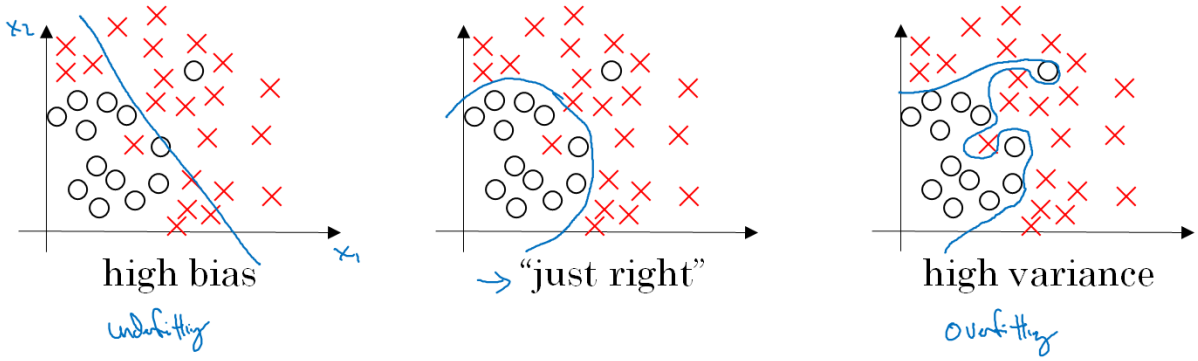
[1주차] Practical aspects of Deep Learning

요즘 추세는 개발셋(dev set)과 테스트셋은 전체의 매우 작은 퍼센트로 합니다. 데브셋 또는 개발셋의 목적은 다른 알고리즘들을 시험해 보고 어떤게 더 좋은지 알아내려는 것이니까, 개발셋은 단지 평가하기에 충분히 많으면 됩니다. 2개의 다른 알고리즘에서 선택하거나 열개의 알고리즘에서 선택하거나 할 때 어떤 것이 더 좋은지 빨리 결정하는데 전체 데이터의 20%나 필요하지는 않습니다.

백만 개의 예시가 있는 경우, dev를 위한 만개의 예시, 테스트를 위한 만개의 예시이면 충분합니다. 이 경우, 비율은 1퍼센트가 되기 때문에, 98퍼센트 트레이닝, 1퍼센트 dev, 나머지 1퍼센트가 테스트가 되겠습니다.

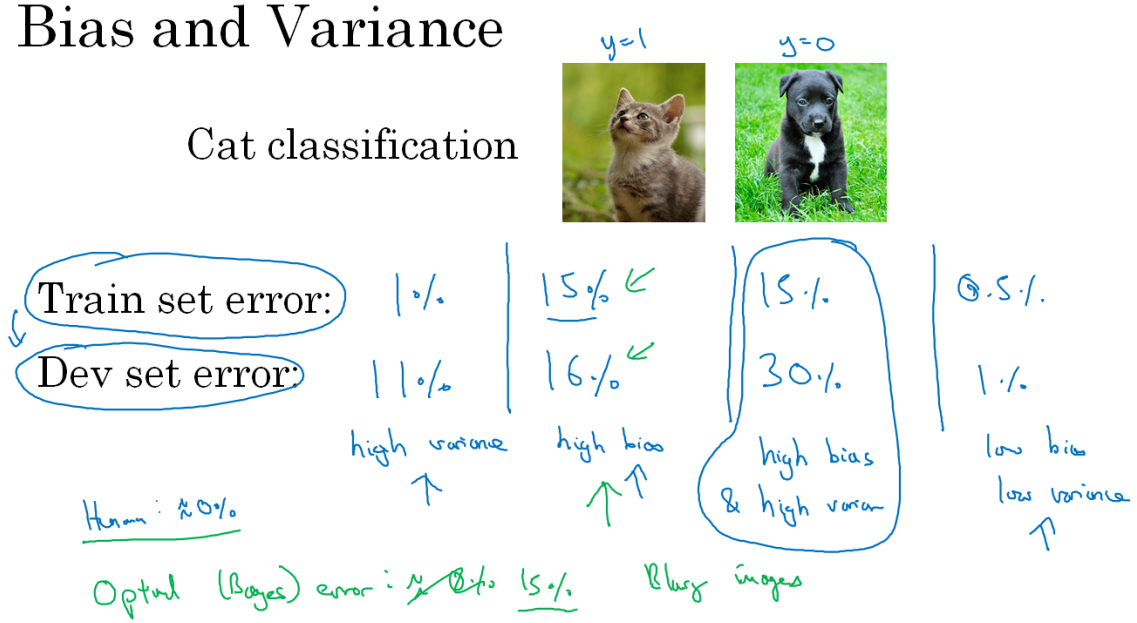
꽤 작은 데이터세트를 보유하고 있다면 이런 전통적인 비율이 괜찮을 것입니다. 하지만 훨씬 더 큰 데이터세트를 보유하고 있으면, dev와 테스트 세트를 더 작은 비율로 하는 것도 괜찮습니다.

Bias and Variance



Andrew Ng

Bias and Variance



Andrew Ng

편향과 편차에 대해서 이해하기 위한 2가지 핵심 수치는 바로 train set 오류와 dev set 또는 development 세트 오류입니다.

[예시1]

training 오류가 1퍼센트이로 dev set 오류가 역시 마찬가지로 의논 목적을 위해, 11퍼센트라고 가정해봅시다. 이 예제에서는 그러면 트레이닝세트에서는 아주 잘 작동하고, development set에서는 아주 못하는 것으로 볼 수 있죠. 그러면 **트레이닝세트를 overfit**했다고 볼 수 있습니다.

[예시2]

만약 **트레이닝 데이터에도 잘 피팅되지 않는 경우**라면, 이것은 **데이터를 underfitting** 하는 것입니다. 그럼 이 알고리즘은 **큰 편향**을 갖는 것입니다. 이것은 dev set에 비교적 괜찮게 일반화되고 있는 반면에 dev set의 성능은 트레이닝 세트대비 1퍼센트 못하고 있습니다. 그렇기 때문에 이 알고리즘은 큰 편향의 문제가 생깁니다. 이전 슬라이드에서 봤던 **가장 왼쪽 plot**과 비슷합니다.

[예시3]

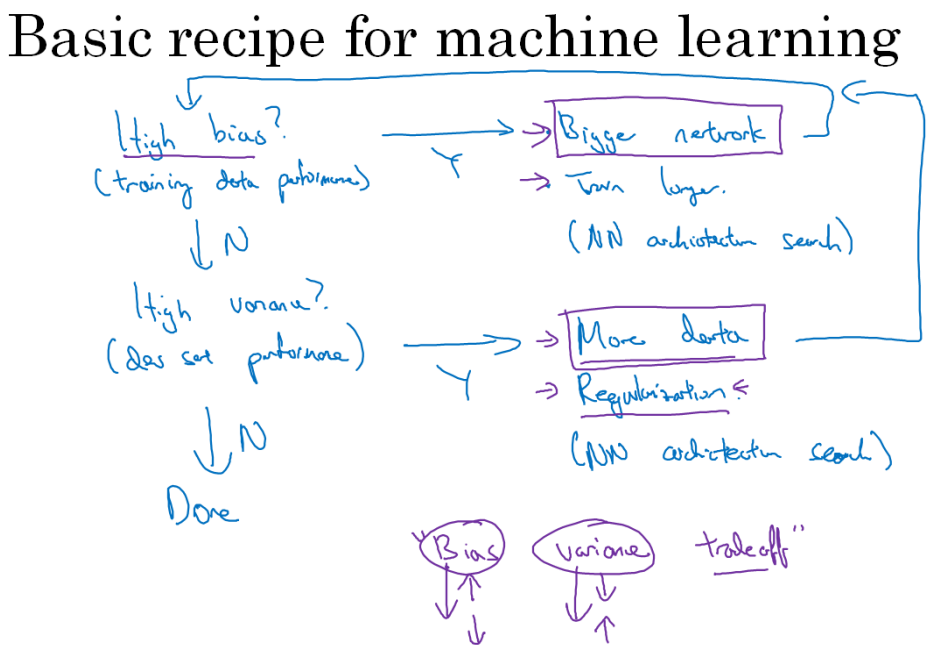
트레이닝 세트가 15퍼센트라고 해봅시다. 그러면 편향이 꽤 큰 편인데요. dev set에서 평가하면 훨씬 더 좋지 않습니다. 30퍼센트라고 해보겠습니다. 이 경우, 이 알고리즘은 **큰 편향**을 갖고 있다고 진단할 것입니다. 트레이닝 세트에서 작 작동하지 않고, 큰 편차를 갖기 때문이죠. 거의 최악의 경우인 것이죠.

[예시4]

마지막 예제를 보겠습니다. 0.5 트레이닝 세트 오류가 있을 경우, 그리고 1퍼센트 dev set 오류를 갖고, 이 경우에 유저는 꽤 만족할 수도 있습니다.

> 가지고 가실 내용은, 트레이닝세트를 봄으로써, 데이터가 얼마나 잘 fitting 하는지 감을 잡을 수 있다는 것입니다. 트레이닝세트에서 dev set로 넘어가는 경우, 얼마나 편차의 문제가 심한지 직감할 수 있습니다.

> 이런 모든 내용은 바로 base error가 꽤 작은 값을 갖고, 트레이닝과 dev set가 같은 분포에서 그려진 가정하에 성립합니다. 이 가정들이 부합하지 않는 경우에는, 할 수 있는 조금 더 정교한 부분이 있습니다. 이 부분은 나중에 다루도록 하겠습니다.



Andrew Ng

첫 번째로는 큰 편향이나 큰 변동이 존재하는지 여부에 따라서 시도할 수 있는 방법이 상당히 다르다는 점입니다. 그러므로 저는 보통 트레이닝 dev set를 이용해서 편향이나 변동이 있는 경우, 진단법을 찾을 것 같아요. 그런 이후, 후속 방편을 찾아서 다음 방법을 모색하는 방식으로 접근할 것 같습니다. 예를 들어, 큰 편향이 문제가 된다고 하면, 트레이닝 데이터를 더 수집하는 것은 도움이 되지 않을 것입니다.

규제 선형 모델-릿지,라쏘,엘라스틱넷

1. 규제 선형 모델의 개요

선형모델의 비용 함수는 RSS(잔차제곱의 합)를 최소화하는, 실제값과 예측값의 차이를 최소하는 것만 고려

→ 학습 데이터 지나치게 맞추고, 회귀 계수 커짐

→ 테스트 데이터 예측 성능 저하

→ 비용함수는 학습데이터의 잔차 오류 값을 최소로 하는 RSS 최소화 방법과 과적합 방지위해 회귀 계수 커지면 안됨.

$$\text{비용 함수 목표} = \text{Min}(\text{RSS}(W) + \text{alpha} + ||W||)$$

(alpha:학습 데이터 적합 정도, 회귀 계수값 크기 제어하는 튜닝 파라미터)

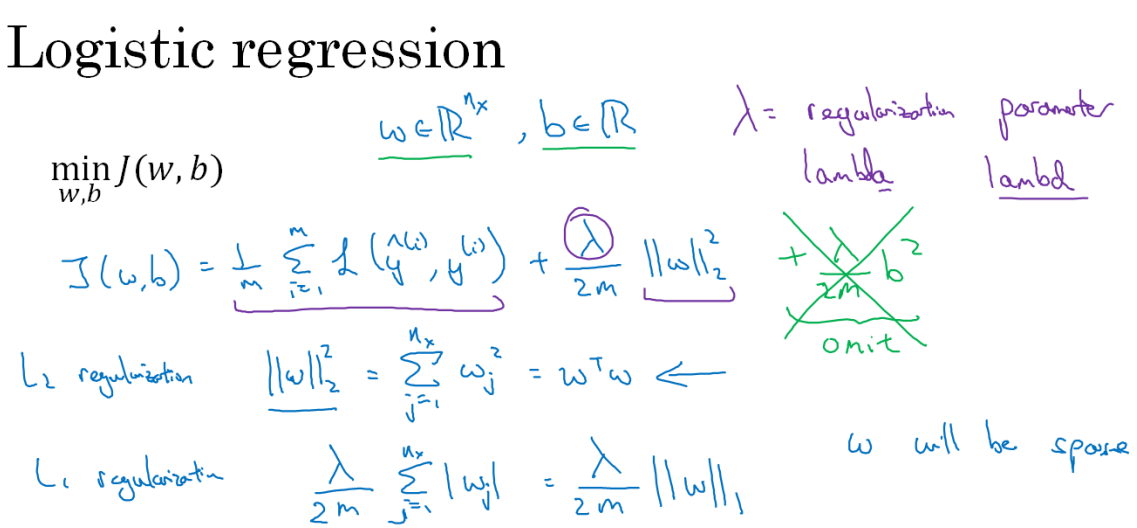
(alpha 크게 하면 과적합 개선 가능, 작게하면 학습데이터 적합을 개선)

alpha를 0 에서 부터 지속적으로 증가시키면 회귀계수 값의 크기 감소.

규제: 회귀 계수 값의 크기를 감소시켜 과적합 개선 시키는 것

1. L2규제: ||W||**제곱**에 패널티를 부여 -->릿지 회귀
2. L1규제: ||W||에 패널티 부여 -->라쏘 회귀

L2 일반화

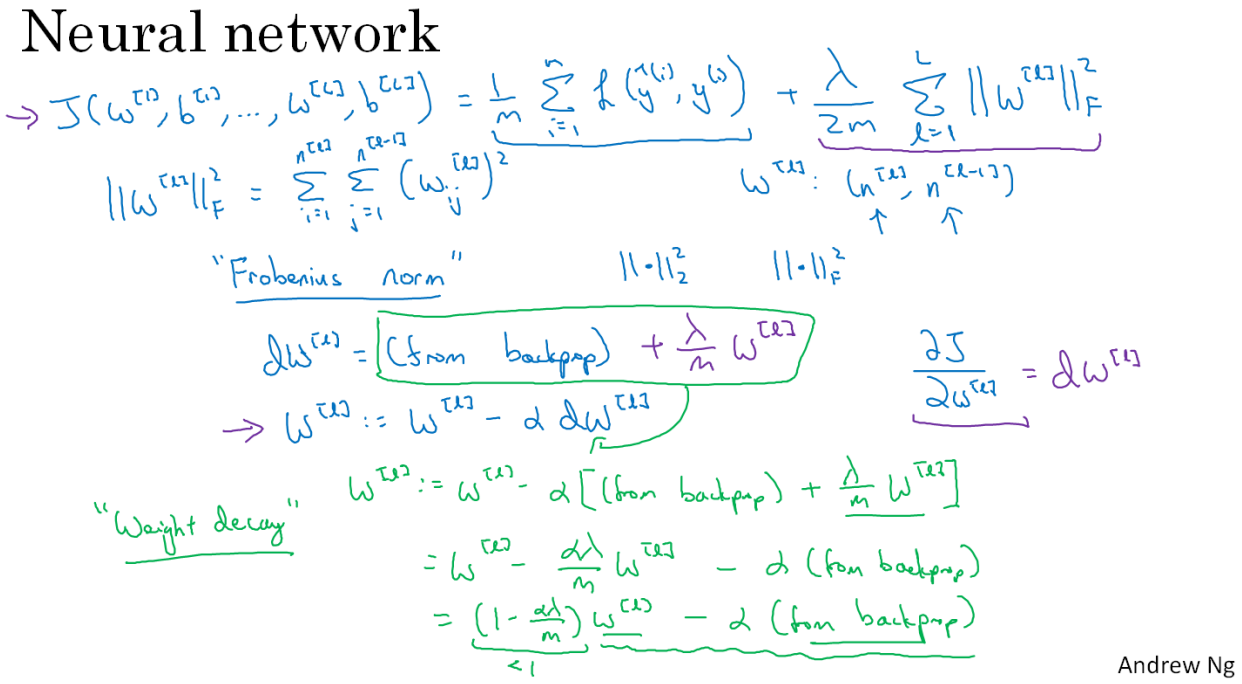


Andrew Ng

왜 매개 변수 w만 일반화 하져? 왜 우리는 여기 b에 무언가를 더 더하지 않는거져? 왜냐하면 b는 많은 파라미터들 중에 단지 하나의 파라미터일 뿐이기 때문입니다. 실제로 저는 주로 그것을 포함 시키는 데에는 신경을 쓰지 않아요.

L1 일반화

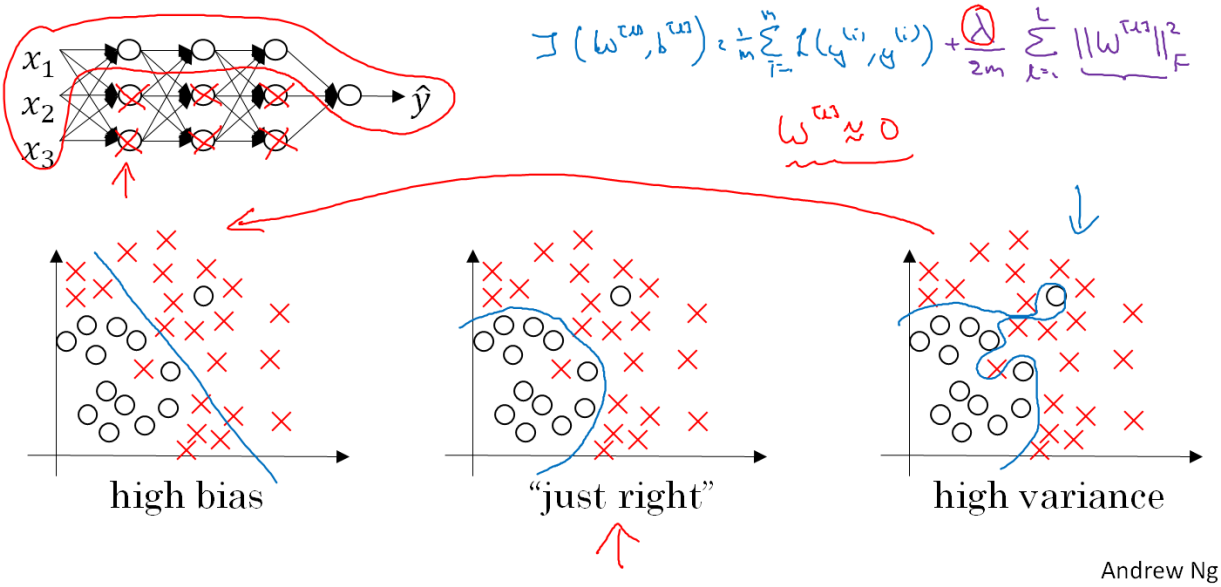
당신은 다른 사람들이 L1일반화에 대해 얘기하는 것을 들어본 적이 있을거에요. 그리고 그것은 이 L2 표준 대신에 더하였을 때 입니다. 대신에 이런 요서를 씁니다. 그리고 이것은 벡터 w의 파라미터의 L1 표준이라고 불립니다. 그러니까 저기 아랫쪽의 subscript 1 인거 보이시져? 그리고 제 생각에 당신이 분모에 m을 넣었던지 아니면, 2m을 넣었는지는 상관없이 단지 스케일링 상수일 뿐입니다.



당신은 어떻게 gradient가 이와 함께 하강하도록 구현할 것인가? 이전에는 backprop을 사용하여 dw를 완성했을 것이다. backprop은 우리에게 w에 관한 J의 편도함수나, w의 임의의 주어진[|]을 줄 것 입니다. 그 후에 당신은 w[l]을w[l]- 학습률 곱하기 d로 업데이트합니다. 이는 우리가 목표에 이 추가적인 일반화 용어를 추가하기 전 입니다. 자, 이제 우리가 이 일반화 용어를 목표에 추가했으니, 당신이 해야할 일은 dw에 이를 더하세요. lambda/m 곱하기 w. 그 후에, 당신은 그저 이 업데이트 된 것을 이전같이 계산하면 됩니다.

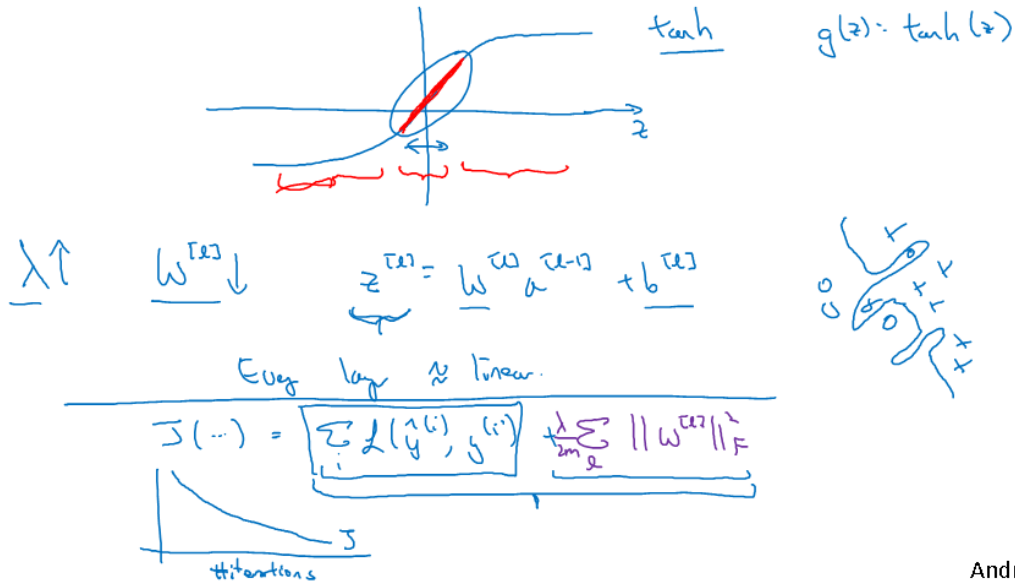
그래서 이 용어는 매트릭스 w[l]이 무엇이든 당신은 좀 더 작게 만들거에요. 그렇죠? 이것은 마치 당신이 매트릭스 w와 그것을 곱하고 매트릭스 w에서 alpha lambda/m 이것을 빼세요. 마치 당신이 매트리스 w를 이 수와 곱하는 것 처럼요. 그것은 1보다 조금 더 적은 숫자일거에요. 그래서 이것이 바로 L2 평균화 일반화가 weight decay라고도 불리는 이유 입니다.

How does regularization prevent overfitting?



Andrew Ng

How does regularization prevent overfitting?



Andrew Ng

이 같은 경우에는, Z의 값이 꽤 작은 이상, 즉, Z가 오로지 작은 범위의 파라미터 값만 갖는다면, 여기정도 될 수 있겠죠, 그럼 여러분은 단순히 tan h 함수의 선형적인 부분을 사용하는 것

입니다.

Z값이 이와 같이 큰 값이나 작은 값까지 이동할 수 있는 경우에만, activation 함수가 덜 선형적인 함수로 정의되는 것입니다.

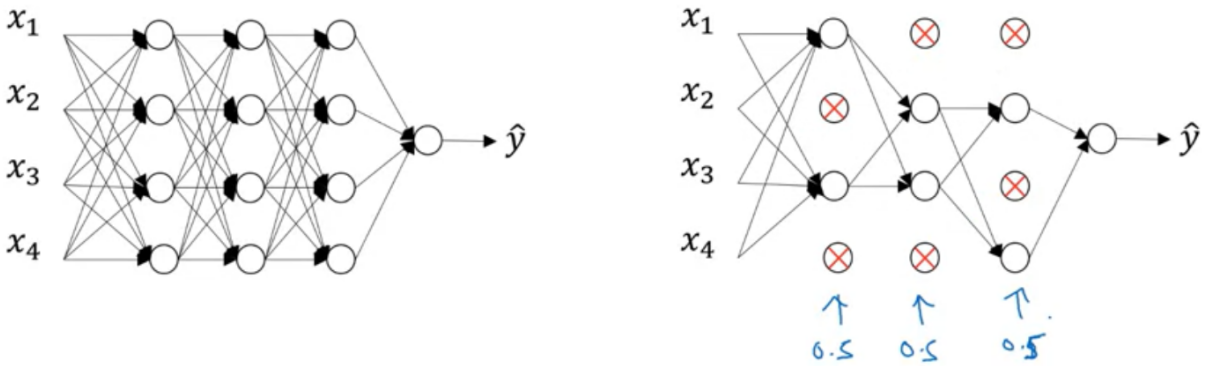
여기서 여러분이 생각할 수 있는 부분은, **램다의 값이 즉, 일반화 파라미터의 값이 큰 경우, 그럼 파라미터들이 꽤 작을 것입니다.**

> 요약해보자면, 일반화가 매우 커지는 경우weight 인 W가 작아지고, Z의 값 또한 꽤 작을 것입니다. 그러면 G(z)는 대략적으로 선형일 것입니다. 이것은 마치 모든 층이 대략적으로 선형인 것과 비슷합니다. 마치 선형회귀와 같이 말이죠. 그러므로 훨씬 덜 overfit하는 것이구요.

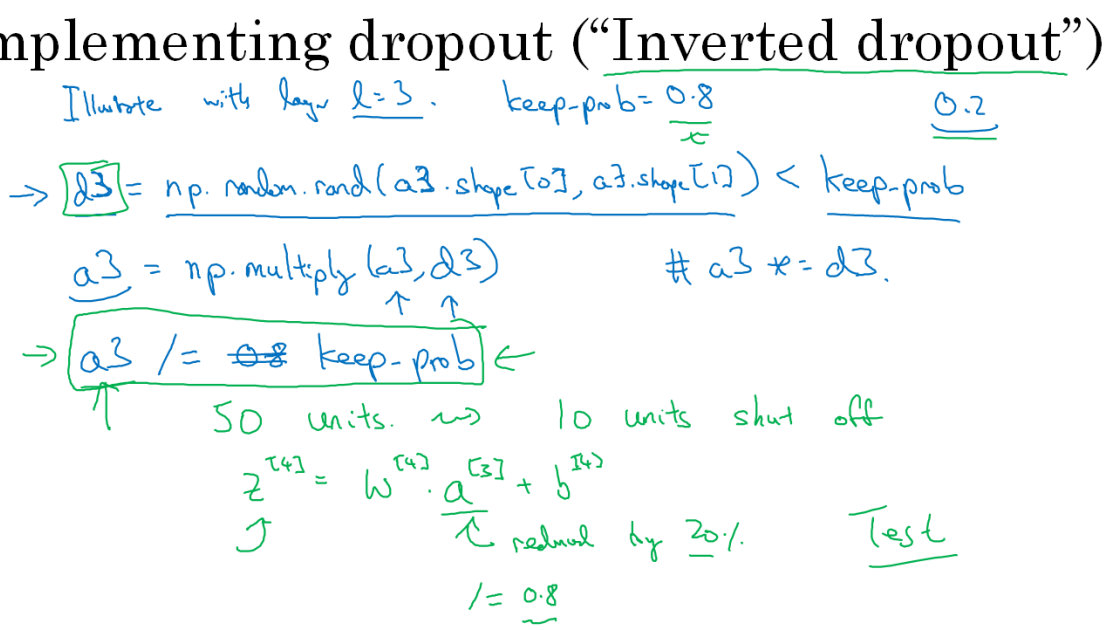
일반화를 도입하는 경우, J가 새로운 뜻이 있다는 것을 기억하십시요. 이전에 정의되었던 J는 이런 항인데요. 이 경우, 점진적으로 감소하는 것을 못 볼 것입니다. 그러므로 기울기 강하를 디버그 하는 경우, 이 새로운 J로 plot할 수 있도록 합니다. 이 두번째 항도 추가될 수 있도록 말이죠.

DropOut

Dropout regularization



여기 각각의 층 별로, 각각의 노드별로, 동전을 던져서 노드를 유지할 것인지 제거할 것인지 각각 0.5 확률이 있다고 할 것입니다. 동전을 던진 후에, 여기 노드들을 제거하겠다고 할 수도 있습니다. 그러면, 여기서 하는 것은 들어오고 나가는 링크 또한 제거를 같이 할 것입니다. 그렇게 되면 훨씬 더 작은, 감소된 네트워크가 남을 것입니다. 그 다음에 후 방향전파 트레이닝을 진행합니다.



Andrew Ng

여기서 할 것은, vector d를 설정해서, d^3는 3이라는 층에대한 dropout vector일 것인데요, d 3이라는 것은 np.random.rand(a) 입니다. 이것은 a3와 같은 모양일 것입니다. 저는 이것이 어떤 숫자보다 작은지 확인할 것입니다. 그걸 keep.prob이라 부를 텐데요. 그러면 **keep.prob**은 숫자가 됩니다. 이전에는 0.5였는데요, 이번 예제에서는 0.8을 이용하겠습니다. 또한, **특정 숨겨진 유닛이 유지될 확률**이 있을 것입니다. 그러므로 keep.prob 은 0.8입니다. 이러면 0.2는 숨겨진 유닛을 제거할 확률입니다. 이것이 하는 것은 random matrix를 생성합니다.

d3는 그러면 매트릭스일 것입니다. 각각의 예시이나 숨겨진 유닛에 대해서 d3가 1이 될 확률이 0.8 해당하는 경우을 가르킵니다. 또, 0이 될 확률이 20퍼센트인 경우를 가르키죠. 0.8보다 작은 랜덤 숫자가 0.8의 확률로 1이거나 참일 확률입니다. 20퍼센트 또는 0.2의 확률로 거짓일 것입니다.

그 다음으로 할 일은, 세번째 층에서 activation을 가집니다. 밑에 예제에서 a3라고 부르겠습니다. 그러면 a3는 산출하는 activation이 있습니다. 그리고 a3를 이전의 a3 곱하기 이거는 element wise multiplication인데요, 이것은 또한 a3 곱하기는 d3와 동일하다고 적을 수 있습니다. 이것이 하는 것은 d3의 모든 요소가 0인 경우, 이런 경우 즉, 각각의 element가 0인 경우는 20퍼센트 확률이었는데요, 여기 이 multiply operation은 **d3의 요소들을 0으로 만**

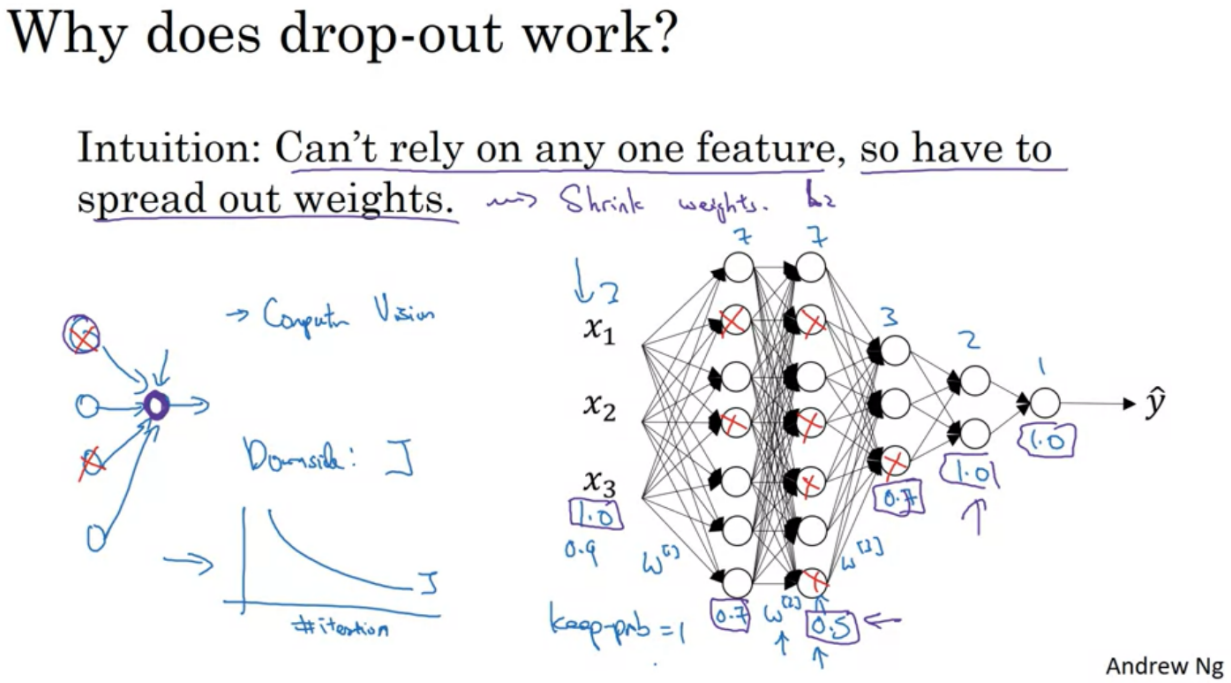
드는 역할을 합니다. 이것을 파이썬에서 하면, 엄밀히 이야기하면 d3가 boolean array 가 됩니다. 그 값이 참, 거짓은 경우를 얘기하죠, 0과 1의 값 대신에요.

마지막으로, 우리는 a3를 가지고 확장을 할 것입니다.

50 단위 또는 50개의 신경세포가 세번째 층에 있다고 가정해보겠습니다. 그러면 a3가 50 x 1차원 또는 인수분해를 하는 경우, 50 x m 차원일 수 있겠습니다. 그럼 80퍼센트의 확률로 유지하거나, 20퍼센트 확률로 제거하는 경우의 수가 있다고 해봅시다. 이 뜻은, 평균적으로 10 유닛은 닫거나, 10유닛이 0으로 된다는 뜻과 같습니다.

이제 z^4의 값을 보면, 20퍼센트의 a3의 element가 0이 될 것이라는 겁니다.

z^4의 기대 값을 감소하지 않게 하기 위해서는 이것을 가져야 합니다. 그리고 0.8로 나눕니다. 왜냐면, 이게 조정되거나 다시 20퍼센트만큼 올라갈 수 있기 때문입니다.



알고 보니 공식적으로 dropout은 L2 일반화가 변형된 버전이라고 할 수 있는데요, 그러나 비중 별로 부과하는 L2 페널티가 다릅니다. 곱셈이 적용되는 activation의 크기에 따라 말이죠, 요약하자면, dropout은 l2 일반화 과 비슷한 효과를 가지고 있다고 보여줄 수 있습니다. L2

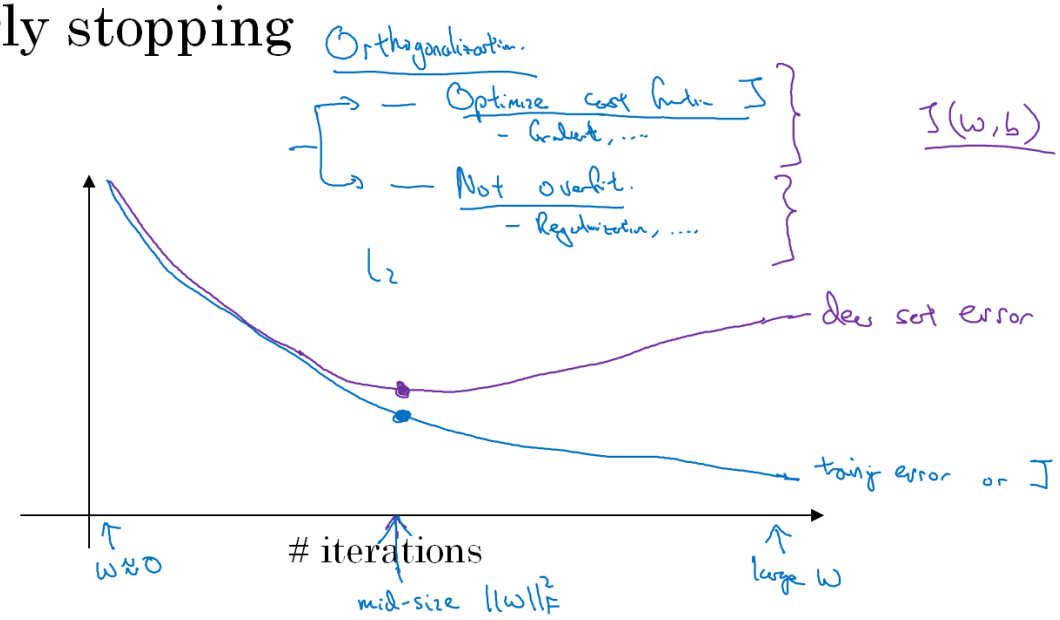
일반화 과 적용되는 부분에서의 차이만 있고 다른 입력값의 스케일에 따라 더 변형한다는 차이가 있습니다.

여기는 3개의 입력 특성이 있는 네트워크입니다. 이것은 7개의 숨겨진 유닛이고요, 7, 3, 2, 1입니다. 저희가 선택해야 했던 매개 변수 중 하나는 바로 keep.prob입니다. 이것은 층마다 유닛을 유지할 확률을 나타내죠. 그렇기 때문에 keep.prob을 층별로 변하게 하는 것이 가능한데요,

W2의 over fitting을 방지하기 위해, 잘하면 여기 이 층을 이것은 2번 층이겠네요, 이것은 keep-prob이 꽤 낮을 수 있습니다. 예를 들어, 0.5일 수 있겠죠. over fitting에 대해 그닥 걱정을 안할 층들에서는 조금 더 큰 값의 key-prob이 있을 수 있겠죠. 잘하면 0.7일 수 있겠습니다.

> 조금 더 간략한 유형의 dropout을 적용하기 위해서 key-prob을 낮게 설정할 수 있습니다. 이것은 마치 일반화 매개 변수인 L2 regularization Lambda의 값을 올리는 것과 유사합니다. 다른 층보다 **더 일반화** 하기 위해서 말이죠.

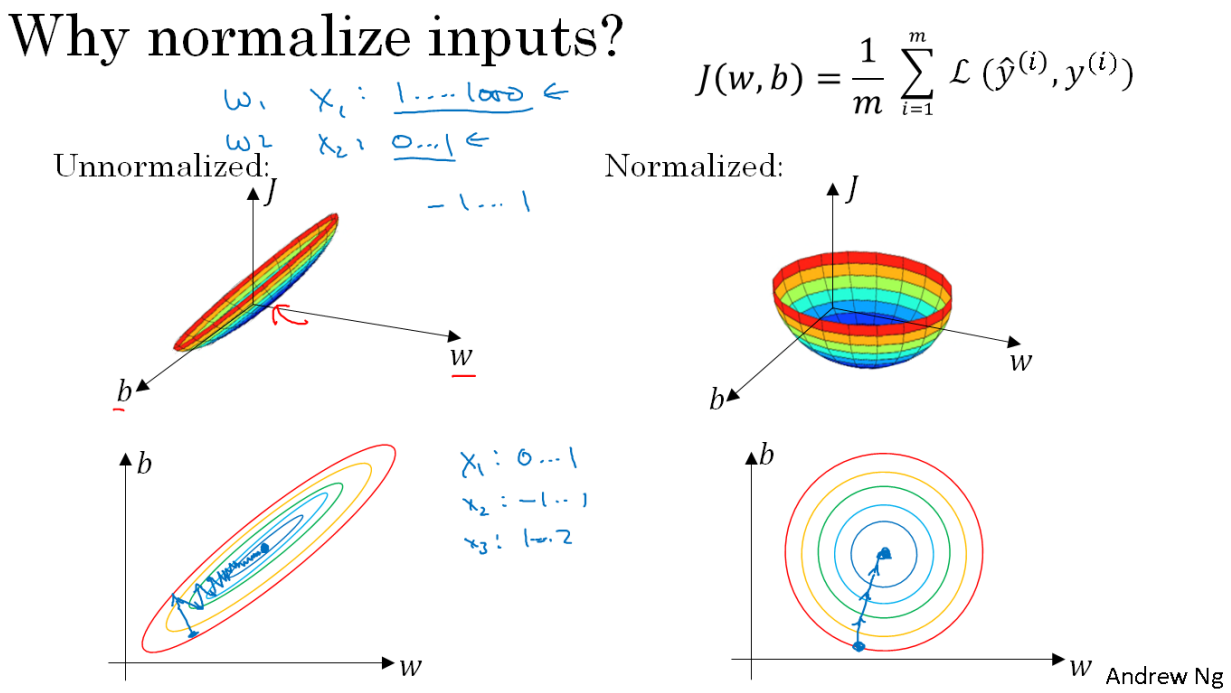
Early stopping



Andrew Ng

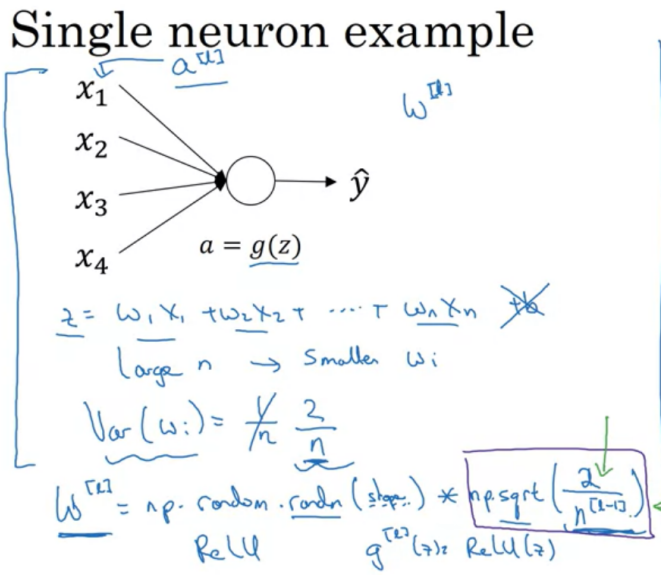
early stopping이 하는 것은 신경망이 여기까지 테스트를 잘 수행했으니 반쯤 지난 시점에서 신경망에서의 트레이닝을 그만 진행하고 싶습니다. 값은 여기까지만 dev set오류를 얻은 데까지만 갖는 것입니다.

반복 수행을 하면서 w의 값은 커지고 또 커질텐데요, 그러면 여기에서는 신경망에서의 w 파라미터수는 훨씬 더 큰 값으로 되어 있을 것입니다. early stopping은 w의 비율이 중간 정도 되는 시점에서 중지시키는 역할을 합니다.



입력 값의 특성을 왜 표준화하려고 하는 것일까요? 이전에 말씀 드렸던 것처럼 비용 함수는 오른쪽 위에 나와 있는 것과 같이 정의할 수 있습니다. 표준화되지 않은 입력 특성을 이용하면, 비용함수가 이렇게 보일 가능성이 큽니다. 푹 퍼진 그릇처럼 생겼는데요, 최소값이 아마도 저기 정도로 보이는 쭉 늘어진 모양의 비용함수 입니다.

반면에, 이 특성을 표준화시키면, 비용함수는 평균적으로 더 대칭 성향을 띄우는 모양이 될 것입니다. 좌측에 보이는 비용함수와 같이 기울기 강하를 이용하면, 굉장히 작은 교육 비율을 이용해야 될 것입니다. 그 이유는, 기울기 강하가 더 많은 단계를 거쳐 최소값에 도달하기 전까지 계속 왔다 갔다 할 수 있기 때문입니다. 반면에 여러분의 함수가 조금 더 구형의 윤곽선을 띄고 있다면 어디서 시작하더라도 기울기 강하가 바로 최소 점에 도달할 수 있습니다.



그러면 z가 폭발적인 값을 갖지 않고 또 너무 작지 않게 하기 위해선 n이 크면 클수록 Wi는 더 작게 원할 것입니다. 맞죠? z는 wixi값의 합이기 때문에 그렇죠, 그리고 이런 항들을 더하는 경우, 각각의 항이 작은 값을 가져야 좋습니다. **한가지 합리적인 방법은 wi의 편차를 1 나누기 n으로 하는 것입니다.**

한가지 합리적인 방법은 wi의 편차를 1 나누기 n으로 하는 것입니다. 여기서 n은 뉴런으로 들어가는 입력 특성의 개수입니다. 실제로 할 수 있는 것은, 비중인 W 매트릭스를 설정합니다. 특정 층에 대해서 말이죠. 이것은 np.random.randn로 합니다. 그 후로, 이것이 생긴 매트릭스의 모양이 어떻던 간에, 여기 넣고요 이것을 곱하기 루트 1나누기 l 층의 뉴런에 들어가는 입력 특성의 개수입니다. 이것은 n(l-1)인데요. 그 이유는 그것이 l층에 들어가는 각각의 유닛별 숫자입니다.

그리고 제가 n에서 n 위첨자 l-1로 간 이유는, 여기 예제에서의 n개의 입력값이 있는 로지스틱 회귀분석이 있는데요 n 개의 입력 특성을 갖습니다. 더 일반적인 케이스는, l층은 n (l-1)층을 가질 것이고 이 입력값이 층마다 있을 것입니다.

Gradient check for a neural network

Take $W^{[1]}, b^{[1]}, \ldots, W^{[L]}, b^{[L]}$ and reshape into a big vector θ.

concatenate

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

Take $dW^{[1]}, db^{[1]}, \ldots, dW^{[L]}, db^{[L]}$ and reshape into a big vector $d\theta$.

concatenate

Is $d\theta$ the gradient of $J(\theta)$?

Andrew Ng

그러니까 여러분이 해야 할 일은 W 매트릭스를 이용하여 벡터로 다시 만들면 됩니다. 모든 W 들을 가지고 벡터로 변경하신 후에, 자이언트 벡터 세타를 형성할 수 있도록 벡터를 연결시켜 줍니다. 자이언트 벡터는 theta (쎄타)라고 발음합니다. 원가 함수 J는 W와 B들의 함수라고 할 수 있죠. 즉, 이제는 이 원가 함수가 쎄타의 함수라고 할 수 있겠습니다.

Gradient checking (Grad check)

$J(\theta) = J(\theta_1, \theta_2, \theta_3, \ldots)$

for each i:

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

$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$ | $d\theta_{approx} \overset{?}{\approx} d\theta$

Check $\frac{\|d\theta_{approx} - d\theta\|_2}{\|d\theta_{approx}\|_2 + \|d\theta\|_2}$ $\approx$ 10^{-7} — great! ← / 10^{-5} → 10^{-3} — worry. ←

$\varepsilon = 10^{-7}$

Andrew Ng

Gradient checking implementation notes

- Don't use in training – only to debug

- If algorithm fails grad check, look at components to try to identify bug.

- Remember regularization.

- Doesn't work with dropout.

- Run at random initialization; perhaps again after some training.

Andrew Ng

다음으로, grad check를 하는 경우, 일반화를 이용한다면, **일반화 항**을 기억하십시오. 만약 비용함수J의 쎄타가 1 나누기 m 곱하기 loss의 합 그리고 더하기 일반화 항이라면, 그리고 합 나누기 L 그리고 wl 제곱, 그러면 이것이 J의 정의입니다. d theta는 J의 기울기고요 쎄타에 대해서요, 그리고 일반화 항도 있습니다. 잊지 말고 여기 항을 추가하십시오. 다름으로, grad check는 dropout에서 작동하지 않습니다. 반복할 때마다. dropout이 무작위로 숨겨진 유닛의 일부를 제거하기 때문입니다.

> 제가 권장 드리는 것은, dropout을 끄고, dropout없이 알고리즘이 옳은지 grad check을 통해 확인한 뒤, dropout을 가시 켜는 겁니다.