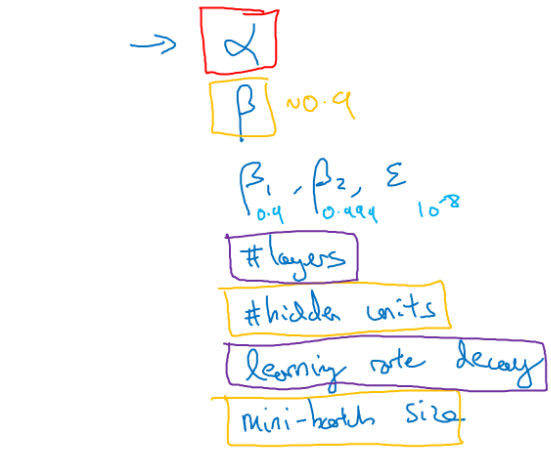
[3주차] Hyperparameter tuning, Regularization and Optimization

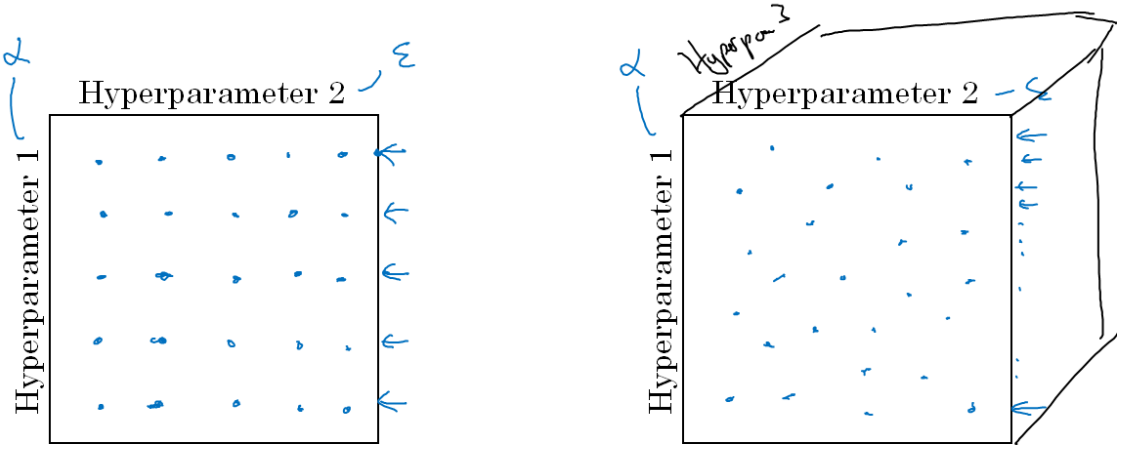


오렌지색으로 동그라미친 것들은, 여기 이 3개의 하이퍼 파라미터는 러닝속도 알파 값 다음으로 중요한 것입니다.

그리고 Adam 알고리즘을 이용하는 경우에는, 저는 거의 항상 베타1, 베타2, 그리고 앱실론은 튜닝하지 않습니다. 거의 항상 0.9, 0.999, 10의 마이너스 8승 값으로 사용하는데요, 여러분이 원하시면 이 값들도 튜닝하실 수 있습니다.

알파가 가장 중요한 것이 되겠고요, 그 다음으로 오렌지색으로 동그라미 친 것이 중요하고, 그 다음은 보라색으로 동그라미 친 것이 중요할 것입니다.

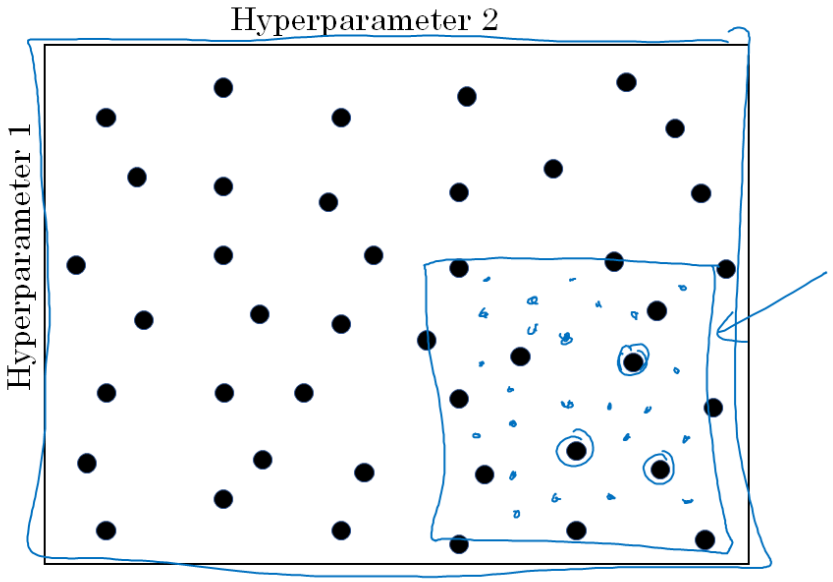
Try random values: Don't use a grid



Andrew Ng

저는 예제를 오로지 2개의 하이퍼 파라미터만 사용하여 설명드렸는데요, 실제로는 여러분이 어 많은 숫자의 하이퍼 파라미터를 사용하여 그 값을 찾을 수 있습니다. 예를 들어, 3개의 하이퍼 파라미터값일 수도 있겠죠, 이런 경우레는 사각형이 아니라 여기 이 다이멘션이 하이퍼 파라미터 3인 정육면체에서 값을 찾을 것입니다.

Coarse to fine

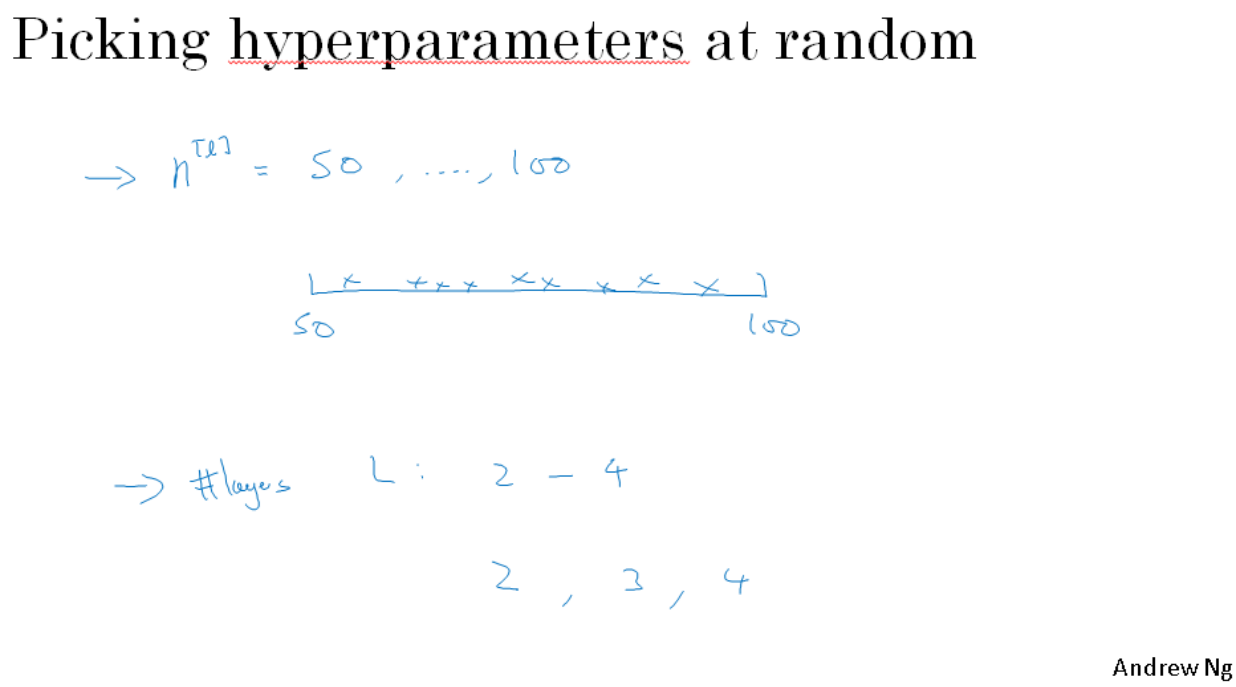


Andrew Ng

하이퍼 파라미터를 샘플링하는 경우에, 흔한 진행 방식은 coarse to fine sampling scheme 입니다.

여기 이 포인트가 가장 잘 작동한다고 알아냈다고 해봅시다. 그리고 그 주위의 멱채의 포인트들도 말이죠. 그러면 마지막 진행절차 도중에, 여러분은 여기 이 조금한 하이퍼 파라미터 부분을 줌인하여 여기 이 공간에대해 조금도 밀도있게 샘플링을 진행합니다. 또다시, 임의의 샘플링을 진행할 수도 있겠죠. 일단 여기 안쪽에서 서칭하는 것에 집중한다고 하면, 여러분이 생각하기에 여기 이 부분이 가장 최적의 하이퍼 파라미터 세팅이 있다고 하면, 여기 파란색 사각형 내부에서 말이죠.

> 2가지 염두하실 점은, 임의의 샘플링을 진행하고, 충분한 서칭을 진행합니다. 그리고 선택적으로 coarse to fine 서칭 단계를 도입할지 고려합니다.

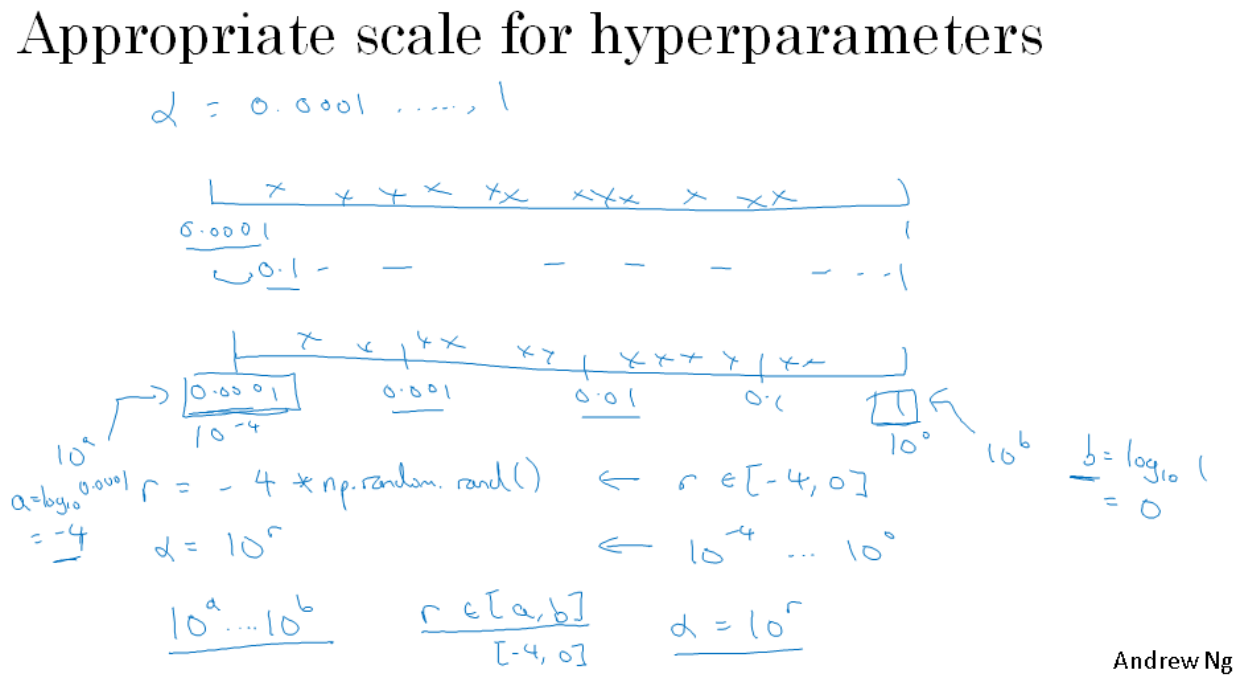


> 적합한 scale을 골라서 하이퍼 파라미터를 탐색해야 합니다.

[case 1]

여러분이 hidden unit의 개수인 n[l]을 찾으려고 한다고 해봅시다. l개의 층에서 말이죠. 그리고 좋은 범위는 50 에서 100 사이라고 해보겠습니다.

또는 여러분이 신경망 네트워크 층의 개수를 결정하려고 한다면, 이것을 대문자 L이라고 하겠습니다. 여러분은 총 층의 개수가 2개에서 4개 사이여야 한다고 생각할 수도 있습니다. 그러면 균일화되게 임의로 고르는 경우, 2, 3, 그리고 4 사이에서 고르는 것이 합리적인 선택일 수 있습니다.



[case 3] - Bad

다른 예제를 보시죠. 여러분이 러닝속도인 알파값을 서칭한다고 해보겠습니다. 이 경우에, 0.0001이 낮은 경계선이고, 그리고 높은 경계선이 1일 수 있겠죠. 여러분이 0.0001에서 1까지인 숫자라인을 그리고, 임의로 균일화되게 샘플링을 진행하면, 대략적으로 샘플값의 90퍼센트 정도가 0.1과 1사이에 있을 것입니다. 그러면 90퍼센트 정도의 자원을 0.1과 1에 집중하는 것입니다. 10퍼센트의 자원을 0.0001 과 0.1에 쏟는 것이구요. 옳은 방법처럼 보이지 않죠.

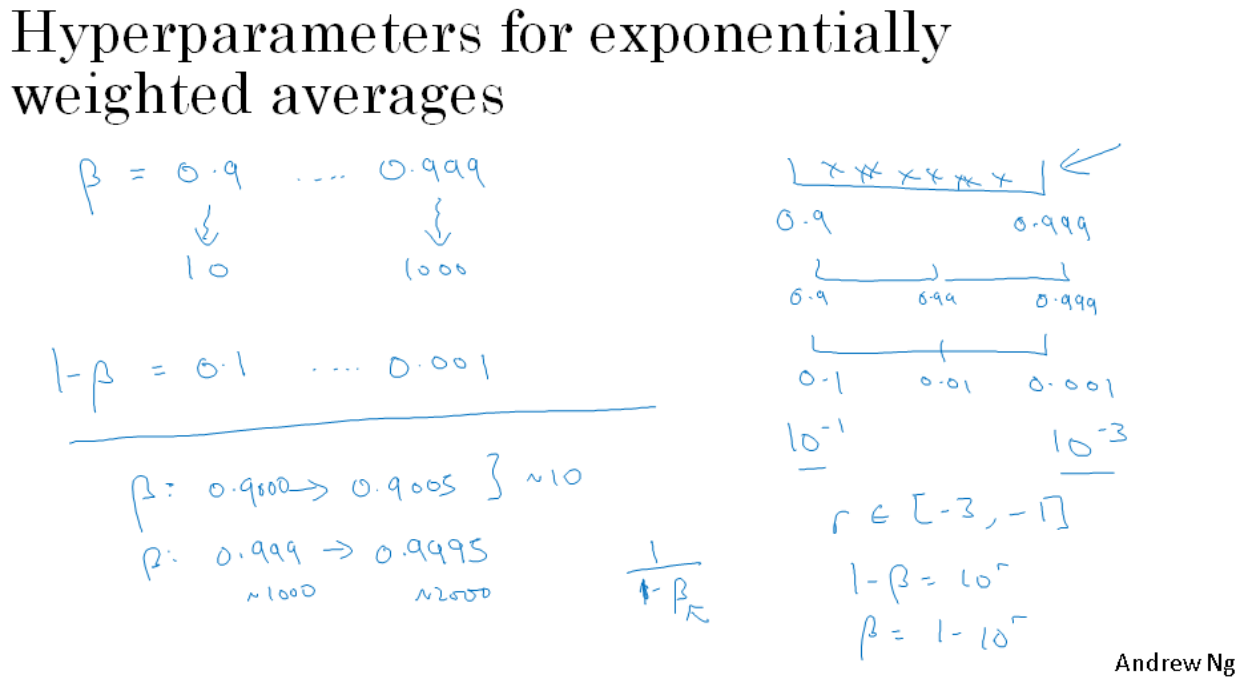
[case 3] - Good

이런 방법 대신에, 하이퍼 파라미터를 log scale에서 서칭하는 것이 더 합리적인 방법으로 보입니다. 여기 log scale에서 균일화되게 샘플링을 진행합니다. 그렇게하면 0.0001과 0.001 사이에 집중하고, 0.001과 0.01 등등에 집중되는 것을 볼 수 있습니다. 그러므로 파이썬에서는 이렇게 도입하는 것이 좋습니다.

r= -4 * np.random.rand() 이런식으로 말이죠. 그리고 임의로 선택된 알파값은 10의 r승이 됩니다.

그런 후에, r을 임의로 a와 b사이에서 샘플링합니다. 이 경우에는, r의 값은 -4와 0사이입니다. 그리고 알파의 값을 임의로 샘플링된 하이퍼 파라미터값으로 10의 r승으로 지정할 수 있습니다.

> 복습하자면, log scale에서 샘플링하려면, 로그를 적용하여 a의 값을 알아내고, 큰 값을 지정하여, 로그를 적요하여 b의 값을 알아냅니다.



[case 4]

마지막으로 조금 헷갈리는 부분은 beta 하이퍼 파라미터 샘플링인데요, 기하급수정 가중평균값을 구하는 베타값입니다.

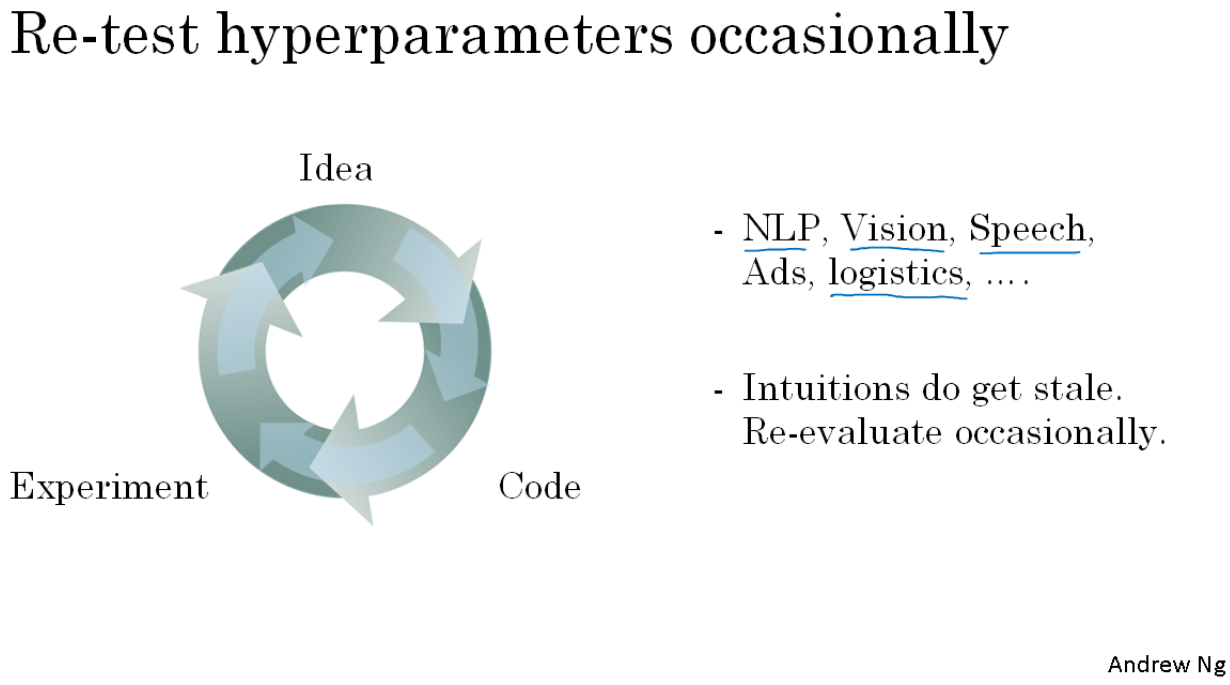
여러분은 베타값이 0.9 에서 0.999사이라고 생각한다고 해보겠습니다. 이것이 여러분이 서칭하려고하는 범위라고 할 수 있겠죠. 기억하실 것은, 기하급수적 가중 평균값을 구하는 경우에, 0.9의 값을 사용하는 것은 마치 10개의 값에서 평균을 구하는 것과 같습니다. 마치 10일 동안의 평균기온을 구하는 것처럼 말이죠. 반면에 0.999의 값을 사용하는 것은 1000개의 값에서 평균치를 구하는 것과 같습니다.

생각하기 가장 쉬운 방법으로는, 1 빼기 베타에서 그 값의 범위를 찾는 것입니다. 그러면 이제 범위는 0.1에서 0.001이 되겠죠. 저희는 베타 사이에서 샘플링을 할텐데요, 0.1에서 0.1사이의 값, 그리고 0.1과 0.001사이에서 말입니다.

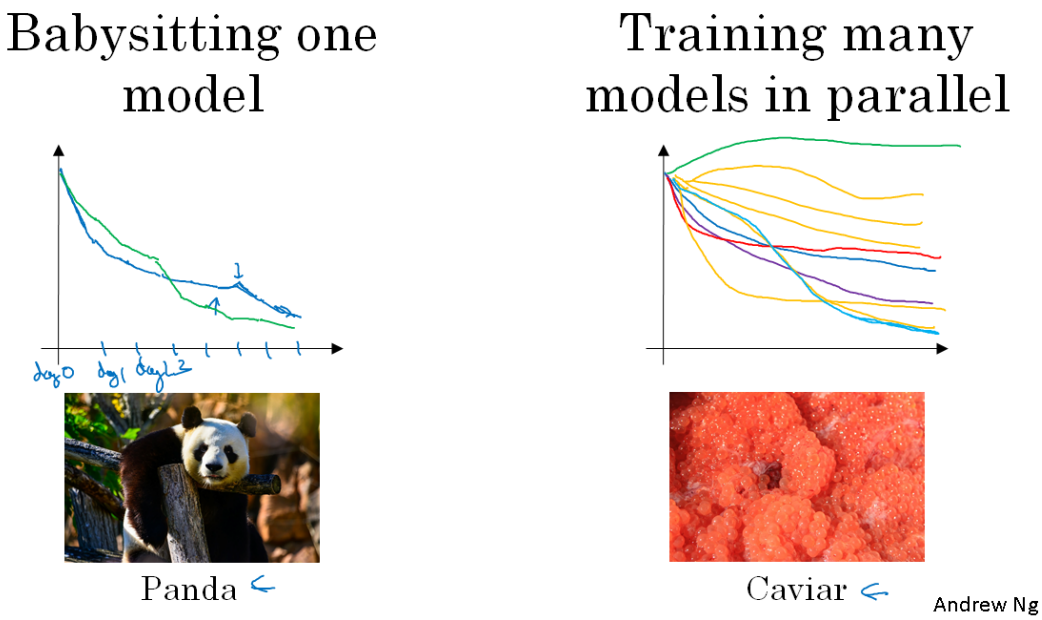
→이것은 10의 -1승이구요, 이것은 10의 -3승입니다.

거꾸로 바뀌였죠, 큰 값이 왼쪽, 작은값이 오른쪽에 있습니다. 그러므로 여러분은 r을 임의로 균일화되게 샘플링을 합니다. -3에서 -1값 사이에서 말이죠. 그리고 1-베타를 10의 r승으로 합니다. 그러므로 베타는 1 빼기 10의 r승이 되겠죠. 그러면 이것이 임의로 샘플된 하이퍼 파라미터의 값이 됩니다.

> 왜 선형의 scale로 샘플링을 하는게 나쁜 아이디어인지에 대해서 말이죠. 그 이유는, 베타가 1과 근접할 경우, 결과값에 대한 민감도가 변합니다. 아주 조금의 베타값 변화에도 말이죠. 또는, 1-베타값이 0과 근접할때도 그럴 수 있죠



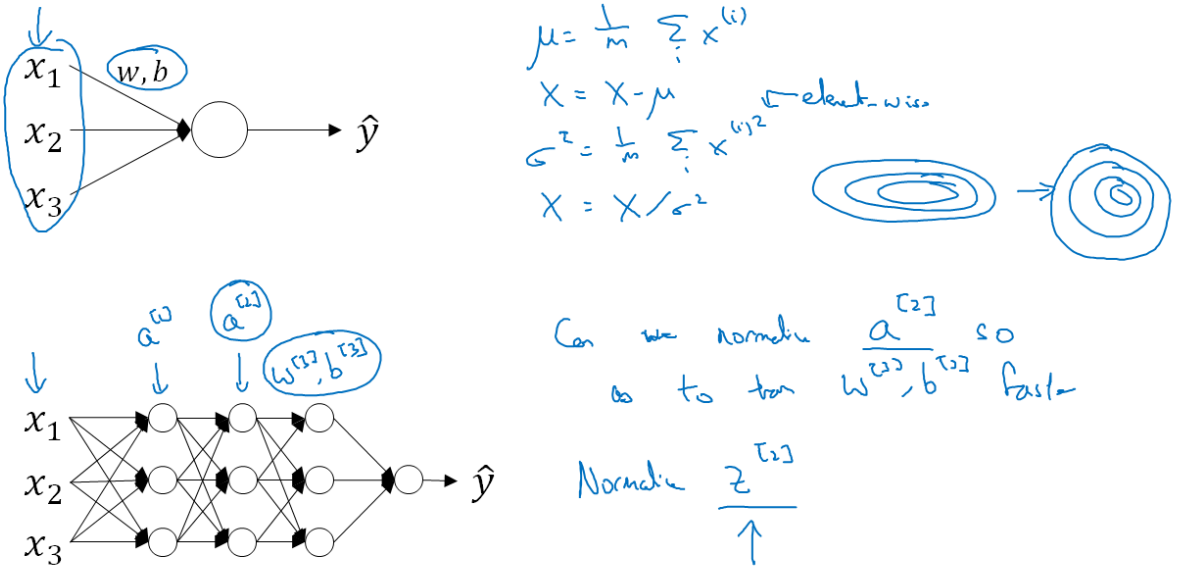
여러분의 하이퍼 파라미터 최적 설정값이 오래될 수 있습니다. 그렇기 때문에 저는 여러분이 retesting 하거나 하이퍼 파라미터를 재평가하는 것을 추천드립니다. 몇개월에 한번씩은 말이죠. 여러분이 가지고 있는 값들에 대해서 만족하기 위해서 말입니다.



하이퍼 파라미터 서칭법

1. **1개의 모델을 babysit (panda approach)**
   - 아주 큰 데이터가 있고 산출 자원은 많이 없을 때 적용
   - 산출자원이라 하면 CPU나 GPU같은것이 많이 없어 1가지 모델만 트레이닝할 여력 밖에 없는 경우 또는 아주 작은 수의 모델만 한번에 트레이닝 시킬 수 밖에 없는 경우에 해당
2. **parallel 방식으로 여러 모델을 트레이닝 (caviar approach)**
   - 어떤 하이퍼 파라미터의 특정 설정 값으로 혼자 작동하게 놔둘 수 있는데요.
   - 동시에 다른 설정값의 하이퍼 파라미터를 가진 다른 모델을 시작할 수 있습니다.

Normalizing inputs to speed up learning



Andrew Ng

배치 정규화는 여러분의 하이퍼 파라미터 서치 문제를 훨씬 더 쉽게 만들어 줍니다. 그리고 신경망을 더욱 튼튼하게 만들어주죠.

입력값의 특성을 일반화시키는 것이 속도를 높혀줄 수 있다는 기억하실 것입니다. 그러면 평균값을 구하고, 트레이닝세트에서 평균값을 빼고, 편차를 계산합니다.

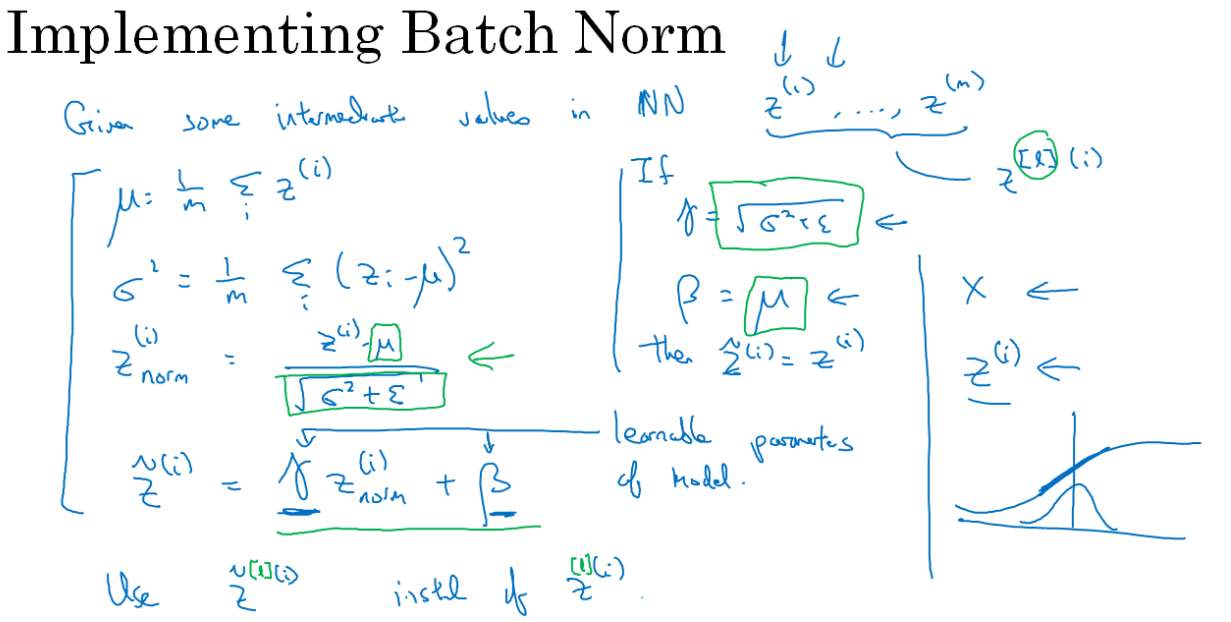
xi 제곱의 합이죠. 이것은 element-wise squaring입니다.

더깊은 모델에서...

평균값과 a2의 variance를 정규화해서 w3 와 b3의 트레이닝을 더 효율적으로 만드는 것이 좋지 않을까요?

로지스틱 회귀분석 같은 경우엔, x1,x2,x3를 정규화 시켜줌으로써 w와 b를 더 효율적으로 트레이닝 시킬 수 있다는 것을 알았습니다. 여기서 문제는, 어떤 숨겨진 층에 대해, a의 값을, a2라고 하겠습니다. 이 값을 정규화할 수 있는지가 문제입니다. 여기 예제에서는 어떤 숨겨진 층에서도 이런데요, w3와 b3를 더 빨리 트레이닝 시키는 방법이겠죠. 맞죠? a2는 다음 층의 입력값이기 때문에, w3와 b3의 트레이닝에 영향을 줍니다.

> 그러면 이것이 배치 정규화가 하는 역할입니다. 보통 줄여서 batch norm 이라고도 합니다. 엄밀히 이야기하면 a2가 아닌 z2를 정규화 시키는 것일 텐데요,



이것은 모두 l층에 맞춰진 것인데요, [l]은 생략하겠습니다.

variance의 값은 여러분이 예상하는 공식을 이용하여 계산하구요, zi들을 각각 이용해서 정규화합니다. 그러면 zi 정규화된값을 평균값에서 빼고 표준편차로 나누어주면서 갖게되는데요, **숫자적으로 안정감을 주기위해서 보통 앱실론을 분모에 더합니다.** 여기 시그마 제곱과 같이 말이죠. 이것은 어떤 추정치의 경우 0이 되는 경우도 있습니다.

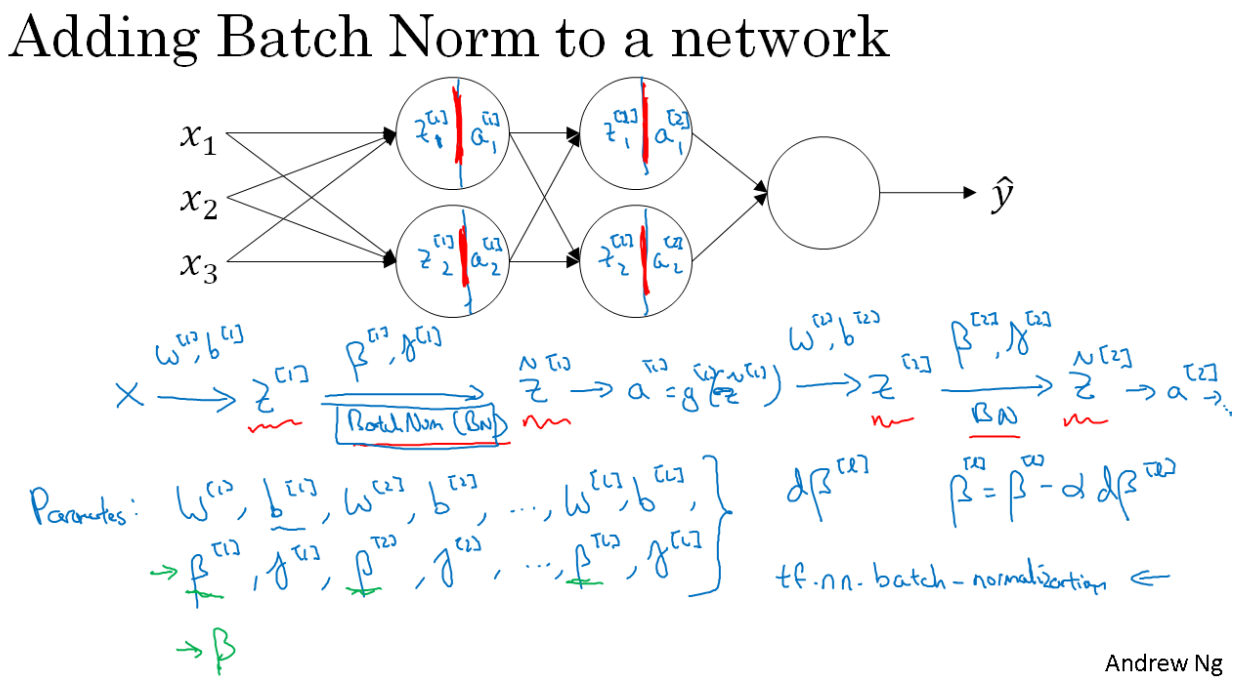
그럼 이제 저희는 z의 값을 갖고, 평균값이 0, 그리고 standard unit variance를 갖가되게 정규화 시켰습니다.

그렇지만 저희는 숨겨진 유닛이 항상 평균값 0 과 variance 1의 값을 갖길 바라진 않죠. 숨겨진 유닛은 아마도 다른 분포도를 갖는 것이 더 말이 되겠죠. 그러므로 저희는 대신해서 이것을 zi 라고 부를텐데요, 이것은 감마 zi 정규화 더하기 베타입니다.

예를 들어, 시그모이드 activation 함수의 경우, 여기 모든 값이 밀집되어 있는 것은 원치않습니다. 여러분은 이 것이 더 큰 variance나 평균값을 가져 0보다 다른 값을 가지면서 비선형의 시그모이드 함수가 나오길 바랄 것입니다. 선형함수의 모양을 갖는 것보다 말이죠. 그러므로 파라미터 감마와 베타를 이용해서 여러분의 zi 값이 원하는 범위의 값을 갖도록 할 수 있는 것입니다.

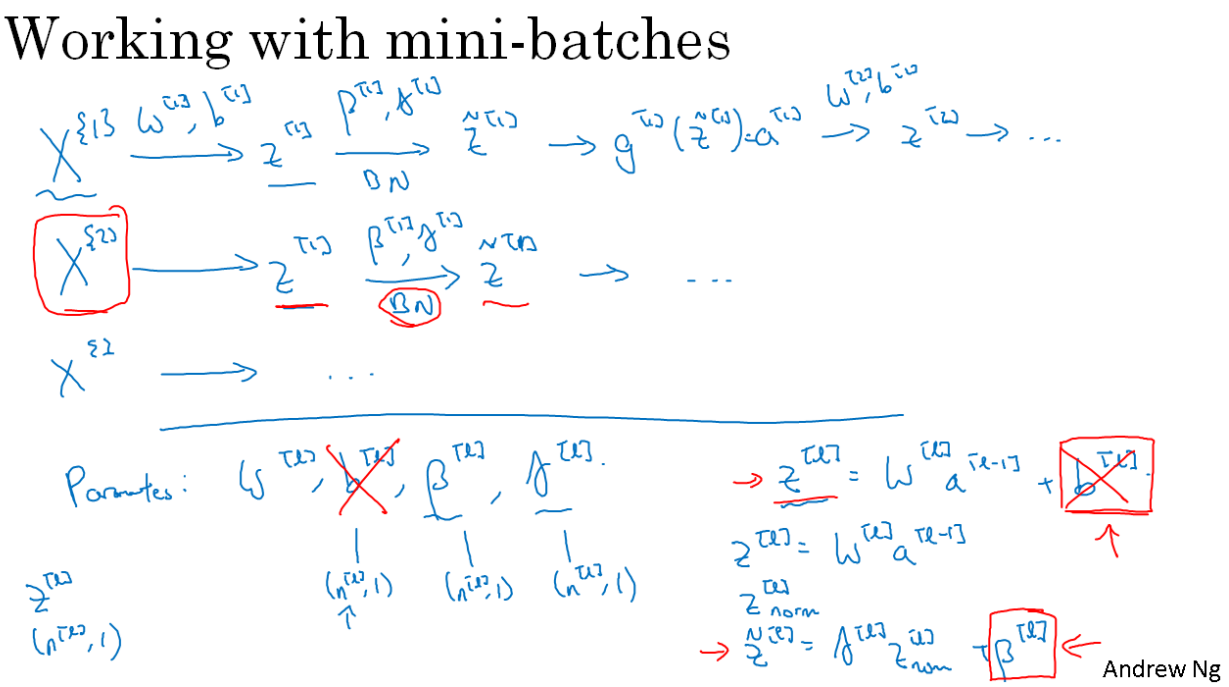
기울기 강하 와 모멘텀이나 RmsProp 또는 Adam을 이용해서 감마와 베타 파라미터r를 업데이트 할 것입니다. 신경망의 weight를 업데이트 하는 것과 마찬가지로 말이죠. 자 그럼 감마와 베타의 효과는 z tilde의 평균값이 여러분이 원하는 값으로 되게 해주는 것입니다. 만약 **감**

마가 시그마 제곱 더하기 앱실론의 루트값인 경우, 즉 감마가 여기 분모 항이라고 하면, 그리고 **베타가 뮤,** 즉 여기 위의 값이라고 하면 감마 z 정규화 더하기 베타의 효과는 이 식을 거꾸로 하는 것과 일치합니다.



명료성을 위해 **여기 베타**를 보십시오, 여기 값은 여러 기하급수적 가중 평균을 구했을 때 산출했던 모멘텀에서 있었던 하이퍼 파라미터 베타와는 상관이 없습니다. Adam paper의 저자들은 베타를 사용해서 하이퍼 파라미터를 표기합니다. 그리고 배치 정규화 paper의 저자들은 여기 이 파라미터를 나타내기위해 베타를 썼고요, 하지만 여기 2개는 완전히 다른 베타들입니다.

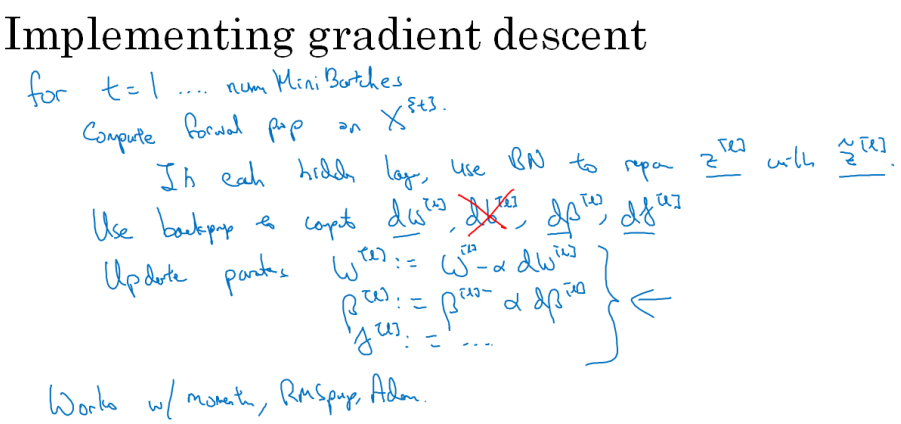
저는 2개의 경우 모두에서 베타를 쓰기로 했는데요



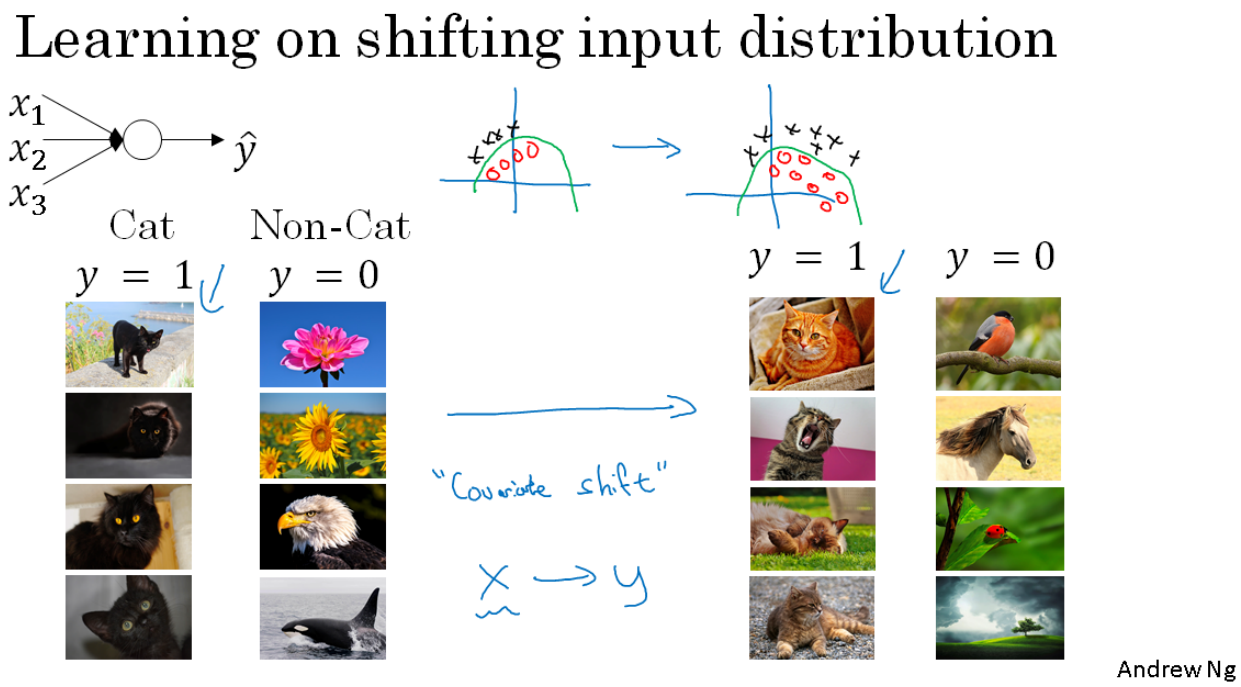
그런데 배치 정규화를 도입하는 것은 딥러닝 프레임웍에서 보통 한줄의 코드입니다. 현재까지 배치 정규화가 마치 전체 트레이닝 세트에서 한번에 트레이닝 시키는 것처럼 이야기 했는데요, 마치 배치 기울기 강하를 쓰는 것처럼 말이죠. 실제로는, 배치 정규화는 보통 트레이닝 세트의 미니 배치와 같이 적용됩니다.

z가 계산된 방식을 한번 보겠습니다. ZL = WL x A of L - 1 + B of L 인데요, 이것이 뜻하는 바는, bl이 어떤 값을 갖더라도, 이 것은 빠질 것입니다. 그 이유는 배치 정규화 단계에서, ZL의 평균값을 구한 뒤에, 평균값을 뺄 것이기 때문입니다. 그러므로 여기 모든 미니 배치 예제에서 상수를 더하는 것은 아무 변화도 일으키지 않습니다.

마지막으로 ZL의 다이멘션은 하나의 예시에서 진행할 경우, 그 값은 NL x 1 인데요 그러면 BL 은 다이멘션 NL x 1인데요, 만약 NL이 l층에서의 숨겨진 유닛개수라고 하면 말이죠. 그러면 베타 L과 감마 L의 디아멘션도 역시 마찬가지로 NL x 1이 될것입니다. 그 이유는 이것이 숨겨진 유닛의 개수이기 때문입니다. NL개의 숨겨진 유닛이 있고, 각각의 숨겨진 유닛에 대해서 베타 L, 감마 L을 통해 평균값과 variance이 scale 되는 것입니다.



여러분이 미니 배치 기울기 강하를 사용한다는 가정하에 T=1에서 미니 배치 숫자 까지 반복하는데요, 그리고 미니 배치 XT에 대해서 전 방향전파을 도입할텐데요, 그리고 각각의 숨겨진 유닛에 대해서 전 방향전파을 하는 것은 ZL을 Z tilde L로 바꾸기 위해 배치 정규화를 사용하십시오. 그러면 여기 미니 배치 안에서 z의 값은 정규화된 평균값과 variance를 갖게 되는데요, 여기 값과 버전은 z tilde l이 됩니다. 그 다음에 후 방향전파를 이용해서 dw,db, 그리고 모든 l의 값에 대해서 d 베타, d 감마, 사실 엄밀히 이야기하면, B를 없앴으므로, 이거는 사실 사라집니다. 마지막으로 파라미터를 업데이트 시킵니다.



[장점 1]

모든 특성을 정규화함으로써 입력 특성 x가 비슷한 범위의 값을 주고, 이로 인해 러닝의 속도가 증가합니다. 입력 층에 대한 것뿐만 아니라 숨겨진 유닛을 위해서도 그렇게 하는 것입니다.

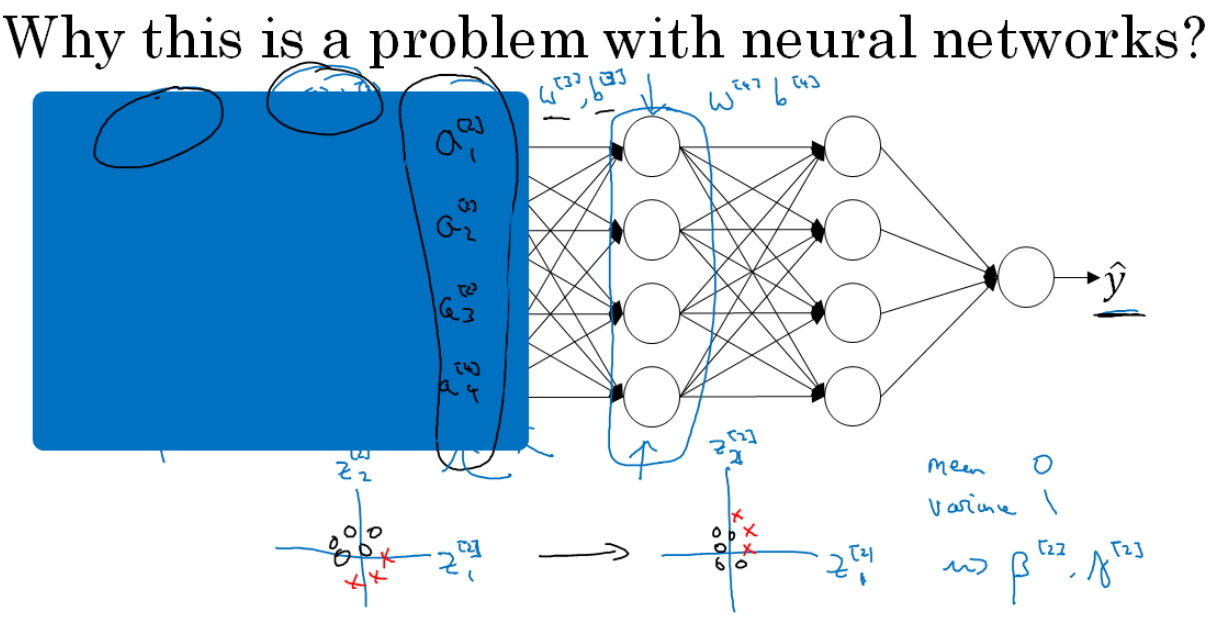
[장점 2]

weight를 만듭니다. 여러분의 네트워크보다 더 나중 또는 더 깊은 곳 같은 경우, 예를 들어, 10번째 층에서는 weight의 변화에 더 더 튼튼합니다.

Ex)

여러분의 러닝 알고리즘이 이런 초록색 decision 경계선을 단순히 왼쪽의 데이터를 보고 발견하는 것은 예상하지 않은 것입니다.

이런 데이터의 분포가 변하는 아이디어를 covariate shift라고 하는 거창한 이름으로 불리는데요, 기본적인 아이디어는 만약 X에서 Y로 가는 매핑을 배운 경우, x의 분포도가 만일 변경하면 러닝 알고리즘을 다시 트레이닝 시켜야 할 수도 있다는 것입니다.



그러면 잠시 왼쪽에 있는 노드들은 가려보겠습니다. 세번째 숨겨진 층의 기준으로는 이러한 값을 갖고, 이 값들을 A_2_1, A_2_2, A_2_3, 그리고 A_2_4 라고 하겠습니다. 그러나 이런 값들은 차라리 X1, X2, X3, X4인 특성이라고 칭하는 것이 낫겠습니다. 그리고 세번째 숨겨진 층의 역할은 이 값들을 갖고 ŷ 을 매핑하는 것입니다.

그러면 기울기 강하를 진행하는데요 여기 파라미터들 W_3_B_3,W_4_B_4, 그리고 W_5_B_5 도 말이죠, 잘하면 이런 파리미터를 배워서 네트워크가 여기 왼쪽에 그린 값들에 대해 매핑을 잘할 수 있게 될 수도 있겠죠. ŷ 값을 말이죠.

네트워크 또한 W_2_B_2 와 W_1B_1의 파라미터를 사용합니다. 그렇기 때문에 여기 파라미터의 값이 변하면 여기 값, 즉 a2의 값도 변할 것입니다. 그러면 세번째 층의 시각에서 보면, 여기 숨겨진 유닛의 값은 항상 변하는 것입니다. 그러므로 이전 슬라이드에서 언급했던 covariate shift 문제로 인해 영향을 받게 되는 것입니다.

> 그러므로 배치 정규화가 하는 것은, 이런 숨겨진 유닛의 분포가 왔다 갔다 이동하는 양을 줄여줍니다.

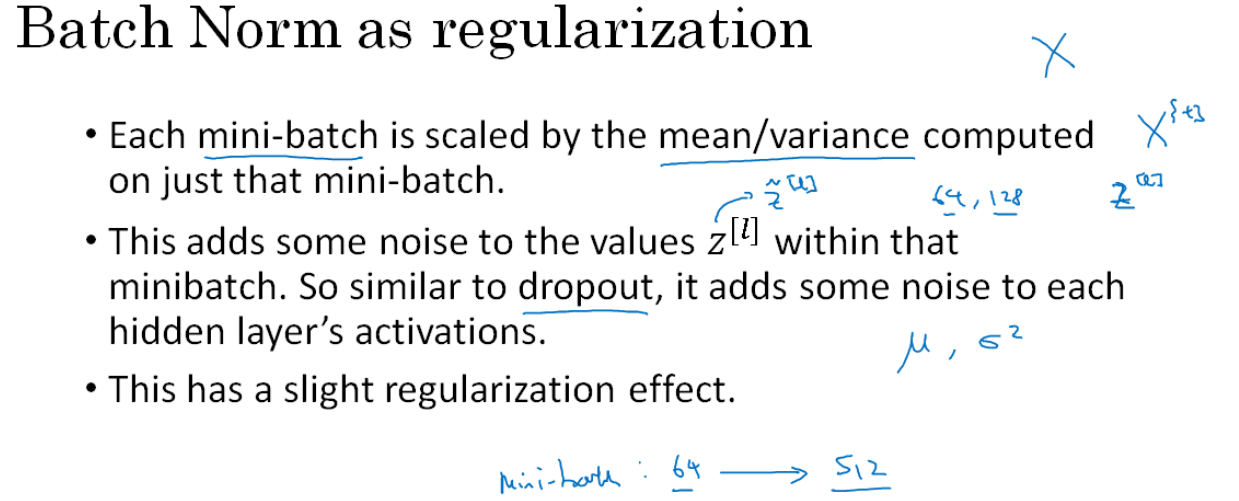
> 배치 정규화가 주장하는 것은 for Z_2_1 Z 와 Z_2_2의 값이 변할 수 있다는 것입니다. 그러면 정말로 신경망이 이전에 있는 층들의 파라미터들을 업데이트하면서 그 값이 변할 것입니다. 하지만 배치 정규화가 보장하는 것

> 은, 어떻게 변하더라도, Z_2_1 와 Z_2_2 의 평균값과 variance는 똑같을 것이라는 점입니다.

그러므로 구체적으로 Z_2_1 와 Z_2_2 값이 변하더라도 그들의 평균값과 variance는 변동없이 그대로 각각 0과 1의 값을 유지할 것입니다. 꼭 평균값 0과 variance 1이 아닐수는 있는데요, 어쨋든 베타2와 감마2에 지배되는 값으로 변동없이 유지될 것입니다.

여기 이런 값들을 더 안정감 있게 해주고, 그럼으로 인해, 신경망의 나중의 층들이 그 자리를 지킬 수 있게 틀을 마련해줍니다. 입력값의 분포도가 약간 변하기는 하지만, 더 작게 변하고, 이것이 하는 것은 이전 층이 계속 러닝하면서도, 여기 이 부분이 다른 층에게 압박하는 다음 층들이 적응할 수 있도록 배워야 하는 양이 줄어드는데요, 다르게 말하자면, 앞에층의 파라미터가 배워야하는 것들과 뒷층의 파라미터들이 해야하는 것들의 coupling을 약화시켜줍니다. 그렇게해서 각각의 층이 스스로 러닝을 진행할 수 있도록 해주는 것이죠. 다른 층과는 조금 더 독립적으로 말이죠. 이것은 결과적으로 전체 네트워크의 속도를 올려주는 효과를 줍니다.

일반화 효과

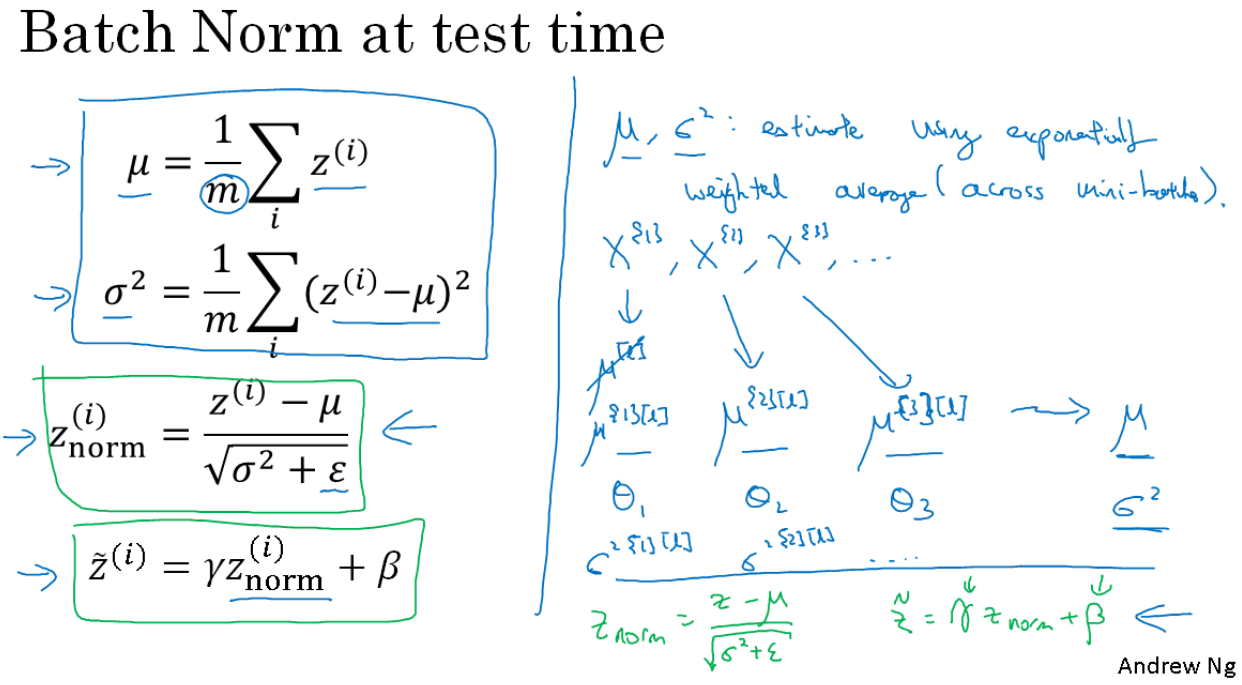


여기 미니 배치 에서 산출된 평균값과variance는 전체 데이터세트에서 계산된 값보다 더 많은 noise를 함유하고 있기 때문에, 미니 배치에서만 산출됐기 때문이죠, 예를 들어, 64 또는

128 또는 256 또는 더 큰 트레이닝 예시일 수 있습니다. 그렇게 mean 과 variance가 더 작은 데이터 샘플에서 계산됐기 때문에 더 noisy 합니다.

그렇기 때문에 dropout과 비슷하게 각각의 hidden layer activation에 noise를 더합니다. dropout 이 noise가 있는 방식은 숨겨진 유닛을 가지고 0에 곱하고 어떨 확률에 곱합니다.

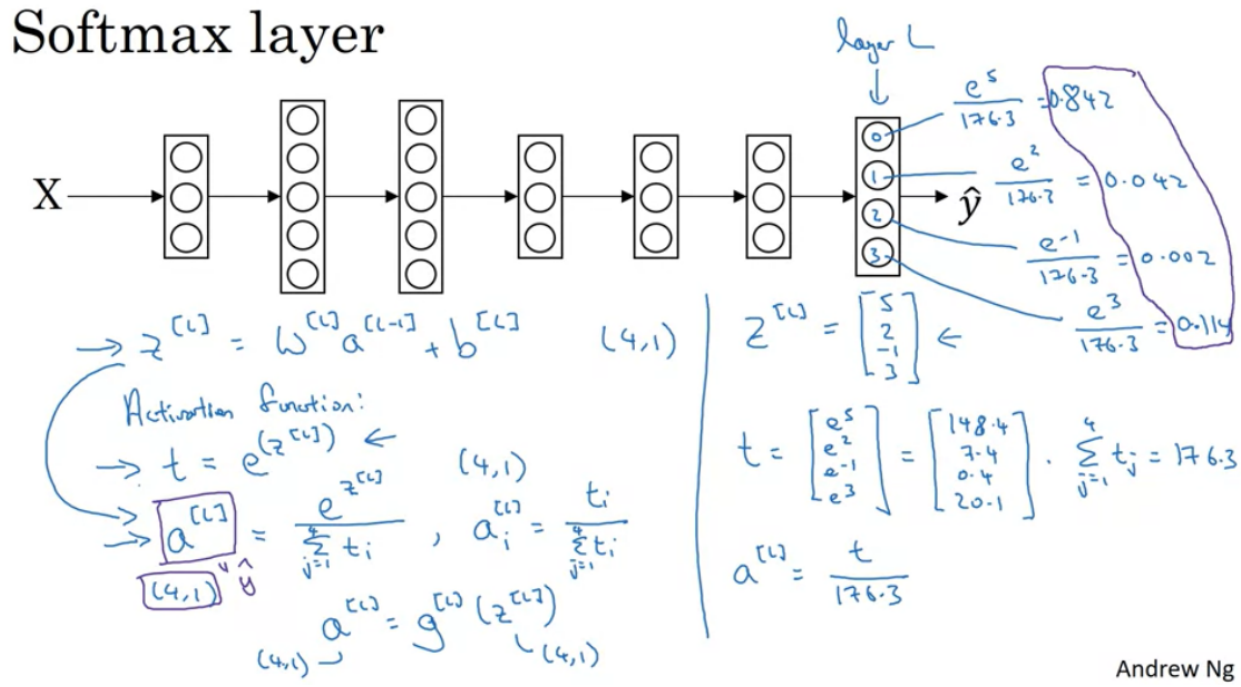
여러분이 알아야하는 내용이 한가지 있습니다. 그것은 배치 정규화가 한대의 미니 배치씩 데이터 처리한다는 것입니다. 이것은 미니 배치 에서 평균값을 계산하고, variance를 계산합니다. 테스트타임에서는 예사하는 것들을 만드려고 하고, 신경망을 평가하려 합니다. 여러분은 미비 배치 예시가 없을 수도 있는데요 여러분은 1개의 예시씩 처리하고 있을수도 있습니다. 이런 경우, 테스트타임에 무엇을 해야합니다. 조금다르게 말이죠. 예상하는 것들이 말이 되게요. 다음에 있을 마지막 배치 정규화에 대한 비디오에서는 여러분이 아셔야 하는 냉요에 대해, 트레이닝 된 신경망으로 배치 정규화를 이용하여 예상하는 방법을 배우겠습니다.



하지만 테스트타임에서는, 64, 128, 또는 256개의 예시들을 한번에 처리할 수 있는 미니 배치가 없을 수 있습니다. 그렇기 때문에 뮤와 시그마제곱을 구하기 위한 어떠한 또 다른 방법이 필요합니다. 그리고 만약 여러분이 1개의 예시밖에 없는 경우, 이런 1개의 값을 가지고 평균값과 편차를 구하는 것은 말이 안됩니다.

> 테스트타임에서 신경망을 적용한다는 것은 별도의 뮤와 표준편차의 예측값을 구하는 것입니다. 일반적인 배치 정규화의 도입같은 경우엔, 이 것을 추정하는데요, 기하급수적 가중 평균 방식을 이용하여 그 평균은 미니 배치에서의 값으로하여 구합니다.

여기 층의 첫번째 미니 배치에 있는 시그마 제곱값을 트래킹하기위해 그리고 두번째 첫번째 미니 배치에 있는 시그마 제곱값을 트래킹하기위해 등등의 이유로 기하급수적 가중 평균을 사용합니다. 이렇게 뮤과 시그마 제곱값의 평균을 돌립니다. 각각의 층에서 나오는 값들에 대해서 말이죠. 그렇게해서 다른 미니 배치들에 대하여 신경망을 트레이닝 시킵니다. 그리고 최종적으로 테스트타임에서는, 이 공식 대신해서, 어떠한 z값이던 이 값을 이용해서 그냥 z norm의 값을 구하고, 그리고 뮤와 시그마 제곱값에서 기하급수적 가중 평균치를 구합니다.



zL을 산출한다고 해보겠습니다. 그리고 zL은 4차원 벡터이구요 5,2,-1,-3이라고 해보겠습니다. 이제 할 것은 여기 element-wise exponentiation을 이용해서 여기 벡터 t를 산출할 것입니다. 그러면 t는 e의 5승일 것이고, e의 2승, e의 -1승, e의 3승 이런식으로 될 것입니다. 그리고 이것을 계산기로 구하면, 값은 이렇게 됩니다. e의 5승은 1484, e제곱은 약 7.4, e의 -1승은 0.4 , 그리고 e 세제곱은 20.1 그러므로 벡터t에서 벡터 aL로 가는 것은 여기 입력값을 정규화하여 합이 1이 되게 하는 것입니다. 그러면 t의 element들을 합하면 여기 값을 모두 더하면 176.3이 됩니다. 마지막으로 aL은 여기 벡터 t 나누기 176.3입니다.

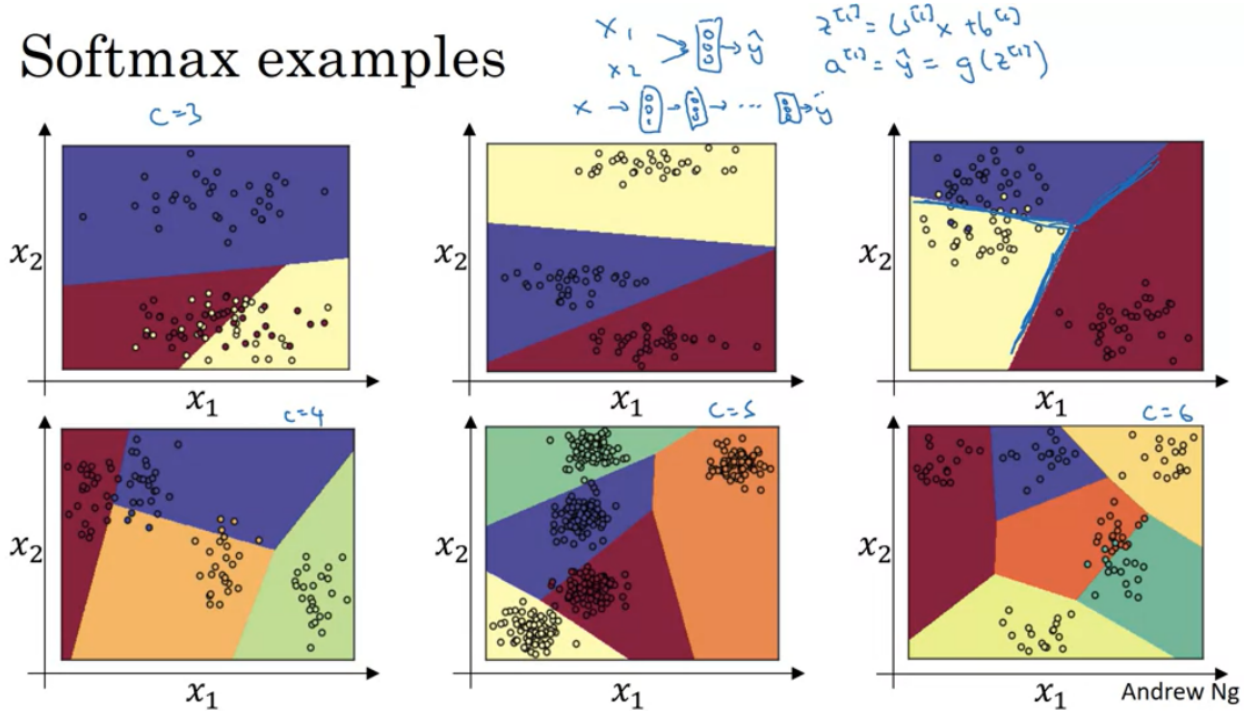
(예를 들어 여기 첫번째 노드는 이것은 e의 5승 나누기 176.3을 줄 것입니다. 그 값은 0.842 입니다. 이 이미지에서는 이것이 z의 값이면 0이라 불릴 확률은 84.2퍼센트 입니다.)

여기 전체 산출 방법에서 exponentiation를 산출해서 일시적 변수 t를 가졌는데요, 그 다음에 normalizing을 통해 이것을 Softmax activation 함수로 요약할 수 있습니다. 그리고 aL = gl (zl) 로 표현할 수 있겠습니다. 이 activation 함수의 조금 특이한 부분은 여기 g activation함수가 4 x 1벡터를 입력값으로 갖고, 4 x 1 벡터를 결과값으로 출력한다는 것입니다.

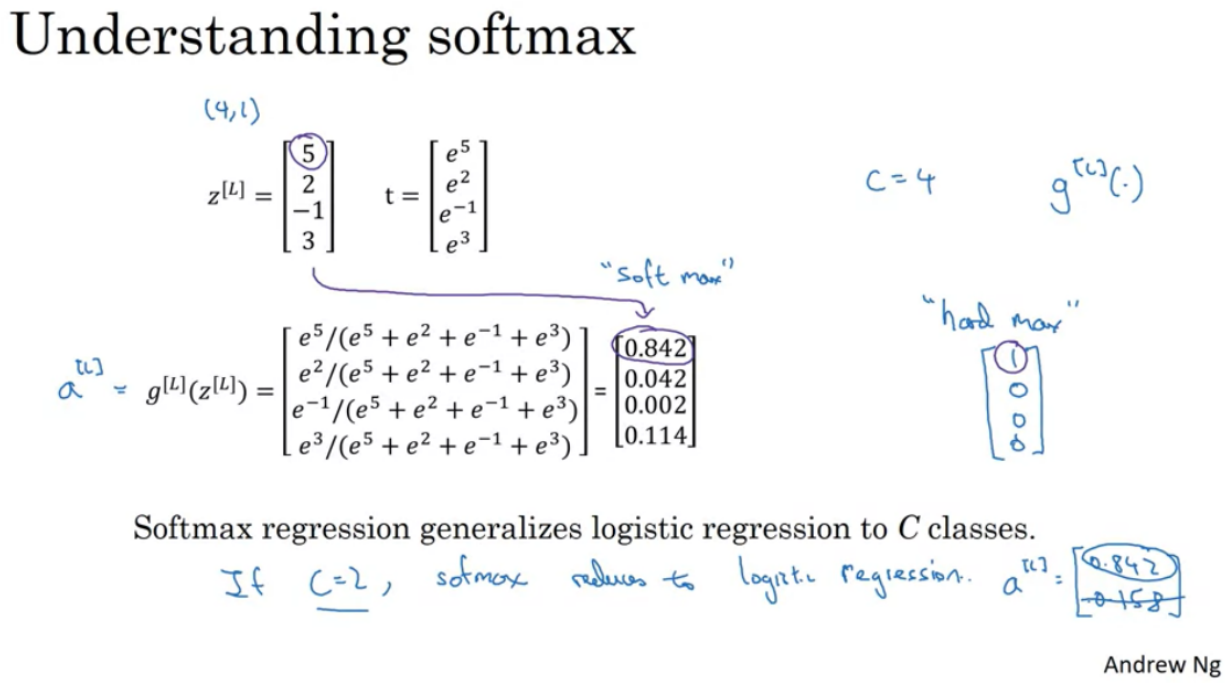
↔이전에는 저희 activation 함수가 single row 입력값을 받았는데요, 예를 들어, 시그모이드 함수와 ReLu activation 함수가 실수를 입력값으로 갖고, 실수를 결과값으로 갖습니다.

> softmax activation 함수의 특이한 부분은 다른 결과값들에 거쳐 정규화가 되야한다는 점인데요 그렇기 때문에 벡터의 값을 갖고, 결과값도 벡터로 가져야 한다는 것입니다.

이제 숨겨진 층이 없는 신경망을 보여드리겠습니다. 이것이 하는 것은 z1 = w1 곱하기 입력값 x 더하기 b입니다. 그리고 결과값 a는 a1 또는 ŷ 은 z1에 적용된 Softmax activation 함수입니다. 이렇게 숨겨진 층이 없는 신경망에서는 Softmax 함수가 대표할 수 있는 것들에 대한 내용을 알려줄텐데요,



> 이렇게 숨겨진 층이 없는 신경망에서는 Softmax 함수가 대표할 수 있는 것들에 대한 내용을 알려줄텐데요,



가장 큰 확률은 여기 첫번때 확률입니다. softmax 라는 이름은 hard max라는 것과 비교한다는 의미에서 만들어졌는데요, 벡터 z와 비교하여 여기 벡터에 맞출 것입니다.

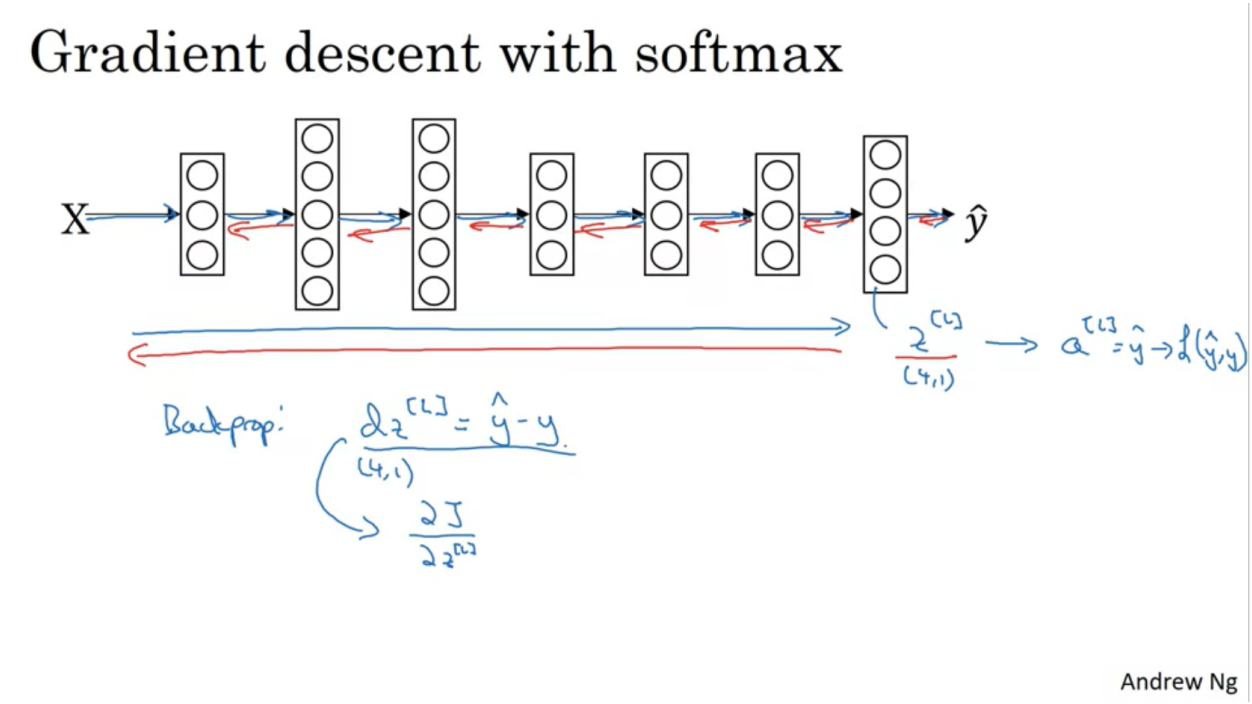
hard max 함수와 같은 경우 z의 요소들을 봐서 가장 큰 요소에 1을 넣고 나머지는 0으로 채울 것입니다. 이것은 매우 hard max 인데요, 가장 큰 요소는 결과값 1을 갖습니다.

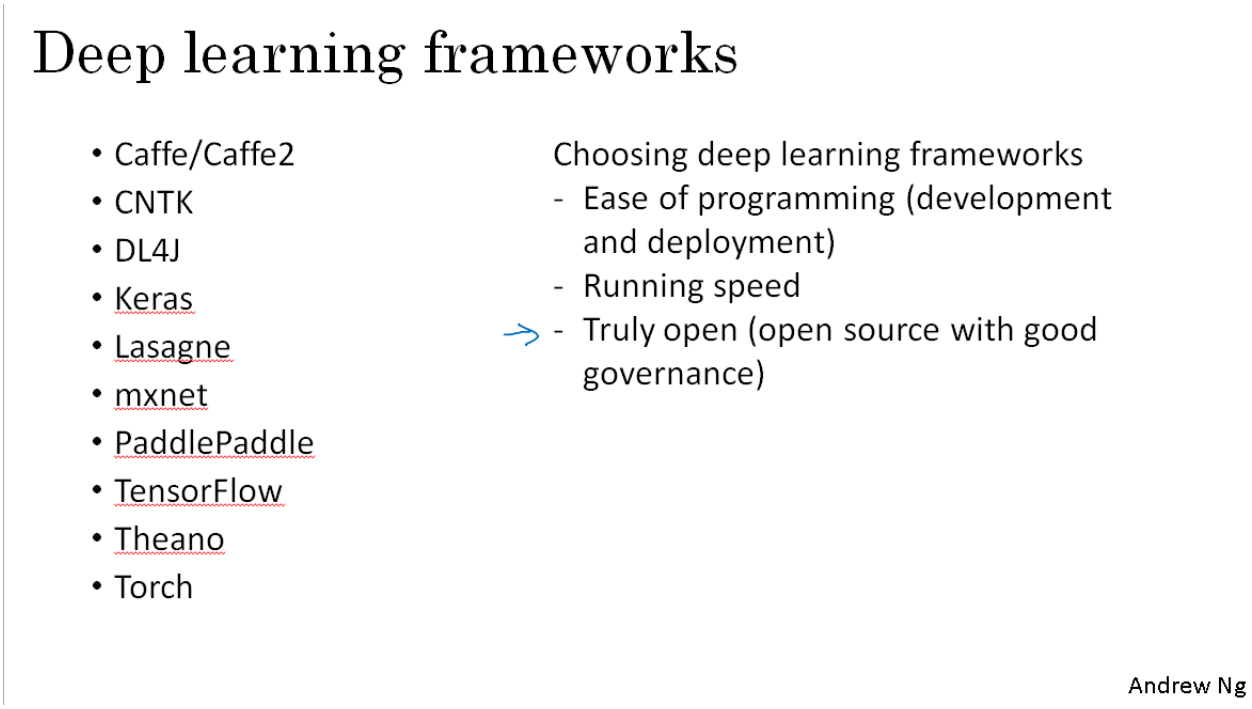
↔softmax는 조금 더 젠틀한 매핑인데요, z에서 여기 확률까지 말이죠.

> 여기서 기억할 것은 softmax regression은 로지스틱 회귀분석의 일반화라는 것입니다. 2개의 이상의 class에 대해서 말이죠.

이것은 한개의 트레이닝 예시에 대한 loss 내용입니다. 전체 트레이닝 세트에서의 j 비용은 어떨까요? 즉, 여기 이런 파라미터들을 세팅하는데 드는 비용을 모든 방법과 bias를 계산하는

방법 중에서, 이것을 여러분이 예상하는 값으로, 모든 트레이닝 세트의 loss의 합을 여러분의 러닝 알고리즘의 예측치를 트레이닝 예시에 거쳐 더한 값입니다.

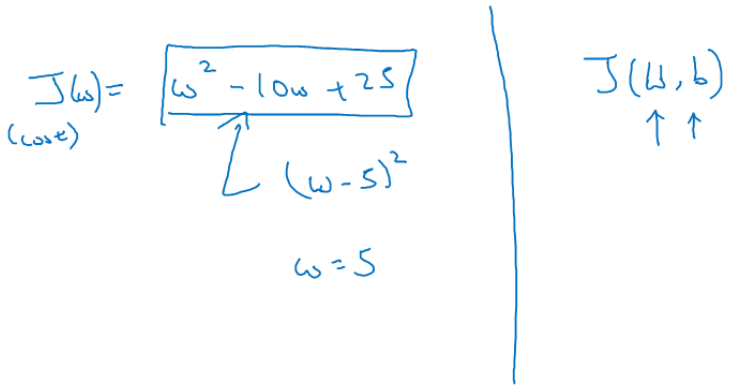




[프레임웍 선택 시 고려할 것]

1. 프로그래밍의 용이함
2. 실행 속도(큰 데이터 셋에서 트레이닝 시)
3. 프레임웍이 완전히 오픈되어있는지

Motivating problem

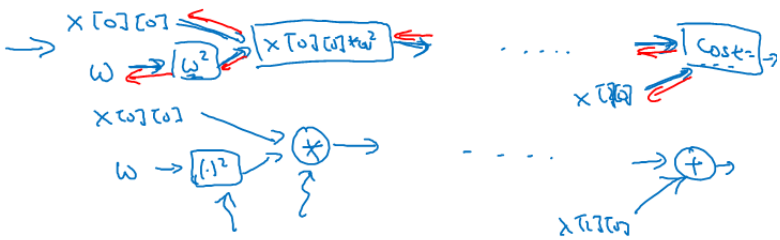


Andrew Ng

여러분이 비용함수 J가 있다고 해보십시요. 최소화하고 싶은 값이지요. 이것을 최소화하기 위해 TensorFlow에 어떻게 도입시킬 수 있는지 알아보겠습니다. 아주 유사한 유형의 프로그램을 이용하여 신경망을 트레이닝 시키고 J(w, b)와 같은 복잡한 비용함수를 갖을 수 있기 때문입니다. 신경망의 파라미터에 따라서 말이죠.

비슷하게, TensorFlow를 이용하여 이 비용함수를 최소화하는 w, b의 값을 자동적으로 찾을 수 있습니다.

Code example



```
import numpy as np
import tensorflow as tf

coefficients = np.array([[1], [-20], [25]])

w = tf.Variable([0],dtype=tf.float32)
x = tf.placeholder(tf.float32, [3,1])
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]    # (w-5)**2
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
print(session.run(w))

for i in range(1000):
    session.run(train, feed_dict={x:coefficients})
print(session.run(w))
```

```
with tf.Session() as session:
    session.run(init)
    print(session.run(w))
```

Andrew Ng