유런11주차

튜닝 프로세스

학습 목표 : 지금까지 신경망을 학습시킬 때 여러 하이퍼파라미터가 관여하는 걸 봤는데 좋은 하이퍼파라미터를 어떻게 효율적으로 구할까?

학습률 알파

모멘텀 베타(0.9)

adam 최적화 알고리즘의 하이퍼파라미터인 베타1(0.9), 베타2(0.999), 엡실론(10^{-8}) - 아담 알고리즘에서는 이것들을 튜닝을 하지 않고 상수들을 항상 사용한다. 하지만 원한다면 튜닝 가능

층의 개수

층의 은닉 유닛 개수

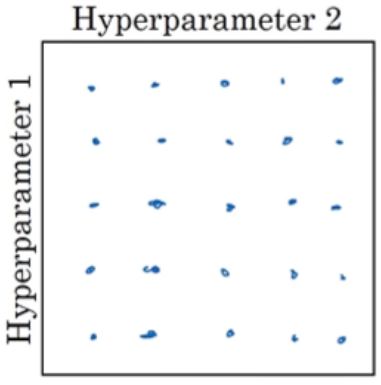
학습률 감쇠

미니 배치 크기

중요도 순: 빨 > 주 > 보 (다른 직관을 갖는 전문가들도 있다)

대부분의 학습에서는 일부 파라미터들이 다른 파라미터보다 중요하다

하이퍼파라미터를 튜닝한다면 어떤 값을 탐색할 지 어떻게 정할 수 있을까?

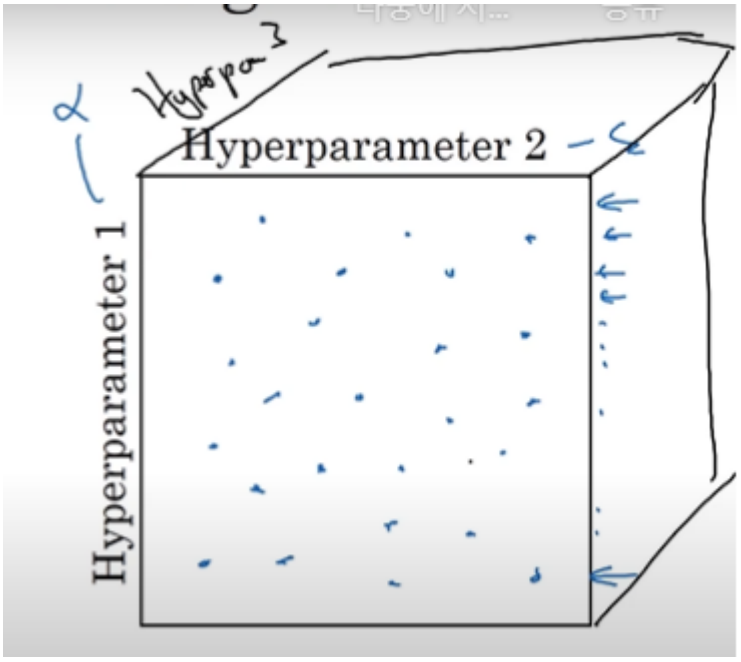


머신러닝이 만들어진지 얼마 되지 않았을 때 두 개의 하이퍼파라미터가 있을 때, 격자점을 탐색하는 것이 일반적이었다. 여기 점들의 값들을 탐색한다. 5*5 격자에서 25개의 점만 생각을 하고 이 중에서 최고의 하이퍼파라미터를 찾는다. 이 방법은 하이퍼파라미터 수가 적을 때 쓸 수 있다.

하지만 딥러닝에서는 균일한 격자점이 아니라 무작이로 점들을 선택한다. 동일하게 25개의 점을 생각한다면 랜덤 점들에 대해서 하이퍼파라미터를 정하는 것이다.

하이퍼 파라미터의 중요도 순위가 있다.

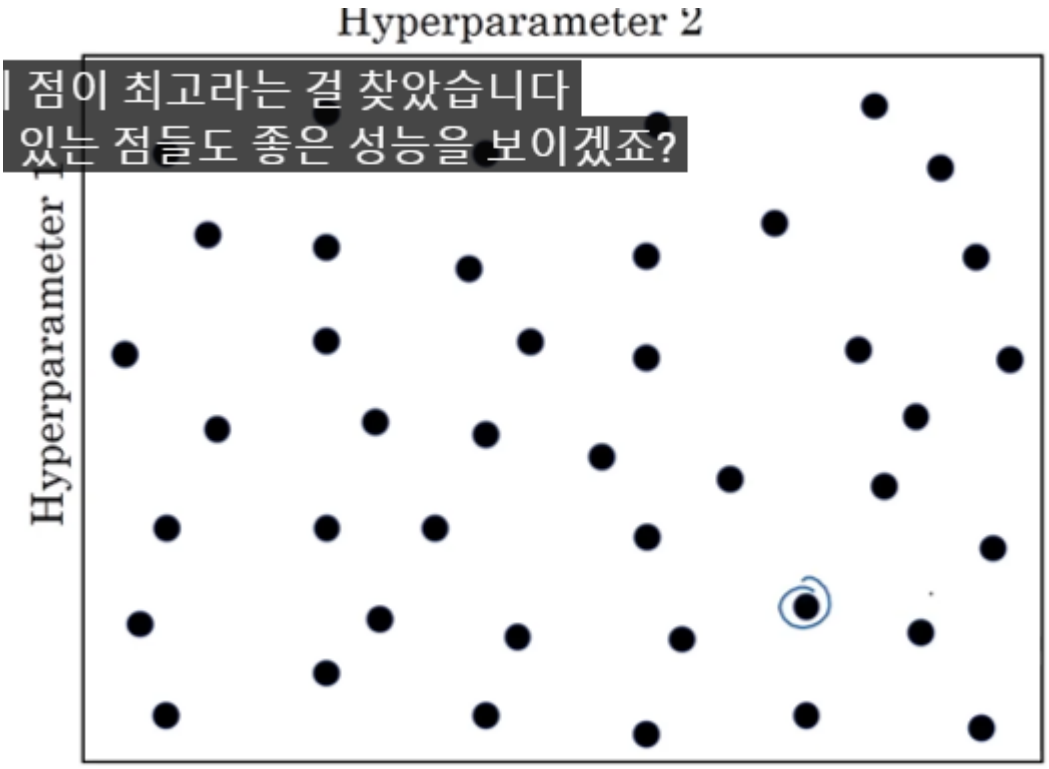
격자점에서 hp1 이 학습률 알파, hp2 가 아담의 엡실론이면 엡실론 값이 달라도 결과는 같은 것을 확인할 수 있을 것이다. 25개의 모델을 학습시켜도 알파 5개에 대해서만 학습시킨 것과 다를 게 없다. 반대로 무작위로 고르면 어떨까? 그러면 25 개의 서로 다른 학습 속도 알파 값을 이용하여 학습시키게 되고 더 좋은 하이퍼파라미터를 잘 찾게 될 것이다.

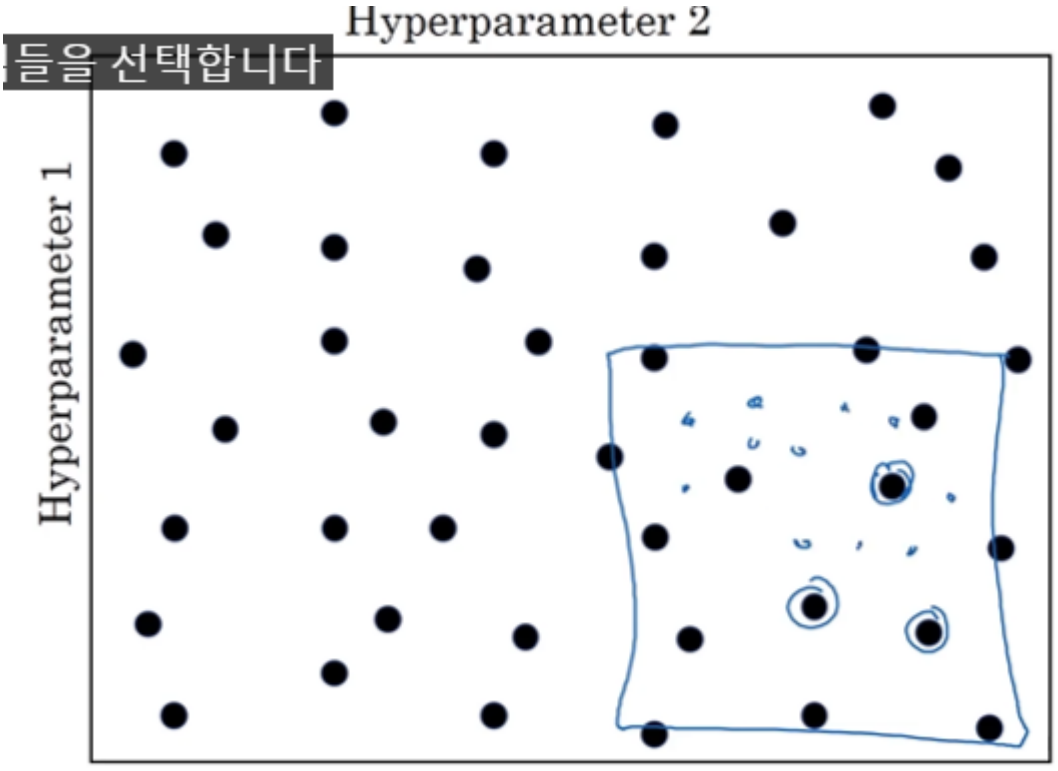


여기서는 두개만 다뤘지만 실제로는 훨씬 많이 다룬다.

그리고 어플리케이션에서 어떤 hp 가 가장 중요한지 미리 알기 어렵다. 격자점보다 무작위로 모델을 정하는 것이 가장 중요한 하이퍼파라미터의 다양한 값을 탐색할 수 있다.

다른 일반적 방법 중 하나는 정밀화 접근이다.





이 점들 중에 특정 점이 최고라는 것을 알았다면 그 주변에 있는 점들도 좋은 성능을 보일 것이다. 더 작은 영역으로 확대해서 더 조밀하게 점을 선택한다. 무작위인 것은 그대로이다. 파란색 사각형 안에 초점을 두고 생각한다. 사각형의 범위를 점차 좁혀나간다.

이렇게 여러 값들을 시험해보며 학습의 목표나 개발 목표에 따라 최고의 파라미터를 고른다.

정리:

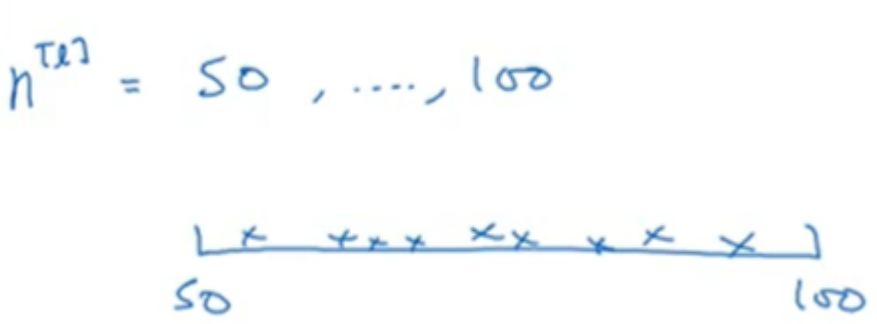
1. 무작위로 하이퍼파라미터를 찾는 것이 더 효율적인 탐색이다
2. 원한다면 정밀화 기법을 사용할 수 있다.

적절한 척도 선택하기

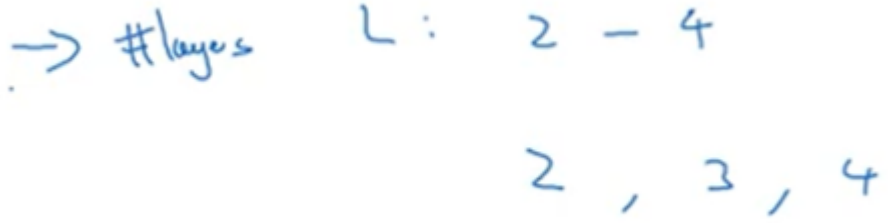
학습 목표: 무작위로 hp 를 찾는 것이 더 효율적인 탐색이지만 가능한 값들 중 공평하게 뽑는 것이라고 할 수는 없다. 대신 적절한 척도를 정하는 것이 중요하다. 어떻게 적절한 척도를 정할까?

어떤 layer 에 대해서 은닉 유닛의 수 n 을 정한다고 해보자.

그리고 n 은 50~100 을 범위로 생각한다.

1. 수직선 상에 무작위하게 값을 고른다고 생각해보자. 꽤 합리적인 방법임
2. 

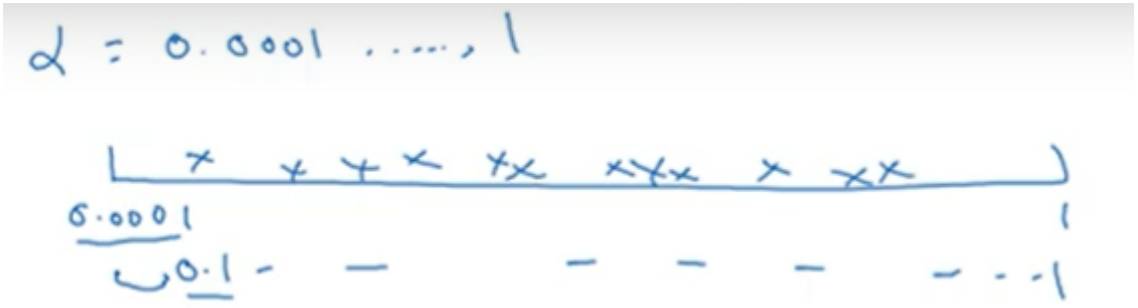
2. 신경망에서 레이어의 숫자 L을 정한다고 했을 때, 층의 숫자가 2~4 사이라고 생각할 수 있다. 2~4숫자를 선택할 때 무작위하게 뽑는 것은 물론, 격자점을 사용해도 문제가 없다.



위 2개의 경우는 가능한 값 중 무작위하게 뽑는 것이 합리적인 경우들이다.

하지만 모든 hp 가 그렇진 않다.

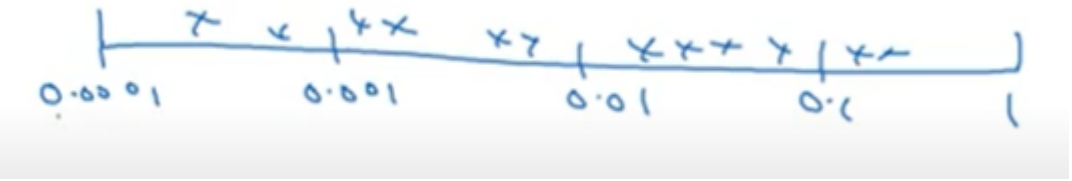
1. 학습률 알파를 찾는다고 해보자. 범위는 0.0001~1 이다.



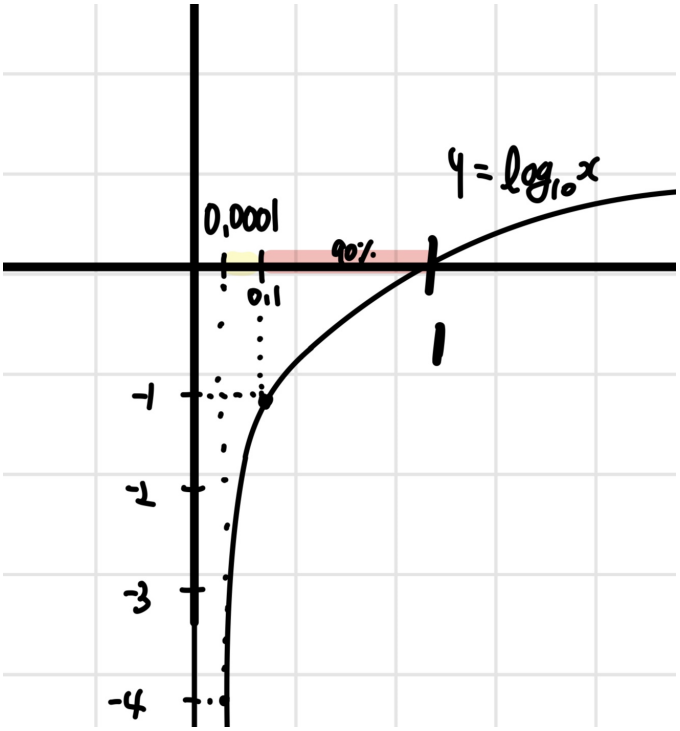
수직선 상에서 균일하게 무작위로 값을 고를 것이다. 약 90%의 샘플이 0.1과 1 사이에 있을 것이다. 따라서 90%의 자원을 0.1 과 1 사이를 탐색하는 데 쓰고 단 10%를 나머지를 탐색하는데 쓴다.

비합리적임.

로그 척도에서 샘플링 하는 법



따라서 선형 척도 대신 로그 척도에서 hp 를 찾는 것이 더 합리적이다. 그러면 0.0001 과 0.001 사이와 0.001과 0.01 사이를 탐색할 때, 더 많은 자원을 쓸 수 있다.



파이썬에서는 어떻게 구현할까?

r= -4*np.random.rand() 를 쓴다.

- np.random.rand() : 0~1 의 값을 균등하게 추출, 소수점 17 자리까지 표현 가능 약 4.5 조 가지의 숫자 추출

그러면 무작위로 선택된 알파값은 10^r이 된다.

r= -4*np.random.rand()

⇒ r은 -4 와 0 사이의 무작위 값이다.

⇒ 알파 값은 10^{-4} ~ 10^0 이 된다.

더 일반적인 경우,

10^a 에서 10^b 까지를 로그 척도로 탐색한다면, (위 예시에서는 0.0001 이 10^a 가 된다) 따라서 a 는 10을 밑으로 하는 log 를 0.0001 에 대해 취하면 얻을 수 있다. 그 결과 a 는 -4가 된다. 마찬가지로 오른쪽 값도 10^b 가 된다. 위에서는 (b= $log1$ = 0)

1. 낮은 값에서 log 를 취해서 a 를 찾고
2. 높은 값에서 log 를 취해 b 를 찾는다.
3. 이렇게 10^a 에서 10^b 까지를 로그 척도로 탐색한다

4. r 은 a 와 b 사이에서 균일하게 무작위로 뽑으면 하이퍼파라미터가 10^r 이다.

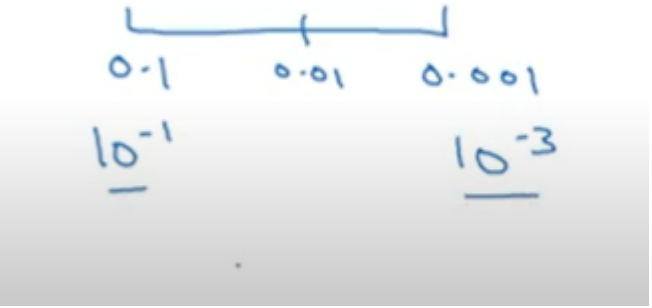
지수가중평균을 계산할 때 사용되는 하이퍼파라미터 베타에 관한 것

베타를 0.9 와 0.999 사이에서 찾는다고 해보자.

0.9의 경우에는 지수가중평균이 최근 10일의 평균 기온처럼 마지막 10개 값의 평균과 비슷하고 0.999 경우에는 마지막 1000개 값의 평균과 비슷하다.

방금 전의 경우처럼 0.9와 0.999 사이의 값을 균일하게 무작위 탐색하는 것은 합리적이지 않다.

더 나은 방법은 1-베타에 대해서 값을 탐색하는 것이다. (0.1~0,001) 이 된다.



그러면 베타를 0.1 에서 0.01 을 거쳐 0.001 사이에서 탐색하는 것이다. 여기서는 큰 값이 왼쪽, 작은 값이 오른쪽에 온다.

여기서 해야 할 일은 -3과 -1 사이에서 균일하게 무작위로 값을 뽑는 것이다.

즉, 1-베타 를 10^r 로 생각하니 베타는 $1 - 10^r$ 이 된다.

따라서 적절한 척도 위에서 무작위로 hp 샘플을 추출할 수 있다.

이 방법을 이용하면 0.9부터 0.99를 탐색할 때와 0.99에서 0.999 를 탐색할 때 동일한 양의 자원을 사용할 수 있다.

이 방법이 왜 필요할까에 대한 수학적 증명

왜 선형척도에서 샘플을 뽑는 것은 안 좋을까?

⇒ 만약 베타가 1에 가깝다면 베타가 조금만 바뀌어도 결과가 아주 많이 바뀌게 된다. 예를 들어 베타가 0.9 에서 0.9005로 바뀌었다면, 결과에 거의 영향을 주지 않는다.

하지만 베타가 0.999 에서 0.9995로 바꾸면 알고리즘의 결과에 큰 영향을 준다. 왜냐하면 첫번째 경우는 대략 10개의 값을 평균내지만 두번째거는 마지막 1000개 값의 지수가중평균을 내는 것에서 마지막 2000개 값의 평균을 내는 것으로 바뀌기 때문이다. (1-(1-베타) 는 베타가 1에 가까워질수록 작은 변화에도 민감하게 반응하기 때문)

따라서 베타가 1보다 가까운 곳에서 더 조밀하게 샘플을 뽑는다. 따라서 가능한 결과 공간을 탐색할 때 더 효율적으로 샘플을 추출할 수 있다.

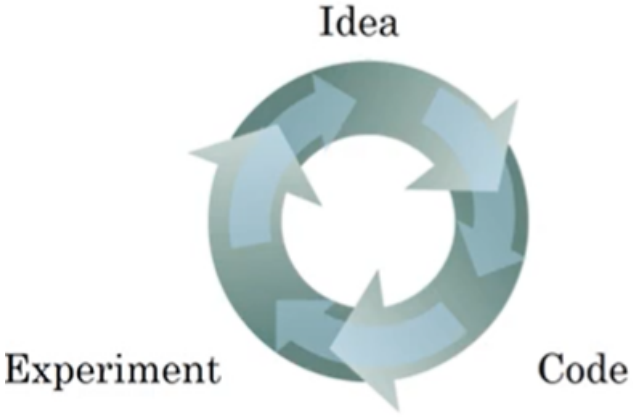
다른 척도가 우선하는 상황에서 균일한 척도에서 샘플링을 하더라도 정밀화 접근을 사용하면 괜찮은 결과를 얻을 수 있다.

하이퍼파라미터 튜닝 실전

학습목표 : 하이퍼파라미터 탐색을 어떻게 할 수 있는지 몇 가지 팁을 안다.

한 어플리케이션에서 얻은 하이퍼파라미터에 대한 직관이 다른 영역에서 쓰일 수도, 아닐 수도 있다.

서로 다른 어플리케이션 영역 간에 공유되는 것들이 있다. 예를 들면 컴퓨터 비전 커뮤니티에서 발전된 cnn 이나 resnet 이 있다. 이런것들이 음성에 잘 적용되고 있고 여기서 발전된 아이디어들은 자연어 처리에서도 잘 적용된다.



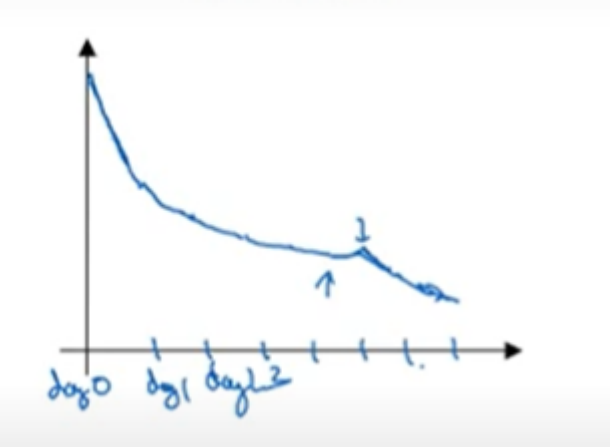
- NLP, Vision, Speech, Ads, logistics, ...

따라서 딥러닝 분야의 사람들이 다른 영역에서 영감을 얻기 위해 다른 분야의 논문을 찾아본다.

하지만 하이퍼파라미터는 그렇지 못한다는 직관을 얻어왔다. 로지스틱 문제만 보더라도 좋은 hp 를 찾았다고 할 때, 알고리즘을 발전시키거나 데이터가 바뀔 수도 있고 서버를 업그레이드 시킬 수도 있다. 이러한 변화들 때문에 hp 가 녹슬기 떄문에 다시 시험해보거나 hp 들이 아직도 만족할만한 결과를 내는지 주기적으로 재평가하는 것을 권장한다.

결국 사람들이 hp를 찾을 때, 두가지 서로 다른 방법을 사용한다.

1. 모델 돌보기(판다 전략)



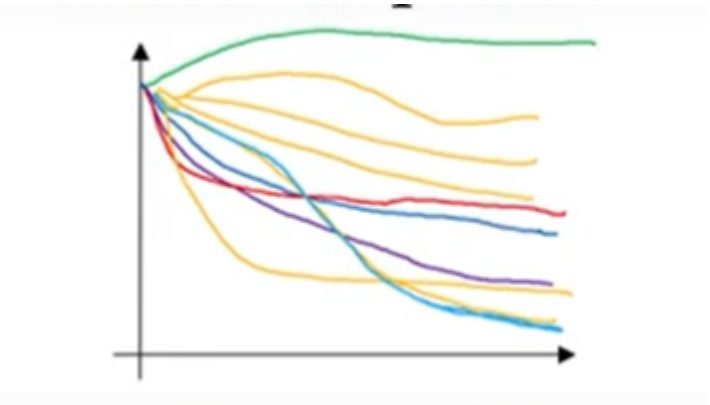
데이터는 방대하지만 컴퓨터 자원이 많이 필요하지 않아 적은 숫자의 모델을 한번에 학습시킬 수 있을 때 사용한다. 이 경우에 학습 과정에서 모델 돌보기를 한다.

예를 들어 0일 차에 무작위하게 매개변수를 설정하고 학습을 시작했다. 그러면 학습곡선에서 비용함수 j 나 개발세트의 오차가 1일차 끝 무렵에 점점 줄어들 것이다. 그러면 학습 속도를 조금 올려서 조금 더 나은지 확인한다. 이런 식으로 관찰을 하다가 잘 줄어들면 모멘텀을 약간 올리거나 학습 속도를 약간 낮춰서 결과를 지켜본다. 이런 식으로 hp 를 계속 조절하다 보면 어떤 날에는 학습 속도가 너무 커서 몇 일 전으로 돌아가기도 할 것이다. 이렇게 매일 성능을 잘 지켜보다가 학습 속도를 조금씩 바꾸면서 모델을 돌보며 학습시킨다. 2,3 주 후에 다른 모델을 초기화해서 다시 돌보기도 가능하다.

따라서 여러 모델을 동시에 학습시킬 컴퓨터 자원이 충분치 않을 때도 사용이 가능하다.

2. 여러 모델을 함께 학습시킨다.(캐비어 전략)

갖고 있는 hp 를 며칠에 걸쳐 스스로 학습시킨다. 동시에 다른 모델의 다른 hp 설정을 다루기 시작한다. 그리고 동시에 세번쨔 모델도 학습시킨다. 다른 여러 모델을 동시에 학습시키면 여러 hp 설정을 확인할 수 있다. 그리고 마지막 날에는 최고 성능을 보이는 것을 고른다. 즉, 하나에 집중하기 보다는 하나 그 이상이 잘 살아남기를 바라며 지켜본다.



⇒ 여러 모델을 학습시키기에 충분한 컴퓨터 성능을 가지고 있다면 캐비어를, 온라인 광고나 cv app 등 많은 데이터가 쓰이는 곳은 학습시키고자 하는 모델이 너무 커서 한 번에 여러 모델을 학습시키기 어려워 판다 전략을 쓴다.

하이퍼파라미터에 상관없이 튼튼한 신경망을 만드는 방법? 있긴 하다. 적용이 가능하다면 hp 탐색이 훨씬 쉽고 빨라진다. 이건 다음 시간에 ~

배치 정규화

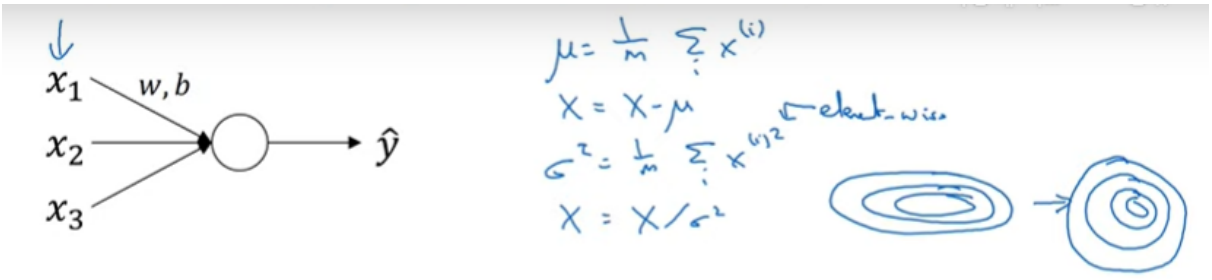
딥러닝이 떠오르면서 가장 중요한 아이디어 중 하나로 배치 정규화 알고리즘이 떠올랐다.

배치 정규화는 hp 의 탐색을 쉽게 만들어줄 뿐만 아니라 신경망화 hp 의 상관관계를 줄여준다.⇒ 더 많은 hp 가 잘 작동한다. 아주 깊은 심층신경망이라도 아주 쉽게 학습될 수 있다.

학습 목표 : 배치 정규화는 어떻게 작동하는가?

전에 로지스틱 회귀 등으로 모델을 학습시킬 때 입력 변수들을 정규화하면 학습이 빨라졌다. 평균을 계산할 때는 입력 변수의 평균을 뺐고, 분산을 계산할 때는 각 입력 변수에 제곱을 취해줬다.

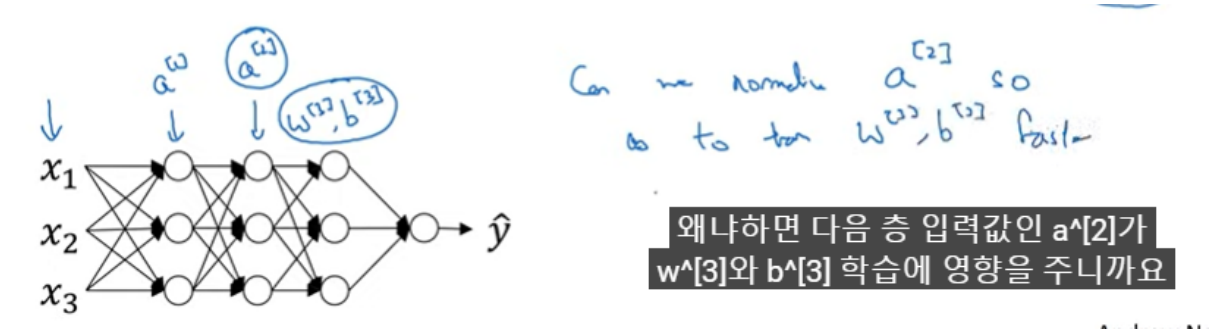
구한 평균과, 분산으로 입력값을 정규화 해줬다.



즉 로지스틱 회귀 등 신경망의 입력 변수들을 정규화하면 등고선 모양이 저렇게 바뀐다.

심층 신경망은 어떨까?

여기서는 입력 변수 x 뿐만 아니라 활성값들도 존재한다. 활성화 값의 평균과 분산을 정규화 하는 것이 w, b 를 구하는데 더 효율적일 수 있다.



예를 들어 다음층 입력값인 a[2] 가 w[3], b[3] 학습에 영향을 주기 때문에 a[2] 를 정규화 하는 것이 더 효율적일 수 있다.

이것이 배치 정규화가 하는 일인데 사실 a[2] 를 정규화한다기 보다는 z[2] 를 정규화하는 것이다. z[2], 를 정규화할 것인지, 활성화 함수 이후의 값인 a[2] 를 정규화할 것인지에 대한 논쟁은 존재한다.

전자가 더 많이 쓰이긴함. 교수님도 활성화 이전 값을 정규화하는 방법을 주로 사용하신다고 하심

어떻게 구현할까?

신경망에서 사잇값들이 주어졌다고 할 때, 은닉 유닛의 값 z(1) ... z(2) 이 있을 때, 이 값들에 대해 평균과 분산을 구하여 z(i)_norm 을 구한다. 또한 수학적 안정성을 위해서 분모에 ϵ 값을 추가한다.(혹시 표준편차가 0인 경우를 대비해서)

이렇게 z 값에 정규화를 거쳐 평균이 0, 표준편차가 1 이 되도록 만들었다. 하지만 은닉 유닛이 항상 평균 0, 분산 1 을 갖는 것이 좋지만은 않다.

왜냐하면 은닉 유닛은 다양한 분포를 가져야하기 때문이다.

그래서 대신 z~ 를 구한다. 이건 정규화된 값에 γ 를 곱하고 β 를 더한다. 감마와 베타는 모델에서 학습시킬 수 있는 변수이다. 즉 여기서 모멘텀, rmsprop, adam 을 이용한 경사하강법을 이용해서 다양한 알고리즘으로 학습시킬 수 있다.

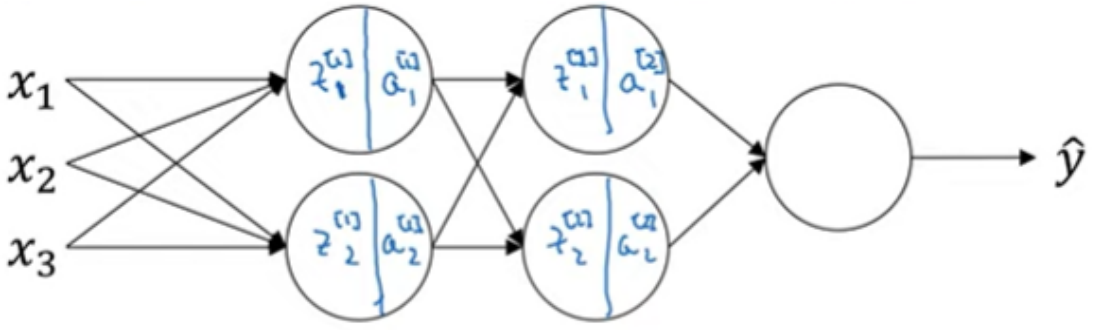
감마와 베타를 이용하면 z~ 의 평균을 원하는 대로 설정할 수 있다.

직관: 배치 정규화는 입력층에만 정규화를 하는 것이 아니라, 신경망 안 깊이 있는 은닉층의 값도 정규화흘 한다. 은닉 유닛 z 의 평균과 분산을 정규화한다. 하지만 입력층과 은닉 유닛을 학습시킬 때의 차이점은 은닉 유닛 값의 평균과 분산이 0, 1로 고정되기를 원치 않는다는 것이다. 예를 들어 시그모이드 활성 함수가 있을 때 값들이 한쪽에 모여있는 것을 원치 않기 때문이다. 더 넓은 영역에 펼쳐 있거나 시그모이드의 비선형성을 살릴 수 있도록 평균이 0 이 아닌 다른 값을 갖게 하는 것이 좋다.

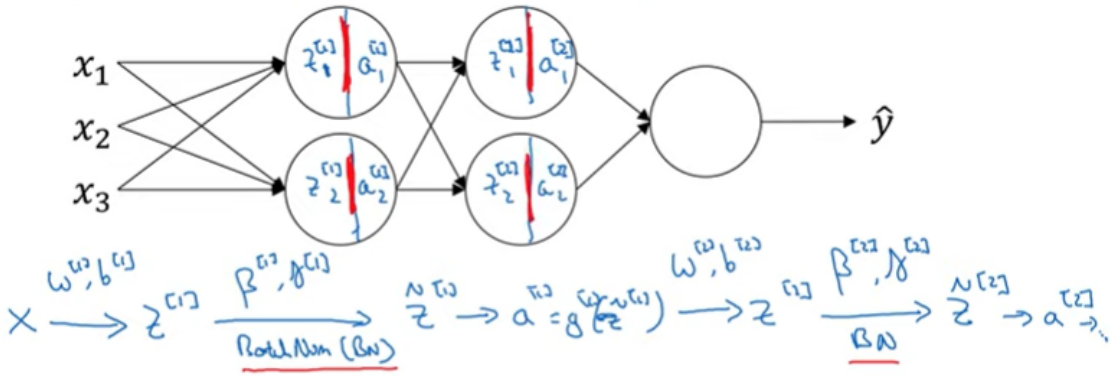
이것이 감마와 베타를 이용하여 원하는 범위의 값을 만들어내는 이유이다. 즉 은닉 유닛이 표준화된 평균과 분산을 갖되, 특정한 평균과 분산은 학습 알고리즘에서 설정할 수 있는 두 변수 감마와 베타에 의해 조절되는 것이다.

배치 정규화 적용시키기

학습 목표: 여러개의 은닉층이 있는 심층 신경망 학습에 배치 정규화가 어떻게 적용되는지 보자.



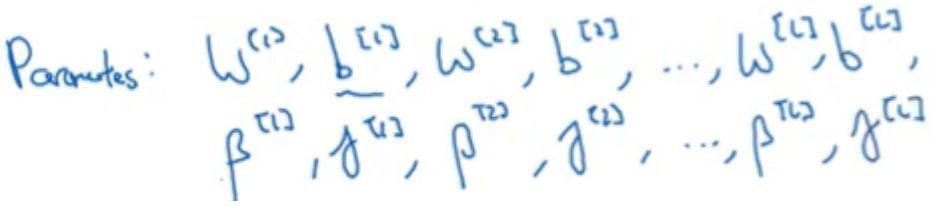
배치 정규화를 하지 않는 다면 입력값은 위와 같다.



배치 정규화는 z 와 a 를 계산하는 사이에 이루어지는데(빨간색으로 친 부분) 이를 계산하면 아래와 같이 신경망 input 값을 나타낼 수 있다. 각각은 각 layer 의 감마와 베타 변수에 의해 결정된다.

즉, 신경망에서 변수들이 w[1], b[1] 이 있는 것을 알 수 있는데 b[1] 가 없어지는 걸 볼 수 있따!

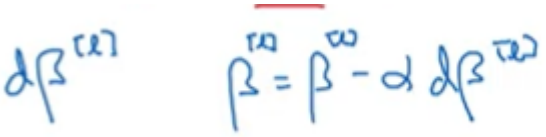
(본래 bias 의 역할?)



여기의 베타는 아담, rms prop 의 베타 변수와 다르다

이제 이 변수들이 알고리즘에 쓰이는데 변수를 찾기 위해 경사 하강법 등 어떤 최적화를 쓸지 고민할 차례이다.

어떤 층에서 $d\beta^{[L]}$ 을 계산했다고 해보자. 그리고 β 를 가중치 업데이트 했다고 해보자. (adam, rmsprop 모멘텀 등을 써서 베타, 감마를 업데이트 할 수도 있다.)



지금까지 경사 하강법을 이용해서 배치 정규화로 전체를 학습 시켰는데, 사실 배치 정규화는 미니 배치에 적용 된다. 실제로는 첫번 째 미니배치에 대해서 z[1]를 구하고, 미니 배치 안에서 z[1] 의 평균과 분산을 계산한 뒤에, 평균을 빼고 표준편차로 나눠 배치 정규화를 진행한다. 베타[1], 감마[1] 을 이용해서 값을 조정한다. 이 과정을 통ㅎ z~[1] 을 얻게 되고, 활성화 함수를 적용해 a[1] 을 얻는다. 그리고 z~[2] 차례대로 얻은 다음, 첫 번째 미니 배치에 대해 경사하강법을 이용해서 과정을 마쳤으면, 두 번째 미니배치로 넘어간다. x{2} 에 대해 비슷한 방법으로 z[1] 를 구하고 배치 정규화를 해서 z~[1]를 한다. 각각의 미니배치의 데이터만을 이용해서 평균과 분산을 얻고 베타와 감마로 보정해서 z~ 를 얻는다.

각 층에 대해 w[l], b[l]이 있고, 베타[l], 감마[l] 이 있다고 하였다. 여기서 z[l] 은 w[l]*a[l-1]+b[l] 로 계산이 된다.

여기서 배치 정규화는 미니 배치를 보고 z[l] 이 평균 0, 분산 1이 갖도록 정규화한 뒤, 베타와 감마를 이용하여 값을 조정해주는 것이다.

여기서 b[l]은 값이 무엇이든지 간에 없어진다. 왜냐하면 배치 정규화의 정규화 과정에서 z의 평균을 계산한 뒤에 빼주기 때문이다. **(먼 소리져)**

즉 미니 배치의 모든 예시에 상수를 더해줘도 결국 평균을 빼주면서 사라지기 때문에 b[l]은 아무런 영향을 끼치지 않는다. 따라서 배치 정규화를 쓴다면 이 변수를 없앨 수 있다.

즉, z[l]=w[l]*a[l-1] 이 된다. 그리고 z[l]_norm 을 계산하고 z~ = gamma[l] * z[l]_norm + beta[l] 계산한다. 여기서 다음 층에 전달되는 z~ 의 평균을 정하기 위해 beta[l] 은 써줘야 한다.

정리하자면, 배치 정규화가 z[l] 의 평균을 0으로 만들기 때문에 b[l] 이 필요없고 beta[l] 이 역할을 대신한다. beta[l] 은 편향 변수를 결정하기 때문에

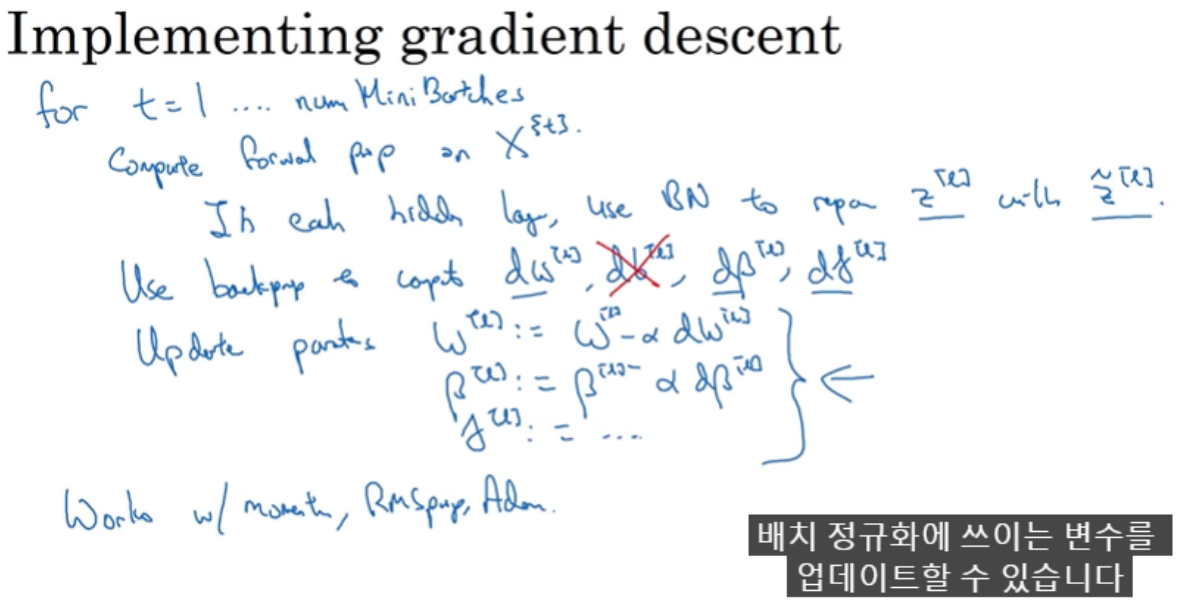
차원:

- z[l] : (n[l](l번 layer 의 은닉 유닛 개수),1) = b[l] = beta[l] = gamma [l]

⇒ 어떤 신경망이든지 간에 n[l] 개의 은닉 유닛을 갖고 있으면 beta[l], gamma[l] 이 각 은닉 유닛의 값을 조정하는데 쓰이기 때문이다.

배치 정규화를 사용하여 경사하강법을 구현하는 법

알고리즘



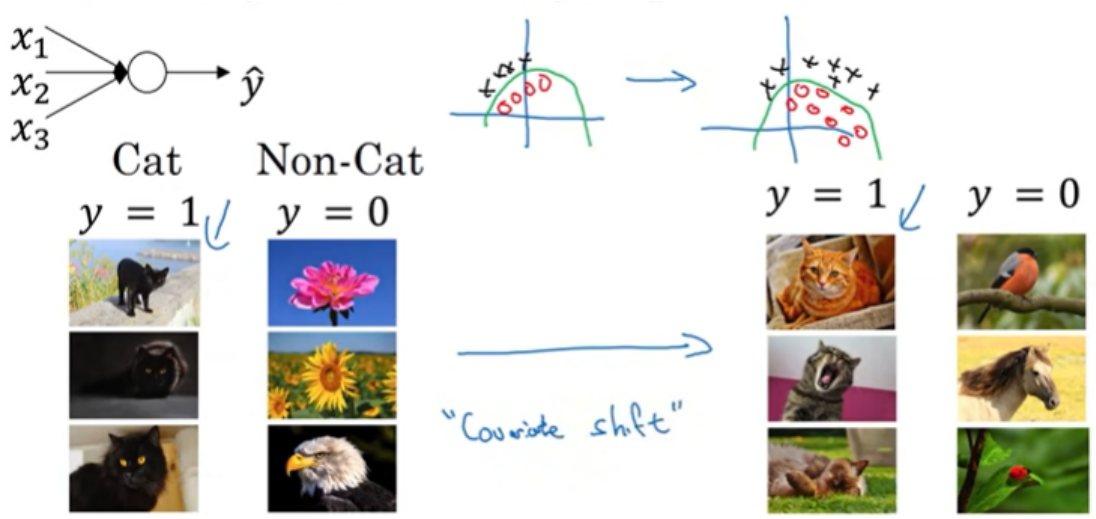
그럼 왜 배치 정규화가 학습을 빠르게 만들어주고 과연 무슨 일을 하는 것일까?

배치 정규화가 잘 작동하는 이유는 무엇일까?

학습 목표: 입력 특성 X를 평균 0, 분산 1로 정규화 하는 것이 학습 속도를 올린다. 어떤 특성은 0~1 값을 갖고 어떤 특성은 1에서 1000 사이의 값을 가질 때, 입력 특성 x 에 대해 비슷한 범위를 갖도록 정규화하여 학습 속도를 높인다. 그래서 배치 정규화가 작동하는 이유가 은닉 유닛과 입력층 모두에서 비슷한 일을 하기 때문이라는 직관은 얻었었다. 하지만 이것 외에도 배치 정규화가 뭘 하는 것인지 더욱 깊이 이해할 수 있는 몇 가지 직관을 살펴본다.

심층 신경망에서 10 번 layer 의 가중치가 1번 layer 처럼 앞쪽 층의 가중치 변화에 영향을 덜 받는다.

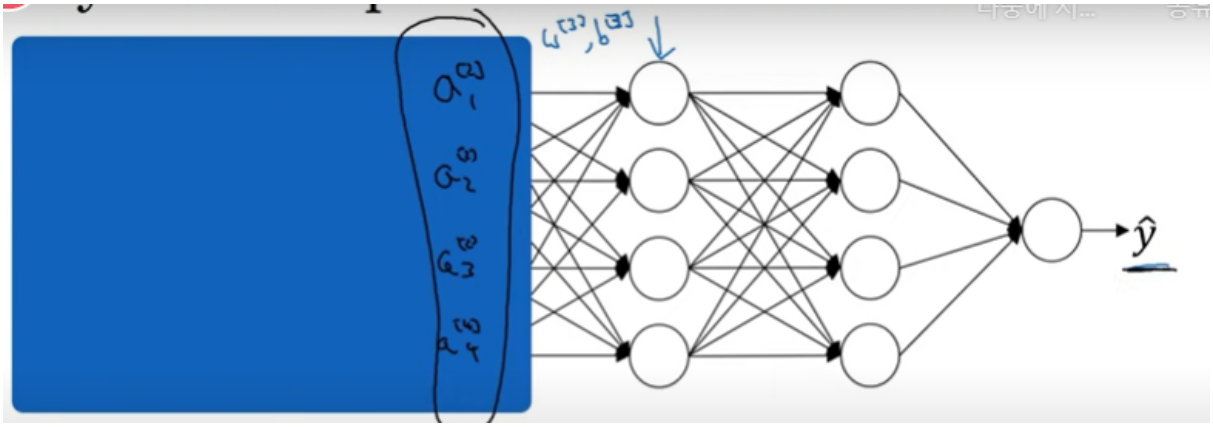
예를 들어 개 고양이 분류하는 신경망이 있다고 해보자. 검정 고양이만 학습을 시켰다면 색깔 고양이가 들어오면 신경망이 좋은 성능을 내지 못한다.



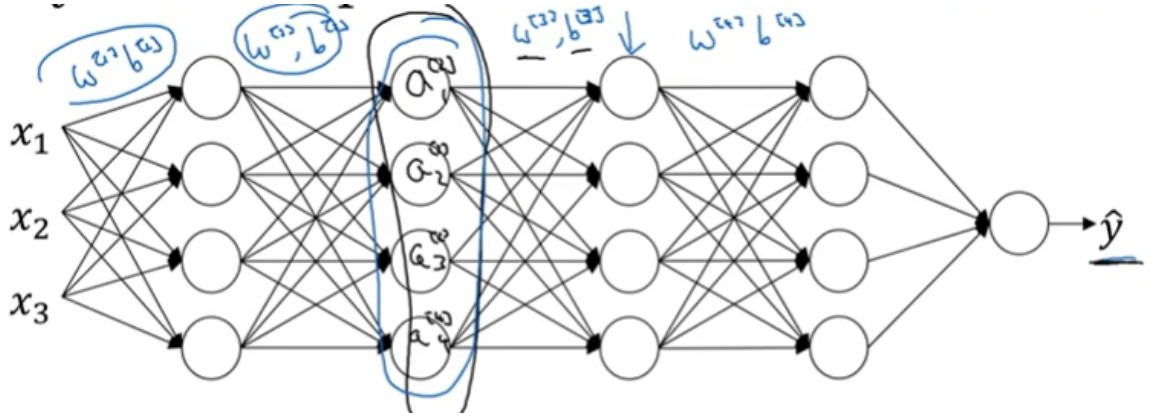
데이터 분포가 변하는 것을 covariate shift 라고 한다.

이 아이디어는 x.y 간의 대응을 학습시킬 때, x 의 분포가 바뀐다면 x, y 로 대응하는 관측 함수가 바뀌지 않더라도 학습 알고리즘을 다시 학습시켜야한다. **(?)**

새롭게 함수를 학습시키면 더 정확해지거나 관측 함수가 함께 움직여서 더 나빠질 수도 있다. 그럼 공변량 변화가 신경망에 어떤 변화를 줄까?



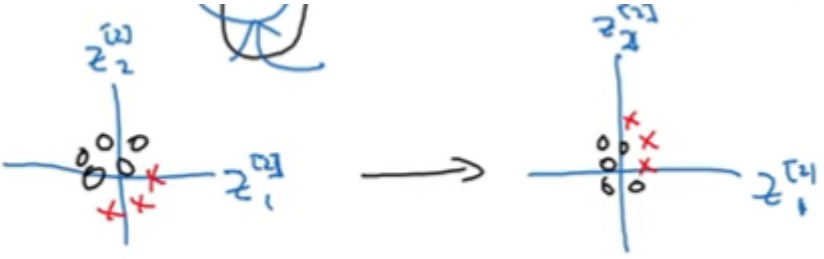
세번째 신경망 전의 신경망은 파란색으로 가렸다. 이때 세번째 레이어로 들어오는 인풋(a) 값을 이용하여 y hat 에 대응을 시켜야한다. 이 때 신경망 성능이 잘 나오도록 w[3],b[3], w[4],b[4], w[5], b[5] 를 학습시켜야한다. 왼쪽의 가려진 값들로부터 y hat 을 대응시키는 것이다.



가림막을 벗기면 w[2], b[2] , w[3], b[3] 에 따라 a 값들이 바뀌는 것을 알 수 있다. 따라서 세 번째 은닉층의 관점에서 이 은닉층 값들을 계속 변한다 . 그러면 공변량 변화의 문제를 계속 겪게 된다.

여기서 배치 정규화는 은닉층 값들의 분포가 변화하는 양을 줄여준다.

이 은닉층들의 값의 분포를 그린다면,



배치 정규화는 z 가 얼마나 바뀌든 간에 z[2]_1, z[2]_2 의 평균과 분산이 동일하게 유지될 것이라는 것을 말한다. 따라서 값이 바뀌더라도 적어도 평균과 분산은 유지될 것이라는 것이다. (베타[2], 감마[2])

즉, 배치 정규화가 하는 일은 앞선 층에서의 매개변수가 바뀌었을 때, 세 번째 층의 값이 받아들여서 학습하게 될 값의 분포를 제한하는 것이다. 즉 배치 정규화는 입력값이 바뀌어서 발생하는 문제를 더욱 안정화시킨다. 딸서 뒤쪽 층은 더 쉽게 학습할 수 있을 것이다. 앞쪽 층이 계속 바뀌더라도 뒤쪽 층이 그것 때문에 겪는 부담을 줄인다. 그리고 이것은 또 앞쪽 층의 매개변수와 뒤쪽 층의 매개변수 간의 관계를 약화시킨다. 따라서 신경망의 각 층이 다른 층과 상관없이 스스로 배울 수 있게 된다. 이걸로 전체 신경망의 학습 속도를 향상시킨다.

정리하자면, 평균과 분산이 일정하도록 제한시키면서 뒤쪽 의 층 관점에서 앞선 층이 너무 많이 변화하지 않는다.

두번째 효과: regularization effect

각각의 미니배치 x{t} 가 가진 z[l] 에 대해서 그 미니 배치의 평균과 분산에 따라 값을 조정할 것이다. 여기서 미니 배치에서 계산한 값은 전체 데이터로부터 계산한 것에 비해 잡음을 갖고 있다. 상대적으로 작은 데이터에 대해서 추정한 것이기 때문이다.

z[l]~z^[l] 에도 잡음이 낄 수 밖에 없다. 즉 드롭아웃처럼 **(?)** 은닉층의 활성 함수에 잡음이 끼어있다. 드롭아웃에서는 은닉층을 가져와서 확률에 따라 0을 곱하거나 1을 곱했다. 하지만 배치 정규화는 표준편차로 나누기 때문에 곱셈 잡음도 있고 평균을 빼니 덧셈 잡음도 있다. 그래서 배치 정규화는 드롭아웃처럼 약간의 일반화 효과를 가지고 있다. 왜냐하면 은닉층에 잡음을 추가하는건 이후의 은닉층이 하나의 은닉층에 너무 의존하지 않도록 하기 때문이다. 잡음이 엄청 작아서 효과가 크진 않기 때문에 드롭아웃을 같이 써도 된다.

드롭아웃에서 큰 미니 배치를 사용한다면,, 잡음이 줄어들고 일반화 효과도 줄어들 것이다.

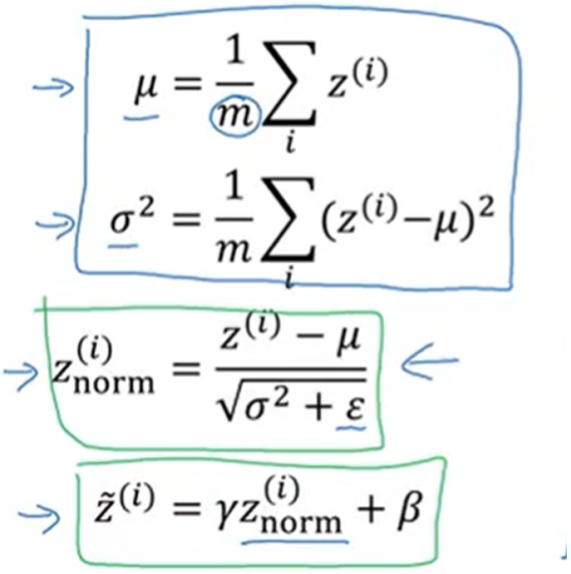
하지만 일반화 목적으로 쓰진 말고 학습 속도를 향상 시키는데 사용해라.

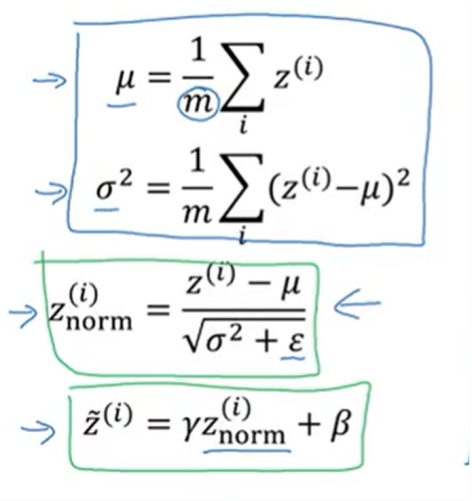
테스트 관점에서는 예측이 더 잘 맞도록 조금 다른 접근을 해야한다.

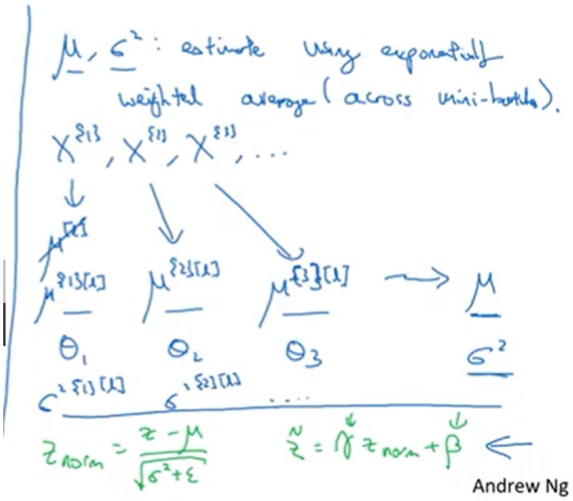
테스트시의 배치 정규화

테스트 과정에서는 64 개 등의 샘플을 포함하는 미니배치가 없으므로 동시에 처리할 수 없기 때문에 뮤와 시그마^2 를 처리할 다른 방법이 필요하다.

해결책: 여러 미니 배치에 걸쳐서 구한 뮤와 시그마^2 의 지수가중평균을 추정치로 사용한다.




$\mu = \frac{1}{m}\sum_i z^{(i)}$
$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)}-\mu)^2$
$z^{(i)}_{\text{norm}} = \frac{z^{(i)}-\mu}{\sqrt{\sigma^2+\varepsilon}}$
$\tilde{z}^{(i)} = \gamma z^{(i)}_{\text{norm}} + \beta$


μ, σ^2: estimate using exponentially weighted average (across mini-batches).
θ_1
θ_2
θ_3
$z_{\text{norm}} = \frac{z-\mu}{\sqrt{\sigma^2+\varepsilon}}$
$\tilde{z} = \gamma z_{\text{norm}} + \beta$
Andrew Ng