[딥러닝 2단계] 2. 신경망 네트워크의 정규화

1. 정규화

높은 분산 문제를 해결하기 위한 방법

1. 정규화
   과대적합을 막고 신경망의 분산을 줄임
2. 더 많은 훈련 데이터 얻기
   비용이 많이 들어감

Logistic regression (로지스틱 회귀)

- **로지스틱 회귀**: 비용함수 J 를 최소화
  = $min_{w,b}J(w,b)$
- 비용함수 $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\left\|w\right\|_2^2$
- λ: 정규화 매개변수
  - 개발세트 혹은 교차 검증 세트를 주로 사용
  - 다양한 값을 시도해 훈련 세트에 잘 맞으면서 두 매개변수의 노름을 잘 설정해 과대적합을 막을 수 있는 값을 찾음
  - λ는 설정이 필요한 또 다른 하이퍼 파라미터

1) L2 정규화

$\|w\|\,2^2 = \sum j = 1^{n_x} w_j^2 = w^T w$

- 가장 일반적인 정규화
- **왜 매개변수 w만 정규화할까?**
  - b도 정규화 가능하지만 보통은 생략
  - 매개변수 w는 높은 차원의 매개변수 벡터 (특히 높은 분산 가질때)이지만 b는 하나의 숫자

- 많은 매개변수 중 b는 오직 하나의 매개변수이기 때문에 실질적인 차이가 생기지 않음

2) L1 정규화

$\frac{\lambda}{2m}\sum_{j=1}^{n_x}|w| = \frac{\lambda}{2m}\|w\|_1$

- m 앞에 곱하는 2는 스케일링 상수
- w는 희소해지는데 이는 w 벡터 안에 0이 많아진다는 의미
  - 모델 압축에 도움
  - 특정 매개변수가 0일 경우 메모리가 적게 필요하기 때문에
  - 그러나 모델을 희소하게 만들기 위해 L1 정규화를 사용하는 것은 큰 도움이 되지 않음
  - 그래서 모델을 압축하겠다는 목표가 있지 않은 이상 L1 정규화를 많이 사용하지 않음
- 네트워크를 훈련할 때는 L2 정규화를 훨씬 많이 사용

Neural Network

- 신경망의 비용함수
  - 모든 파라미터 w^[1], b^[1]부터 w^[L], b^[L]까지의 매개변수를 갖는 함수
  - 훈련샘플의 m까지의 손실의 합을 m으로 나눈 값 + 정규화항

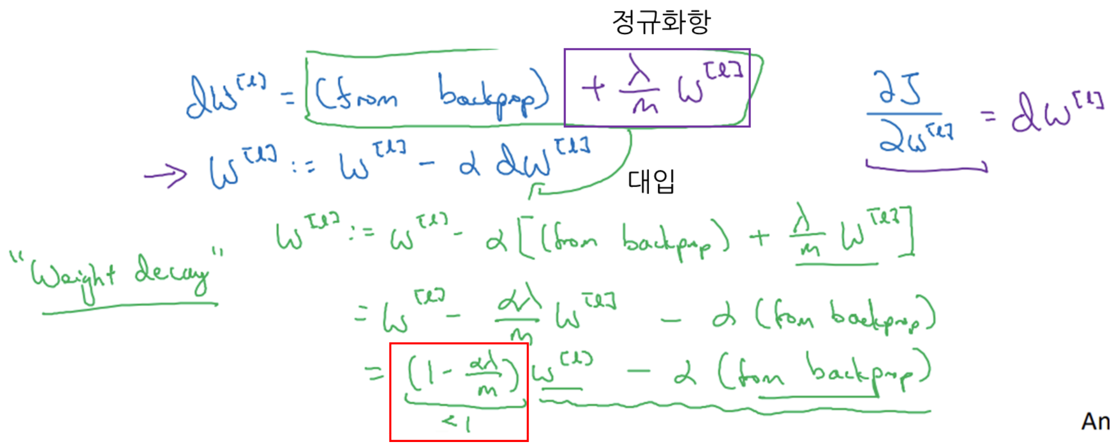
> $J(w^{[1]}, b^{[1]}, ..., w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\left\|w^{[l]}\right\|_F^2$

- $\|w\|F^2 = \sum i = 1^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} (w_{ij}^{[l]})^2$
- 이 행렬의 norm을 **프로베니우스 노름**이라고 부름
  - i와 j에 해당하는 행렬의 원소 제곱의 합
    합의 범위: i는 $n^{[l]}$, j는 $n^{[l-1]}$
  - 해당 층 L-1과 L의 은닉 유닛의 개수
    -> w 행렬의 차원

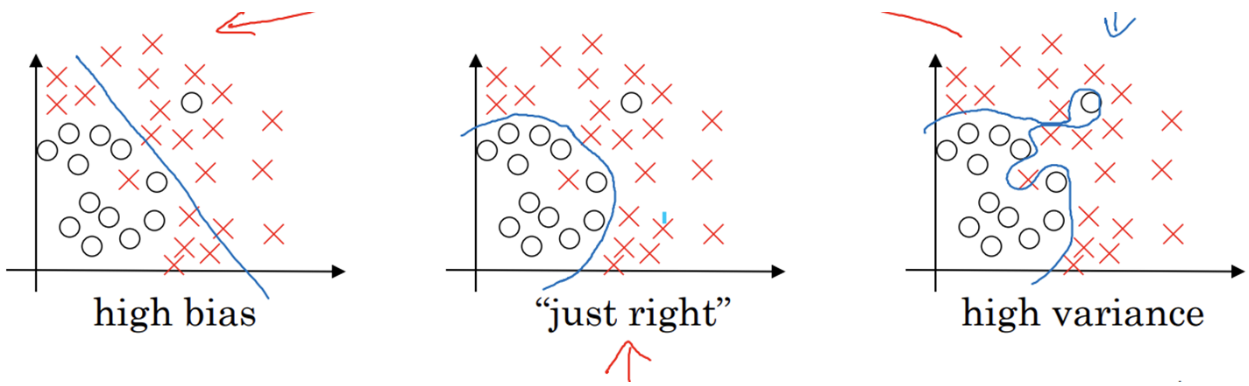
경사하강법

- 정규화 항 $\frac{\lambda}{m}w^{([l])}$ 추가

- L2 정규화는 weight decay라고도 불림
  - 가중치 행렬 $w^{[l]}$에 1보다 작은 값을 곱해주므로 어떤 값이든 값이 약간 더 작아짐



2. 왜 정규화는 과대적합을 줄일 수 있을까?



- 과대적합 문제가 있는 신경망
  w,b에 대한 비용함수 J는
  $J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \left\| w^{[l]} \right\|_F^2$
- 정규화를 위해 추가정인 항 추가해줌
- 가중치 행렬이 너무 커지지 않도록
  프로베니우스 노름을 해줌

1) L2, 프로베니우스 노름을 줄이는 것이 왜 과대적합을 줄일 수 있을까?

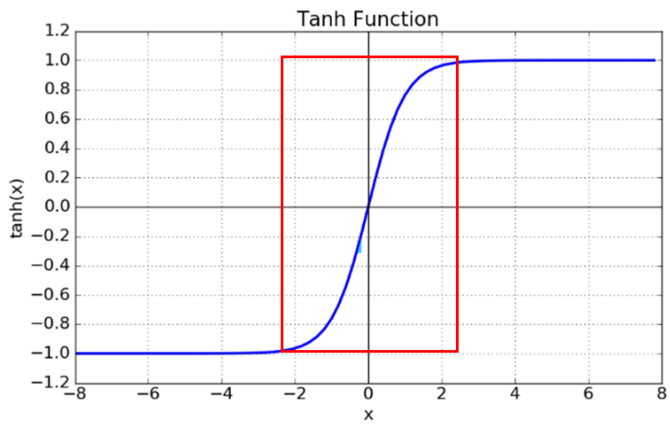
- λ를 크게 만들어서 가중치 행렬 w를 0에 상당히 가깝게 만들 수 있음

- 많은 은닉 유닛을 0에 가까운 값으로 설정해 은닉 유닛 영향력 줄어듬
  - > 훨씬 간단하고 작은 신경망이 됨
  - 로지스틱 회귀 유닛에 가까워짐
- > 높은 편향의 경우와 가깝게 만들어줌
  - 중간의 경우와 가깝게 하는 람다 값을 찾는게 좋음

> λ 크게 하면 w는 0에 가깝게 설정되고 은닉 유닛의 영향력을 0에 가깝게 줄임으로써 로지스틱 회귀에 가까운 네트워크를 만든다

- 실제로는 모든 은닉유닛을 사용하지만 각각의 영향력이 훨씬 작아진 것
  - > 간단한 네트워크 된 것은 맞음

2) tanh 활성화 함수 사용한다고 가정



z가 작은 범위에서 tanh 함수는 거의 선형

- z가 작은 범위의 매개변수를 갖는 영역이면 tanh 함수의 선형 영역 사용
  - z가 더 작아지거나 커지면 활성화 함수는 선형을 벗어남
- λ가 커질 때 비용함수가 커지지 않으려면 상대적으로 w가 작아질 것임
  - w가 작으면 z도 상대적으로 작은 값
  - g(z)는 거의 1차원 함수
- 따라서 모든 층은 선형 회귀처럼 거의 직선의 함수를 갖게 됨

    - > 모든 층이 선형이면 전체 네트워크도 선형
- 깊은 네트워크여도 선형 함수만을 계산하여 비선형 결정의 경계 맞추기 불가능
    - = 과대적합된 데이터 세트 맞추기 어렵

> 정규화 매개변수가 매우 크면 매개변수 w는 매우 작음
> -> b 무시하면 z값은 상대적으로 작음
> -> z가 상대적으로 작으면 tanh 경우 활성화 함수 선형
> -> 전체신경망은 선형 함수로부터 그리 멀지 않은 곳에서 계산
> -> 매우 복잡한 비선형 함수보다 간단
> -> 과대적합 가능성 낮아짐

구현 팁

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

- 가중치 너무 커지는 것 막기 위해 추가 항 사용
- 경사하강법에서 반복의 수에 대한 함수로 비용함수 J 정의
    - J가 단조감소하기 원함
- J 첫 항만을 그리면 단조감소 x, 두번째 항까지 포함해야 J가 단조감소
    - 그러므로 경사하강법을 디버깅 할 때, 두번째 항을 포함한 새로운 비용함수 J를 그리고 있는지 확인해야함

3. 드롭아웃 정규화

- L2 외의 강력한 정규화 기법

Dropout regularization

- **드롭아웃 정규화** : 신경망의 각각의 층에 노드를 삭제하는 확률을 설정하는 것
- 각 노드마다 0.5확률로 노드 유지, 삭제

- 삭제된 노드의 들어가는 링크, 나가는 링크 모두 삭제
- 더 작고 간소화된 네트워크
- 이걸로 하나의 샘플을 훈련하고 다른 샘플에 대해서도 노드 삭제
- 각 훈련 샘플에 대해 감소된 네트워크로 훈련
- > 각각의 샘플에서 더 작은 네트워크를 훈련시키는 방식 -> 네트워크 정규화 가능

Implementing dropout

ex) 층이 3인 예시
단일 층에서 드롭아웃 구현

- 층 3에 대한 드롭아웃 벡터 d3 설정
  `d3=np.random.rand(a3.shape[0],a3.shape[1])<keep_prob`
  - a3와 같은 모양, keep_prob과 비교 여기서는 0.8로 설정
  - 이수는 은닉 유닛이 유지될 확률
  - 어떤 은닉 유닛 삭제 확률 = 0.2
  - 이 코드는 무작위 행렬 생성
- d3는 각 샘플과 은닉유닛에 대해 0.8의 확률로 d3가 1의 값을 가지고, 0.2의 확률로 0의 값을 가짐

  `a3 = np.multiply(a3,d3) #a3*=d3`
- 3번째 층의 활성화 a3는 예전 a3 값에 요소별 곱셈으로 d3를 곱해준것
- d3의 원소를 곱해 대응되는 a3의 원소를 20퍼 확률로 0으로 만듦

  `a3 /= keep_prob`
- 최종적으로 얻은 a3를 0.8로 나눠줌 (keep_prob) 매개변수로
- if) 세번째 은닉층에 50개의 유닛이 있다고 가정
  - 세번째 은닉층 50개 유닛(뉴런) a3는 (50,m) 차원
    -> 평균적으로 10개가 0의 값을 가지게 됨
    -> a^[3] 20퍼센트 줄어듦
  - z^[4] 줄이지 않기 위해 /=0.8 20퍼센트 정도 값을 다시 원래대로
  - 역드롭아웃 : keep_prob을 다시 나눠줌으로써 a3의 기대값을 같게 유지

- 역 드롭아웃이 테스트 쉽게 만들어줌
    - 스케일링 문제가 적게 때문
- d 벡터로 서로 다른 훈련 샘플마다 다른 은닉 유닛들을 0으로 만들게 됨
- 여러 번 반복 0이 되는 은닉 유닛 무작위로 달라짐
    - 하나의 샘플에서 계속 같은 은닉 유닛 0 x
    - 경사 하강법 1번 반복마다 달라짐
      -> 같은 훈련 세트 2번째 반복할 때는 0이 되는 은닉 유닛 패턴 달라짐

Making predictions at test time

훈련 후 테스트 시간

$$a^{[0]} = X$$

No drop out.

$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = \dots$$
$$\downarrow$$
$$\hat{y}$$

X=예측하고 싶은 샘플=a^[0]

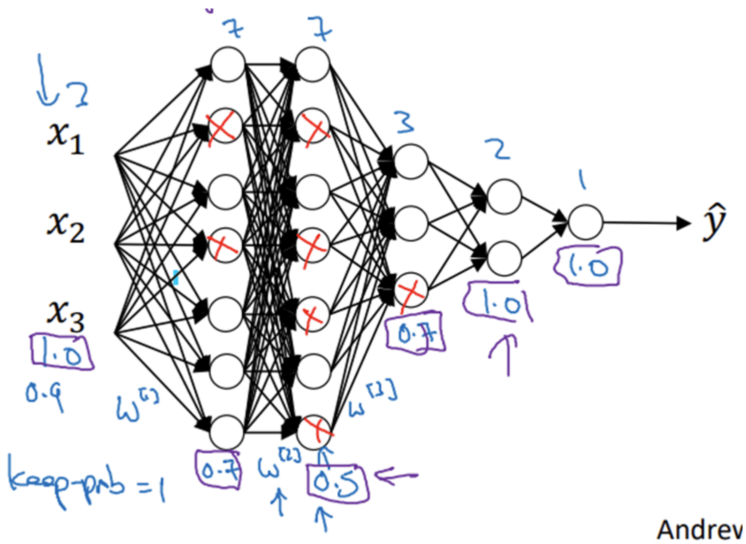
- 테스트에서는 드롭아웃 사용 x
    - 은닉 유닛 삭제할지 무작위로 정하지 x
    - 테스트에서는 예측을 하는 것이므로 결과가 무작위로 나오는것 원하지 x
    - 테스트에서는 노이즈만 증가시킬뿐
    - 이론적으로) 드롭아웃 서로 다른 은닉 유닛을 예측과정에서 여러번 반복해 평균 낼 수 있으나, 비효율적이고 비슷한 결과를 냄
- 마지막 층에 이르면 y 예측값 얻음

- 역드롭아웃
  - 테스트에서 드롭아웃 구현하지 않아도 활성화 기대값 크기가 변하지 않음
  - 테스트 할 때 스케일링 매개변수를 추가해주지 않아도 됨

4. 드롭아웃의 이해

왜 이것이 잘 작동할까?

Why does drop-out work?



1. 모든 반복마다 더 작은 신경망 사용
2. 단일 유닛의 관점

- 유닛의 일 : 입력을 받아 의미 있는 출력 생성
- 드롭아웃을 통해 입력 무작위 삭제 가능
  - 보라색 유닛 : 어떤 특성에도 의존 불가
  - 즉, 특정 입력에 유난히 큰 가중치 부여하지 않음
  - 4개 입력 각각에 가중치 분산
  - > 가중치 노름의 제곱값 줄어듦
  - L2처럼 가중치 줄이고 과대적합 막는데 도움됨
  - l2 의 적응형으로 보여지기도 함

        - 다른 가중치는 다르게 취급(다른 가중치에 적용..?), 가중치에 곱해지는 활성화의 크기에 따라 다름, 서로 다른 크기 입력에 더 잘 적응
    - 드롭아웃, L2 정규화 비슷한 효과
- keep_prob 층마다 바꾸는 것도 가능
    1. 과대적합의 우려가 적은 층 -> 더 높은 keep_prob 설정해도 괜찮음
    2. 과대 적합 우려 없는 층 -> 1도 괜찮음
        - 1: 모든 유닛 유지하고 해당 층에서는 드롭아웃 사용 x
    3. 과대 적합 우려 높은 층 -> 강력한 드롭아웃 위해 keep_prob 낮게 설정
        - L2에서 다른 층보다 더 많은 정규화가 필요한 층에서 매개변수 람다를 증가시키는 것과 비슷
        - 단점 : 교차 검증을 위해 더 많은 하이퍼파라미터가 생김
        - 대안 : 어떤 층은 드롭아웃하고, 어떤 층은 적용하지 않아서 매개변수를 드롭아웃을 적용한 층에 대한 keep_prob 하나만 갖는 것
- 이론적으로 입력층에서도 드롭아웃 가능
    - 하지만 1이 가장 흔한 값
    - 입력 특성 절반 이상 날리는게 별로이기 때문

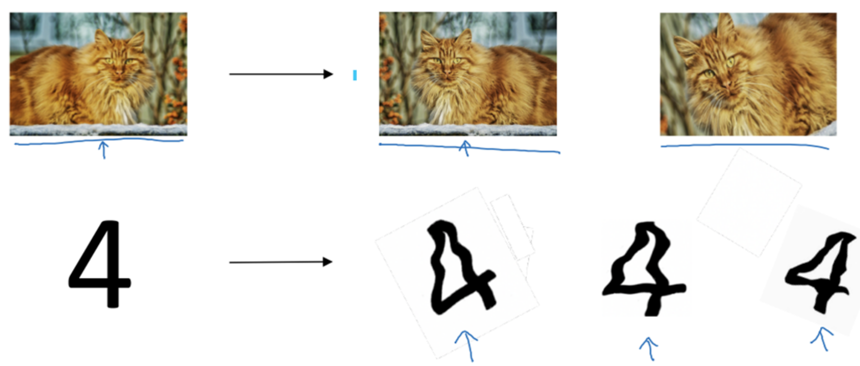
몇가지 구현팁

- 컴퓨터 비전 드롭아웃 구현 최초 성공 사례들 많이 나옴
    - 원인: 아주 많은 픽셀 값 사용하여 대부분의 경우 데이터 부족
    - > 컴퓨터 비전에서 드롭아웃 매우 빈번하게 사용
    - 기본값: 드롭아웃
- 기억할 점
    - 정규화 기법, 과대적합 해결 방법
      -> 과대적합 문제 생기기 전까지는 드롭아웃 사용 x
    - 컴퓨터 비전은 충분한 데이터 x -> 과대적합 많음 -> 드롭아웃 사용
    - 그러나, 다른 분야에도 일반화는 x

- **드롭아웃의 큰 단점** : 비용함수 J 잘 정의되지 x
    - 모든 반복마다 한 뭉치의 노드 삭제 -> 경사 하강법의 성능을 이중으로 확인한다면 모든 반복에서 잘 정의된 비용함수 J가 하강하는지 확인이 어렵
    - 디버깅 어렵
    - keep_prob을 1로 설정해 드롭아웃 효과 멈추고 코드를 실행시켜 J가 단조 감소하는지 확인
    - 드롭 아웃이 있을 때 코드 변경 x - 코드/경사하강법 잘 작동하는지 함수보고 확인하는 것 외에 다른 방법 필요하기 때문

5. 다른 정규화 방법들

1. Data augmentation



1) 고양이 분류기 훈련

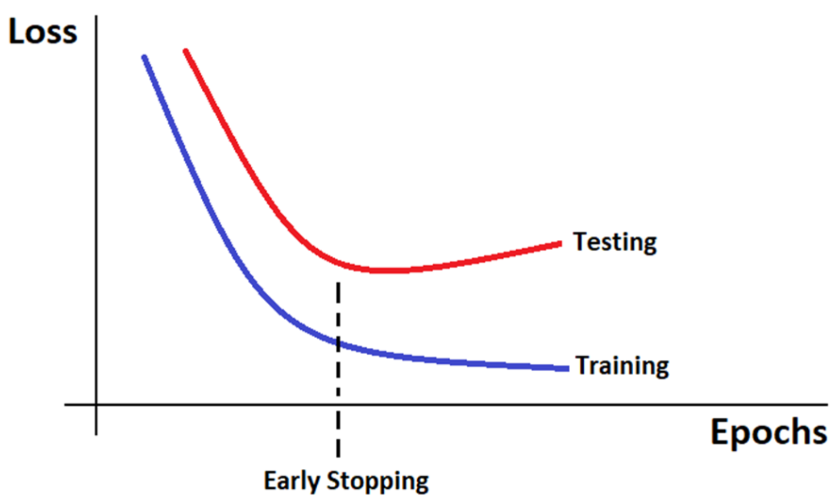
- 더 많은 훈련 데이터가 과대적합 해결 -> 높은 비용, 불가능
- but, 수평방향으로 뒤집은 이미지 훈련 세트에 추가시켜 훈련 세트를 2배로 늘림
- 새로운 m개의 독립적인 샘플을 얻는 것보다 중복된 샘플이 많아져 좋지 않음
    - 하지만 새로운 고양이 사진 구하지 않고 할 수 있는 방법
    - 고양이 이미지는 뒤집어도 고양이
- 무작위로 이미지를 편집해 새로운 샘플 얻기
    - 이미지의 무작위적인 왜곡, 변형
- 새로운 샘플을 얻는 것보다 더 많은 정보를 추가하는 x

- 그러나 컴퓨터적인 비용이 들지 않고 할 수 있다는 장점 있음
    - 비싸지 x
- > 정규화하여 과대적합 줄이기 가능

2) 시각적인 문자 인식

- 숫자 -> 회전/왜곡 훈련 세트 추가 여전히 숫자 4
- data augmentation은 정규화 기법과 비슷하게 사용가능

2. Early stopping(조기 종료)



- 훈련 오차를 그리거나 비용함수 J가 단조 감소하는 형태로 그려져야함
- 조기 종료에서는 개발 세트 오차도 함께 그려줌
    - 개발 세트 분류 오차, 로지스틱 손실 함수라고 볼 수 있음
    - 개발 세트 오차가 아래로 내려가다가 다시 증가
    - 신경망이 중간부분에서 가장 잘 작동하는 것을 알 수 있음 -> 신경망 훈련 멈추고 중간 부분의 개발 세트 오차 만든 값을 최적으로 삼음
- 왜 이 방법 작동?
    - 신경망에서 많은 반복을 실행시키지 않은 경우 매개변수 w는 0에 가까움 (무작위의 작은 값으로 초기화했기 때문)

- 반복할수록 w 점점 커짐
- 반복 중간에 멈추면 w가 중간 크기의 값을 갖는 상태
- L2 정규화와 비슷하게 매개변수 w에 대해 더 작은 노름을 갖는 신경망 선택해서 신경망이 덜 과대적합
- 신경망을 조기로 종료

- **조기종료 단점**:
  - 머신러닝 과정은 서로 다른 몇가지 단계로 이루어짐
  - 비용함수 J를 최적화하는 알고리즘을 원함
    graident,...
  - J를 최적화한 후 과대적합을 막기 위한 몇가지 도구들
    정규화, 데이터 더 추가하기
  - 아주 많은 하이퍼파라미터 : 알고리즘 선택 복잡
    - 비용함수 J 최적화할 때 집중하는 것은 w,b 찾기
    - 과대적합 막는 것은 완전히 다른 일 - 분산 줄이기
    - > orthogonalization(직교화): 한번에 하나의 할 일만을 생각하는 것
- 그런데 조기 종료는 이 둘을 섞어버림
  - 이 두 문제에 대해 독립적 작업 불가
  - 왜냐하면, 경사 하강법을 일찍 멈춰 비용함수 J 최적화를 멈춤 (J 줄이기 잘 못과 동시에 과대적합을 막으려고 함
  - > 두 문제를 해결하기 위해 혼합된 하나의 도구를 사용
    -> 문제 더 복잡해짐
- **조기종료 대안**: L2 정규화 사용
  - 오래 신경망 훈련 가능 -> 하이퍼 파라미터의 탐색 공간이 더 분해하기 쉽고 찾기 쉬워짐
  - 단점: 정규화 매개변수 람다에 많은 값을 시도해야 하는데 람다에 많은 값 대입은 컴퓨터적 비용이 많이 듬
- **조기 종료 장점**: 경사 하강법 1번만 실행해서 작은 W, 중간 W, 큰 W 값을 얻게 됨, 많은 값을 시도할 필요 없이 비슷한 효과 가능

해당글은 부스트코스의 [딥러닝 2단계] 2. 심층 신경망 성능 향상시키기 강의를 듣고 작성한 글입니다.