

Training Neural Networks I

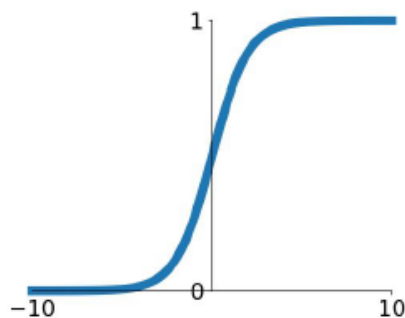


Fast Campus
Start Deep Learning with TensorFlow

Activation Functions

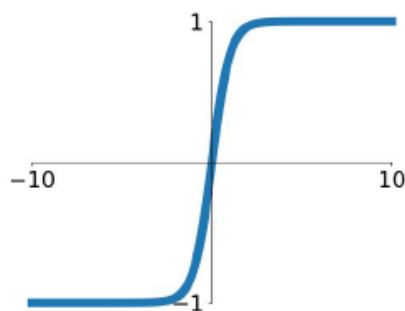
Sigmoid

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



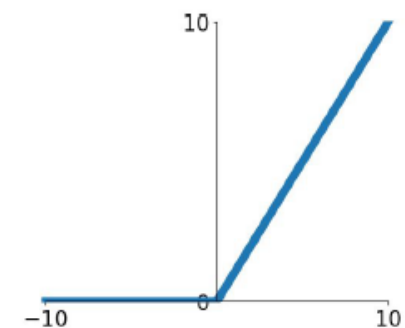
tanh

$$\tanh(x)$$



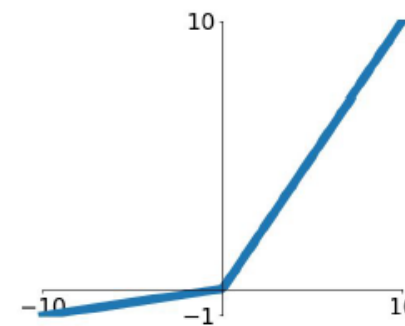
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

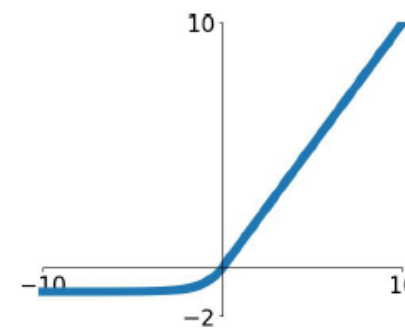


Maxout

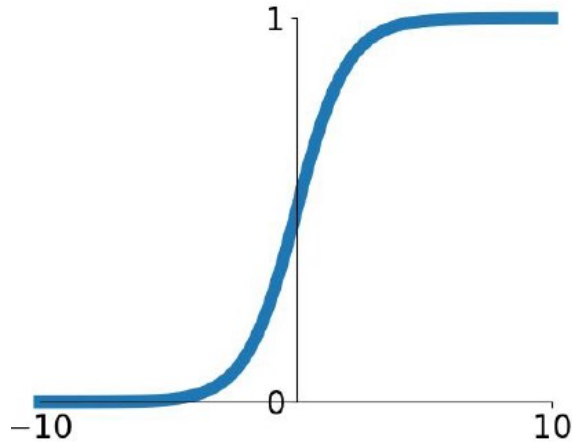
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Sigmoid



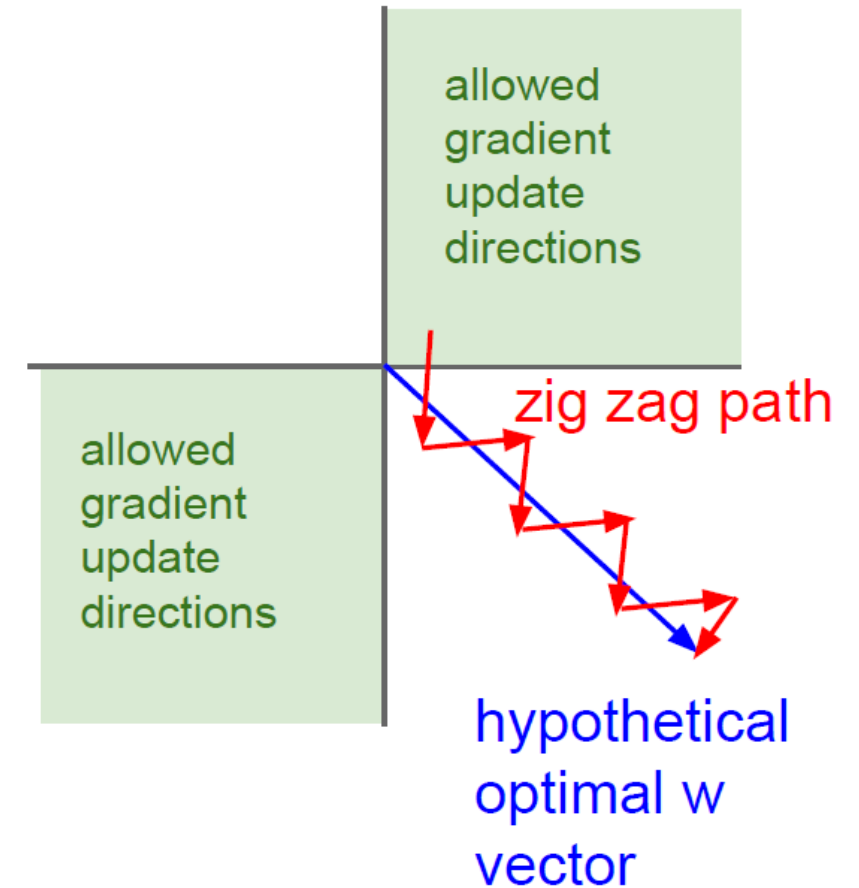
$$f(x) = \frac{1}{1 + e^{-x}}$$

- Squashes numbers to range [0, 1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron
- 3 Problems
 - Saturated neurons “kill” the gradients
 - Sigmoid outputs are not zero-centered
 - Exp() is a bit compute expensive

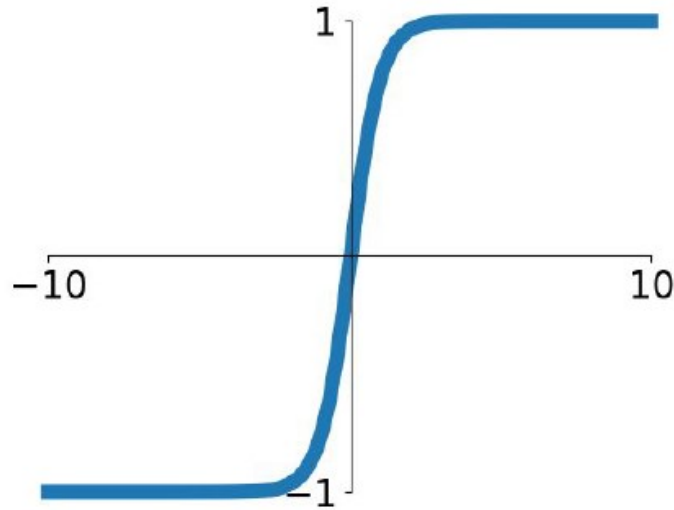
What happens when the input to a neuron is always positive

$$f\left(\sum_i w_i x_i + b\right)$$

- What can we say about the gradients on \mathbf{w} ?
 - Always all positive or all negative
 - This is also why you want zero-mean data



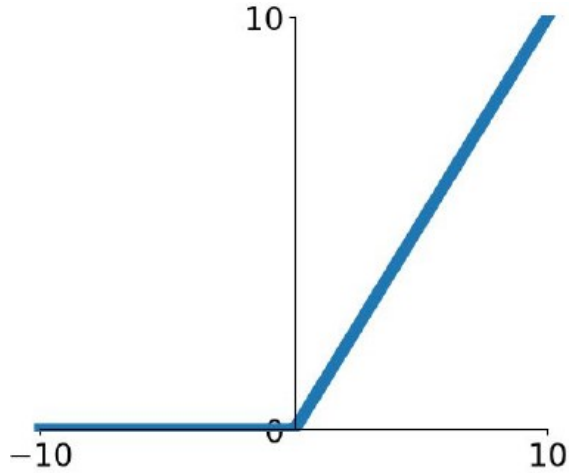
Tanh



$$f(x) = \tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Squashes numbers to range $[-1, 1]$
- Zero centered
- Still kills gradient when saturated

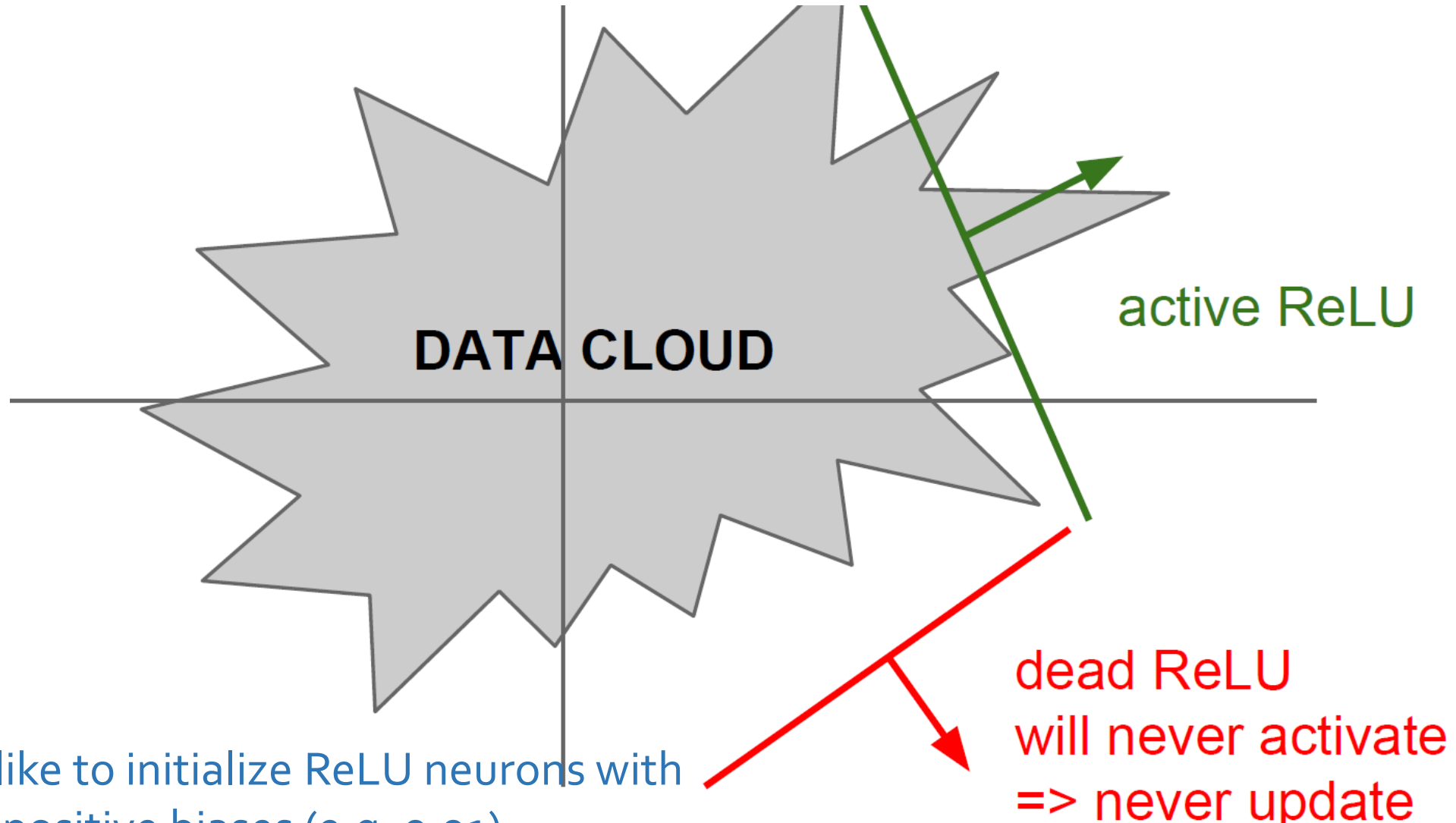
ReLU(Rectified Linear Unit)



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

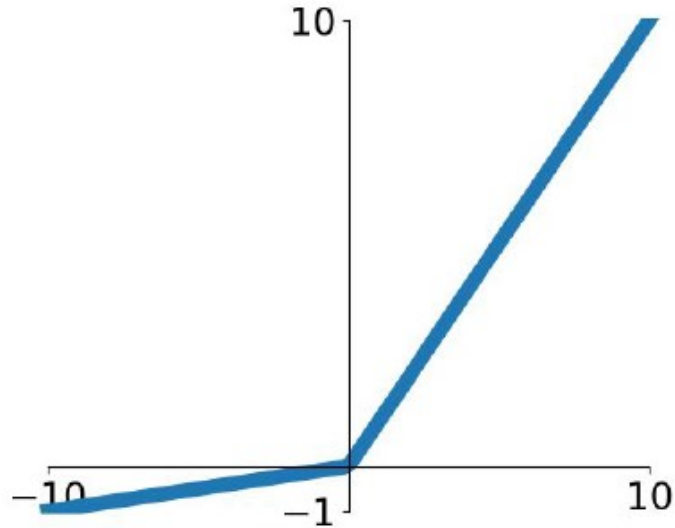
- Does not saturate
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice(e.g. 6x)
- Actually more biologically plausible than sigmoid
- Problems
 - Not zero-centered output
 - An annoyance – dead ReLU

Dead ReLU



People like to initialize ReLU neurons with slightly positive biases (e.g. 0.01)

Leaky ReLU, PReLU(Parametric-)

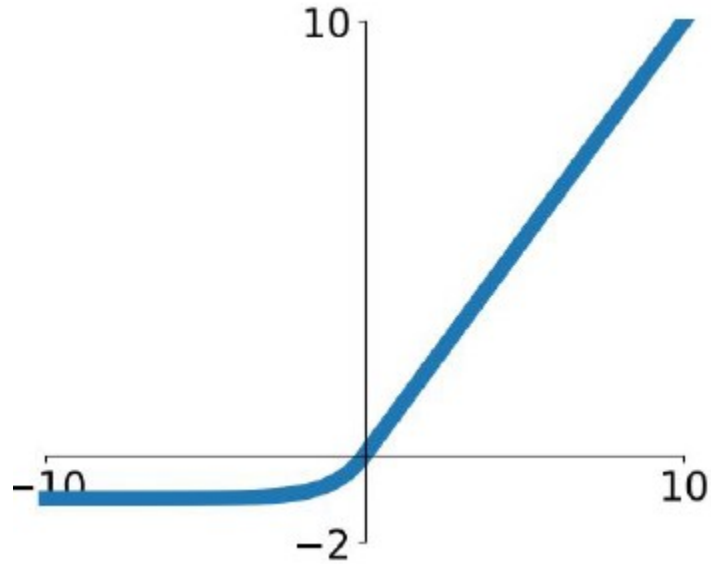


$$f(x) = \max(0.01x, x)$$

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

- Does not saturate
- Computationally efficient
- Converge much faster than sigmoid/tanh
- Will not 'die'

ELU(Exponential Linear Units)



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

- All benefits of ReLU
- Closer to zero mean output
- Computation requires exp()

Maxout

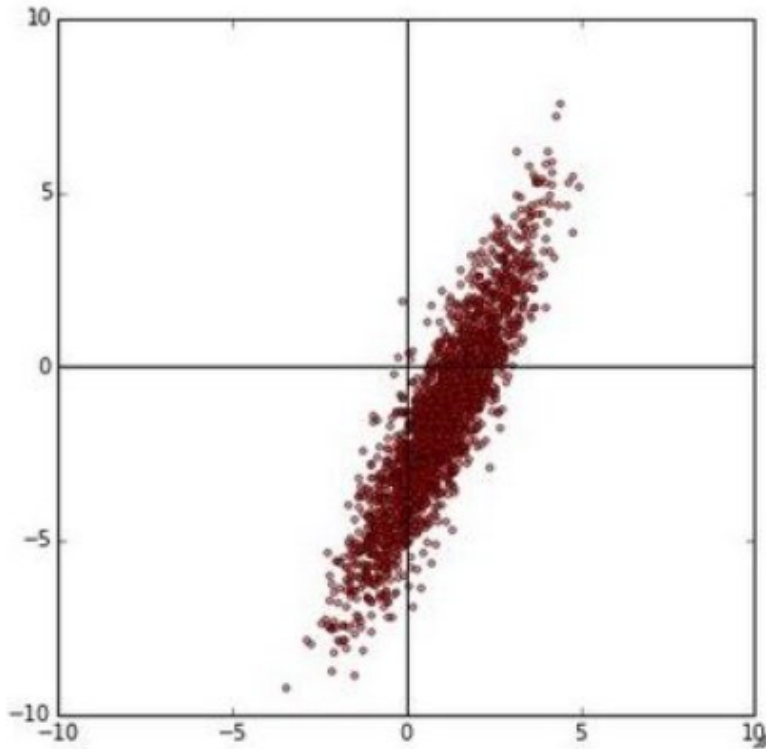
- Generalizes ReLU and Leaky ReLU
- Linear Regime! Does not saturate! Does not die!

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

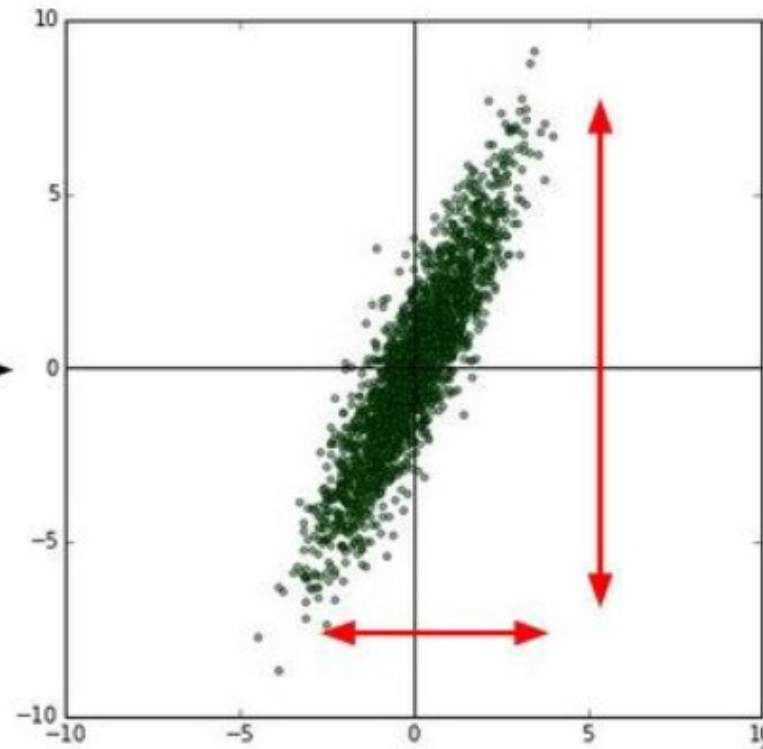
- Problem
 - Doubles the number of paramters/neuron

Data Preprocessing

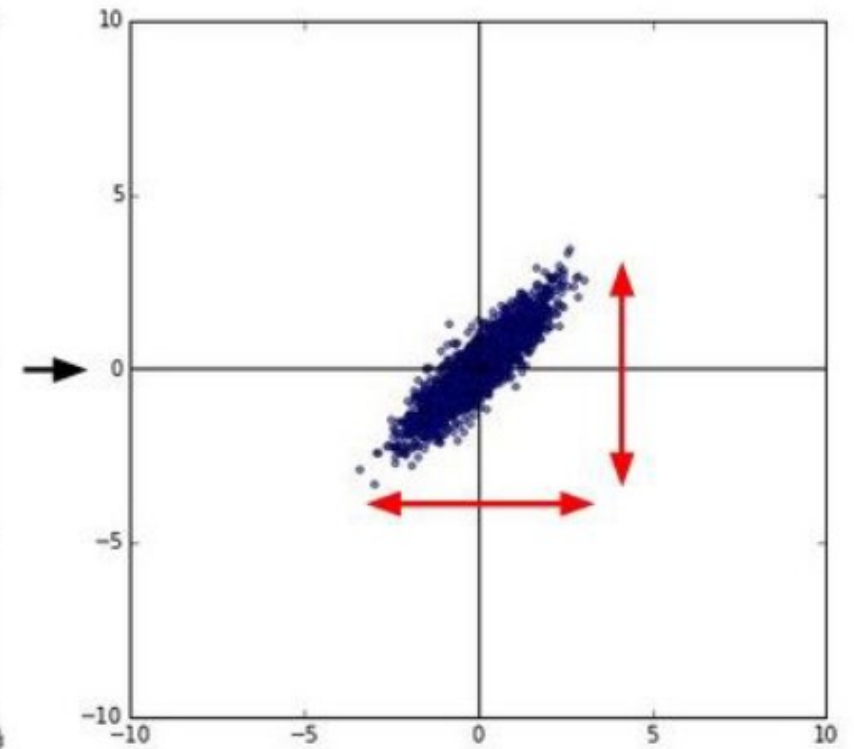
original data



zero-centered data



normalized data



In Practice for Images

- 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
- Not common to normalize variance

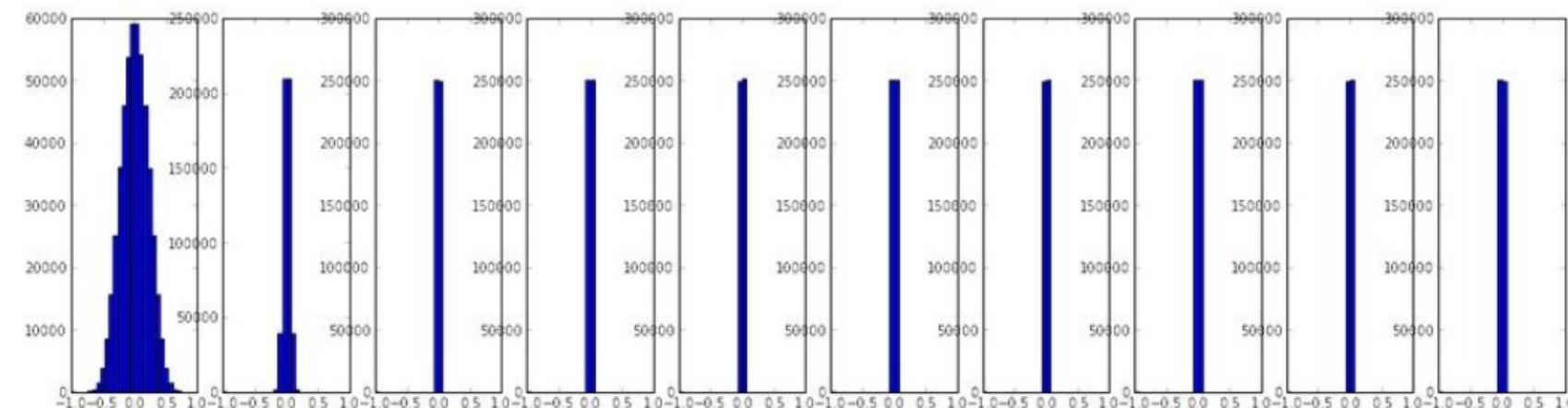
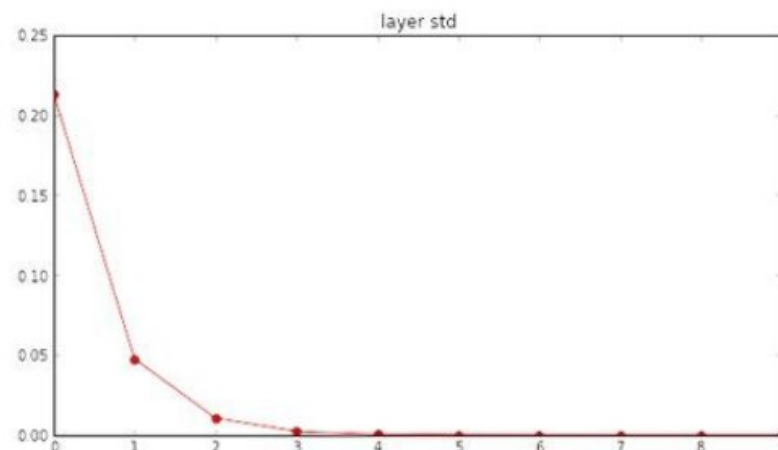
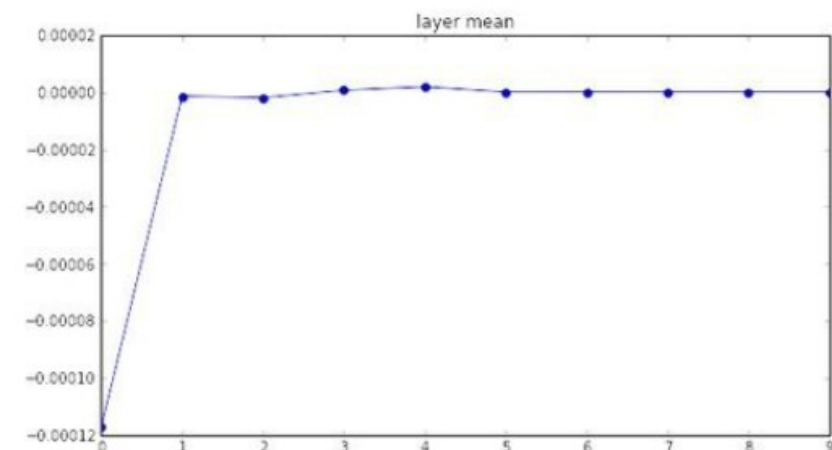
Weight Initialization

- First idea : Small random numbers
 - Gaussian with zero mean and $1e-2$ standard deviation

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

- Let's look at some activation statistics
 - 10-layer network with 500 neurons on each layer
 - Using tanh activation function

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?

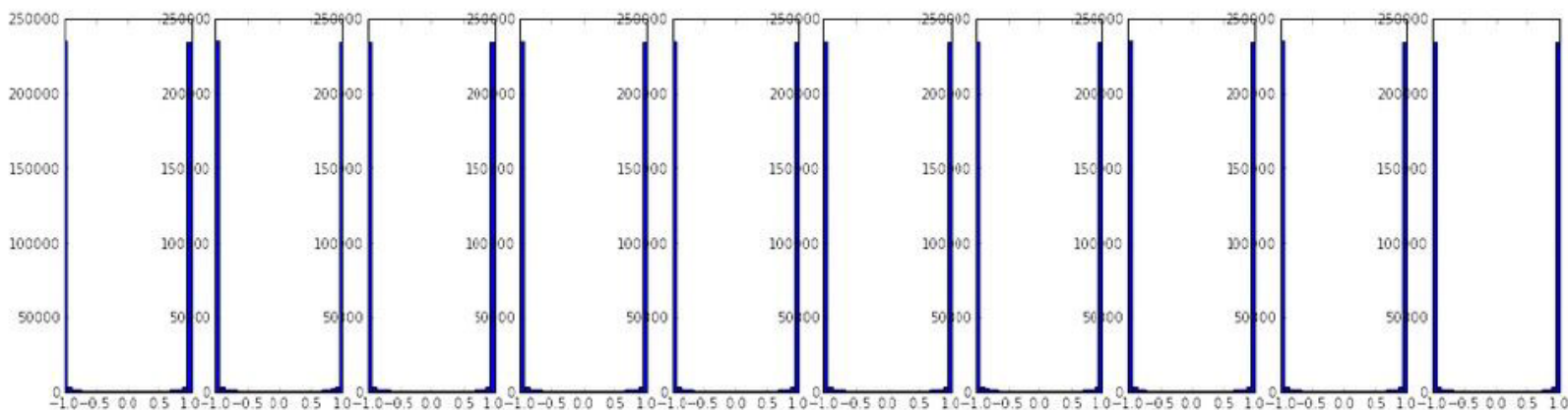
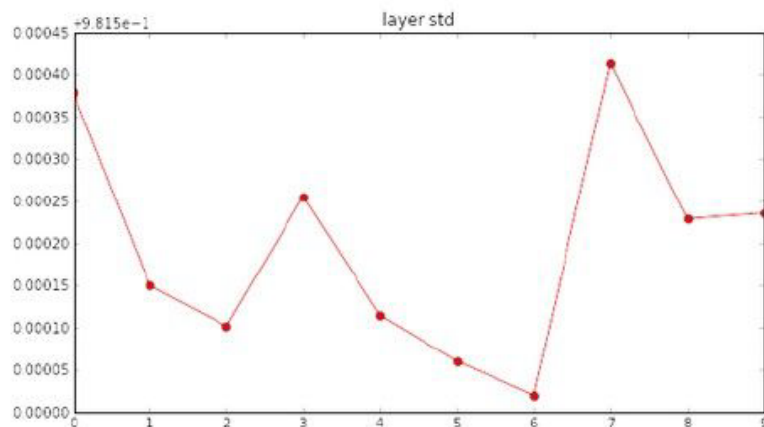
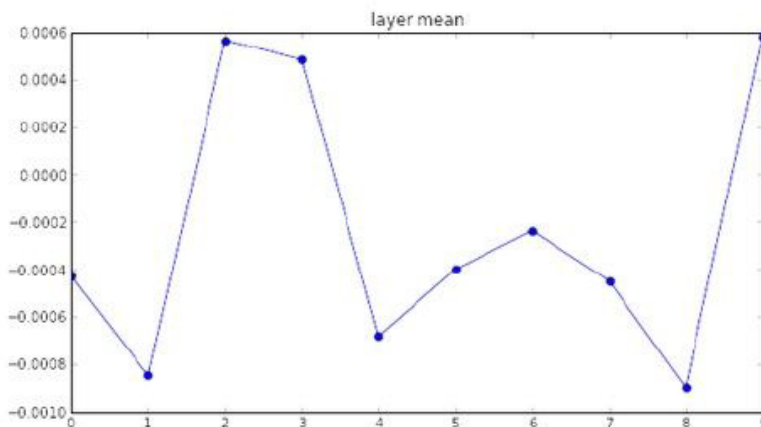
Hint: think about backward
pass for a $W \cdot X$ gate.


```
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.

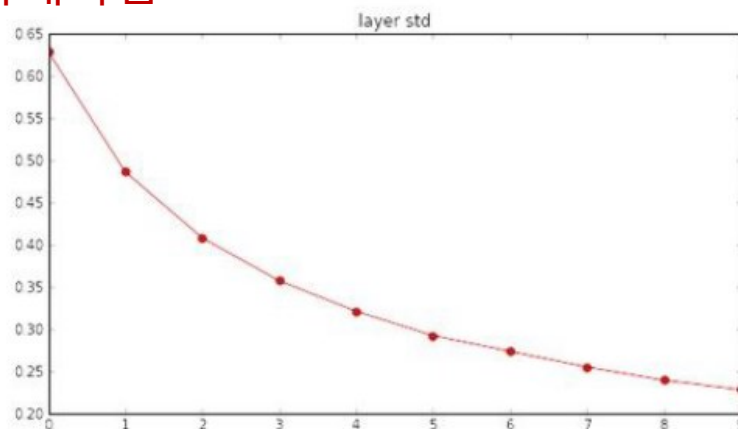
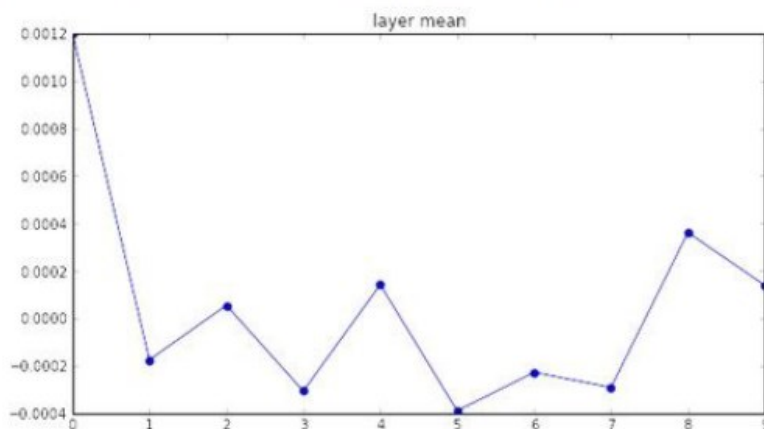


input layer had mean 0.001800 and std 1.001311
 hidden layer 1 had mean 0.001198 and std 0.627953
 hidden layer 2 had mean -0.000175 and std 0.486051
 hidden layer 3 had mean 0.000055 and std 0.407723
 hidden layer 4 had mean -0.000306 and std 0.357108
 hidden layer 5 had mean 0.000142 and std 0.320917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

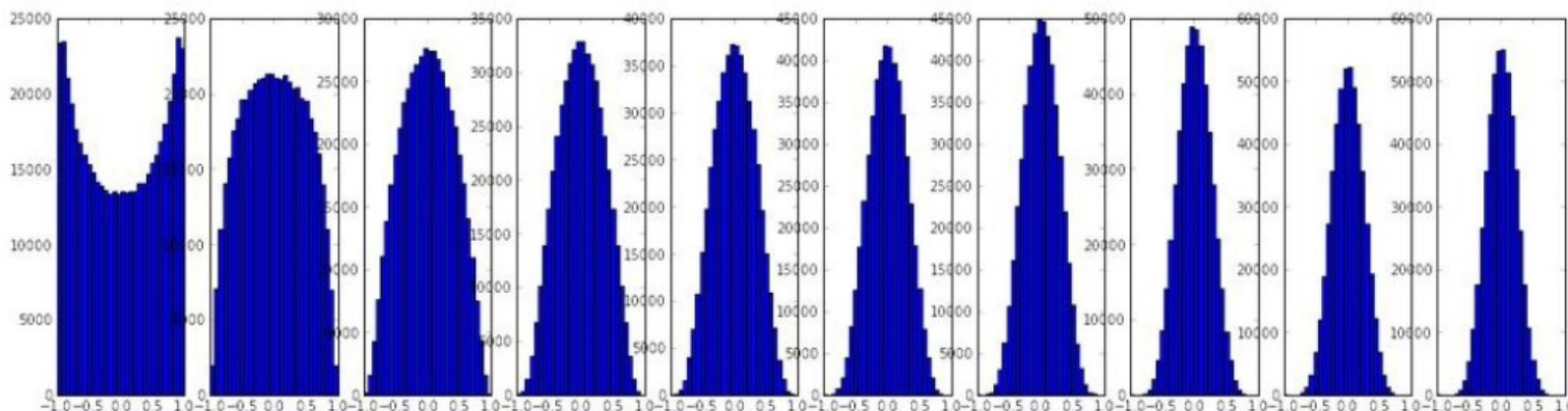
```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

Input neuron 수가 다른데 같은 initialization 값을 사용하는 것은 문제가 있음
 → input 이 많으면 더 작은 weight 값으로 시작해야함

“Xavier initialization”
 [Glorot et al., 2010]



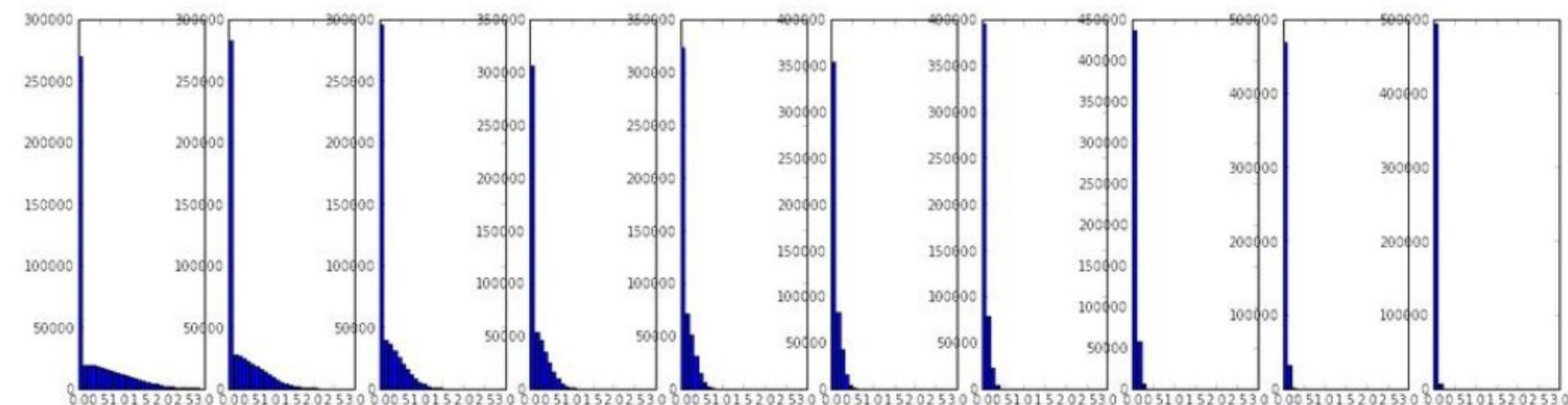
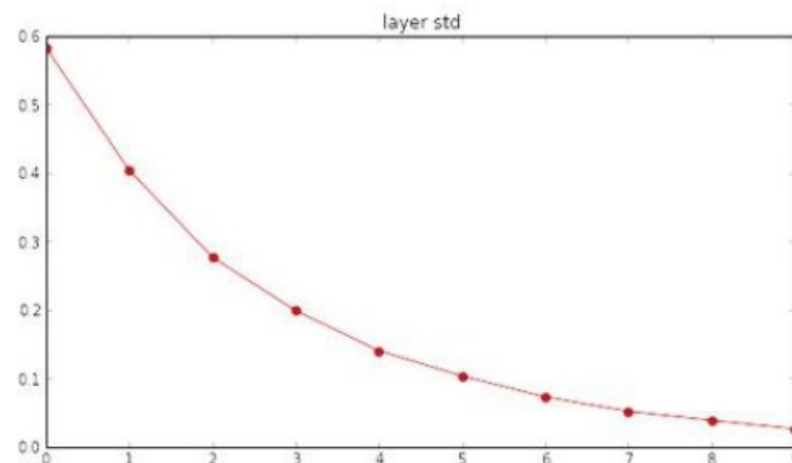
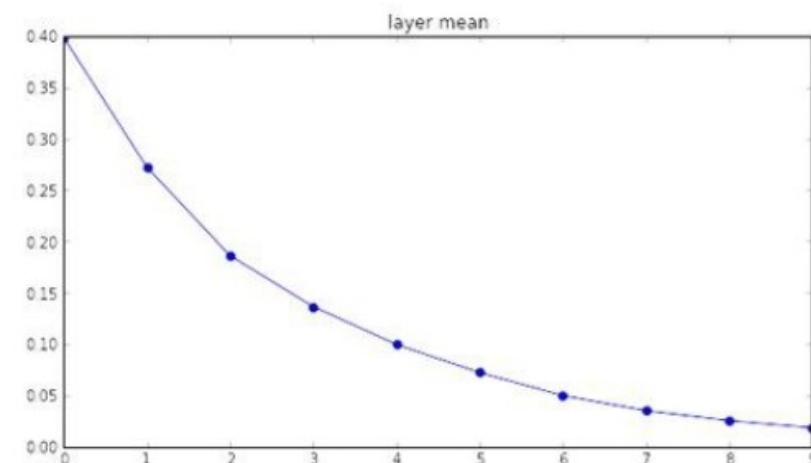
Reasonable initialization.
 (Mathematical derivation assumes linear activations)



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.



Weight Initialization

- Xavier Initialization
 - Activation function은 linear라고 가정하고, in/out의 variance를 같게 해보자

$$\text{forward: } \text{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}$$

$$\text{backward: } \text{Var}(W_i) = \frac{1}{n_{\text{out}}}$$

$$\text{Xavier Initialization: } \text{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

Weight Initialization

- He Initialization
 - Activation function을 ReLU나 PReLU로 하고, variance를 같게 해보자

ReLU

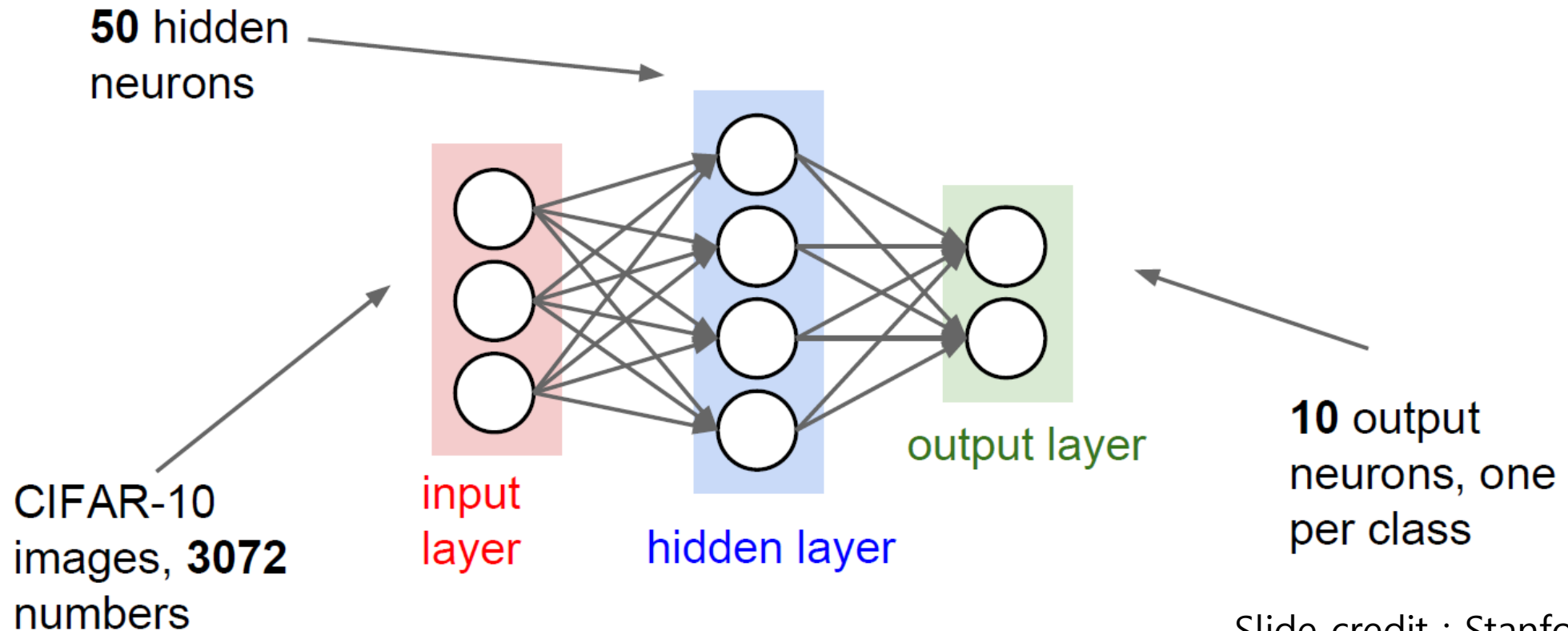
$$\left\{ \begin{array}{l} Var[w_l] = \frac{2}{n_l} \implies \text{standard deviation (std)} = \sqrt{\frac{2}{n_l}} \\ w_l \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_l}}\right) \text{ and } \mathbf{b} = 0 \end{array} \right.$$

PReLU

$$\frac{1}{2}(1 + a^2)n_l \underline{Var[w_l]} = 1$$

Babysitting the Learning Process

1. Preprocess the data
2. Choose the architecture
 - How many layers? How many hidden neurons? ...



Babysitting the Learning Process

3. Double check that the loss is reasonable

- Initial loss must be about $-\log(1/n)$ w/o regularization loss
- w/ regularization loss, loss must increase

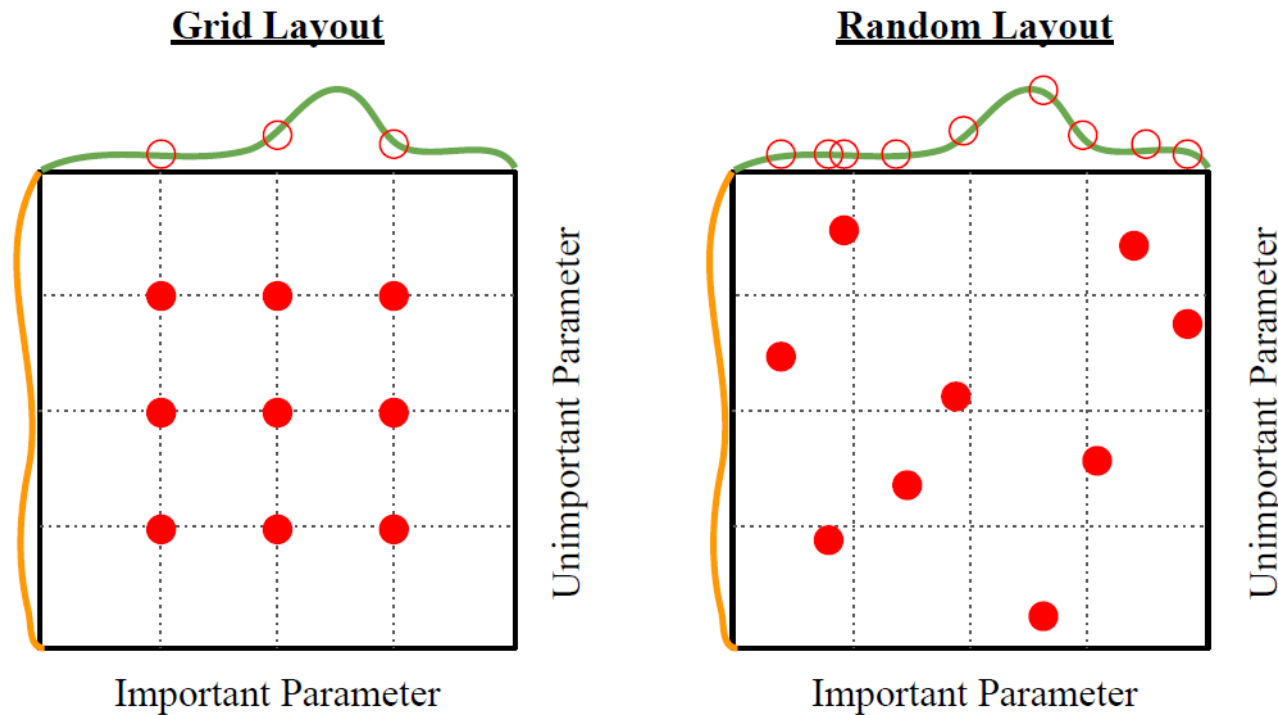
4. Let's try to train now → make sure that you can overfit very small portion of the training data

- Take the small examples(e.g. 20) from training set
- Turn off regularization
- Use simple vanilla sgd

Babysitting the Learning Process

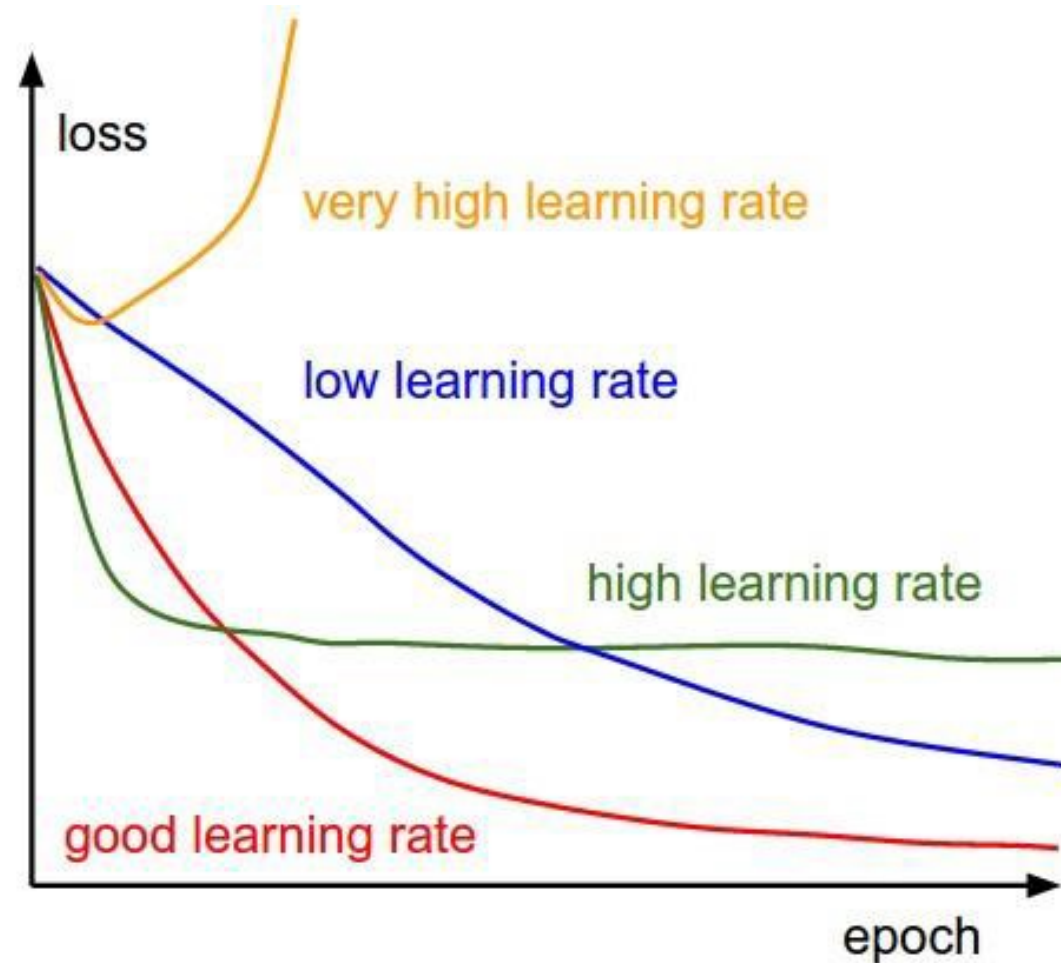
5. Search a good learning rate(cross-validation strategy)

- First stage : only a few epochs to get rough idea of what params work
- Second stage : longer running time, finer search
- Random Search vs Grid Search → Use Random Search!



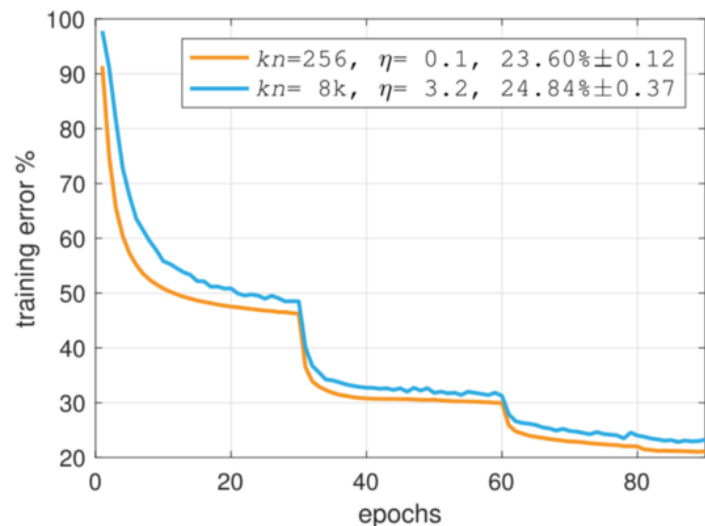
Learning Rate

- learning rate에 따라서 최종 결과가 달라짐

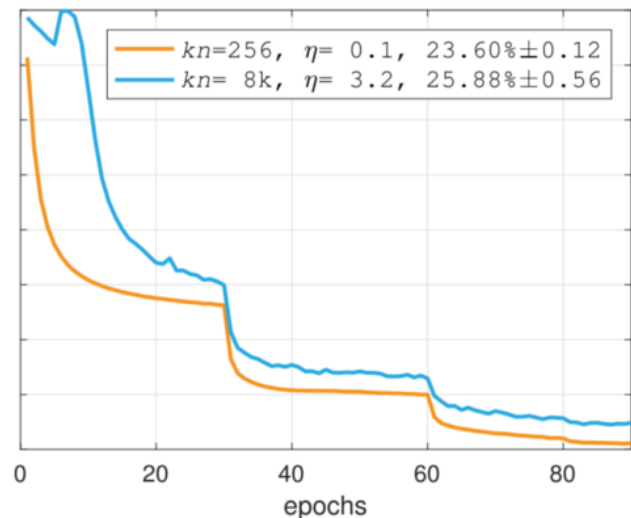


Learning Rate Decay

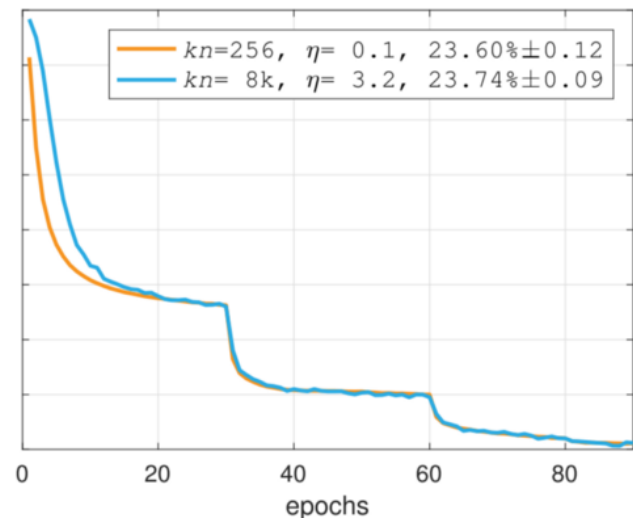
- Learning rate이 너무 크면 수렴을 못할 가능성이 있고, 너무 작으면 local minima 혹은 saddle point에서 못빠져나옴
- Learning Rate Decay
 - 처음에는 크게 움직이다가 일정 조건이 되면 learning rate을 낮춰서 점점 작게 움직이는 방법



(a) no warmup



(b) constant warmup



(c) gradual warmup

Cyclic Learning Rate

- Learning rate decay로 saddle point를 빠져나갈 수 있을까?
- Saddle point에서는 learning rate을 키워서 빠져나가는 것이 효과적일 수 있음 → Learning rate을 주기적으로 변경해보자

