3. 최적화 문제 설정

날짜	@2023년 11월 6일

▼ 목차

입력값의 정규화

1️⃣ 정규화 방법

1. 평균을 0으로 만들기

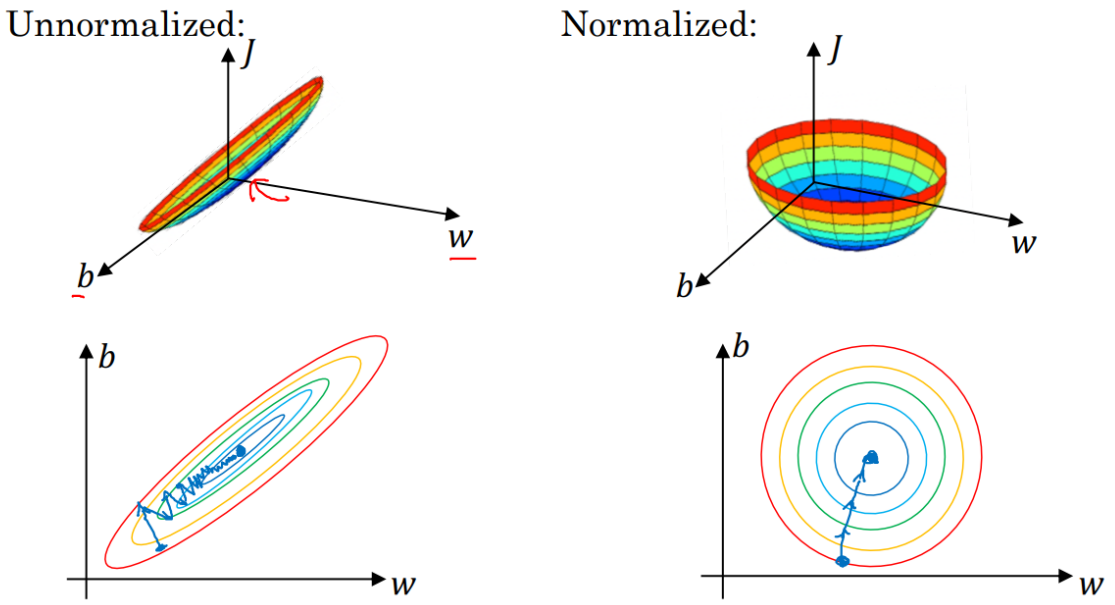
$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$
$$x := x - \mu$$

2. 분산을 1로 만들기

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

- 테스트 세트 정규화 시 훈련 데이터에 사용한 μ, σ를 사용해야 함

2️⃣ 정규화를 해야 하는 이유?



- 정규화를 통해 비용함수의 모양은 더 둥글고 최적화하기 쉬운 모습이 됨
  - 따라서 학습 알고리즘을 빨리 실행시킬 수 있게 됨

경사소실/경사폭발

1️⃣ 경사소실과 경사폭발

- 매우 깊은 신경망을 훈련시킬 때 경사의 소실과 폭발 문제가 발생함
- $g(z) = z, b^{[l]} = 0$이라고 가정했을 때 $\hat{y} = w^{[l]}w^{[l-1]} \cdots w^{[2]}w^{[1]}x$가 됨
  - 이때 모든 가중치 행렬 $w = 1.5E$(E는 단위행렬)라고 가정하면, $\hat{y} = 1.5^{(l-1)}Ex$가 되고 더 깊은 신경망일수록 $\hat{y}$의 값은 기하급수적으로 커짐
  - 반대로 모든 가중치 행렬 $w = 0.5E$라고 가정하면 $\hat{y} = 0.5^{(l-1)}Ex$가 되고 더 깊은 신경망일수록 $\hat{y}$의 값은 기하급수적으로 감소함

경사하강법에서 w의 값이 단위행렬보다 크다면 경사의 폭발, w의 값이 단위행렬보다 작다면 경사의 소실 문제가 발생함을 알 수 있음

- 경사의 소실과 폭발 발생 시 학습시키는 데 많은 시간이 걸리므로, 가중치의 초깃값을 신중하게 결정하는 것이 중요함

심층 신경망의 가중치 초기화

1️⃣ 가중치 초기화 방법

1. w_i의 분산을 $\frac{1}{n}$으로 설정함 (n: 입력 특성의 개수)
2. ReLU 활성화 함수 사용 시 w_i의 분산을 $\frac{2}{n^{[l-1]}}$로 설정
3. tanh 활성화 함수 사용 시 w_i의 분산을 $\frac{1}{n^{[l-1]}}$ 또는 $\frac{2}{n^{[l-1]}+n^{[l]}}$로 설정

기울기의 수치 근사

- 경사 검사를 함으로써 역전파를 알맞게 구현했는지 확인할 수 있음

1️⃣ 중심차분법

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

- 이때, ϵ은 굉장히 작은 수
- 이 수치 미분은 중심차분법보다 오차가 더 높음

경사 검사

- 우선 모델 안의 모든 변수(W, b)를 하나의 벡터(θ)로 concatenate함
- 비용함수는 $J(W, b)$에서 $J(\theta)$로 변하며, 이에 대한 수치미분을 계산함
  - $d\theta^{[i]}_{approx} = \frac{J(\theta_1, \cdots, \theta_i+\epsilon, \cdots) - J(\theta_1, \cdots, \theta_i-\epsilon, \cdots)}{2\epsilon}$
- 최종적으로 수치미분과 일반미분을 비교함
  - $d\theta^{[i]}_{approx} \approx d\theta$
  - 유사도 계산 시 유클리디안 거리 사용
    - $\frac{\left\| d\theta^{[i]}_{approx} - d\theta \right\|_2}{\left\| d\theta^{[i]}_{approx} \right\|_2 + \left\| d\theta \right\|_2}$
  - 일반적으로 거리가 10^{-7}보다 작으면 잘 계산되었다고 판단함

경사 검사 시 주의할 점

- 속도가 굉장히 느리므로 훈련 시에는 사용하지 않으며, 디버깅 시에만 사용
- 알고리즘이 경사 검사에 실패한 경우 어느 원소 부분에서 실패했는지 찾아봐야 함
    - 특정 부분에서 계속 실패한 경우, 그 경사가 계산된 층에서 문제가 발생했음을 확인 가능
- $d\theta$는 θ에 대응하는 J의 정규화 항($\frac{d}{dm}\sum_m \left\|w^{[l]}\right\|_F^2$)도 포함하므로 경사 검사 계산 시 포함해야 함
- 드롭아웃에서는 무작위로 노드를 삭제하므로 경사 검사를 적용하기 어려움
    - 따라서 통상적으로는 드롭아웃을 끄고 알고리즘이 최소한 드롭아웃 없이 맞는지 확인한 후 드롭아웃을 켜는 방식을 활용함
- 간혹 무작위 초기화를 해도 경사 검사가 잘 되는 경우가 있을 수 있음
    - 이때는 훈련을 조금만 시킨 후 경사 검사를 다시 해 봐야 함

출석퀴즈 오답노트

▼ 1. 데이터 정규화에 대해 맞는 설명 모두 고르기

- 정규화를 통해 비용함수의 모양은 더 둥글고 대칭에 가까워진다
- 정규화는 학습을 빠르고 안정적으로 수행할 수 있도록 돕는다
- 훈련 시 경사 폭발과 같은 문제가 발생할 수 있다 (O)

▼ 4. 경사 검사에 대해 맞는 설명 모두 고르기

- 학습 시에 역전파가 알맞게 구현되었는지 확인하기 위해 수행된다 (X)
- 파라미터 주변값을 활용하여 기울기를 추정한다
- 경사 검사를 구현 한다면 비용 함수는 J(W, b) 에서 J(θ)로 변한다

▼ 8. 경사 검사를 수행할 때 코드 빈칸에 들어갈 수 작성

In [6]:

```
# GRADED FUNCTION: gradient_check

def gradient_check(x, theta, epsilon=[redacted]):
    """
    Implement the backward propagation presented in Figure 1.

    Arguments:
    x -- a real-valued input
    theta -- our parameter, a real number as well
    epsilon -- tiny shift to the input to compute approximated gradient with formula(1)

    Returns:
    difference -- difference (2) between the approximated gradient and the backward propagation gradient
    """

    # Compute gradapprox using left side of formula (1). epsilon is small enough, you don't need to worry about the limit.
    thetaplus = theta + epsilon
    thetaminus = theta - epsilon
    J_plus = forward_propagation(x, thetaplus)
    J_minus = forward_propagation(x, thetaminus)
    gradaporox = (J_plus - J_minus) / (2 * epsilon)

    # Check if gradapprox is close enough to the output of backward_propagation()
    grad = backward_propagation(x, theta)
    numerator = np.linalg.norm(grad - gradapprox)
    denominator = np.linalg.norm(grad) + np.linalg.norm(gradapprox)
    difference = numerator / denominator

    if difference < epsilon:
        print("The gradient is correct!")
    else:
        print("The gradient is wrong!")

    return difference
```

In [7]:

```
x, theta = 2, 4
difference = gradient_check(x, theta)
print("difference = " + str(difference))
```

답: `1e-7`