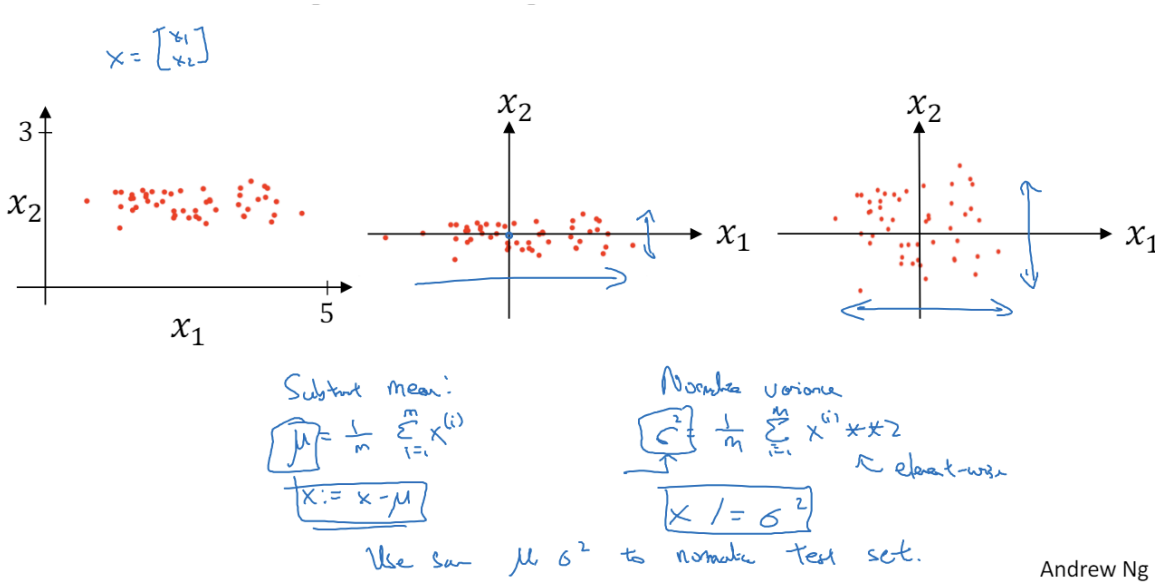
[Week 7] 1. Normalizing Inputs

신경망의 훈련을 빠르게 하는 방법 중 하나는 입력을 정규화 하는 것이다.

Normalize

1. 평균을 뺀다(= 평균을 0으로 만든다)
2. 분산을 정규화한다. (모든 특성 벡터의 분산을 1로 만든다)

   여기서는 x1이 x2보다 더 큰 분산을 가지고 있음

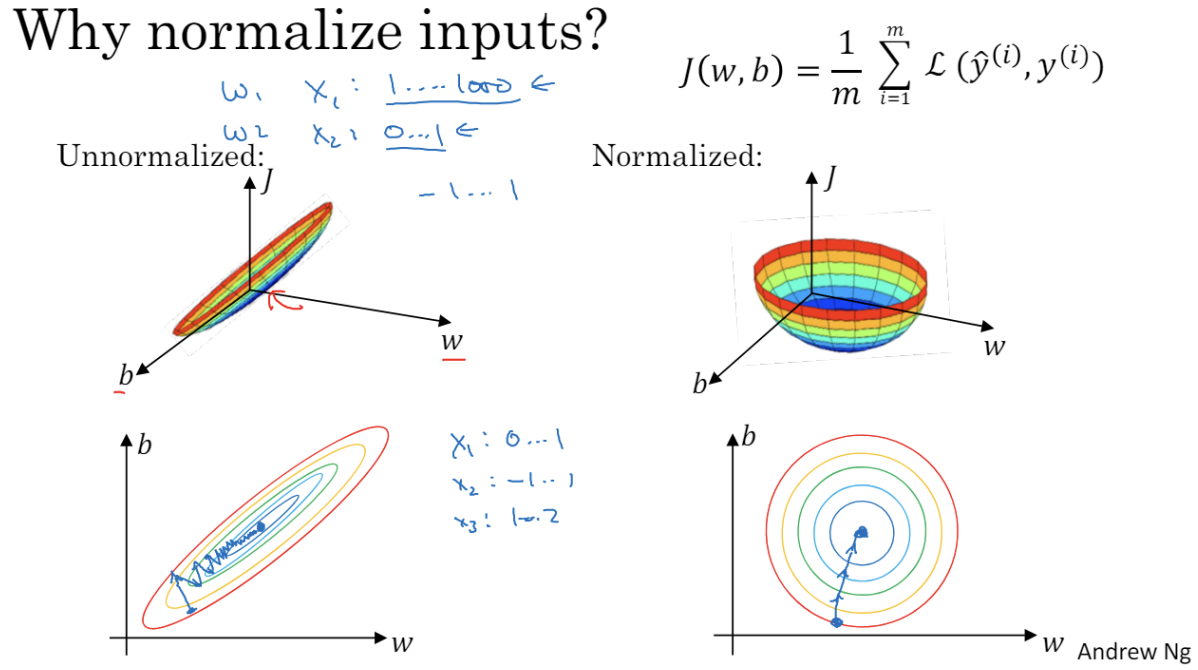


!

Why normalize inputs?

비용함수는 다음과 같다.

우리가 정규화를 하지 않았다고 해보자. 이때 특성들이 매우 다른 scale을 가지고 있다고 생각해보자. 특성 벡터 x1의 범위는 1~1000이고, x2의 범위는 0~1이라면, 아래의 왼쪽 그래프처럼 비용함수가 비대칭적으로 나타나게 된다. 그러나 정규화를 하면 비용함수는 평균적으로 대칭적인 모양을 가지게 된다(오른쪽).



정규화 하지 않은 비용함수에서의 gradient descent를 사용하게 되면 매우 작은 learning rate를 사용하게 된다. 반면에 정규화를 한 비용함수의 경우 바로 최솟값으로 갈 수 있다. 즉, 각각의 특성벡터가 서로 비슷한 분산을 가져야 비용함수 J를 빠르게 최적화할 수 있게 된다는 것을 알 수 있다. **⇒ 정확히 왜????**

[Week 7] 2. Weight Initialization - Vanishing/Exploding Gradients

‘깊은’ 신경망을 훈련시킬 때, 기울기가 아주 작아지거나 커지는 문제가 발생할 수 있다. 만약 기울기가 매우 작다면 경사하강법에서 가중치 값이 매우 조금씩 움직이게 되기 때문에 학습하는데 매우 오래 걸릴 것이다. 경사 손실 및 폭발을 줄이기 위한 방법으로는 가중치를 무작위로 초기화하는 과정에서 **분산을 적절하게 설정**하는 것이 있다.

Weight initialization

하나의 뉴런을 먼저 예시로 살펴보자.

(사진)

우리는 z의 값이 너무 크거나 작아지지 않도록 만들어야 한다. 가중치의 개수(n, input feature의 개수)가 많아질수록, w의 값은 작아져야 한다. 더 많은 항들이 더해지기 때문에(w_n*x_n), 이 값들이 너무 커지지 않게 하기 위해서는 각각의 항의 값이 작아야 함을 알 수 있다. 이러한 원리를 바탕으로, 아래의 식을 사용한다.

random.randn... (사진)

n^[l-1] : 층 l에서 해당 층의 각 유닛에 대해 n^[l-1]의 입력을 갖는다(input 개수)

왜냐면 각 레이어에 대한 가중치인 W^[l]을 1보다 너무 커지거나 너무 작아지지 않게 설정해서 폭발하거나 소실되지 않게 하기 때문이다.

각 활성화 함수에서 분산 설정 방법

- ReLU

$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

- Tanh

$$Var(W) = \sqrt{\frac{1}{n_{in}}}$$

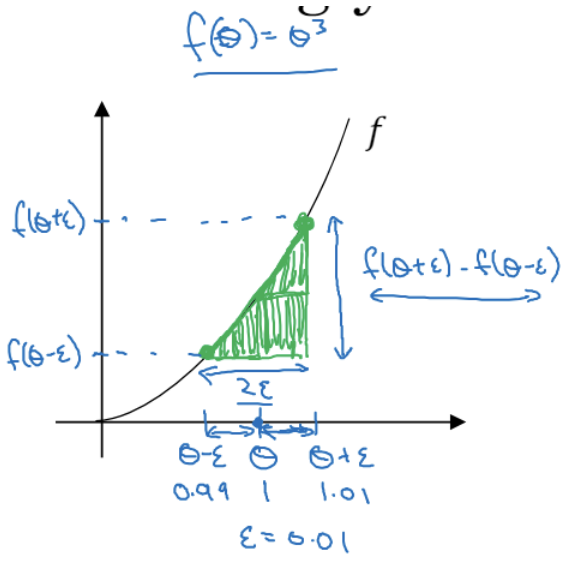
- Xaivier initialization

$$Var(W) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

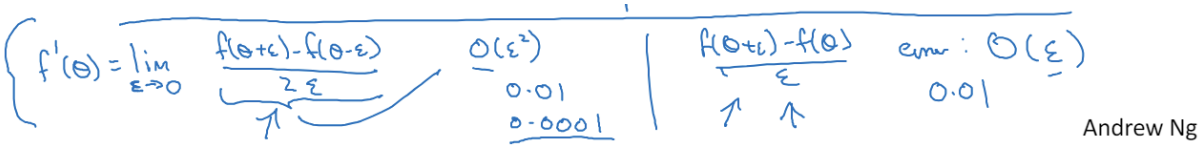
[Week 7] 3. Gradient Checking

Back propagation을 구현할 때 경사검사라는 테스트가 있다. 이를 역전파를 맞게 구현했는지 확인할 수 있다.

Numerical Approximations of Gradients



더 큰 삼각형에서 기울기를 구하는 것(높이/너비)이 도함수를 근사하는 데 더 나은 값을 제공한다. 위의 삼각형이나 아래의 삼각형보다! 왜냐 한쪽의 차이가 아닌 양쪽의 차이를 사용하기 때문이다. 위의 도함수를 구하는 방식을 경사 검사에서 사용하게 되면, 한쪽의 차이를 이용했던 것보다 두배는 느리게 실행될 것이다. 하지만 이 방식이 더 정확한 기울기를 만들기 때문에 더 낫다.



Andrew Ng

오른쪽의 방식(양쪽의 차이를 이용)은 오차를 빅오표기법으로 나타내면 앱실론의 제곱이고, 왼쪽의 방식(한쪽의 차이를 이용)은 그냥 앱실론이기 때문에 오른쪽의 방식으로 기울기를 구하는 것이 더 정확함을 알 수 있다.

Gradient Checking

경사 검사는 시간을 절약하고 역전파의 구현에 대한 버그를 찾는데 도움을 준다. 그렇다면 어떻게 사용하는 것일까?

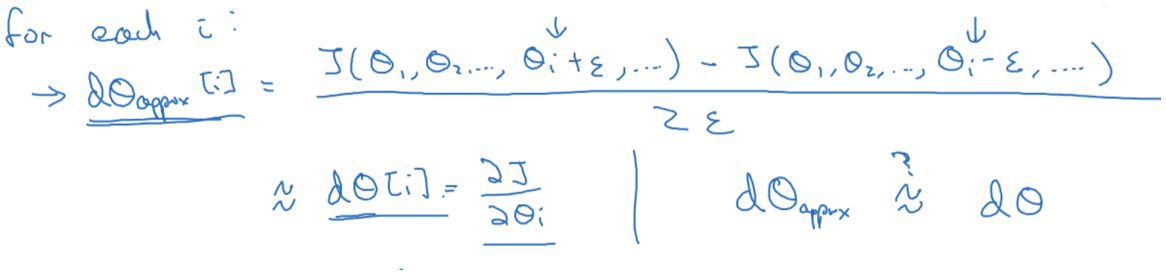
1st. 각 레이어에 대한 W,b 들을 하나의 큰 벡터 **θ**로 바꾸는 것(concatenate)이다.

비용함수 J(w^[1],b^[1], ..., w^[l],b^[l])가 w와 b에 대한 식이 아닌 J(θ)에 대한 함수로 바꿈

2nd. 각 레이어에 대한 dW,db들을 하나의 큰 벡터 **dθ**로 만든다.

w^[1]과 dw^[1]은 같은 차원이고, b^[1]과 db^[1]은 같은 차원이므로 θ와 dθ는 같은 차원

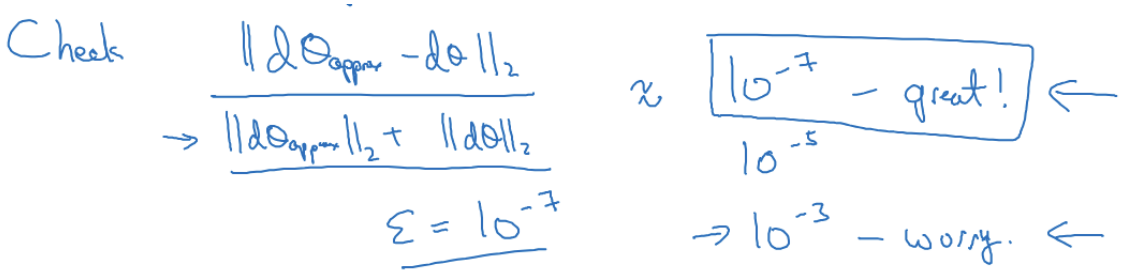
여기서 질문! dθ가 비용함수 J(θ)의 기울기인가?



θ는 매우 큰 matrix이기 때문에 여러개의 θ1,θ2,.. 로 나눌 수 있다. 이때 θi에 대해서만 변화시켜보자(다른 θ의 요소들은 그대로 둔다). 우리가 위에서 사용한 방식으로 기울기를 구한다.

이때 dθ_i가 비용함수 J의 도함수라면 J에대한 θ_i의 편미분이 dθ[i]여야 한다.

우리는 모든 i에 대해 계산하게 되면, dθapprox 값을 구할 수 있게 된다. 이때 dθ와 근사한지 확인해야 한다.(수치 미분과 일반 미분 값을 비교하는 것이다) 이를 확인하는 방법으로 아래처럼 유클리드 거리를 계산할 수 있다.



두 값의 차이를 계산했을 때, 보통 10^(-7)보다 작으면 잘 계산되었다고 판단한다. 10^(-5)면 나쁘지 않다, 10^(-3)이면 버그의 가능성이 크기 때문에 자세히 살펴보아야 한다. 개별적

인 θ의 값을 각각 살펴보고 dθapprox[i]와 dθ[i]의 차이가 심한 값을 추적해서 계산이 옳지 않은 곳을 찾아야 한다. 만약 경사 검사를 실시하여 값을 비교했을 때 오차가 크다면 디버깅의 과정을 거쳐 작은 값이 나오도록 해야 한다.

Gradient Checking implementation Notes

1. 훈련 시 사용 X. 디버깅 시에만 사용 O

   속도가 느리기 때문

2. 경사 검사에 실패한다면 개별적인 컴포넌트를 살펴볼것. 특정 부분에서 계속 실패한다면 그 경사값이 나온 layer의 문제
3. 경사 검사를 할 때 정규화 항을 포함하는 것을 잊지 말것
4. dropout을 끄고 할것(keep_prob=1.0)
5. 무작위 초기화 과정에서 w와 b값이 0에 가깝게 설정되었을 때 경사하강법이 맞게 구현된 경우. 역전파의 구현이 w와 b가 0에 가까울 때만 맞는것일 수 있음. w와 b가 커지면 부정확해진다. 네트워크를 잠시 훈련해서 작은 무작위 초기값에서 벗어나도록 하고 다시 경사검사를 실행한다.