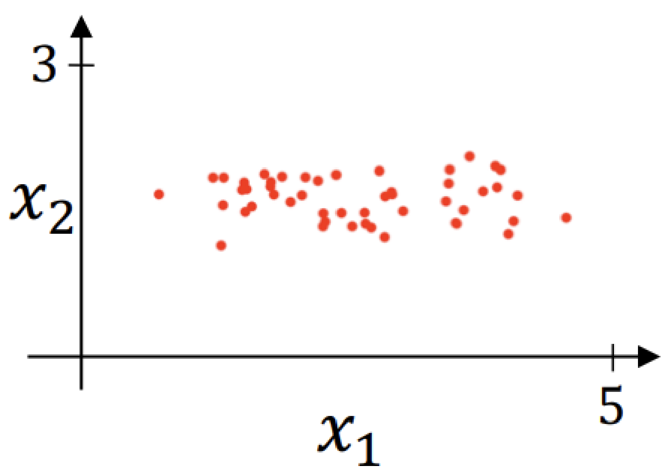
[딥러닝 2단계] 3. 최적화 문제 설정

1. 입력값의 정규화

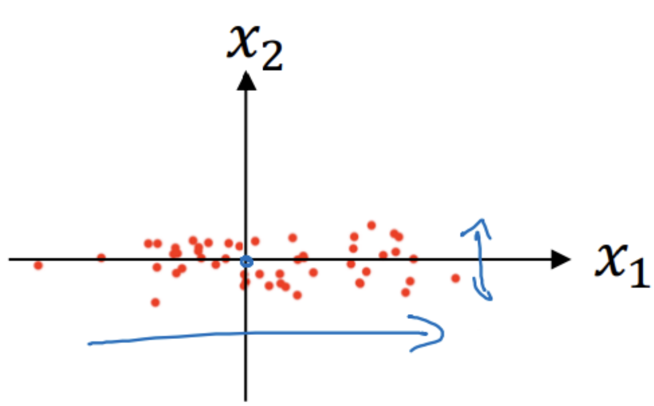
신경망 훈련 빠르게 할 수 있는 하나의 기법 = 정규화

Normalizing training sets

- 입력 특성 x: 2차원 [x1, x2]
- 훈련세트 산포도

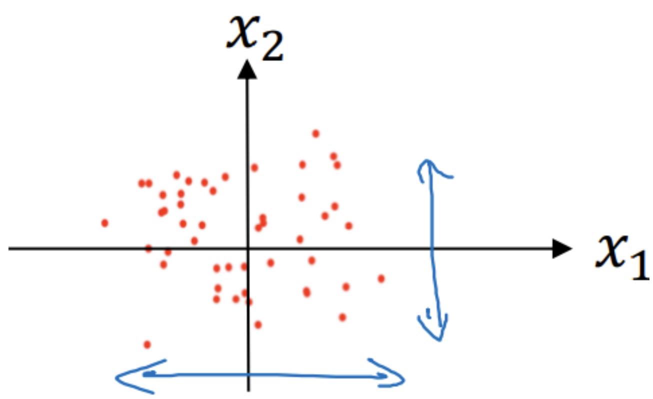


- **입력 정규화 단계**
    - 평균을 빼는것, 즉 0으로 만들기



> 💡 $\mu = \frac{1}{m}\sum_{i=1}^{m} X^{(i)}$
> $X := X - \mu$

- 분산을 정규화하는 것
    - * : element-wise
    - x1의 분산>x2의 분산



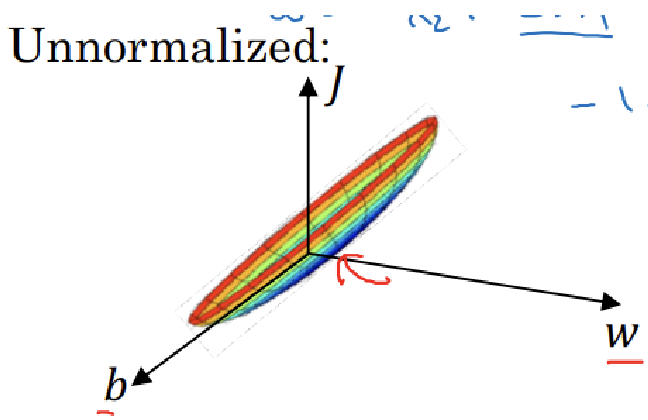
> 💡 $\sigma^2 = \frac{1}{>}m\sum_{i=1}^{m} X^{(i)} * *2$
> $X/ = \sigma^2$

- 테스트 세트를 정규화할 때도 같은 μ와 σ를 사용해야함

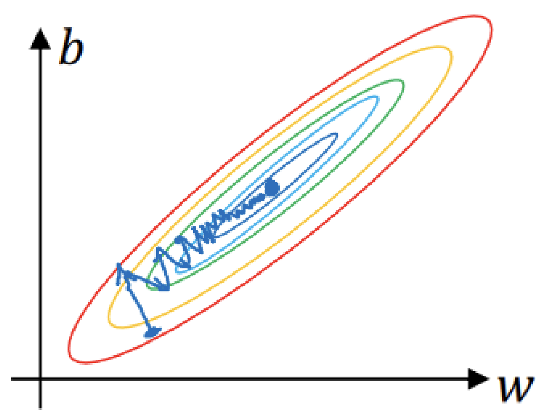
Why normalize inputs?

왜 입력 특성을 정규화하기를 원할까?

- 정규화하지 않은 입력특성을 사용할 때
    - 매우 구부러진 활처럼 가늘고 긴 모양의 비용함수

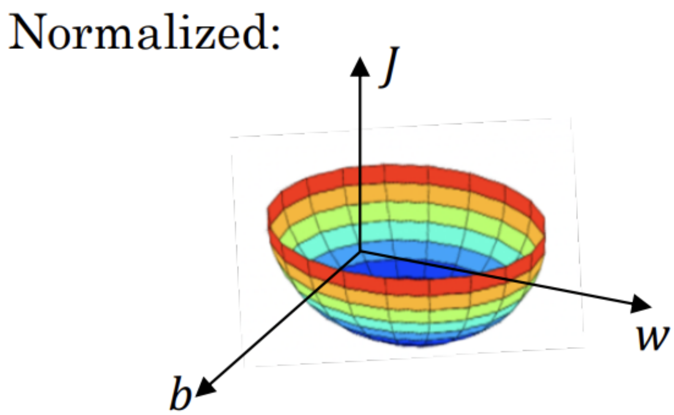


- 만약 특성들이 매우 다른 크기를 가지고 있다면 (ex. x1 1~1000, x2 0~1) 매개변수에 대한 비율, 값의 범위는 w1, w2 굉장히 다른 값을 가지게 됨
- 경사하강법 실행 -> 매우 작은 학습률

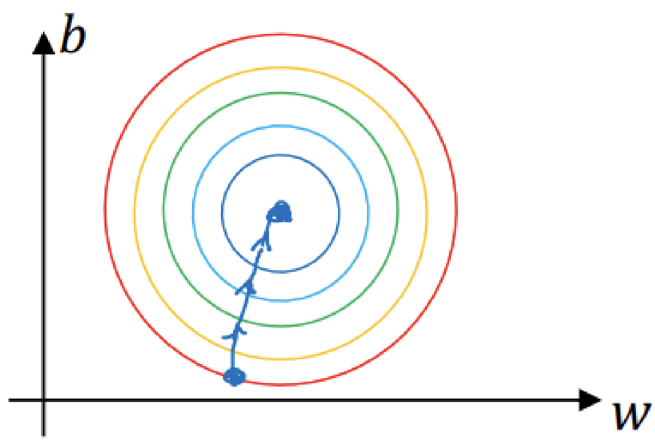


- 최솟값에 이르는 길 찾기 전까지 많은 단계 필요

- 특성을 정규화
    - 비용 함수는 평균적으로 대칭적인 모양



- 경사하강법 어디서 시작하든 최솟값으로 바로 갈 수 있음



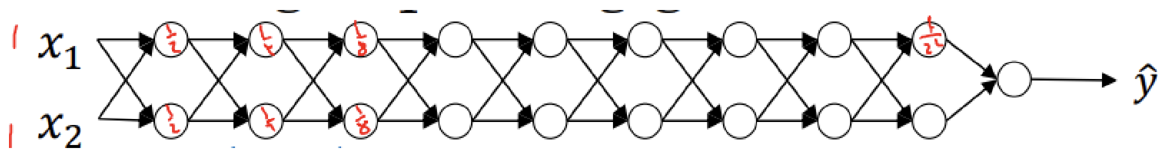
- 특성이 비슷한 크기를 가질 때, 비용함수가 더 둥글고 최적화하기 쉬운 모습이 된다는 직관
- 평균 0으로 설정, 모든 특성 비슷한 크기로 보장할 수 있는 분산 설정
  -> 학습 알고리즘 빠르게 실행
- 특성 크기 비슷하면 이 단계는 중요하지 않으나, 이 정규화는 어떤 해도 가하지 않으므로 되도록 하는게 좋음

2. 경사소실/경사폭발

- 신경망을 훈련시키는 것의 문제점: **경사의 소실과 폭발**
- 미분값 혹은 기울기 아주 작아지거나 커질 수 있음
- > 훈련 어렵

Vanishing/exploding gradients

- 매우 깊은 신경망 훈련



가정

- 매개변수 $w^{[1]}, ...w^{[L]}$
- 활성화 함수 g(z)=z가 선형 활성화 함수
- $b^{[l]} = 0$

> 💡 $\hat{y} = w^{[L]}w^{[L-1]}w^{[L-2]}...w^{[3]}w^{[2]}w^{[1]}X$
> - $w^{[1]}X = z^{[1]} = g(z^{[1]}) = a^{[1]}$
> - $w^{[2]}w^{[1]}X = g(w^{[2]}a^{[1]}) = g(z^{[2]}) = a^{[2]}$
> - $w^{[3]}w^{[2]}w^{[1]}X = a^{[3]}$

1. $w^{[l]} > I$
   $w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$라고 해보자
   $\hat{y} = w^{[L]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} X$
   -> $1.5^{L-1}X$

- 매우 깊은 신경망 갖는다면 y의 값 폭발

1. $w^{[l]} < I$
   $w^{[l]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$라고 해보자
   $\hat{y} = w^{[L]} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{L-1} X$
   = $0.5^{L-1}X$

- w를 1보다 작은 값으로 교체하면 기하급수적으로 감소

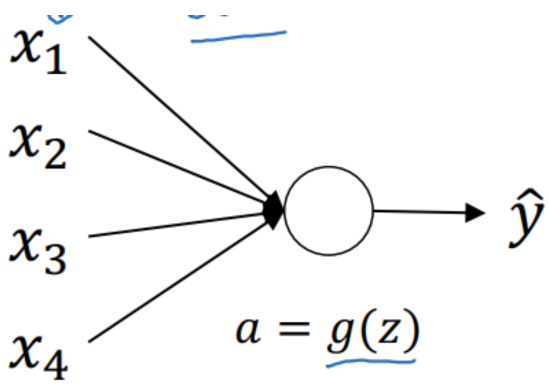
💡 가중치 $w^{[l]}$이 **단위행렬보다 조금 더 크다면**, 매우 깊은 네트워크의 경우 **활성값은 폭발**할 수 있음
가중치$w^{[l]}$이 **단위행렬보다 조금 더 작다면**, 매우 깊은 네트워크긔 경우 활성값은 **기하급수적으로 감소**할 것

- 경사하강법에서 계산하는 경사가 층의 개수에 대한 함수로 기하급수적으로 증가하거나 감소한다는 것을 보여주는데 사용할 수 있음
- 현대 신경망은 보통 L=150
- 깊은 신경망에서 활성값이나 경사가 L에 대한 함수로 기하급수적으로 증가, 감소한다면 값들은 아주 커지거나 작아짐
  -> 훈련 시키는 것이 어려워짐 (특히 기하급수적으로 작은 경우)
  -> 학습시키는데 아주 오랜 시간이 걸림

3. 심층 신경망의 가중치 초기화

경사소실, 폭발 문제 해결법: 신경망에 대한 가중치 초기화를 신중하게 선택

1. 하나의 뉴런이 있는 예제



- 특성 4개: $x_1, ...x_4$
- a=g(z)

$z = w_1x_1 + w_2x_2 + ... + w_nx_n$~~+b~~

- b=0

- n=뉴런으로 들어가는 입력 특성의 개수
- z의 값이 너무 크거나 작아지지 않도록 만들어야함
    - n이 커지면 w_i들이 작아져야함

$Var(w_i) = \frac{1}{n}$

$w^{[l]} = np.random.randn(shape) * np.sqrt(\frac{1}{n^{[l-1]}})$

ReLU 활성화 함수를 사용하는 경우

$Var(w_i) = \frac{2}{n}$

$w^{[l]} = np.random.randn(shape) * np.sqrt(\frac{2}{n^{[l-1]}})$

tanh 활성화 함수를 사용하는 경우

- 세이비어 초기화

1)

$w^{[l]} = np.random.randn(shape) * np.sqrt(\frac{1}{n^{[l-1]}})$

2)

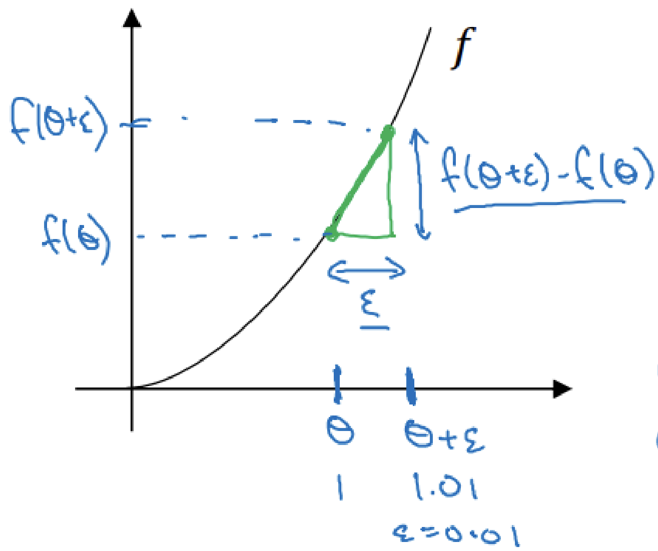
$w^{[l]} = np.random.randn(shape) * np.sqrt(\frac{2}{n^{[l-1]}+n^{[l]}})$

4. 기울기의 수치 근사

Checking your derivative computation

한쪽의 차이를 이용하는 경우

$f(\theta) = \theta^3$



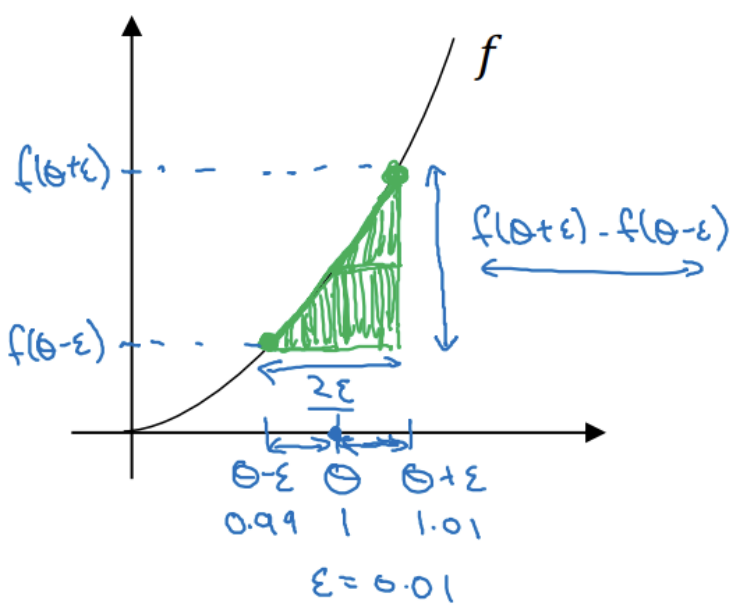
💡 $\frac{f(\theta+\epsilon)-f(\theta)}{\epsilon} \approx g(\theta)$

$\frac{1.01^3-1^3}{0.01} = 3.0301 \approx 3$

- 근사오차=0.0301

양쪽의 차이를 이용하는 경우

$f(\theta) = \theta^3$



- 더 큰 삼각형에서 너비 분의 높이를 구하는 것이 θ에서의 도함수를 근사하는데 더 나은 값을 제공

> 💡 $\frac{f(\theta+\epsilon)-f(\theta-\epsilon)}{2\epsilon} \approx g(\theta)$

$\frac{1.01^3-0.99^3}{2(0.01)} = 3.0001 \approx 3$
$g(\theta) = 3\theta^2 = 3$

- 근사오차=0.0001

> 💡 양쪽의 차이를 사용하는 것이 한쪽의 차이를 사용하는 것보다 더 정확

미적분 추가 이론

$\epsilon < 1$이므로

$f'(\theta) = \lim_{\epsilon \to 0} \frac{f(\theta+\epsilon)-f(\theta-\epsilon)}{2\epsilon}$

-> $O(\epsilon^2)$ -> 0.0001

$f'(\theta) = \lim_{\epsilon \to 0} \frac{f(\theta+\epsilon)-f(\theta)}{\epsilon}$

-> $O(\epsilon)$ ->0.01

5. 경사 검사

경사 검사 -> 시간 절약, 역전파의 구현에 대한 버그를 찾는데 도움

Gradient check for a neural network

1. 매개변수 $W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}$를 하나의 큰 벡터 θ로 바꾸기

- $W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}$를 벡터로 만들어 concatenate
- $J(W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}) = J(\theta)$

2. $dW^{[1]}, db^{[1]}, ..., dW^{[L]}, db^{[L]}$를 큰 벡터 $d\theta$로 만들기

- $dW^{[1]}, db^{[1]}, ..., dW^{[L]}, db^{[L]}$를 벡터로 만들어 concatenate

질문) dθ가 비용함수 J(θ)의 기울기인가?

Gradient checking (Grad check)

- J는 이제 매우 큰 매개변수 θ에 관한 함수
  J(θ)=J(θ1,θ2,θ3,...)로 확장 가능
- 경사 검사 구현하기 위해 반복문 구현

for each i:
$d\theta^{[i]}_{approx} = \frac{J(\theta_1,\theta_2,...,\theta_i+\epsilon,...)-J(\theta_1,\theta_2,...,\theta_i-\epsilon,...)}{2\epsilon}$
$\approx d\theta^{[i]} = \frac{\partial J}{\partial \theta_i}$

- $d\theta_{approx} \approx d\theta$인지 확인해야 함

방법

- 이 두 벡터의 유클리드 거리 계산
  =$\|d\theta_{approx} - d\theta\|_2$
  - 이 값을 제곱하지 않는다는 것을 명심
- 벡터의 길이로 정규화하기 위해 $\|d\theta_{approx}\|_2 + \|d\theta\|_2$로 나눠줌
  - 이 식을 비율로 바꿔줌

💡 $\frac{\|d\theta_{approx} - d\theta\|2}{\|d\theta approx\|_2 + \|d\theta\|_2}$

- $\approx 10^{-7}$ - 근사가 매우 잘 되었다는 뜻
  -> 올바른 구현을 한 것
- $\approx 10^{-5}$ - 괜찮지만 너무 큰 벡터 원소가 있는 것은 아닌지 이중으로 확인
- $\approx 10^{-3}$ - 버그의 가능성이 높으므로 의심
  -> 특정 i에 대해 $d\theta^{[i]}_{approx}$와 $d\theta^{[i]}$의 차이가 심한 값을 추적해서 미분 계산이 옳지 않은 곳이 있는지 확인

6. 경사 검사 시 주의할 점

1. 훈련에서 경사 검사를 사용하지 말고 디버깅을 위해서만 사용

- 모든 i의 값에 대한 $d\theta_{approx}$를 계산하는 것은 매우 느림

2. 만약 경사 검사의 알고리즘이 실패한다면 개별적인 컴포넌트를 확인해 버그를 확인

- $d\theta_{approx}$와 dθ가 매우 먼 경우 서로 다른 i에 대해 어떤 $d\theta_{approx}^{[i]}$의 값이 $d\theta^{[i]}$와 매우 다른지 확인
- 항상 버그를 바로 찾을 수 있게 하는건 아니지만 어디서 버그를 추적할 수 있을지에 대한 추측 제공

3. 정규화를 기억

$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \left\| w^{[l]} \right\|_F^2$

- dθ는 θ에 대응하는 J의 경사로 정규화항을 포함

4. 경사 검사는 드롭아웃에서는 작동하지 않음

- 드롭아웃은 모든 반복마다 은닉 유닛의 서로 다른 부분 집합을 무작위로 삭제
- > 드롭아웃이 경사하강법을 시행하는 비용함수 J를 계산하는 쉬운 방법이 없음
- 드롭아웃을 끄고(keep_prob=1.0) 알고리즘이 드롭아웃 없이 맞는지 이중 검사하기 위해 경사 검사를 사용하고 드롭 아웃을 켜는 것을 추천

5. 무작위 초기화에서 w,b가 0에 가까울 때 경사하강법의 구현이 맞게 된 경우

- 경사하강법 실행하면 w,b 점점 커짐
  - 역전파의 구현이 w,b가 0에 가까울 때만 맞는 것일 수 있음
- 방법

1. 초기화 상태에서 경사 검사를 실행
2. 네트워크를 훈련해서 w,b가 0보다 커질 시간을 줌
3. 경사 검사 한 번 더 실행

해당글은 부스트코스의 [딥러닝 2단계] 3. 최적화 문제 설정 강의를 듣고 작성한 글입니다.

velog 링크