[Week3]_문가을

3. 파이썬과 벡터화

파이썬의 브로드캐스팅

차원이 다른 행렬끼리 계산을 할 수 있도록 자동으로 차원을 변환해 계산해줌.

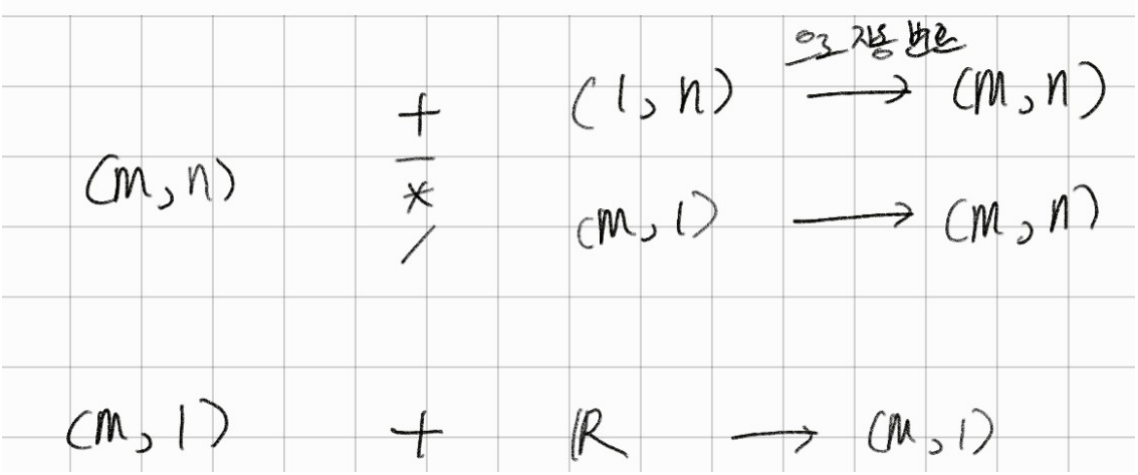
예 :

```
[1]  # 칼로리의 백분율을 브로드캐스팅을 이용해 구할 것임
     import numpy as np

     A = np.array([[56.0, 0.0, 4.4, 68.0],
                   [1.2, 104.0, 52.0, 8.0],
                   [1.8, 135.0, 99.0, 0.9]])

     cal = A.sum(axis=0) # vertical로 계산, axis=1 은 horizontal로 계산
     percentage = 100*A/cal.reshape(1,4) #(3,4)행렬을 (1,4)행렬로 나눔. ->broadcasting
     # reshape는 쓰지 않아도 됨. 하지만 행렬의 차원이 확실하지 않을때 사용하면 좋음.
     print(percentage)
```

```
[[94.91525424  0.          2.83140283 88.42652796]
 [ 2.03389831 43.51464435 33.46203346 10.40312094]
 [ 3.05084746 56.48535565 63.70656371  1.17035111]]
```



파이썬과 넘파이 벡터

브로드캐스팅은 장점과 단점을 모두 갖고 있음

장점) 간단하게 코드 작성 가능

단점) 자세한 작동 방법을 모르면 찾기 힘든 오류가 발생함.

```
[2] a = np.random.randn(5)
    print(a)
```

```
[-0.69301829  0.46122259  1.15617533 -0.42409548  0.60857955]
```

```
[3] print(a.shape) # 랭크가 1인 array
```

```
(5,)
```

```
[4] print(a.T) #transpose를 하여도 원래와 똑같이 생김.
```

```
[-0.69301829  0.46122259  1.15617533 -0.42409548  0.60857955]
```

```
[5] print(np.dot(a, a.T)) #행렬이 나와야할 것 같지만 하나의 수만 나옴.
```

```
2.5799680643344827
```

→ 신경망을 구현할 때, (n,)인 랭크가 1인 배열을 아예 사용하지 않는 것이 좋음.

```
[6] a = np.random.randn(5,1) #(5,1) 열 벡터
    print(a)
```

```
[[-1.03181837]
 [ 1.62670641]
 [-1.7682061 ]
 [ 1.20781377]
 [-1.85843636]]
```

```
[7] print(a.T) #(1,5) matrix
```

```
[[-1.03181837  1.62670641 -1.7682061   1.20781377 -1.85843636]]
```

```
[8] print(np.dot(a, a.T))
```

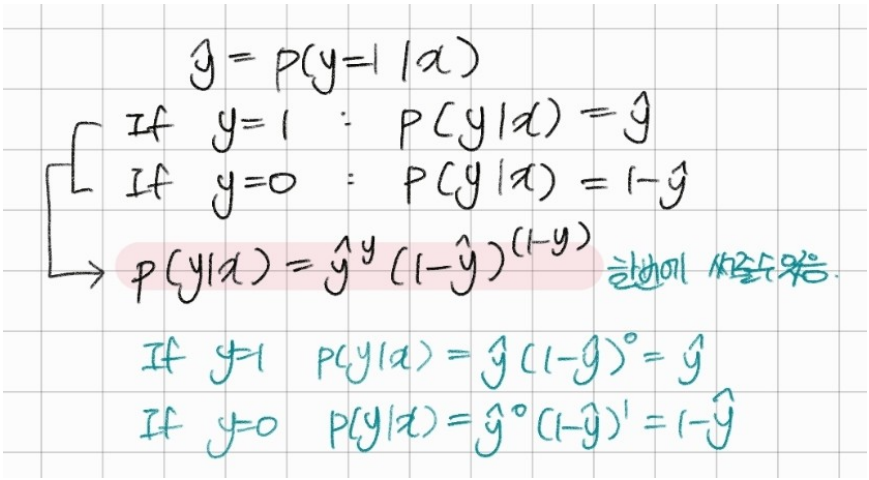
```
[[ 1.06464916 -1.67846557  1.82446754 -1.24624444  1.91756878]
 [-1.67846557  2.64617376 -2.87635221  1.96475841 -3.02313035]
 [ 1.82446754 -2.87635221  3.12655282 -2.13566368  3.28609851]
 [-1.24624444  1.96475841 -2.13566368  1.45881411 -2.24464503]
 [ 1.91756878 -3.02313035  3.28609851 -2.24464503  3.45378571]]
```

👉 <랭크 1 배열을 대체하는 방법>

1. assert(a.shape(5,1))
2. a = a.reshape((5,1))

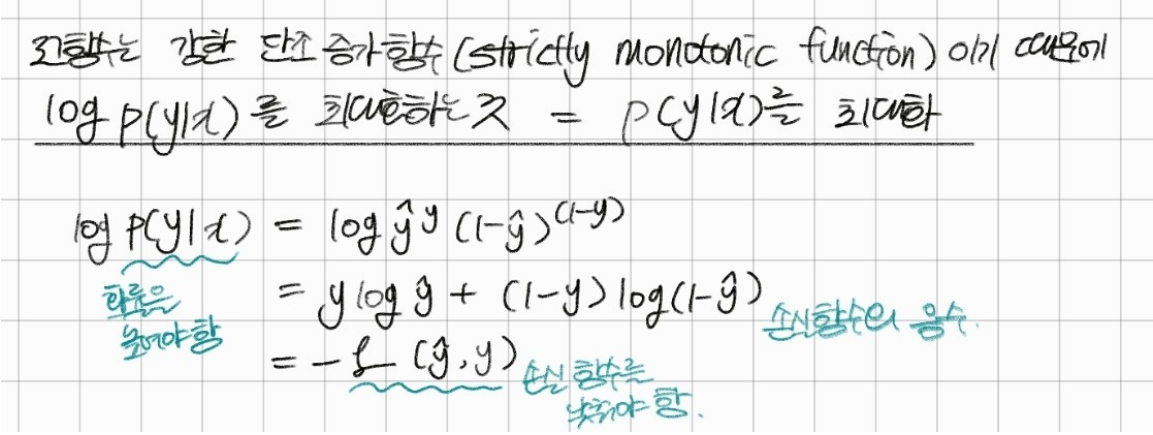
로지스틱 회귀의 비용함수 설명

로지스틱 회귀에서



손실함수 (Loss Function)

확률을 최대화 시키는 것이 목적임.

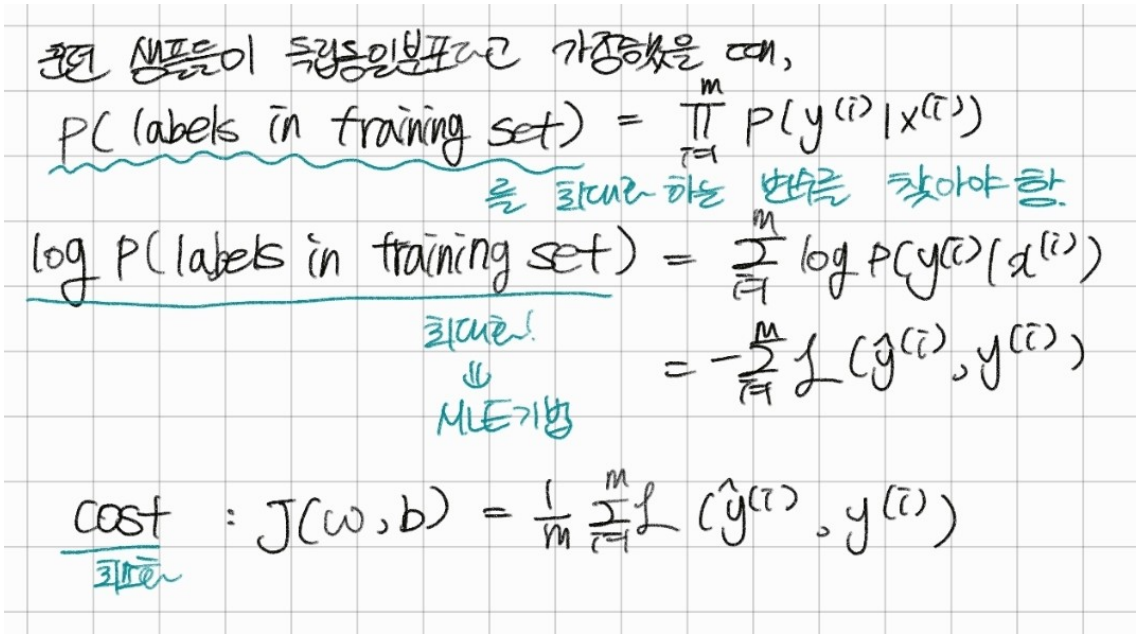


참고용 :

만약 다음 두 조건 중 하나를 만족시키면, **강한 단조 함수**(영어: strictly monotonic function)라고 한다.

- 임의의 $x, y \in I$에 대하여, $x < y$이면 $f(x) < f(y)$. 이 경우, f를 **강한 증가 함수**(영어: strictly increasing function)라고 한다.
- 임의의 $x, y \in I$에 대하여, $x < y$이면 $f(x) > f(y)$. 이 경우, f를 **강한 감소 함수**(영어: strictly decreasing function)라고 한다.

비용함수 (Cost Function)



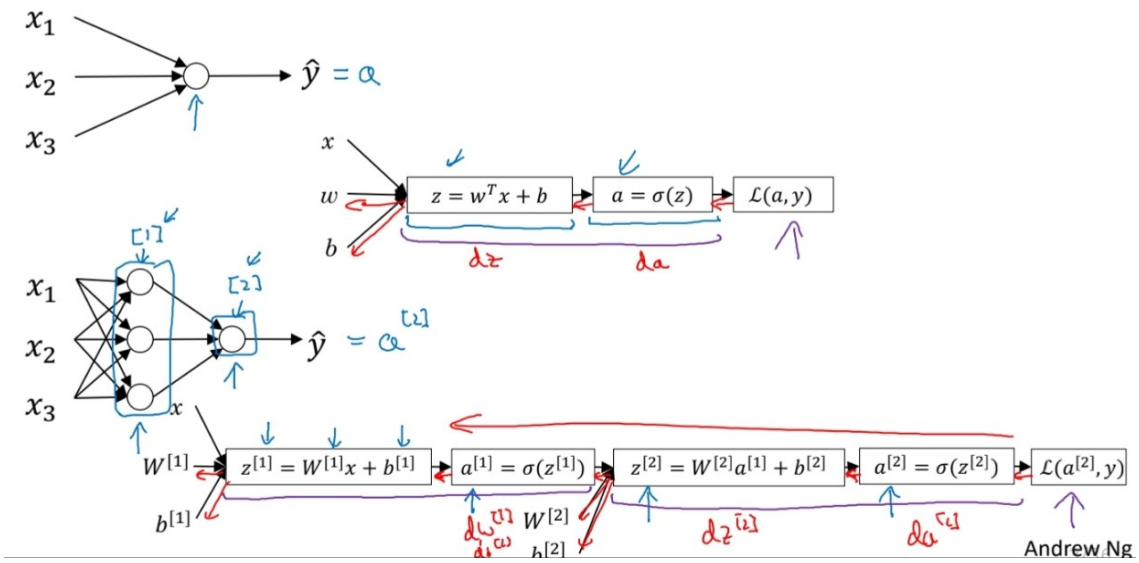
최대가능도방법 (最大可能度方法, 영어: maximum likelihood method) 또는 **최대우도법**(最大尤度法)은 어떤 확률변수에서 표집한 값들을 토대로 그 확률변수의 모수를 구하는 방법이다. 어떤 모수가 주어졌을 때, 원하는 값들이 나올 가능도를 최대로 만드는 모수를 선택하는 방법이다. 점추정 방식에 속한다.

---

4. 얕은 신경망 네트워크

신경망 네트워크 개요

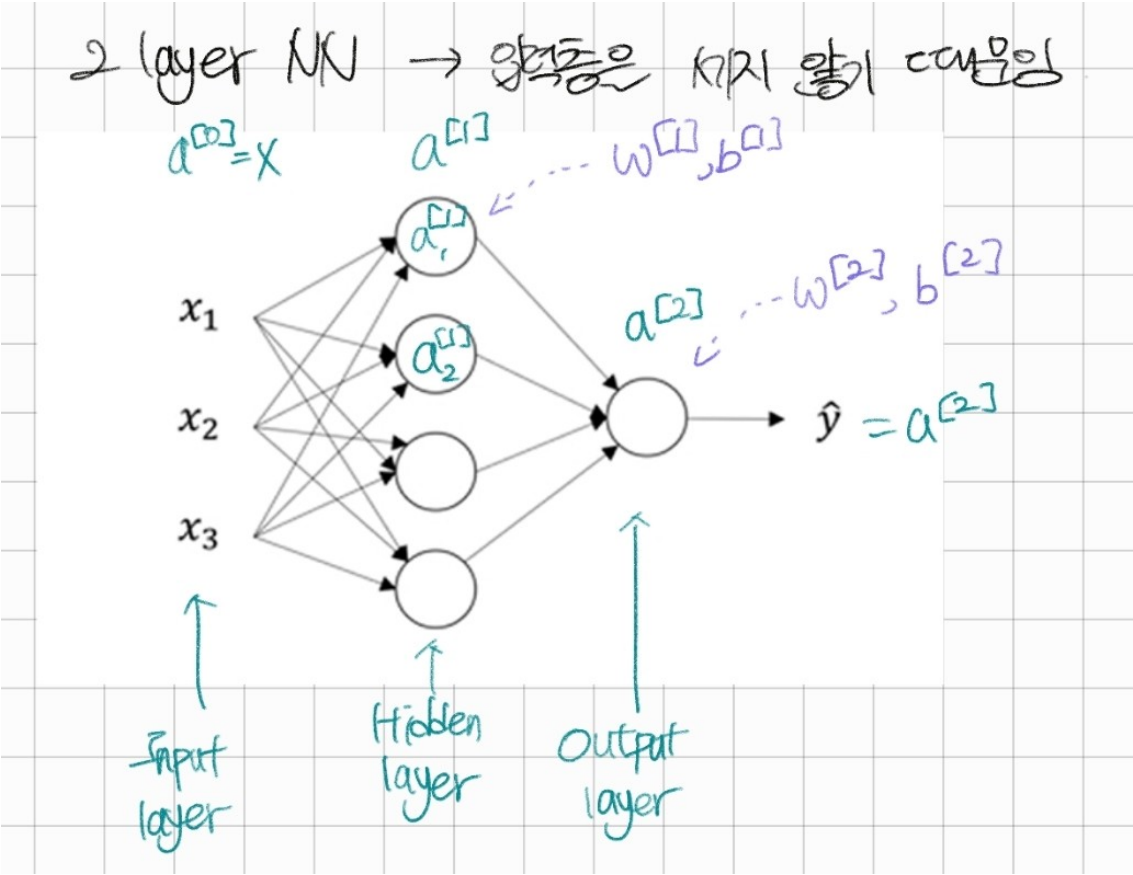
신경망은 로지스틱 회귀를 두 번 반복해주는 것임.



신경망 네트워크의 구성 알아보기

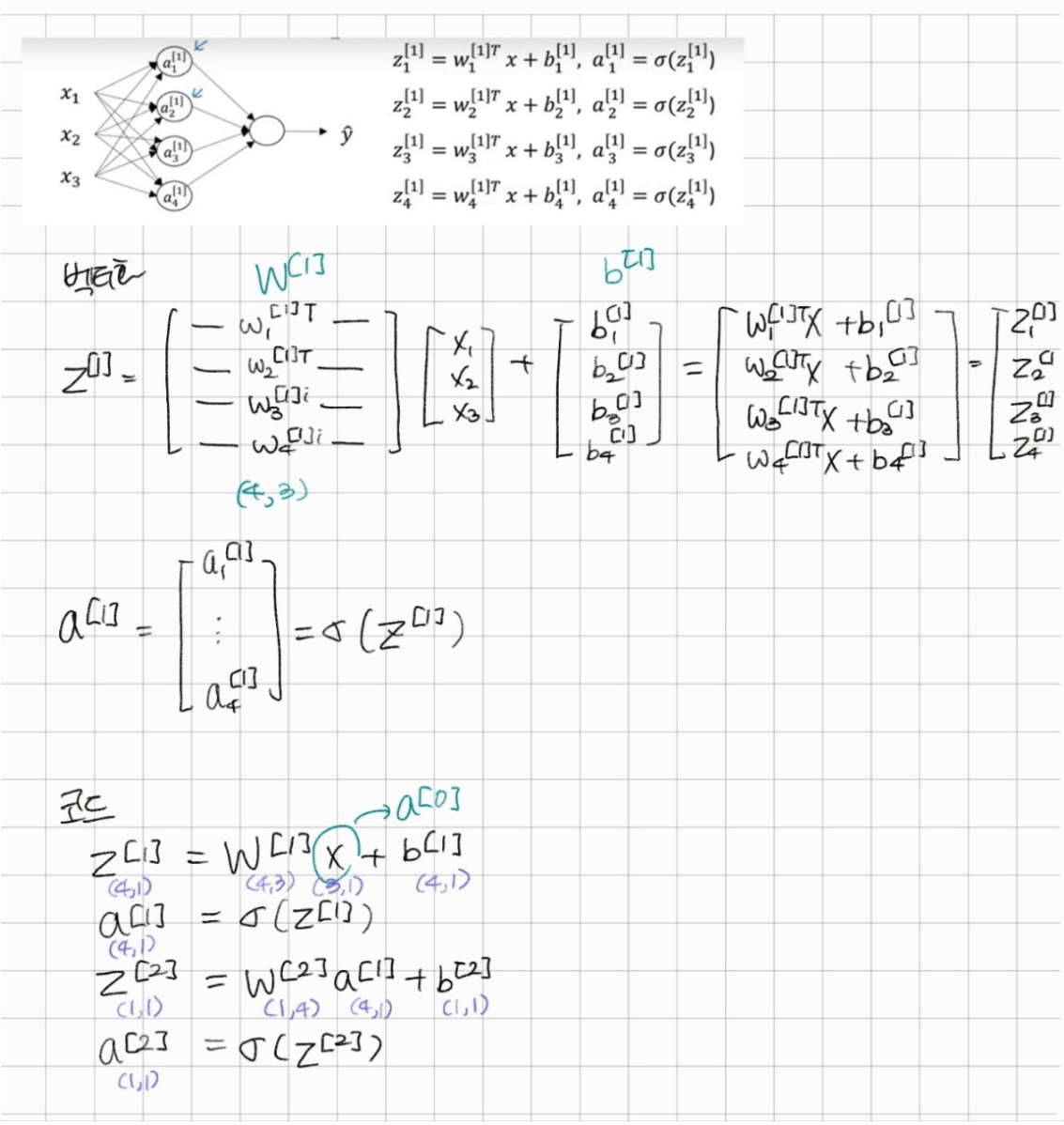
- Input layer : 입력값 X
- Hidden layer : 훈련세트가 입력값 X와 출력값 Y로 이루어져 있는데, 은닉층의 실제 값은 훈련세트에 기록되지 않음.
- Output layer : 예측값인 y_hat의 계산을 책임짐.

2층 신경망 예시 :



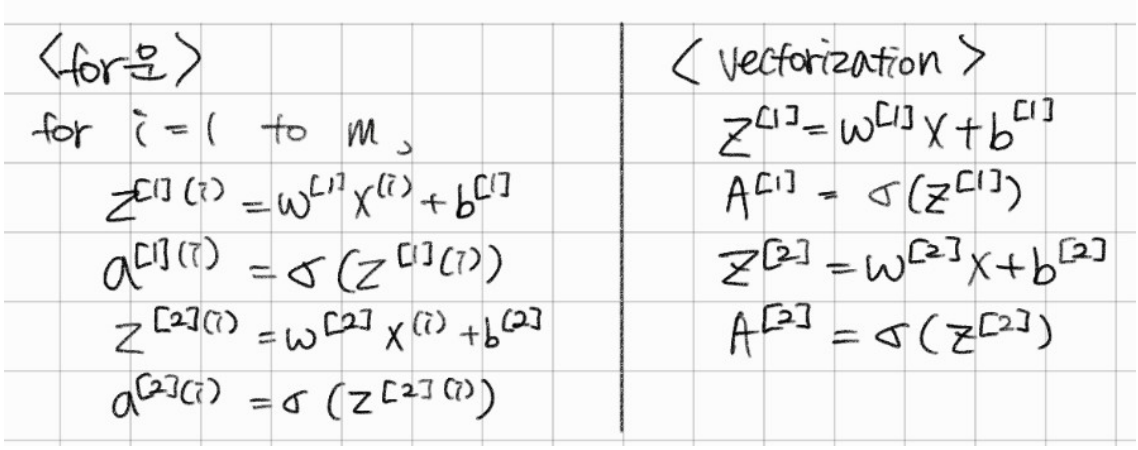
신경망 네트워크 출력의 계산

각 노드마다 z와 a 계산함

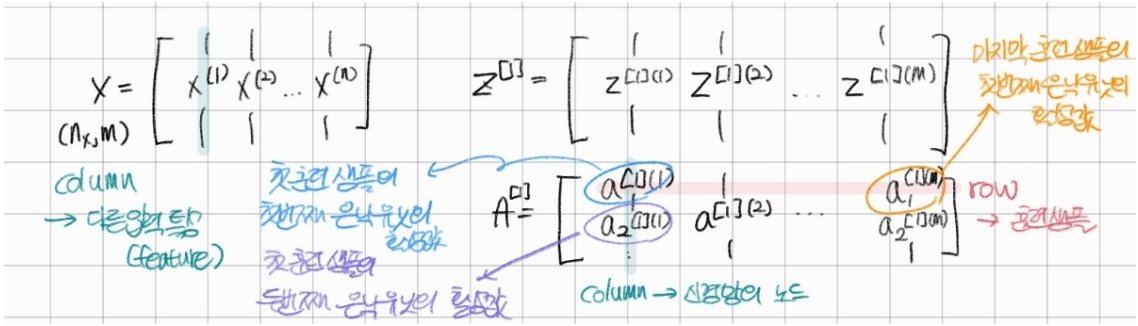


많은 샘플에 대한 벡터화

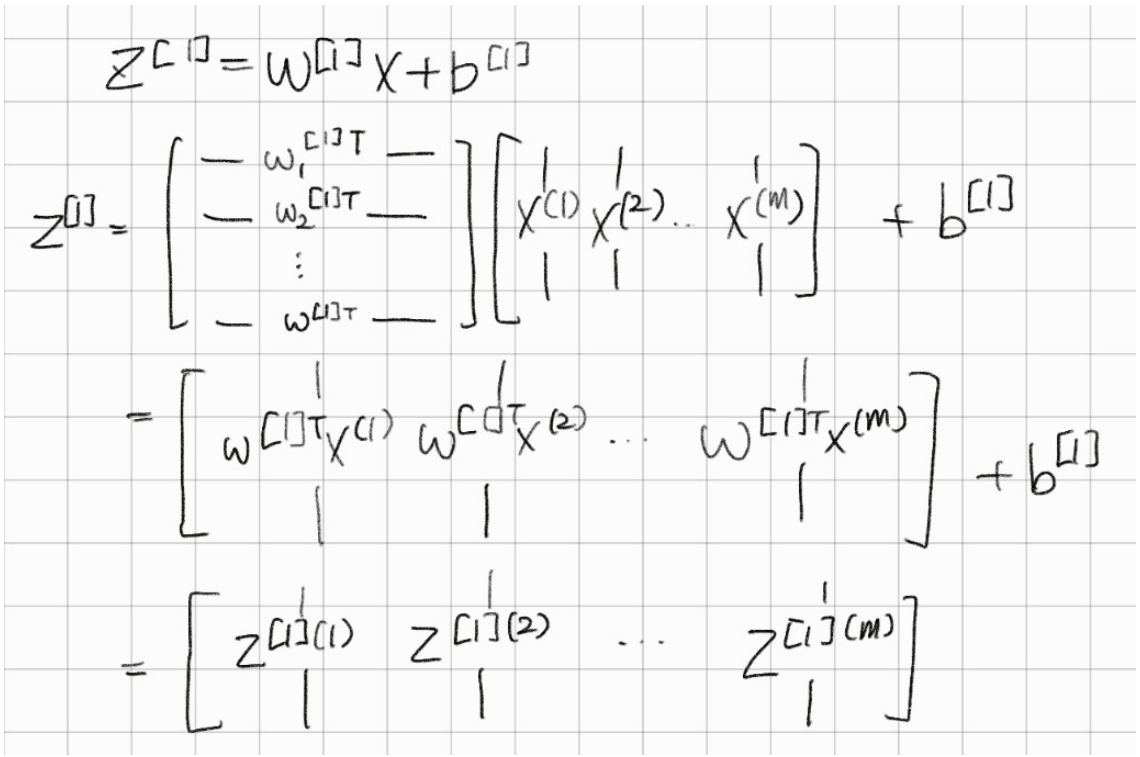
훈련 샘플을 행렬의 열로 쌓아 모든 샘플에 대한 출력값을 거의 동시에 계산할 수 있도록 함.



*참고



벡터화 구현에 대한 설명



$$Z^{[1]} = W^{[1]} X + b^{[1]}$$

$$Z^{[1]} = \begin{bmatrix} - & w_1^{[1]T} & - \\ - & w_2^{[1]T} & - \\ & \vdots & \\ - & w^{[1]T} & - \end{bmatrix} \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} + b^{[1]}$$

$$= \begin{bmatrix} | & | & & | \\ w^{[1]T} x^{(1)} & w^{[1]T} x^{(2)} & \cdots & w^{[1]T} x^{(m)} \\ | & | & & | \end{bmatrix} + b^{[1]}$$

$$= \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$