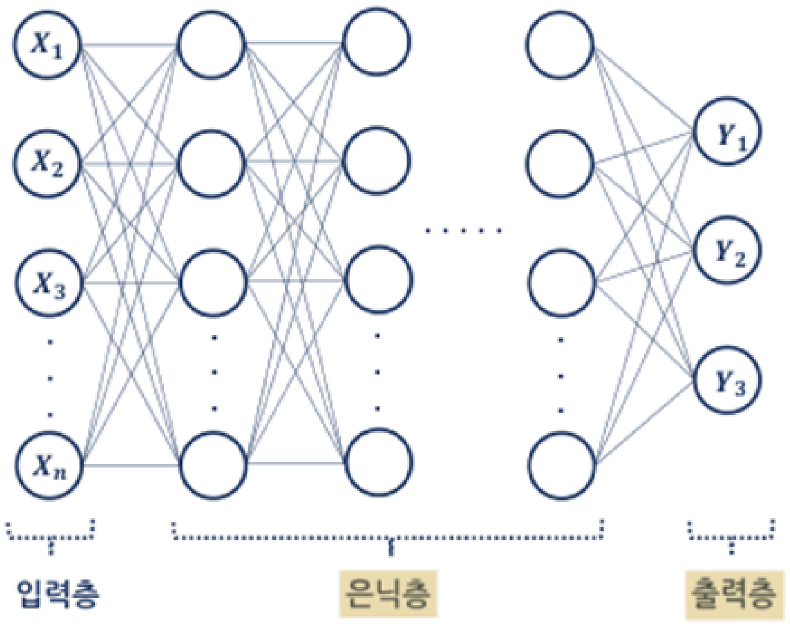
18강. 신경망에서의 활성함수와 손실함수

- 은닉층에서의 활성함수
- 출력층에서의 활성함수
- 네트워크의 손실함수

신경망에서의 활성함수와 손실함수



은닉층에서의 활성함수

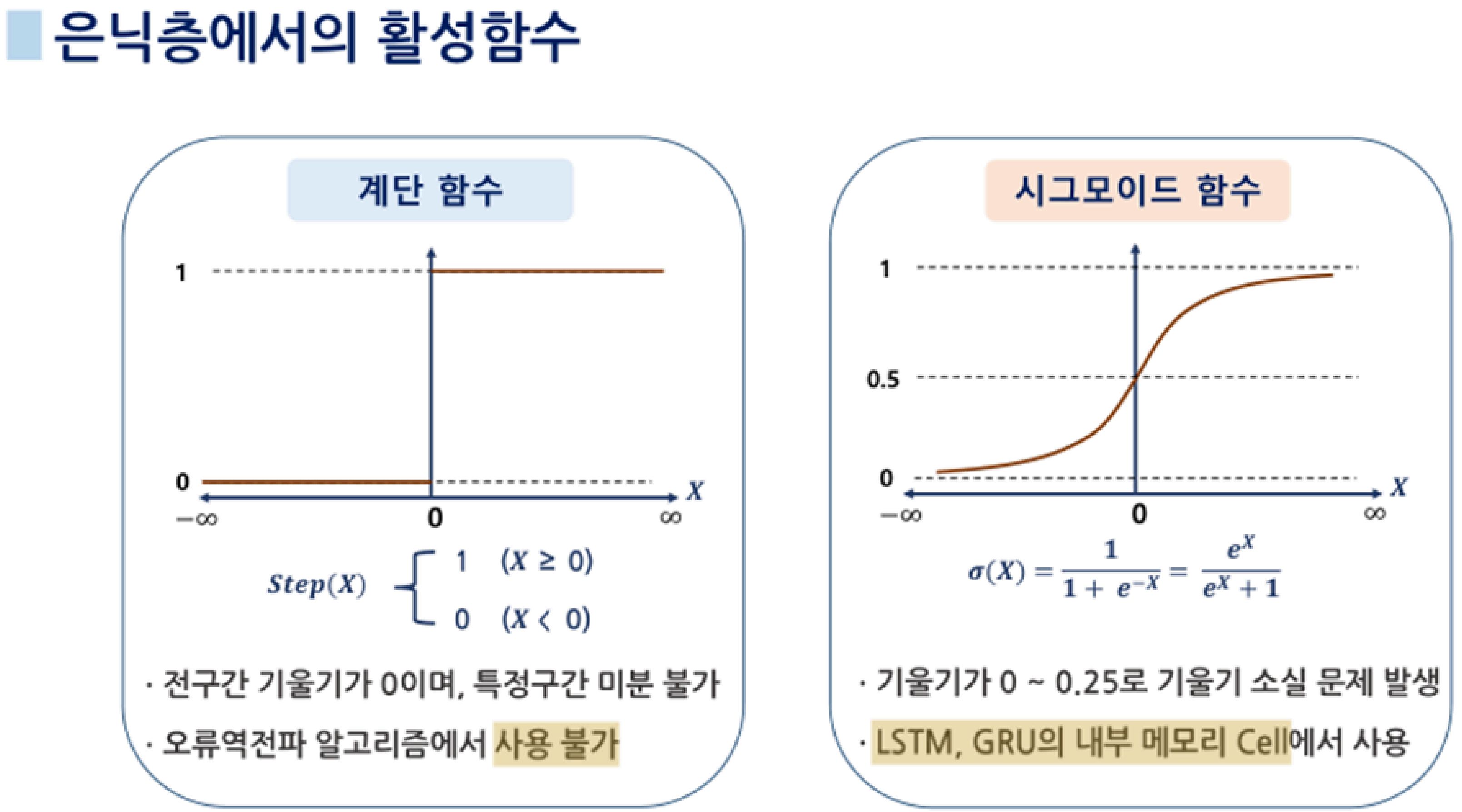
- 비선형 문제의 해결이 목적
- 기울기 소실 문제의 해결을 고려
- 신경망의 유형에 따른 적용

출력층에서의 활성함수

- 신경망의 목적에 맞는 함수 선정

네트워크의 손실함수

- 출력층의 활성함수와 손실함수는 Pair



계단 함수

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

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

```
if __name__ == '__main__':
    x = np.arange(-5.0, 5.0, 0.1)
    y = step_function(x)
```

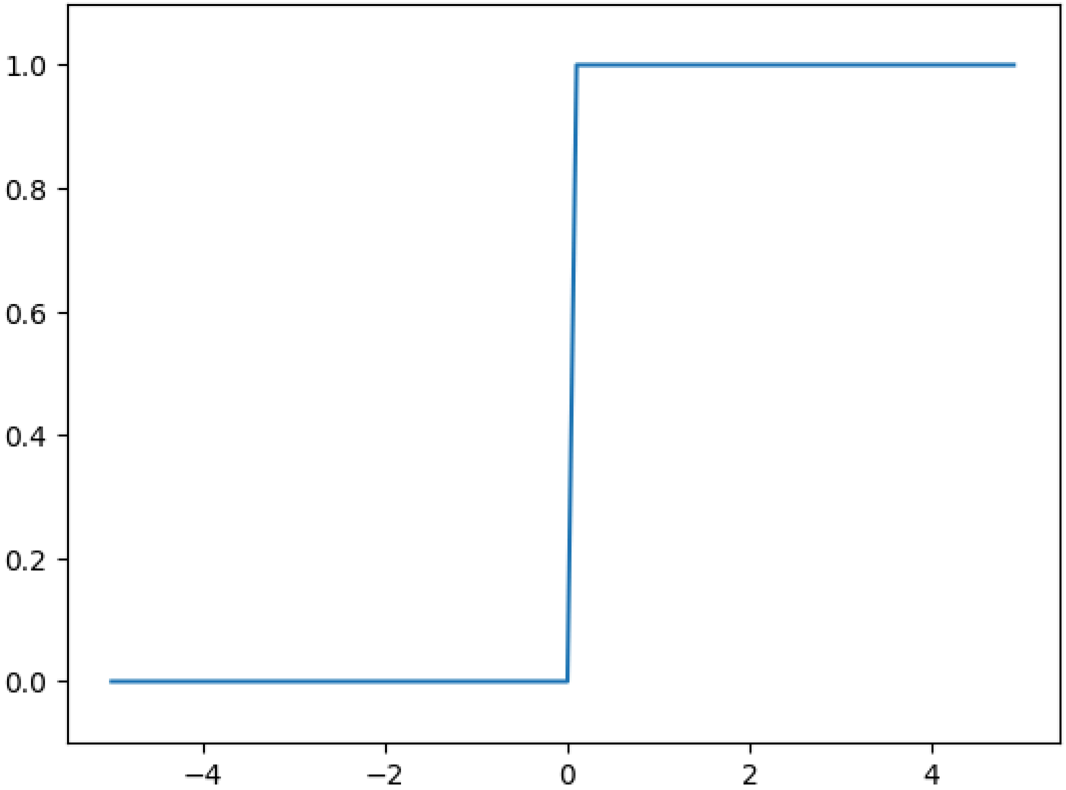
```
print(x)
print(x.shape)
```

```
[-5.00000000e+00 -4.90000000e+00 -4.80000000e+00 -4.70000000e+00
 -4.60000000e+00 -4.50000000e+00 -4.40000000e+00 -4.30000000e+00
 -4.20000000e+00 -4.10000000e+00 -4.00000000e+00 -3.90000000e+00
 -3.80000000e+00 -3.70000000e+00 -3.60000000e+00 -3.50000000e+00
 -3.40000000e+00 -3.30000000e+00 -3.20000000e+00 -3.10000000e+00
 -3.00000000e+00 -2.90000000e+00 -2.80000000e+00 -2.70000000e+00
 -2.60000000e+00 -2.50000000e+00 -2.40000000e+00 -2.30000000e+00
 -2.20000000e+00 -2.10000000e+00 -2.00000000e+00 -1.90000000e+00
 -1.80000000e+00 -1.70000000e+00 -1.60000000e+00 -1.50000000e+00
 -1.40000000e+00 -1.30000000e+00 -1.20000000e+00 -1.10000000e+00
 -1.00000000e+00 -9.00000000e-01 -8.00000000e-01 -7.00000000e-01
 -6.00000000e-01 -5.00000000e-01 -4.00000000e-01 -3.00000000e-01
 -2.00000000e-01 -1.00000000e-01 -1.77635684e-14  1.00000000e-01
  2.00000000e-01  3.00000000e-01  4.00000000e-01  5.00000000e-01
  6.00000000e-01  7.00000000e-01  8.00000000e-01  9.00000000e-01
  1.00000000e+00  1.10000000e+00  1.20000000e+00  1.30000000e+00
  1.40000000e+00  1.50000000e+00  1.60000000e+00  1.70000000e+00
  1.80000000e+00  1.90000000e+00  2.00000000e+00  2.10000000e+00
  2.20000000e+00  2.30000000e+00  2.40000000e+00  2.50000000e+00
  2.60000000e+00  2.70000000e+00  2.80000000e+00  2.90000000e+00
  3.00000000e+00  3.10000000e+00  3.20000000e+00  3.30000000e+00
  3.40000000e+00  3.50000000e+00  3.60000000e+00  3.70000000e+00
  3.80000000e+00  3.90000000e+00  4.00000000e+00  4.10000000e+00
  4.20000000e+00  4.30000000e+00  4.40000000e+00  4.50000000e+00
  4.60000000e+00  4.70000000e+00  4.80000000e+00  4.90000000e+00]
(100,)
```

```
print(y)
print(y.shape)
```

```
[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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
(100,)
```

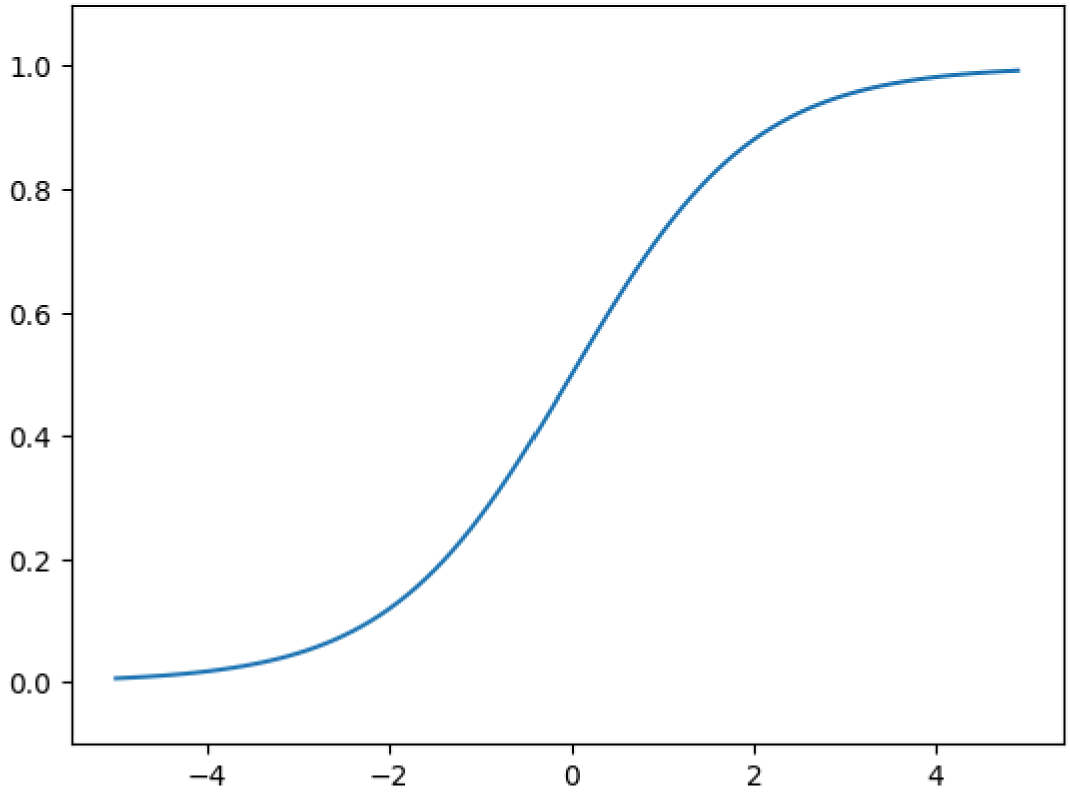
```
plt.plot(x, y)
plt.ylim(-0.1, 1.1)
plt.show()
```

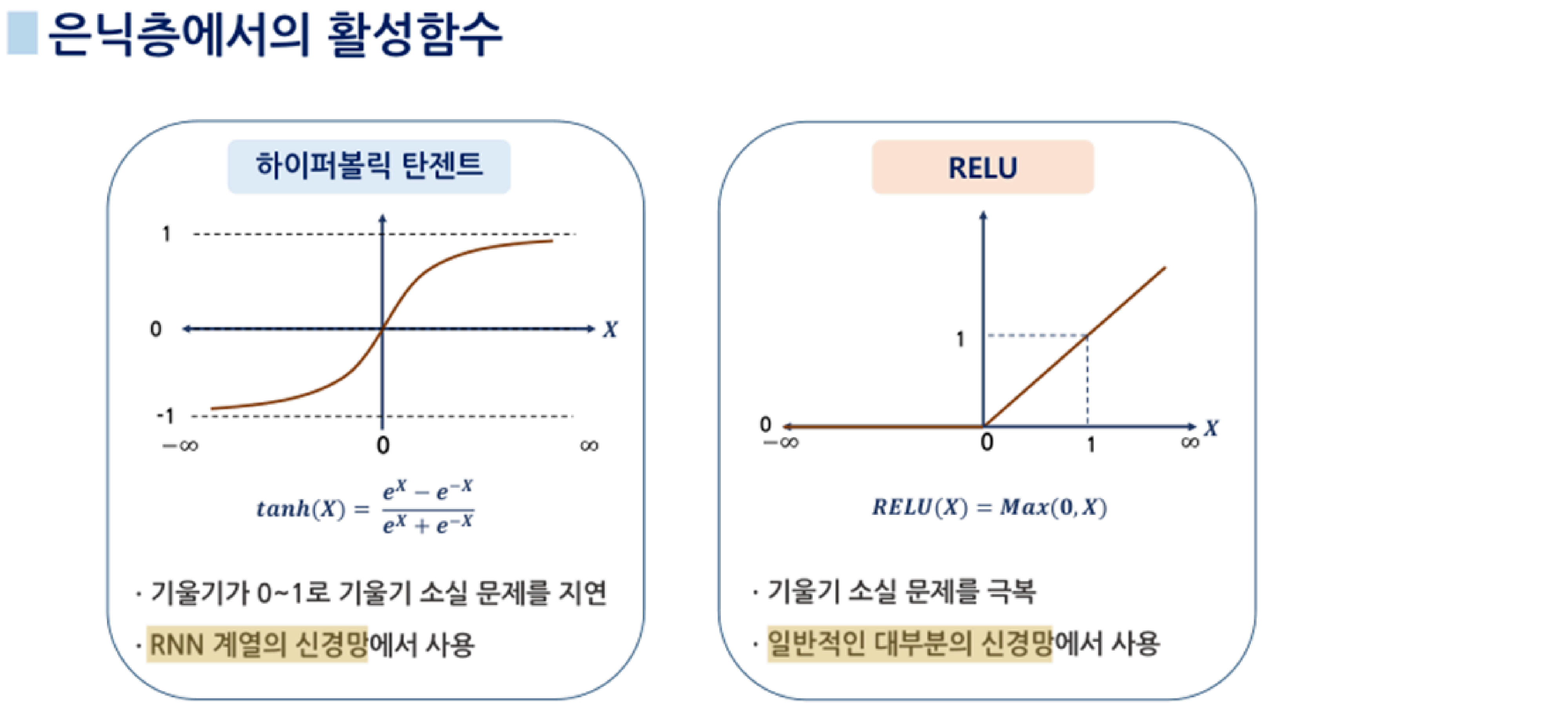


시그모이드 함수

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

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

x = np.array([-1.0, 1.0, 2.0])
y = sigmoid(x)
print(y)

x = np.arange(-5.0, 5.0, 0.1)
y = sigmoid(x)
plt.plot(x, y)
plt.ylim(-0.1, 1.1)
plt.show()
```

```
[0.26894142 0.73105858 0.88079708]
```



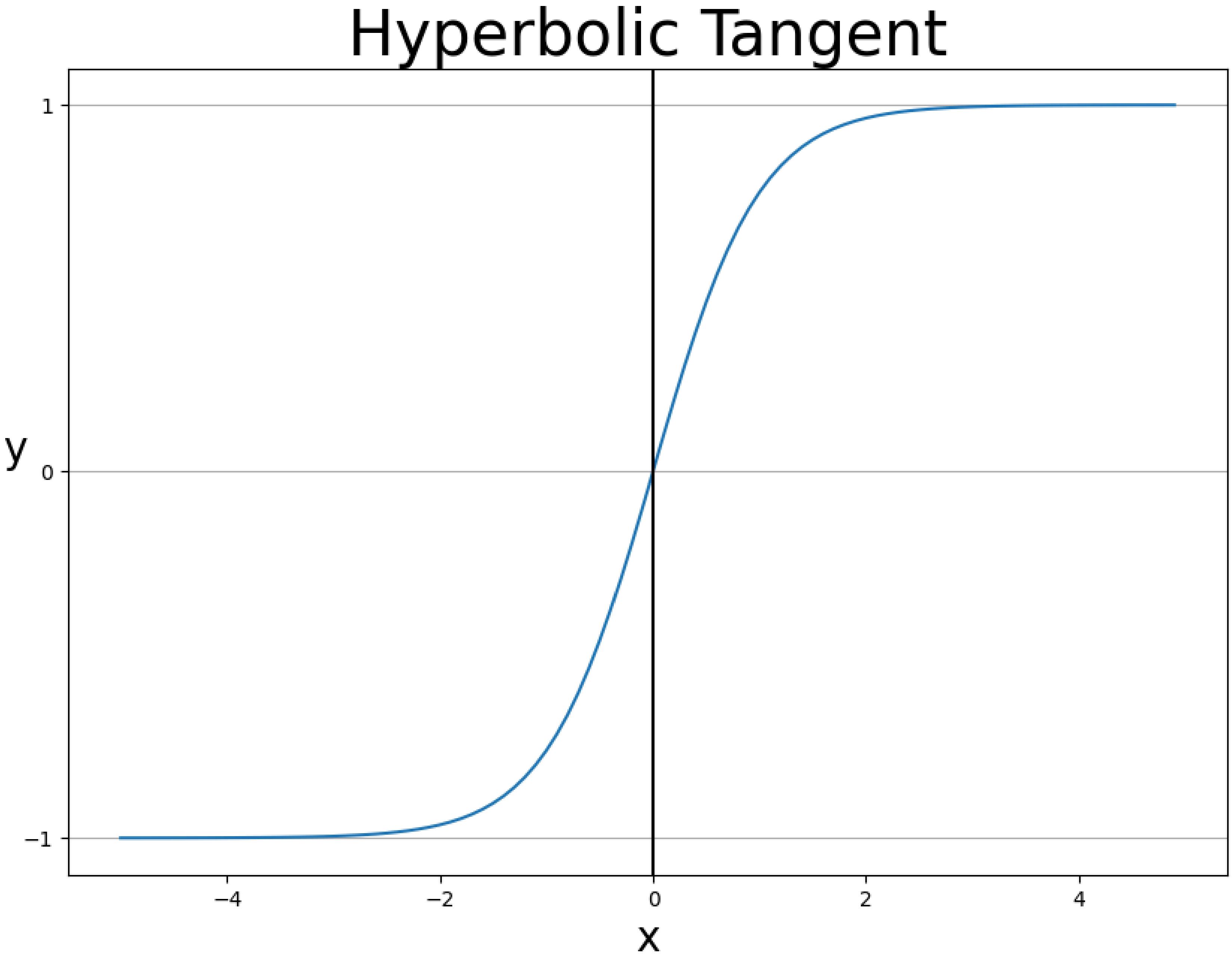


하이퍼볼릭 탄젠트 함수

```
import numpy as np
def tanh(x):
    p_exp_x =  np.exp(x)
    m_exp_x = np.exp(-x)
    y = (p_exp_x - m_exp_x) / (p_exp_x + m_exp_x)
    return y
```

```
import matplotlib.pyplot as plt
x = np.arange(-5.0, 5.0, 0.1)
y = tanh(x)
```

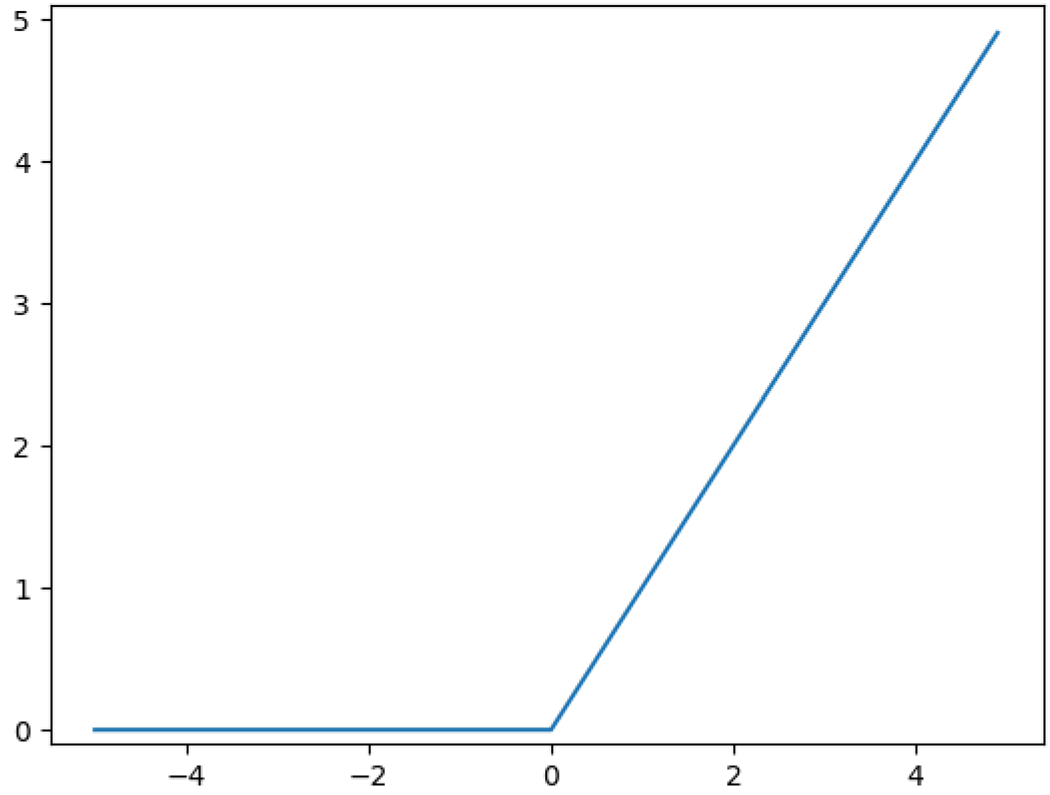
```
fig = plt.figure(figsize=(10, 7))
fig.set_facecolor('white')

plt.plot(x, y)
plt.title("Hyperbolic Tangent", fontsize=30)
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20, rotation=0)

plt.yticks([-1.0, 0.0, 1.0])
plt.axvline(0.0, color='k')

ax = plt.gca()
ax.yaxis.grid(True)

plt.show()
```

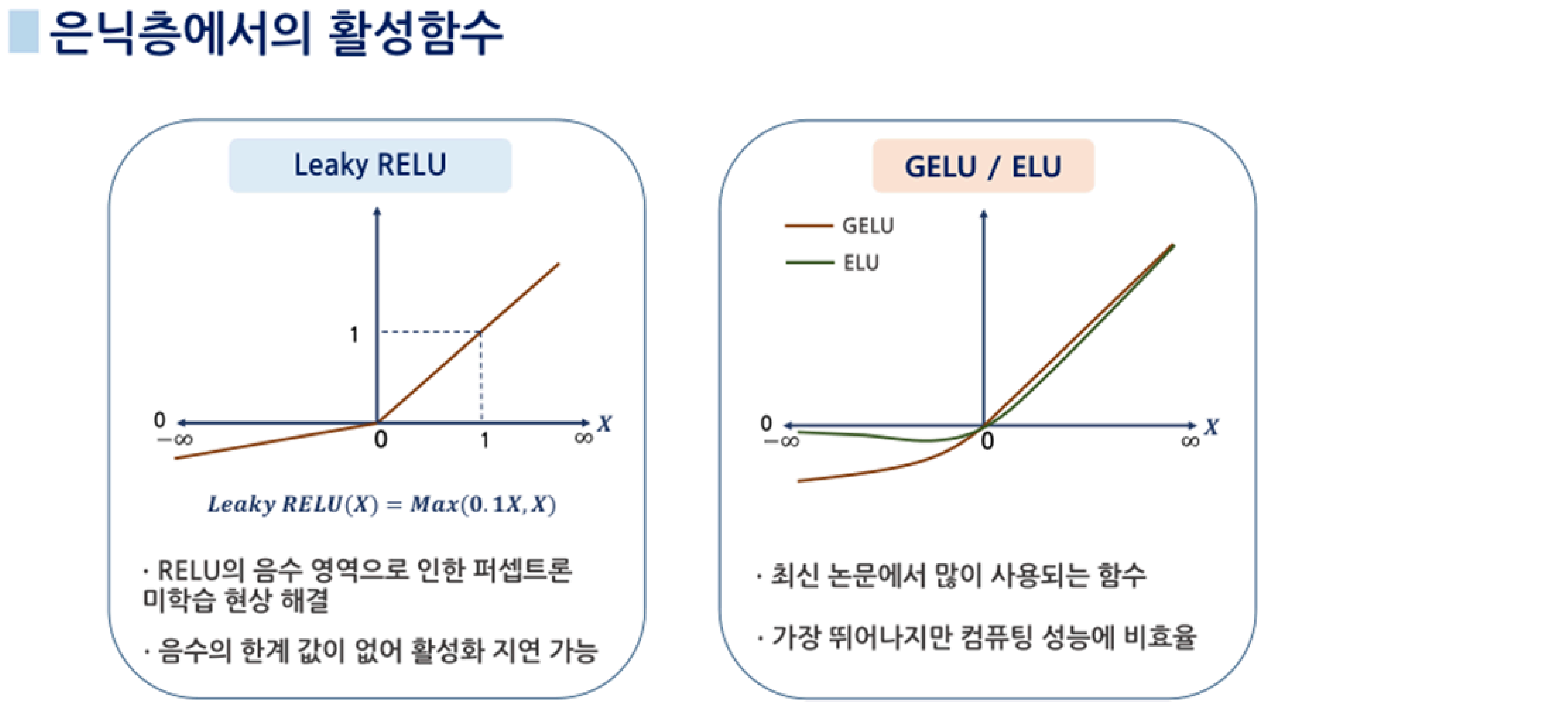

Hyperbolic Tangent
1
y
0
−1
−4
−2
0
2
4
x

렐루 함수

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


def ReLU(x):
    return np.maximum(0, x)


x = np.array([-1.0, 1.0, 2.0])
print(ReLU(x))

x = np.arange(-5.0, 5.0, 0.1)
y = ReLU(x)
plt.plot(x, y)
plt.ylim(-0.1, 5.1)
plt.show()
```

```
[0. 1. 2.]
```



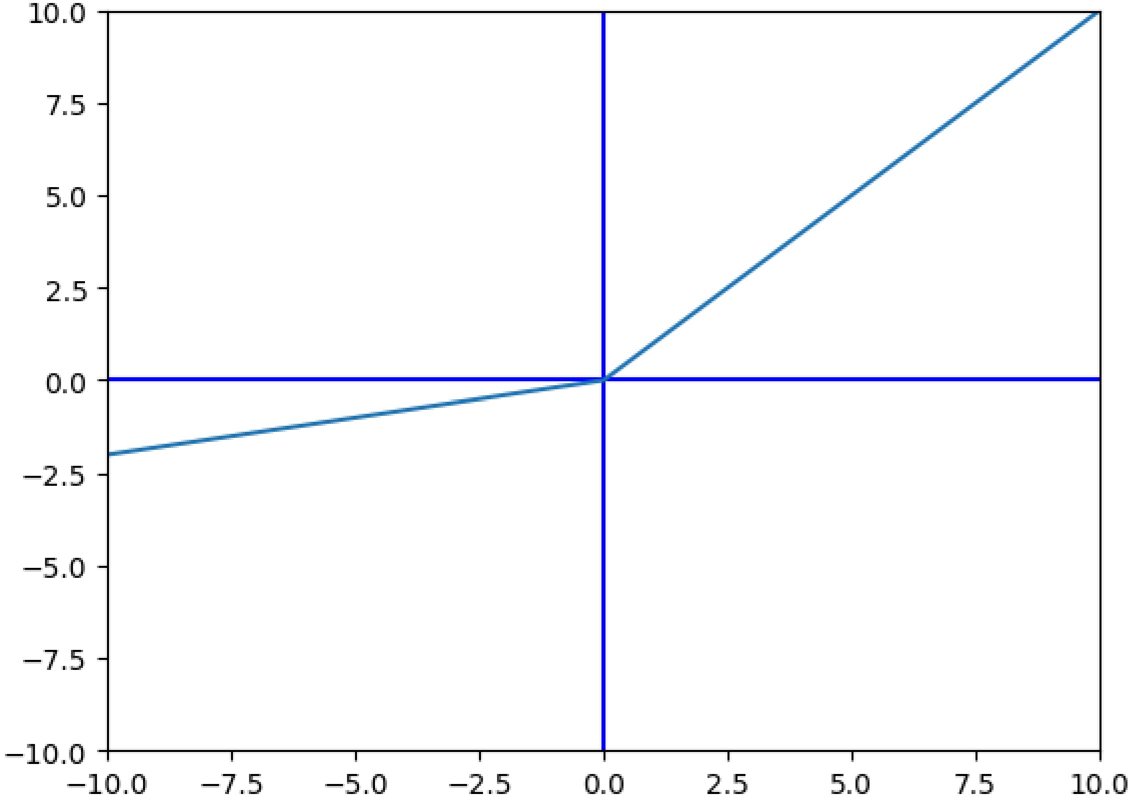


Leaky ReLU

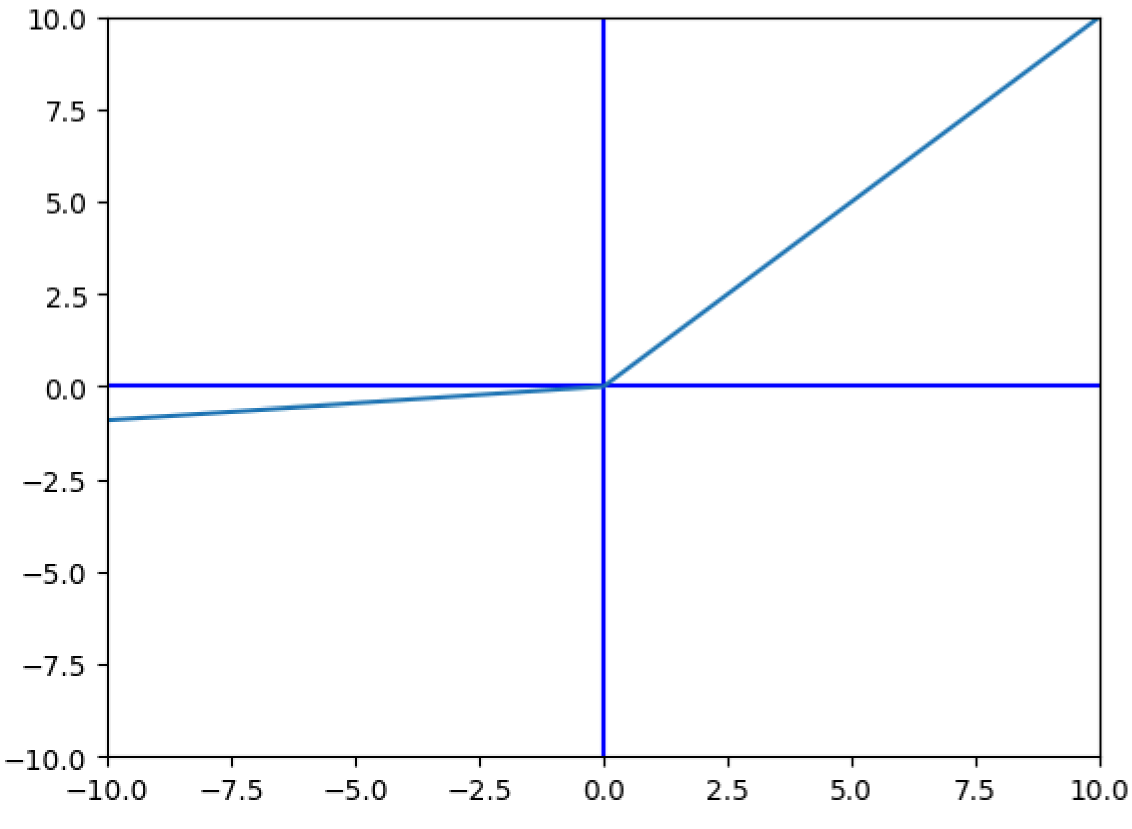
```
def leakyrelu(A,x):
  if x<0:
    return A*x
```

```
  else:
    return x
```

```
X=[x for x in range(-10,11)]
Y=[leakyrelu(0.2,x) for x in range(-10,11)]
plt.xlim((-10,10))
plt.ylim((-10,10))
plt.plot([0,0],[-10,10],color='blue')
plt.plot([-10,10],[0,0],color='blue')
plt.plot(X,Y)
plt.show()
```



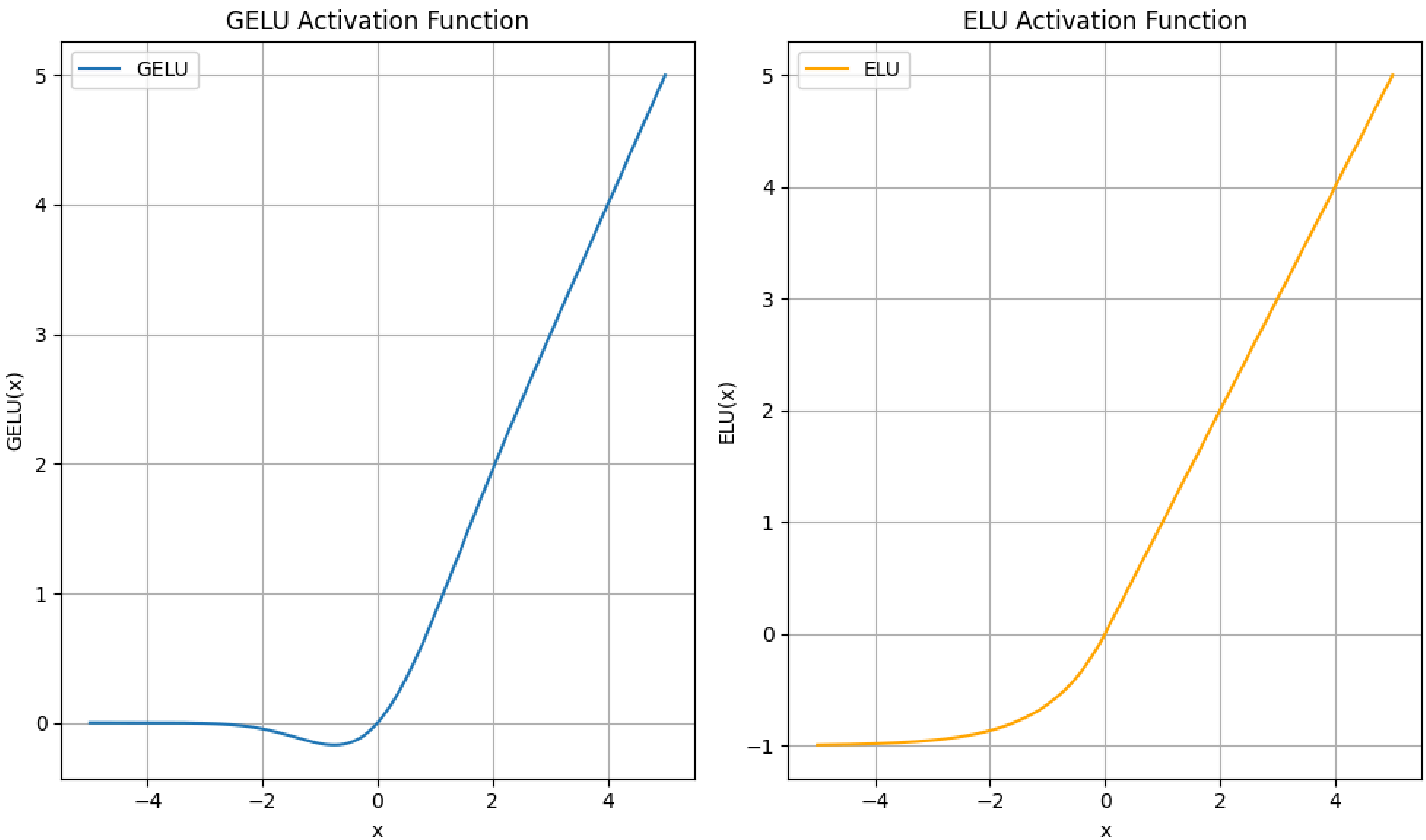
```
X=[x for x in range(-10,11)]
Y=[leakyrelu(0.09,x) for x in range(-10,11)]
plt.xlim((-10,10))
plt.ylim((-10,10))
plt.plot([0,0],[-10,10],color='blue')
plt.plot([-10,10],[0,0],color='blue')
plt.plot(X,Y)
plt.show()
```

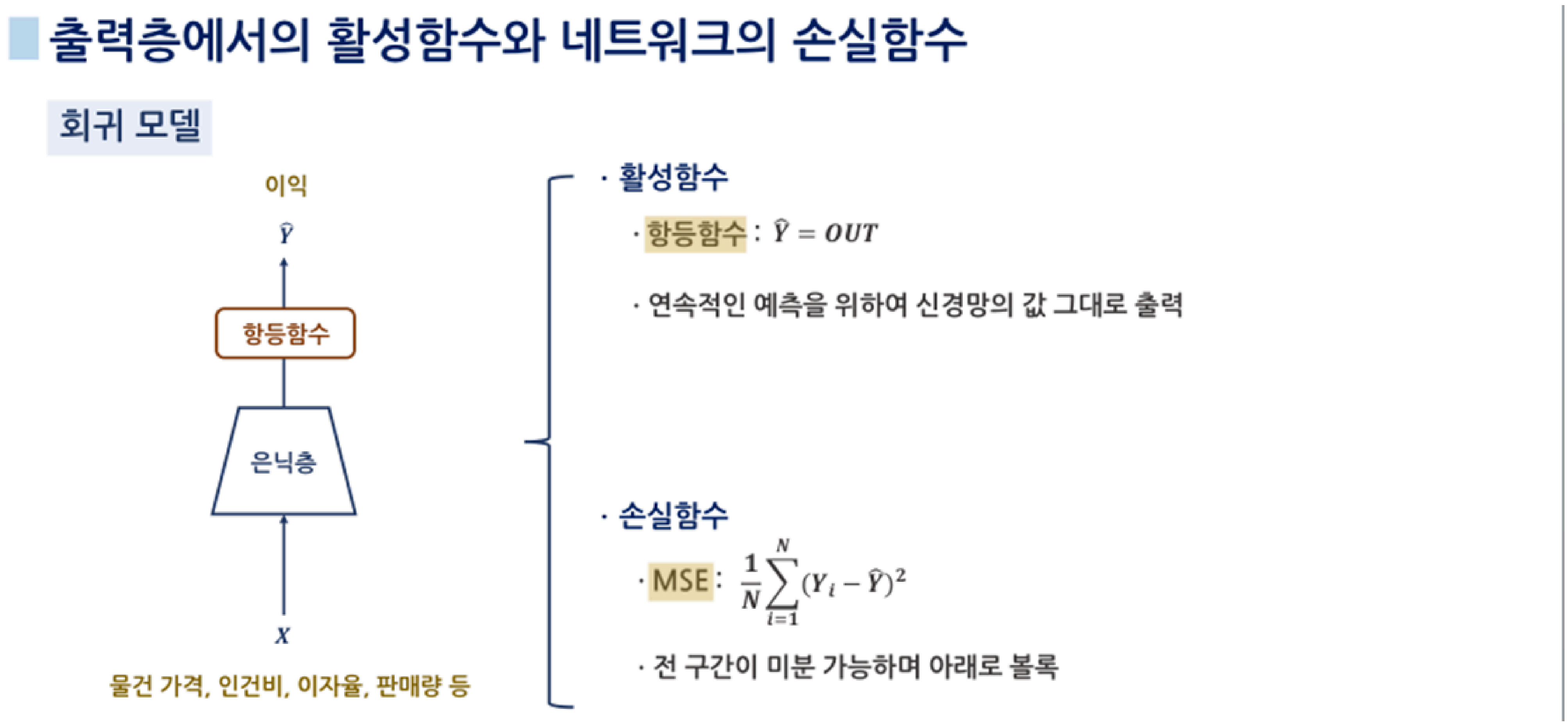


GELU

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import erf

def gelu(x):
    """Gaussian Error Linear Unit (GELU) activation function."""
    return 0.5 * x * (1 + erf(x / np.sqrt(2)))

def elu(x, alpha=1.0):
  """Exponential Linear Unit (ELU) activation function."""
  return np.where(x > 0, x, alpha * (np.exp(x) - 1))

# Generate values for x-axis
x = np.linspace(-5, 5, 400)

# Calculate GELU and ELU values
gelu_values = gelu(x)
elu_values = elu(x)

# Plotting
plt.figure(figsize=(10, 6))

plt.subplot(1, 2, 1)
plt.plot(x, gelu_values, label='GELU')
plt.title('GELU Activation Function')
plt.xlabel('x')
plt.ylabel('GELU(x)')
plt.grid(True)
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(x, elu_values, label='ELU', color='orange')
plt.title('ELU Activation Function')
plt.xlabel('x')
plt.ylabel('ELU(x)')
plt.grid(True)
plt.legend()
```

```

plt.tight_layout()
plt.show()
```





출력층 함수란?

- 흘러온 확률의 숫자를 취합해 결론을 내는 함수
- 신경망으로 구현하고자 하는 문제에 따라 사용하는 출력층 함수가 다름
- 회귀(Regression)의 경우, 항등함수를 사용

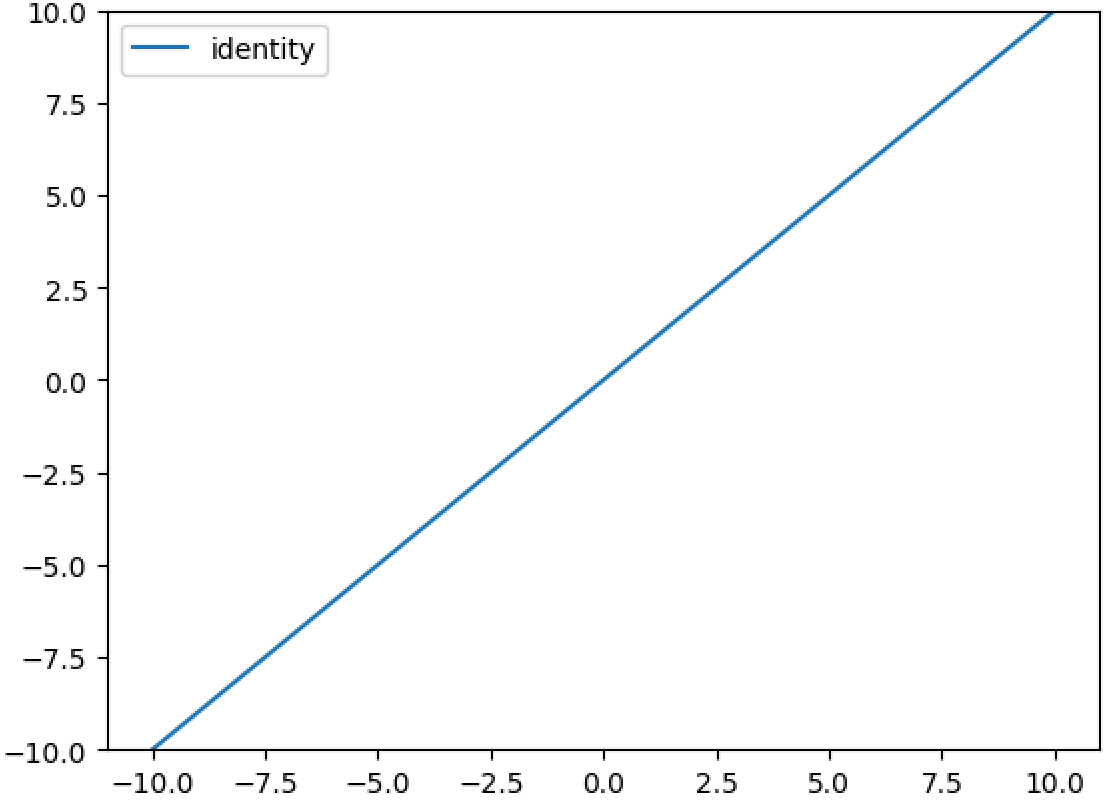
항등함수란

- 어떤 변수도 자기 자신을 함수값으로 하는 함수

```
def identity(x):
    return x

identity(30)
```

```
30
```

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

def identity_func(x):
    return x

x = np.arange(-10, 10, 0.01)
plt.plot(x, identity_func(x), linestyle='-', label="identity")
plt.ylim(-10, 10)
plt.legend()
plt.show()
```



MSE(Mean Squared Error, 평균 제곱근 오차)

$$(\frac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2$$

```
p = np.array([3,4,5])  # 예측값
act = np.array([1,2,3])  # 실제값 (오차가 있는것!)

```

```
def my_mse(pred, actual):
  return ((pred-actual) ** 2).mean()

print(my_mse(p, act))    # 수가 크던 작던 상관x, 다른 값과 비교대상으로 사용하기 위함(상대적으로 적은 값이 더 실제값과 가깝다라고 평가)
```

```
4.0
```

MAE(Mean Absolute Error, 평균 절대값 오차)

$$(\frac{1}{n})\sum_{i=1}^{n}|y_i - x_i|$$

```
def my_mae(pred, actual):
  return np.abs(pred-actual).mean()

print(my_mae(p, act))
```

```
2.0
```

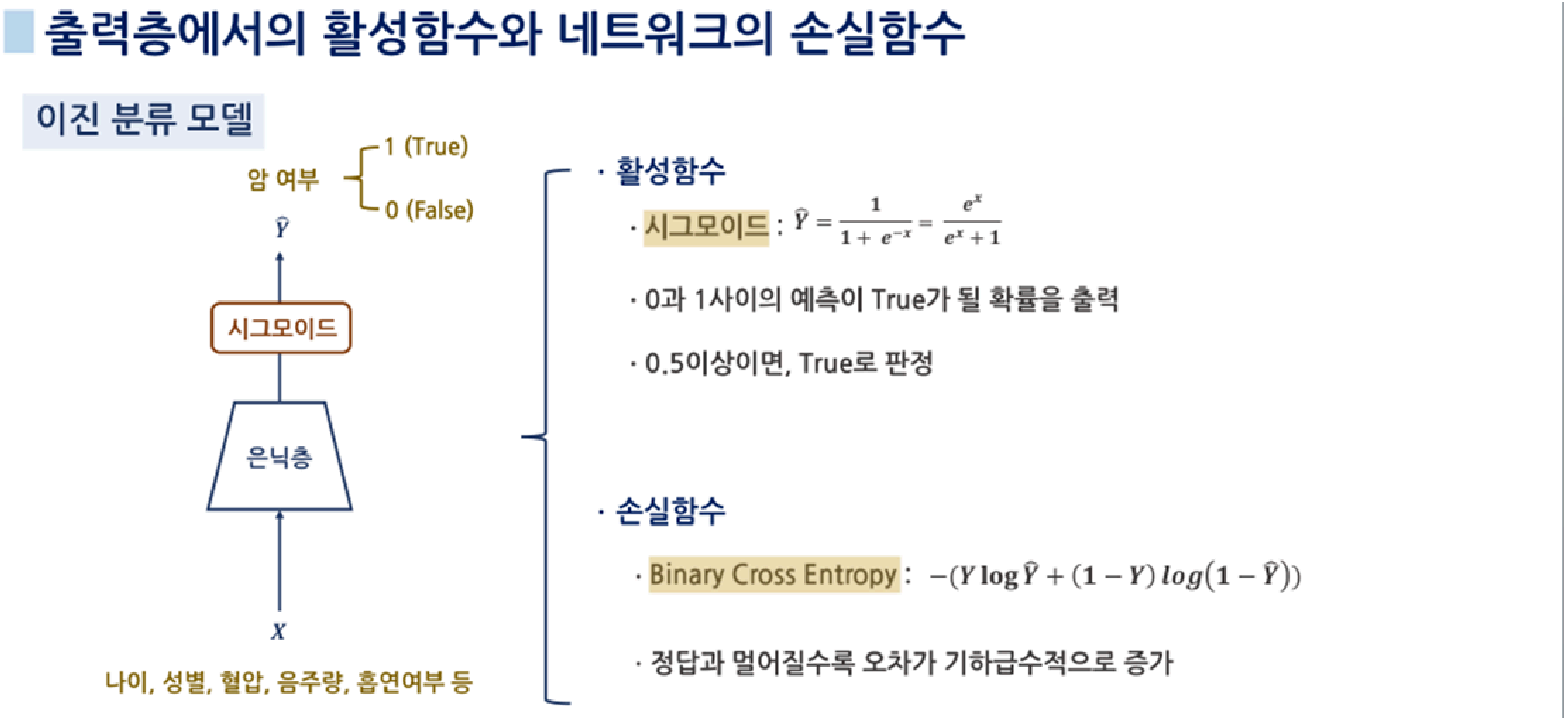
RMSE(Root Mean Squared Error)

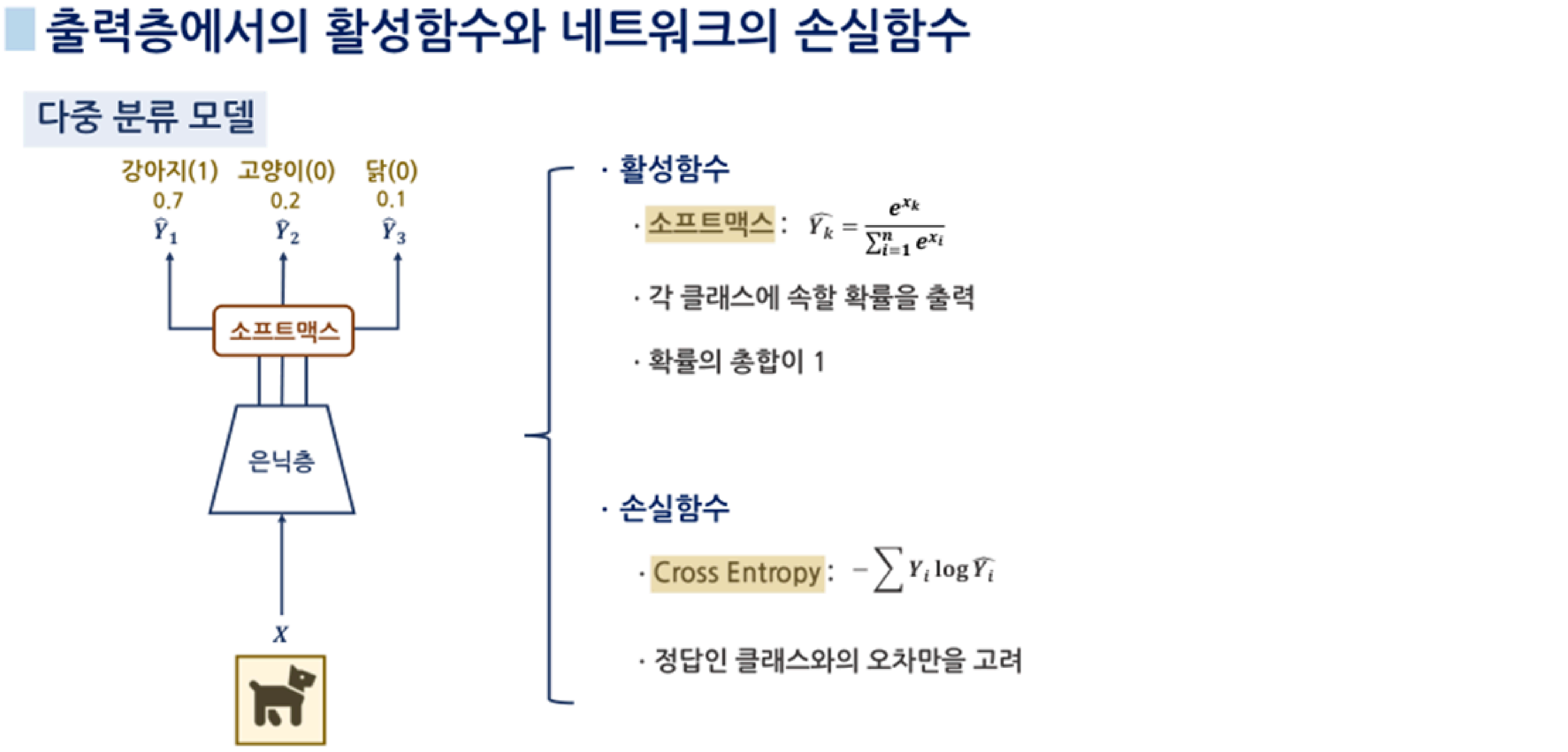
$$\sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2}$$

```
def my_rmse(pred,actual):
  return np.sqrt(my_mse(pred, actual))
```

```
print(my_rmse(p, act))
```

```
2.0
```

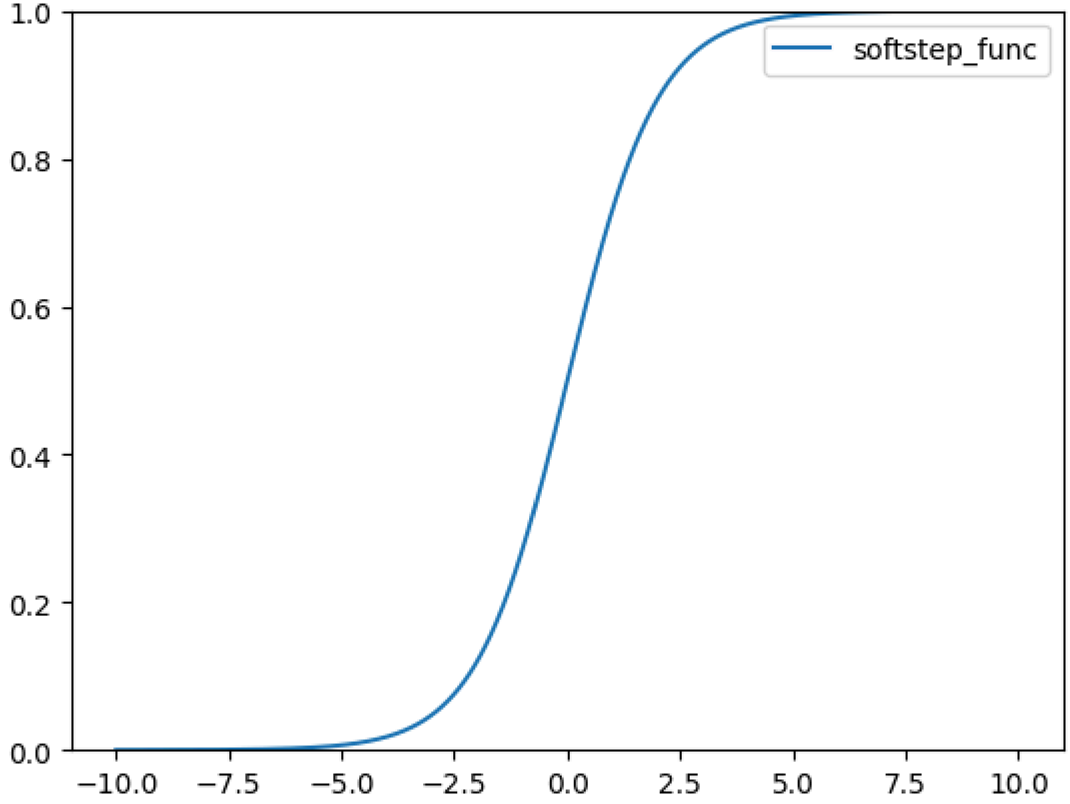




분류(Classfication의 경우

- 시그모이드(Sigmoid) 함수 - 2클래스 분류에서 사용(ex, 개 vs 고양이 분류)
- 소프트맥스(Softmax) 함수 - 다중 클래스 분류(ex, 정상 폐사진 vs 폐결절,패혈증... 등 분류)

```
#로지스틱(Logistic) 또는 시그모이드(Sigmoid)라고 불리는 함수 정의 - http://www.gisdeveloper.co.kr/?p=8285
import numpy as np
import matplotlib.pylab as plt
def softstep_func(x):
    return 1 / (1 + np.exp(-x))
x = np.arange(-10, 10, 0.01)
plt.plot(x, softstep_func(x), linestyle='-', label="softstep_func")
plt.ylim(0, 1)
plt.legend()
plt.show()
```



소프트 맥스(Softmax) 함수란?

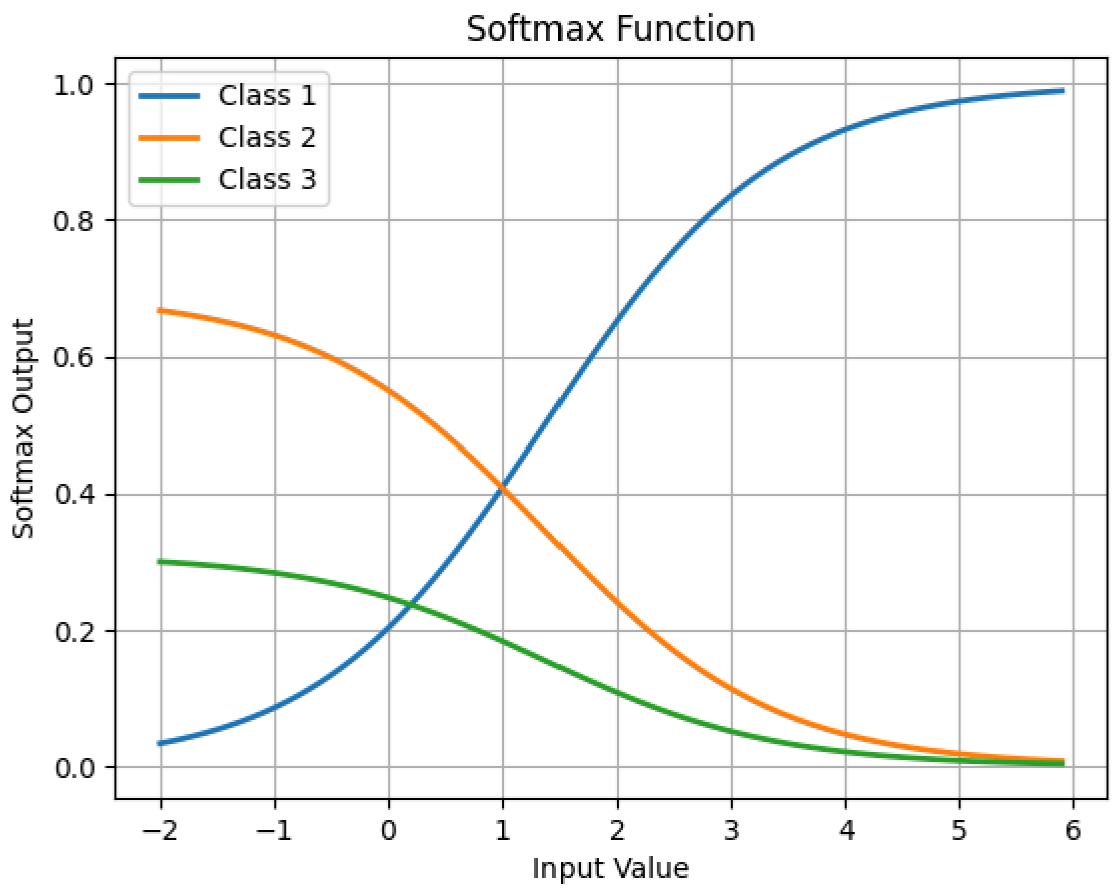
- 0과 1사이의 숫자를 출력하는 함수로, 출력하는 값은 확률

```
def softmax(a):
    C = np.max(a)
    exp_a = np.exp(a-C)
    sum_a = np.sum(exp_a)

    return exp_a / sum_a
```

```
import numpy as np

def softmax(x):
  """
  Computes the softmax function for a given input array.

  Args:
    x: A NumPy array of numbers.

  Returns:
    A NumPy array with softmax values, where each value represents a probability.
  """
  e_x = np.exp(x - np.max(x)) # Subtract max for numerical stability
  return e_x / e_x.sum(axis=0)

# Example usage
logits = np.array([2.0, 1.0, 0.1])
probabilities = softmax(logits)
print(probabilities) # Output: [0.65900308 0.24242973 0.09856719]
print(np.sum(probabilities)) # Output: 1.0
```

```
[0.65900114 0.24243297 0.09856589]
1.0
```

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

def softmax(x):
  """
  Compute softmax values for each set of scores in x.
  """
  e_x = np.exp(x - np.max(x)) # Subtract max for numerical stability
  return e_x / e_x.sum(axis=0)

# Example usage:
x = np.arange(-2, 6, 0.1)
scores = np.vstack([x, np.ones_like(x), 0.2 * np.ones_like(x)])

plt.plot(x, softmax(scores).T, linewidth=2)
plt.xlabel("Input Value")
plt.ylabel("Softmax Output")
plt.title("Softmax Function")
plt.legend(["Class 1", "Class 2", "Class 3"])
plt.grid(True)
plt.show()
```



```
# The below code implements the softmax function
# using python and numpy. It takes:
# Input: It takes input array/list of values
# Output: Outputs a array/list of softmax values.


# Importing the required libraries
import numpy as np

```

```
# Defining the softmax function
def softmax(values):

    # Computing element wise exponential value
    exp_values = np.exp(values)

    # Computing sum of these values
    exp_values_sum = np.sum(exp_values)

    # Returing the softmax output.
    return exp_values/exp_values_sum


if __name__ == '__main__':

    # Input to be fed
    values = [2, 4, 5, 3]

    # Output achieved
    output = softmax(values)
    print("Softmax Output: ", output)
    print("Sum of Softmax Values: ", np.sum(output))
```

```
Softmax Output:  [0.0320586  0.23688282 0.64391426 0.08714432]
Sum of Softmax Values:  1.0
```

Why is Binary Cross-Entropy Important?

- 딥러닝 모델 학습

> - 이진 교차 엔트로피는 이진 분류 작업에서 신경망 학습의 손실 함수로 사용됨
> - 예측 오류를 최소화하기 위해 모델의 가중치를 조정하는 데 도움이 됨

- 확률적 해석

> - 모델의 예측에 대한 확률적 해석을 제공하므로 의료 진단이나 사기 탐지와 같이 예측의 신뢰도를 이해하는 것이 중요한 응용 분야에 적합함

- 모델 평가

> - 이진 분류 모델의 성능을 평가하는 명확하고 해석 가능한 지표임
> - BCE 값이 낮을수록 모델 성능이 우수함을 나타냄

- 불균형 데이터 처리

> - 특히 한 클래스의 빈도가 다른 클래스보다 훨씬 높은 불균형 데이터 세트가 있는 상황에서 유용할 수 있음
> - 확률 예측에 중점을 두므로 클래스 불균형이 있는 경우에도 모델이 정확한 예측을 수행하도록 학습하는 데 도움이 됨

```
import numpy as np
from keras.losses import binary_crossentropy

# Example true labels and predicted probabilities
y_true = np.array([0, 1, 1, 0, 1])
y_pred = np.array([0.1, 0.9, 0.8, 0.2, 0.7])

# Compute Binary Cross-Entropy using NumPy
def binary_cross_entropy(y_true, y_pred):
    bce = -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    return bce

bce_loss = binary_cross_entropy(y_true, y_pred)
print(f"Binary Cross-Entropy Loss (manual calculation): {bce_loss}")

# Compute Binary Cross-Entropy using Keras
```

```
bce_loss_keras = binary_crossentropy(y_true, y_pred).numpy()
print(f"Binary Cross-Entropy Loss (Keras): {bce_loss_keras}")
```

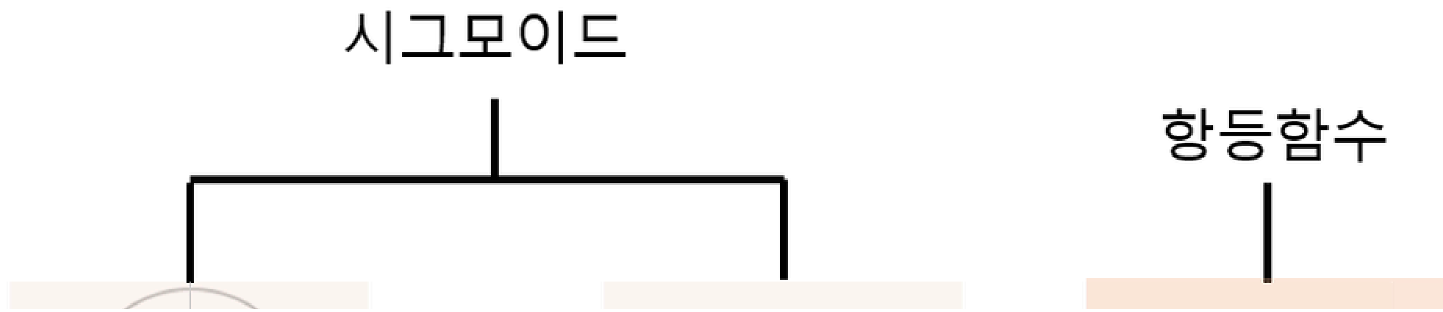
```
Binary Cross-Entropy Loss (manual calculation): 0.20273661557656092
Binary Cross-Entropy Loss (Keras): 0.20273661557656092
```

```
# The below code implements the cross entropy
# loss between the predicted values and the
# true values of cass labels. The function:
# Inputs: Predicted values, True values
# Output: The cross entropy loss between them.


# Importing the required library
import torch.nn as nn
import torch

# Cross Entropy function.
def cross_entropy(y_pred, y_true):

    # computing softmax values for predicted values
    y_pred = softmax(y_pred)
    loss = 0

    # Doing cross entropy Loss
    for i in range(len(y_pred)):

        # Here, the loss is computed using the
        # above mathematical formulation.
        loss = loss + (-1 * y_true[i]*np.log(y_pred[i]))

    return loss

# y_true: True Probability Distribution
```

```
y_true = [1, 0, 0, 0, 0]

# y_pred: Predicted values for each calss
y_pred = [10, 5, 3, 1, 4]

# Calling the cross_entropy function by passing
# the suitable values
cross_entropy_loss = cross_entropy(y_pred, y_true)

print("Cross Entropy Loss: ", cross_entropy_loss)
```

```
Cross Entropy Loss:  0.010199795719758164
```

아래의 신경망을 직접 구현


시그모이드
항등함수