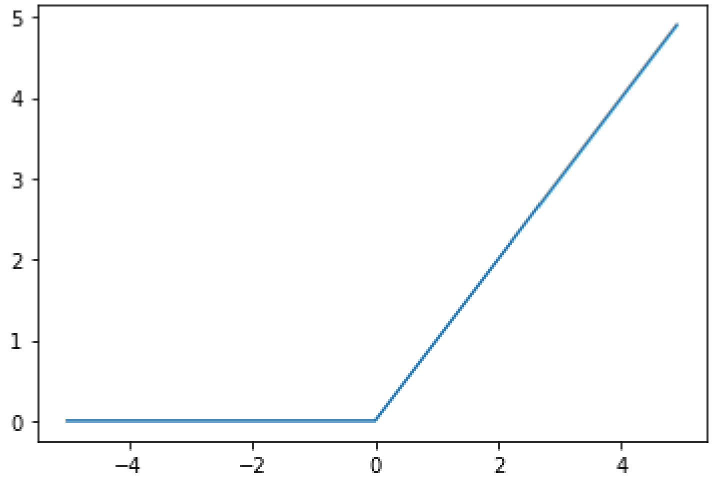


In [56]:

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

In [57]:

```
x = np.arange(-5, 5, 0.1)
y = ReLu_function(x)

plt.plot(x, y)
plt.show()
```

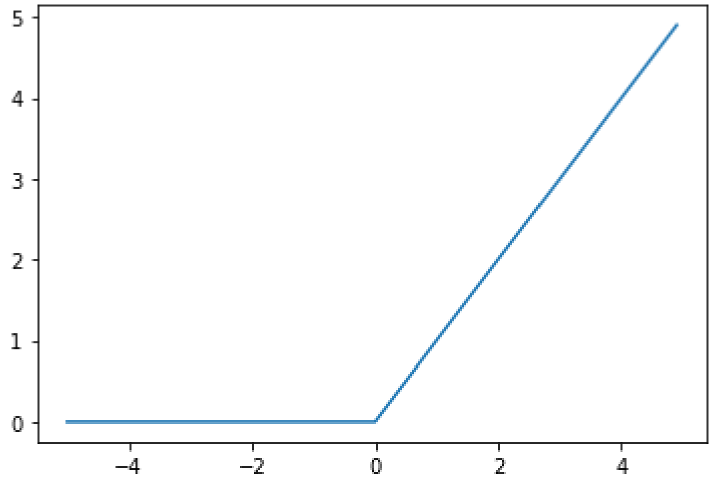


In [58]:

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

In [59]:

```
x = np.arange(-5, 5, 0.1)
y = Leaky_ReLu_function(x)

plt.plot(x, y)
plt.show()
```

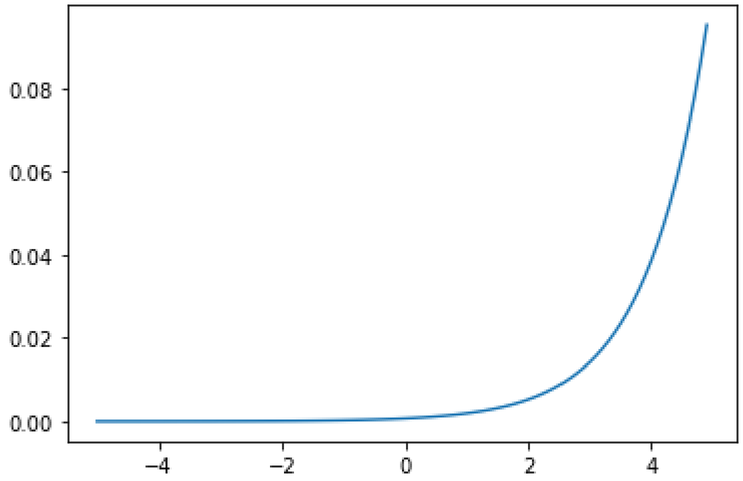


In [60]:

```
def softmax_function(x):
    exp_x = np.exp(x)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x

    return y
```

In [61]:

```
x = np.arange(-5, 5, 0.1)
y = softmax_function(x)

plt.plot(x, y)
plt.show()
```

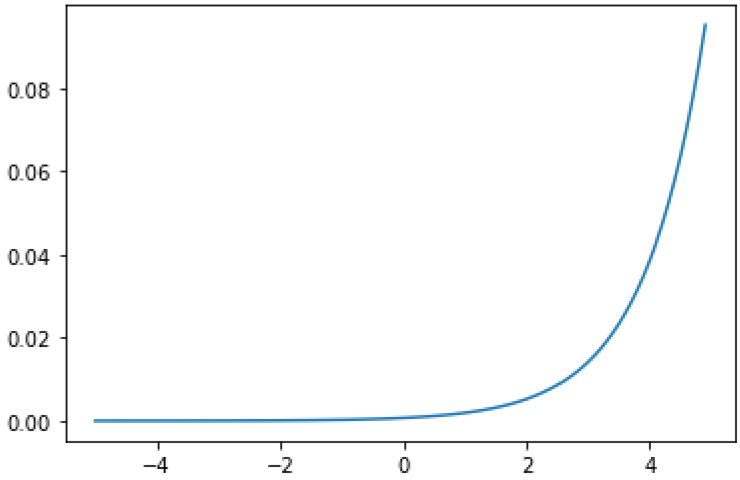


In [62]:

```
def softmax_function(x):
    input_max = np.max(x)
    exp_x = np.exp(x - input_max)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x

    return y
```

In [63]:

```
x = np.arange(-5, 5, 0.1)
y = softmax_function(x)

plt.plot(x, y)
plt.show()
```



In [84]:

```
def sigmoid_funcion(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("x의 형태(모양) : ", x.shape)
print("w1의 형태(모양) : ", w1.shape)
print("b1의 형태(모양) : ", b1.shape)

a1 = np.dot(x, w1) + b1
z1 = sigmoid_funcion(a1)

print("a1 : ", a1)
print("z1 : ", z1)
```

```
x의 형태(모양) :  (2,)
w1의 형태(모양) :  (2, 3)
b1의 형태(모양) :  (3,)
a1 :  [0.3 0.7 1.1]
z1 :  [0.57444252 0.66818777 0.75026011]
```

In [80]:

```
w2 = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
b2 = np.array([0.1, 0.2])

print("w2의 형태(모양) : ", w2.shape)
print("b2의 형태(모양) : ", b2.shape)

a2 = np.dot(z1, w2) + b2
z2 = sigmod_funcion(a2)

print("a2 : ", a2)
print("z2 : ", z2)
```

```
w2의 형태(모양) :  (3, 2)
b2의 형태(모양) :  (2,)
a2 :  [0.51615984 1.21402696]
z2 :  [0.62624937 0.7710107 ]
```

In [81]:

```
w3 = np.array([[0.1, 0.3], [0.2, 0.4]])
b3 = np.array([0.1, 0.2])

a3 = np.dot(z2, w3) + b3
y = sigmod_funcion(a3)

print("a3 : ", a3)
print("y(최종 출력값) : ", y)
```

```
a3 :  [0.31682708 0.69627909]
y(최종 출력값) :  [0.57855079 0.66736228]
```

In [82]:

```
import numpy as np
import pickle
from dataset.mnist import load_mnist

(x_train, y_train), (x_test, y_test) = load_mnist(normalize = True, flatten = True)

def sigmoid_funcion(x) :
    return 1 / (1 + np.exp(-x))

def softmax_function(x):
    exp_x = np.exp(x)
    sum_exp_x = np.sum(exp_x)
    y = exp_x / sum_exp_x
    return y

def init_network():
    with open ("./sample_weight.pkl", 'rb') as f:
        network = pickle.load(f)
    return network
```

In [85]:

```
def predict(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_funcion(a1)

    a2 = np.dot(z1, W2) + b2
    z2 = sigmoid_funcion(a2)

    a3 = np.dot(z2, W3) + b3
    y = softmax_function(a3)


    return y
```

In [86]:

```
network = init_network()

print("미리 저장된 네트워크 (가중치, 편향) : ", network, sep = '\n')
```

```
미리 저장된 네트워크 (가중치, 편향) :
{'b2': array([-0.01471108, -0.07215131, -0.00155692,  0.12199665,  0.11603302,
       -0.00754946,  0.04085451, -0.08496164,  0.02898045,  0.0199724 ,
        0.19770803,  0.04365116, -0.06518728, -0.05226324,  0.0113163 ,
        0.03049979,  0.04060355,  0.0695399 , -0.07778469,  0.0692313 ,
       -0.09365533,  0.0548001 , -0.03843745,  0.02123107,  0.03793406,
       -0.02806267, -0.01818407,  0.06870425,  0.0542943 ,  0.0674368 ,
        0.06264312, -0.0233236 , -0.01589135,  0.01860516,  0.01839287,
       -0.01568104, -0.07422207, -0.01606729, -0.02262172, -0.01007509,
        0.0434415 , -0.12020151,  0.02802471, -0.07591944, -0.00533499,
       -0.08935217, -0.0181419 ,  0.0330689 , -0.01812706, -0.07689384,
       -0.02715412, -0.03847084, -0.05315471, -0.02153288,  0.06898243,
        0.02431128, -0.00333816,  0.00817491,  0.03911701, -0.02924617,
        0.07184725, -0.00356748,  0.02246175,  0.03987982, -0.04921926,
        0.02454282,  0.05875788,  0.08505439, -0.00190306, -0.03044275,
       -0.06383366,  0.0470311 , -0.12005549,  0.03573952, -0.04293387,
        0.03283867, -0.03347731, -0.13659105, -0.00123189,  0.00096832,
        0.04590394, -0.02517798, -0.02073979,  0.02005584,  0.010629  ,
        0.01902938, -0.01046924,  0.05777885,  0.04737163, -0.04362756,
        0.07450858,  0.05077952,  0.06648825,  0.04064002,  0.00265163
```

In [87]:

```
accuracy_cnt = 0

for i in range(len(x_test)):
    y = predict(network, x_test[i])
    p = np.argmax(y)

    if p == y_test[i]:
        accuracy_cnt += 1

print("정확도 : " + str(round((float(accuracy_cnt) / len(x_test)) * 100, 3)) + "%")
```

```
정확도 : 93.52%
```