Lecutre 3: Neural Networks

Today:

00:00) Revisit feature transform

00:10) What is a neural net?

00:20) Chain-rule reminder

00:30) Vanilla back-propagation

00:45) ---- Break ----

01:00) Vanilla back-propagation (continued)

02:30) Differential computational graph

01:45) ---- Break ----

02:00) Differential computational graph (continued)

02:10) Can anything be back-propagated?

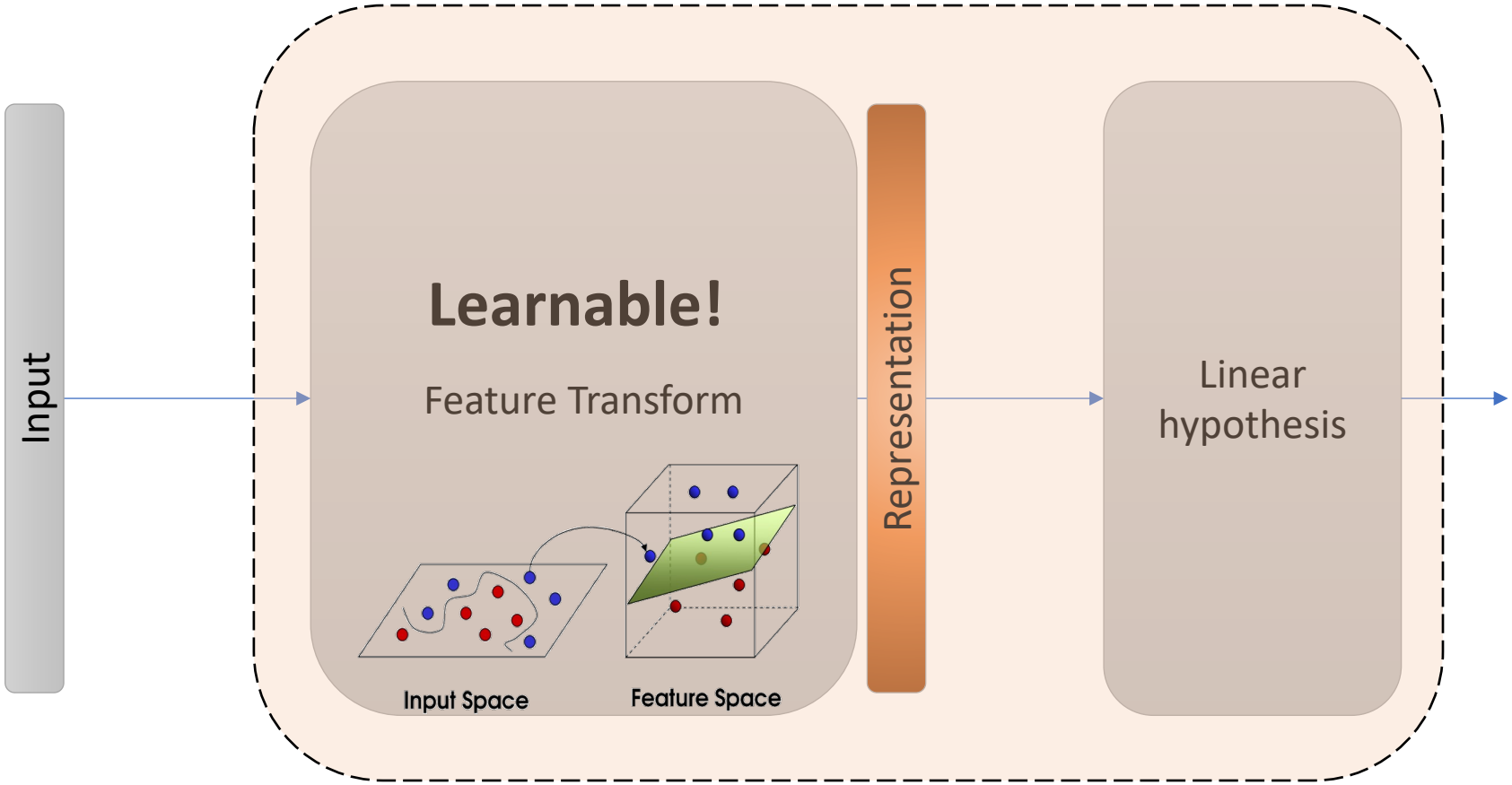
02:20) Why are neural networks better than others?

02:35) Demo

02:45) ---- Break ----

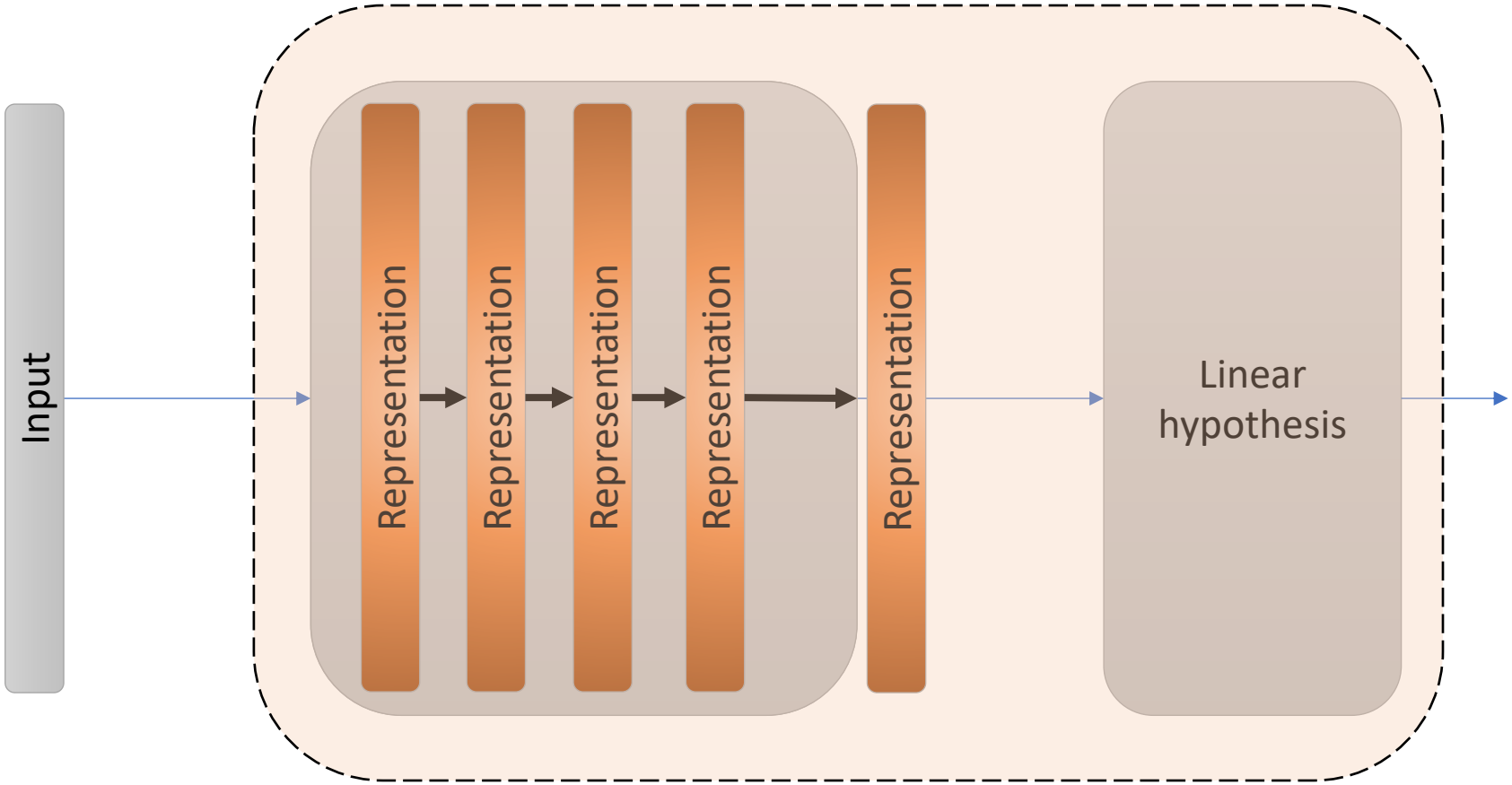
03:00) Tutorial

Feature transform

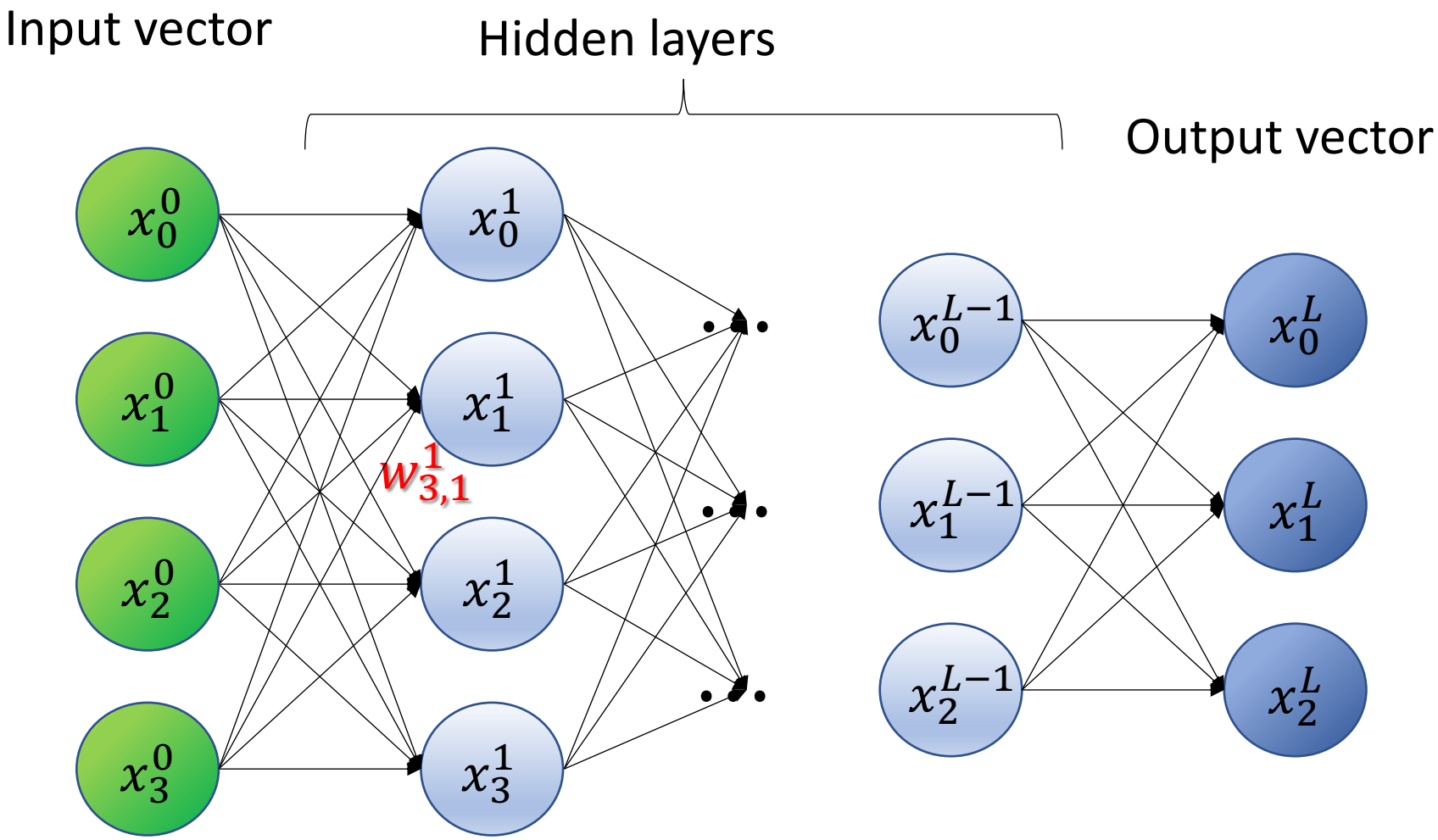


Non-linear hypothesis!

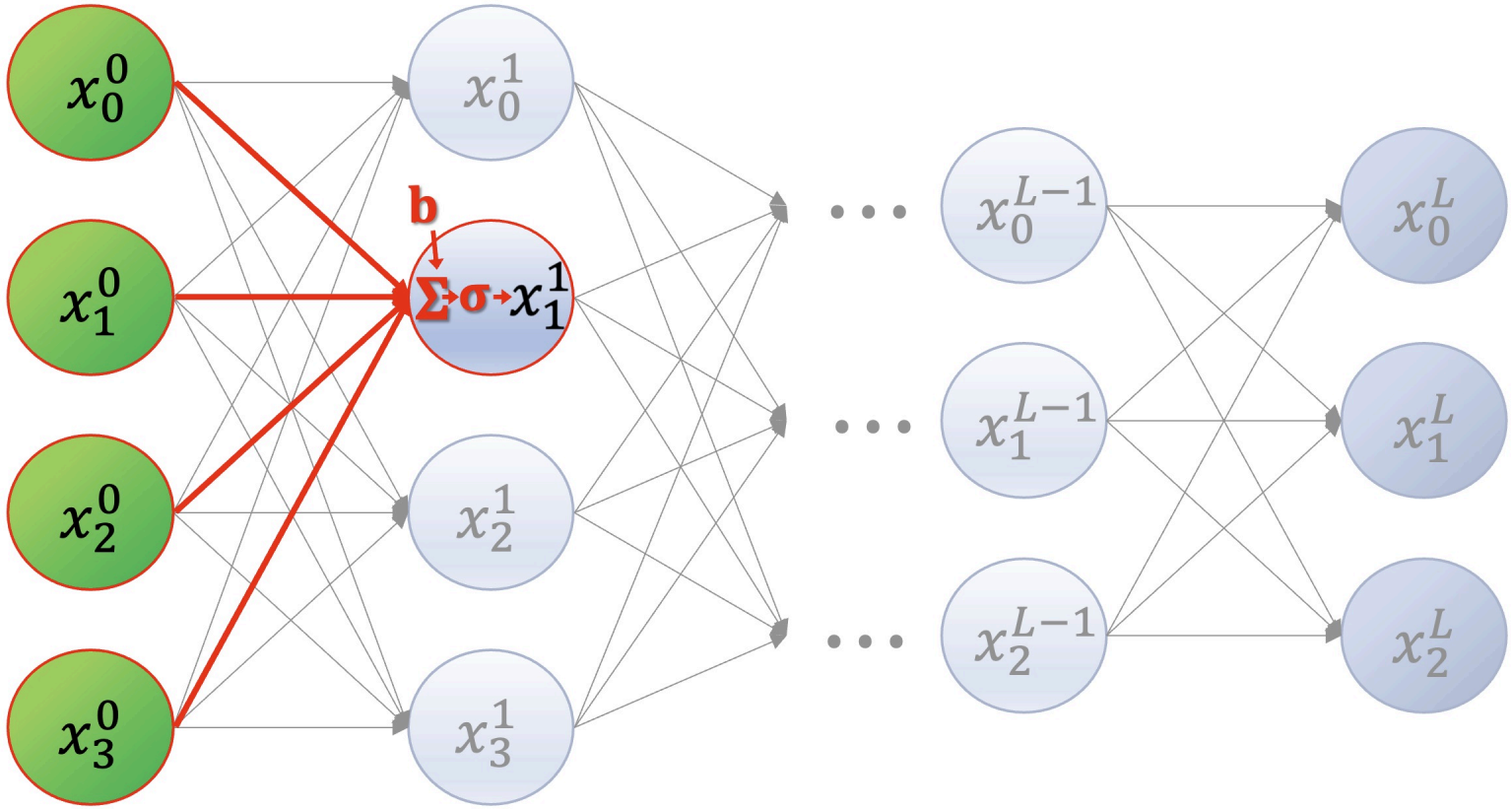
Feature transform



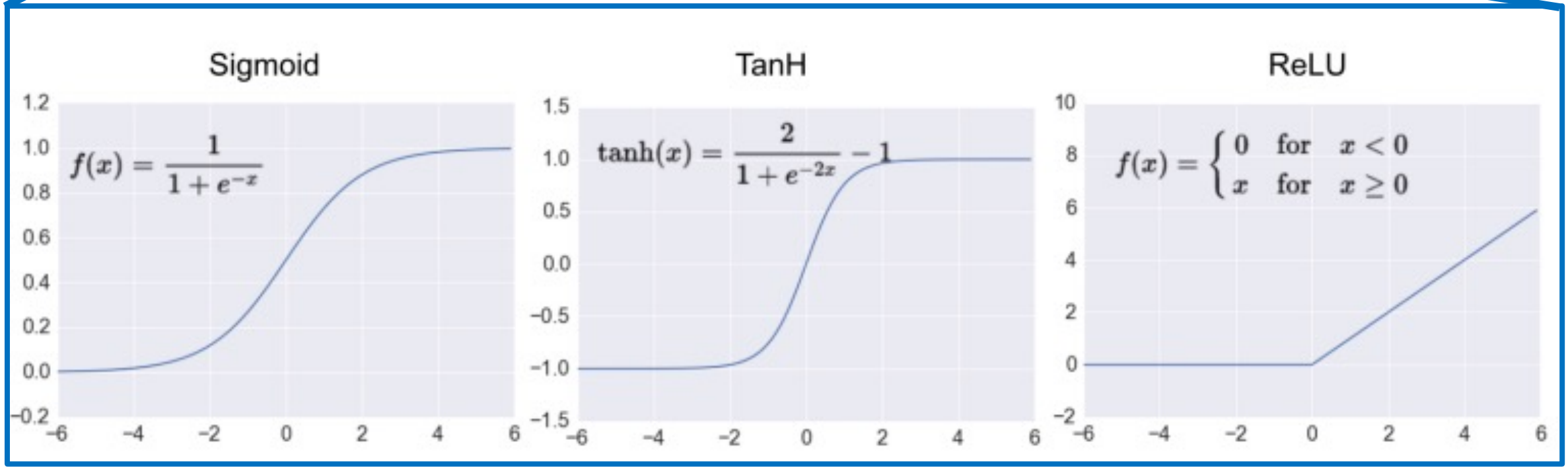
Non-linear hypothesis!


Input vector
Hidden layers
Output vector
x_0^0
x_1^0
x_2^0
x_3^0
x_0^1
x_1^1
x_2^1
x_3^1
$w_{3,1}^1$
x_0^{L-1}
x_1^{L-1}
x_2^{L-1}
x_0^L
x_1^L
x_2^L

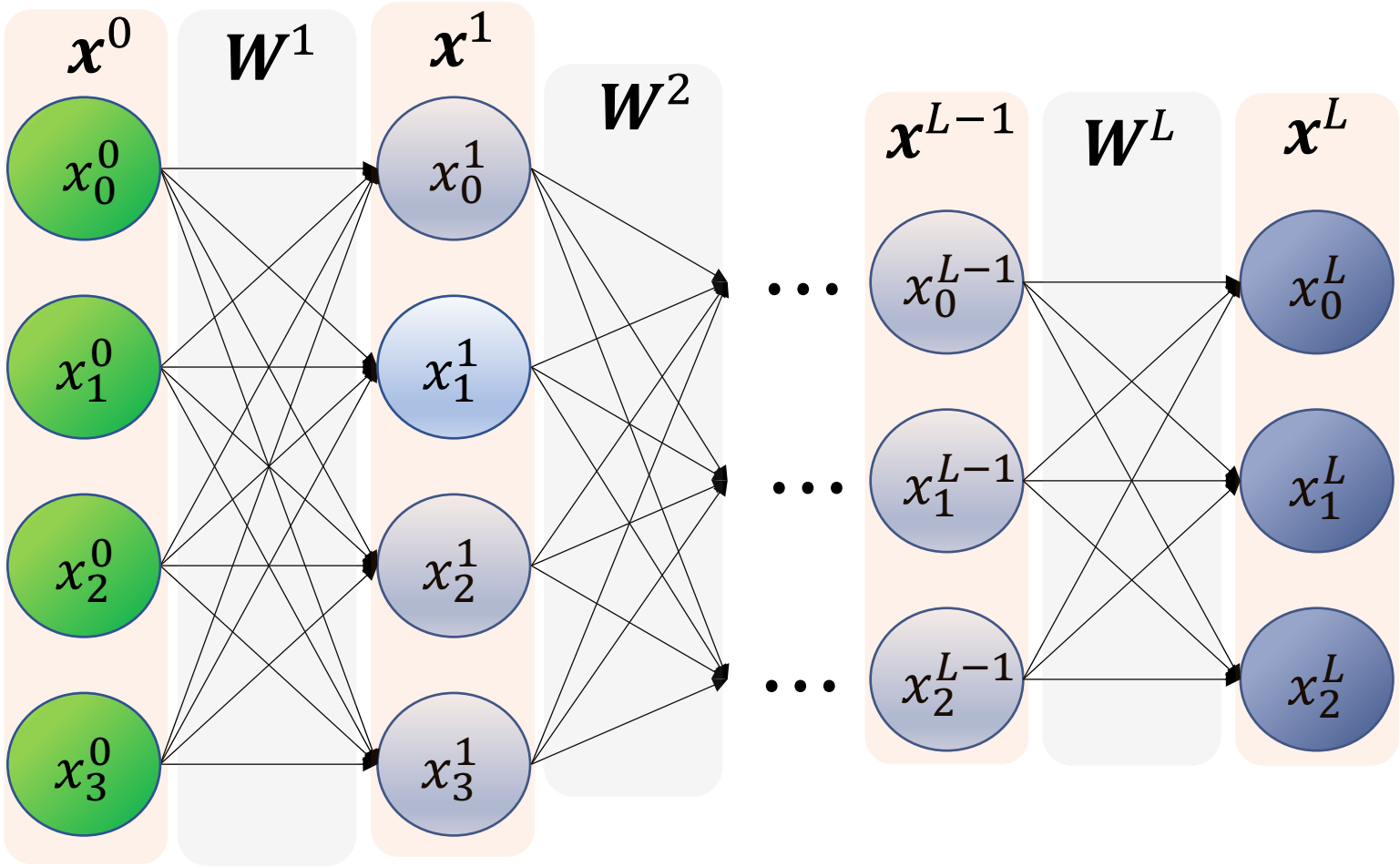
$$x_j^l = \sigma\left(\sum_i w_{ij}^l \cdot x_i^{l-1} + b_j^l\right)$$



$$x_j^l = \sigma\left(\sum_i w_{ij}^l \cdot x_i^{l-1} + b_j^l\right)$$

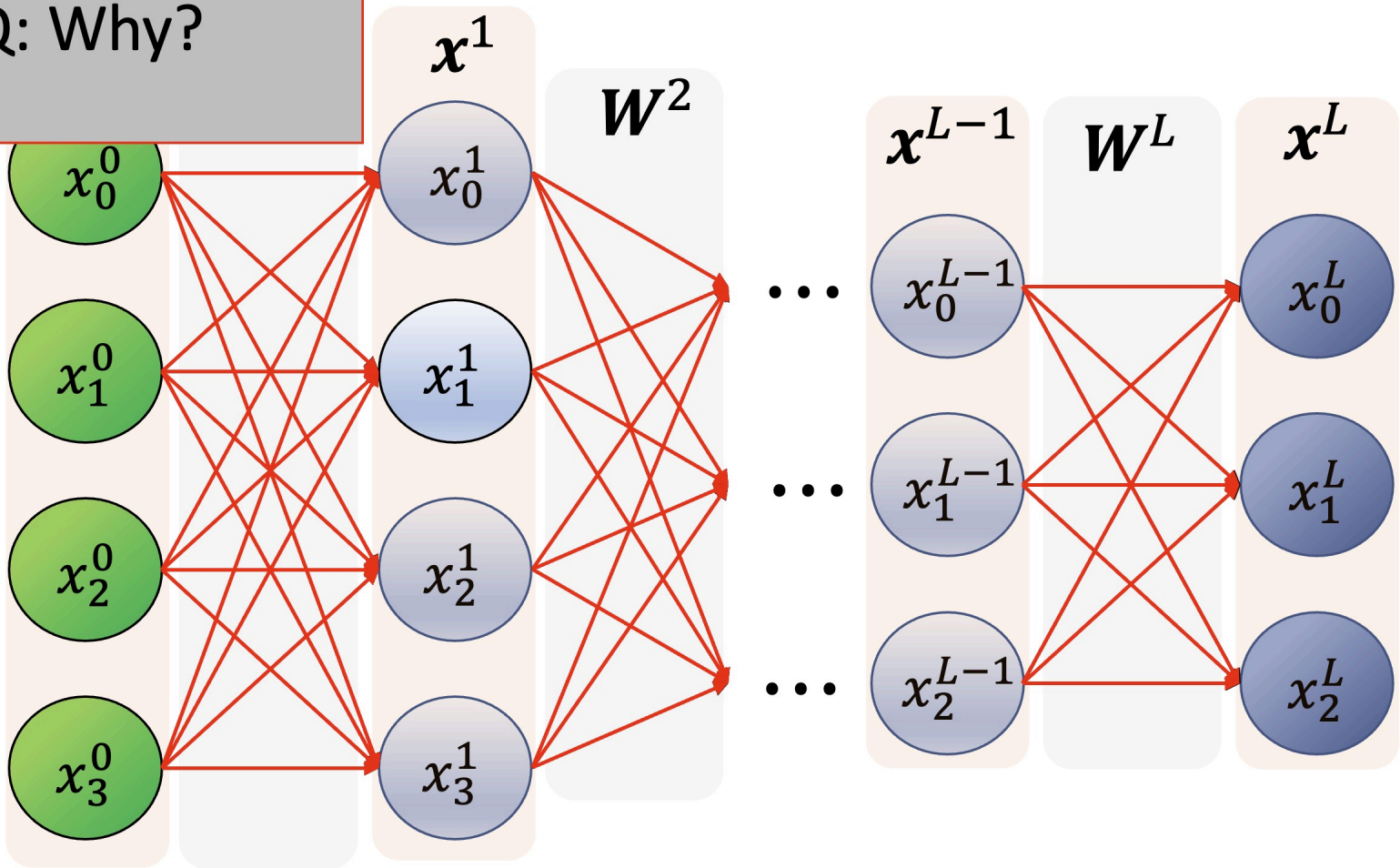


$$x_j^l = \sigma\left(\sum_i w_{ij}^l \cdot x_i^{l-1} + b_j^l\right) \qquad \boldsymbol{x}^l = \boldsymbol{\sigma}\left(\boldsymbol{W}^{l^T}\boldsymbol{x}^{l-1} + \boldsymbol{b}^l\right)$$

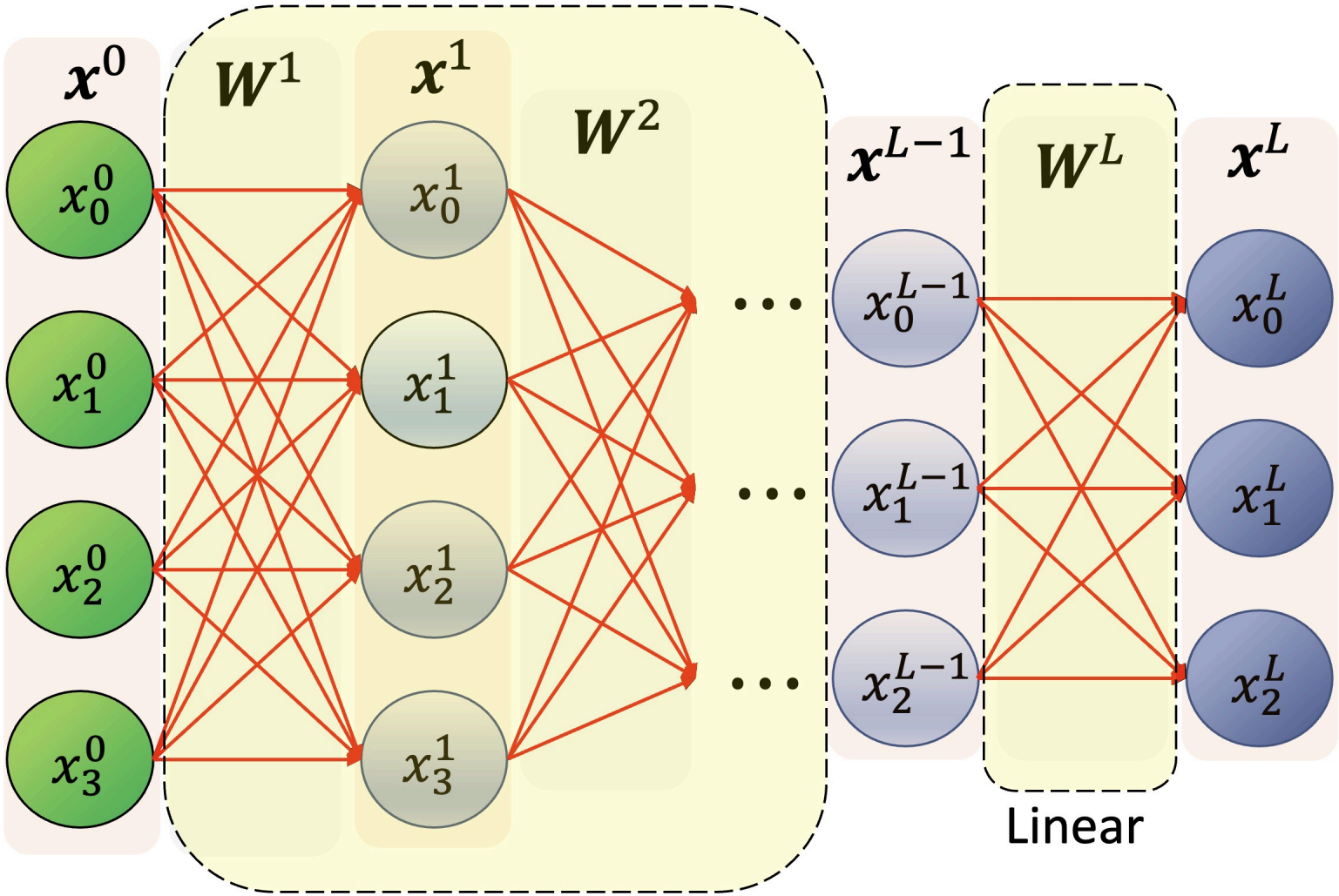


$$x_j^l = \sigma\left(\sum w_{ij}^l \cdot x_i^{l-1} + b_j^l\right) \qquad \boldsymbol{x}^l = \boldsymbol{\sigma}\left(\boldsymbol{W}^{l^T}\boldsymbol{x}^{l-1} + \boldsymbol{b}^l\right)$$

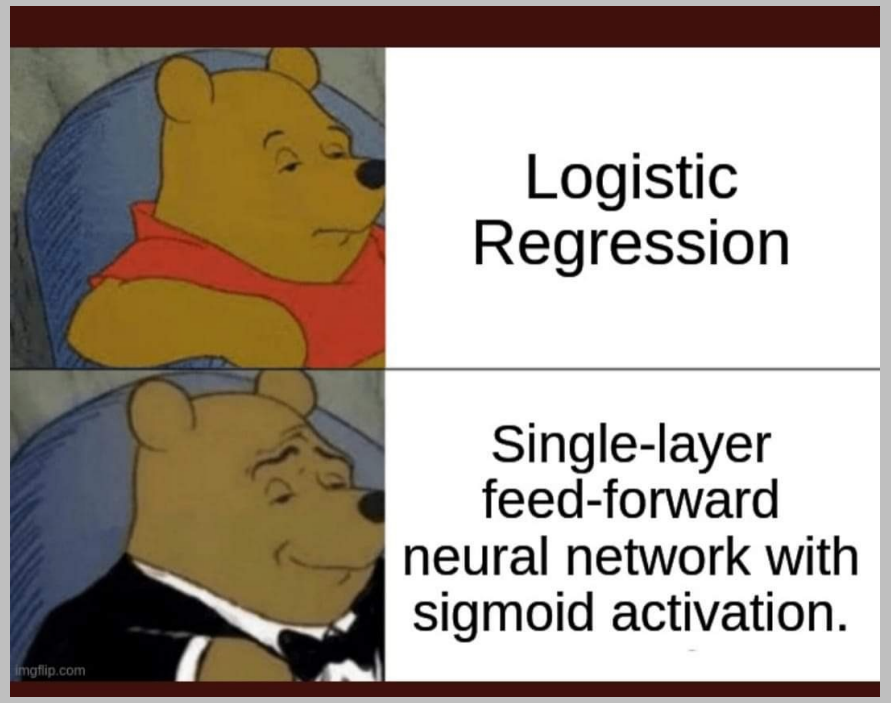
Q: Why?



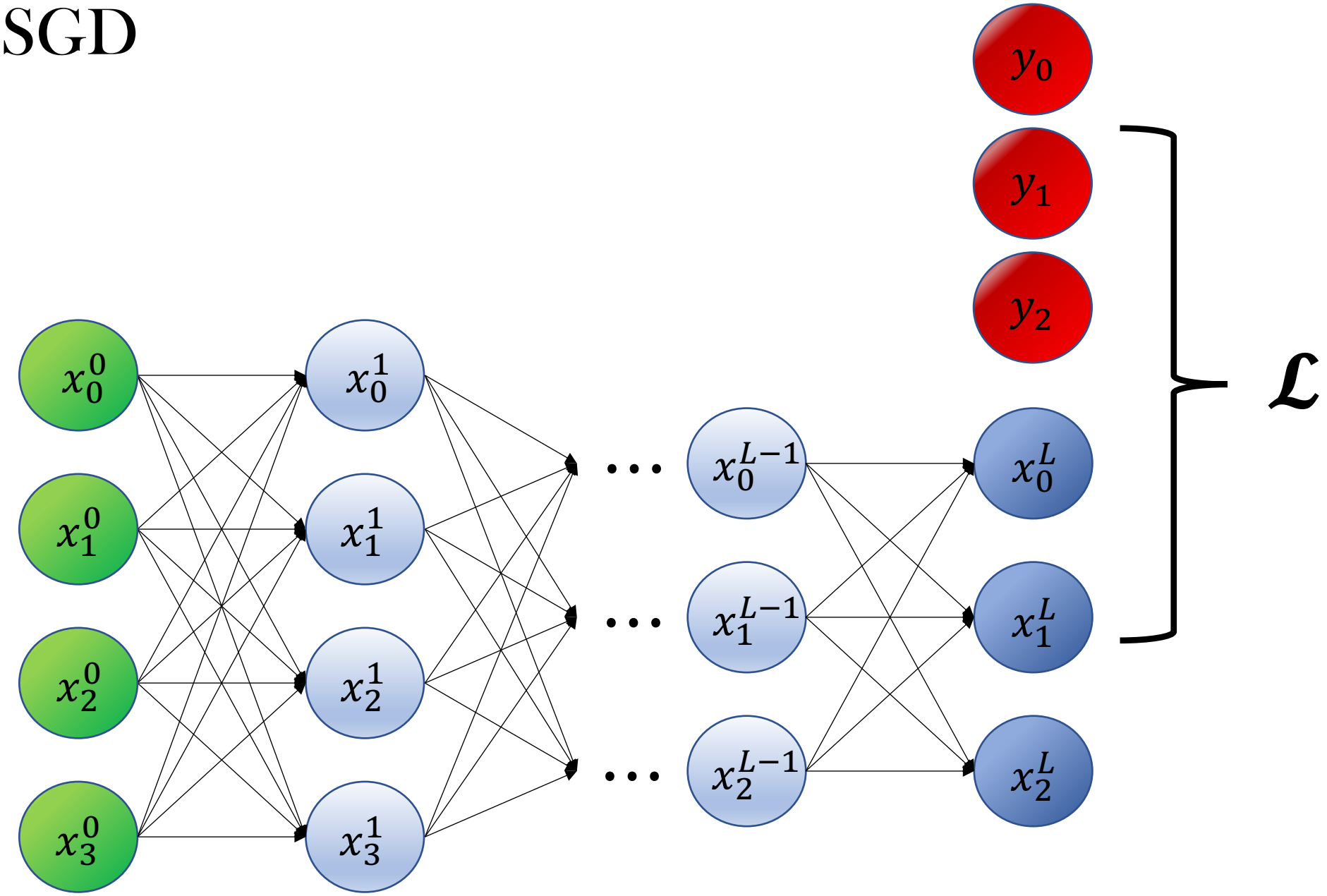
$$x_j^l = \sigma\left(\sum_i w_{ij}^l \cdot x_i^{l-1} + b_j^l\right) \qquad \boldsymbol{x}^l = \boldsymbol{\sigma}\left(\boldsymbol{W}^{l^T}\boldsymbol{x}^{l-1} + \boldsymbol{b}^l\right)$$



Q: What do you call a single layered net?



Learning by SGD



Learning by SGD

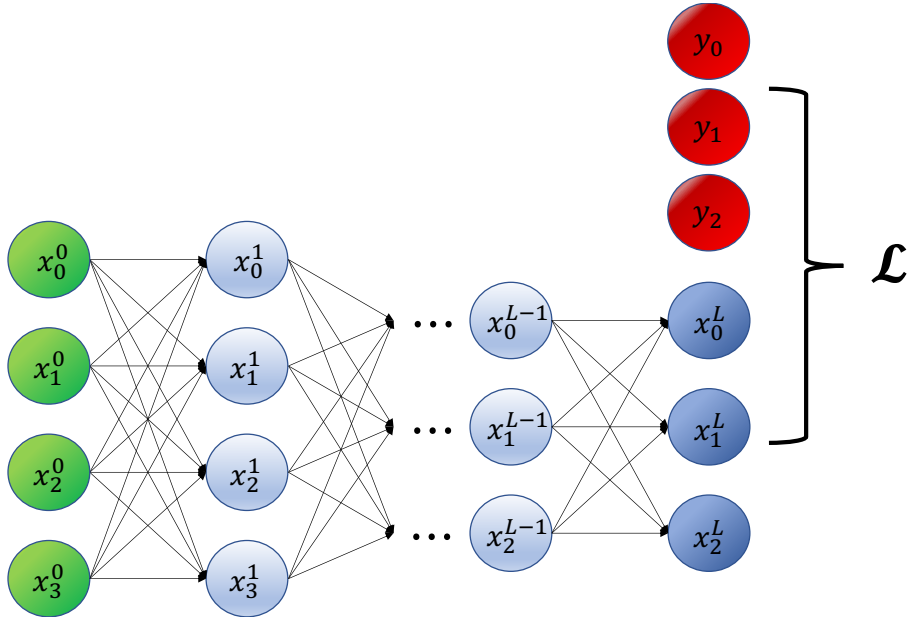
We need $\frac{\partial \mathcal{L}(\boldsymbol{\theta};\, \boldsymbol{x}, \boldsymbol{y})}{\partial w_{ij}^{l}}$, $\frac{\partial \mathcal{L}(\boldsymbol{\theta};\, \boldsymbol{x}, \boldsymbol{y})}{\partial b_{j}^{l}}$ To all l,I,j

$$x_j^L = \sum_i W_{ij}^L x_i^{L-1} + b_j^L$$

$$= \sum_i W_{ij}^L \sigma\left(\sum_k W_{ki}^{L-1} x_k^{L-2} + b_i^{L-1}\right) + b_j^L$$

$$= \sum_i W_{ij}^L \sigma\left(\sum_k W_{ki}^{L-1} \sigma\left(\sum_m W_{mk}^{L-2} x_m^{L-3} + b_k^{L-2}\right) + b_i^{L-1}\right) + b_j^L$$

$\vdots$

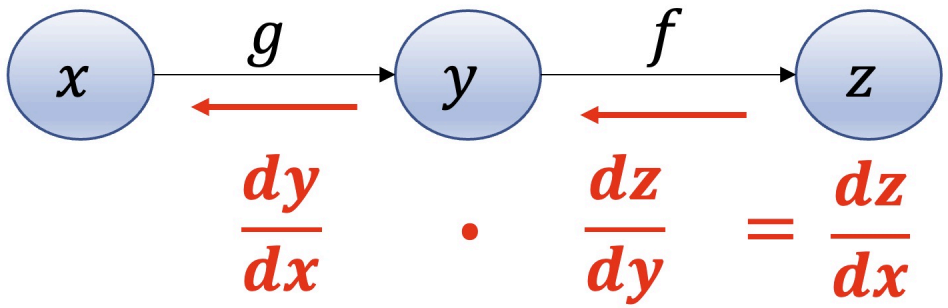


Chain rule reminder

$$f\big(g(x)\big)' = f'\big(g(x)\big) \cdot g'(x)$$

$$y = f(x), \qquad z = g(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\,\frac{dy}{dx}$$

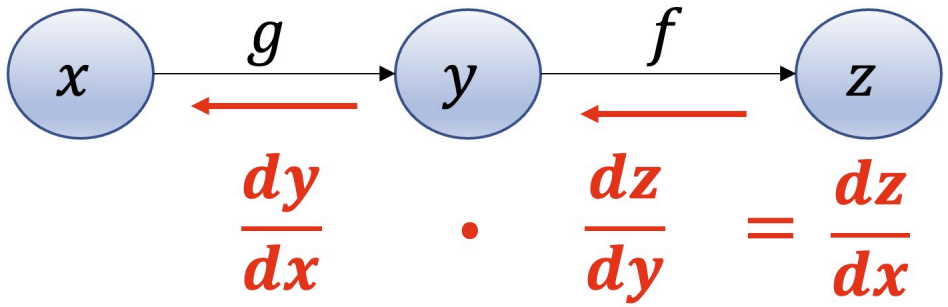


Chain rule reminder

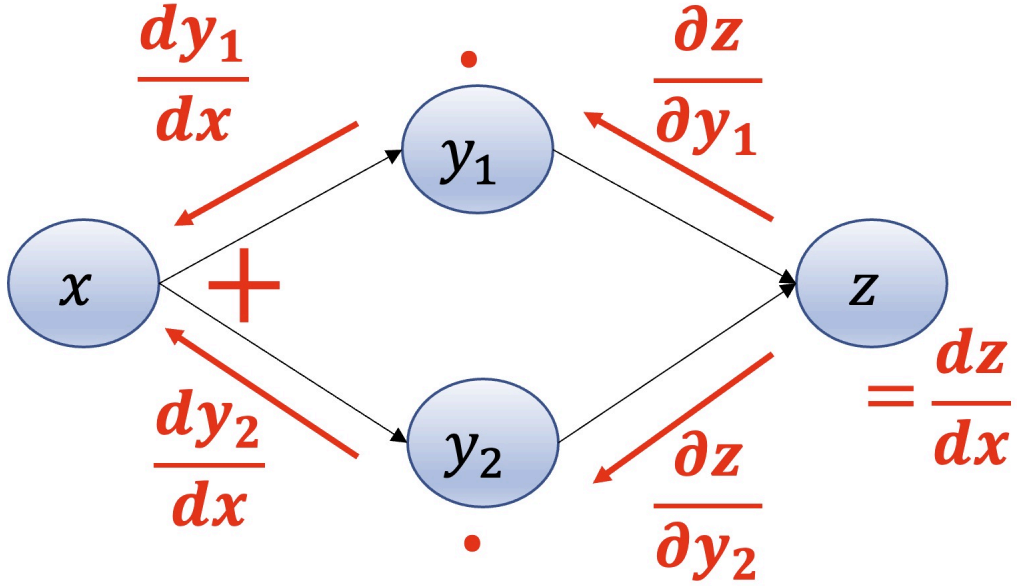
$$f\big(g(x)\big)' = f'\big(g(x)\big) \cdot g'(x)$$

$$y = f(x), \qquad z = g(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\,\frac{dy}{dx}$$



$$\frac{dz(y_1, y_2)}{dx} = \frac{\partial z}{\partial y_1}\,\frac{dy_1}{dx} + \frac{\partial z}{\partial y_2}\,\frac{dy_2}{dx}$$

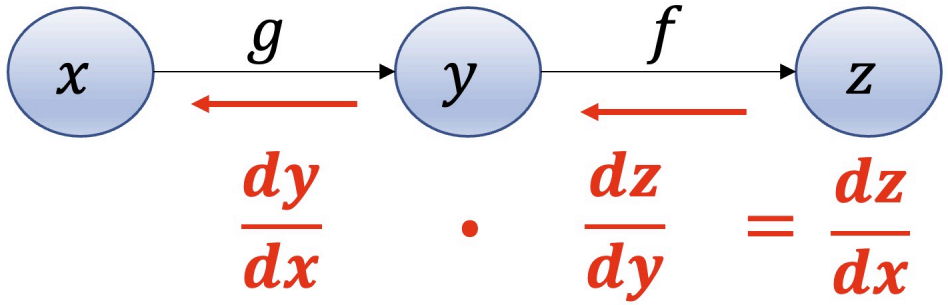


Chain rule reminder

$$f\big(g(x)\big)' = f'\big(g(x)\big)\cdot g'(x)$$

$$y = f(x), \qquad z = g(y)$$

$$\frac{dz}{dx} = \frac{dz}{dy}\,\frac{dy}{dx}$$

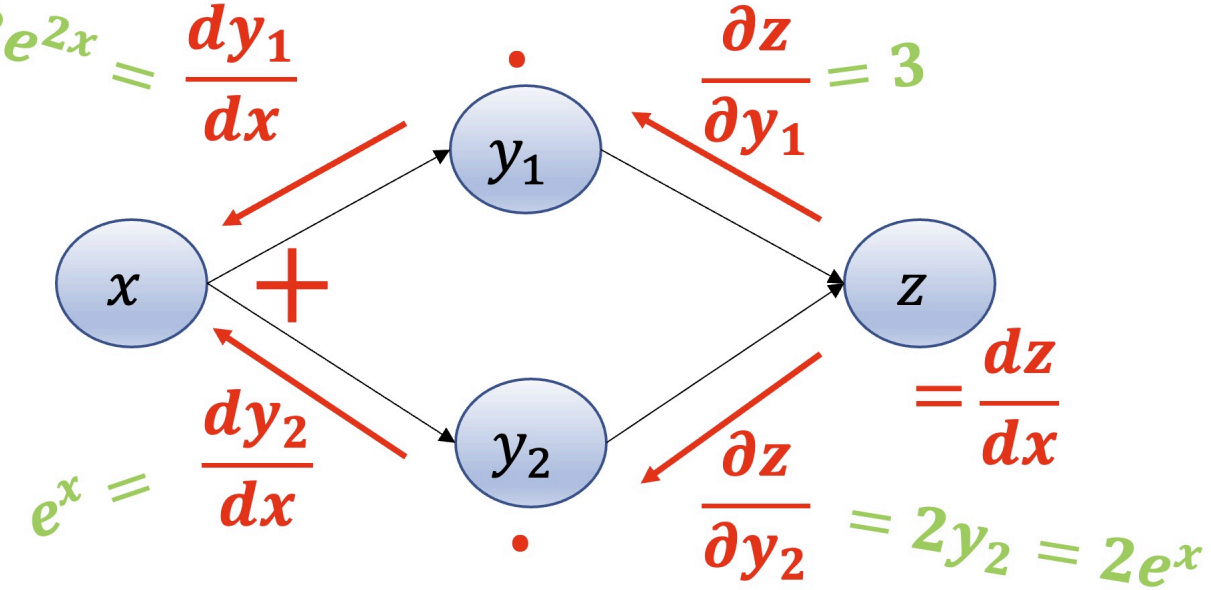


$$\frac{dz(y_1, y_2)}{dx} = \frac{\partial z}{\partial y_1}\,\frac{dy_1}{dx} + \frac{\partial z}{\partial y_2}\,\frac{dy_2}{dx}$$

3 $\quad 4x + 2e^{2x}$ $\quad 2e^{x}$ $\quad e^{x}$

$$y_1 = 2x^2 + e^{2x}$$
$$y_2 = e^{x}$$
$$z = 3y_1 + {y_2}^2$$

$$= 6x^2 + 4e^{2x}$$

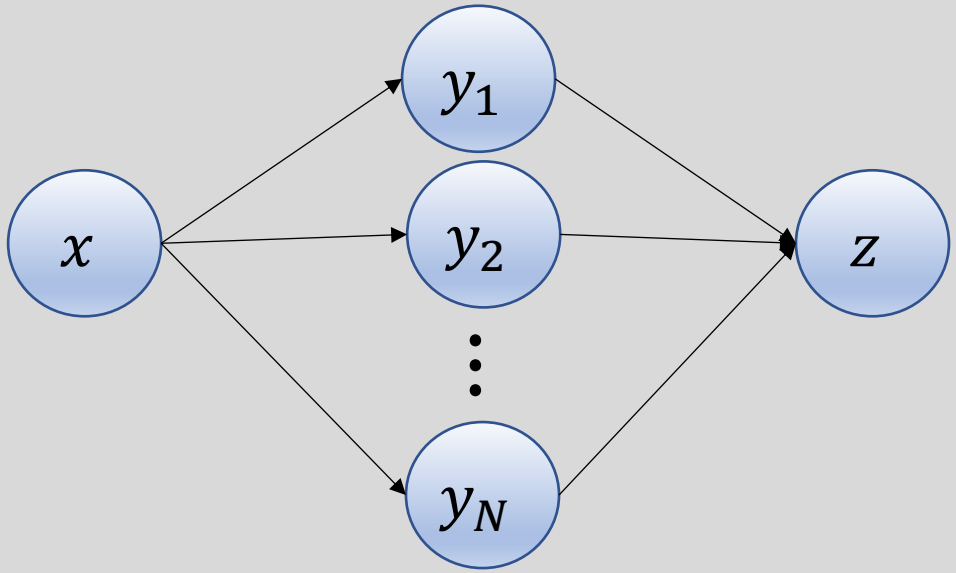


$$\frac{dz(y_1, y_2)}{dx} = 12x + 8e^{2x}$$

Chain rule reminder

Conclusion:

$$\frac{df(y_1, y_2 \dots y_N)}{dx} = \sum_n \frac{\partial f}{\partial y_n} \frac{dy_n}{dx}$$



Back Propagation - preliminaries

$$x_j^l = \sigma\left(\underbrace{\sum_i w_{ij}^l \cdot x_i^{l-1} + b_j}_{z_j^l}\right)$$

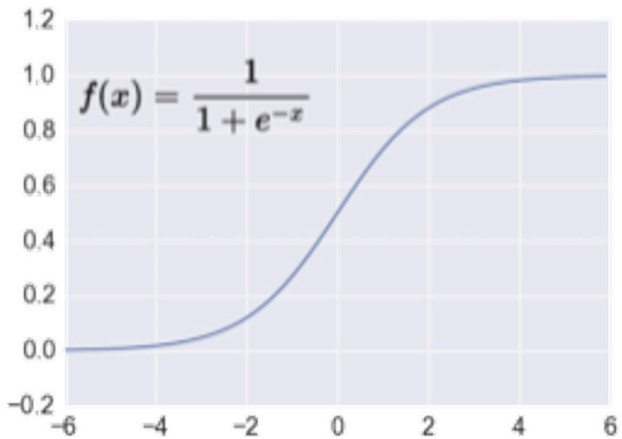
$$\frac{\partial \mathcal{L}}{\partial w_{ij}^l} = \frac{\partial \mathcal{L}}{\partial x_j^l} \cdot \frac{\partial x_j^l}{\partial w_{ij}^l}$$

$\triangleq g_j^l$
Obtained by backprop

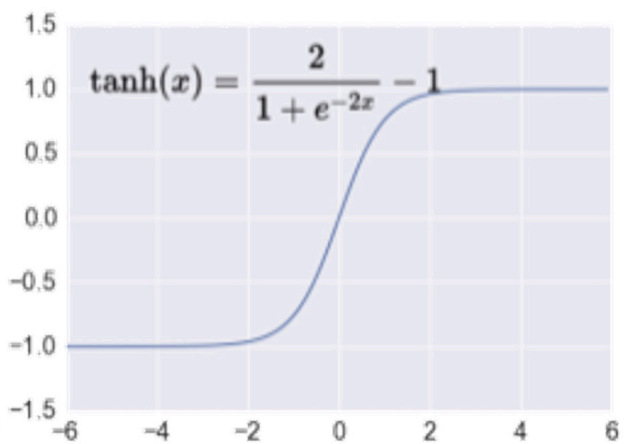
Easy!
$x_i^{l-1} \cdot \sigma'(z_j^l)$

Derivatives of common activations are easy!

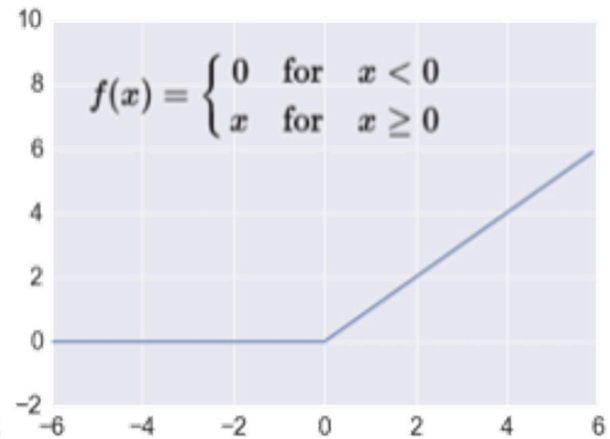
Sigmoid



TanH



ReLU



$\sigma'(z) = \sigma(z)\left(1 - \sigma(z)\right)$

$\tanh'(z) = 1 - \tanh^2(z)$

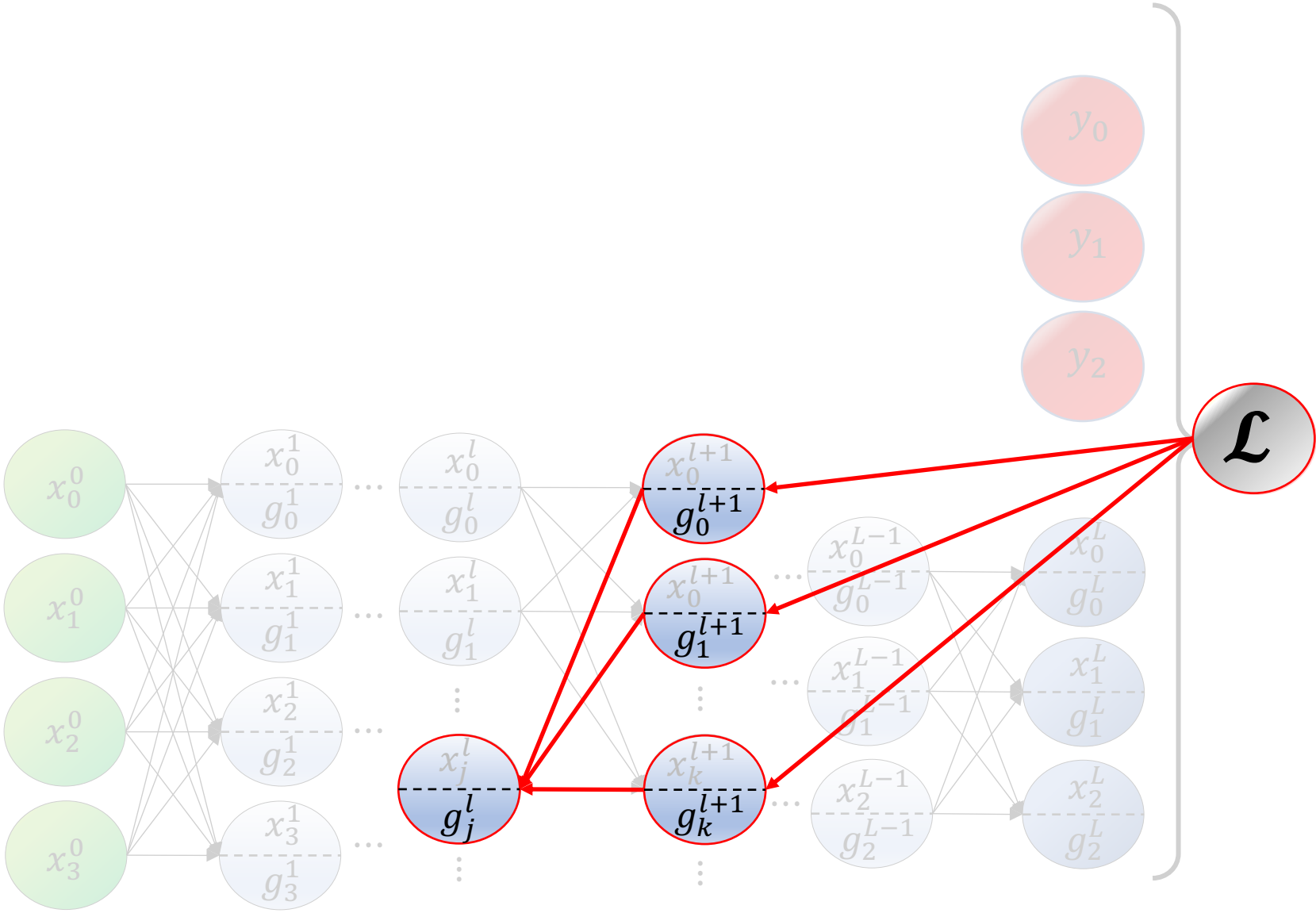
1 for z>0, 0 otherwise

Back Propagation

$$g_j^l \triangleq \frac{\partial \mathcal{L}}{\partial x_j^l}$$

$$= \sum_k \frac{\partial \mathcal{L}}{\partial x_k^{l+1}} \cdot \frac{\partial x_k^{l+1}}{\partial x_j^l}$$

$$x_j^l = \sigma\left(\underbrace{\sum_i w_{ij}^l \cdot x_i^{l-1} + b_j}_{z_j^l}\right)$$



Back Propagation

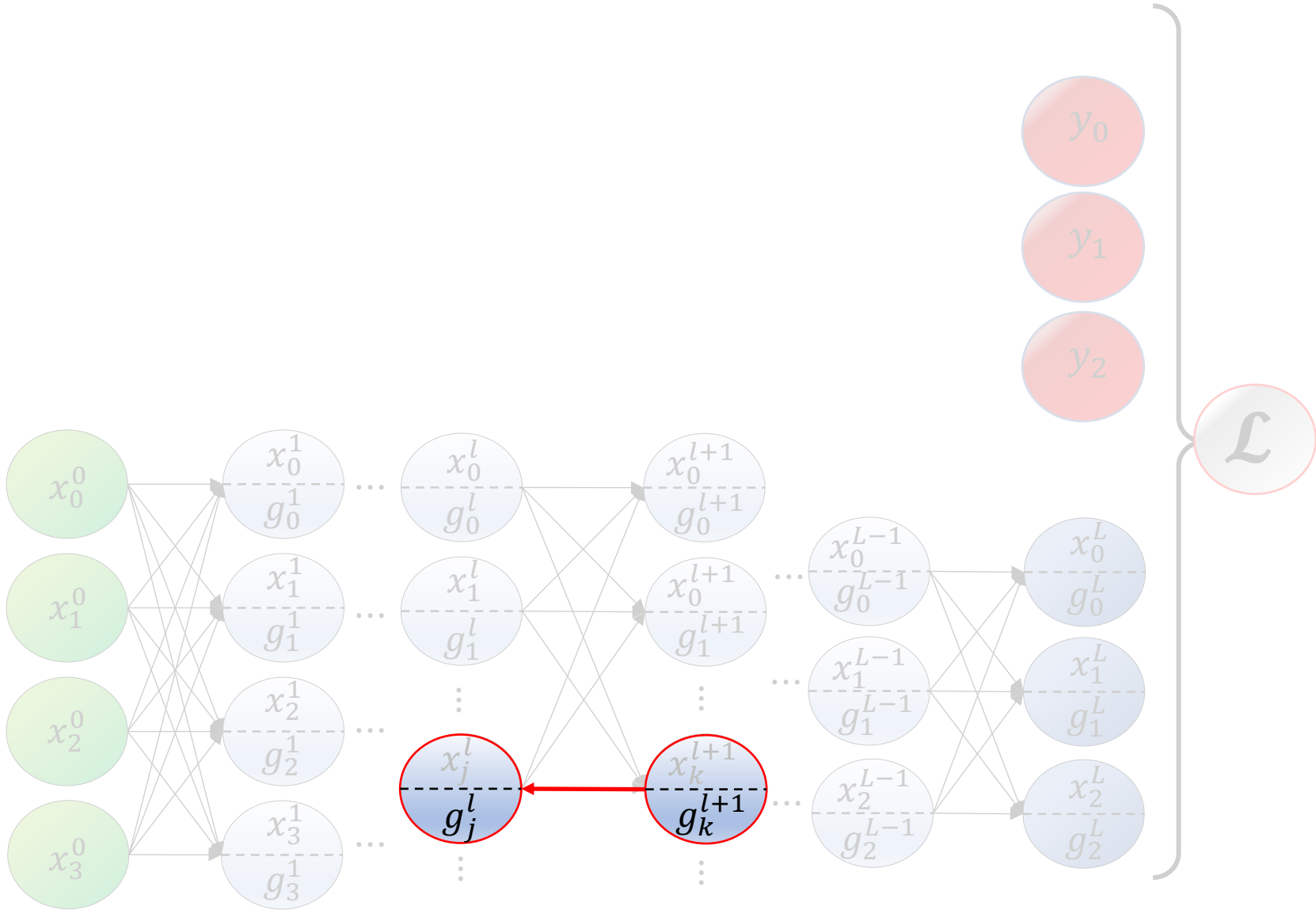
$$g_j^l \triangleq \frac{\partial \mathcal{L}}{\partial x_j^l}$$

$$= \sum_k \frac{\partial \mathcal{L}}{\partial x_k^{l+1}} \cdot \boxed{\frac{\partial x_k^{l+1}}{\partial x_j^l}}$$

$$w_{jk}^{l+1} \cdot \sigma'\left(z_k^{l+1}\right)$$

$$x_j^l = \sigma\left(\sum_i w_{ij}^l \cdot x_i^{l-1} + b_j\right)$$

$$z_j^l$$



Back Propagation

$$g_j^l \triangleq \frac{\partial \mathcal{L}}{\partial x_j^l}$$

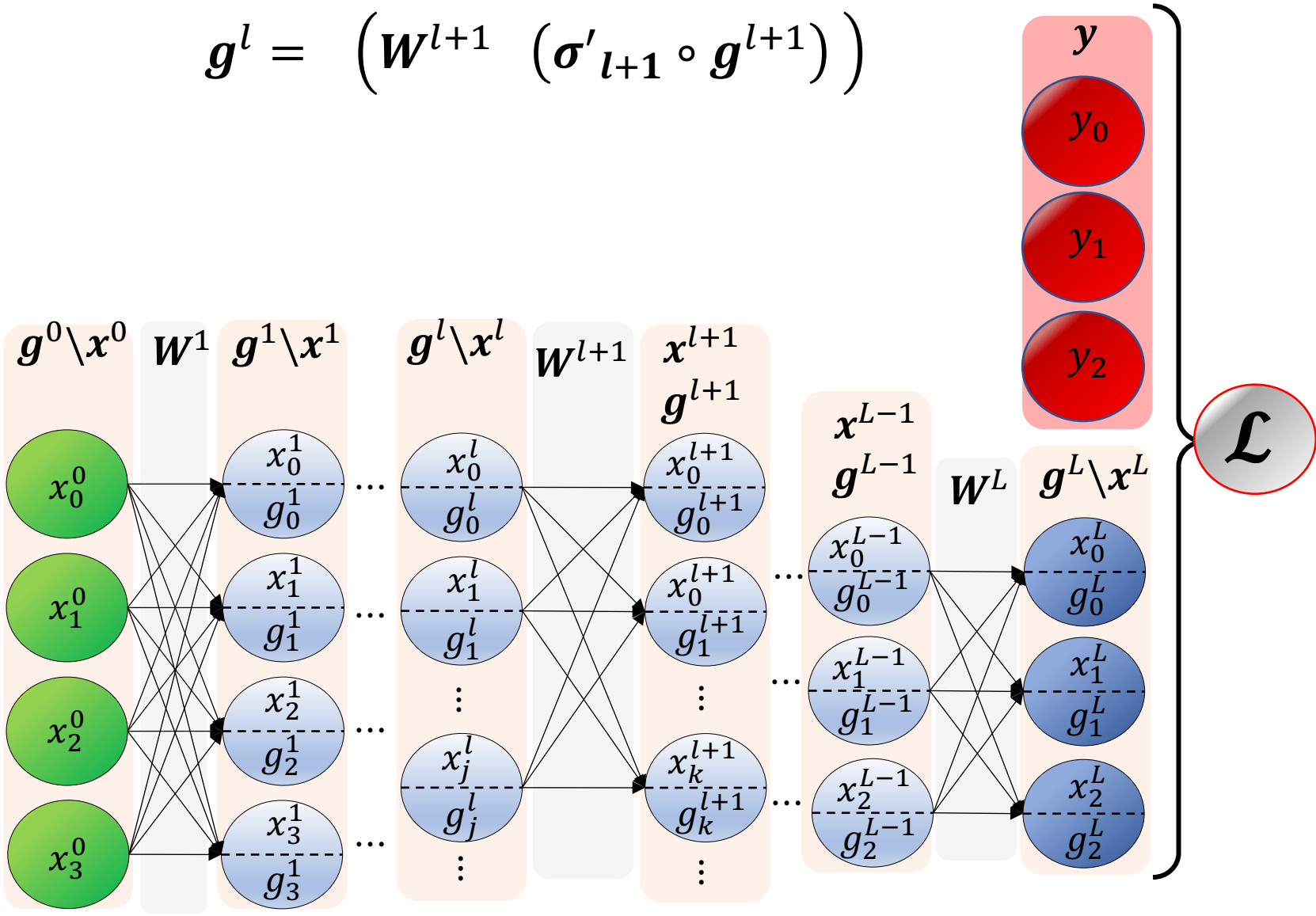
$$= \sum_k \frac{\partial \mathcal{L}}{\partial x_k^{l+1}} \cdot \frac{\partial x_k^{l+1}}{\partial x_j^l}$$

$$= \sum_k g_k^{l+1} \cdot w_{jk}^{l+1} \cdot \sigma'\left(z_k^{l+1}\right)$$

$$\boldsymbol{g}^l = \left(\boldsymbol{W}^{l+1} \left(\boldsymbol{\sigma'}_{\boldsymbol{l+1}} \circ \boldsymbol{g}^{l+1}\right)\right)$$

Stopping criterion:

$$g_j^L = \frac{\partial \mathcal{L}}{\partial x_j^L}$$



Back Propagation

$$g_j^l \triangleq \frac{\partial \mathcal{L}}{\partial x_j^l}$$

$$= \sum_k \frac{\partial \mathcal{L}}{\partial x_k^{l+1}} \cdot \frac{\partial x_k^{l+1}}{\partial x_j^l}$$

$$= \sum_k g_k^{l+1} \cdot w_{jk}^{l+1} \cdot \sigma'\left(z_k^{l+1}\right)$$

Stopping criterion:

$$g_j^L = \frac{\partial \mathcal{L}}{\partial x_j^L}$$

$$\boldsymbol{x}^l = \boldsymbol{\sigma}\left(\boldsymbol{W}^{l^T} \quad \boldsymbol{x}^{l-1} \quad + \boldsymbol{b}\right)$$

$$\boldsymbol{g}^l = \left(\boldsymbol{W}^{l+1} \quad \left(\boldsymbol{\sigma'}_{\boldsymbol{l+1}} \circ \boldsymbol{g}^{l+1}\right)\right)$$

Linear activation

Not Transposed

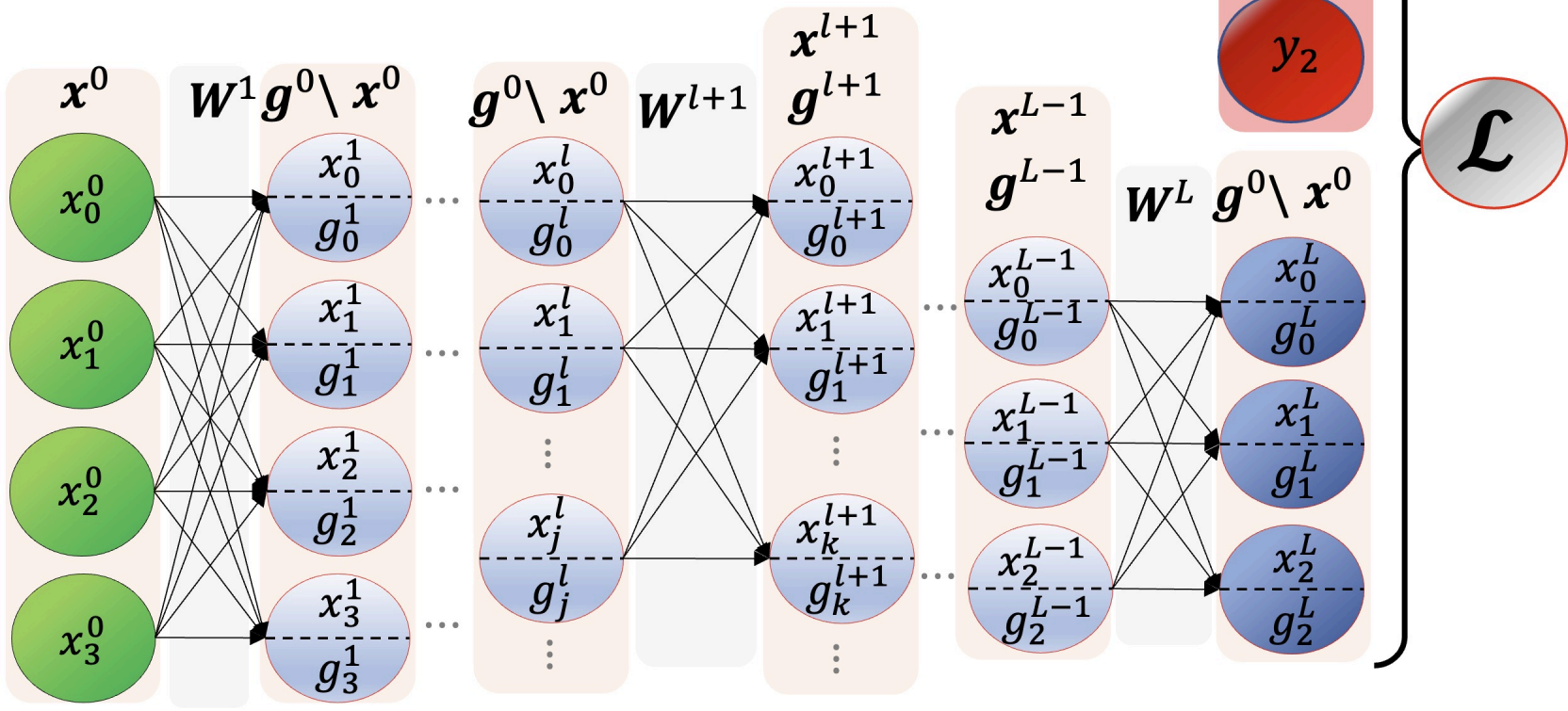
Multiply by activation derivative

No bias



- Initialize weights
- Repeat until convergence:
  1. Sample a batch from the data: $\{(\boldsymbol{x_i}, \boldsymbol{y_i}) \dots\}$
  2. Forward pass: $\boldsymbol{x}^l = \boldsymbol{\sigma}\left(\boldsymbol{W}^{l^T}\boldsymbol{x}^{l-1} + \boldsymbol{b}^l\right)$, $\mathcal{L} = \text{loss}(x^L, y)$
  3. Backward pass: $\boldsymbol{g}^L = \frac{\partial \mathcal{L}}{\partial x^L}$, $\boldsymbol{g}^l = \left(\boldsymbol{W}^{l+1}\left(\boldsymbol{\sigma'}_{\boldsymbol{l+1}} \circ \boldsymbol{g}^{l+1}\right)\right)$
  4. Calculate weights gradient: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l} = \boldsymbol{x}^l \cdot \left(\boldsymbol{\sigma'}(\boldsymbol{z}^l) \circ \boldsymbol{g}^l\right)^T$, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} = \boldsymbol{\sigma'}(\boldsymbol{z}^l) \circ \boldsymbol{g}^l$
  5. Update weights: $\boldsymbol{W}^l := \boldsymbol{W}^l - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l}$, $\boldsymbol{b}^l := \boldsymbol{b}^l - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l}$

Back Propagation



Let's get more generic



Yann LeCun

January 5, 2018 ·

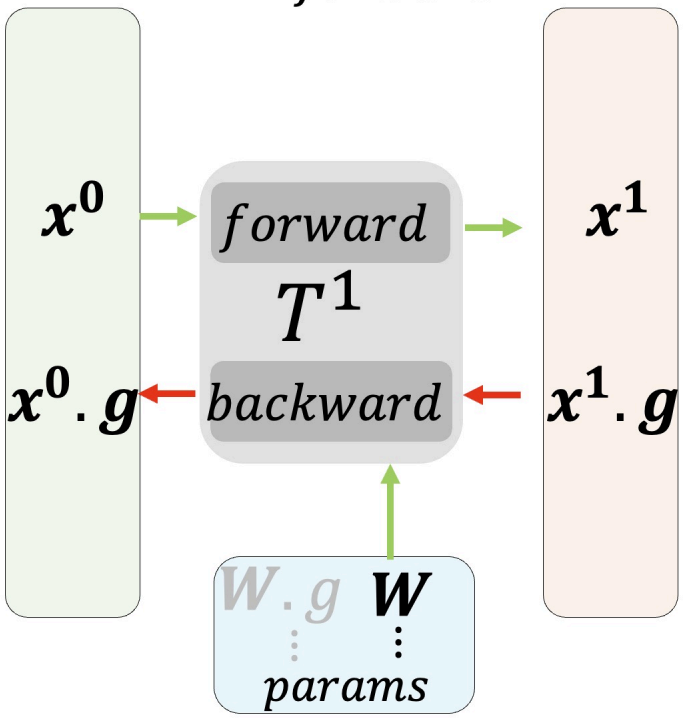
OK, Deep Learning has outlived its usefulness as a buzz-phrase.
Deep Learning est mort. Vive Differentiable Programming!

Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers.

But the important point is that people are now building a new kind of software by assembling networks of parameterized functional blocks and by training them from examples using some form of gradient-based optimization.

Let's get more generic

$$\boldsymbol{x}^{l+1} = T^{l+1}_{forward}\left(\boldsymbol{x}^{l}; T^{l+1}.params\right)$$

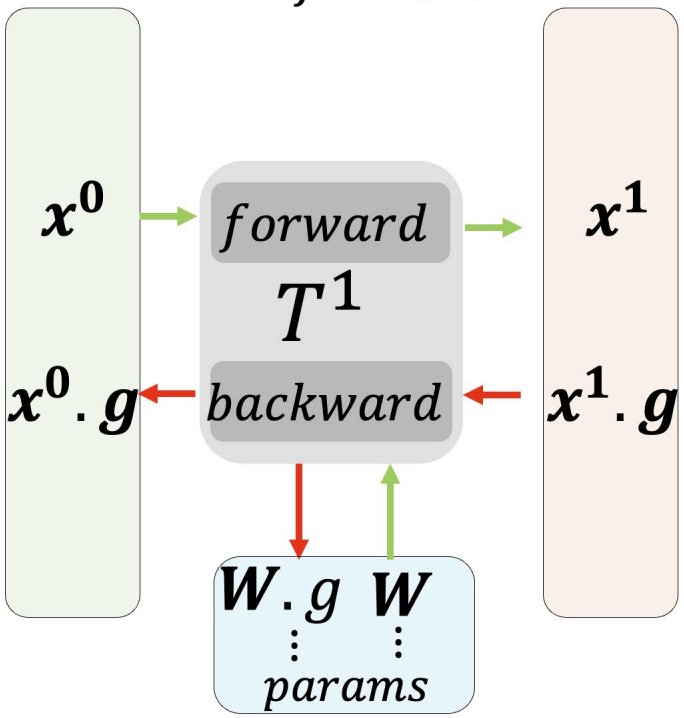


$x^{l+1}.grad$

$$\boldsymbol{x}^{l}.grad = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{l}} = \boxed{\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^{l+1}}} \cdot \frac{\partial \boldsymbol{x}^{l+1}}{\partial \boldsymbol{x}^{l}} = T^{l+1}_{backward}\left(\boldsymbol{x}^{l+1}.grad;\quad T^{l+1}.params,\quad \boldsymbol{x}^{l}\right)$$

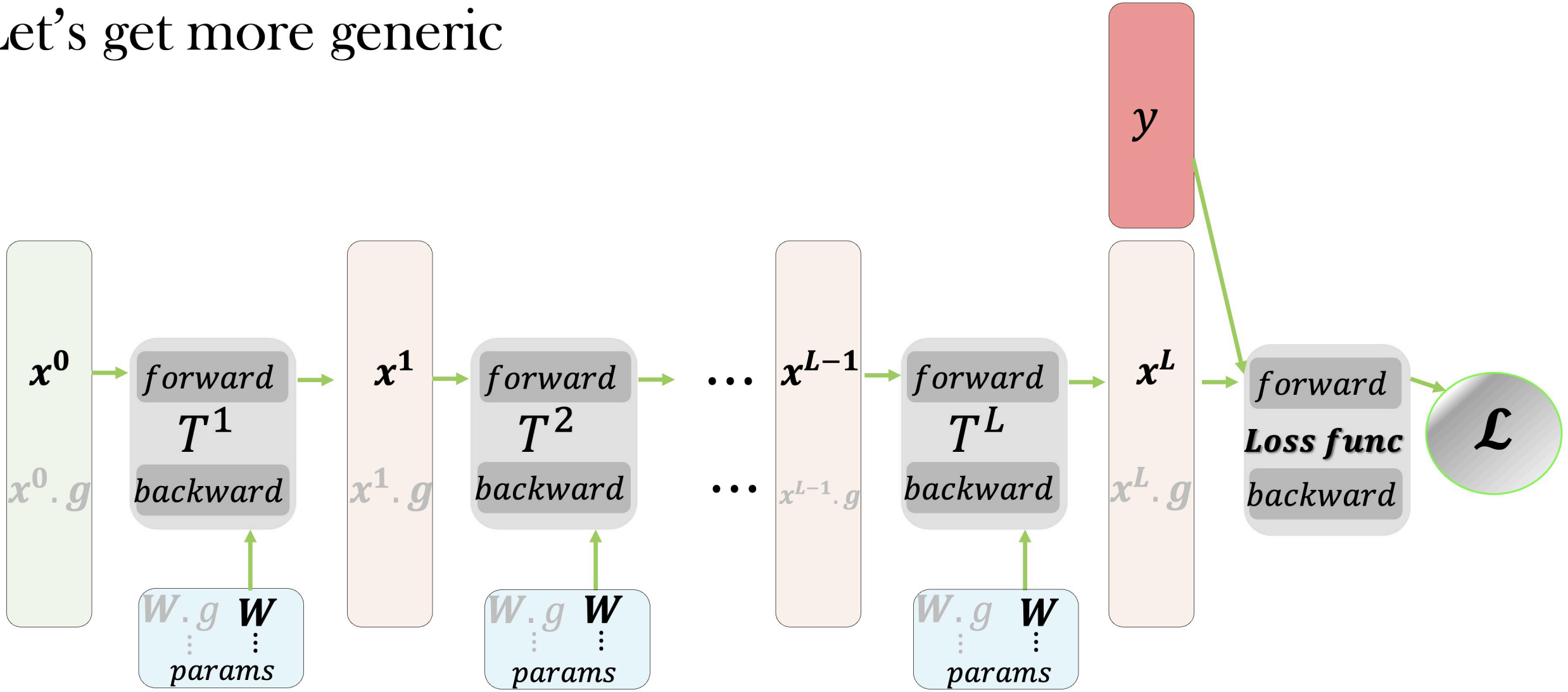
Let's get more generic

$$\boldsymbol{x}^{l+1} = T^{l+1}_{forward}\left(\boldsymbol{x}^{l}; T^{l+1}.params\right)$$

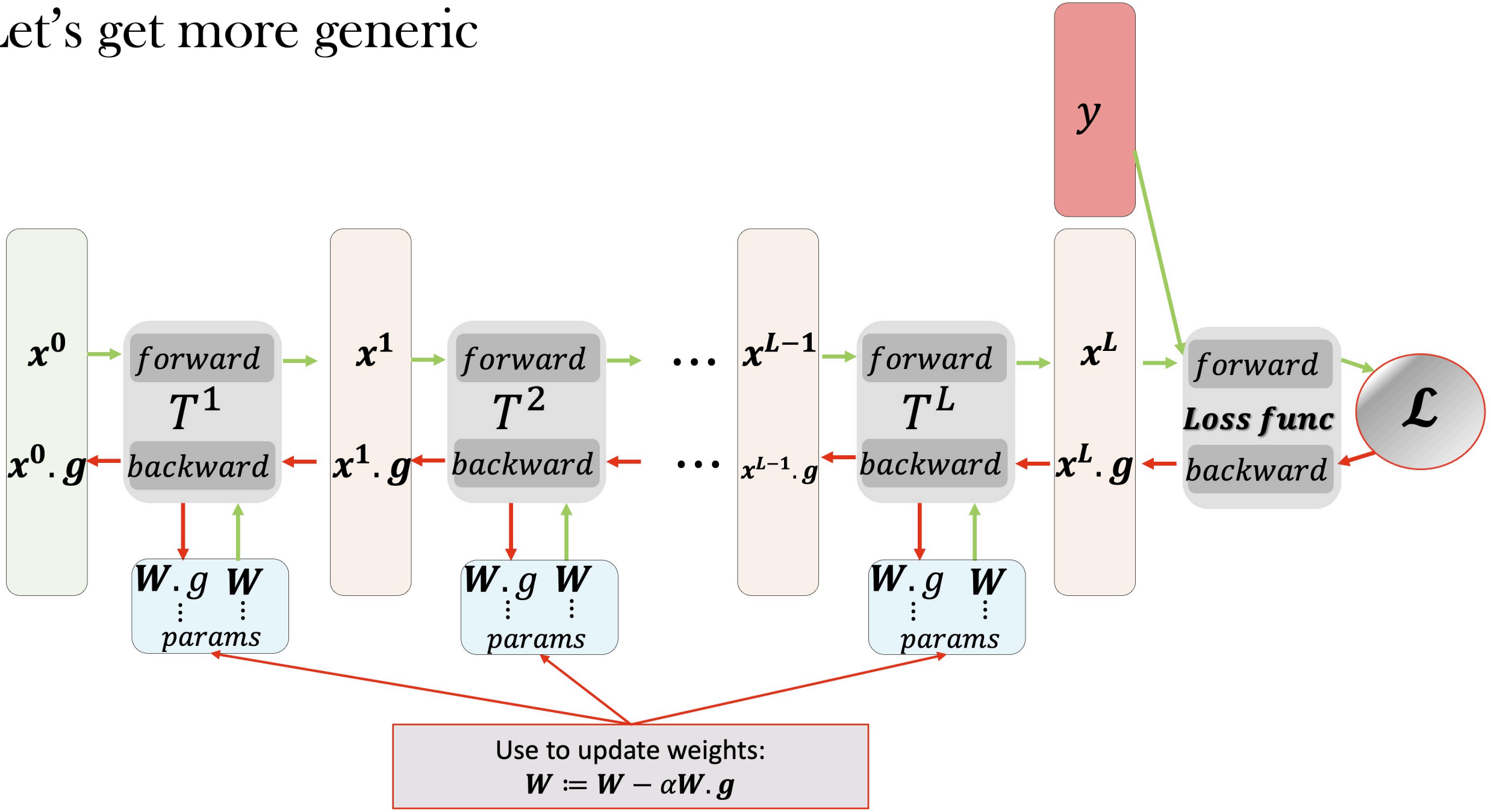


$$\begin{pmatrix} \boldsymbol{x}^{l}.grad \\ T^{l+1}.params.grad \end{pmatrix} = T^{l+1}_{backward}\left(\boldsymbol{x}^{l+1}.grad;\quad T^{l+1}.params,\quad \boldsymbol{x}^{l}\right)$$

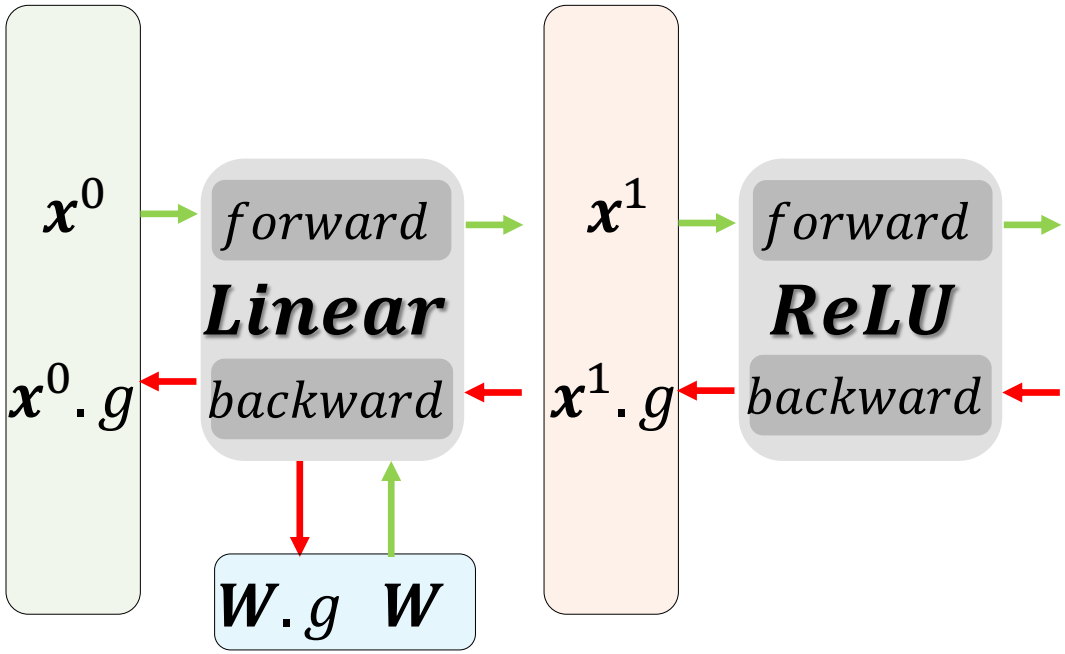
Let's get more generic



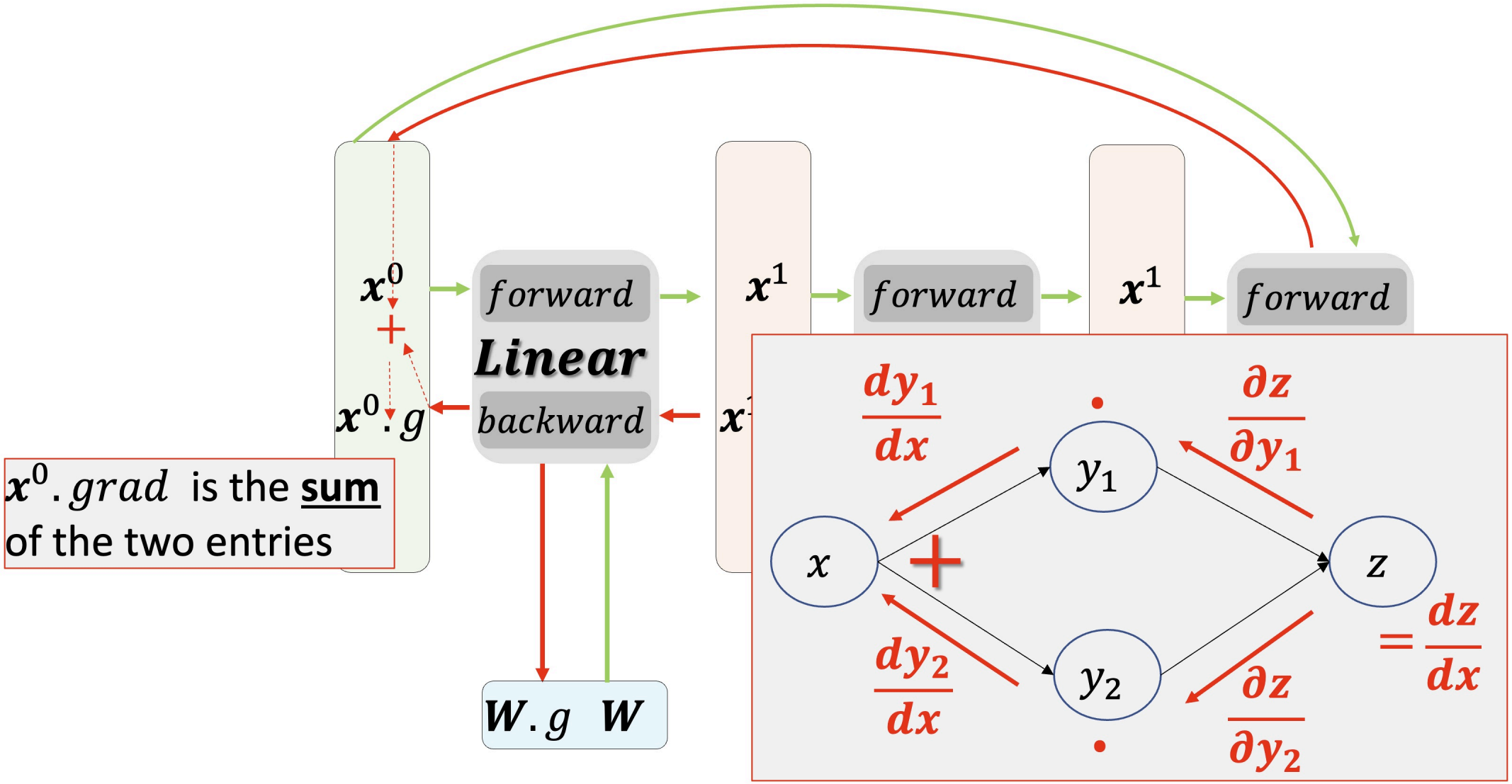
Let's get more generic



Example: Standard layer

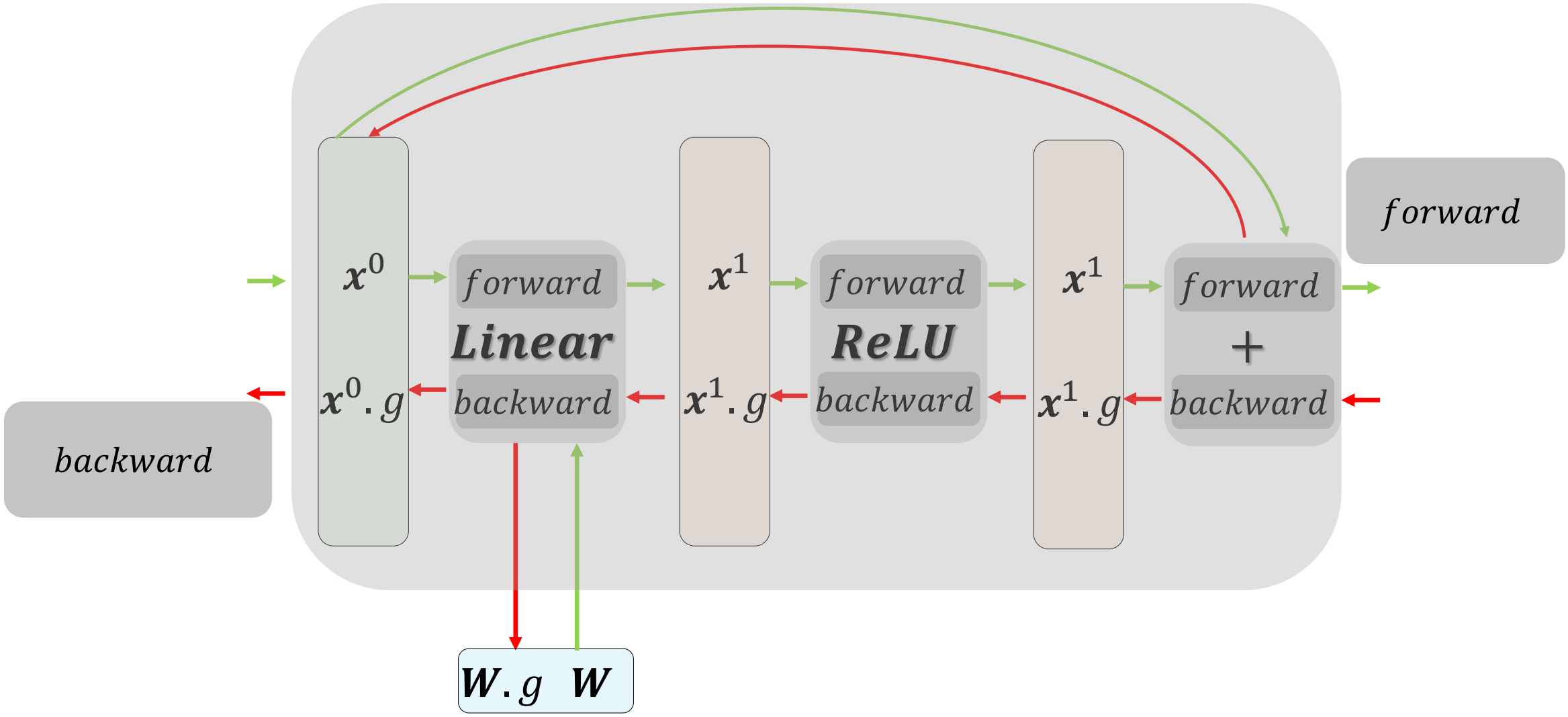


BTW : You can backprop any DAG!



BTW : You can backprop any DAG!

BTW2: Layers (NN modules) can be nested!



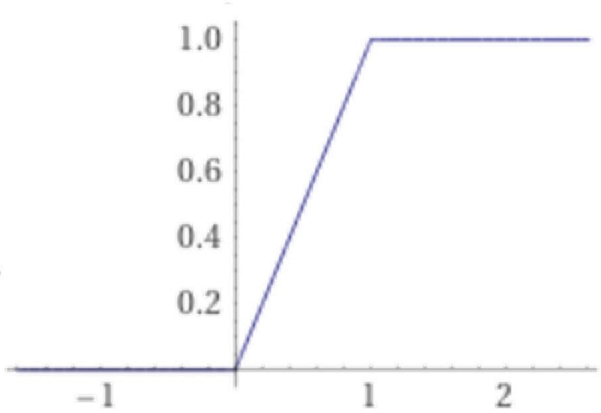
Yes you should understand backprop



Andrej Karpathy Dec 19, 2016 · 7 min read

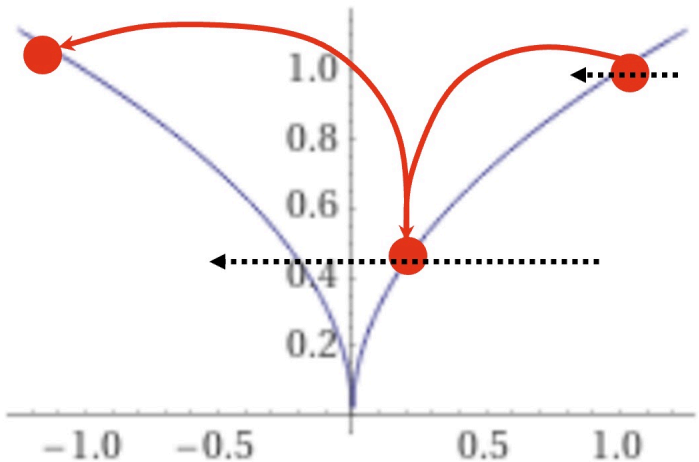
Q: Can we use any function inside a network?

- Has to be differentiable with respect to input and params
- Implications to optimization can be deadly:



  - 0-1 clamp layer in the end of the network.
    - Ideas how to solve this?

  - Square root loss ($L_{1/2}$ norm)

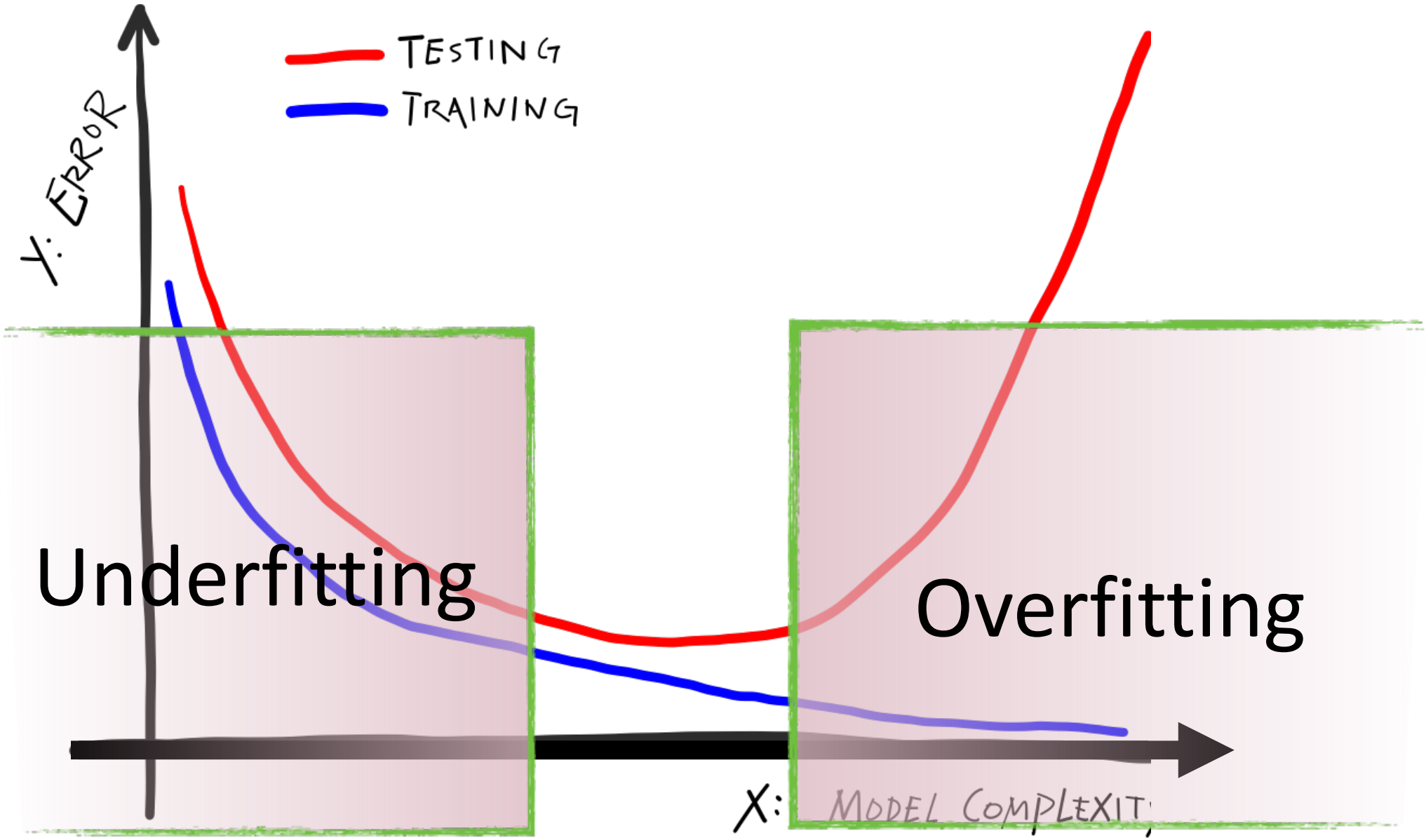


Now, let's talk about generalization

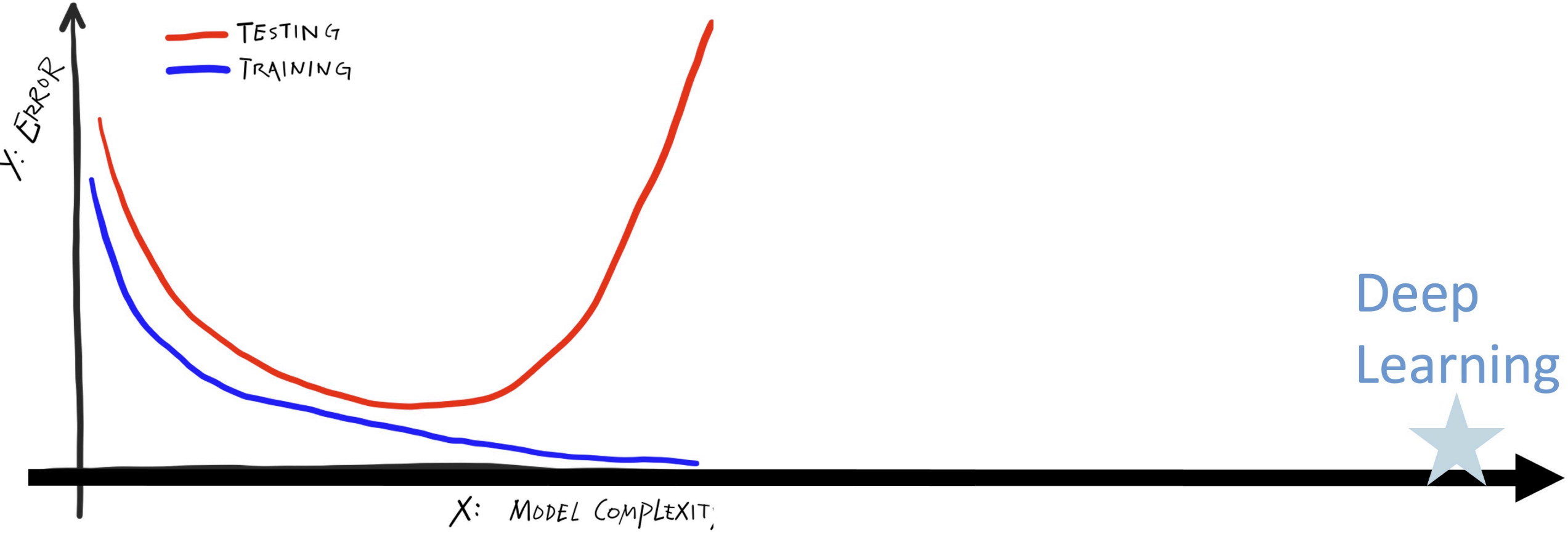
Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, Oriol Vinyals;
UNDERSTANDING DEEP LEARNING REQUIRES RETHINKING GENERALIZATION;
ICLR 2017

Some slides taken from the ICLR oral

Model-Complexity determines Generalization



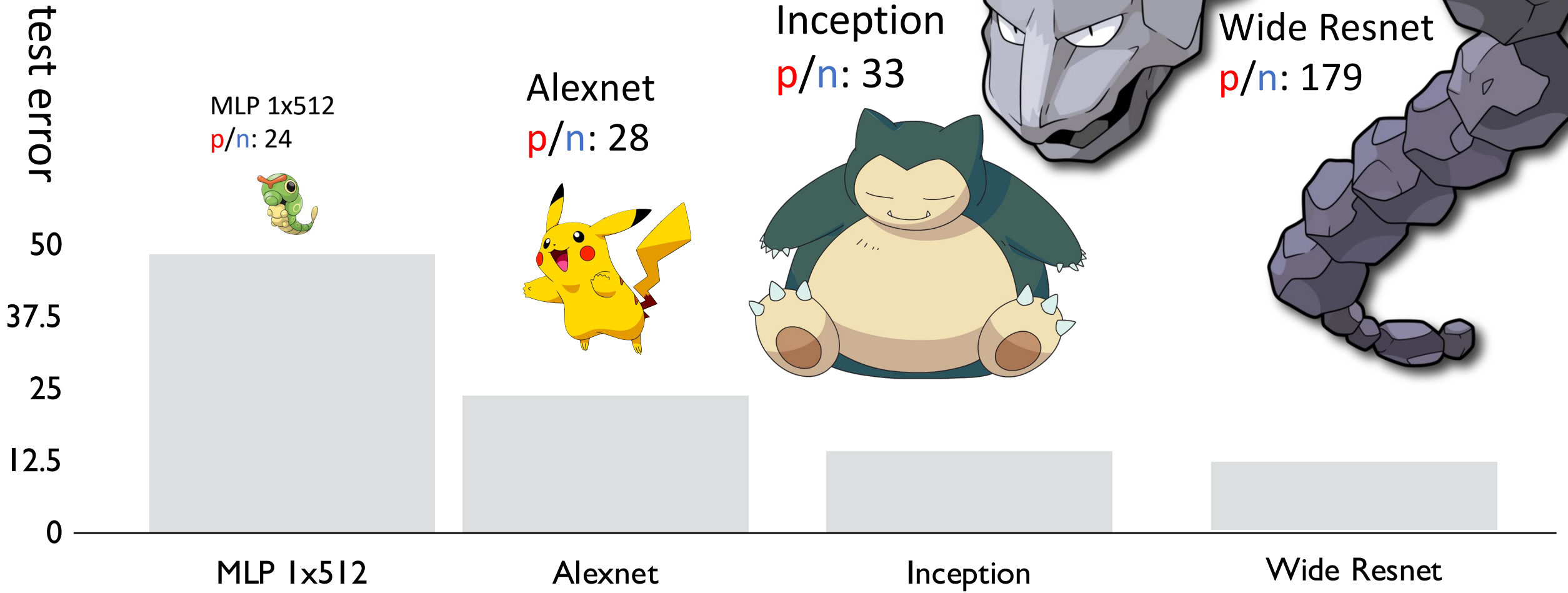
Model-Complexity determines Generalization



Networks are bigger because they train on more data

Parameter Count

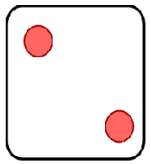
Num Training Samples



Randomization Test



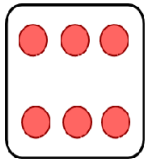
Dog



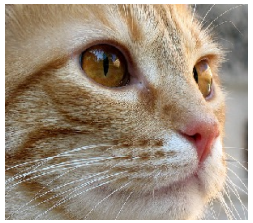
Cat



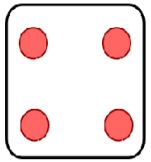
Flower



Dog



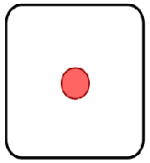
Cat



Bus

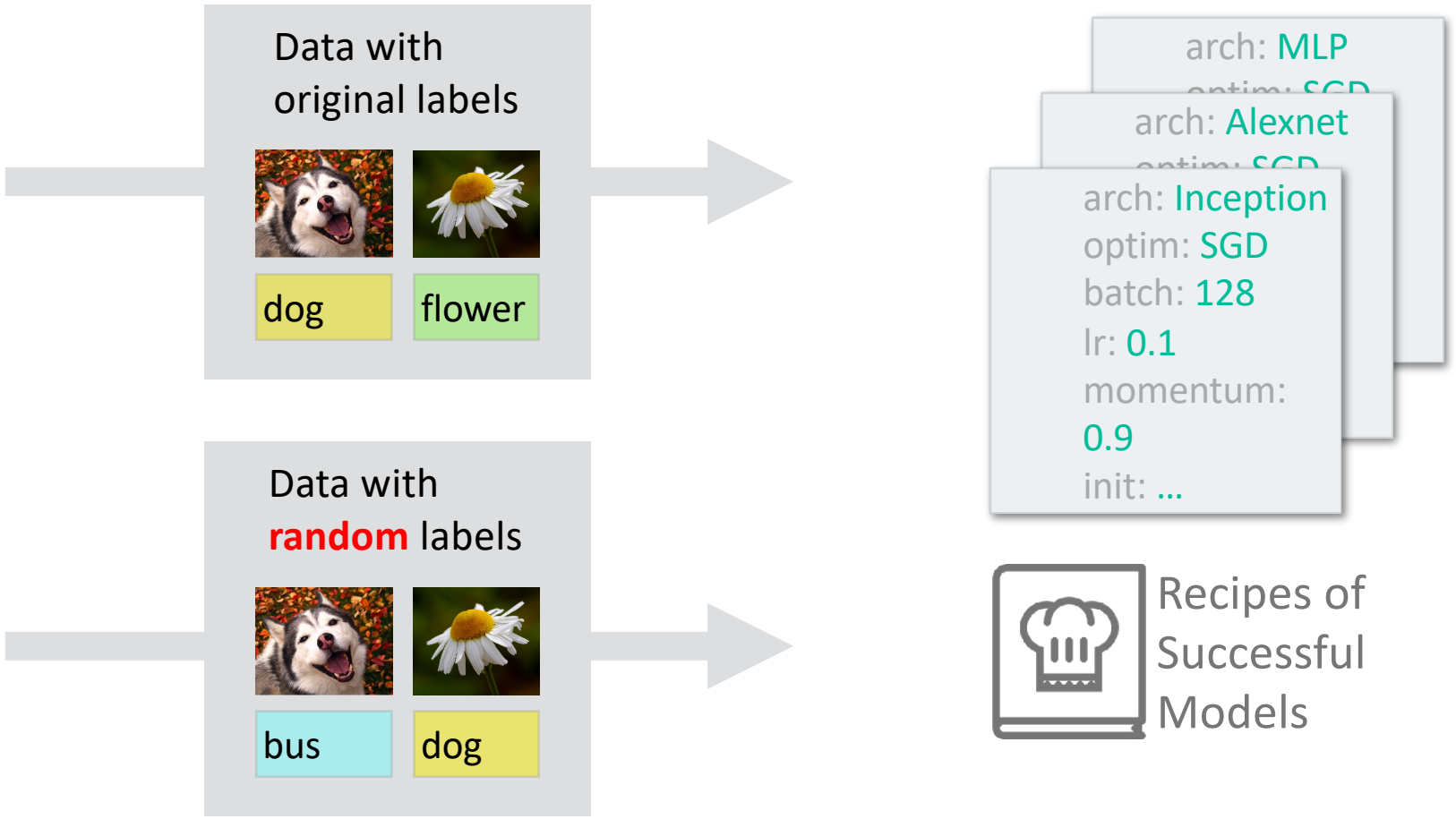


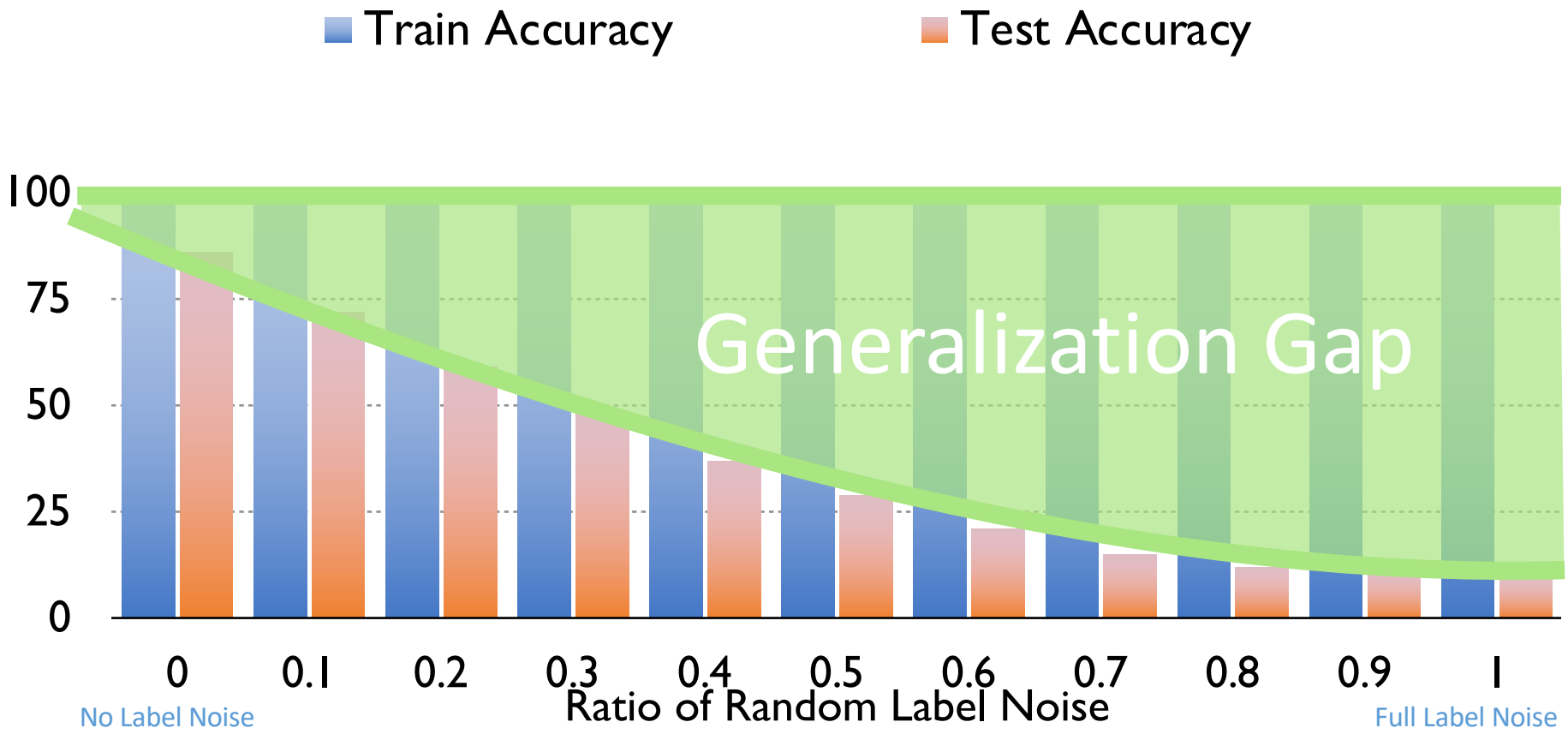
Flower



Bird

Randomization Test




Train Accuracy
Test Accuracy
100
75
50
25
0
Generalization Gap
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
No Label Noise
Ratio of Random Label Noise
Full Label Noise

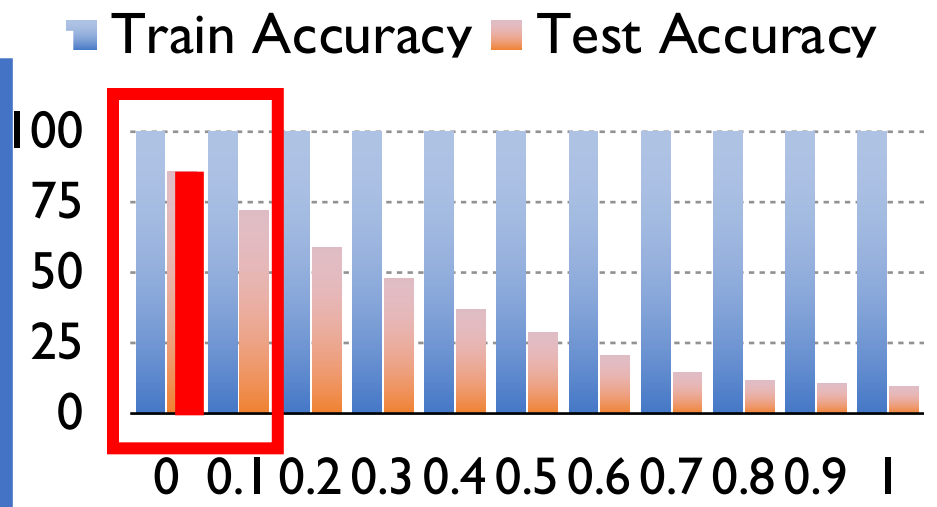
Why is this amazing?

Networks are able to fit random labels → So they have huge expressiveness, SO WHAT? → When trained normally, Networks generalize!

Network behavior changes according to data

A simple rule exists? Network learns it.

No underlying logic? Network memorizes.



Why is this amazing?

Networks are able to fit random labels → So they have huge expressiveness, SO WHAT? → When trained normally, Networks generalize!



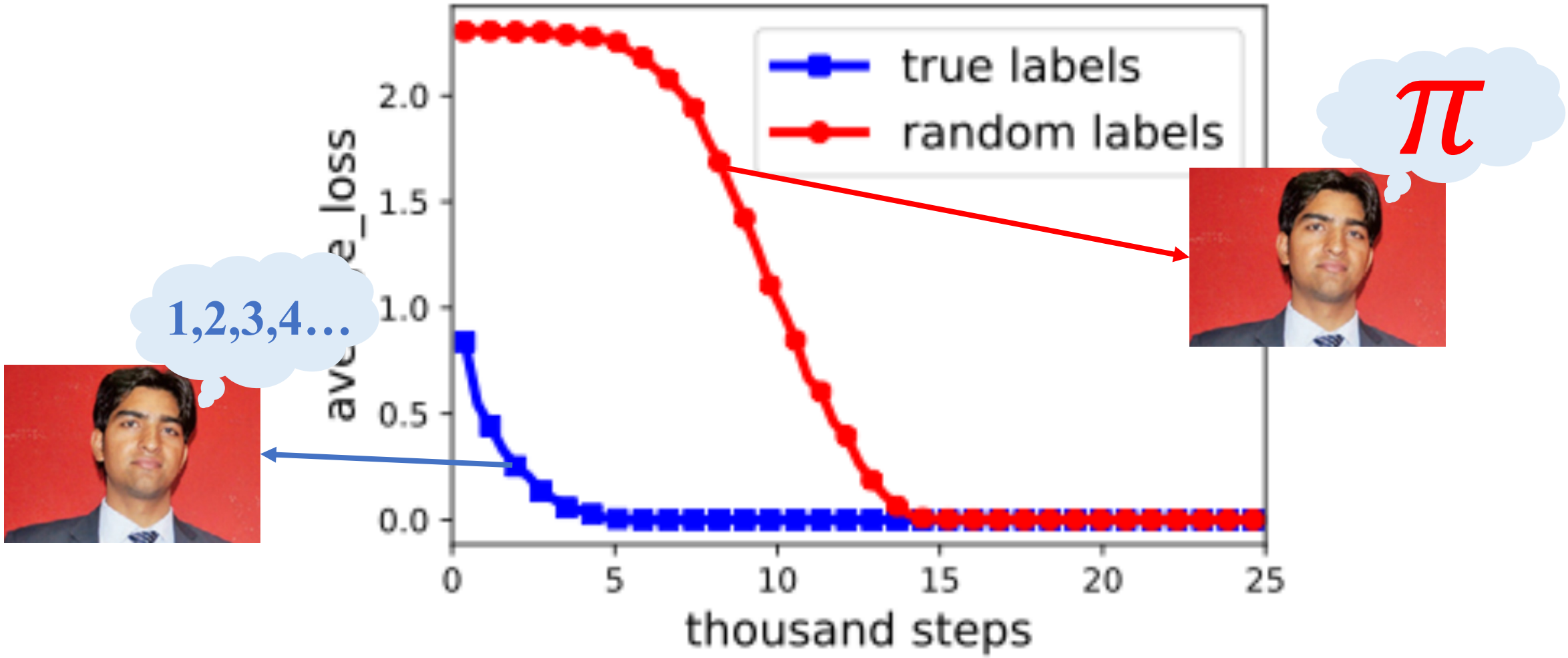
Suresh Kumar Sharma with his certificates. He got into Limca Book of Records for memorizing 70,030 digits of the value of Pi

Q1: Can Suresh predict the 70031st digit?

Q2: How long will it take Suresh to be able to say 1,2,3,4…70030?
Will he then be able to predict the 70031st number?

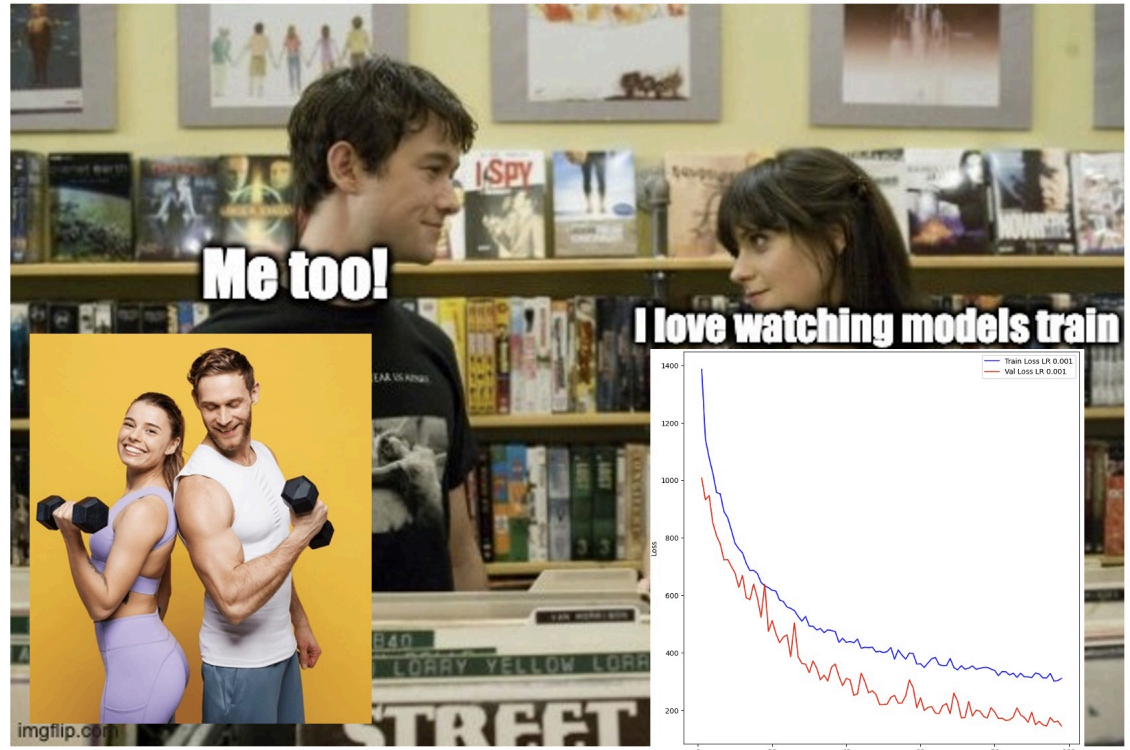
Q3: Does Suresh use the same process in both cases?

Training


true labels
random labels
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
25
thousand steps
1,2,3,4…
π

http://playground.tensorflow.org

Thank you!



Next week's lecture:

Convolutional Neural Networks

