

softmax 활성화함수 계산하기 전 y값 전처리

소프트맥스 함수는 로지스틱 함수의 다차원 일반화이다. 다항 로지스틱 회귀에서 쓰이고, 인공신경망에서 확률분포를 얻기 위한 마지막 활성화함수로 많이 사용된다.

확률분포의 예) 신발, 가방, 티셔츠, 핸드폰 사진을 넣으면 4개의 분류중 1개를 선택해야함.

- 이때 신발, 가방, 티셔츠, 핸드폰은 y값이 되는데 이러한 한글은 계산이 불가능함으로 반드시 숫자로 변경해야함.

사용자가 숫자번호를 부여함

```
1 yList=['가방','가방','신발','핸드폰','티셔츠','핸드폰']
2 ydata=[]
3 for y in yList:
4     if y=='신발':
5         ydata.append(0)
6     elif y=='가방':
7         ydata.append(1)
8     elif y=='티셔츠':
9         ydata.append(2)
10    elif y=='핸드폰':
11        ydata.append(3)
12 ydata
```

[1, 1, 0, 3, 2, 3]

가나다라순서에 의해서 숫자번호가부여됨

```
# 문자열을 숫자로 변환해주는 모듈
from sklearn.preprocessing import LabelEncoder
```

```
yList=['가방','가방','신발','핸드폰','티셔츠','핸드폰']
e = LabelEncoder()
e.fit(yList)
ydata=e.transform(yList)
ydata
```

array([0, 0, 1, 3, 2, 3])

softmax 활성화함수 계산하기 전 y값 전처리

소프트맥스 함수는 로지스틱 함수의 다차원 일반화이다. 다항 로지스틱 회귀에서 쓰이고, 인공신경망에서 확률분포를 얻기 위한 마지막 활성화함수로 많이 사용된다.

확률분포의 예) 신발, 가방, 티셔츠, 핸드폰 사진을 넣으면 4개의 분류중 1개를 선택해야함.

- 이때 신발, 가방, 티셔츠, 핸드폰은 y값이 되는데 이러한 한글은 계산이 불가능함으로 반드시 숫자로 변경해야함.
- y값은 중복제거한 갯수만큼 확률분포를 작성하게 됨
 - 아래 예제는 총 4개의 경우의 수가 있음을 의미함. 원핫인코딩은 4개의 비트로 준비해야함.

```
yList=['가방','가방','신발','핸드폰','티셔츠','핸드폰']  
array([0, 0, 1, 3, 2, 3])
```

```
np.unique(ydata), len(np.unique(ydata))      (array([0, 1, 2, 3]), 4)
```

```
import tensorflow as tf  
y_encoded = tf.keras.utils.to_categorical(ydata)  
y_encoded
```

```
array([[1., 0., 0., 0.],  
       [1., 0., 0., 0.],  
       [0., 1., 0., 0.],  
       [0., 0., 0., 1.],  
       [0., 0., 1., 0.],  
       [0., 0., 0., 1.]], dtype=float32)
```

가방은 0=> [1,0,0,0]

티셔츠는 2=> [0,1,0,0]

softmax 활성화함수 계산하기 전 y값 전처리

■ 1번 인공지능 개념잡기.pptx자료 내용임

모 델	y값개수	y값에따라활성화함수	y값에따라오차함수	비고
예측 (선형회귀)	1	생략 Dense(1)	MSE,MAE,RMSE	
분류 (이항분류)	1	activation='sigmoid'	binary_crossentropy	
분류 (이항분류)	2	activation='softmax'	categorical_crossentropy	y값원핫인코딩
			sparse_categorical_crossentropy	y값이숫자
분류 (다항분류)	2이상	activation='softmax'	categorical_crossentropy	y값원핫인코딩
			sparse_categorical_crossentropy	y값이숫자

```
output_layer= tf.keras.layers.Dense(units=4,activation='softmax')
```

```
model = keras.Sequential([input_layer,output_layer])
model.compile(loss='categorical_crossentropy',
              metrics=['accuracy'])
```

softmax활성화함수 계산식

$$y_k = \frac{e^{a_k}}{\sum_{i=1}^n e^{a_i}}$$

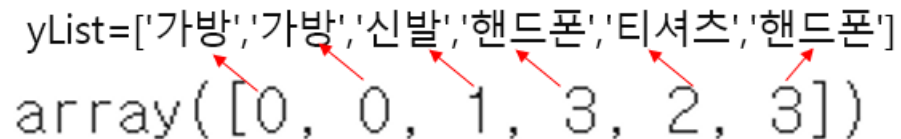
- n = 출력층의 뉴런 수(총 클래스의 수), k = k 번째 클래스
- 만약, 총 클래스의 수가 3개라고 한다면 다음과 같은 결과가 나오게 된다.

$$\text{softmax}(z) = \left[\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}} \right] = [p_1, p_2, p_3]$$

사진을 x값으로 입력하면 softmax함수는 원핫인코딩이 4개로 되어 있을때는
예측한 결과가 [0.1, 0.8, 0.03, 0.07] 과 같이 4개의 비트에 다 더해서 1이 되는 숫자값이 나옴.
해석하면 0번위치에10%, 1번위치에 80%, 2번위치에3%, 3번위치에7%의 확률숫자가 나온다는뜻.
가장 큰 확률을 출력한 1번위치값은 [0,1,0,0] 과 같이 해석할수 있음

원핫인코딩할때 1값은 '신발' 레이블임으로 출력에 1이 출력 또는 '신발 ' 이 출력되게 사용자가 프로그램 해야함.

```
yList=['가방','가방','신발','핸드폰','티셔츠','핸드폰']  
array([0, 0, 1, 3, 2, 3])
```



softmax활성화함수 계산식

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
재건축여부등급=['A','B','A','A','C','A','A','B']

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

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

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

x데이터		y값
방갯수	건축년도	재건축여부등급
1	1	A
2	0	B
1	0	A
1	1	A
2	0	C
1	0	A
3	1	A
4	1	B

y값을 원핫인코딩해야함. 안하면 loss에서 spare categorical_crossentropy를 해야함
#tf.one_hot(y, 3,on_value=1.0, off_value=0.0)

```
from sklearn.preprocessing import LabelEncoder
e = LabelEncoder()
e.fit(y)
ydata=e.transform(y)
ydata

import tensorflow as tf
y_encoded = tf.keras.utils.to_categorical(ydata)
y_encoded
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.]], dtype=float32)
```

softmax활성화함수 계산식

```
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=2,activation='relu')
output_layer= tf.keras.layers.Dense(units=3,activation='softmax') # 3개중에 한개임

model = keras.Sequential([
    input_layer,
    hi,
    output_layer
])
model.compile(loss='categorical_crossentropy',
              metrics=['accuracy'])

print(model.fit(x, y_encoded))
model.summary()
```

1/1 [=====] - 1s 709ms/step - loss: 1.1592 - accuracy: 0.1250
<keras.callbacks.History object at 0x7f5d2d8d7c10>
Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 2)	6
dense_6 (Dense)	(None, 3)	9
Total params: 15		
Trainable params: 15		
Non-trainable params: 0		

- 0번레이어는 히든레이어이며 relu사용하면 선형,시그모이드,소프트맥스 모두 계산공식이 같음

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

```
<tf.Tensor: shape=(8, 2), dtype=float32, numpy=
array([[0.2836423, 0.17686711],
       [0.08685242, 0.2836423],
       [0.17686711, 0.08685242],
       [0.2836423, 0.17686711],
       [0.08685242, 0.46367165],
       [0.46367165, 0.5536864],
       [0.5536864, 0.2836423],
       [0.17686711, 0.08685242]], dtype=float32)>
```

unit1

unit2

1 model.get_weights()

```
[array([[ 0.09001469, -0.3316152 ],
       [ 0.19678986, -0.6274897 ]], dtype=float32),
 array([-0.00316227, 0.          ], dtype=float32),
 array([[ -0.37944448, -0.7884051,  0.32119763],
       [ -0.42937422, -0.35763377, -0.810234  ]], dtype=float32),
 array([ 0.00316227, -0.00316225, -0.00316227], dtype=float32)]
```

1

2

1

유닛1자료임

w
b

0.09001469	0.19678986
-0.00316227	

x1(방갯수)	x2(건축년도)	선형회귀값 =x1*w1+x2+w2+b	relu
1	1	0.28364228	0.28364228
2	0	0.17686711	0.17686711
1	0	0.08685242	0.08685242
1	1	0.28364228	0.28364228
2	0	0.17686711	0.17686711
1	0	0.08685242	0.08685242
3	1	0.46367166	0.46367166
4	1	0.55368635	0.55368635

유닛2자료임

w
b

-0.3316152	-0.3316152
0	

x1(방갯수)	x2(건축년도)	선형회귀값 =x1*w1+x2+w2+b	relu
1	1	-0.66639267	0
2	0	-0.66639267	0
1	0	-0.33477747	0
1	1	-0.66639267	0
2	0	-0.66639267	0
1	0	-0.33477747	0
3	1	-1.32962307	0
4	1	-1.66123827	0

- 1번레이어는 최종출력물 레이어여서 예측함수를 사용해도 결과는 같음.

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

```
1 # y예측값
2 model.predict(k)
```

```
array([[0.32289037, 0.28571355, 0.39139608],
       [0.32796055, 0.30315286, 0.36888662],
       [0.33166817, 0.31807628, 0.3502555 ],
       [0.32289037, 0.28571355, 0.39139608],
       [0.32796055, 0.30315286, 0.36888662],
       [0.33166817, 0.31807628, 0.3502555 ],
       [0.31277582, 0.2571189 , 0.43010527],
       [0.30703232, 0.24327502, 0.4496927 ]], dtype=float32)
```

```
1 model.get_weights()
```

```
[array([[ 0.09001469, -0.3316152 ],
       [ 0.19678986, -0.6274897 ]], dtype=float32),
 array([[-0.00316227, 0.          ], dtype=float32),
 array([[ -0.37944448, -0.7884051 ,  0.32119763],
       [ -0.42937422, -0.35763377, -0.810234  ]], dtype=float32),
 array([ 0.00316227, -0.00316225, -0.00316227], dtype=float32)]
```

unit1

unit2

unit2

유닛1

2

유닛2

유닛3

결과값

w	-0.3794445	-0.42937422	e값(np.exp값)
b	0.00316227		2.718281828
유닛1 relu값 (x1)	유닛2 relu값 (x2)	선형값=x1*w1+x2*w2+b	np.exp(선형값)
0.28364228	0	-0.104464227	0.900807021
0.17686711	0	-0.063948979	0.938052859
0.08685242	0	-0.029793401	0.970646047
0.28364228	0	-0.104464227	0.900807021
0.17686711	0	-0.063948979	0.938052859
0.08685242	0	-0.029793401	0.970646047
0.46367166	0	-0.172775382	0.841326571
0.55368635	0	-0.206930959	0.813075783

w	-0.7884051	-0.35763377	e값(np.exp값)
b	-0.00316225		2.718282
유닛1 relu값 (x1)	유닛2 relu값 (x2)	선형값=x1*w1+x2*w2+b	np.exp(선형값)
0.28364228	0	-0.22678727	0.79709
0.17686711	0	-0.142605182	0.867096
0.08685242	0	-0.071637141	0.930869
0.28364228	0	-0.22678727	0.79709
0.17686711	0	-0.142605182	0.867096
0.08685242	0	-0.071637141	0.930869
0.46367166	0	-0.368723351	0.691617
0.55368635	0	-0.439691392	0.644235

w	0.32119763	-0.810234	e값(np.exp값)
b	-0.00316227		2.718281828
유닛1 relu값 (x1)	유닛2 relu값 (x2)	선형값=x1*w1+x2*w2+b	np.exp(선형값)
0.28364228	0	0.087942958	1.091925835
0.17686711	0	0.053647027	1.05511211
0.08685242	0	0.024734521	1.025042957
0.28364228	0	0.087942958	1.091925835
0.17686711	0	0.053647027	1.05511211
0.08685242	0	0.024734521	1.025042957
0.46367166	0	0.145767968	1.156927713
0.55368635	0	0.174680473	1.190865643

분모 (유닛1+유닛2+유닛3)	유닛1/분모	유닛2/분모	유닛3/분모
2.789823185	0.322890363	0.28571357	0.391396071
2.860261316	0.327960545	0.30315284	0.368886613
2.926557613	0.331668183	0.3180763	0.35025552
2.789823185	0.322890363	0.28571357	0.391396071
2.860261316	0.327960545	0.30315284	0.368886613
2.926557613	0.331668183	0.3180763	0.35025552
2.689871003	0.312775806	0.25711892	0.430105277
2.648176632	0.307032308	0.24327501	0.449692678

- 오차함수(손실함수)는 유튜브를 보고 참조하여 보세요.

```
1 model.evaluate(x,y_encoded)
```

```
1/1 [=====] -  
[1.1543426513671875, 0.125]
```

<https://towardsdatascience.com/cross-entropy-for-classification-d98e7f974451>