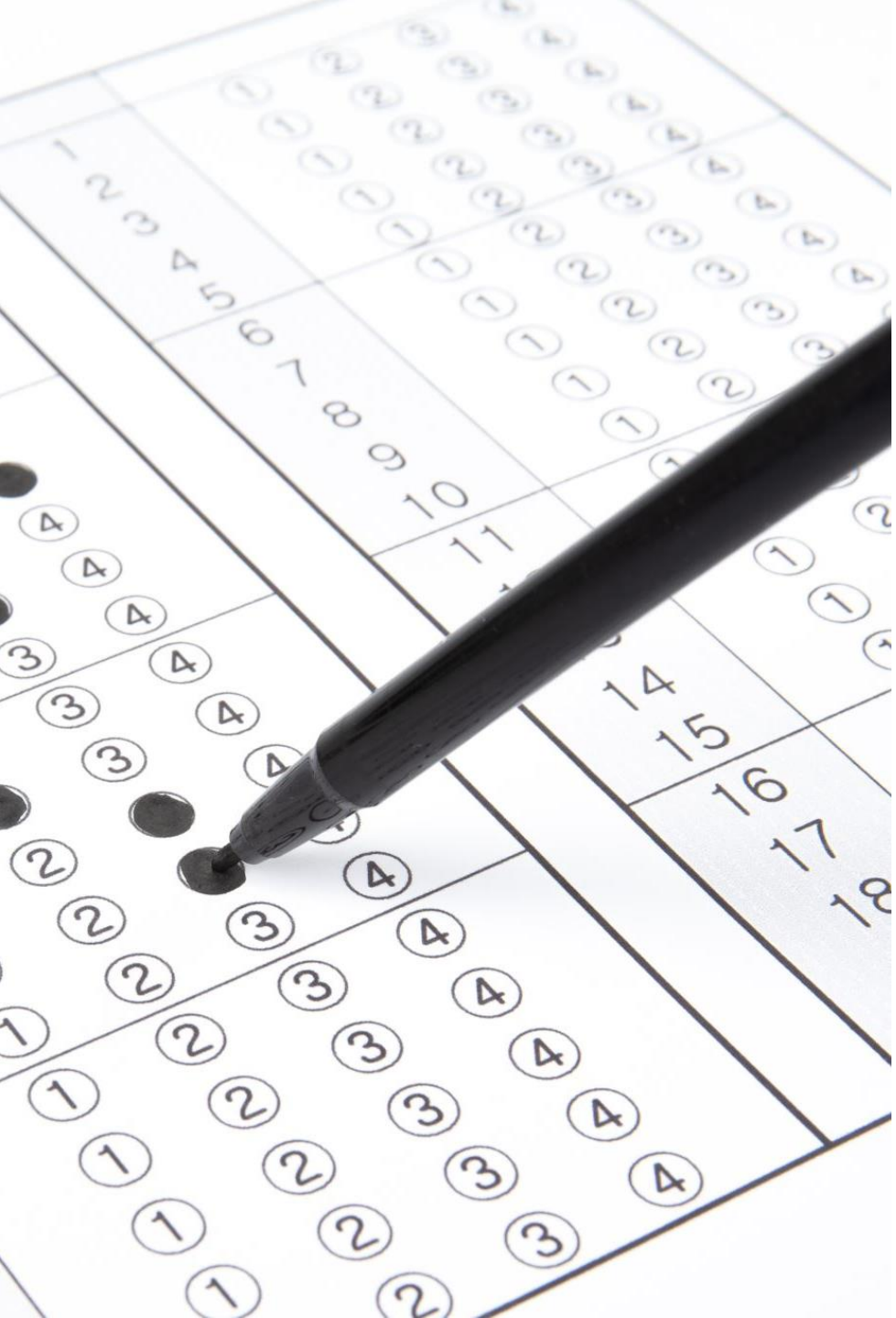
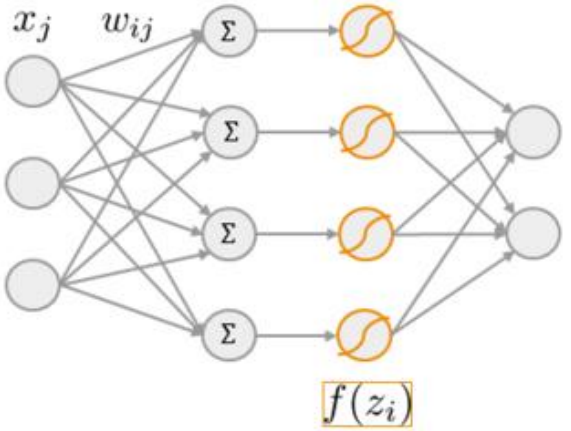


cs231n summary

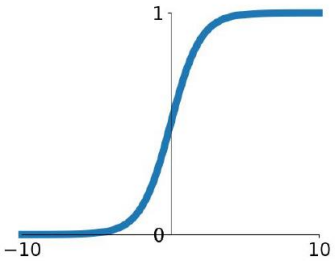
LECTURE 6

Activation Function

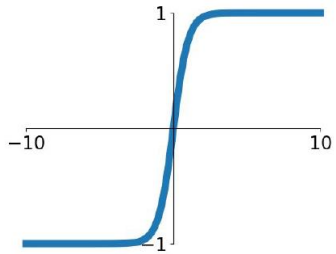
- Linear → non-linear why? Linear-multi-layer = single layer
- E.g $w_1(w_2 \dots (w_n x + b_n) \dots .) = w_{n+1}x + b_{n+1}$



Sigmoid and tanh



Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$



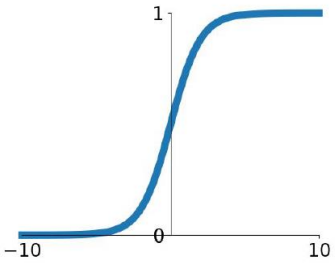
tanh(x) $= 2\sigma(2x) - 1$

	Sigmoid	Tanh(scaled of sig)
장점	[0,1]로 squash(뭉갬)	[-1,1]로 squash
	Nice interpretation as a saturating firing rate of a neuron.	Zig-zag path가 적다
단점	• Exp() 비용이 비쌈 • Saturation 이 weight 학습을 죽임(기울기 0인 지점)	
	0-cente가 아님 → 비효율적인 gradient 탐색(zig-zag path)	

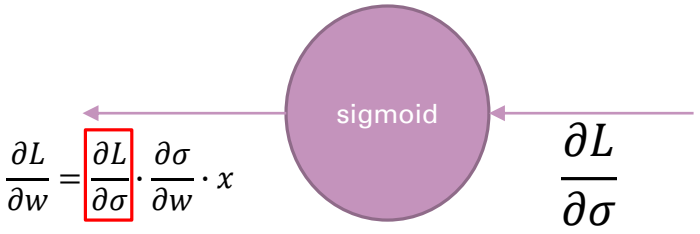
Sigmoid and tanh

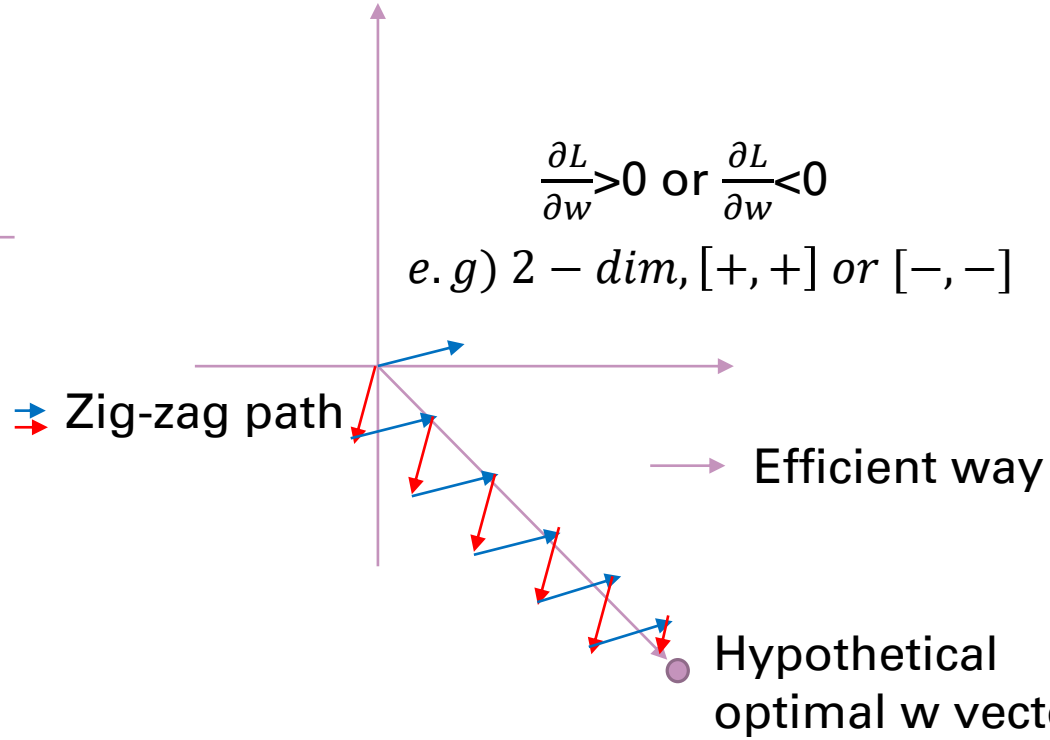
- Output is not zero-centered

If all x>0

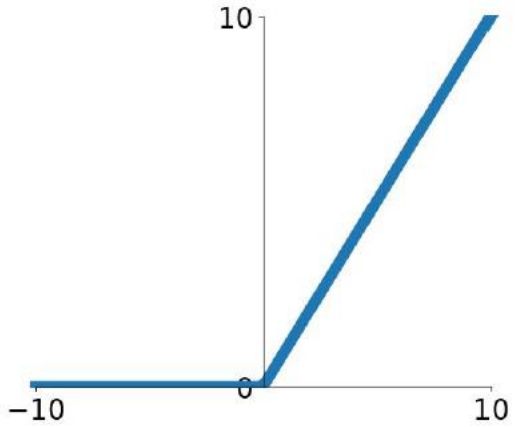


Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$





ReLu



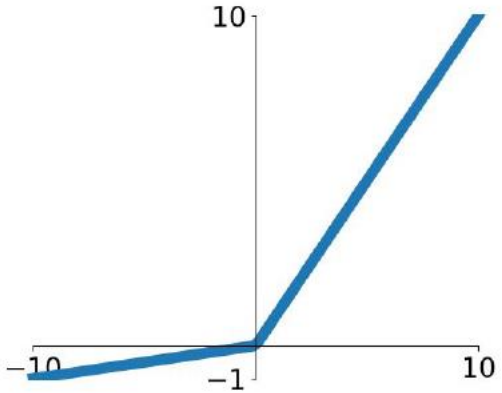
ReLU $f(x) = \max(0, x)$

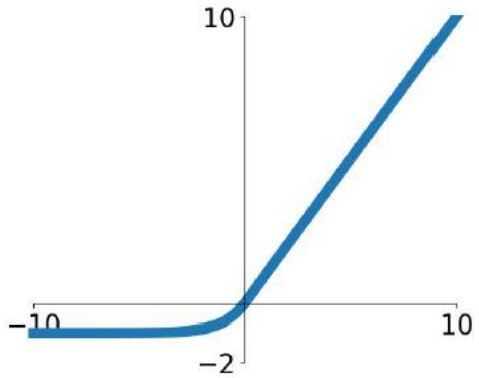
- \+ region, no saturation(포화)
- 계산적 이점
- Sigmoid/tanh보다 빠른 수렴속도(6배)
- Sigmoid보다 생물학적으로 유사함

단점

- 0-중심이 아님 : 비효율적인 decent탐색
- \- region , gradient died

Other ReLu





Leaky ReLU

$$f(x) = \max(0.01x, x)$$

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

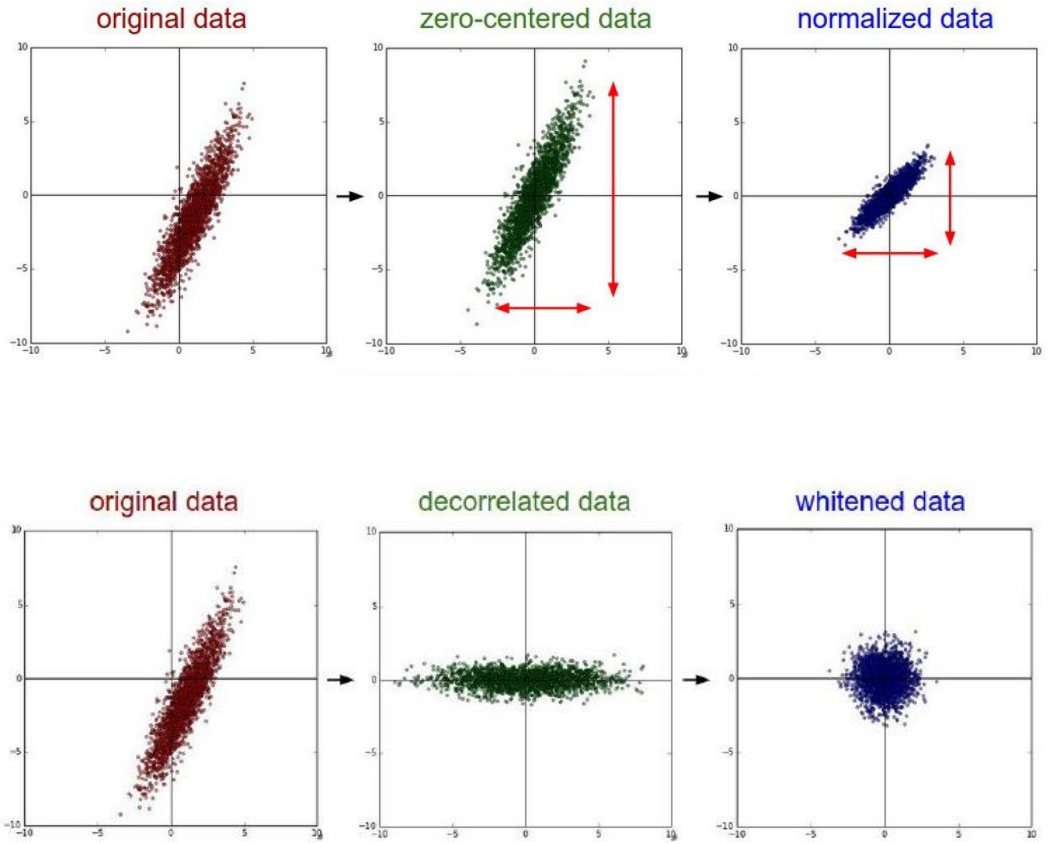
Exp ReLu

Max out Neuron

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

- ReLu, Leaky ReLU 의 일반화
- NO saturate, NO die
- 문제는 하나의 activation function에 두개의 변수(뉴런)

Data Processing



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)

Weight Initialization

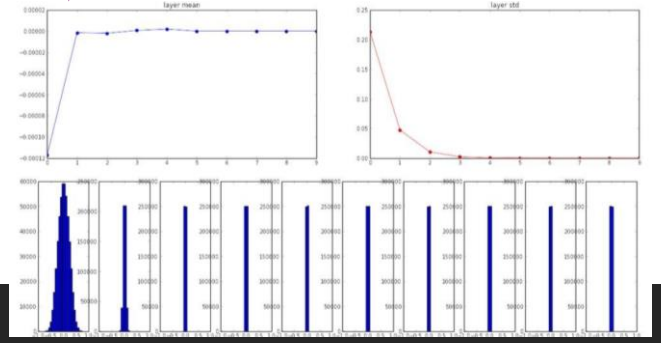
- W값을 초기화하는 것에 관심이 없었으나 performance에 꽤 영향을 주기 때문에 관심이 생김

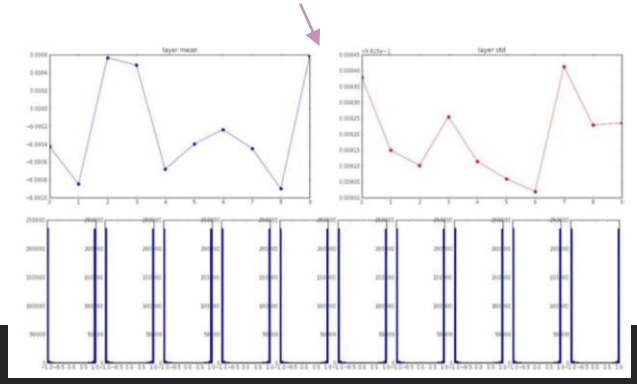
10layer – 500 neuron tanh activation function

- W =0 으로 초기화하면 모든 w가 같은 값으로 updating → 적합하지않음

- W = 0.01*np.random.randn(D,H) 충분히 작은 랜덤 숫자 → 단일 층에는 잘작동하나 층이 깊어지면 activation 을 0으로 만들어버림 → 업데이팅이 안됨

- W = 1*np.random.randn(D,H) 큰 랜덤 숫자 → -1,1로 포화되서 gradien가 0이 되서 업데이팅이 안됨.

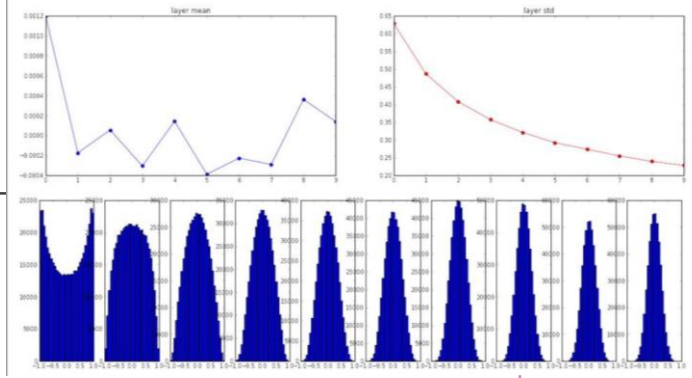




```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
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]



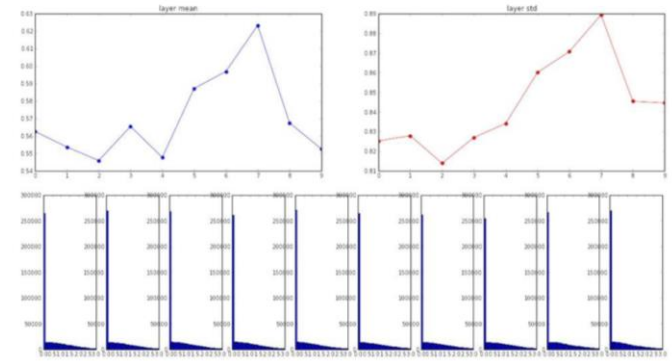
Reasonable initialization.
(Mathematical derivation assumes linear activations)

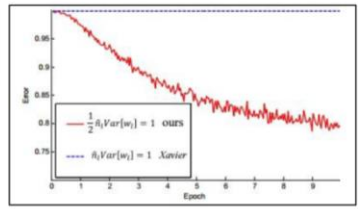
But not well in ReLU

```
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)



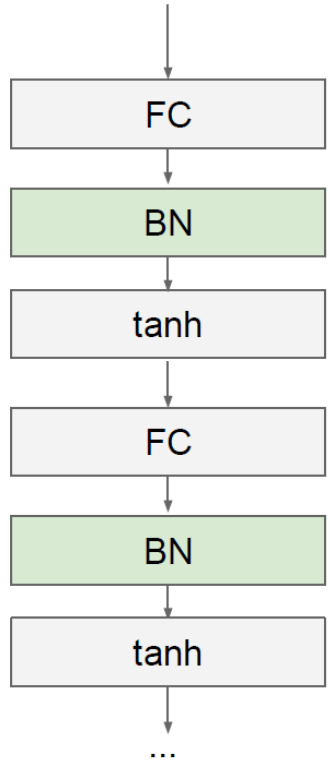


Batch Normalization [Ioffe and Szegedy , 2015]

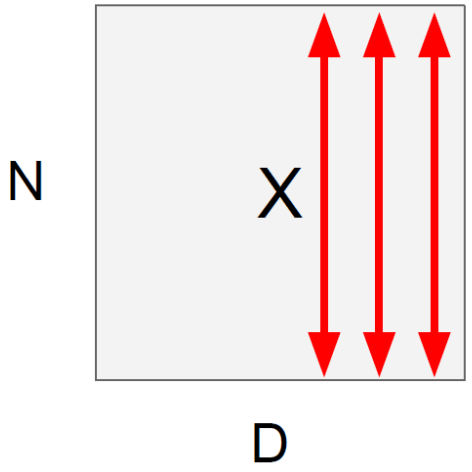
Input 의 distribution이 normalization되면 좋을텐데..

Input whitening? (mean = 0 , dev = 1) → 단점이 많다

공분산 계산,inverse계산, 가장 큰 문제 : 일부 parameter등의 희석(e.g bias)



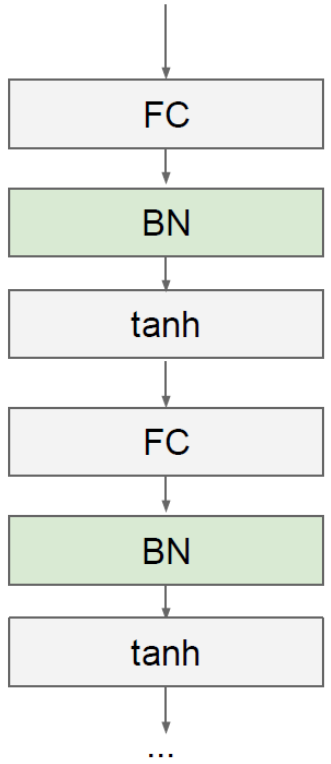
Batch Normalization [Ioffe and Szegedy , 2015]



1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$



Batch Normalization [Ioffe and Szegedy , 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

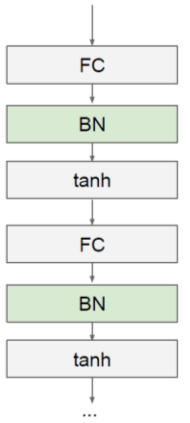
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Gradient가 매우 잘 전달됨
- 학습 진도율이 높아짐
- W초기화에 의존하지 않아도됨
- Regularization 의 효과가 있어 drop out 을 쓸필요가 적어진다.

Batch Normalization [Ioffe and Szegedy , 2015]

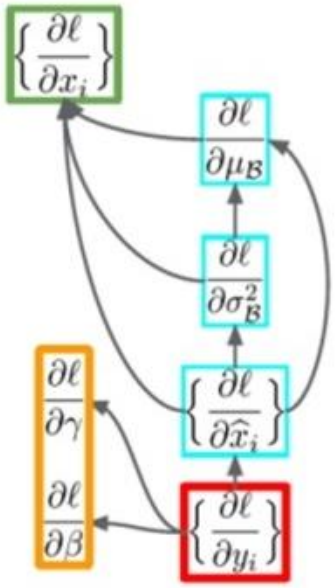


Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

Problem: do we necessarily want a unit gaussian input to a tanh layer?

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

γ, β 는 학습가능



$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma^2_{\mathcal{B}} + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} + \frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

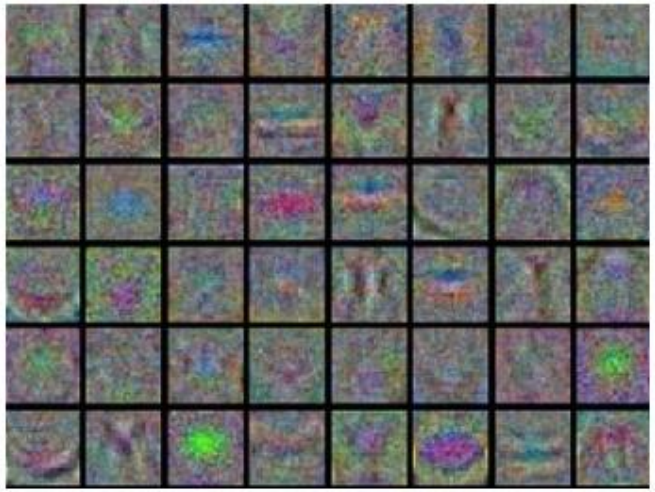
$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

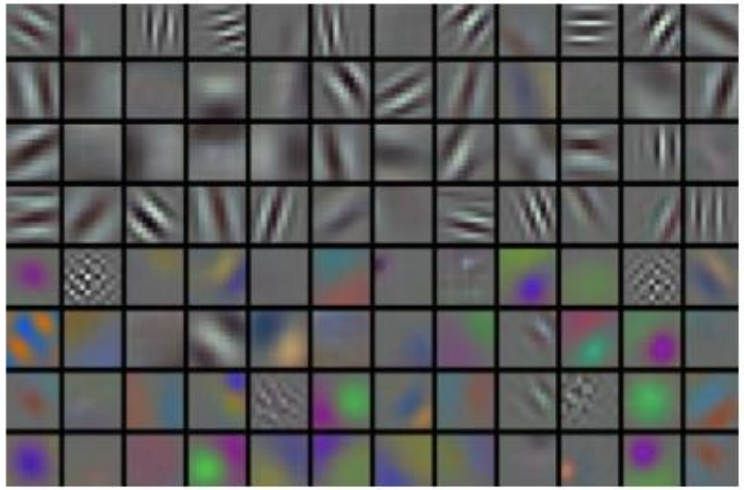
$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Baby sitting Learning Process

- Training 모니터를 어떻게 할까
- Hyperparameter를 원하는 방향으로 조정
- 1) 데이터 전처리
- 2) 아키텍쳐 정하기
- Loss 값을 확인 – regularization이 있을때 없을때를 비교.
  - learning rate를 조정
- 기타 파라미터값들..
- Hyperparameter 설정에 힌트가 될수도있음

Baby sitting Learning Process





신경망 첫 층의 웨이트값(weight)를 시각화한 에. **좌측**: 특징값(feature)에 잡음(noise)이 많을 때 나타날 수 있는 증상: 수렴하지 않은 망(network), 적절하지 않은 학습 속도(learning rate), 매우 낮은 정규화 페널티(regularization penalty). **우측**: 부드럽고 깨끗하며 다양한 피쳐값들이 보이는 경우 훈련이 잘 진행되고 있다는 지표일 수 있다.

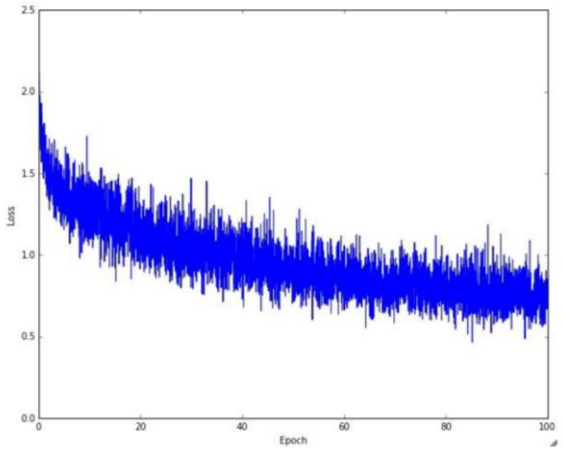
Hyperparameter Optimization

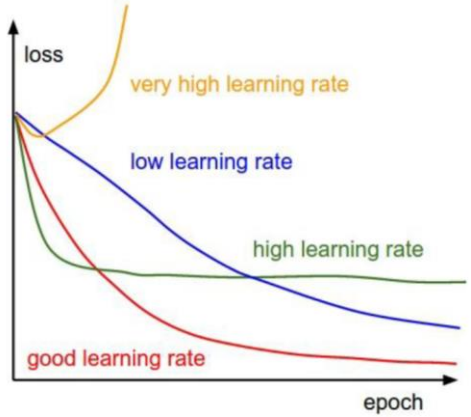
- 학습속도의 초기값(the initial learning rate)
- 학습속도 경감 계획, 이를테면 경감 상수 (learning rate decay schedule (such as the decay constant))
- L2나 드랍아웃 페널티의 정규화 강도 (regularization strength (L2 penalty, dropout strength))

몇가지 TIP

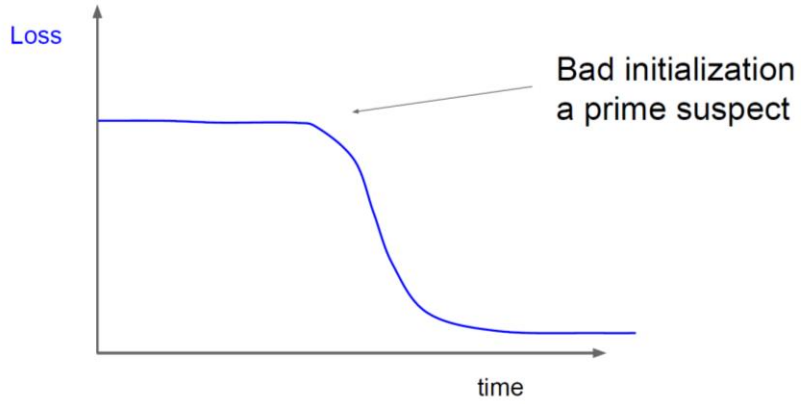
- **초모수의 범위 (Hyperparameter ranges)**. 로그 스케일로 초모수를 찾아라
- **그리드 검색보다는 임의 검색 (Prefer random search to grid search)**
- **가장 좋은 값이 경계에 있으면 조심하라 (Careful with best values on border) (범위오류일지도)**
- **성긴 검색에서 촘촘한 검색으로 (Stage your search from coarse to fine)**.
- **베이지안 초모수 최적화 (Bayesian Hyperparameter Optimization)**

Hyperparameter Optimization

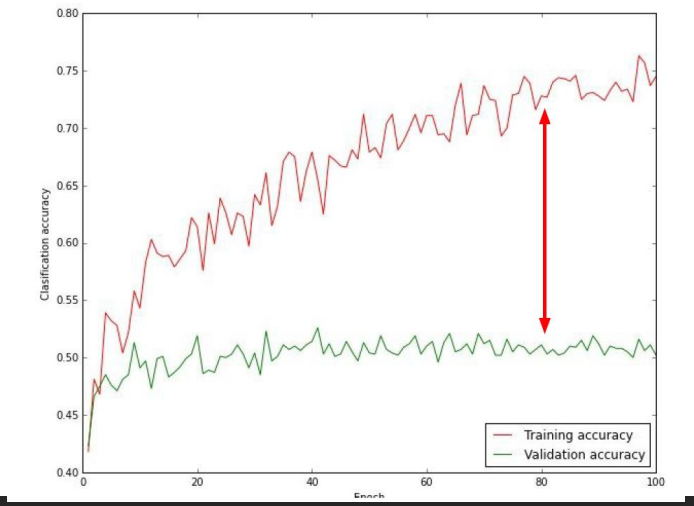




빠른 수렴속도를 갖는게 좋다.
Epoch(자료가 학습(SGD)에 사용된 수)



초기설정이 별로였음



Gap 이 크면 overfitting 줄이는 게 좋다.

Thanks