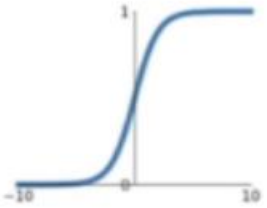
활성화함수란?

딥러닝 네트워크에서 노드에 입력된 값들을 어떠한 계산식으로 결과를 내서(활성화함수) 그 값을 다음 레이어로 전달하는데, 이 때 사용하는 함수를 활성화 함수(Activation Function)라고 한다.

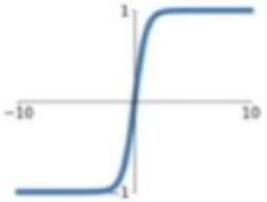
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



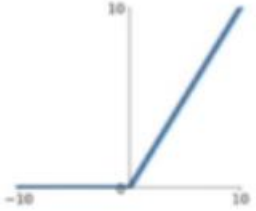
tanh

$$\tanh(x)$$



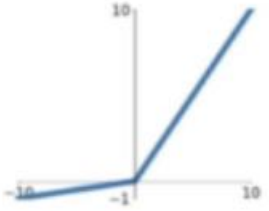
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

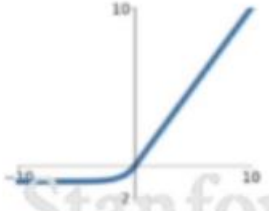


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



sigmoid 활성화 함수식

y값이 0또는 1 두수중 1개에 해당하는 값이 출력될수 있게 하려면 y값의 출력범위를 0~1사이의 숫자한개가 나올수 있도록 계산되는 수식이 필요함(**이항분류**). 이 수식을 제공하는 함수가 sigmoid임

▶출력y예) 0.7 / 0.234 / 0.8 => 프로그래머가 임계값(예를들면 0.5)을 기준으로 그보다 크면 1, 작으면 0으로 코딩해야함

```
#시그모이드함수식을 제공해주는 모듈
from scipy.special import expit
xList = [0.25,7,9,20]
for x값 in xList:
  print(x값,'input x값을 sigmoid로 계산한결과-->',expit(x값))
```

```
0.25 input x값을 sigmoid로 계산한결과--> 0.5621765008857981
7 input x값을 sigmoid로 계산한결과--> 0.9990889488055994
9 input x값을 sigmoid로 계산한결과--> 0.9998766054240137
20 input x값을 sigmoid로 계산한결과--> 0.9999999979388463
```

직접 계산식 작성

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

```
#자연상수e
e=2.718281828459045
xList = [0.25,7,9,20]
for x값 in xList:
  print(1/ (1 + e** - x값))
```

```
0.5621765008857981
0.9990889488055994
0.9998766054240137
0.9999999979388463
```

자연로그,자연상수 이해하기
https://ourcalc.com/natural-log-calculator/ -

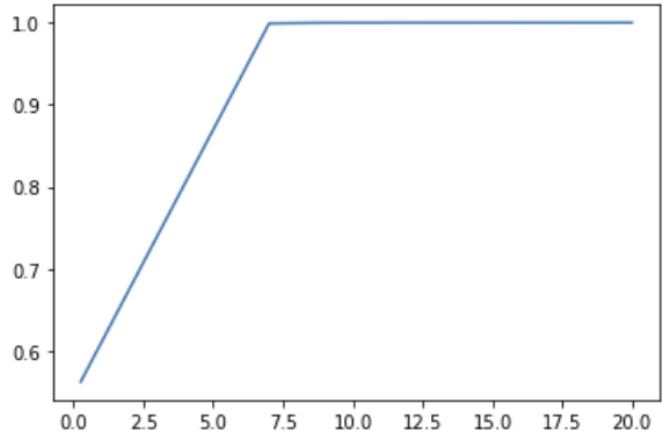
sigmoid 활성화 함수식

- **넘파이함수의 np.exp를 이용하여서 자연상수와 자연로그를 계산할수도 있음.**

```
import numpy as np
xList = [0.25,7,9,20]
yList=[]
for x값 in xList:
  yList.append(1/ (1+np.exp(-x값)))
print(yList)

import matplotlib.pyplot as plt
plt.plot(xList,yList)
```

```
[0.5621765008857981, 0.9990889488055994, 0.9998766054240137, 0.9999999979388463]
[<matplotlib.lines.Line2D at 0x7fc69d49d550>]
```



텐서플로우의 sigmoid함수에 대하여 확인

샘플

```
import pandas as  pd
roomCnt=[1,2,1,1,2,1,3,4]  # 방갯수
year=[2001,1997,1998,2020,1994,1999,2000,2010] #건축년도
재건축여부=[1,0,0,1,1,1,1,0]

df=pd.DataFrame({'방갯수':roomCnt,'건축년도':year,'재건축여부':재건축여부})

df['건축년도']=df['건축년도']/2000       #스케일링,표준화,정규화
df['건축년도']=df['건축년도'].astype('int32')     # 정수값으로 변환

x=df[['방갯수','건축년도']].values
y=df['재건축여부'].values
df
```

x데이터		y값
방갯수	건축년도	재건축여부
1	1	1
2	0	0
1	0	0
1	1	1
2	0	1
1	0	1
3	1	1
4	1	0

x변수가 2개임으로
w값은 2개필요,
b는 한개필요

선형회귀식이라면
방갯수*w1+건축년도*w2+b임

선형,시그모이드,소프트맥스의 계산공식은 달라도 필요한 w와b의 갯수는 같음

유닛1개에 필요한 파라미터갯수는 3 개임

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
tf.random.set_seed(1234)  # w값 바꾸지 않기 위해 사용함
input_layer = tf.keras.layers.InputLayer(input_shape=(2,))
output_layer= tf.keras.layers.Dense(units=1,activation='sigmoid')

model = keras.Sequential([input_layer,output_layer])
model.compile(loss='binary_crossentropy', metrics=['accuracy'])
print(model.fit(x, y))
model.summary()  # 구축된 레이처층을 보여줌
```

```
1/1 [==============================] - 1s 799ms/step - loss: 0.7385 - accuracy: 0.2500
<keras.callbacks.History object at 0x7fc622169210>
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_17 (Dense)            (None, 1)                 3

=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
```

텐서플로우의 sigmoid함수에 대하여 확인

- **layer(0)번 input 레이어에 입력된 x값에 w,b가 계산된(시그모이드로) output 값을 확인하기 위한 작업**

```
tmp=tf.keras.Model(inputs=model.input,
                   outputs=model.layers[0].output)(x)
tmp
```

```
<tf.Tensor: shape=(8, 1), dtype=float32, numpy=
array([[0.43392918],
       [0.55593234],
       [0.5284482 ],
       [0.43392918],
       [0.55593234],
       [0.5284482 ],
       [0.48891923],
       [0.5165997 ]], dtype=float32)>
```

- **어떠한 w 와 b가 생성되었는지를 확인해서 직접 계산해 보려함.**
  - 아래결과로는 x1(방갯수)의 w1은 0.11075377이고 x2(건축년도)의 w2는 -0.37975395이며 b(바이어스)는 0.00316227임

```
model.get_weights()
```

```
[array([[ 0.11075377],
        [-0.37975395]], dtype=float32), array([0.00316227], dtype=float32)]
```

결과동일함

- **w와 b값을 할당받고 시그모이드함수를 계산해봄**

```
#model.get_config()
w=model.get_weights()[0]
b=model.get_weights()[1]

선형회귀식=np.dot(x,w)+b
1/(1+np.exp(-선형회귀식))
```

w	0.11075377	-0.37975395		e값(np.exe값)
b	0.00316227			2.718281828
x1(방갯수)	x2(건축년도)	선형회귀값 =x1*w1+x2+w2+b	시그모이드함수식 1/ (1+ np.exp(-선형회귀값))	
1	1	-0.26583791	0.433929166	
2	0	0.22466981	0.555932378	
1	0	0.11391604	0.528448253	
1	1	-0.26583791	0.433929166	
2	0	0.22466981	0.555932378	
1	0	0.11391604	0.528448253	
3	1	-0.04433037	0.488919222	
4	1	0.0664234	0.516599747	

참고: 히든레이어와가 있을때 파라미터 개수

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
tf.random.set_seed(1234)  # w값 바꾸지 않기 위해 사용함
input_layer = tf.keras.layers.InputLayer(input_shape=(2,))
hi=tf.keras.layers.Dense(units=1,activation='relu')
output_layer= tf.keras.layers.Dense(units=1,activation='sigmoid')

model = keras.Sequential([input_layer,hi,output_layer])
model.compile(loss='binary_crossentropy', metrics=['accuracy'])
print(model.fit(x, y))
model.summary()  # 구축된 레이처층을 보여줌
```

input값

x변수2개, w1,w2,b
총 3개의 파라미터 필요

히든레이어
유닛1로 설정됨

input값을 거쳐갈 unit1개 준비됨
3개파라미터 * 1개유닛 =3개파라미터
여기서 유닛갯수는 다음레이어로 넘어가는 x값의 갯수와 같음.

output
유닛1로 설정됨

히든레이어유닛이 1개
즉 x값으로 생각하는 변수가 1개임으로
w1개, b1개 해서 총 2개의 파라미터필요하며 이러한 유닛이 1개임으로
총 **2**개의 파라미터 임

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 1)                 3

dense_23 (Dense)             (None, 1)                 2

=================================================================
```

참고: 히든레이어와 'relu' 활성화 함수에서 출력값

hi=tf.keras.layers.Dense(units=1,activation='relu')

```
tmp=tf.keras.Model(inputs=model.input,
                   outputs=model.layers[0].output)(x)
tmp
```

```
<tf.Tensor: shape=(8, 1), dtype=float32, numpy=
array([[0.        ],
       [0.22466981],
       [0.11391602],
       [0.        ],
       [0.22466981],
       [0.11391602],
       [0.        ],
       [0.06642342]], dtype=float32)>
```

w	0.11075378	-0.37975395
b	0.00316224	

x1(방갯수)	x2(건축년도)	선형회귀값 =x1*w1+x2+w2+b	relu (0미만은0, 0이상은 선형회귀값)
1	1	-0.26583793	0
2	0	0.2246698	0.2246698
1	0	0.11391602	0.11391602
1	1	-0.26583793	0
2	0	0.2246698	0.2246698
1	0	0.11391602	0.11391602
3	1	-0.04433037	0
4	1	0.06642341	0.06642341

```
model.get_weights()
```

```
[array([[ 0.11075378],
        [-0.37975395]], dtype=float32),
 array([0.00316224], dtype=float32),
 array([[-0.60811526]], dtype=float32),
 array([0.00316227], dtype=float32)]
```

input->히든

히든->output

참고: 히든레이어와 'relu' 활성화 함수에서 출력값

output_layer= tf.keras.layers.Dense(units=1,activation='sigmoid')

w	0.11075378	-0.37975395
b	0.00316224	

x1(방갯수)	x2(건축년도)	선형회귀값 =x1*w1+x2+w2+b	relu (0미만은0, 0이상은 선형회귀값)
1	1	-0.26583793	0
2	0	0.2246698	0.2246698
1	0	0.11391602	0.11391602
1	1	-0.26583793	0
2	0	0.2246698	0.2246698
1	0	0.11391602	0.11391602
3	1	-0.04433037	0
4	1	0.06642341	0.06642341

w	-0.60811526	e값(np.exe값)
b	0.00316227	2.718281828

히든relu출력값을 x로 계산함. x*w+b한 선형회귀값	시그모이드함수식 1/ (1+ np.exp(-선형회귀값))
0.0031622	0.500790567
-0.133462864	0.466683723
-0.0661118	0.483478067
0.0031622	0.500790567
-0.133462864	0.466683723
-0.0661118	0.483478067
0.0031622	0.500790567
-0.037230819	0.49069337

model.get_weights()

```
[array([[ 0.11075378],
        [-0.37975395]], dtype=float32),
 array([0.00316224], dtype=float32),
 array([[-0.60811526]], dtype=float32),
 array([0.00316227], dtype=float32)]
```

input->히든

히든->output

```
tmp=tf.keras.Model(inputs=model.input,
                   outputs=model.layers[1].output)(x)
tmp
```

```
<tf.Tensor: shape=(8, 1), dtype=float32, numpy=
array([[0.5007906 ],
       [0.46668372],
       [0.48347807],
       [0.5007906 ],
       [0.46668372],
       [0.48347807],
       [0.5007906 ],
       [0.49069336]], dtype=float32)>
```

마지막레이어임으로 아래와 같이 해도 결과나옴
model.predict(x) #x값에 대한 최종 output값

텐서플로우의 sigmoid함수의 오차함수

```
model.compile(loss='binary_crossentropy', metrics=['accuracy'])
```

```
# 최종 output값계산
print(model.predict(x))
```

```
[[0.5007906 ]
 [0.46668372]
 [0.48347807]
 [0.5007906 ]
 [0.46668372]
 [0.48347807]
 [0.5007906 ]
 [0.49069336]]
```

```
# binary_crossentropy 값 출력
print('-'*300)
model.evaluate(x, y)
```

```
------------------------------
1/1 [==============================
[0.6909422874450684, 0.75]
```

```
### binary-crossentropy(직접계산한값)
yhat=model.predict(x)
y=y.reshape(8,1)
err=y * np.log(yhat) + (1 - y) * np.log(1 - yhat)
-np.mean(err)
```

```
0.6909422129392624
```