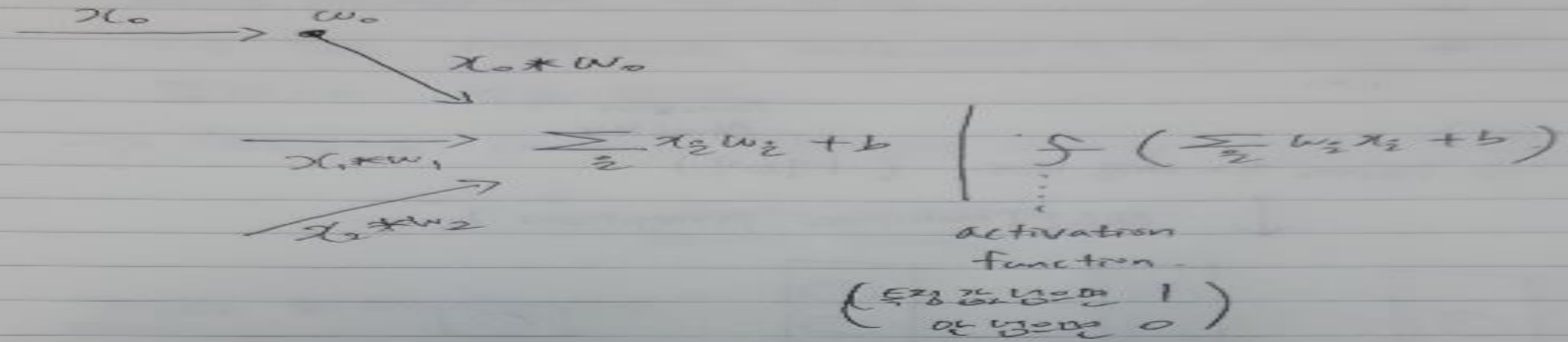


딥러닝의 기본 개념과 문제, 그리고 해결

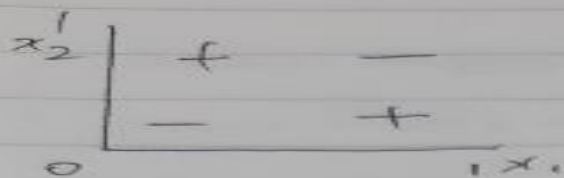
Activation Functions



<OR>



<AND>

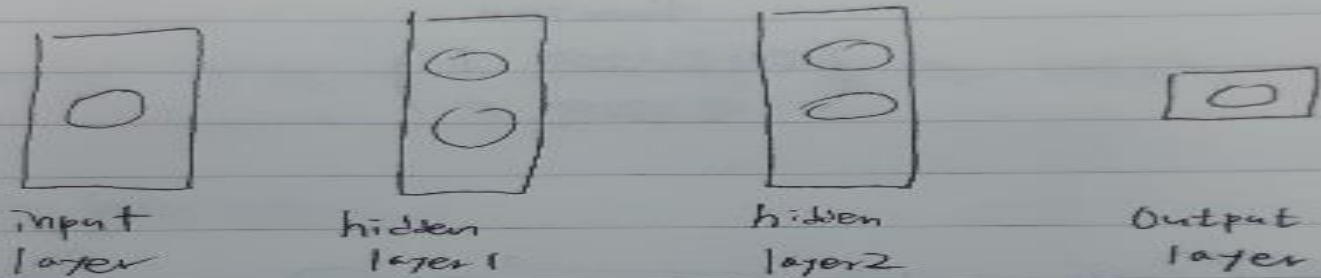


<XOR>

→ linear 하게 해결 불가능 (1969)

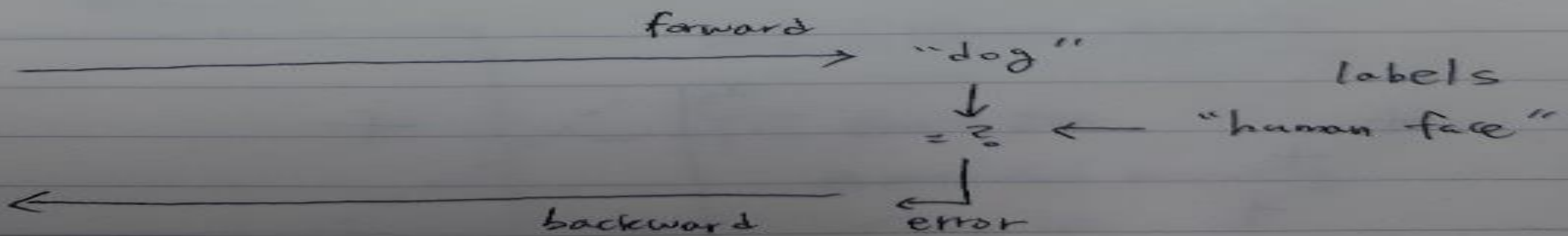
Minsky
Minsky

↓ "MLP (Multi-layer perceptrons)"



"Back propagation"

"해결" (1974, 1982 Paul Werbos)
1986 Hinton



"Back propagation" 의 한계.

→ 층 많은 layer에서 backward 할 때 error 비율이
작아져 학습 불가..

→ Machine learning 알고리즘이 더 잘됨. (SVM, Random Forest 등).

2006, Hinton.
2007, Bengio.

37/40을 잘 보면 학습 가능

2010, "Image Net", 2010.

2010~2015.

CNN...

Tensor Manipulation

```
t = np.array([0., 1., 2., 3., 4., 5., 6.])
# rank
print(t.ndim)
# shape
print(t.shape)
# 처음, 그다음, 맨뒤
print(t[0], t[1], t[-1])
# 2번째부터 5바이트 앞까지, 4번째부터 끝바이트 앞까지
print(t[2:5], t[4:-1])
# 모든것에서부터 끝까지
print(t[:2], t[3:])
```

```
1
(7,)
0.0 1.0 6.0
[2. 3. 4.] [4. 5.]
[0. 1.] [3. 4. 5. 6.]
```

```
t = np.array([[1., 2., 3.],
              [4., 5., 6.],
              [7., 8., 9.],
              [10., 11., 12.]])
print(t.ndim)
print(t.shape)
```

```
2
(4, 3)
```

```
t = tf.constant([1,2,3,4])
tf.shape(t).eval()
```

```
array([4])
```

```
t = tf.constant([[1,2],
                 [3,4]])
tf.shape(t).eval()
```

```
array([2, 2])
```

```
t = tf.constant([[[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]], [[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]]]])
tf.shape(t).eval()
```

```
array([1, 2, 3, 4])
```

```
# matmul
# 1x2
matrix1 = tf.constant([[3., 3.]])
# 2x1
matrix2 = tf.constant([[2.],[2.]])
print(matrix1.shape)
print(matrix2.shape)
# 1x1
tf.matmul(matrix1, matrix2).eval()
```

```
(1, 2)
(2, 1)
```

```
array([[12.]], dtype=float32)
```

```
# multiply - Broadcasting(shape 달라도 연산수행, 조심해서 사용)
# 1x2
matrix1 = tf.constant([[3., 3.]])
# 2x1
matrix2 = tf.constant([[2.],[2.]])
print(matrix1.shape)
print(matrix2.shape)
# 2x2
(matrix1*matrix2).eval()
```

```
(1, 2)
(2, 1)
```

```
array([[6., 6.],
       [6., 6.]], dtype=float32)
```

```
# random
tf.random_normal([3]).eval()
```

```
array([ 1.3117507,  1.5195972, -1.3779445], dtype=float32)
```

```
# random
tf.random_uniform([2]).eval()
```

```
array([0.7228352 , 0.26291537], dtype=float32)
```

```
# random
tf.random_uniform([2, 3]).eval()
```

```
array([[0.5954708 , 0.07614183, 0.878857  ],
       [0.4209664 , 0.42503965, 0.20690048]], dtype=float32)
```

```
# reduce_mean
print(tf.reduce_mean([1,2], axis=0))
x = [[1., 2.],
      [3., 4.]]
```

```
# 모든 값의 평균
tf.reduce_mean(x).eval()
```

Tensor("Mean_1:0", shape=(), dtype=int32)

2.5

```
# axis=0, 1과3 그리고 2와4의 평균 계산
tf.reduce_mean(x, axis=0).eval()
```

array([2., 3.], dtype=float32)

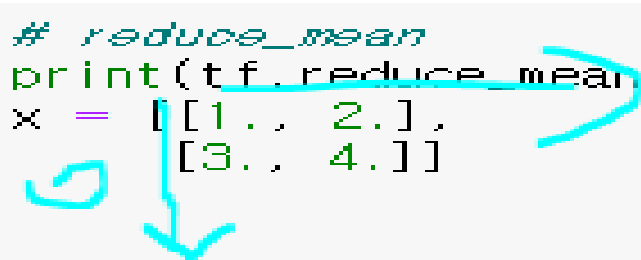
```
# axis=1, 1과2 그리고 3과4의 평균 계산
tf.reduce_mean(x, axis=1).eval()
```

array([1.5, 3.5], dtype=float32)

```
# axis=-1, 위와같은
tf.reduce_mean(x, axis=-1).eval()
```

array([1.5, 3.5], dtype=float32)

```
# reduce_mean
print(tf.reduce_mean([1,2], axis=0))
x = [[1., 2.],
      [3., 4.]]
tf.reduce_mean(x).eval()
```



```
# 1과2 그리고 3과4의 합
tf.reduce_sum(x, axis=-1).eval()
```

array([3., 7.], dtype=float32)

```
# 1과2 그리고 3과4의 합의 평균 ((3+7)/2)
tf.reduce_mean(tf.reduce_sum(x, axis=-1)).eval()
```

5.0

```
x = [[0, 1, 2],
      [2, 1, 0]]
# 최대값의 index를 반환
tf.argmax(x, axis=0).eval()
```

array([1, 0, 0], dtype=int64)

```
# argmax : 최대값의 index를 반환
tf.argmax(x, axis=-1).eval()
```

array([2, 0], dtype=int64)


```
t = np.array([[[0, 1, 2],
               [3, 4, 5]],
              [[6, 7, 8],
               [9, 10, 11]]])

t.shape

(2, 2, 3)
```

```
# rank2로 만들기
tf.reshape(t, shape=[-1, 3]).eval()

array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

```
# rank3로 만들기
tf.reshape(t, shape=[-1, 1, 3]).eval()

array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11]]])
```

```
# squeeze
tf.squeeze([ [0], [1], [2] ]).eval()

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

```
# expand
tf.expand_dims([0, 1, 2], 1).eval()

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

```
# one_hot (0 or 1 or 2, 값 종류 3개)
# rank=2에서 결과는 자동으로 rank+1을해서 rank=3이 된다
tf.one_hot([ [0], [1], [2], [0] ], depth=3).eval()
```

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

```
# shape 조정(rank=2)
t = tf.one_hot([ [0], [1], [2], [0] ], depth=3)
tf.reshape(t, shape=[-1, 3]).eval()
```

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

```
# Casting
tf.cast([1.8, 2.2, 3.3, 4.9], tf.int32).eval()

array([1, 2, 3, 4])
```

```
# Casting
tf.cast([True, False, 1 == 1, 0 == 1], tf.int32).eval()

array([1, 0, 1, 0])
```

```
# Stack
x = [1, 4]
y = [2, 5]
z = [3, 6]

# Pack along first dim.
tf.stack([x, y, z]).eval()

array([[1, 4],
       [2, 5],
       [3, 6]])
```

```
tf.stack([x, y, z], axis=1).eval()

array([[1, 2, 3],
       [4, 5, 6]])
```

```
# 이 shape과 똑같은데 1로 채워줄
x = [[0, 1, 2],
      [2, 1, 0]]

tf.ones_like(x).eval()

array([[1, 1, 1],
       [1, 1, 1]])
```

```
# 이 shape과 똑같은데 1로 채워줄
x = [[0, 1, 2],
      [2, 1, 0]]

tf.zeros_like(x).eval()

array([[0, 0, 0],
       [0, 0, 0]])
```

```
# 복수개의 tensor를 합쳐줄
for x, y in zip([1, 2, 3], [4, 5, 6]):
    print(x, y)
```

```
1 4
2 5
3 6
```

```
for x, y, z in zip([1, 2, 3], [4, 5, 6], [7, 8, 9]):
    print(x, y, z)
```

```
1 4 7
2 5 8
3 6 9
```