3. 최적화 문제 설정

Normalizing

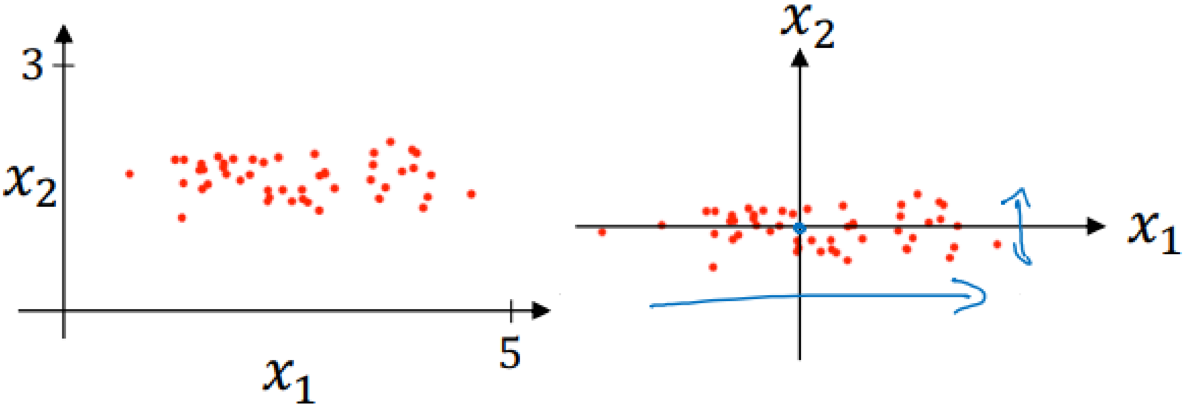
정규화 하는 방법

1) 평균 빼기

- mean이 0이 될 때까지 x에서 평균을 뺀다

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

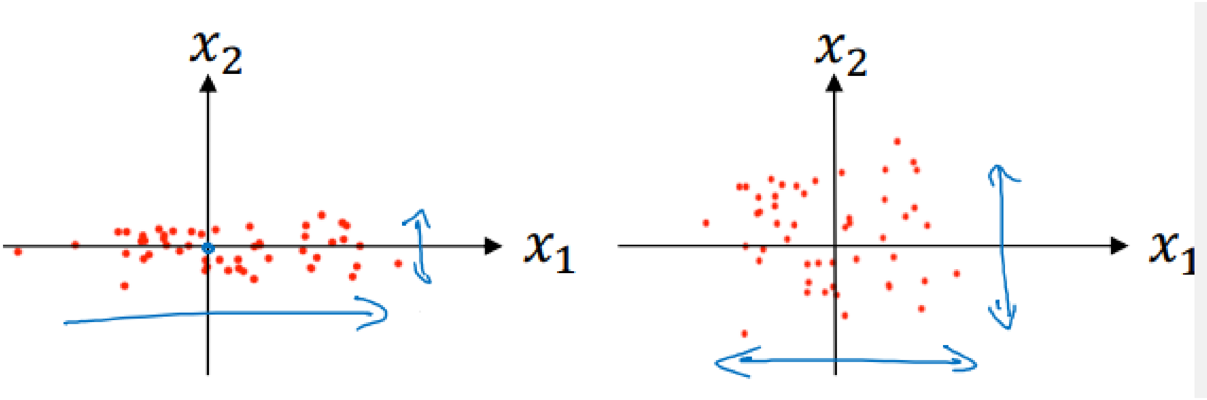
$$x := x - \mu$$



2) 분산 정규화

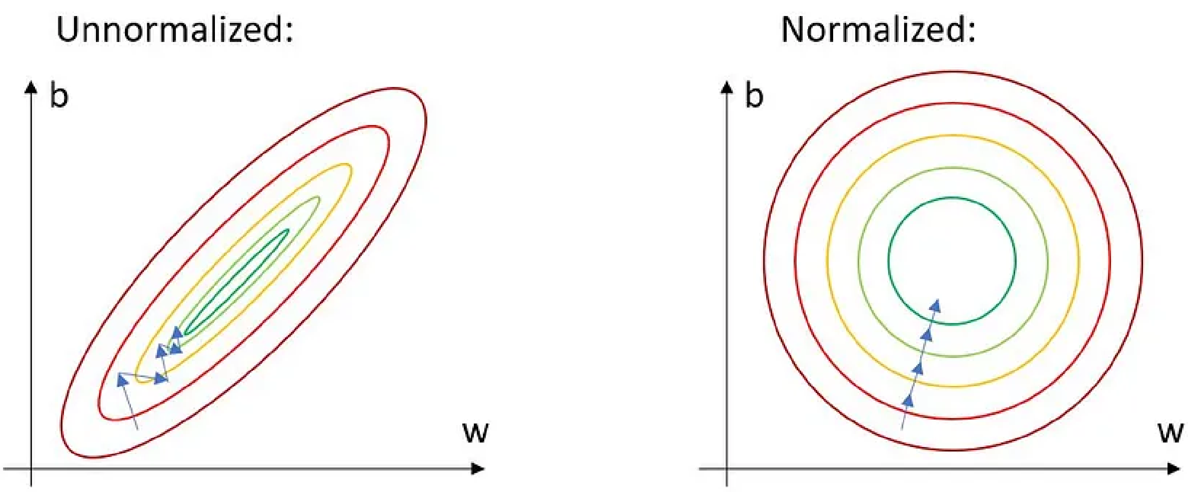
$$\sigma^2 = \frac{1}{m} \sum_{i}^{m} x^{(i)} ** 2$$

$$x \mathrel{/}= \sigma^2$$



- 주의점
  - test set 정규화할 때 같은 mean, variance를 써야함

정규화하는 이유



- 각각의 feature의 스케일이 다를 때, 예를 들어 x1은 0~1이고 x2는 10~1000일 때 w의 값 또한 스케일에 따라 차이가 커진다
- 이럴 때 정규화를 하지 않은 cost function은 learning rate의 값이 매우 작아야하기 때문에 최적화하기 어렵다
- **normalize를 통해 features scale을 통일하면 cost function이 원에 가까운 모양이 되면서 쉽고 빠르게 최적화할 수 있다**

Vanishing/Exploding Gradients

Vanishing/Exploding Gradients Problem

- 깊은 신경망을 훈련시킬 때 미분값 또는 기울기가 매우 작아지거나 커질 수 있음

$$\hat{y} = W^{[l]} W^{[l-1]} \cdots W^{[3]} W^{[2]} W^{[1]} x$$

$$z^{[1]} = W^{[1]} x$$

$$a^{[1]} = g(z^{[1]}) = z^{[1]}$$

$$a^{[2]} = g(z^{[2]}) = g(W^{[2]} a^{[1]})$$

$$a^{[3]}$$

$$\vdots$$

① $W^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \qquad \hat{y} = W^{[l]} \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{l-1} x \simeq 1.5^L x$

② $W^{[l]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \qquad \hat{y} = W^{[l]} \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}^{l-1} x = 0.5^L x$

- 신경망이 깊을수록(L이 클수록) 활성값이 기하급수적으로 증가(감소)하여 훈련이 어려워짐

Weight Initialization in a Deep Network

Vanishing/Exploding Gradients problem을 해결하기 위한 가중치 초기화 방법

1) w[l] 의 분산을 1/n으로 설정 (n: 입력 feature 개수)

2) Relu를 사용하는 경우

- var(w[i]) = 2/n

```
w[l] = np.random.rand(shape) * np.sqrt(2/n[l-1])
```

3) tanh을 사용하는 경우

- var(w[i]) = 1/n (Xavier initialization)

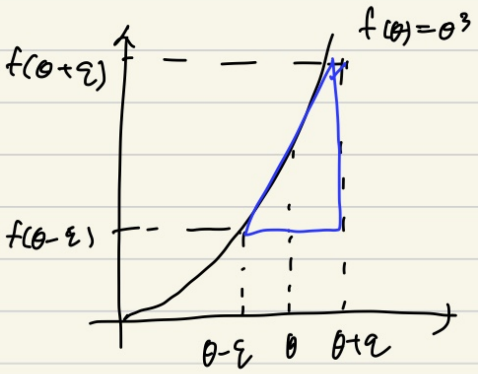
```
w[l] = np.random.rand(shape) * np.sqrt(1/n[l-1])
```

4) other

```
w[l] = np.random.rand(shape) * np.sqrt(2/n[l-1]+n[l])
```

Gradient Checking

Numerical Approximation of Gradients



$(\varepsilon = 0.01)$

$$\frac{f(\theta+\varepsilon) - f(\theta-\varepsilon)}{2\varepsilon} \approx g(\theta)$$

$$\begin{cases} \dfrac{(1.01)^3 - (0.99)^3}{2(0.01)} = 3.0001 \\ g(\theta) = 3\theta^2 = 3 \end{cases}$$

approx error : 0.0001

- 양쪽의 차이를 이용하여 기울기를 계산하는 것이 한쪽만 이용하는 것보다 정확도 높음

Gradient Checking 구현

- 모든 w, b를 하나의 벡터 θ로 concatenate한다.

$$J(W^{[1]}, b^{[1]} \cdots) = J(\theta)$$

- 기울기 계산

for each i :

$$d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2, \cdots, \theta_i + \varepsilon, \cdots) - J(\theta_1, \theta_2, \cdots, \theta_i - \varepsilon, \cdots)}{2\varepsilon}$$

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$$

- 두 벡터의 유클리드 거리를 계산하여 gradient check

두 벡터의 유클리드 거리 계산

$$\frac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2} \approx 10^{-7}$$

$$(\varepsilon = 10^{-7})$$

계산한 결과가 ε과 가까울수록 좋음.

Tips for Gradient Checking

1) 트레이닝 과정에서는 사용하면 안 된다.

- 계산이 매우 느리기 때문에 debug할 때만 사용해야함

2) 경사 검사 알고리즘 실패 시 개별적인 컴포넌트를 확인해본다.

- 어떤 i에서 문제가 발생하는지 확인한다

3) regularization 고려하기

4) drop out과 함께 사용하지 않는다.

- keep_prop=1로 설정해 drop out을 적용하지 않은 상태에서 경사 검사를 먼저 실시한 후 drop_out을 적용한다,

5) 초기에 경사 검사가 잘되는 경우 → 훈련을 조금 진행해 w, b가 0에서 멀어지게 한 후 다시 검사를 진행한다