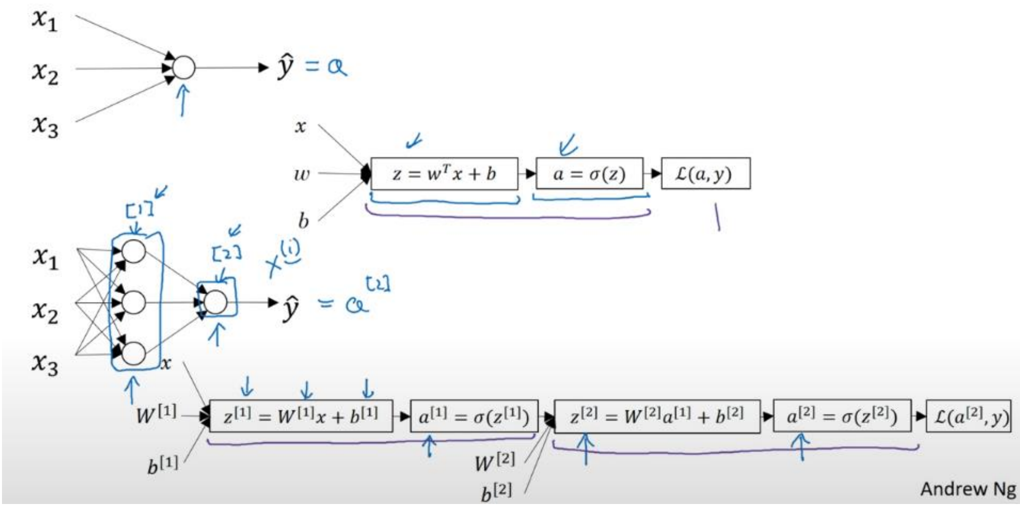
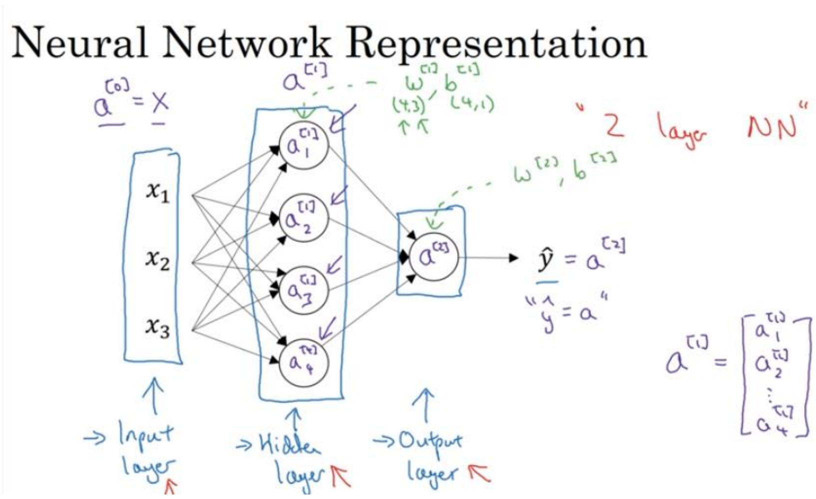
3주차 정리

<신경망 네트워크 개요>



- 신경망은 시그모이드 유닛을 쌓아서 만든다.
- [1] 와 같이, 층이라고 부르는 위첨자 사용하여 신경망의 층 나타낸다.
- 여기서 x^(1) 훈련샘플 첫번째를 의미하고, [1]은 레이어를 뜻한다.
- 위의 그림으로는 z를 계산하고 a를 구하고, z[2]를 구하고 a[2]를 구하고 L을 구한다.
- 역방향 계산을 이용해서 da, dz 구한다.

<신경망 네트워크의 구성 알아보기>



- 입력층 / 은닉층 / 출력층 (y hat 계산) 으로 구성되어있다.
- 은닉층의 실제값은 훈련 세트에 나오지 않아서 알 수 없다.
- 이전에 벡터 x로 표기하던 것을 activation 의미하는 a^[0]으로 표기한다.

  이는 신경망의 층들이 다음 층으로 전달하는 값이다.

- 입력층의 활성값이 a^[0]
- a^[1]은 (1,4)벡터. 은닉 노드가 4개라서 4차원이 된다.
- 위의 그림에서출력층은 a^[2]를 만든다.
- 위의 그림은 2층 신경망이다.

  입력층은 세지 않는다! (은닉층이 첫번째, 출력층이 2번째)

- 은닉층와 출력층은 매개변수가 연관되어 있다.
- w는 (4,3) 벡터, b는 (4,1) 벡터

  b/c 은닉노드 4개 입력 특성이 3개라서.
- 출력층도 w^[2], b^[2]와 연관되어 있다.
- w^[2]는 (1,4)벡터, b^[2]는 (1,1) 벡터

  b/c 은닉층 4개, 출력층에 노드 하나라서.

<신경망 네트워크 출력의 계산>

로지스틱 회귀를 나타내느는 원은 두 단계 (z 구하고 시그모이드 함수에 대입)로 구한다.

⇨ 신경망 네트워크에서는 이 과정을 반복한다.

★ 노드는 두 단계의 계산을 한다.

★ <u>대괄호 안에 있는게 층 번호, 아래 첨자는 노드 번호</u>

<첫번째 노드 계산 과정>

z=w^T+b 를 하고 (이 값들은 모두 [1] 붙고, 첫번째 노드니까 _1도 붙음)

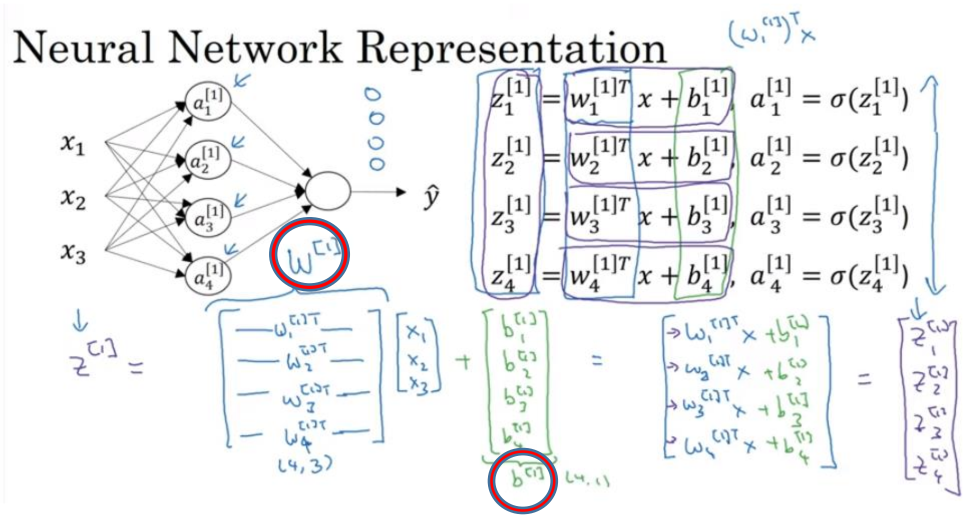
a_1^[1]=시그모이드(z_1^[1])

<은닉층의 두번째 노드 계산 과정>

z_2^[1]=w_2^[1]Tx+b_2^[1]

a_2^[1]=시그모이드(z_2^[1])

⇨ 첫번째 층 두번째 노드

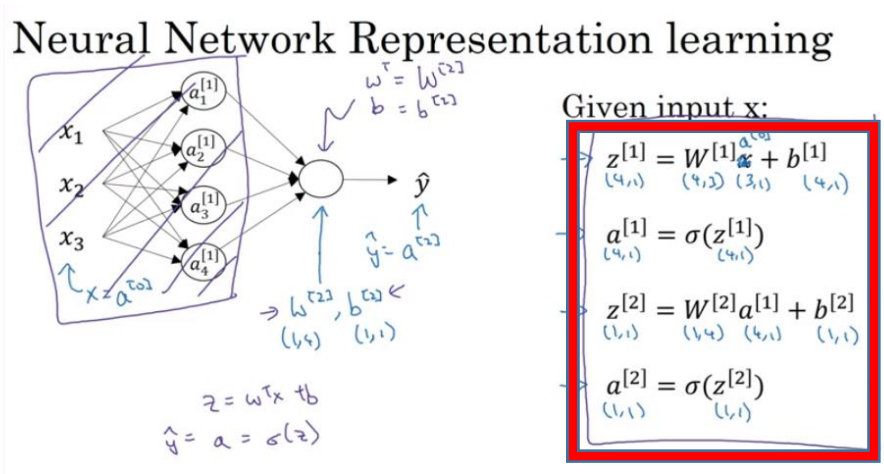


★ z를 벡터화, w를 행렬처럼 쌓기! (각 유닛은 상응하는 w가 있음)

그 네 벡터를 쌓으면 (4,3) 행렬이 된다.

★ 한층에 노드가 여러 개면 세로로 쌓아서 벡터 z^[1]을 만든다.

W^[1]는 (4,3) 행렬, b^[1]은 (4,1) 벡터



z[1]=W[1]*x+b[1]

a[1]=시그모이드(z[1])

x(입력 벡터)는 a[0], yhat은 a[2]

출력층에는 w[2]: (1,4)벡터, b[2]는 하나의 수 => z[2]는 실수가 된다

w는 W[2]T와 비슷, b는 b[2]와 비슷

<많은 샘플에 대한 벡터화>

다수의 훈련 샘플에 대해 벡터화하는 법! 결과는 로지스틱 회귀와 비슷하다 => 행렬의 열로 쌓기

- m개의 훈련샘플이 있다면 x^(1).. x^(n) 이 yhat … yhat^(m)
- a[2](i) 는 i번째 훈련샘플, 2번째 층을 의미

z[1](i)=W[1]x(i)+b(i)

a[1](i)=시그모이드(z[1](i))

이 과정을 벡터화하자

<u>X는 훈련샘플이 열로 쌓인 행렬!</u> (x(1),x(2) 이렇게 나란히 쌓는다.) : (n,m) 행렬

Z[1]=W[1]+b[1], A[1]=sigmoid(Z[1])

Z[2]=W[2]A[1]+b[2], A[2]=sigmoid(Z[2])

a[1](1), a[1](2) … a[1](m)을 열로 쌓으면 A[1]

★ 세로는 은닉 유닛의 번호

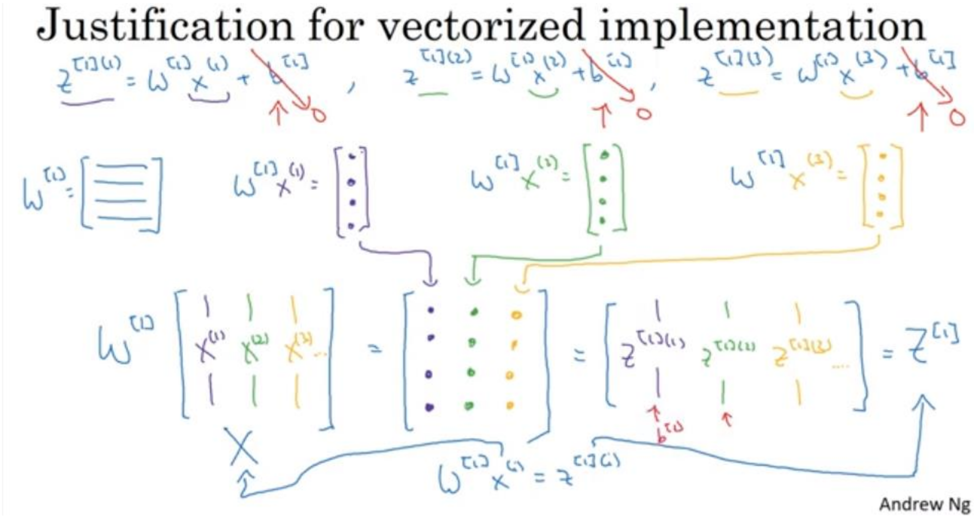
★ 가로로 움직이면 은닉 유닛은 고정, 훈련샘플이 바뀐다

★ 행렬 A의 가로는 다른 훈련의 샘플을 의미

★ 세로 번호는 다른 은닉 유닛을 의미

★ X의 가로는 다른 훈련 샘플 의미, 세로는 다른 입력 특성 의미

<벡터화 구현에 대한 설명>



단순화 하기 위해 b를 0이라고 하자.

W[1]x[1], W[1]x[2], W[1]x[3] 즉 Z[1](3)은 모두 하나의 칼럼 벡터가 될 것이다

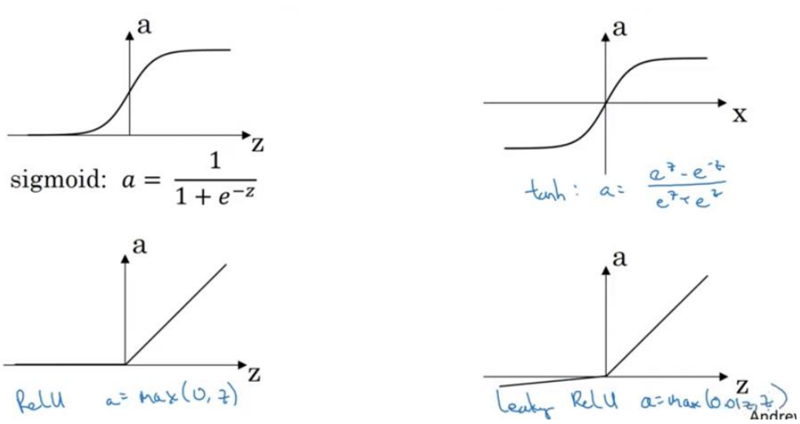
X는 x(1),x(2).. 를 모두 가로로 쌓아서 만든 행렬

XW = z[1](1),z[1](2),z[1](3).. 이 열벡터로 표기된 것과 같다. =Z[1]

파이썬 브로드캐스팅 이용하면 여기에 b 더할 수 있다.

그리고x=a[0], 즉 x(i)=a[0](i)

<활성화 함수>



신경망을 만들 때 은닉층과 출력층에서 어떤 활성화 함수를 쓸지 선택해야함!

은닉층에서는 tanh활성화함수, 출력층에서는 시그모이드 함수

이진분류의 출력층에는 시그모이드 함수, 다른경우에는 relu가 활성화함수의 기본값

시그모이드 함수:

- 이진분류에서 사용 (나머지는 tanh 함수가 더 좋음)
- 출력층에서는 0과 1 사이로 출력하는게 더 좋으니까 시그모이드 활성화 함수를 쓴다

tanh 함수:

- 범위는 y축이 -1에서1까지 : (e^z-e^-z)/(e^z+e^-z) S자형
- 원점을 지나고 시그모이드와 비교했을 때 비율이 달라짐.
- 은닉 유닛에 대해 탄젠트h 함수가 더 좋음: 평균값이 0에 더 가깝기 때문

  (데이터의 중심을 0.5에서 0으로 만들어 학습을 더 쉽게함_

<u>시그모이드함수와 tanh함수의 단점</u> : z가 굉장히 크거나 작으면 함수의 도함수가 굉장히 작아진다.

z갸 크거나 작으면 함수의 기울기가 0에 가까워지고, 경사하강법이 느려진다.

⇨ ReLu 정류선형유닛 사용! : max(0.z) 양수일때는 도함수가 1이고 z가 음수이면 도함수가 0

z가 0일때의 도함수는 정의 X

RELU 함수: 가장 많이 쓰이는 활성화 함수

- relu 단점: z가 음수일 때 도함수가 0임.

  ⇨ leaky relu : 음수일때도 약간의 기울기를 준다 : max(0.001z,z)

장점: 대부분의 z에 대해 기울기가 0과 매우 달라서 신경망은 더 빠르게 학습할 수 있다.

(학습 느리게 하는 원인 : 함수의 기울기가 0에 가까워질 때)

<왜 비선형 활성화 함수를 써야할까?>

선형 활성화 함수(항등함수)라고 가정하면 yhat을 x에 대한 선형 함수로 계산한다.

신경망은 입력의 선형식만을 출력하게 된다 = 은닉층이 없는 것과 같음

(두 선형함수의 조합은 하나의 선형함수와 같음)

출력층에서 선형활성화함수를 쓰고, 은닉 유닛은 비선형 함수를 써야한다.

<활성화 함수의 미분>

시그모이드의 도함수 : d/dzg(z) = g(z)(1-g(z)) , g(z)=1/(1+e^-z)

g'(z)=a(1-a), a=g(z)

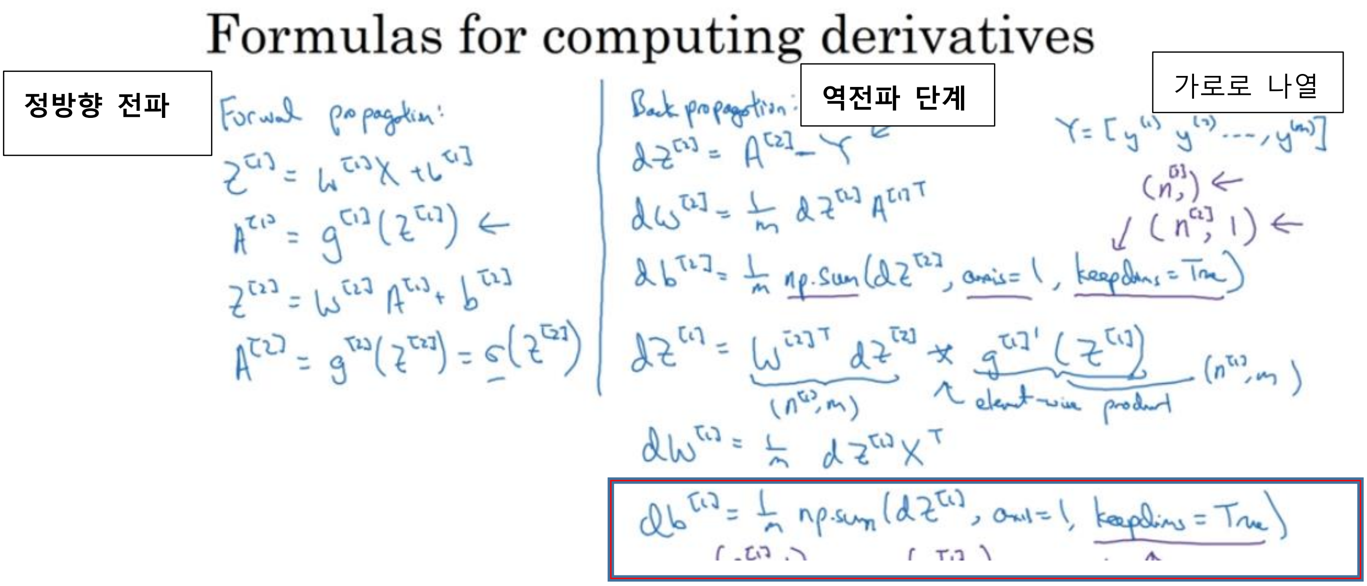
tanh 함수 도함수: 1-(tanh(z))^2

g'(z)=1-a^2

RELU 도함수: g'(z)=0 if z<0, 1 if z>=0

leaky RELU 도함수: g'(z)=0.01 if z<0, 1 if z>0

<신경망 네크워크와 경사하강법>



비용함수는 손실함수의 평균

0이 아닌 값으로 변수 초기화

경사하강법 반복할 때마다 예측값 계산, 경사하강법 반복하면 => w^[1]=w^[1]-알파(학습률)dw^[1]

np.sum ( ) 에서 axis=1 => horizontal 의미, keemdims=True => 잘못된 1차원 배열을 출력하지 않고 (n,1) 벡터로 출력되도록 한다

<역전파에 대한 이해>

계산 그래프를 이용해서 식 유도해보기

da=-y/a + (1-y)/(1-a) ; dz=a-y

★ 역전파 구현할 때 팁 : 차원이 일치하는지 확인해라

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y \qquad dZ^{[2]} = A^{[2]} - Y$$

$$J(\ldots) = \frac{1}{m}\sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$$

$$dW^{[2]} = dz^{[2]} a^{[1]^T} \qquad dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$

$$db^{[2]} = dz^{[2]} \qquad db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

elementwise product

$$\underbrace{dz^{[1]}}_{(n^{[1]},1)} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]}) \qquad \underbrace{dZ^{[1]}}_{(n^{[1]},m)} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]},m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]},m)}$$

전에는 n[1],1

요소별 곱셈

$$dW^{[1]} = dz^{[1]} x^T \qquad dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

지금은 n[1],m

$$db^{[1]} = dz^{[1]} \qquad db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

m으로 나누는 이유는 비용함수가 손실함수의 합을 m을 나눈 것이기 때문

<랜덤 초기화>

(b를 0으로 초기화하는 것은 괜찮다)

가중치를 0으로 초기화하면

- 어떤 샘플의 경우에는 a[1]_1 이랑 a[1]_2 가 같은 값을 가진다
- 은닉 유닛이 같은 값으로 계산하고, dz[1]_1 이랑 dz[1]_2도 같음. => 가중치의 결과값이 항상 같다
- 은닉 유닛이 출력 유닛에 같은 값 전달하고 은닉 유닛은 같은 함수를 가지게 된다.

⇨ 변수 랜덤 초기화

w[1]= np.random.rand((2,2)) * 0.01 (작게 만드는)

⇨ 왜 0.01? 가중치의 초기값은 매우 작은 값으로 하는 것이 좋음.

가중치가 너무 크고 활성값 계산하면 w와 z도 큰 값이 되고 경사의 기울기가 매우 작아서 학습속도가 느려짐

b[1]= np.zeros((2,1))