Training Neural Net

(How to set the Neural networks?)

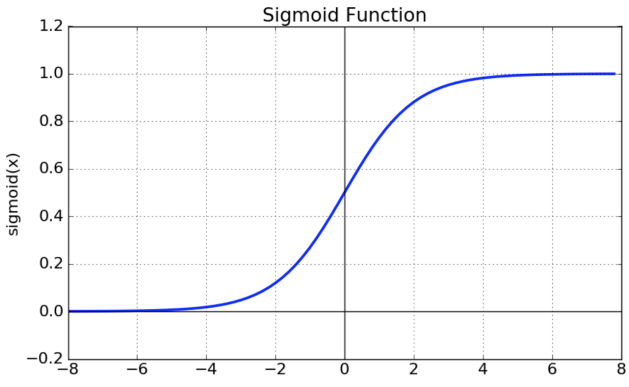
정재영

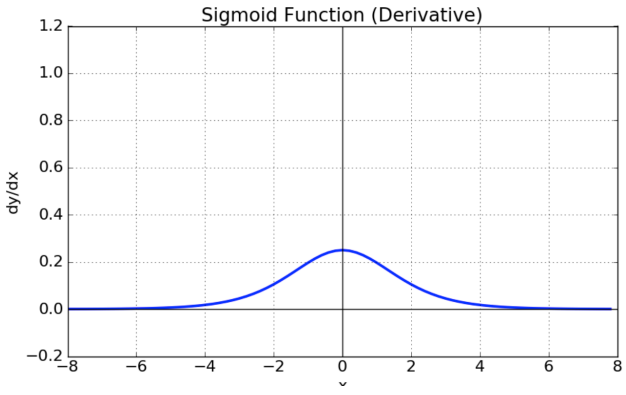
Content

1. Activations

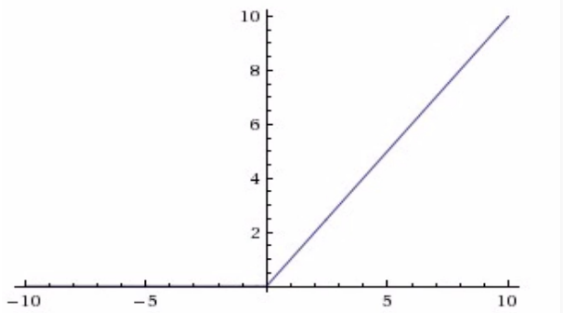
본격적인 논의에 앞서, 그 배경이 되는 활성화 함수 쟁점에 대해 간략히 언급하는 것이 적절할 것 같다. ReLU가 활성화 함수로서 인기를 끌기 이전에 주로 사용되었던 sigmoid 함수는 모델을 학습시킬 때 다음의 3 가지 주된 단점을 갖고 있다고 알려져 있다:

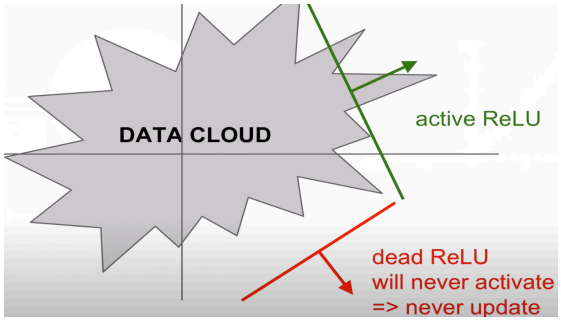
1) Not 0-centered, 2) Vanishing Gradient (stop to update grads), 3) Expensive cost in computation (by exp f)





활성화 함수를 사용하는 데 있어서 이 단점들을 해결하기 위해, 2012 년부터 ReLU 함수를 본격적으로 사용하기 시작하였다. ReLU 함수는 정의역이 0일 때 미분 불가능하다는 난점이 있지만, 다른 모든 영역에서는 위의 2) 같은 곤경이 나타나지 않았고, 1)에 대해서는 여전히 극복하지는 못하지만 연산적으로 매우 효율적이게 해주고, 더욱이 3)을 해결해주어 (exp을 사용하지 않기 때문에) 모델을 학습시킬 때 sigmoid 함수 보다 훨씬 더 좋은 성능을 얻게 해준다.



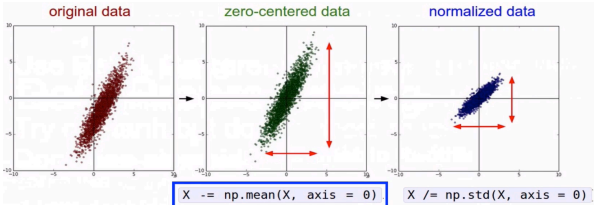


Q. ReLU 역시도 음수인 경우에 grads 가 모두 0 아닌가?

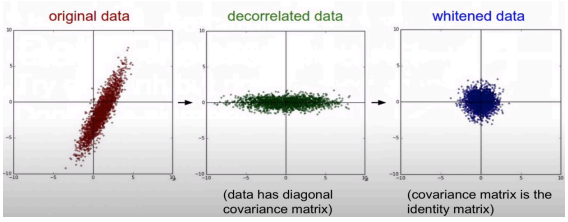
맞다. 그래서 음수인 경우에는 active 하지 않고, 그래서 grads를 업데이트 시키지 못한다. 하지만 양수인 경우에는 항상 grads를 업데이트 시킬 수 있다. 이처럼 ReLU가 deactive 한 경우를 소위 dead ReLU 라고 부르는데, 대체로 이러한 현상이 나타나는 원인은 잘못된 초기화, 너무 큰 학습률 lr 설정[1] 때문이다. 물론 이것은 분명 문제적이고, 따라서 ReLU 역시 아직 이상적인 함수가 아니다. 우리는 더 좋은 함수를 찾던가, 아니면 다른 방법으로 이 문제를 극복할 필욕가 있다. (이후에 이 학습률 문제를 극복하기 위해 Leaky-ReLU, PReLU 등이 등장했다.)

2. 전처리 과정 (참고)

- 일반적인 전처리 (1): 0 centered 해주기 위해, 각각의 입력값 X에서 평균을 뺀 뒤 정규화를 해준다. (img분석 시 자주 사용)



- 일반적인 전처리 (2): 위와 동일한 목표로 PCA(주성분분석;차원 줄이기) 수행 뒤에, 백색화whitening 해준다. (이미지 분석 시 일반적으로 사용 x)



[1] lr이 너무 큰 경우에는 뉴런의 값이 한번 dead ReLU zone에 빠지게 되면 다시 돌아오지 못하게 되어 vanishing grads 현상이 나타난다.

3. 가중치 초기화

우리는 방금 이미지 처리 시 전처리 과정이 0-centered를 수행한다는 것을 알아보았다. 이번에는 가중치 초기화에 대해 알아보자. 위에서 말했듯이 이것이 적절히 되지 않는다면, dead ReLU 현상이 일어날 수 있다. 따라서 초기화를 잘 하는 것은 매우 중요하다. cs231n을 참고할 때, 가중치 초기화의 종류는 4 가지가 있다.

3.1 여러가지 가중치 초기화 방법

1) ∀ Ws = 0,

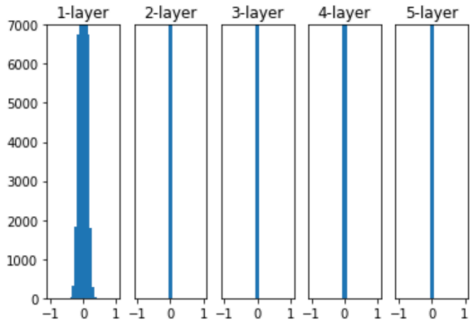
문제: 이 설정 때문에 모두 동일한 일만을 수행할 것. 즉, 이 경우에는 모두 동일한 출력값을 산출해낼 것이다. 그러면 역전파 시 모든 뉴런이 동일한 그래디언트 값을 갖게 된다. 학습이 잘 되려면, 각 뉴런의이 가중치에 따라 비대칭적이어야 (어떤 뉴런은 가중치가 크고, 다른 뉴런은 작은 경우) 한다. (=> 가중치를 부여할 의미가 없어진다)

2) Randomly setting ∀ Ws (better)

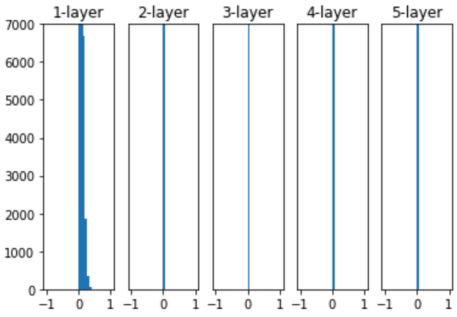
한계: 소규모small 연결망에서 이것은 좋은 선택일 수 있지만, 더 깊은 deeper 신경망에서는 모든 활성화가 0이 되기 때문에 학습이 문제가 나타날 수 있다:

① **작은 난수: 학습률이 0.01**일 때, 평균은 0으로 수렴하지만 std는 급속하게 0으로 수렴해서 모든 activation이 0이 된다.

그리고 이러한 설정을 고려할 때, 역전파 시의 그래디언트는 어떤 형태인가? 위 설정에 따를 때, X는 0에 가깝기 때문에, dW1 = X * dW2 이니까 dW1 또한 0에 가깝게 되고, 그러면 gradient는 0에 수렴하게 된다. 즉 Vanishing gradient 현상이 나타나게 된다.



[small random numbers — tanh][2]

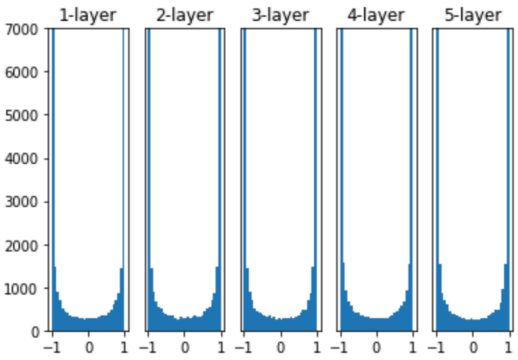


[small random numbers — ReLU]

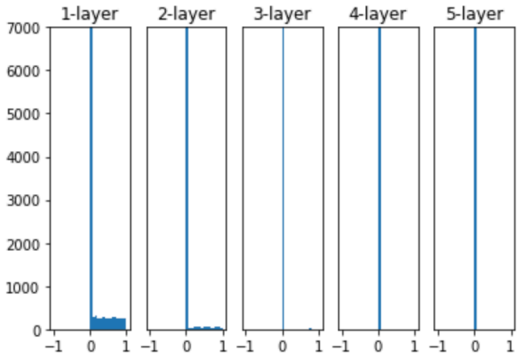
---

[2] https://github.com/musicjae/cs231n/blob/master/assignment2/weight_initializations.ipynb 에서, 초기화와 활성화 함수를 바꿔 가며 그래프를 비교할 수 있다.

② **큰 난수:** 마찬가지로, **학습률이 1**인 경우에도, 뉴런이 -1 또는 1 로 saturated 되어 gradients 는 0 으로 수렴하기 때문에, 손실loss이 업데이트 되지 않는다.



[large random numbers —tanh]

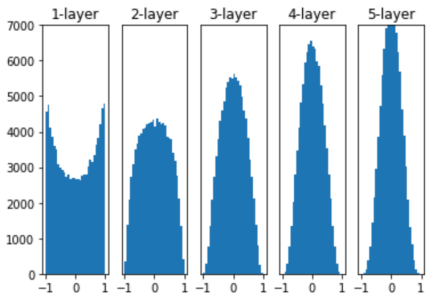


[large random numbers —ReLU]

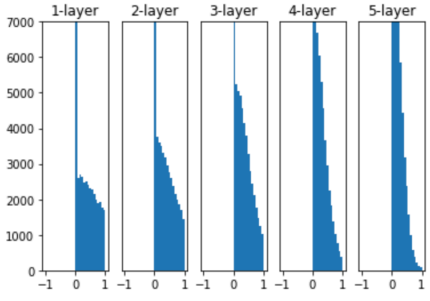
3) Xavier 초기화 (2010)

w = np.random.radn(fan_in, fan_out[3]) / sqrt(fan_in)

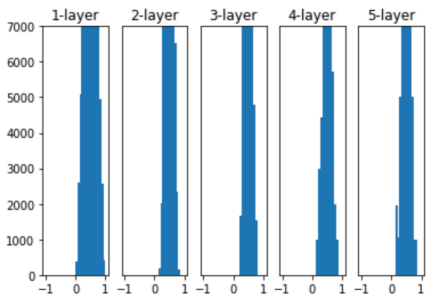
Tanh 함수에 잘 적용된다. 하지만 ReLU 함수에 사용 시 std가 0으로 수렴하는 곤경을 지닌다.



[Xavier—tanh]



[Xavier — relu]



[Xavier — sigmoid]

위 실습에서 나타나듯이, tanh 경우에는 학습이 잘 일어나는 데 반하여, relu, sigmoid 경우에는 점점 std가 0에 수렴하는 양상을 보인다.

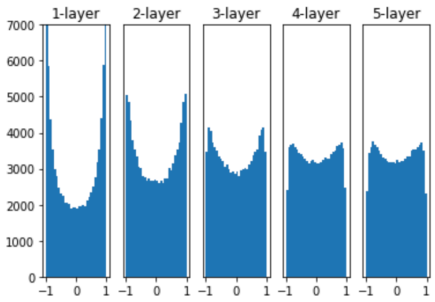
---

[3] fan_in: Input Layer의 뉴런 개수, fan_out: Output Layer의 뉴런 개수

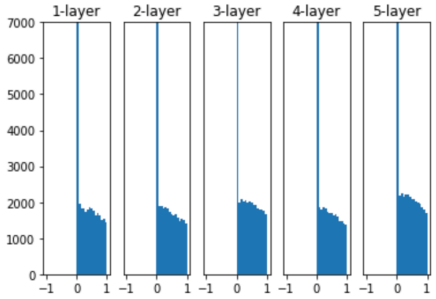
4) Kaming He 초기화 (2015)

W = np.random.radn(n_input, n_output) / sqrt(n_input **/ 2**)

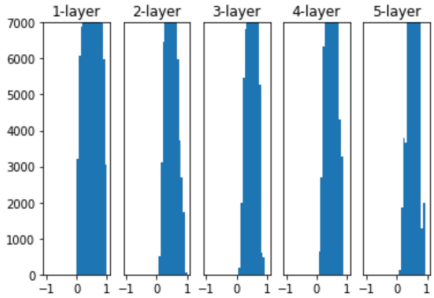
위에서 보았듯이, Xavier 초기화에서는 ReLU 활성화 적용 시 레이어가 깊어질수록 output이 0에 가까워지는 현상이 나타난다. He 초기화는 아래 그림에서 처럼 이런 문제를 개선해준다. 하지만 He 활성화 때 조차도 sigmoid 활성화 적용 시 나타나는 0으로 수렴하는 현상은 Xavier에 비해 유의미하게 개선되지 않는다.



[he—tanh]

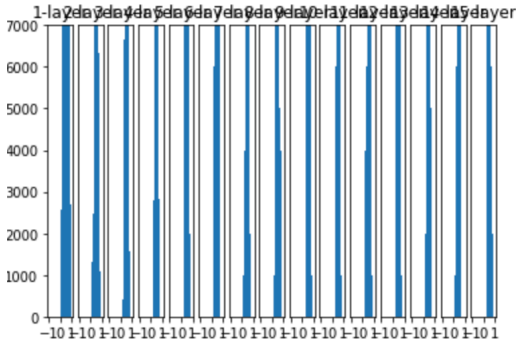


[he—relu]

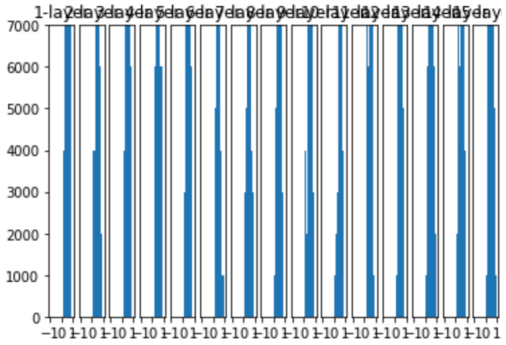


[he—sigmoid]

아래와 같이 은닉층이 15 개인 경우에도 he 초기화는 Xavier 초기화에 비해 개선되지 않는 것 같다. 따라서 이번 실습에 따르면 sigmoid 경우에 Xavier 초기화와 He 초기화 간의 뚜렷한 차이는 없어 보인다.



[Xavier—-sigmoid]



[he —- sigmoid]

3.2 이번 실습을 통해 도출된 잠정적 결론

어떤 초기화가 가장 좋은가? 이 단순한 질문에 대한 항상적인 답은 없다. 하지만 활성화 함수는 일반적으로 ReLU를 사용하고, 가중치 초기화는 매우 다양하게 쓰이지만, 가장 좋은 선택이 될 수 있는 것은 tanh 함수 사용 시에는 Xaiver 초기화, ReLU 함수 사용 시에는 he 초기화일 것이다.

4. Batch Normalization [4]

:Vanishing grad 현상이 나타나지 않게 해주면서 학습 속도를 가속화 시켜보자는 생각에서 고안된 방법.

"One way to make deep networks easier to train is to use more sophisticated optimization procedures such as SGD+momentum, RMSProp, or Adam. Another strategy is to change the architecture of the network to make it easier to train."[5]

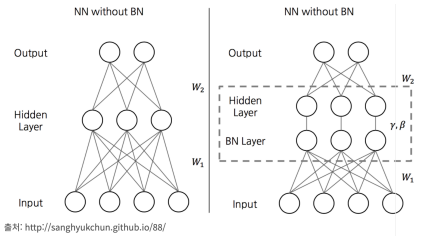
4.1 구체적인 고안 동기:

모델의 학습 과정에서 불안정성이 나타나는 원인은 내부에서 일어나는 covariate shift[6] 때문인 것 같다. 즉, 각 layer를 거치면서 input의 분포가 달라지는 현상이 나타나기 때문에 학습 과정에서 불안정성이 나타난다. 이러한 탓에 gradient가 0으로 수렴하는 현상, 즉 vanishing gradient가 나타난다.[7] 이것을 극복하기 위해, 각 layer를 거칠 때 마다 Normalization을 해주는 것이 바로 이 Batch Normalization이다.

"실제에서, *saturation* 문제와 *vainishing gradient* 문제는 *ReLU*와 주의깊은 초기화를 사용함으로써 처리된다. *(*우리는 이것을 앞에서 보았다*)* 하지만 만약 신경망이 학습할 때 *nonlinearity inputs*의 분포가 더 안정적인 채로 있게 해준다면, 최적화는 덜 *saturated* 될 것이고, 학습은 더 빨라질 것이다*"*[8]

4.2 Batch Normalization의 이점:

- 학습 효율 ↑
- 가중치 초기값에 의존 ↓
- 과적합 발생 가능성 ↓
- Vanishing Gradients 발생 가능성 ↓
- 높은 학습률을 허용해도 더 빠른 학습이 가능



출처: http://sanghyukchun.github.io/88/

---

[4] "Batch normalization is now considered a standard part of the neural network toolkit" from NIPS conference 2015

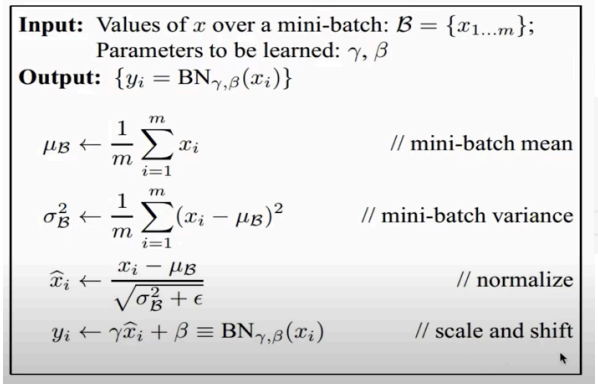
[5] cs231n assignment2 batch nomalizataion

[6] "Covariate Shift: 학습 기간 동안, <신경망 parameter>가 변화함에 따라 <신경망 활성화의 분포>가 변화하는 것", Ibid p.2

[7] " 심층 신경망 학습은 이전 layer의 모수가 변할 때, 학습 시간 동안 각 layer의 분포가 변한다는 사실에 의해 곤경과 마주하게 된다complicated. 이것은 lower 학습률, 신중한 W 최적화를 요구함으로써 학습을 느리게 하고, saturating nonlinearities로 모델 학습을 어렵게 한다. 우리는 이 현상을 internal covariate shift 라고 지칭한다." , Sergey Ioffe 외, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2015)

[8] ibid p.2

Batch Normalization




```
mu_b = np.mean(x, axis = 0) #평균
var = 1/float(N)*np.sum((x-mu_b)**2, axis = 0) # variance 구하기
x_hat = (x-mu_b)/np.sqrt(var+eps) # 정규화, eps: 수치적 안정성의 상수
y = gamma*x_hat + beta # scale & shift
out = y
```

4.3 실용적 접근

입력값 x에서 임의의 작은-batch를 골라 만들어진 N x D인 mini-batch 입력값 x'이 있다고 해보자. 이 batch에 대해 (1) mean, std 를 계산해주고, (2) 정규화해준다:

FC-Net ======> BN ========> Activation (보통 배치 정규화는 이와 같은 위치에 나열된다)

5. 학습 과정 돌보기 Babysitting the Learning Process (학습률 찾아가기)

전처리: zero centered

⇩

신경망의 아키텍처 결정: 은닉층은 몇 개인가? 몇 개의 node를 둘 것인가?

⇩

loss 체크: reg를 바꿔가면서 loss 변화 확인

⇩

Training data 일부분으로 과적합 여부 확인: (1) 일부의 data를 고른다.
(2) reg = 0.0에서 sgd를 사용
(3) 이 경우 반드시 과적합이 나와야 한다. (loss ≅ 0, accuracy ≅ 1) // meaning: 가중치 업데이트, 역전파가 제대로 작동하고 있고, 앞에서 설정된 학습률도 적절하다. 만약 과적합이 나오지 않으면, 문제가 있는 것.

⇩

학습률 learning rate 찾아가기: * lr too high → loss 값 = Nan
* 정확한 lr 결정은 교차 검증으로 해야 해

6. 모수parameter 최적화 (교차검증을 통해 hyperparameter 결정하기)

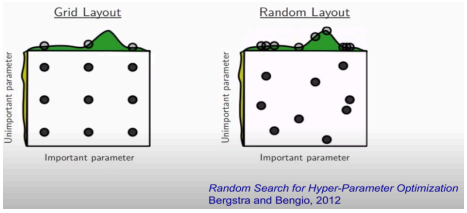
6.1 교차 검증의 전략:

Coarse (A few epoch) ====> Fine (Longer running time)

6.2 Hyperparameter에 대한 두 가지 탐색 과정

(1) 랜덤 탐색: logspace에 랜덤하게 값을 골라오기 ==> val_acc ↑

(2) 그리드 탐색: 등간격으로 모든 범위를 커버할 수 있도록 exhaustive 탐색하기 // worse.



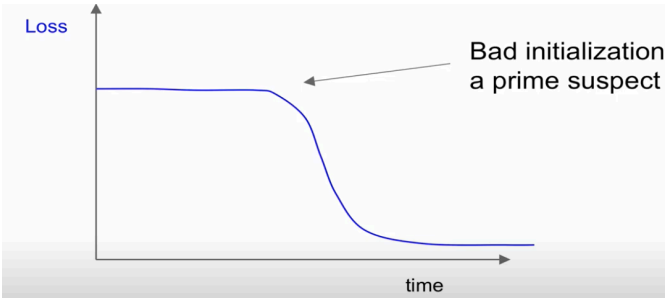
(1) 위 그림을 참고해보면, Grid 탐색은 중요한 모수의 3 지점만을 탐색. 반면에, 랜덤 탐색은 9 개의 지점을 탐색. 더 많은 중요한 모수의 지점을 탐색하는 랜덤 탐색이 better.

(2) 랜덤 탐색에서 중요치 않은 모수는 하나의 중요한 모수에 대해 한번만 탐색하는 데 반하여, 그리드 탐색에서 중요치 않은 모수는 하나의 중요한 모수에 대해 3 번을 탐색한다 => 비효율적.

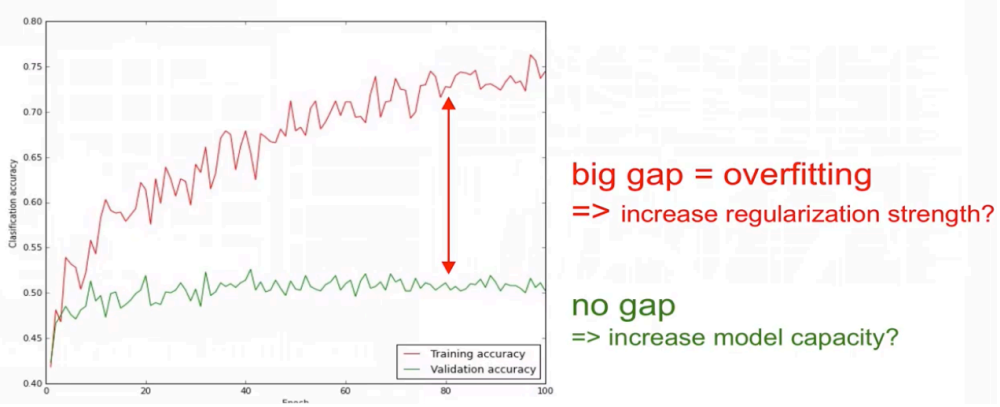
6.3 모니터링 대상

(1) loss

- 손실 함수가 bump 하는 경우: 초기화를 잘못했음을 의심



(2) 정확도 (학습 정확도, 검증 정확도)



(3) 가중치 업데이트 / 가중치 크기 (1/1000이 ideal)

```
# assume parameter vector W and its gradient vector dW
param_scale = np.linalg.norm(W.ravel())
update = -learning_rate*dW # simple SGD update
update_scale = np.linalg.norm(update.ravel())
W += update # the actual update
print update_scale / param_scale # want ~1e-3
```

ratio between the values and updates: ~ 0.0002 / 0.02 = 0.01 (about okay)
want this to be somewhere around 0.001 or so

(추가 예정)

Reference

[1] cs231n lectures
[2] Sergey Ioffe, Christian Szegedy, *'Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift'*, 2015