[Week9]_이원주

최적화 문제 설정

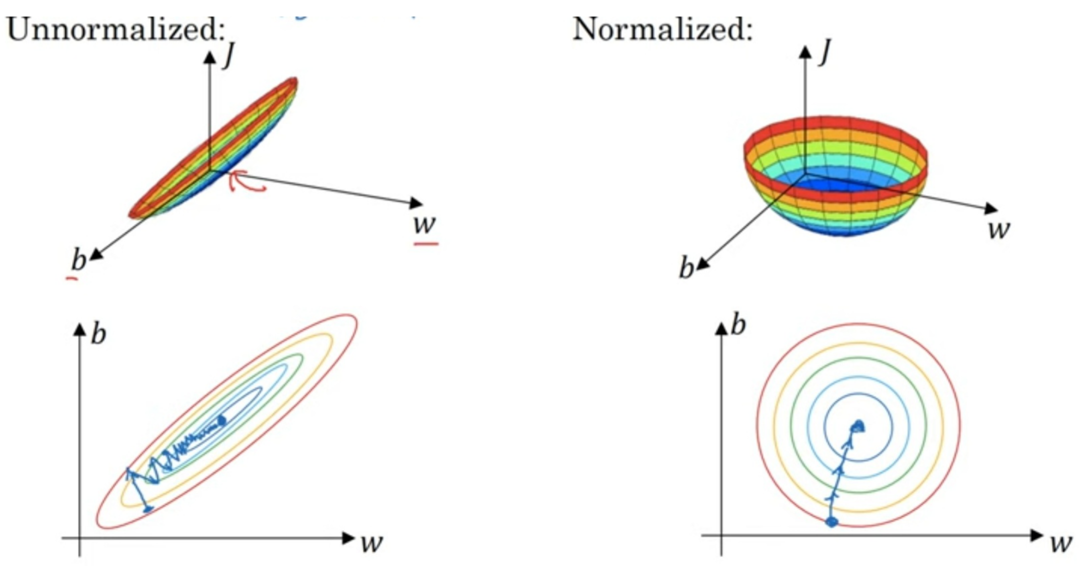
정규화 (Normalizing)

[요약]

- **정규화 (Normalizing)**
  - 효과 : 학습을 빠르게 만듦.
  - 방법 :
    - 확통에서 배운 정규화 공식이랑 똑같음.
    - 이때 Train set / Test set에서 같은 μ, σ값 사용

- **효과 : 학습을 빠르게 만듦.**
  - ▼ 이유

    직관적으로 이해해보자고.



- 정규화 전 (왼쪽)

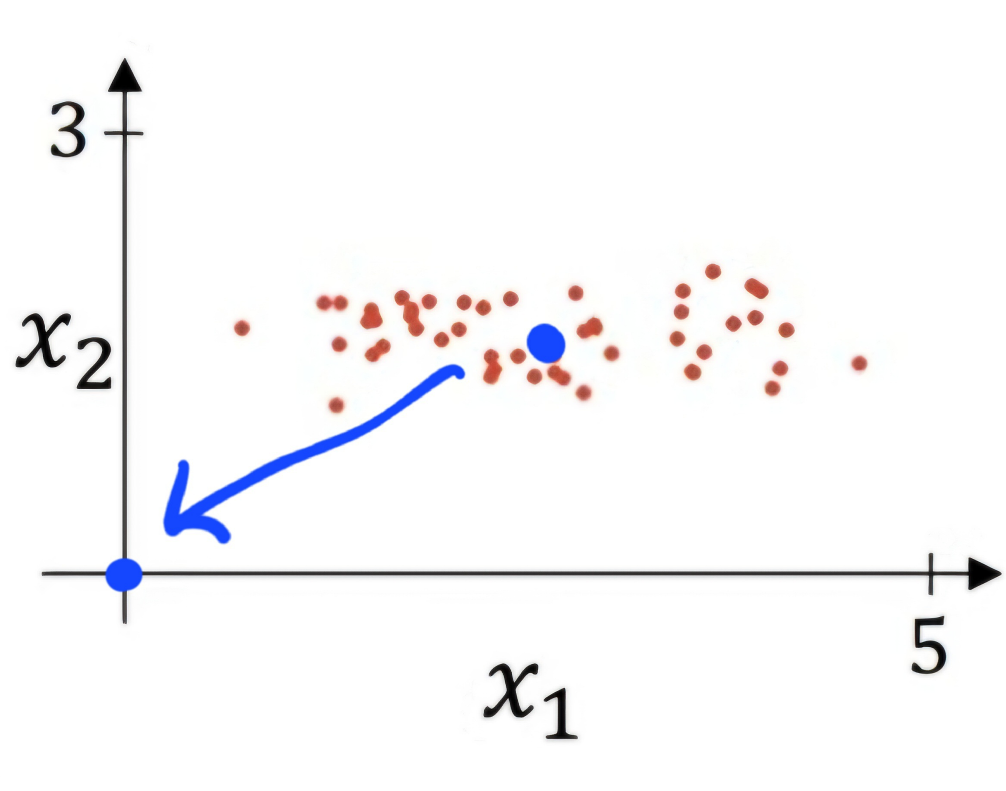
  → 학습 시 지그재그로 이동. 따라서 학습률을 작게 해야 함.

- 정규화 후 (오른쪽)

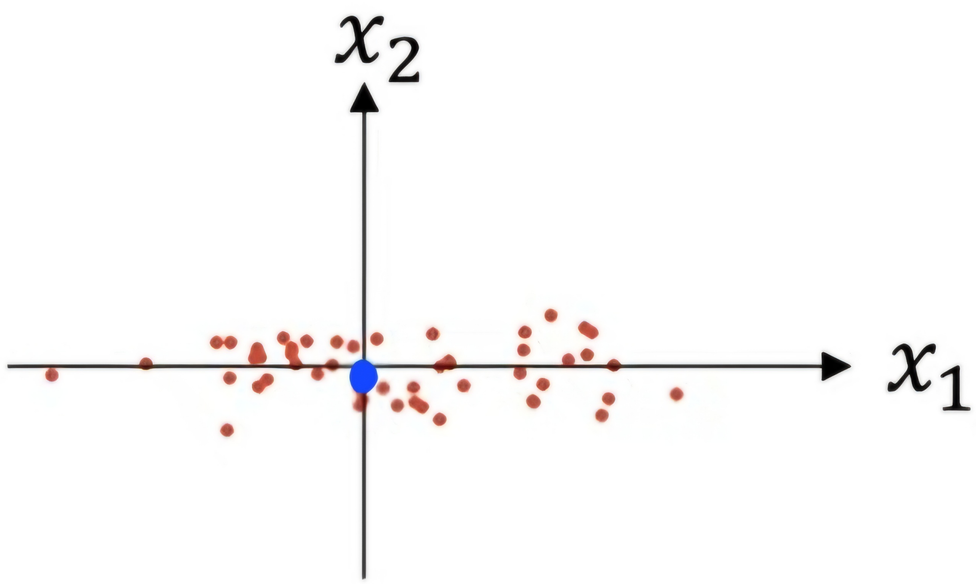
  → 학습 시 어디서 시작했든, 중심을 향해 직선으로 이동.

- 방법
  - `평균 = 0`, `분산 = 1` 만들기

    ▼ 평균 → 0

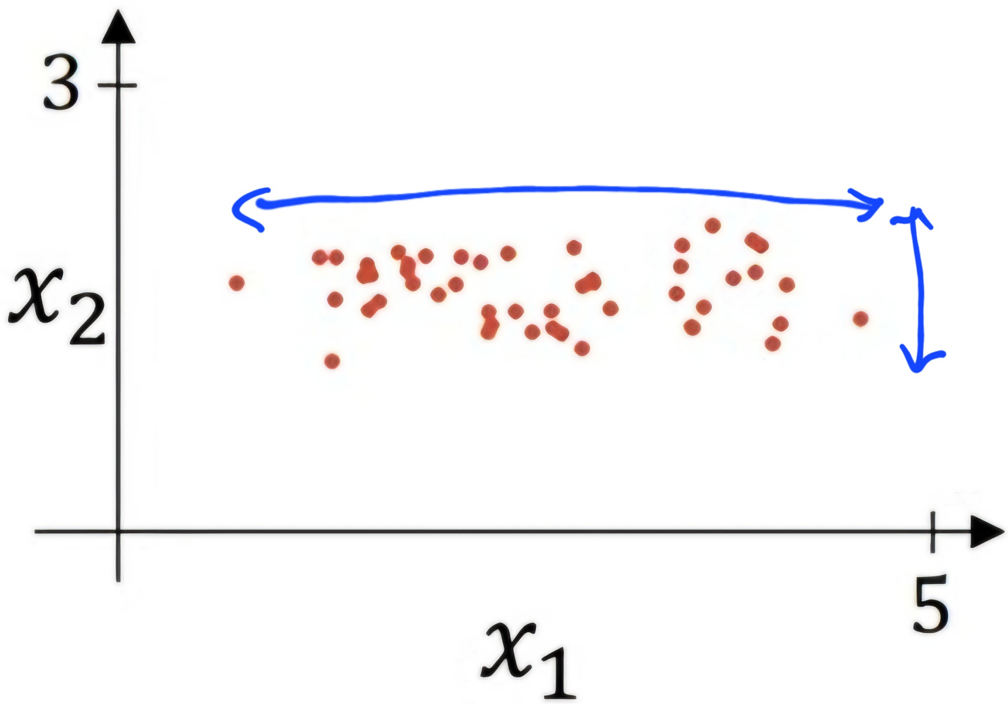


before

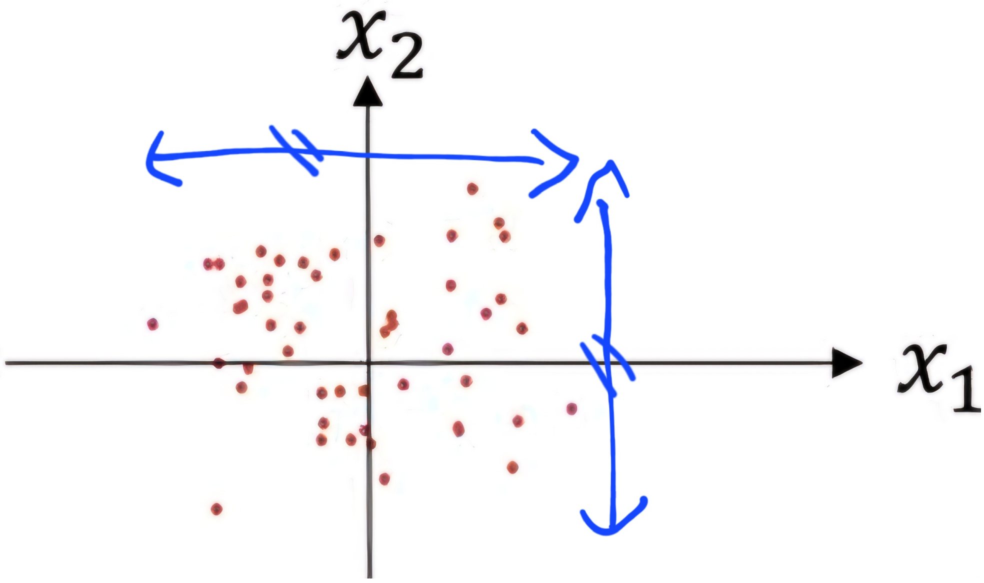


after

▼ 분산 → 1



before



after

- 공식

$$x = \frac{x - \mu}{\sigma}$$

    - **확통에서 배운 정규화 공식이랑 똑같음.**
    - 즉, input의 분포를 정규분포로 만들기

- **이때 Train set / Test set에서 같은 μ, σ값 사용**
    - 두 set에서 각각 μ, σ값을 추정하지 말고. 두 set을 다르게 정규화하고 싶진 않잖아.

경사 소실/폭발

[요약]

- **경사***gradients***의 소실***vanishing* **/ 폭발***exploding*
  - 개념
    - 신경망 깊이가 너무 깊을 때 생길 수 있는 문제.
    - 비용함수의 경사가 기하급수적으로 감수/증가하여 학습을 방해하는 현상. → 해결 시 학습을 빠르게 만듦.
  - 해결 방법 *(Aha!)*
    - 가중치를 랜덤하게 초기화할 때 → 분산을 $\frac{1}{층으로들어오는입력개수}$로 설정

- 용어
  - 경사의 소실 (vanishing gradients)
  - 경사의 폭발 (exploding gradients)

개념

- 신경망 깊이가 매우 깊을 때 생길 수 있는 문제
- 비용함수의 경사가 기하급수적으로 감수/증가하여 학습을 방해하는 현상.
  - input → $\hat{y}$을 구하는 과정에서 각 층을 거치며 $w^{[i][j]}$가 계속 곱해지게 되는데,
  - 신경망 깊이가 깊을수록 w가 너무 많이 곱해짐.
  - 그럼 z, a가 기하급수적으로 감소/증가하게 됨.

    ← w 값 중 `0.9` 이런 게 많았으면 감소,

    `1.5` 이런 게 많았으면 증가
  - 비용함수의 경사(걔네를 미분한 값)도 마찬가지로 감소/증가

해결책

- Aha!
  - 가중치를 랜덤하게 초기화할 때 → 1보다 너무 커지거나 작아지지 않게 설정한다.

- 방법
  - $n^{[l-1]}$: l층으로 들어오는 입력 개수 ← $(\because)$ $(l-1)$층 노드 개수
  - $n^{[l-1]}$이 클수록 w 값은 작게 하고 싶음.
    - ▼ $(\because)$
      - $z = w_1 x + w_2 x + ... + w_n x$
      - 우린 z값이 소실/폭발하는 일 없이, 좀 비슷하게 유지되면 좋겠음.
      - n이 크면 x가 많이 더해짐. → 각 항은 더 작아져야 함. → w_i를 작게.
  - 그걸 위해 $(w$의 분산$) = \frac{1}{n}$으로 함.

    $W^{[l]} =$ `np.random.rand(` `shape` `)` `*` `np.sqrt(` $\frac{1}{n}$ `)`

- 이때 $(w$ 의 분산 $) =$ 얼마냐 ← 이것도 하이퍼 파라미터
  - ex)

사용한 활성화 함수	$(w$의 분산$)$
ReLU	$\frac{2}{n^{[l-1]}}$
tanh	$\frac{1}{n^{[l-1]}}$ 또는 $\frac{2}{n^{[l-1]}+n^{[l]}}$

  - 아주 중요한 건 아니지만 그래도 영향 좀 있는 하이퍼 파라미터.

경사 검사

⭐ **[요약]**

- **경사 검사** *= gradient checking* *= grad check*
  - 개념
    - 역전파를 맞게 구현했는지 검사하는 방법
  - *Aha!*
    - 벡터 $d\theta$: 내가 역전파에서 구한 *gradient* *(내 답)*
    - $J'(\theta)$ $J'(\theta)J(\theta)$를 미분해서 구한 *gradient* *(정답)*
    - 둘이 같다면 → 통과!

방법

- **1. 벡터 $\theta, d\theta$ 구하기**
  - ▼ 방법
    - $W^{[1]}, b^{[1]}, ..., W^{[L]}, b^{[L]}$ → 벡터 θ로 reshape
      - W는 가중치 행렬 / b는 스칼라 값 → 일렬로 쫙 펴서 이어붙이면 벡터 θ 됨.
    - $dW^{[1]}, db^{[1]}, ..., dW^{[L]}, db^{[L]}$ → 마찬가지로 벡터 $d\theta$로 reshape
      - dW는 $\frac{dJ}{dw}$를 모아둔 행렬
  - ▼ 이때 벡터 $\theta, d\theta$는 차원이 같음.

    당연함.

    $\theta = (\theta_1, \theta_2, ..., \theta_i, ...)$ 이라고 하면

    $d\theta = (d\theta_1, d\theta_2, ..., d\theta_i, ...)$

    $= (\frac{dJ}{d\theta_1}, \frac{dJ}{d\theta_2}, ..., \frac{dJ}{d\theta_i}, ...)$

    즉, $d\theta$의 각 원소 $d\theta_i$ = 비용함수 J를 벡터 θ의 원소 θ_i로 편미분한 도함수임.

`for each` i

- **$J'(\theta)$ 구하기**
  - 비용함수 $J(W, b)$ → $J(\theta)$
  - ▼ 배경지식
    - 미분을 원래 이렇게 배웠잖아.

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

    - 근데 이렇게 계산하면

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

    - 장 : 오차가 더 작음.
    - 단 : 계산하는 데 드는 cost가 2배 ← (∵) $f(\theta + \varepsilon)$, $f(\theta - \varepsilon)$ 둘 다 계산해야 함.
    - 근데 우린 오차가 작은 게 중요해서 이 방법 쓸 거임.
  - ▼ 방법
    - $J(\theta) = J(\theta_1, \theta_2, ..., \theta_i, ...)$ 이렇게 적을 수 있음.
      - 왜냐면 $\theta = (\theta_1, \theta_2, ..., \theta_i, ...)$ 니까.

    - 이제 이걸 θ_i에 대해 편미분해보자.
      - 위에서 본 미분 방법대로 이렇게.

$$\frac{f(\theta_i + \varepsilon) - f(\theta_i - \varepsilon)}{2\varepsilon}$$

      - 정확한 식은 이거.

$$\frac{dJ}{d\theta_i} = \frac{J(\theta_1, ..., \theta_i + \varepsilon, ...) - J(\theta_1, ..., \theta_i - \varepsilon, ...)}{2\varepsilon}$$

- **if 벡터 $d\theta \approx J'(\theta)$ 확인**
  - ▼ 방법

- 둘 차이가 얼마나 작은지 보면 됨.

$$\frac{||J'(\theta) - d\theta||_2}{||J'(\theta)||_2 + ||d\theta||_2} = ?$$

- 이거 유클리디안 거리 공식임.
- 배경지식 : L2 norm 개념
- 분모 : 둘의 차
- 분자 : 근데 두 값이 원래 커서 차도 큰 거일 수도 있잖아. 그런 경우를 방지하기 위해 두 값의 크기로 나눠줌.

- $\varepsilon = 10^{-7}$일 때

저 값 $=$	해석
$< 10^{-7}$	통과! 구현 잘 됐네!
10^{-5}	주의
$10^{-3} <$	디버깅 필요

`endfor`

- 디버깅 시 → 둘 차이가 컸던 i를 주의 깊게 보기.

구현 Tip

- 이 방법은 속도가 느리니까 → Training 말고 Debug할 때만 쓰기
  - 모든 i에 대해 $J'(\theta)$를 처음부터 계산하면 너무 느리니까
    →
    $d\theta$ 계산하는 역전파 때 $J'(\theta)$도 같이 구하고, 그걸로 디버깅함.
  - 그리고 디버깅 끝나면 $J'(\theta)$ 계산하는 코드는 꺼버림. Training 때는 쓰지 마.

- 둘 차이 너무 크면 → 벡터 θ, $d\theta$의 요소*component*를 확인해보기. 힌트가 될 수 있음.

▼ 무슨 소린지 모르겠음........ 그냥 교수님 설명 그대로 옮김.

예를 들어 어떤 층에서 θ나 $d\theta$의 값이

대응되는 $db^{[l]}$과 매우 멀지만

대응되는 $dw^{[l]}$과는 매우 가까운 경우를 살펴봅시다.

→ θ의 서로 다른 컴포넌트는 b나 w의 다른 컴포넌트에 대응됩니다. 이 경우에는 db (매개변수 b에 대응하는 도함수)를 어떻게 계산하느냐에 따라서 버그가 발생할 것입니다

비슷한 방식으로 $d\theta$, $J'(\theta)$ 둘 차이가 너무 큰 경우

→ 모든 컴포넌트가 dw 혹은 특정한 층의 dw에서 온 것을 발견한다면 → 이를 통해 버그의 위치를 알아내는데 도움을 받을 수 있을 것입니다

- 정규화(Regularization) 했다면 → 잊지 말자!
  - $J(\theta)$에 정규화 항이 있음.
  - 그러면 $d\theta$에도 정규화 항 미분한 게 있겠지? 당연함. $d\theta$의 각 원소 $d\theta_i = \frac{dJ}{d\theta_i}$라니까.
  - 그것도 경사 검증 시 포함하기!

- 드롭아웃을 쓸 때는 → 이렇게!
  - 드롭아웃에서는 이 검증 방법 사용 불가
  - 왜냐면 드롭아웃은 모든 반복마다 노드를 무작위로 삭제하잖아.
    그래서
    J를 구하는 게 힘들어.
  - 그러니까 일단 드롭아웃 없이 구현하고, 경사 검증하고, 그렇게 역전파가 잘 구현된 걸 확인한 후에 드롭아웃을 켜래.

- 거의 일어나지 않지만... 가끔 무작위 초기화를 해도 초기에 경사 검사가 잘 되는 경우

  → 이때는 훈련을 조금 시킨 다음에 경사 검사를 다시 해보기