2021 Spring

Artificial Intelligence & Deep Learning

Prof. Minsuk Koo

Department of Computer Science & Engineering
Incheon National University

5.2 성능 향상을 위한 요령

- 5.2.1 데이터 전처리
- 5.2.2 가중치 초기화
- 5.2.3 모멘텀
- 5.2.4 적응적 학습률
- 5.2.5 활성함수
- 5.2.6 배치 정규화

5.2 성능 향상을 위한 요령

- 여러 기관이 연구결과를 공유
  - 『Neural Networks: Tricks of the Trade』는 연구결과를 한데 묶은 대표적인 책(1998년에 1판, 2012년에 2판)
- 5.2절의 내용은 경험규칙
  - 주어진 데이터에 잘 들어맞을지는 실험을 통해 신중히 확인해야 함
  - Bengio의 권고 [Bengio2012]

"… the wisdom distilled here should be taken as a guideline, to be tried and challenged, not as a practice set in stone. … 이 논문이 제시한 정제된 기법들은 자신의 문제에 적용한 다음 변형하여 새로운 기법을 만드는 길잡이 역할 정도로 받아들여야지 만고불변의 법칙으로 여겨서는 안 된다."

5.2.1 데이터 전처리

규모 문제

- 예) 건강에 관련된 데이터 $(\text{키}(\text{m}), \text{몸무게}(\text{kg}), \text{혈압})^{\text{T}}$
  - 1.885m와 1.525m는 33cm나 차이가 나지만 특징값 차이는 불과 0.33
  - 65.5kg과 45.0kg은 20.5라는 차이
  - 첫 번째와 두 번째 특징은 대략 100배의 규모 차이
- $-\delta_j z_i$가 그레이디언트이기 때문에 첫 번째 특징에 연결된 가중치는 두 번째 특징에 연결된 가중치에 비해 100여 배 느리게 학습됨 → 느린 학습의 요인

5.2.1 데이터 전처리

- 모든 특징이 양수인 경우의 문제
  - [그림 5-6]의 경우 ↑ 표시된 가중치는 모두 증가, ↓ 표시된 가중치는 모두 감소
  - 이처럼 뭉치로 증가 또는 감소하면 최저점을 찾아가는 경로가 갈팡질팡하여 느린 수렴

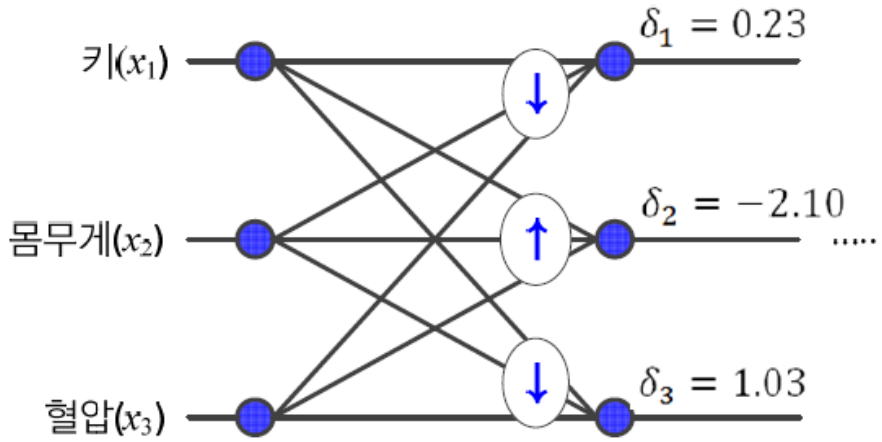


그림 5-6 특징이 모두 양수일 때 가중치가 뭉치로 갱신되는 효과

5.2.1 데이터 전처리

- 식 (5.9)의 정규화는 규모 문제와 양수 문제를 해결해줌
  - 특징별로 독립적으로 적용

$$x_i^{new} = \frac{x_i^{old} - \mu_i}{\sigma_i} \qquad (5.9)$$

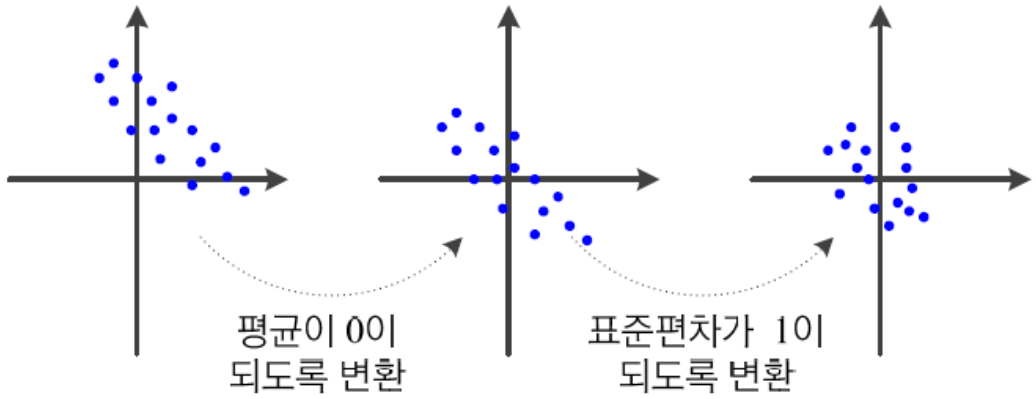


그림 5-7 표준점수로 변환

5.2.1 데이터 전처리

- 명칭값을norminal value 원핫one-hot 코드로 변환
  - 예) 성별의 남(1)과 여(2), 체질의 태양인(1), 태음인(2), 소양인(3), 소음인(4)
    - 명칭값은 거리 개념이 없음
  - 원핫 코드는 값의 개수만큼 비트를 부여
    - 성별은 2비트, 체질은 4비트 부여
    - 예) 키 1.755m, 몸무게 65.5kg, 혈압 122, 남자, 소양인 샘플

$$(1.755, 65.5, 122, 1, 3) \rightarrow (1.755, 65.5, 122, \underbrace{1, 0}_{\text{성별}}, \underbrace{0, 0, 1, 0}_{\text{체질}})$$

e.g. consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet)
- (mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet)
- (mean along each channel = 3 numbers)

5.2.2 가중치 초기화

대칭적 가중치 문제

- [그림 5-8]의 대칭적 가중치에서는 z_1^{l-1}과 z_2^{l-1}가 같은 값이 됨. $-\delta_j z_i$가 그레이디언트기 때문에 u_{11}^l과 u_{12}^l이 같은 값으로 갱신됨 → 두 노드가 같은 일을 하는 중복성 발생
- 난수로 초기화함으로써 대칭 파괴

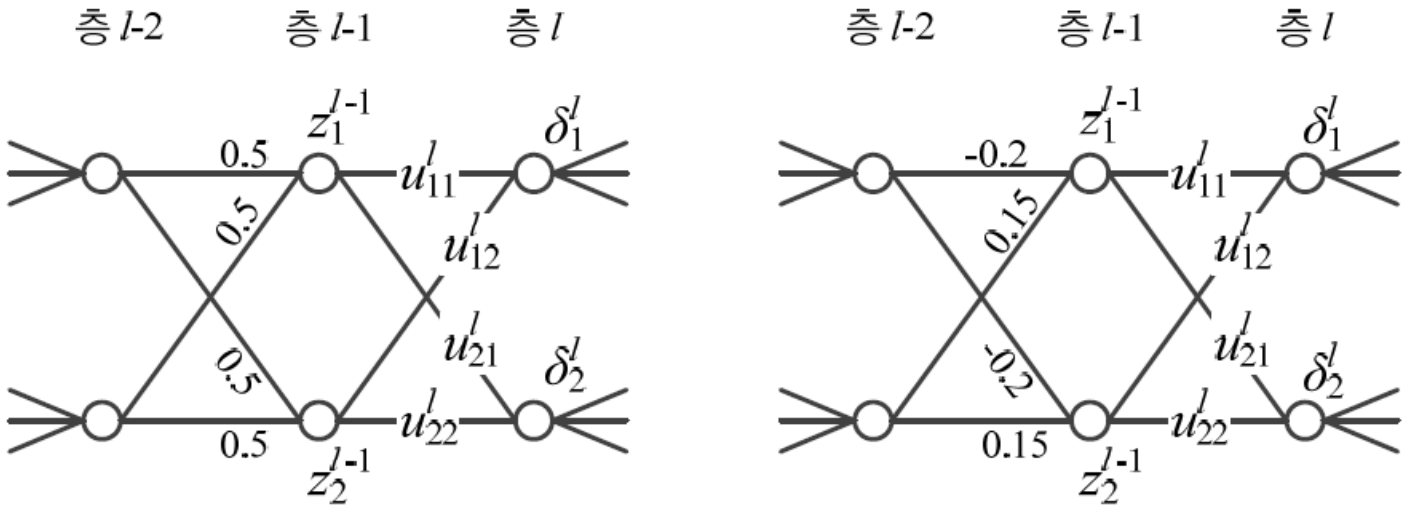


그림 5-8 대칭적 가중치로 초기화된 경우의 중복성 문제

5.2.2 가중치 초기화

- 난수로 가중치 초기화
  - 가우시언 분포 또는 균일 분포에서 난수 추출. 두 분포는 성능 차이 거의 없음
  - 난수 범위는 무척 중요함 → 식 (5.10) 또는 식 (5.11)로 r을 결정한 후 $[-r,r]$ 사이에서 난수 발생

$$r = \frac{1}{\sqrt{n_{in}}} \qquad (5.10)$$

$$r = \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}} \qquad (5.11)$$

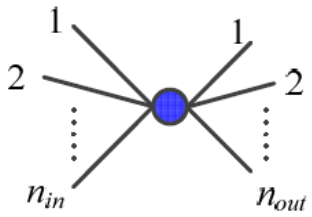


그림 5-9 노드로 들어 오는 에지 개수 n_{in}과 노드에서 나가는 에지 개수 n_{out}

  - 바이어스는 보통 0으로 초기화
- 사례
  - AlexNet [Krizhevsky2012]: 평균 0, 표준편차 0.001인 가우시언에서 난수 생성
  - ResNet [He2016a]: 평균 0, 표준편차 $\sqrt{\frac{2}{n_{in}}}$인 가우시언에서 난수 생성

5.2.2 가중치 초기화의 중요성

- First idea: **Small random numbers**
(gaussian with zero mean and 1e-2 standard deviation)

```
W = 0.01* np.random.randn(D,H)
```

Works ~okay for small networks, but problems with deeper networks.

From Stanford CS231n

5.2.2 가중치 초기화의 중요성

Lets look at some activation statistics

E.g. 10-layer net with 500 neurons on each layer, using tanh non-linearities, and in itializing as described in last slide.

```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)
```

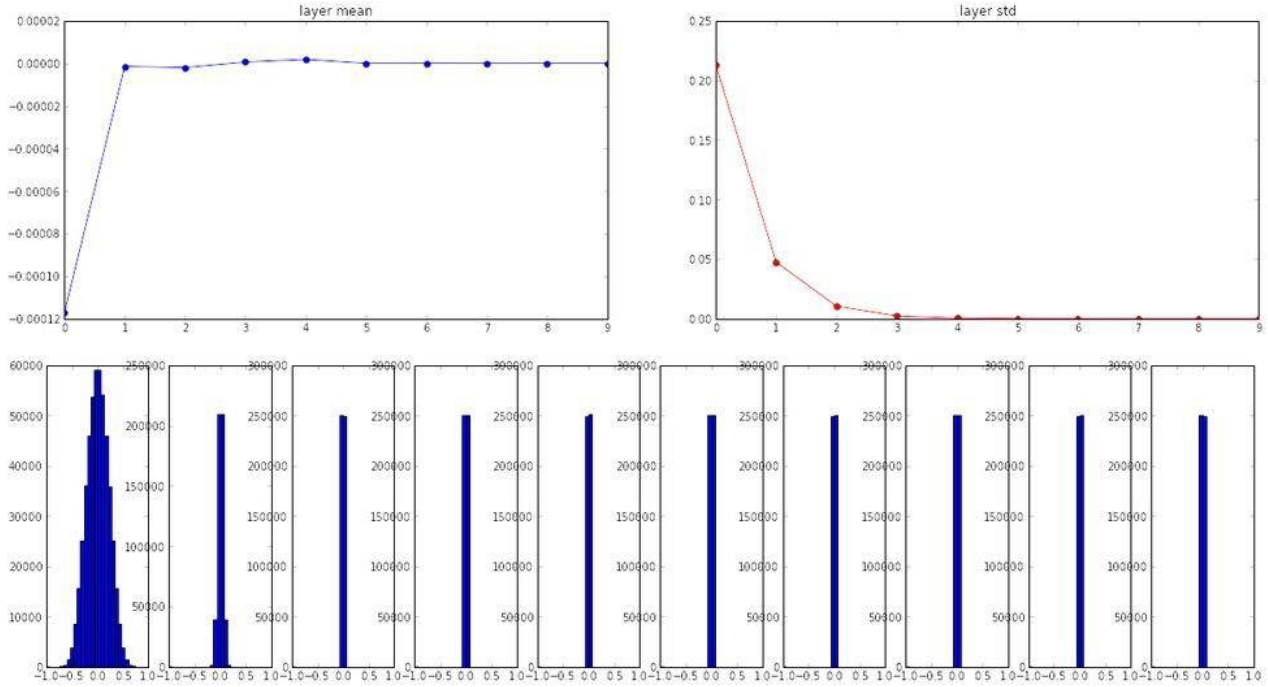
```
act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer
```

```
# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

# plot the raw distributions
plt.figure()
for i,H in Hs.iteritems():
    plt.subplot(1,len(Hs),i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

From Stanford CS231n

5.2.2 가중치 초기화의 중요성

```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```



All activations become zero!

Q: think about the backward pass. What do the gradients look like?

Hint: think about backward pass for a W*X gate.

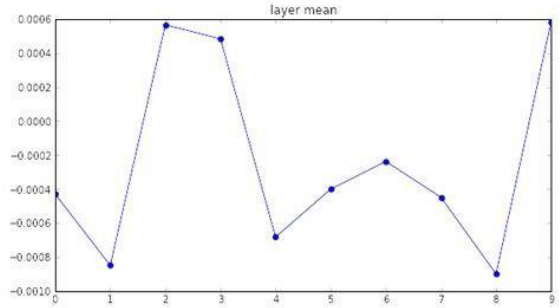
From Stanford CS231n

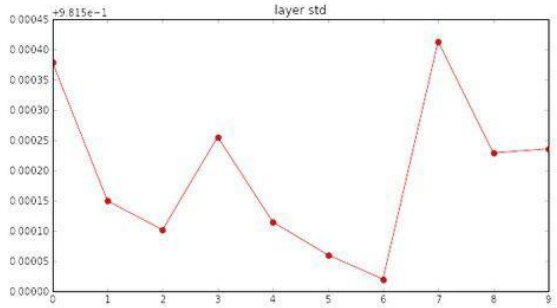
5.2.2 가중치 초기화의 중요성

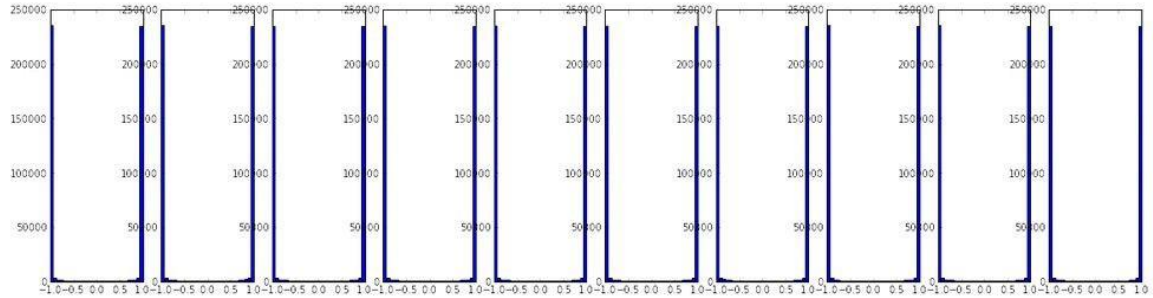
```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```

*1.0 instead of *0.01







Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

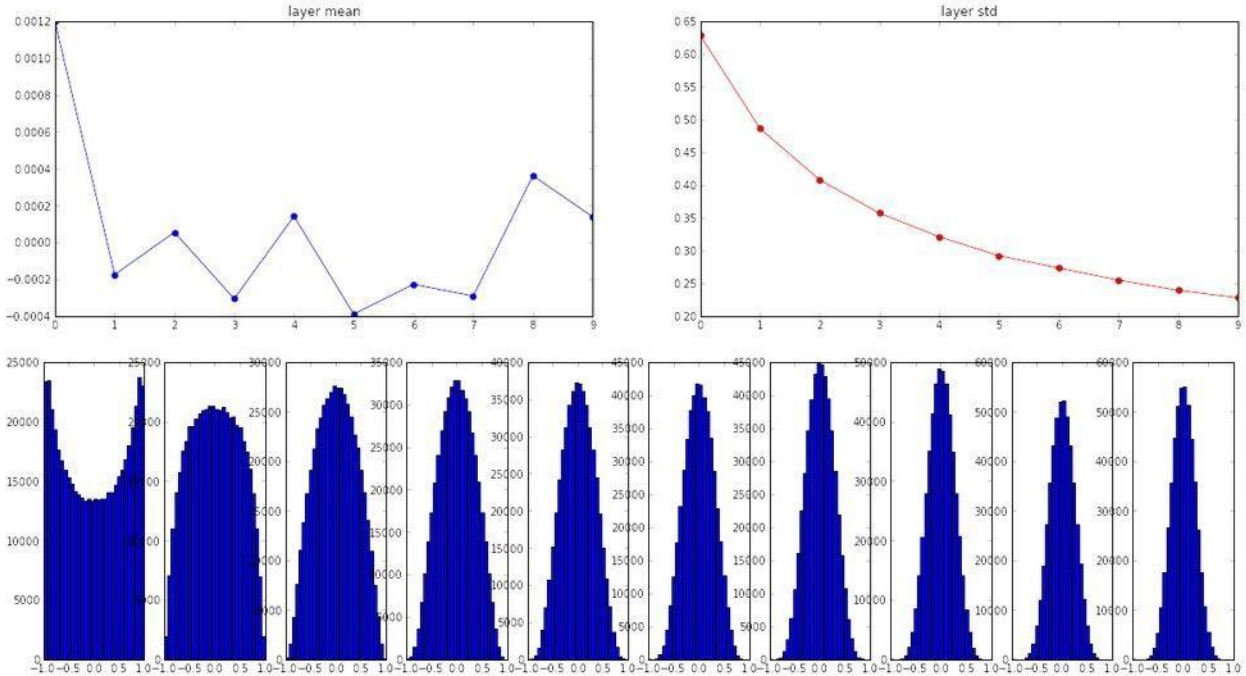
From Stanford CS231n

5.2.2 가중치 초기화의 중요성

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

"Xavier initialization" [Glorot et al., 2010]



Reasonable initialization.
(Mathematical derivation assumes linear activations)

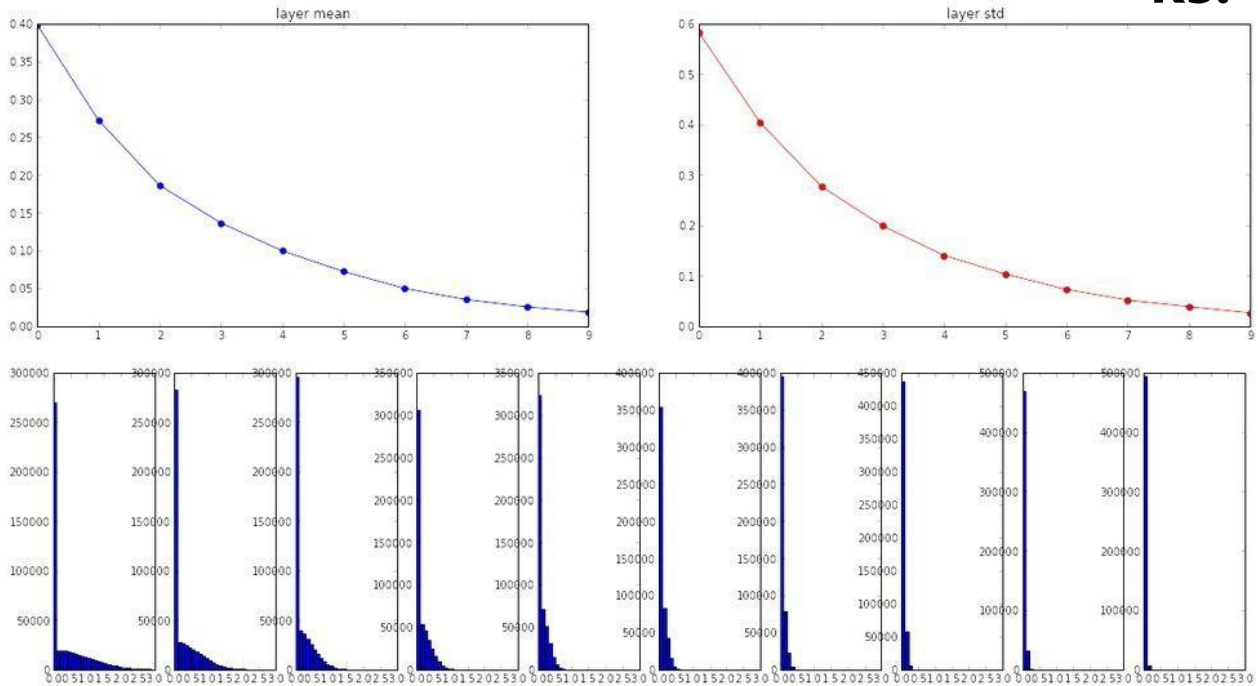
From Stanford CS231n

5.2.2 가중치 초기화의 중요성

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.140299
hidden layer 6 had mean 0.072234 and std 0.103280
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

but when using the ReLU nonlinearity it breaks.



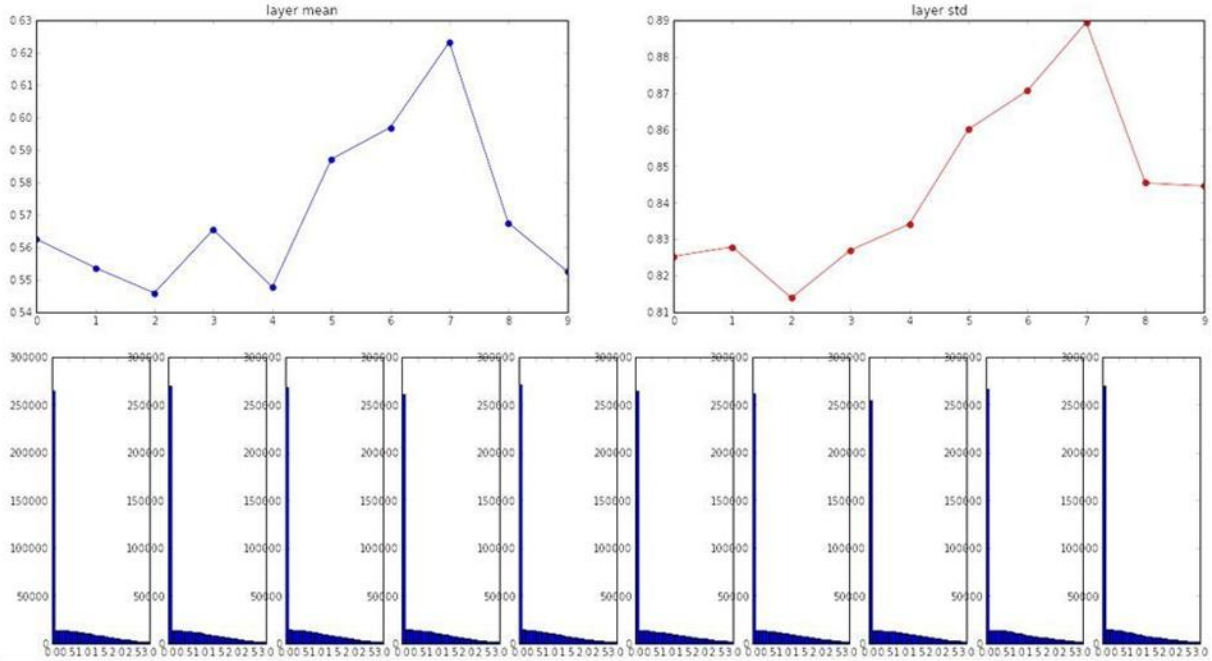
From Stanford CS231n

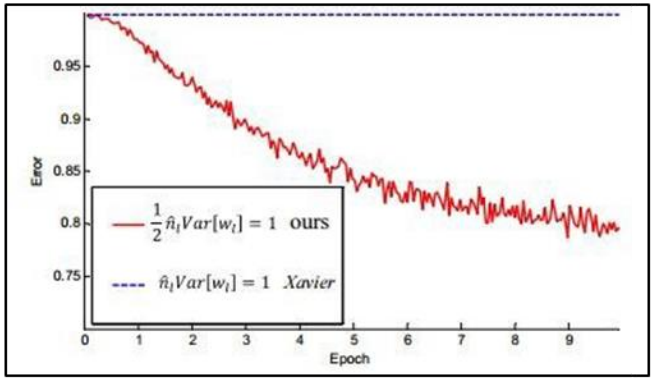
5.2.2 가중치 초기화의 중요성

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

He et al., 2015 (note additional /2)





From Stanford CS231n

5.2.2 가중치 초기화

- 또 다른 방법들
  - [Saxe2014]: 가중치 벡터가 수직이 되도록 설정
  - [Sussillo2014]: 임의 행로random walk 활용하여 설정
  - [Sutskever2013]: 가중치 초기화와 모멘텀을 동시에 최적화
  - [Mishkin2016]: 가중치 분포가 아니라 노드의 출력값 분포가 일정하도록 강제화