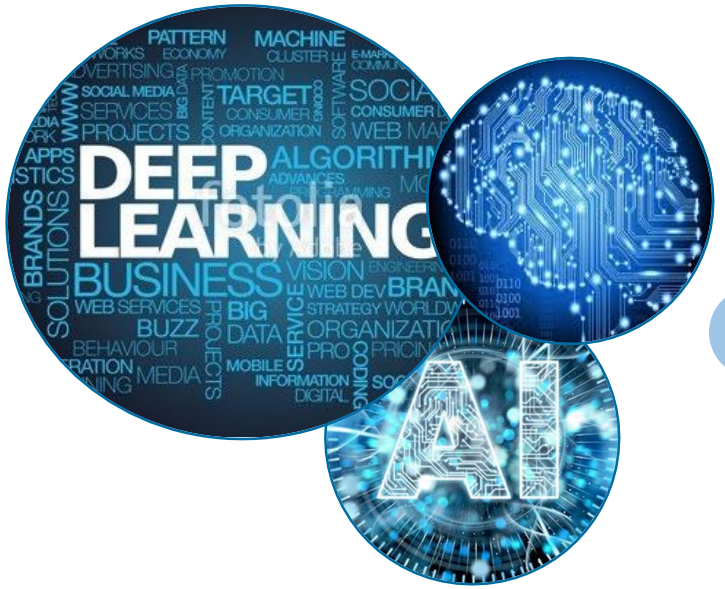
UNIVERSITY OF SCIENCE & TECHNOLOGY

Korea Institute of Science and Technology Information

2021-1 딥러닝기술 및 응용 - Paper Review

Weight Initialization

[Hinton2006] A fast learning algorithm for deep belief net.
[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.
[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.



KISTI-UST
Donghun Yang

2021. 04. 16. FRI.

Contents

01 Why do I need to initialize the weights properly?

02 Deep Belief Network

03 Initialize all weights to Zero

04 Initialize weights randomly

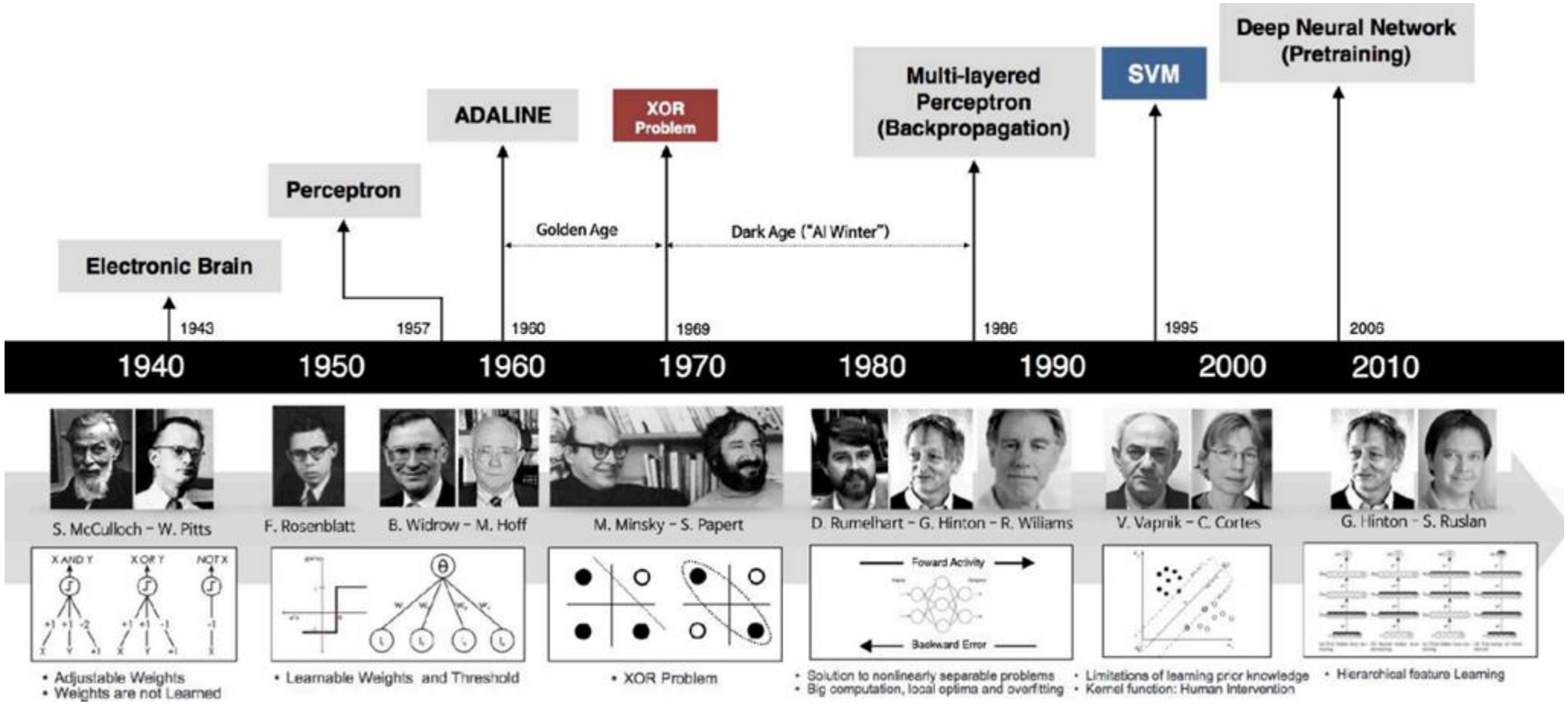
05 Xavier initialization for linear activation function

06 He initialization for ReLU activation function

07 Conclusion

01 Why do we need to initialize the weights properly?

History of AI



01 Why do we need to initialize the weights properly?

Geoffrey Hinton's summary of findings up to today

Geoffrey Hinton's summary of findings up to today

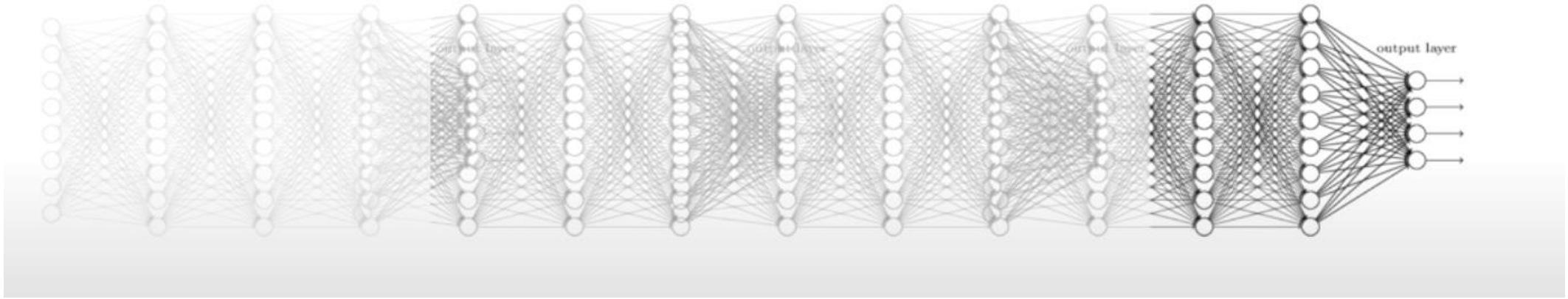
- Our labeled datasets were thousands of times too small.
- Our computers were millions of times too slow.
- We initialized the weights in a stupid way.
- We used the wrong type of non-linearity.

https://www.skynettoday.com/overviews/neural-net-history

01 Why do we need to initialize the weights properly?

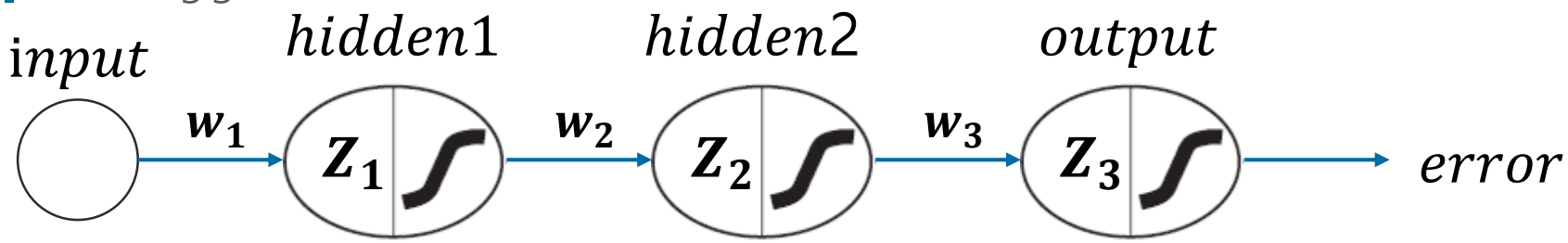
Vanishing gradient

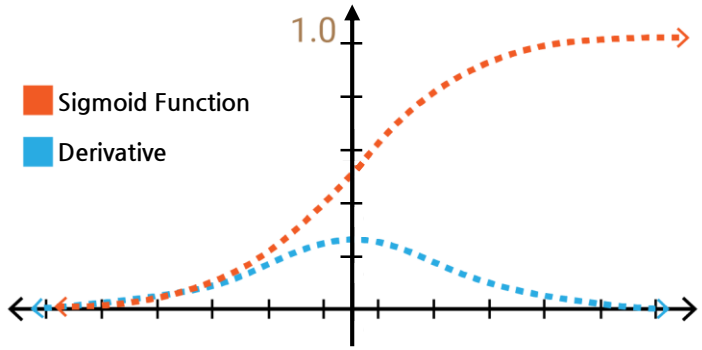
Vanishing gradient (NN winter2: 1986-2006)



01 Why do we need to initialize the weights properly?

Vanishing gradient





$$\frac{\partial error}{\partial w_3} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial w_3} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial z_3} * \frac{\partial z_3}{\partial w_3} = \frac{\partial error}{\partial Sigmoid(z_3)} * \frac{\partial Sigmoid(z_3)}{\partial z_3} * hidden2$$

<0.4

$$z_3 = w_3 * hidden2$$
$$output = Sigmoid(z_3)$$

$$\frac{\partial error}{\partial w_2} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial w_2} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial z_3} * \frac{\partial z_3}{\partial hidden2} * \frac{\partial hidden2}{\partial z_2} * \frac{\partial z_2}{\partial w_2}$$

$$= \frac{\partial error}{\partial Sigmoid(z_3)} * \frac{\partial Sigmoid(z_3)}{\partial z_3} * \frac{\partial z_3}{\partial Sigmoid(z_2)} * \frac{\partial Sigmoid(z_2)}{\partial z_2} * hidden1$$

$$= \frac{\partial error}{\partial Sigmoid(z_3)} * \frac{\partial Sigmoid(z_3)}{\partial z_3} * w_3 * \frac{\partial Sigmoid(z_2)}{\partial z_2} * hidden1$$

<0.4 * <1 * <0.4

$$z_2 = w_2 * hidden1$$
$$hiddn2 = Sigmoid(z_2)$$

$$z_1 = w_1 * input$$
$$hiddn1 = Sigmoid(z_1)$$

$$\frac{\partial error}{\partial w_1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w_1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial z_3} * \frac{\partial z_3}{\partial hidden2} * \frac{\partial hidden2}{\partial z_2} * \frac{\partial z_2}{\partial hidden1} * \frac{\partial hidden1}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

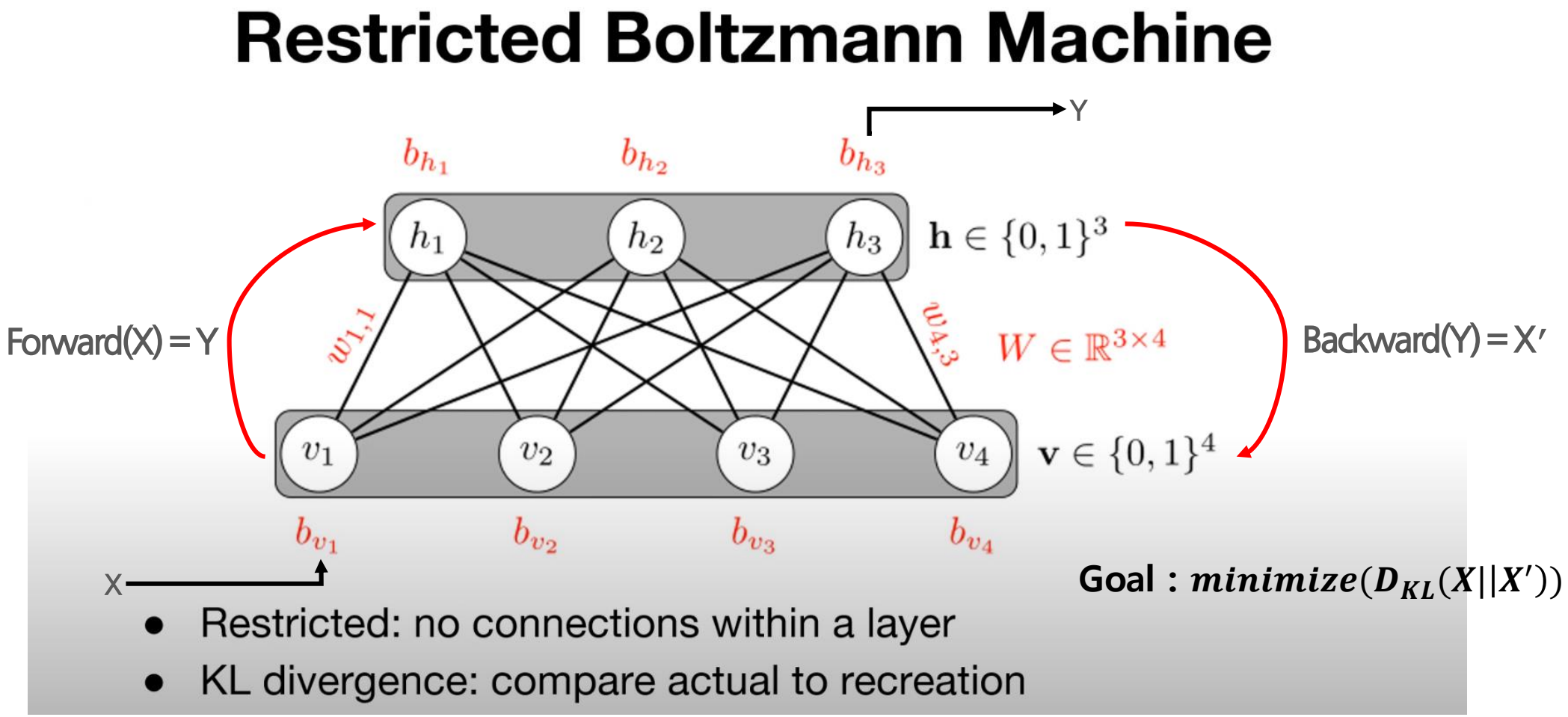
$$= \frac{\partial error}{\partial Sigmoid(z_3)} * \frac{\partial Sigmoid(z_3)}{\partial z_3} * \frac{\partial z_3}{\partial Sigmoid(z_2)} * \frac{\partial Sigmoid(z_2)}{\partial z_2} * \frac{\partial z_2}{\partial Sigmoid(z_1)} * \frac{\partial Sigmoid(z_1)}{\partial z_1} * input$$

$$= \frac{\partial error}{\partial Sigmoid(z_3)} * \frac{\partial Sigmoid(z_3)}{\partial z_3} * w_3 * \frac{\partial Sigmoid(z_2)}{\partial z_2} * w_2 * \frac{\partial Sigmoid(z_1)}{\partial z_1} * input$$

<0.4 * <1 * <0.4 * <1 * <0.4

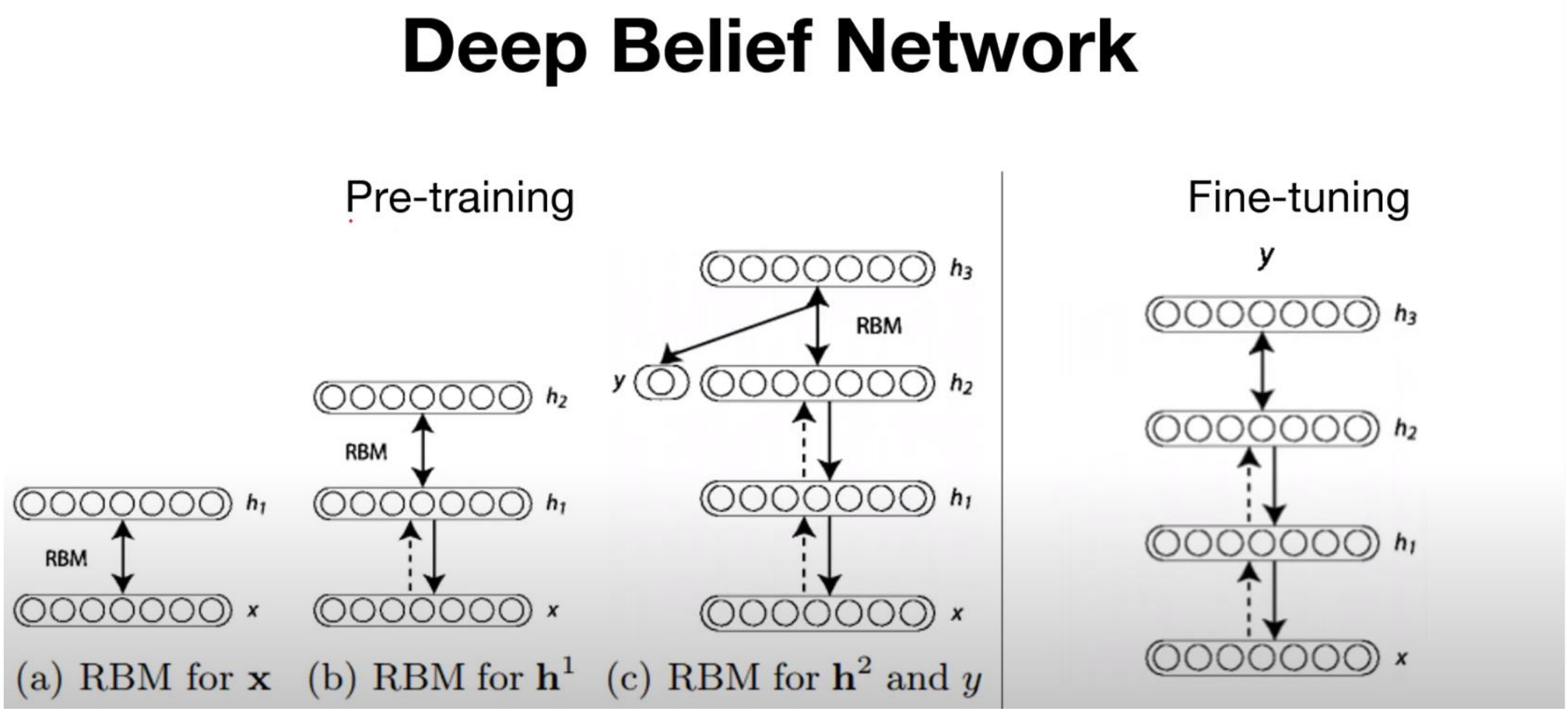
02 Deep Belief Network

[Hinton2006] A fast learning algorithm for deep belief net. - Restricted Boltzmann Machine (RBM)



02 Deep Belief Network

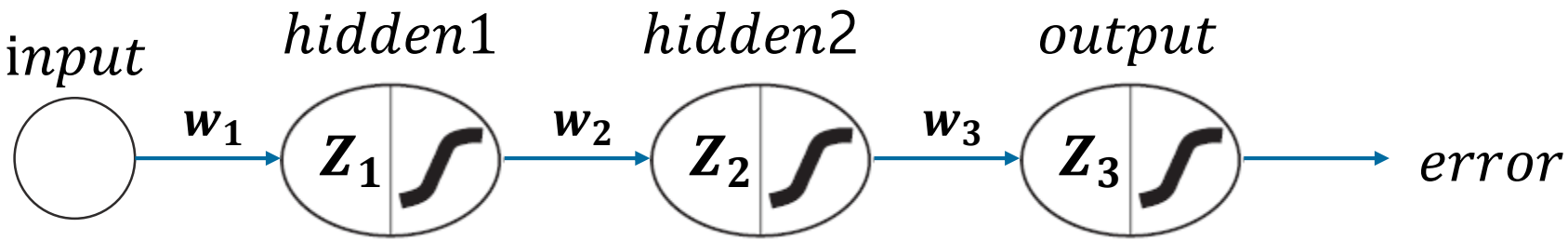
[Hinton2006] A fast learning algorithm for deep belief net. - Restricted Boltzmann Machine (RBM)

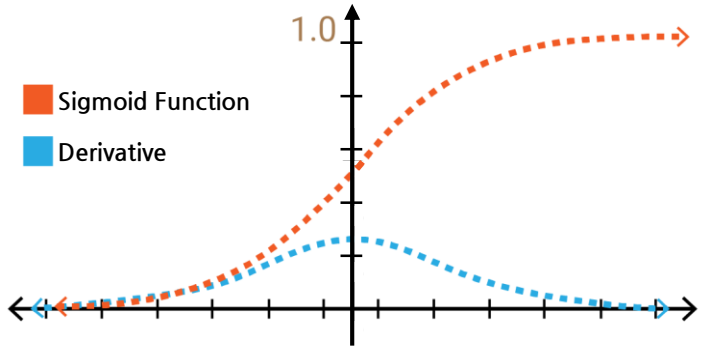


03 Initialize all weights to Zero

Why not initialize all weights to ZERO?

All weights are initialized to Zero





$$\frac{\partial error}{\partial w_3} = \frac{\partial error}{\partial Sigmoid(z_3)} * \frac{\partial Sigmoid(z_3)}{\partial z_3} * hidden2$$

<0.4

$$\frac{\partial error}{\partial w_2} = \frac{\partial error}{\partial Sigmoid(z_3)} * \frac{\partial Sigmoid(z_3)}{\partial z_3} * w_3 * \frac{\partial Sigmoid(z_2)}{\partial z_2} * hidden1$$

<0.4 * 0 * <0.4

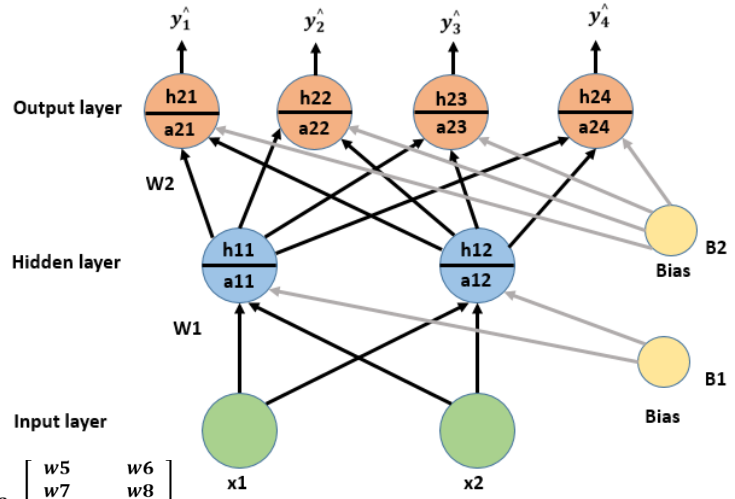
$$\frac{\partial error}{\partial w_1} = \frac{\partial error}{\partial Sigmoid(z_3)} * \frac{\partial Sigmoid(z_3)}{\partial z_3} * w_3 * \frac{\partial Sigmoid(z_2)}{\partial z_2} * w_2 * \frac{\partial Sigmoid(z_1)}{\partial z_1} * input$$

<0.4 * 0 * <0.4 * 0 * <0.4

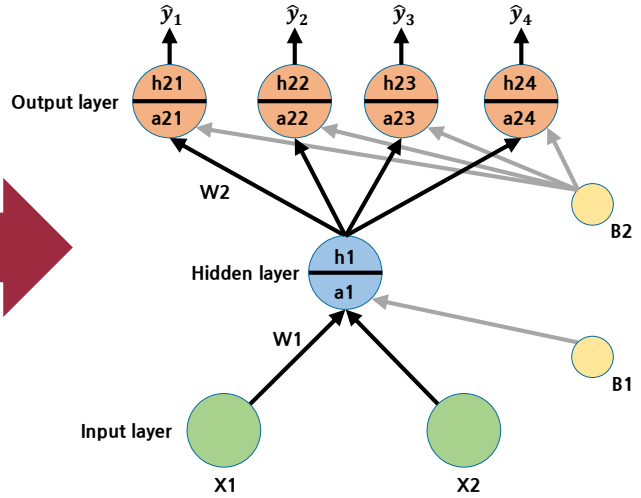
Gradient vanishing problem

03 Initialize all weights to Zero

Why not initialize all weights to CONSTANT VALUE?



$$W1 = \begin{bmatrix} w1 & w2 \\ w3 & w4 \end{bmatrix} \quad W2 = \begin{bmatrix} w5 & w6 \\ w7 & w8 \\ w9 & w10 \\ w11 & w12 \end{bmatrix}$$



Forward Pass

a11 = w1x1 + w2x2 + b1
a12 = w3x1 + w4x2 + b2
a11 = a12 (assume b1 = b2) then h11 = h12
Similarly a21 = a22 = a23 =a24
then h21 = h22 = h23 =h24

Backward Pass

$$\nabla w1 = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a21} \frac{\partial a21}{\partial h11} \frac{\partial h11}{\partial a11} . x1$$

$$\nabla w3 = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a21} \frac{\partial a21}{\partial h11} \frac{\partial h11}{\partial a12} . x1$$

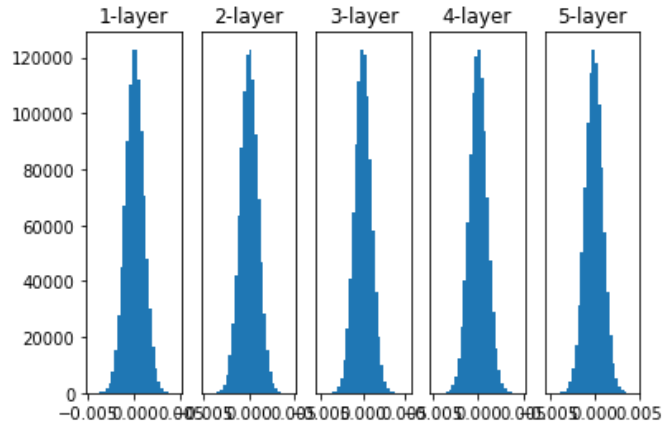
here, $\frac{\partial h11}{\partial a12} = \frac{\partial h11}{\partial a11}$ hence $\nabla w1 = \nabla w3$

Symmetry breaking problem

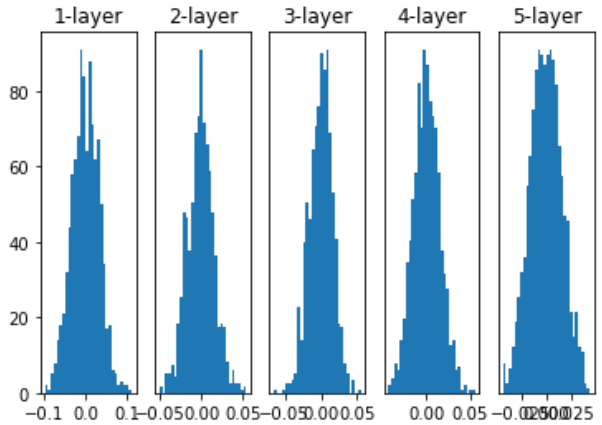
04 Initialize weights randomly

Can we initialize weights RANDOMLY to SMALL VALUE? - *W ~ N(0,0.001)*

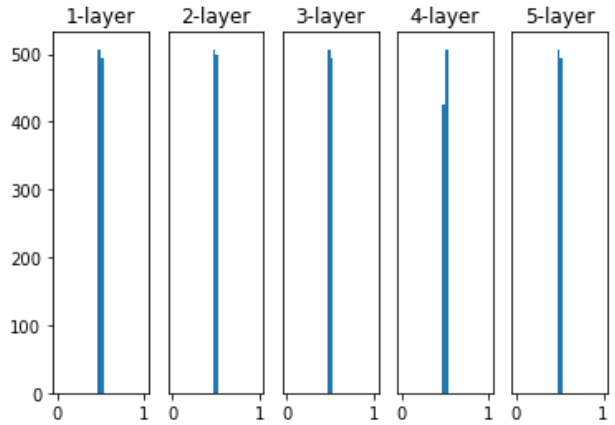
Layer # : 5 (1000 node in each layer)
X = np.random.randn(1, 1000)
W = np.random.randn(node_num, node_num)*0.001 in each layer



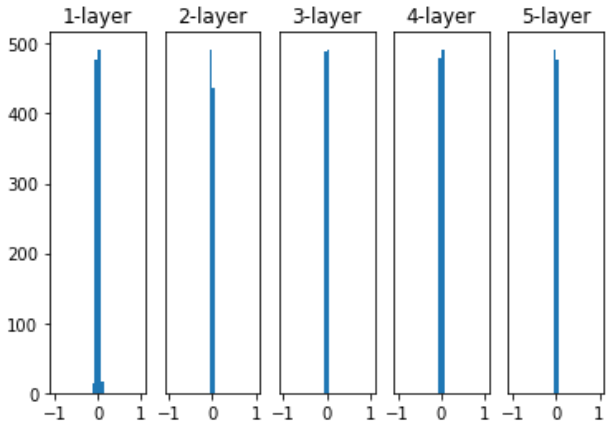
W



$WX = w_1x_1 + w_2x_2 + w_3x_3 + \ \ldots$



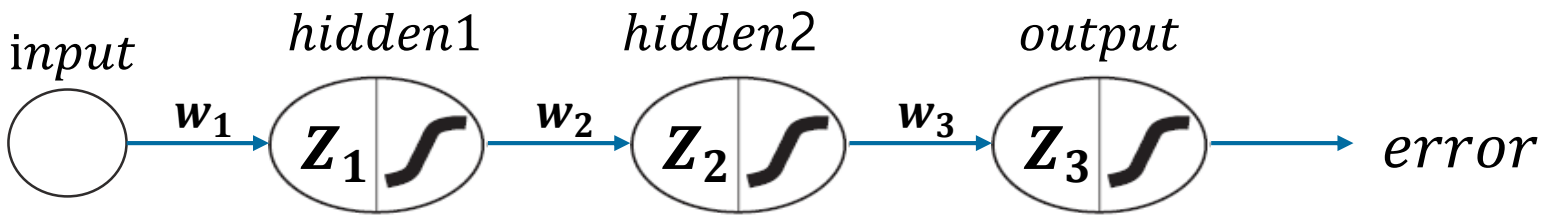
$Sigmoid(WX + b)$

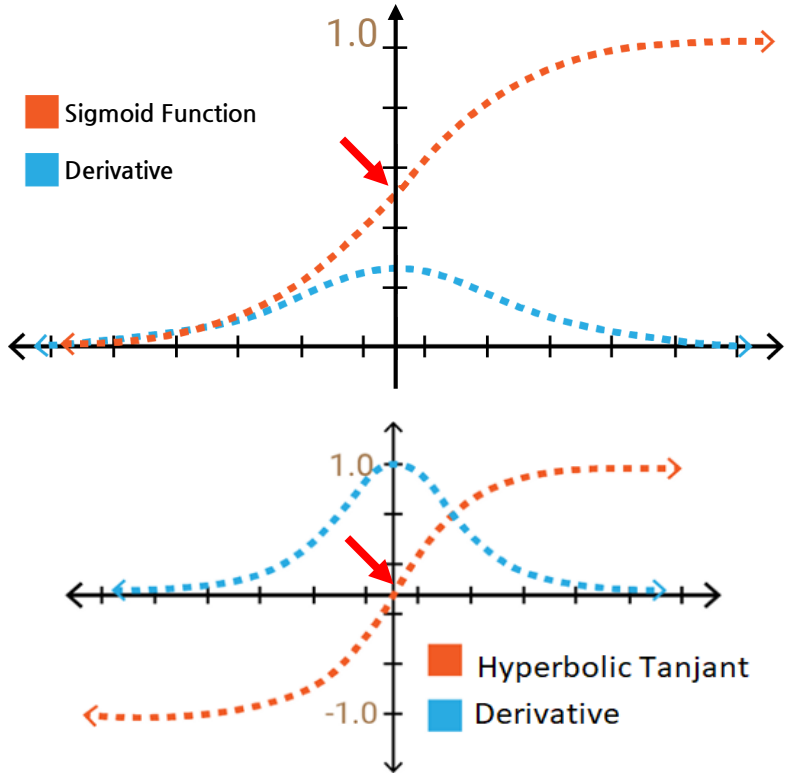


$\tanh(WX + b)$

04 Initialize weights randomly

Can we initialize weights RANDOMLY to SMALL VALUE? - *W ~ N(0,0.001)*





$$\frac{\partial error}{\partial w_3} = \frac{\partial error}{\partial activation(z_3)} * \frac{\partial activation(z_3)}{\partial z_3} * hidden2$$

~0.4 or 1

$$\frac{\partial error}{\partial w_2} = \frac{\partial error}{\partial activation(z_3)} * \frac{\partial activation(z_3)}{\partial z_3} * w_3 * \frac{\partial activation(z_2)}{\partial z_2} * hidden1$$

~0.4 or 1 ~0 ~0.4 or 1

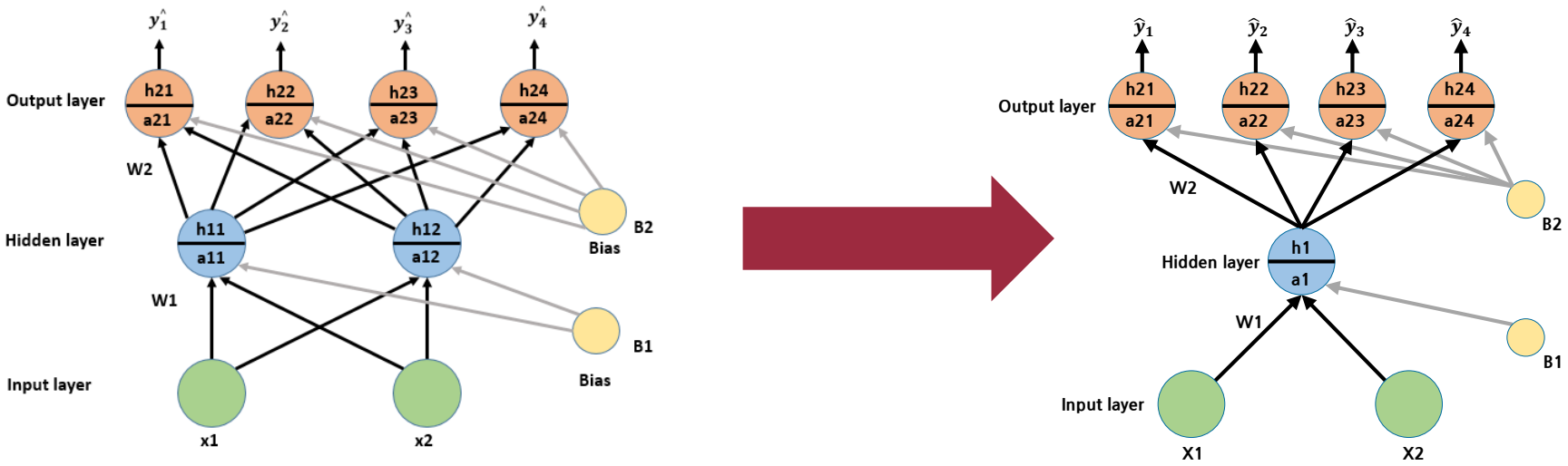
$$\frac{\partial error}{\partial w_1} = \frac{\partial error}{\partial activation(z_3)} * \frac{\partial activation(z_3)}{\partial z_3} * w_3 * \frac{\partial activation(z_2)}{\partial z_2} * w_2 * \frac{\partial activation(z_1)}{\partial z_1} * input$$

~0.4 or 1 ~0 ~0.4 or 1 ~0 ~0.4 or 1

Alleviate gradient vanishing problem?

04 Initialize weights randomly

Can we initialize weights RANDOMLY to SMALL VALUE? - *W ~ N(0,0.001)*

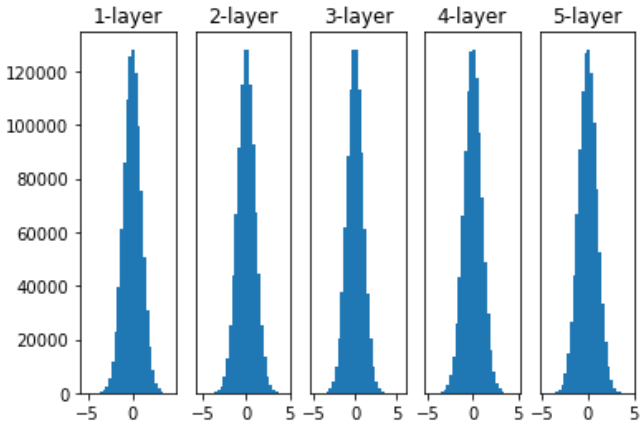


But, still symmetry breaking problem

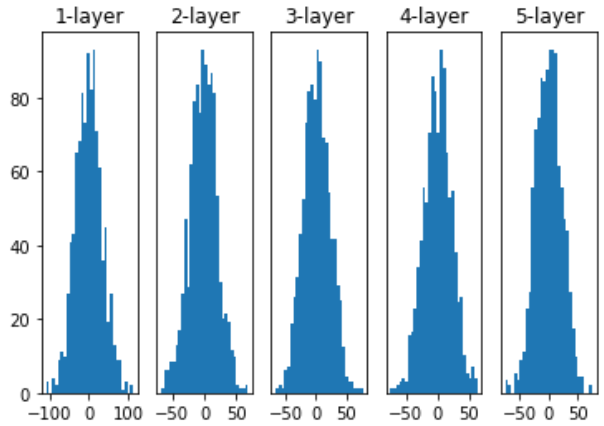
04 Initialize weights randomly

Can we initialize weights RANDOMLY to LARGE VALUE? - $W \sim N((0,1)$

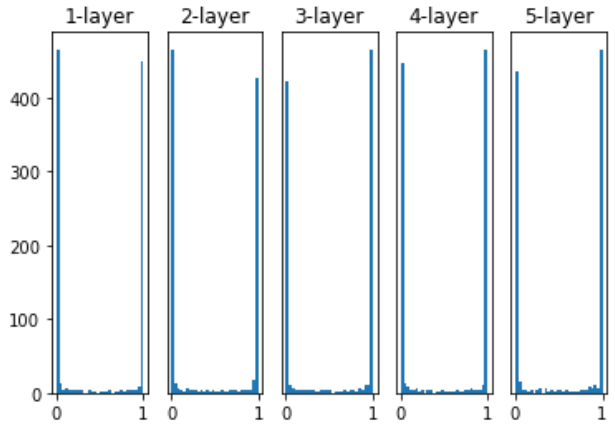
Layer # : 5 (1000 node in each layer)
X = np.random.randn(1, 1000)
W = np.random.randn(node_num, node_num) in each layer



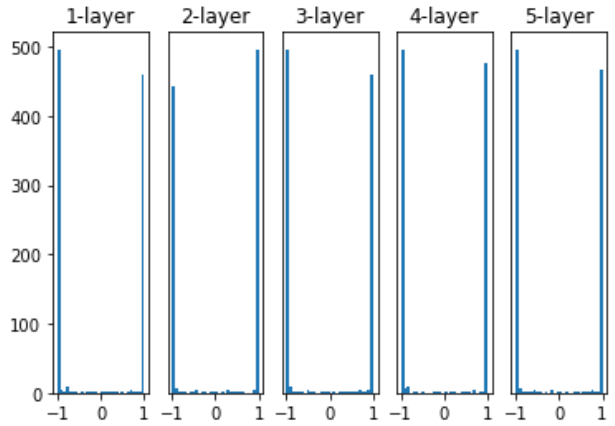
W



$WX = w_1x_1 + w_2x_2 + w_3x_3 + \ \ldots$



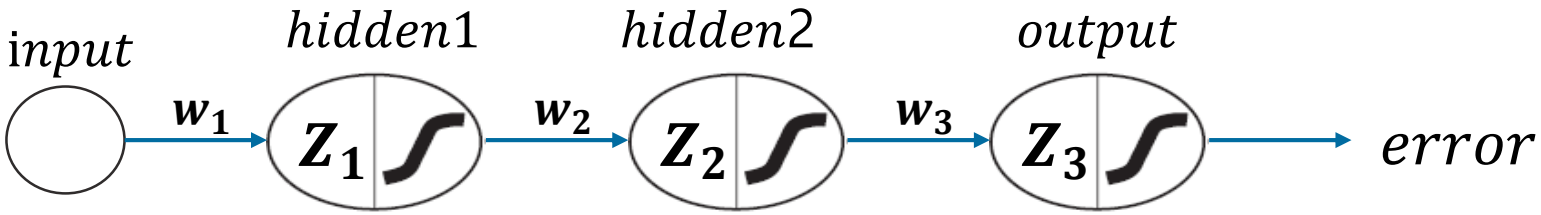
$Sigmoid(WX + b)$

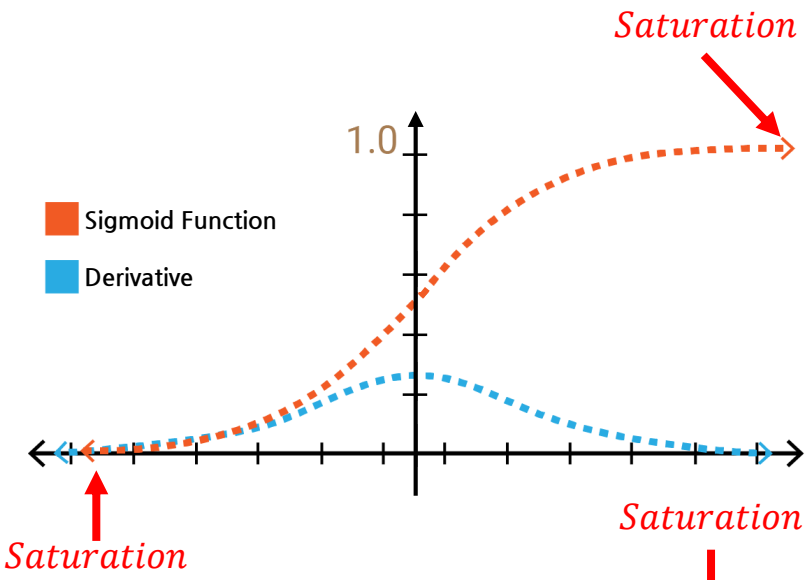


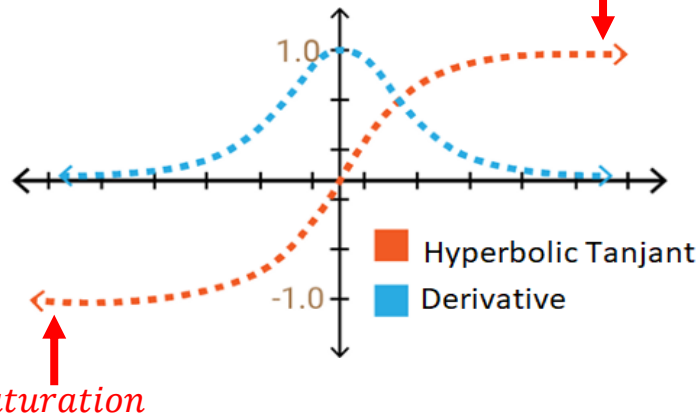
$\tanh(WX + b)$

04 Initialize weights randomly

Can we initialize weights RANDOMLY to LARGE VALUE? - *W ~ N((0, 1)*







$$\frac{\partial error}{\partial w_3} = \frac{\partial error}{\partial activation(z_3)} * \frac{\partial activation(z_3)}{\partial z_3} * hidden2$$

~0

$$\frac{\partial error}{\partial w_2} = \frac{\partial error}{\partial activation(z_3)} * \frac{\partial activation(z_3)}{\partial z_3} * w_3 * \frac{\partial activation(z_2)}{\partial z_2} * hidden1$$

~0 ~0

$$\frac{\partial error}{\partial w_1} = \frac{\partial error}{\partial activation(z_3)} * \frac{\partial activation(z_3)}{\partial z_3} * w_3 * \frac{\partial activation(z_2)}{\partial z_2} * w_2 * \frac{\partial activation(z_1)}{\partial z_1} * input$$

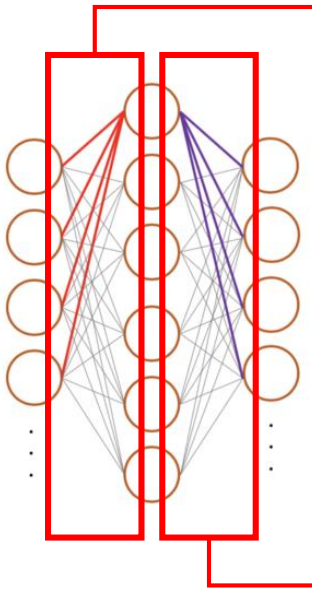
~0 ~0 ~0

Gradient vanishing problem

05 Xavier initialization for linear activation function

[Lecun1998] Efficient BackProp

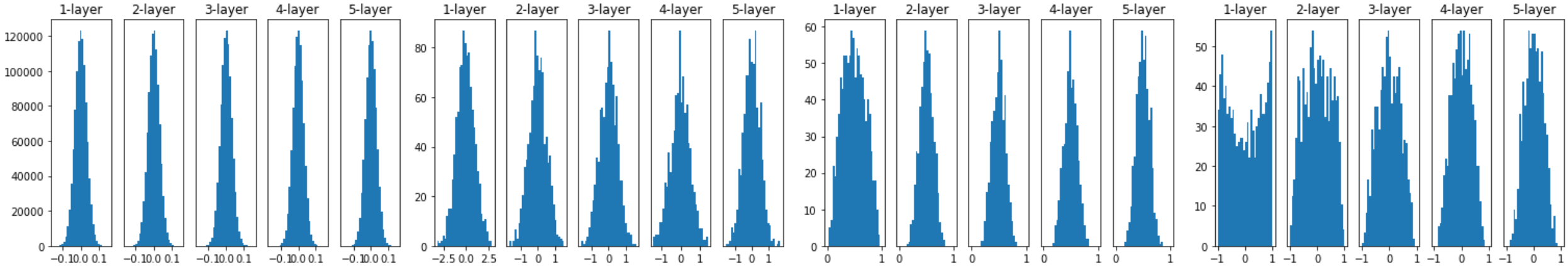
$$W \sim N\left(0, Var(W)\right)$$
$$Var(W) = \frac{1}{n_{in}}$$



Fan_in = input node # = 4
Fan_out = output node # = 6

Fan_in = input node # = 6
Fan_out = output node # = 4

Layer # : 5 (1000 node in each layer)
X = np.random.randn(1, 1000)
W = np.random.randn(node_num, node_num) in each layer



W $\quad WX = w_1x_1 + w_2x_2 + w_3x_3 + \ \ldots$ $\quad Sigmoid(WX + b)$ $\quad \tanh(WX + b)$

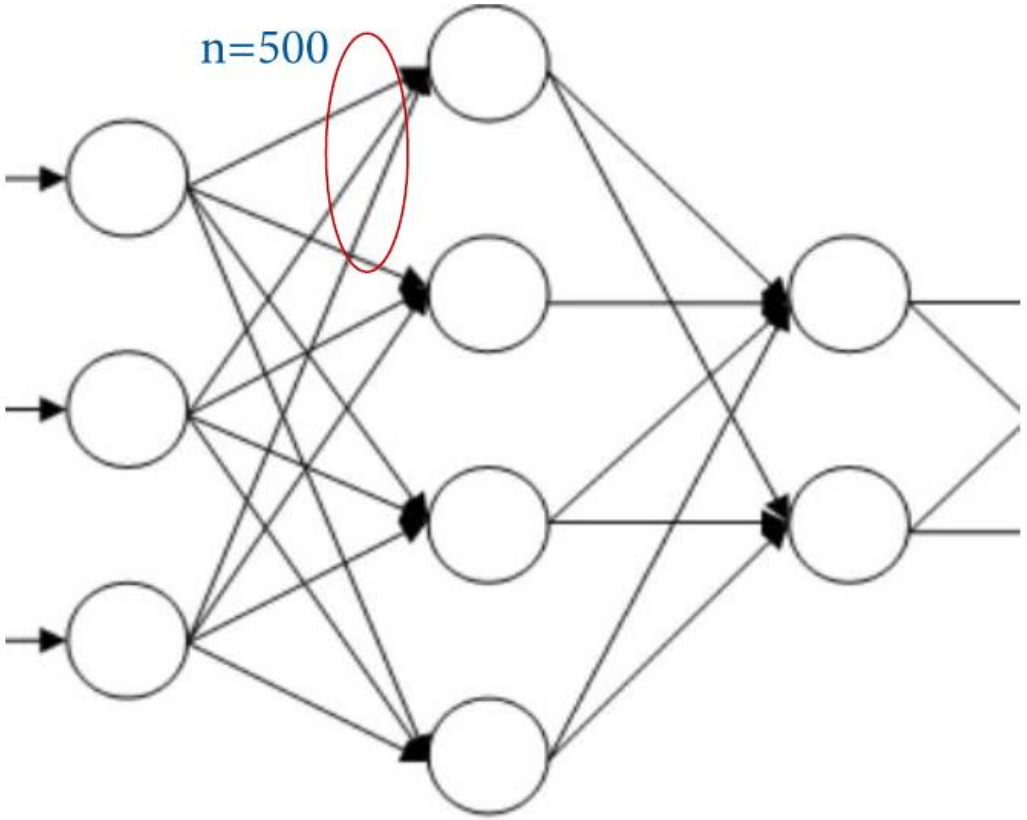
Solve all problem?

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

$$W,X \sim N(0,1)$$
$$Z = \sum WX$$
$$h = Sigmoid(Z)$$

$$E(Z) = 0$$
$$Var(Z) = 1$$



05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

$$\mathrm{Var}(X) = \mathrm{E}[X^2] - \mathrm{E}[X]^2$$

$$\mathrm{Var}(X + a) = \mathrm{Var}(X)$$

$$\mathrm{Var}(aX) = a^2\,\mathrm{Var}(X)$$

Cov(X,X)

$$\mathrm{Var}(aX + bY) = a^2\,\mathrm{Var}(X) + b^2\,\mathrm{Var}(Y) + 2ab\,\mathrm{Cov}(X, Y)$$

$$\mathrm{Var}(aX - bY) = a^2\,\mathrm{Var}(X) + b^2\,\mathrm{Var}(Y) - 2ab\,\mathrm{Cov}(X, Y)$$

If X,Y are uncorrelated or independent, we can remove these terms

$$\mathrm{Var}\left(\sum_{i=1}^{N} X_i\right) = \sum_{i,j=1}^{N} \mathrm{Cov}(X_i, X_j) = \sum_{i=1}^{N} \mathrm{Var}(X_i) + \sum_{i \neq j} \mathrm{Cov}(X_i, X_j)$$

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

$$\mathrm{Var}(aX+bY) \equiv E[(aX+bY)(aX+bY)] - E[aX+bY]E[aX+bY] \qquad \mathrm{Var}(X) \equiv E[X^2] - E[X]^2$$

$$= E[a^2X^2 + 2abXY + b^2Y^2] - E[aX+bY]^2$$

$$= E[a^2X^2] + 2E[abXY] + E[b^2Y^2] - E[aX+bY]^2$$

$$= a^2E[X^2] + 2abE[XY] + b^2E[Y^2] - E[aX+bY]^2$$

$$= a^2E[X^2] + 2abE[XY] + b^2E[Y^2] - \{E(aX)+E(bY)\}\{E(aX)+E(bY)\}$$

$$= a^2E[X^2] + 2abE[XY] + b^2E[Y^2] - \{aE[X]+bE[Y]\}^2$$

$$= a^2E[X^2] + 2abE[XY] + b^2E[Y^2] - (a^2E[X]^2 + 2abE[X]E[Y] + b^2E[Y]^2)$$

$$= 2abE[XY] - 2abE[X]E[Y]$$

$$a^2(E[X^2] - E[X]^2) + b^2(E[Y^2] - E[Y]^2) + 2ab(E[XY] - E[X]E[Y])$$

$$= a^2\mathrm{Var}(X) + b^2\mathrm{Var}(Y) + 2ab\,\mathrm{Cov}(X,Y)$$

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

$$\mathrm{Var}(s) = \mathrm{Var}\left(\sum_i^n w_i x_i\right)$$

See this as one variable

Each $\{w_i x_i\}$ is independent

$$= \sum_i^n \mathrm{Var}(w_i x_i)$$

$$\mathrm{Var}\left(\sum_{i=1}^n X_i\right) = \sum_{i=1}^n \mathrm{Var}(X_i)$$

two variables w and x are independent. see next slide

$$= \sum_i^n [E(w_i)]^2 \mathrm{Var}(x_i) + E[(x_i)]^2 \mathrm{Var}(w_i) + \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

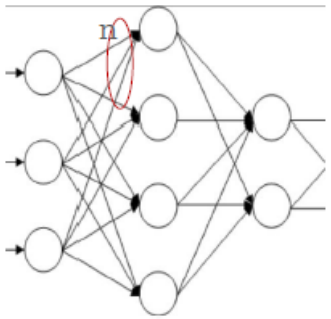
assumption: zero mean inputs and weights

$$= \sum_i^n \mathrm{Var}(x_i)\mathrm{Var}(w_i)$$

Assumptoin: w_i, x_i are drawn from identical distributions, respectively

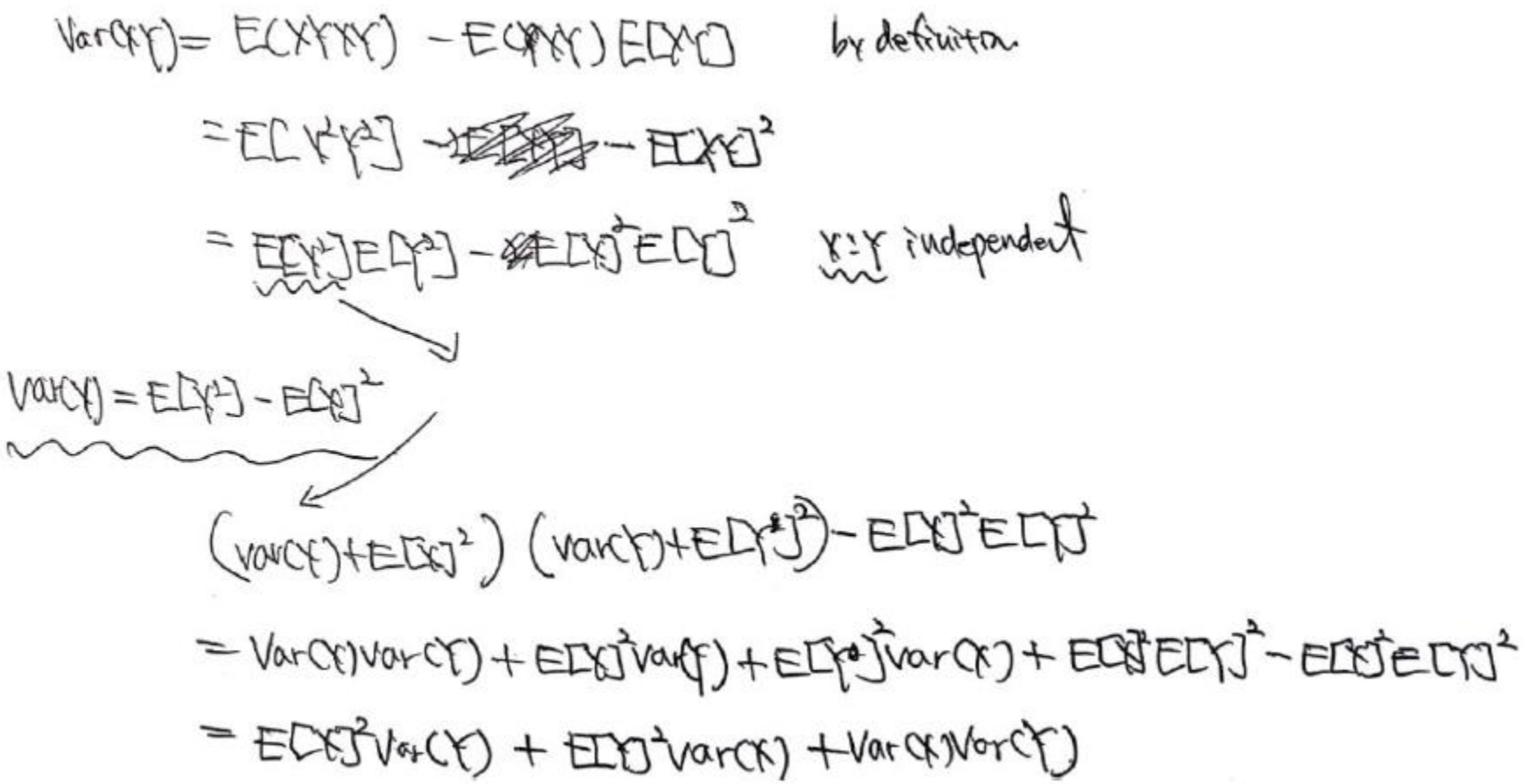
$$= (n\mathrm{Var}(w))\,\mathrm{Var}(x)$$

$z = \sum_j w_j x_j + b$



05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.



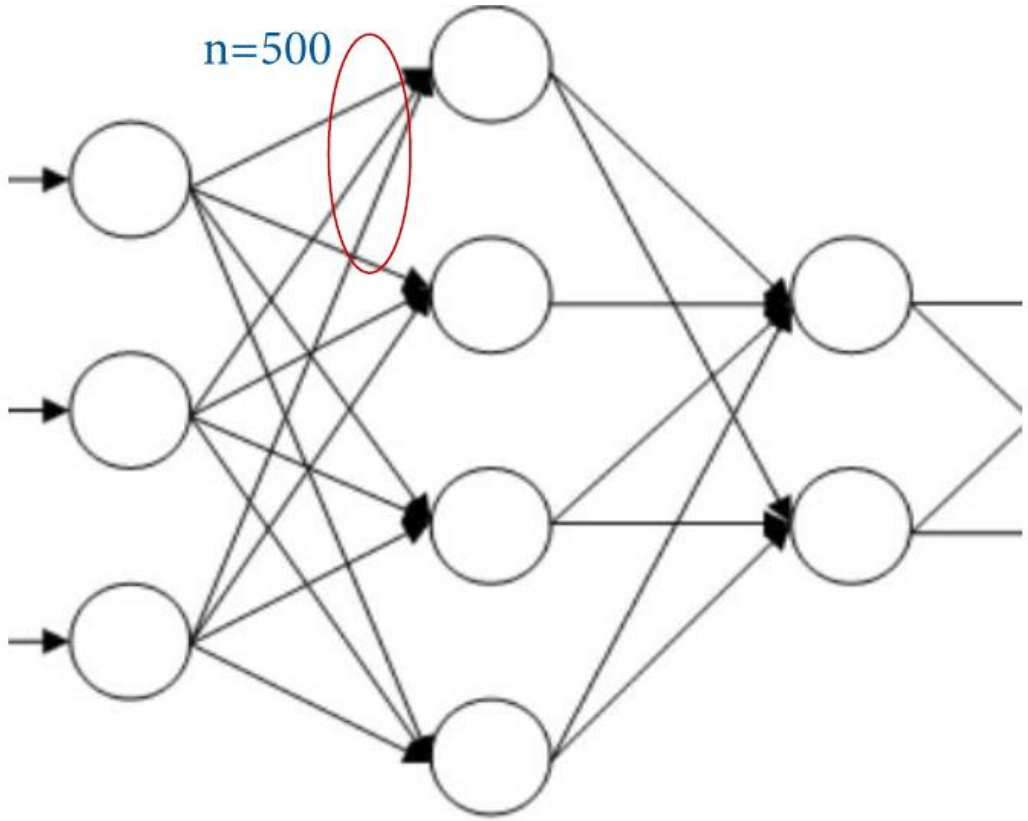
05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

$$Var(Z) = \left(nVar(W)\right)Var(X)$$

$$Var(W) = \frac{1}{\mathrm{n_{in}}} = \frac{1}{500}$$

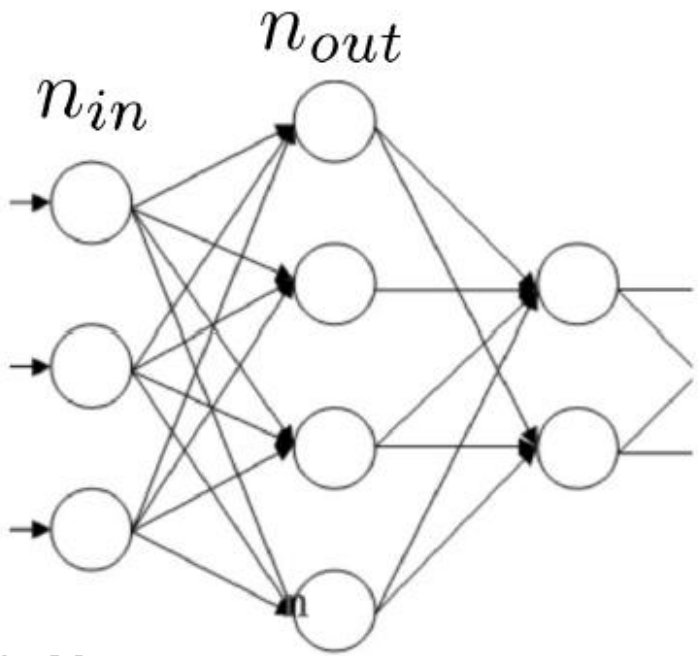
$$W \sim N(0, Var(W))$$



05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

- What this initialization method is considering
  - Not only the forward (as before)
    - for forward, there are *n_in* inputs for each neuron
  - But also backpropagation
    - for backward, there are *n_out* input for each neuron



$$Var(W) = \frac{2}{n_{in} + n_{out}}$$

Harmonic Mean

$$W \sim N(0, Var(W))$$

https://www.programmersought.com/article/73876303829/

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

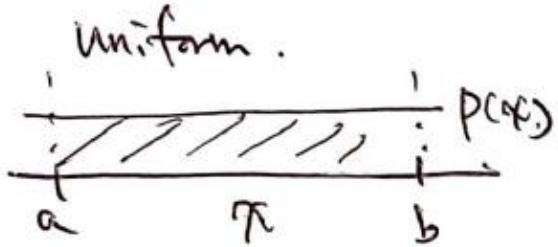
$$\mathrm{Var}(Y) = E[X^2] - E[X]E[X] = E[X^2] - \left(\frac{a+b}{2}\right)\left(\frac{a+b}{2}\right)$$

$$= \int_a^b p(x)x^2 - \left(\frac{a+b}{2}\right)^2 = \int_a^b \frac{1}{(b-a)}x^2 - \left(\frac{a+b}{2}\right)^2$$

$$= \frac{1}{(b-a)}\frac{1}{3}x^3\Big|_a^b - \left(\frac{a+b}{2}\right)^2 = \frac{1}{b-a}\frac{1}{3}(b^3-a^3) - \left(\frac{a+b}{2}\right)^2$$

$$= \frac{1}{3}\frac{1}{b-a}(b-a)(b^2+ab+a^2) - \left(\frac{a+b}{2}\right)^2 = \frac{1}{3}(a^2+ab+b^2) - \frac{a^2+2ab+b^2}{4}$$

$$= \frac{4a^2+4ab+4b^2-3a^2-6ab-3b^2}{12} = \frac{a^2-2ab+b^2}{12} = \boxed{\frac{(a-b)^2}{12}}$$



$p(x) = ?$

$\Downarrow$

$p(x)(b-a) = 1.$

$p(x) = \frac{1}{b-a}.$

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

$$n\,\mathrm{Var}(W) = 1$$

$$n\,\frac{(a-b)^2}{12} = 1.$$

$$b = -a \Rightarrow n\,\frac{(a-b)^2}{12} = n\,\frac{(2a)^2}{12} = n \cdot \frac{4a^2}{12} = \frac{na^2}{3}$$

$$\frac{na^2}{3} = 1 \qquad a^2 = \frac{3}{n} \qquad a = \pm\sqrt{\frac{3}{n}}$$

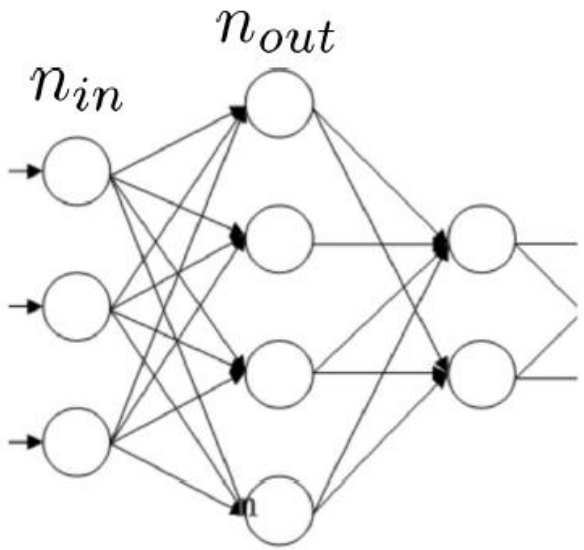
$\therefore$ Draw sample from $\left[-\sqrt{\frac{3}{n}}, \sqrt{\frac{3}{n}}\right]$ uniformly

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

– 1) draw sample from Uniform

$$\left[-\sqrt{\frac{6}{n_i + n_{i+1}}}, \sqrt{\frac{6}{n_i + n_{i+1}}}\right]$$



$Var(W) = \frac{1}{n}$ (Gaussian)

$Var(W) = \frac{3}{n}$ (Uniform)

thus when Gaussian → Uniform,
we can multiply by "3"

$$(Gaussain)Var(W) = \frac{2}{n_i + n_{i+1}} \Rightarrow (Uniform)Var(W) = \frac{2}{n_i + n_{i+1}} \times 3 = \frac{6}{n_i + n_{i+1}}$$

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

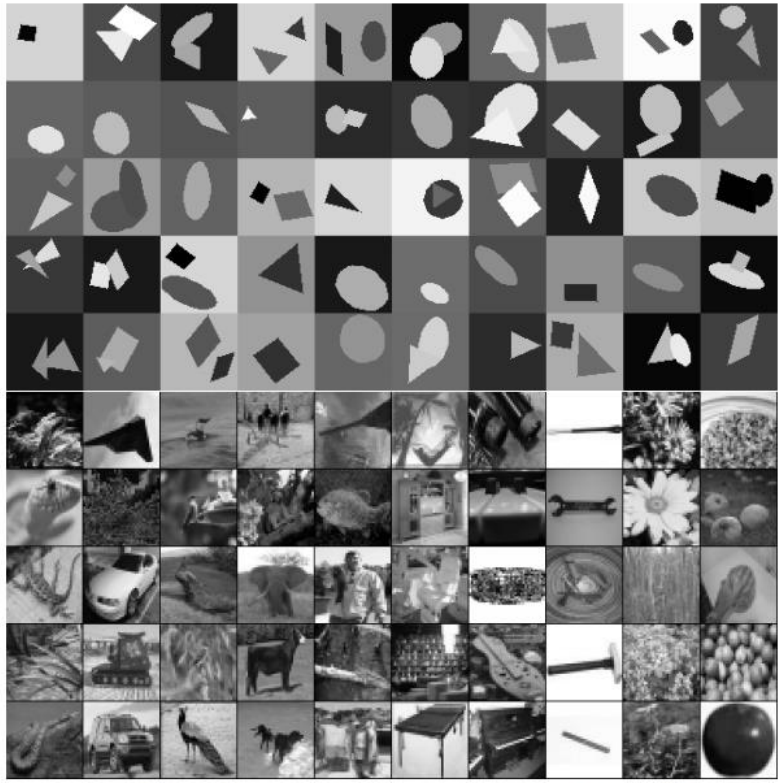


Figure 1: *Top: Shapeset-3×2 images at 64×64 resolution. The examples we used are at 32×32 resolution. The learner tries to predict which objects (parallelogram, triangle, or ellipse) are present, and 1 or 2 objects can be present, yielding 9 possible classifications. Bottom: Small-ImageNet images at full resolution.*

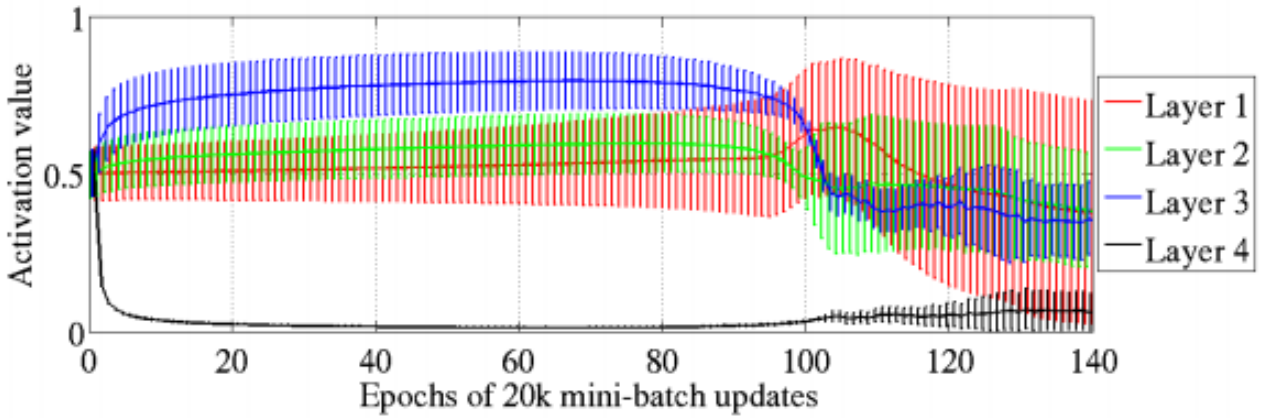


Figure 2: *Mean and standard deviation (vertical bars) of the activation values (output of the sigmoid) during supervised learning, for the different hidden layers of a deep architecture. The top hidden layer quickly saturates at 0 (slowing down all learning), but then slowly desaturates around epoch 100.*

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

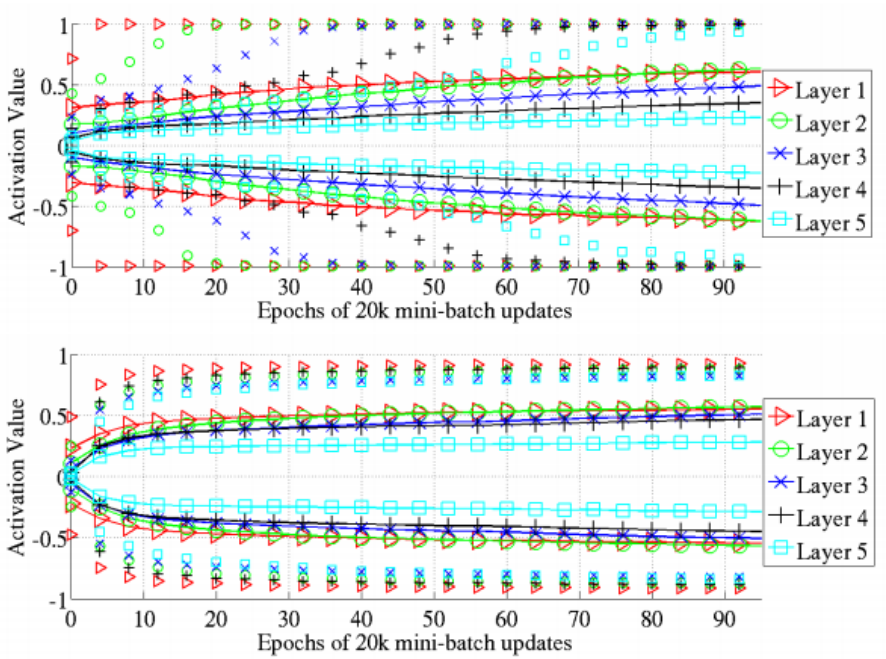


Figure 3: *Top:98 percentiles (markers alone) and standard deviation (solid lines with markers) of the distribution of the activation values for the hyperbolic tangent networks in the course of learning. We see the first hidden layer saturating first, then the second, etc. Bottom: 98 percentiles (markers alone) and standard deviation (solid lines with markers) of the distribution of activation values for the softsign during learning. Here the different layers saturate less and do so together.*

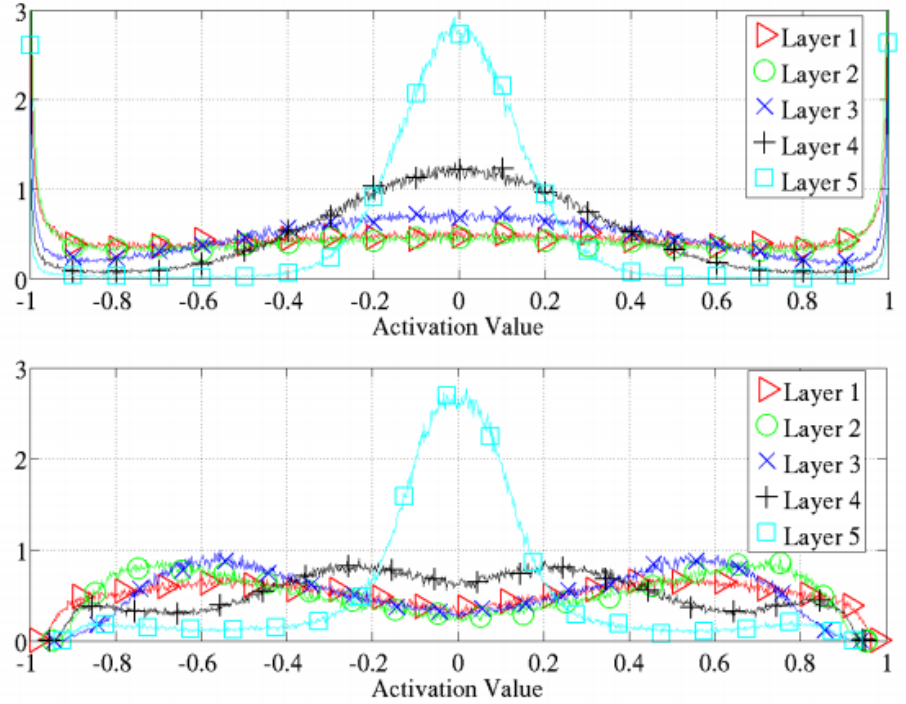


Figure 4: *Activation values normalized histogram at the end of learning, averaged across units of the same layer and across 300 test examples. Top: activation function is hyperbolic tangent, we see important saturation of the lower layers. Bottom: activation function is softsign, we see many activation values around (-0.6,-0.8) and (0.6,0.8) where the units do not saturate but are non-linear.*

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

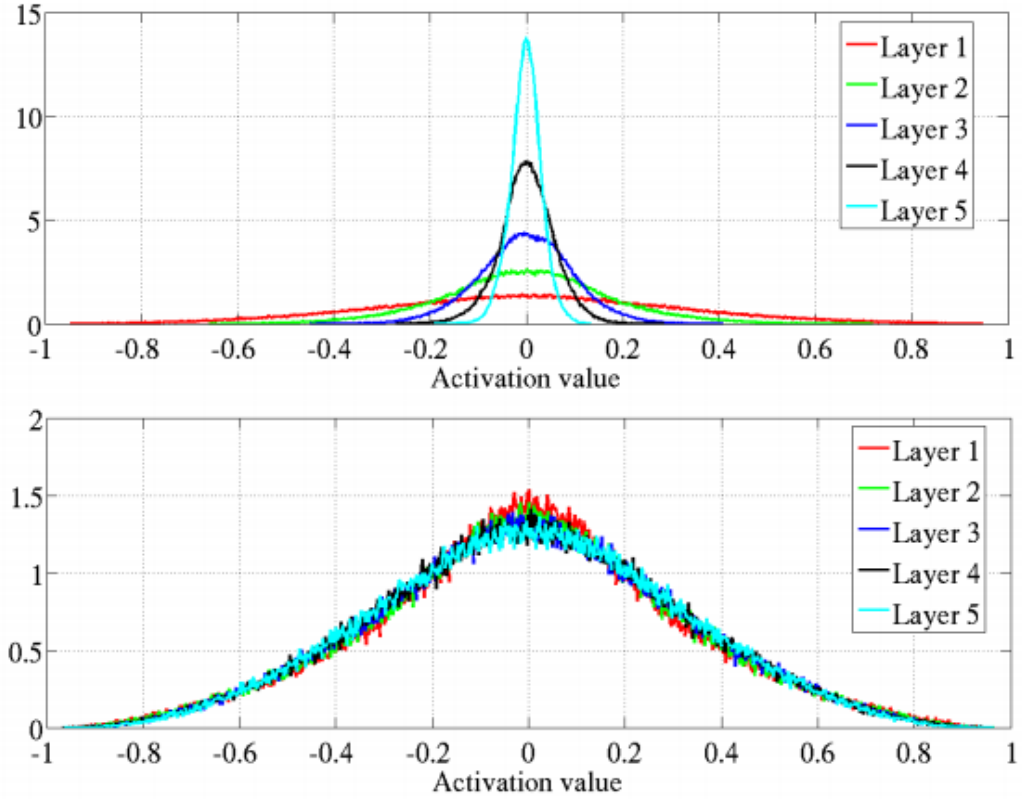


Figure 6: *Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.*

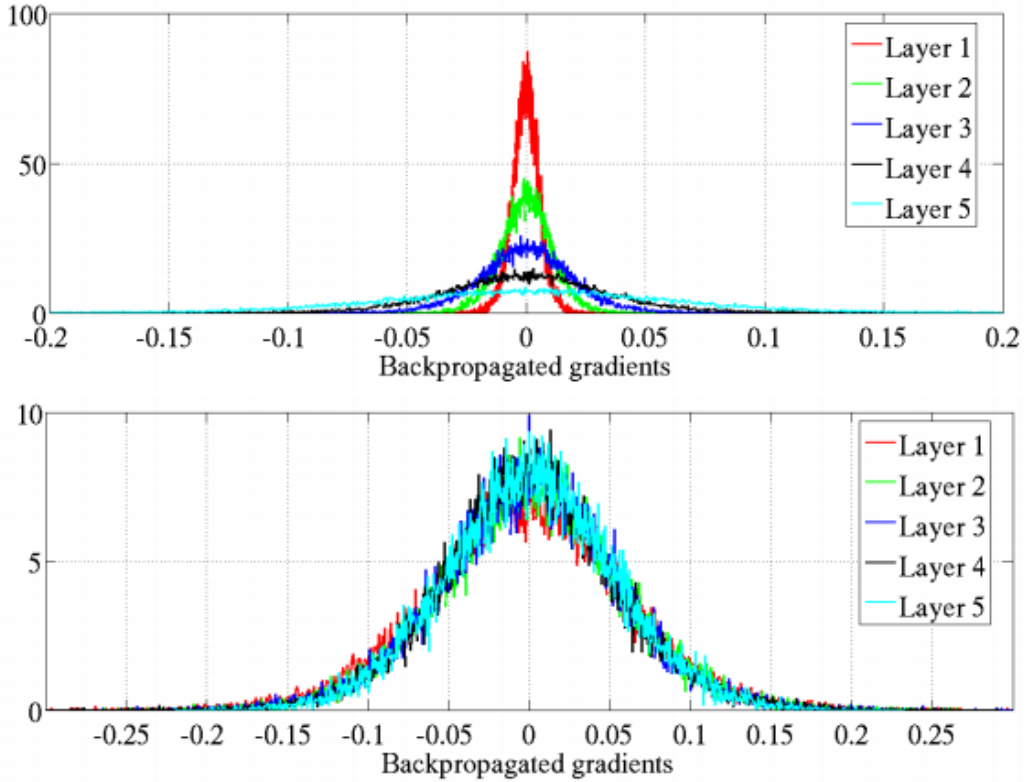


Figure 7: *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.*

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

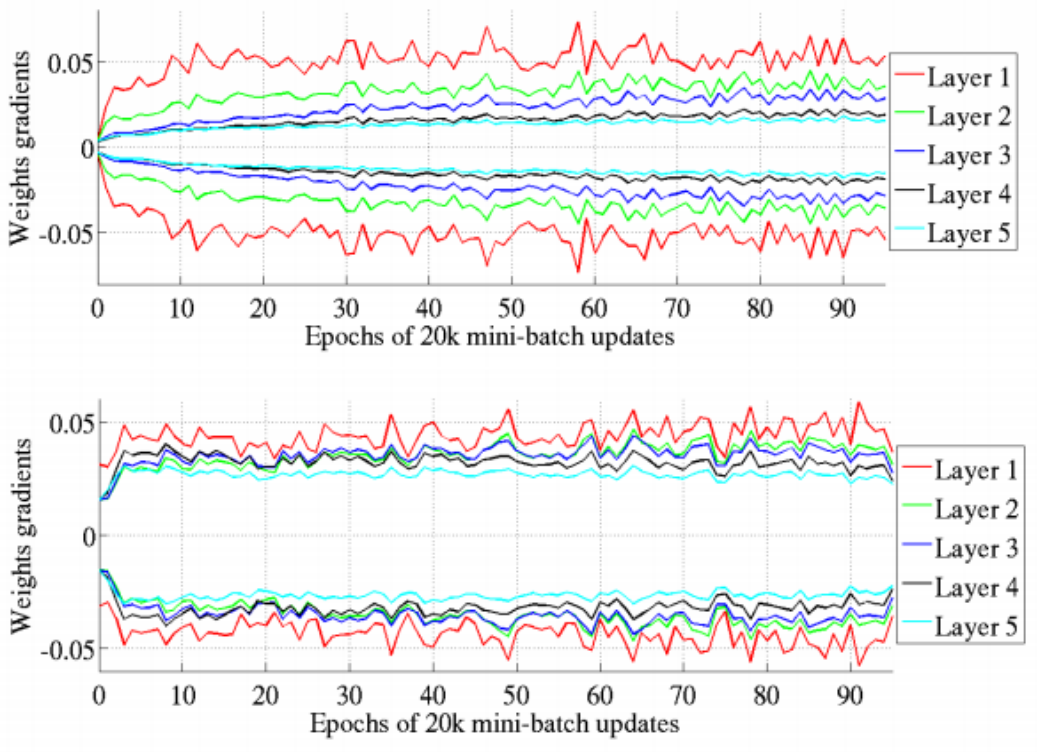


Figure 9: *Standard deviation intervals of the weights gradients with hyperbolic tangents with standard initialization (top) and normalized (bottom) during training. We see that the normalization allows to keep the same variance of the weights gradient across layers, during training (top: smaller variance for higher layers).*

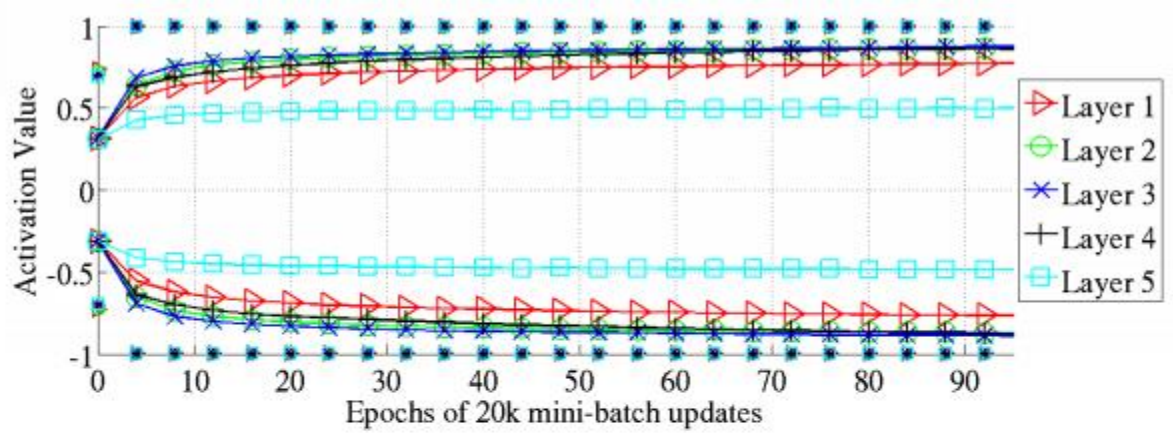


Figure 10: *98 percentile (markers alone) and standard deviation (solid lines with markers) of the distribution of activation values for hyperbolic tangent with normalized initialization during learning.*

activations (flowing upward) and gradients (flowing backward).

05 Xavier initialization for linear activation function

[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

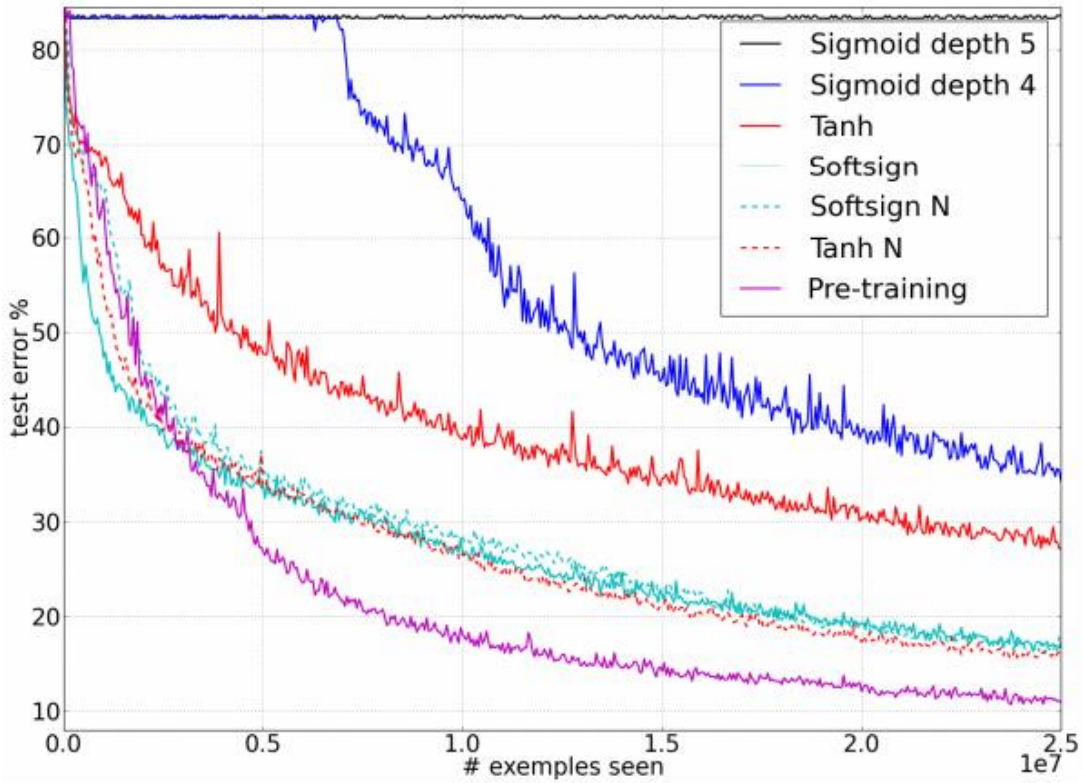


Figure 11: *Test error during online training on the* **Shapeset-**3×2 *dataset, for various activation functions and initialization schemes (ordered from top to bottom in decreasing final error). N after the activation function name indicates the use of normalized initialization.*

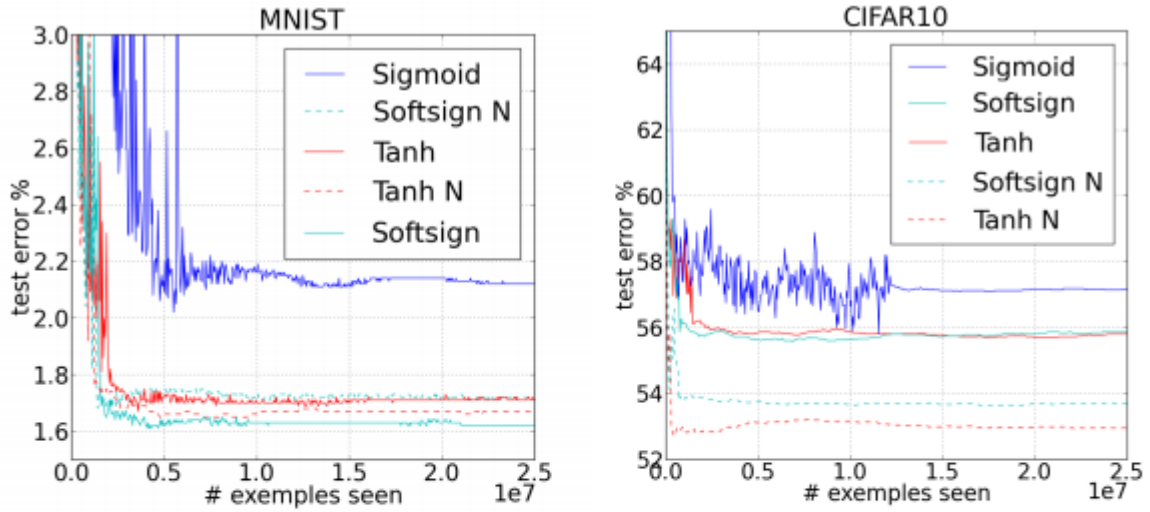


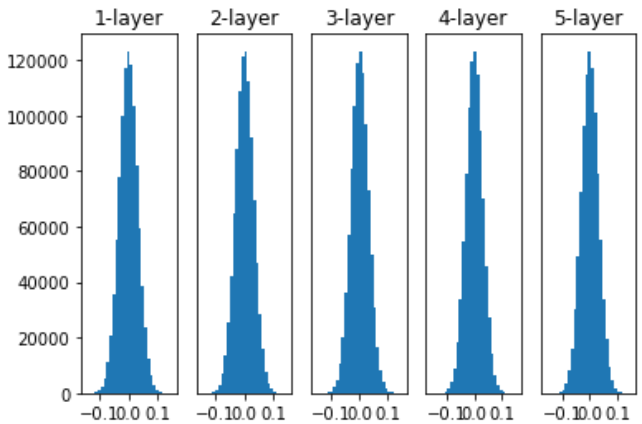
Figure 12: *Test error curves during training on MNIST and CIFAR10, for various activation functions and initialization schemes (ordered from top to bottom in decreasing final error). N after the activation function name indicates the use of normalized initialization.*

05 Xavier initialization for linear activation function

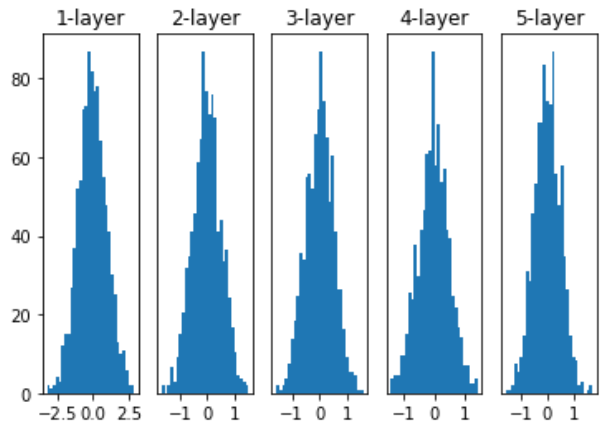
[Glorot2010] Understanding the difficulty of training deep feedforward neural networks.

$$W \sim N\left(0, Var(W)\right)$$
$$Var(W) = \frac{2}{n_{in}+n_{out}}$$

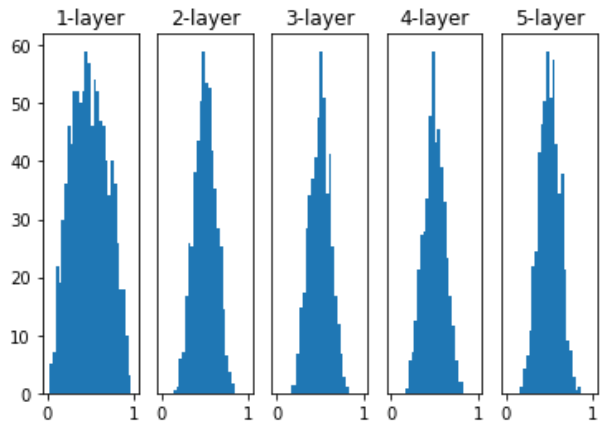
Layer # : 5 (1000 node in each layer)
X = np.random.randn(1, 1000)
W = np.random.randn(node_num, node_num) in each layer



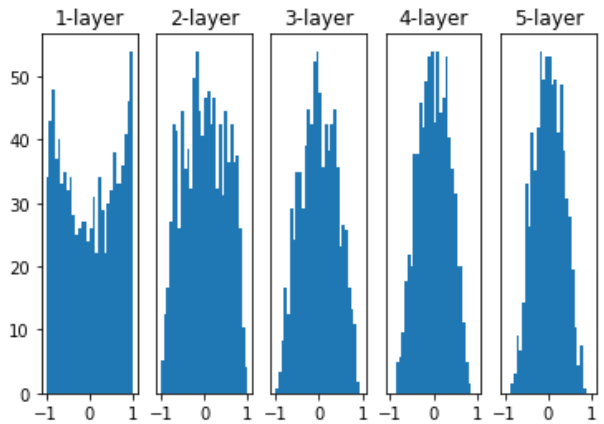
W



$WX = w_1x_1 + w_2x_2 + w_3x_3 + \ \ldots$



$Sigmoid(WX + b)$



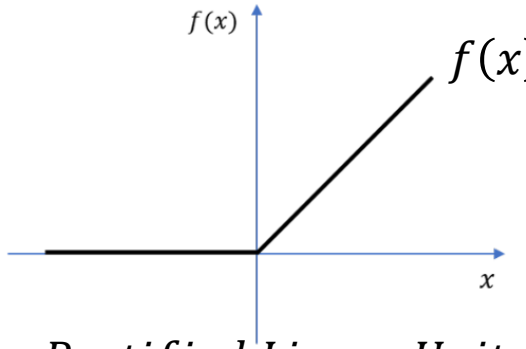
$\tanh(WX + b)$

06 He initialization for ReLU activation function

[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

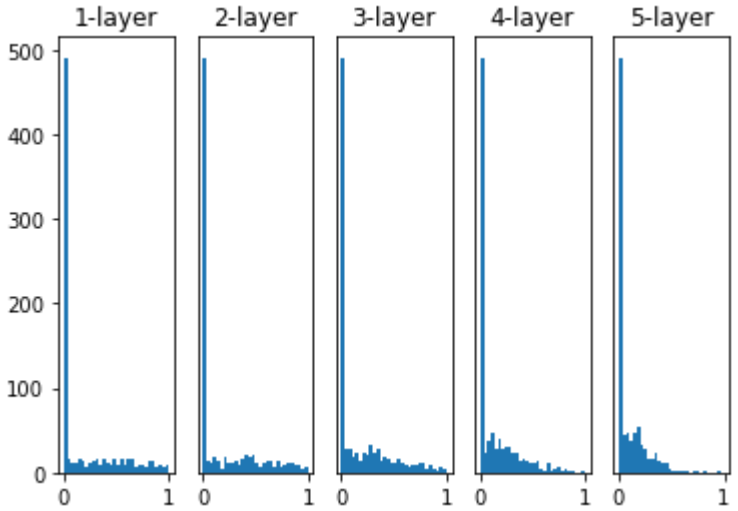
$$W \sim N\left(0, Var(W)\right)$$
$$Var(W) = \frac{2}{n_{in}+n_{out}}$$

Layer # : 5 (1000 node in each layer)
X = np.random.randn(1, 1000)
W = np.random.randn(node_num, node_num) in each layer

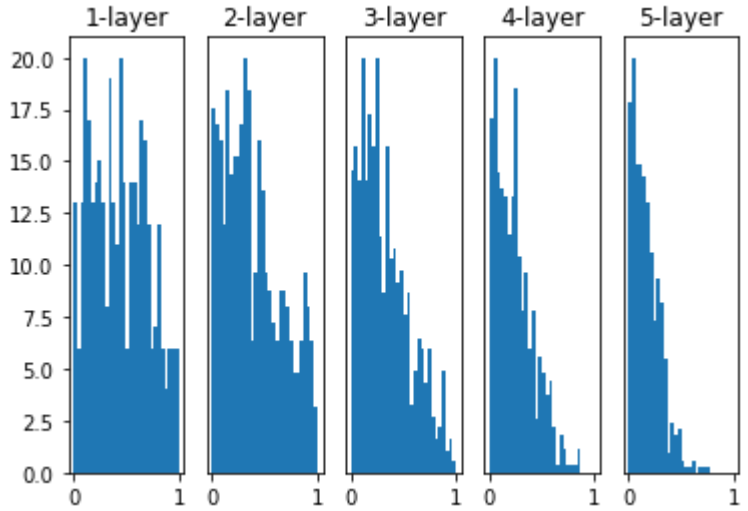


$f(x) = \max(0, x)$

Rectified Linear Unit



ReLU$(WX + b)$



ReLU$(WX + b)$

06 He initialization for ReLU activation function

[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

Forward Pass :

$$\mathbf{y}_l = \mathrm{W}_l \mathbf{x}_l + \mathbf{b}_l$$

$$\mathbf{x}_l = f(\mathbf{y}_{l-1})$$

Assume all x, w are independent

$$Y = W_1X_1 + W_2X_2 + \cdots + W_nX_n$$

$$Var[y_l] = n_l Var[w_l x_l]$$

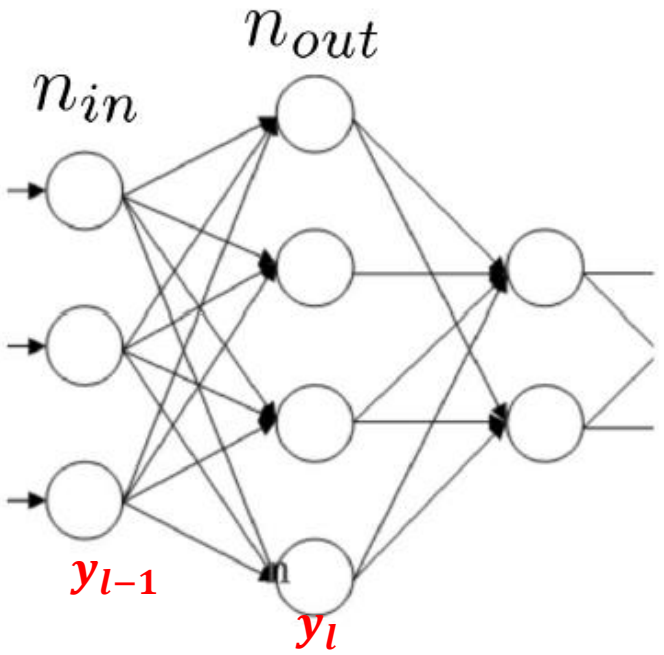
$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

$$Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}]$$

$$\frac{1}{2} n_l Var[w_l] = 1$$

$$Var(W) = \frac{2}{n_l}$$

$$W \sim N(0, Var(W))$$



$$Var[w_l x_l] = E[w_l^2]E[x_l^2] - [E[x_l]]^2[E[w_l]]^2 \quad E[w_l] = 0$$

$$Var[w_l x_l] = E[w_l^2]E[x_l^2]$$

$$Var[w_l] = E[w_l^2] - [E[w_l]]^2 \quad E[w_l] = 0$$

$$Var[w_l] = E[w_l^2]$$

$$Var[w_l x_l] = E[w_l^2]E[x_l^2]$$

$$Var[w_l x_l] = Var[w_l]E[x_l^2]$$

$$E(x_l^2) = \frac{1}{\sigma_{y_{l-1}\sqrt{2\pi}}} \int_0^\infty x^2 e^{-\frac{x^2}{2\sigma_{y_{l-1}}^2}} dx$$

$$= \left(\frac{1}{2}\right) \frac{1}{\sigma_{y_{l-1}\sqrt{2\pi}}} \int_{-\infty}^\infty x^2 e^{-\frac{x^2}{2\sigma_{y_{l-1}}^2}} dx$$

$$= \frac{\sigma_{y_{l-1}}^2}{2} = \frac{1}{2} Var(y_{l-1})$$

$$E[x^2] = \int_{-\infty}^{\infty} x^2 P(x) dx$$

$$= \int_{-\infty}^{\infty} max(0, y)^2 P(y) dy$$

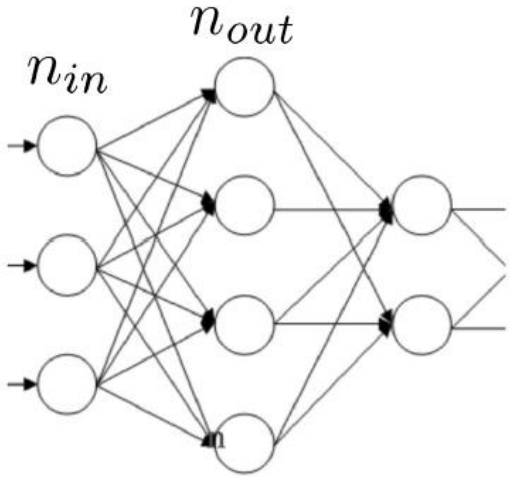
$$= \int_0^{\infty} y^2 P(y) dy$$

$$= 0.5 * \int_{-\infty}^{\infty} y^2 P(y) dy$$

$$= 0.5 * Var(y)$$

06 He initialization for ReLU activation function

[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.



Backward Pass :

$$\Delta \mathbf{x}_l = \hat{\mathbf{W}}_l \Delta \mathbf{y}_l$$
$$\Delta y_l = f'(y_l)\Delta x_{l+1}$$

$w_l, \Delta y_l$: independent
Δx_l : zero mean for all l

$$Var[\Delta x_l] = \hat{n}_l Var[w_l] Var[\Delta y_l]$$
$$= \frac{1}{2}\hat{n}_l Var[w_l] Var[\Delta x_{l+1}]$$

ReLU: 1/2

$$E[\Delta y_l] = E[f'(y_l)\Delta x_{l+1}] = E[f'(y_l)]E[\Delta x_{l+1}]$$
$$\doteq \frac{1}{2}E[\Delta x_{l+1}] = 0$$

$$Var(w_l \cdot \Delta y_l) = [E(w_l)]^2 Var(\Delta y_l) + [E(\Delta y_l)]^2 Var(w_l) + Var(w_l)Var(\Delta y_l)$$

$$Var[\Delta y_l] = Var[f'(y_l)\Delta x_{l+1}]$$
$$= E[(f'(y_l))^2]E[(\Delta x_{l+1})^2] - [E[f'(y_l)]]^2[E[\Delta x_{l+1}]]^2$$
$$= E[(f'(y_l))^2]E[(\Delta x_{l+1})^2] - 0$$
$$= E[(f'(y_l))^2]E[(\Delta x_{l+1})^2]$$
$$= \frac{1}{2}E[(\Delta x_{l+1})^2]$$

$$\frac{1}{2}n_l Var[w_l] = 1$$
$$Var(W) = \frac{2}{n_l}$$
$$W \sim N(0, Var(W))$$

06 He initialization for ReLU activation function

[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

Only Fan in is enough

$$Var(W) = \frac{2}{n_{in+n_{out}}}$$

$$W \sim N(0, Var(W))$$

Xavier Normal

$$Var(W) = \frac{2}{n_{in}}$$

$$W \sim N(0, Var(W))$$

He Normal

$$Var(W) = \frac{6}{n_{in+n_{out}}}$$

$$W \sim U(0, Var(W))$$

Xavier Uniform

Xavier initialization

$$Var(W) = \frac{6}{n_{in}}$$

$$W \sim U(0, Var(W))$$

He Uniform

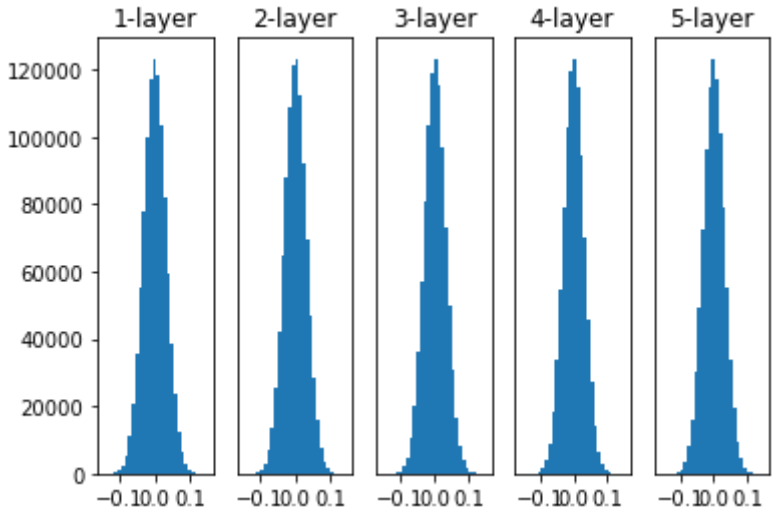
He initialization

06 He initialization for ReLU activation function

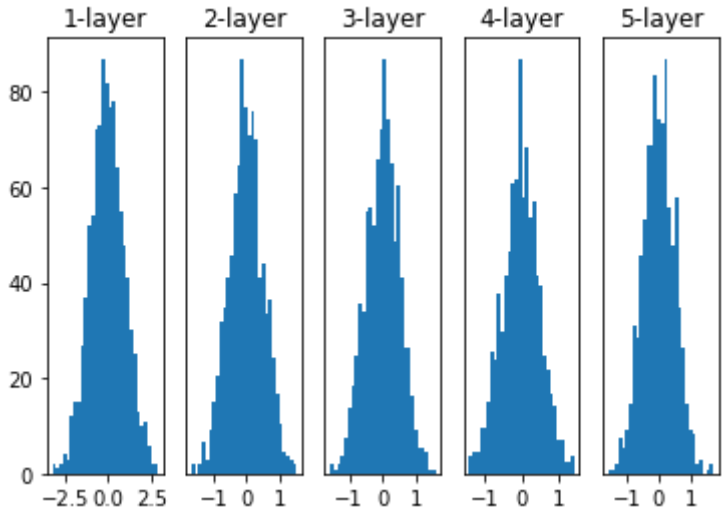
[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

$$W \sim N\left(0, Var(W)\right)$$
$$Var(W) = \frac{2}{n_{in}}$$

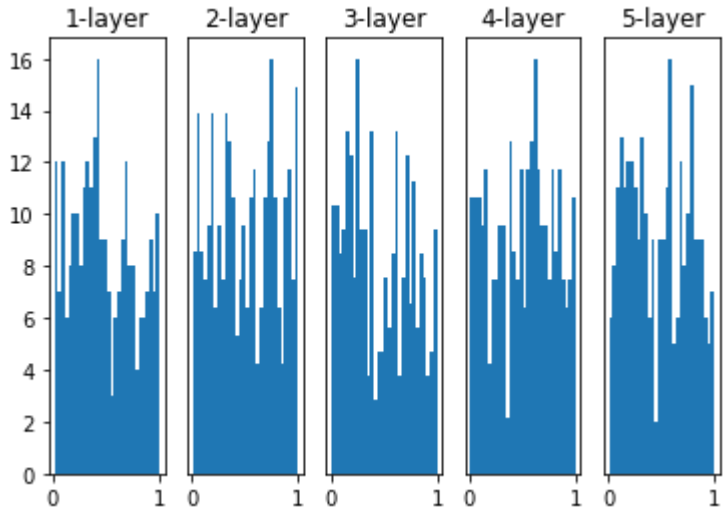
Layer # : 5 (1000 node in each layer)
X = np.random.randn(1, 1000)
W = np.random.randn(node_num, node_num) in each layer



W



$WX = w_1x_1 + w_2x_2 + w_3x_3 + \ \ldots$



$ReLU(WX)$

06 He initialization for ReLU activation function

[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

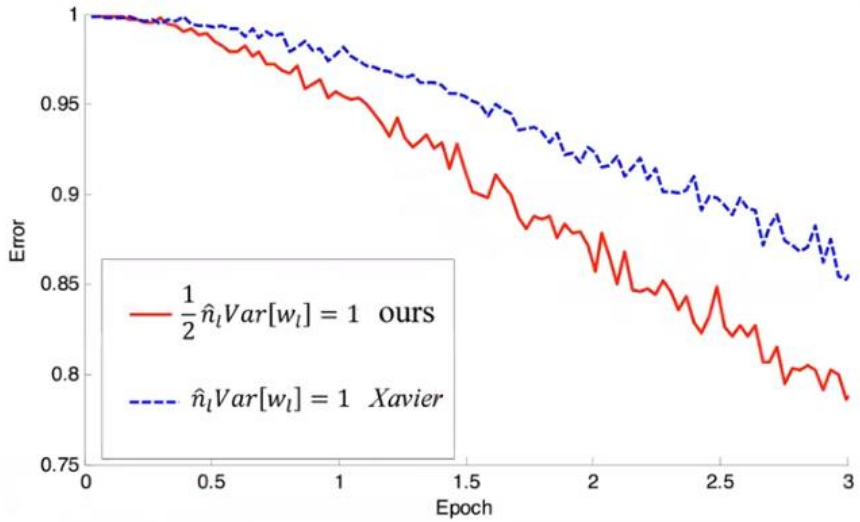


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and "*Xavier*" (blue) [7] lead to convergence, but ours starts reducing error earlier.

19 conv + 3 fc

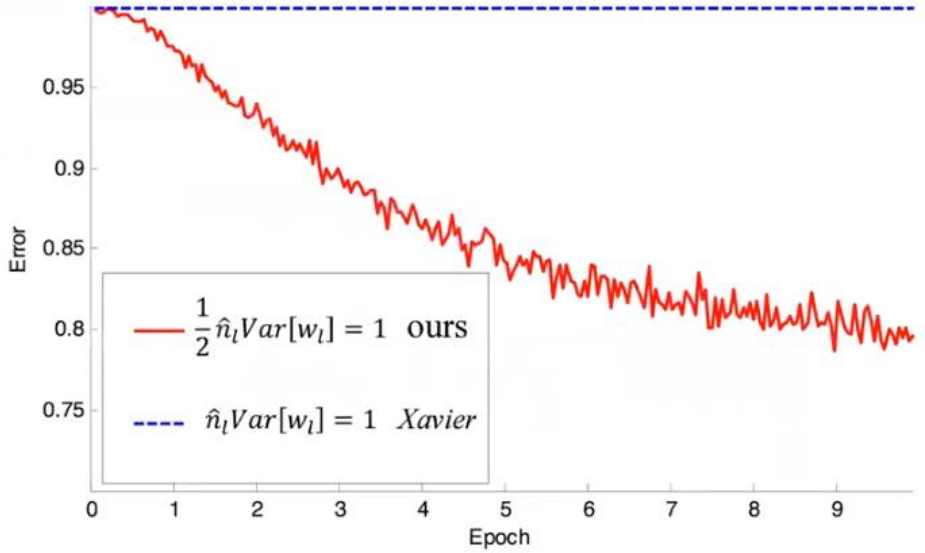


Figure 3. The convergence of a **30-layer** small model (see the main text). We use ReLU as the activation for both cases. Our initialization (red) is able to make it converge. But "*Xavier*" (blue) [7] completely stalls - we also verify that its gradients are all diminishing. It does not converge even given more epochs.

27 conv + 3 fc

06 He initialization for ReLU activation function

[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

- per layer
- channel-shared
- channel-wise

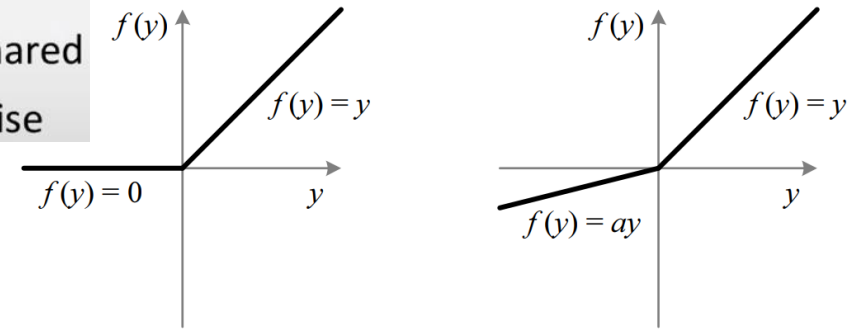


Figure 1. ReLU vs. PReLU. For PReLU, the coefficient of the negative part is not constant and is adaptively learned.

$$f(x_i) = \begin{cases} x_i, & \text{if } x_i > 0 \\ a_i x_i, & \text{if } x_i \leq 0 \end{cases}$$

- ReLU: when $a_i = 0$

$$f(x_i) = max(0, x_i)$$

- PReLU: when a_i is a *learnable parameter*

$$f(x_i) = max(0, x_i) + a_i min(0, x_i)$$

- LReLU: Leaky ReLU, when $a_i = 0.01$

$$f(x_i) = max(0, x_i) + 0.01 min(0, x_i)$$

Initialization in PReLU Case

- Forward Propagation Case: $\frac{1}{2}(1 + a^2) n_l Var[w_l] = 1$
- Backward Propagation Case: $\frac{1}{2}(1 + a^2) \hat{n}_l Var[w_l] = 1$

06 He initialization for ReLU activation function

[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

Architectures

Models

- A: VGG-19 modified for speed
- B: deeper (+3 conv) than A
- C: wider (more filters) than B

input size	VGG-19 [25]	model A	model B	model C
224	3×3, 64 3×3, 64 2×2 maxpool, /2	7×7, 96, /2	7×7, 96, /2	7×7, 96, /2
112	3×3, 128 3×3, 128 2×2 maxpool, /2	2×2 maxpool, /2	2×2 maxpool, /2	2×2 maxpool, /2
56	3×3, 256 3×3, 256 3×3, 256 3×3, 256 2×2 maxpool, /2	3×3, 256 3×3, 256 3×3, 256 3×3, 256 3×3, 256 2×2 maxpool, /2	3×3, 256 3×3, 256 3×3, 256 3×3, 256 3×3, 256 3×3, 256 2×2 maxpool, /2	3×3, 384 3×3, 384 3×3, 384 3×3, 384 3×3, 384 3×3, 384 2×2 maxpool, /2
28	3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 768 3×3, 768 3×3, 768 3×3, 768 3×3, 768 3×3, 768 2×2 maxpool, /2
14	3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 spp, {7, 3, 2, 1}	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 spp, {7, 3, 2, 1}	3×3, 896 3×3, 896 3×3, 896 3×3, 896 3×3, 896 3×3, 896 spp, {7, 3, 2, 1}
fc_1 fc_2 fc_3		4096 4096 1000		
depth (conv+fc)	19	19	22	22
complexity (ops., $\times 10^{10}$)	1.96	1.90	2.32	5.30

Table 3. Architectures of large models. Here "/2" denotes a stride of 2.

06 He initialization for ReLU activation function

[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

Experiments on ImageNet

Performed experiments on the 1000-class ImageNet 2012 dataset.

- 1.2M training images, 40K validation images, 100K test images
- Results measured by top-1/top-5 error rates
- Top-5 error rate is the metric officially used to rank the methods in the classification challenge.

06 He initialization for ReLU activation function

[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

Comparisons between ReLU and PReLU

PReLU reduces top-1 error by 1.05% and top-5 error by 0.23%.

- Large model A
- Channel-wise PReLU
- PReLU improves both small and large models consistently

model A	ReLU		PReLU	
scale s	top-1	top-5	top-1	top-5
256	26.25	8.25	**25.81**	**8.08**
384	24.77	7.26	**24.20**	**7.03**
480	25.46	7.63	**24.83**	**7.39**
multi-scale	24.02	6.51	**22.97**	**6.28**

Table 4. Comparisons between ReLU/PReLU on model A in ImageNet 2012 using dense testing.

06 He initialization for ReLU activation function

[He2015] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

	team	top-1	top-5
in competition ILSVRC 14	MSRA [11]	27.86	9.08†
	VGG [25]	-	8.43†
	GoogLeNet [29]	-	7.89
post-competition	VGG [25] (arXiv v2)	24.8	7.5
	VGG [25] (arXiv v5)	24.4	7.1
	Baidu [32]	24.88	7.42
	MSRA (A, ReLU)	24.02	6.51
	MSRA (A, PReLU)	22.97	6.28
	MSRA (B, PReLU)	22.85	6.27
	MSRA (C, PReLU)	**21.59**	**5.71**

Table 6. The **single-model** results for ImageNet 2012 val set. †: Evaluated from the test set.

	team	top-5 (**test**)
in competition ILSVRC 14	MSRA, SPP-nets [11]	8.06
	VGG [25]	7.32
	GoogLeNet [29]	6.66
post-competition	VGG [25] (arXiv v5)	6.8
	Baidu [32]	5.98
	MSRA, PReLU-nets	**4.94**

Table 7. The **multi-model** results for the ImageNet 2012 test set.

O. Russakovsky et al., "Imagenet large scale visual recognition challenge", 2014.

Human performance yields **5.1%** top-5 error.

"Our result is the first published instance of surpassing humans on this visual recognition challenge."

07 Conclusion

Conclusion

- ✓ It is very important to initialize weights properly in deep network.
- ✓ Don't initialize all weights to zero or constant value.
- ✓ Don't initialize all weights randomly to small value or large value.
- ✓ Xavier initialization is good for linear activation function (Sigmoid, TanH, etc.).
- ✓ He initialization is good for ReLU.
- ✓ [Kumar2017] On weight initialization in deep neural networks.

Thank you !

E-Mail : yangdonghun3@gmail.com
Phone : +82-10-2484-5894