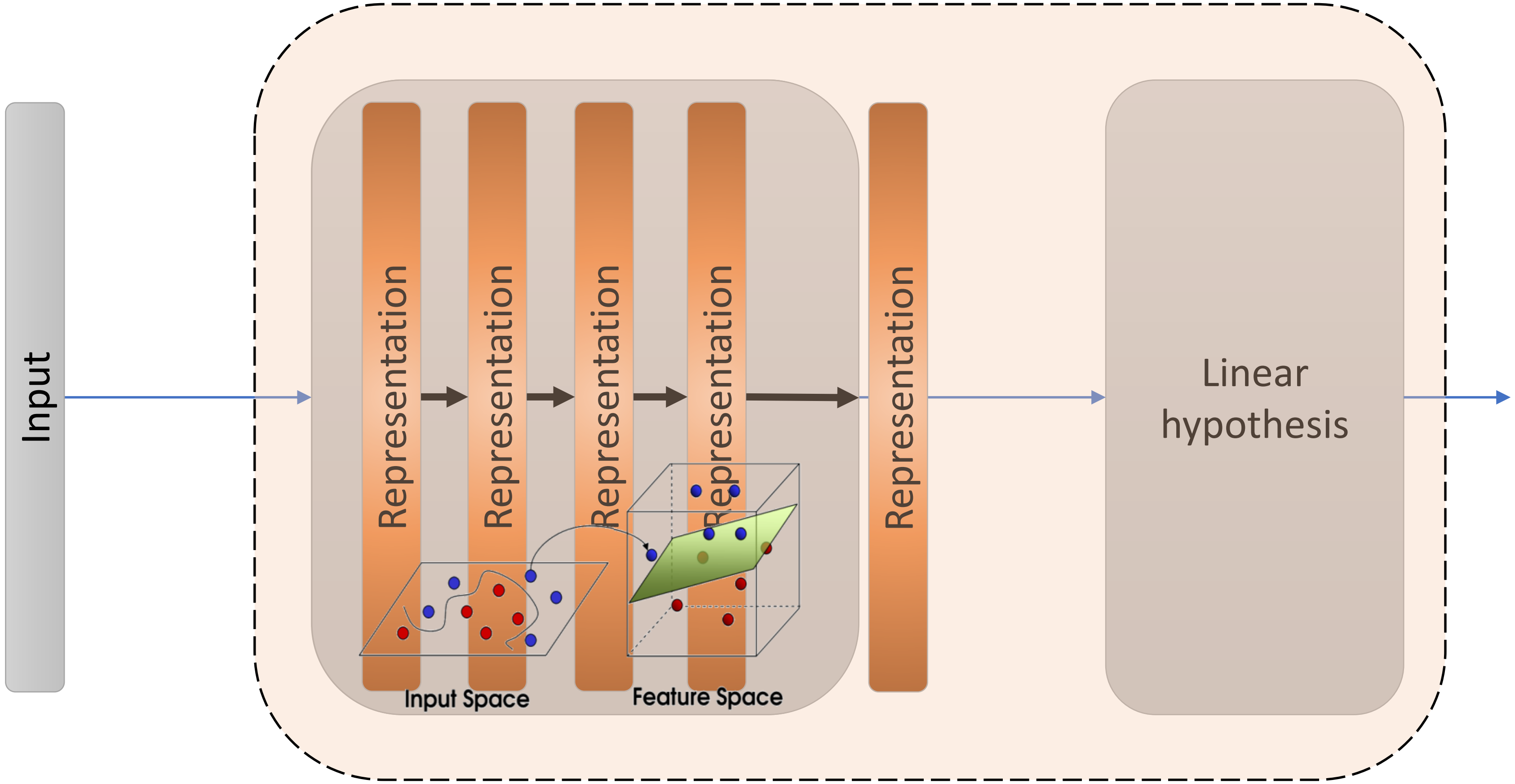
Lecutre2: Neural Networks

Today:

- Revisit feature transform (5%)
- What is a neural net? (10%)
- Derivatives and chain-rule reminder (10%)
- Training a vanilla network (back-prop) (40%)
- Differential computational graph (25%)
- Demo (10%)

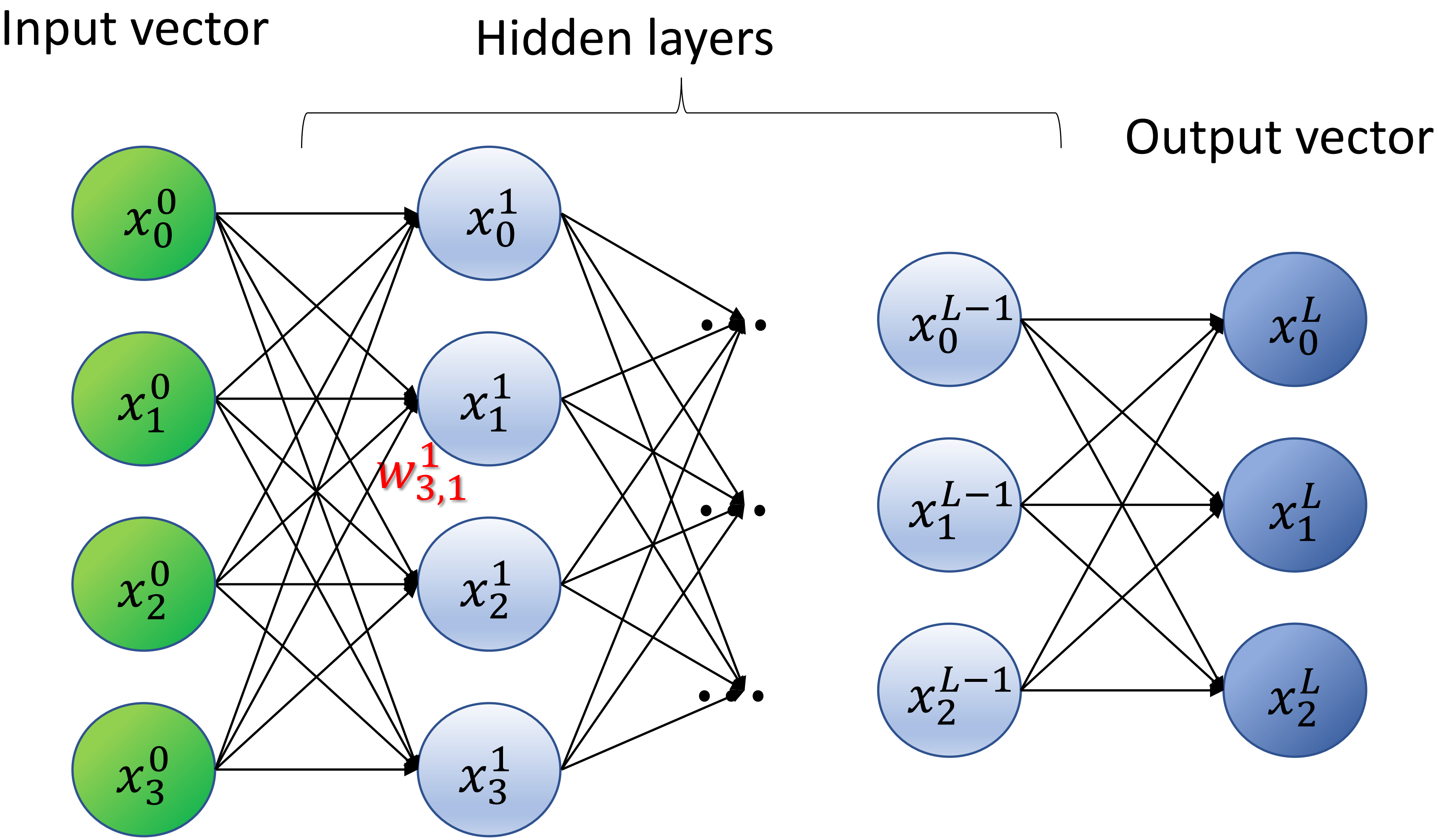
Feature transform



Non-linear hypothesis!

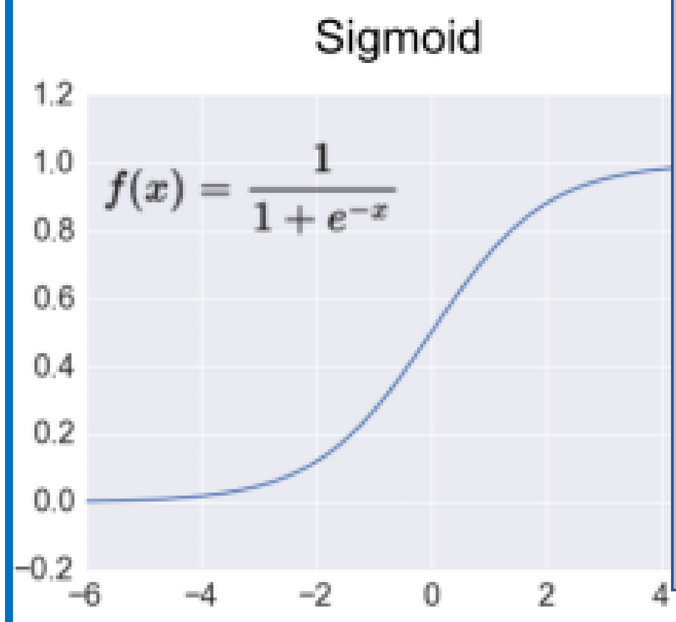
Artificial Neural Networks

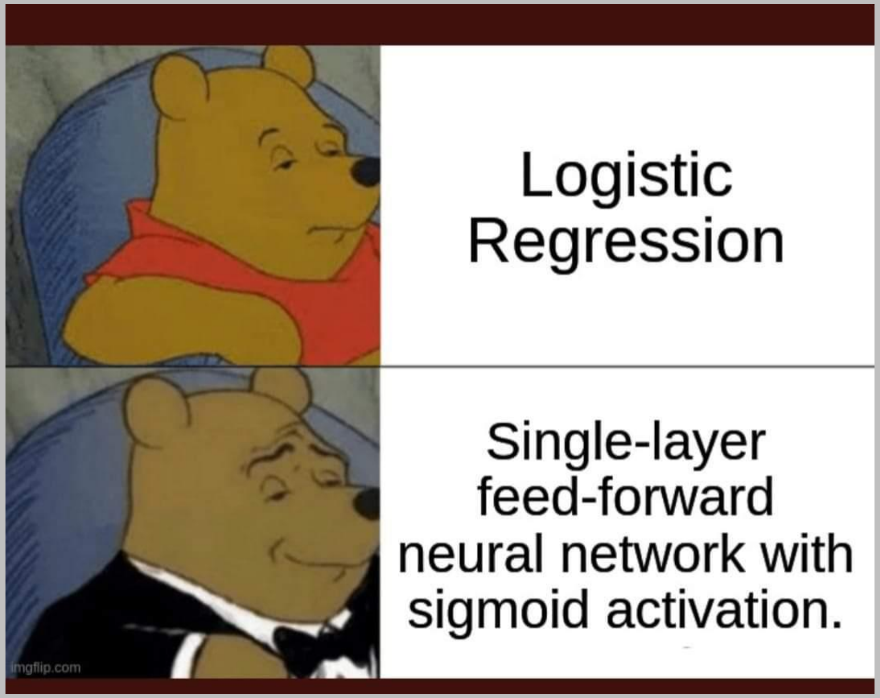
Vaguely inspired by biological neural networks



Q: What do you call a single layered net?

Q: Why?



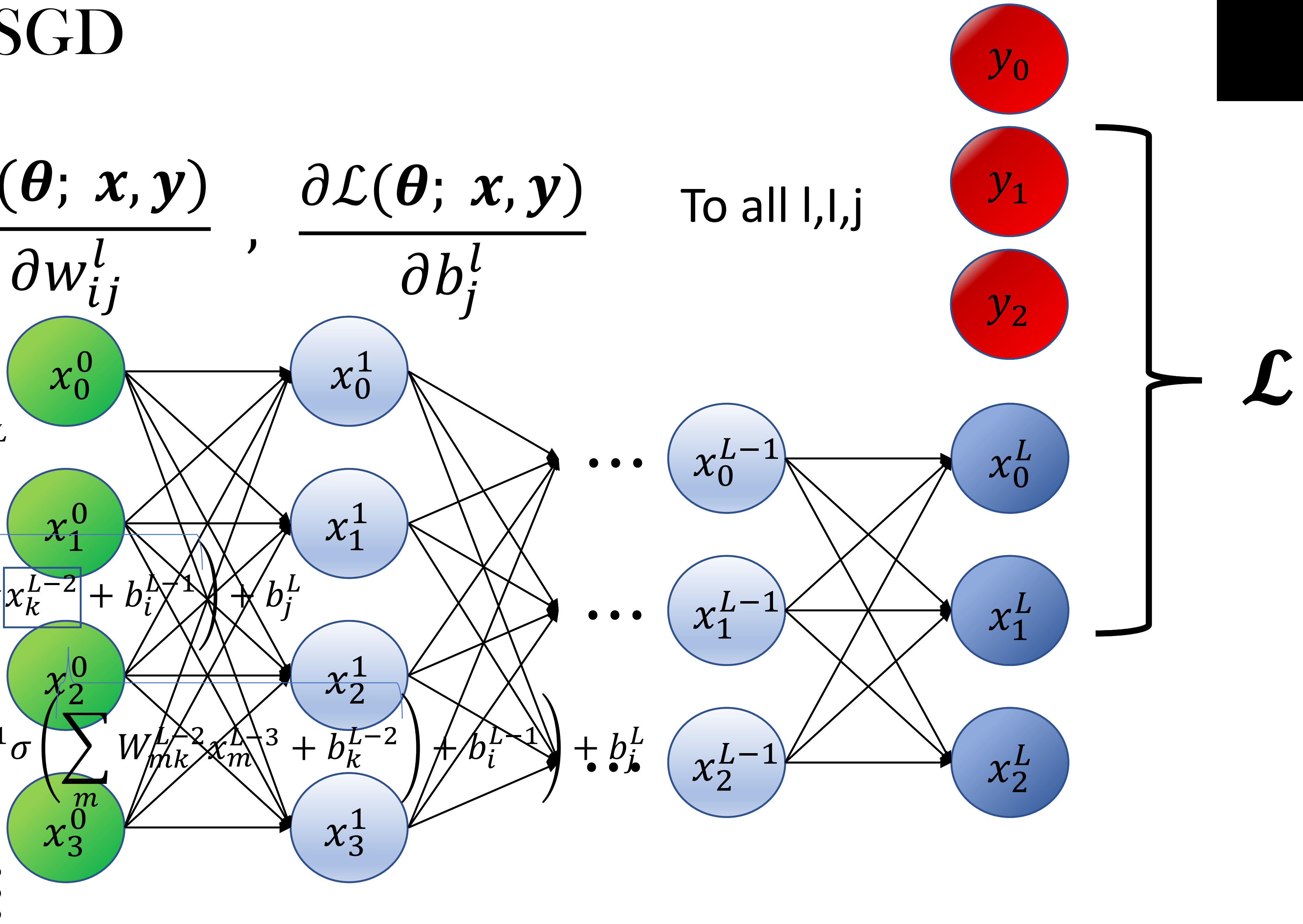


Linear

Feature transform

Learning by SGD

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



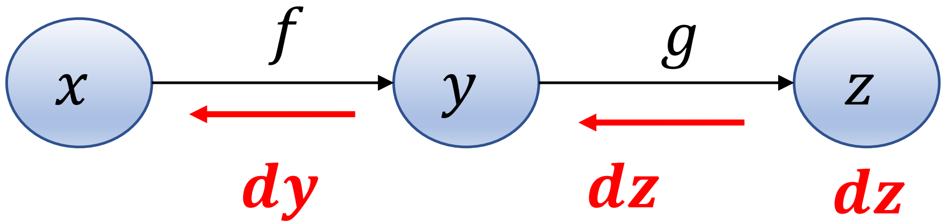
$$x_j^L = \sum_i W_{ij}^L \boxed{x_i^{L-1}} + b_j^L$$

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

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

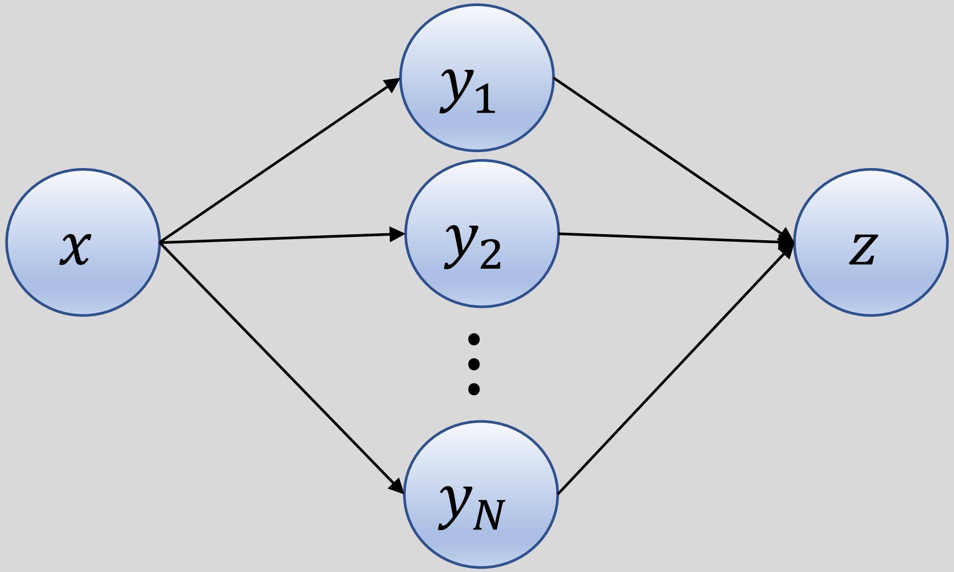
Chain rule reminder

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



Conclusion:

$$\frac{dz(y_1, y_2 \ldots y_N)}{dx} = \sum_n \frac{\partial z}{\partial y_n} \frac{dy_n}{dx}$$



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

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

Back Propagation - preliminaries

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

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

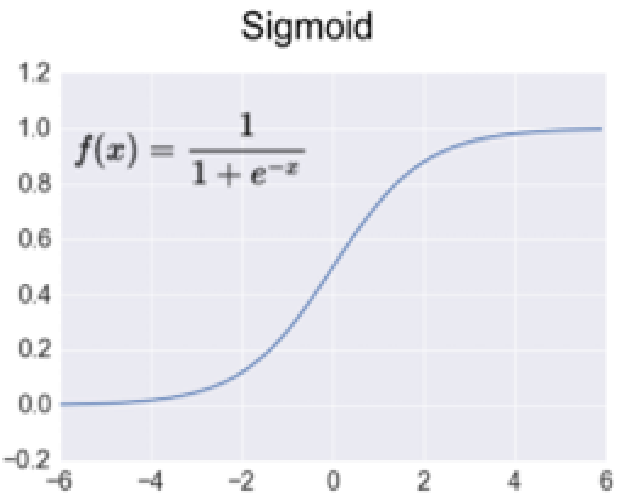
$\triangleq g_j^l$

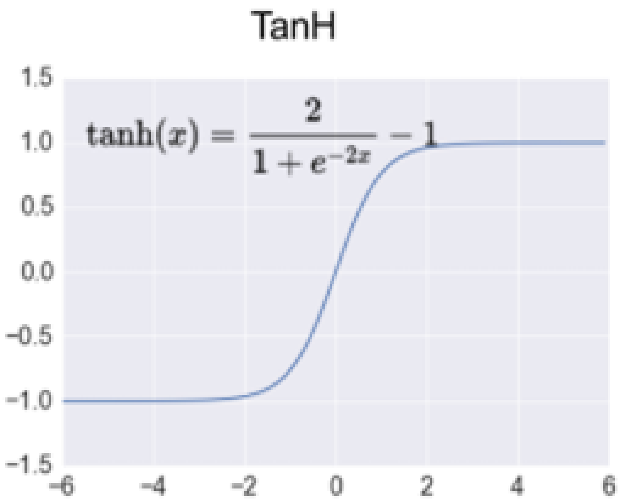
Obtained by backprop

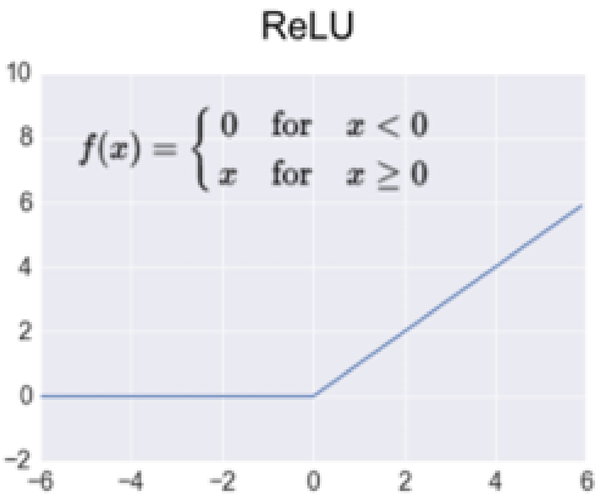
<u>Easy!</u>

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

Derivatives of common activations are easy!







$\sigma'(z) = \sigma(z)\big(1 - \sigma(z)\big)$

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

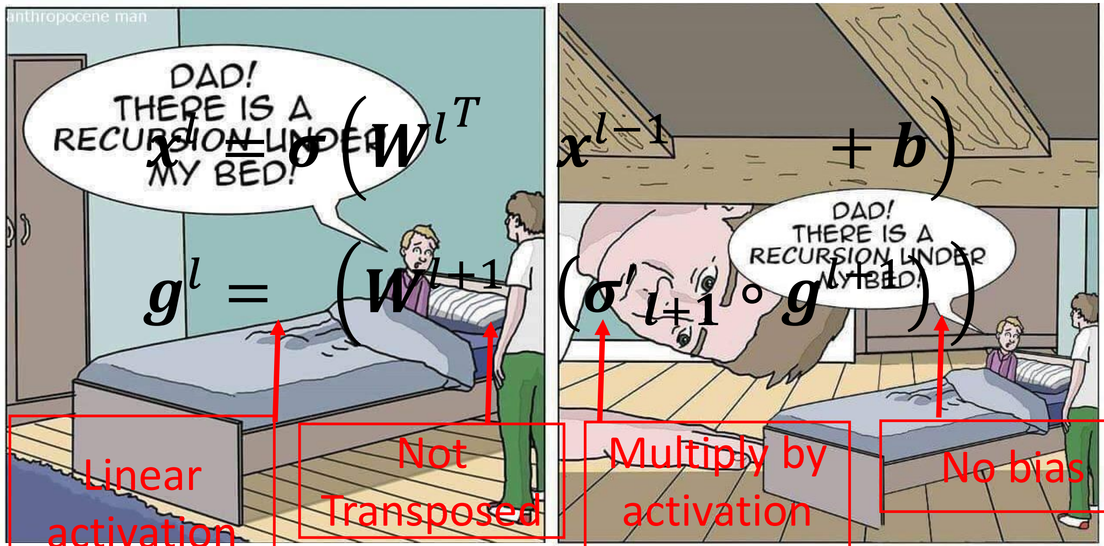
1 for z>0, 0 otherwise

Back Propagation

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

$$= \sum_k \underbrace{\boxed{\frac{\partial \mathcal{L}}{\partial x_k^{l+1}}}}_{g_k^{l+1}} \cdot \underbrace{\boxed{\frac{\partial x_k^{l+1}}{\partial x_j^l}}}_{w_{jk}^{l+1} \cdot \sigma'\left(z_k^{l+1}\right)}$$

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



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

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

Linear activation

Not Transposed

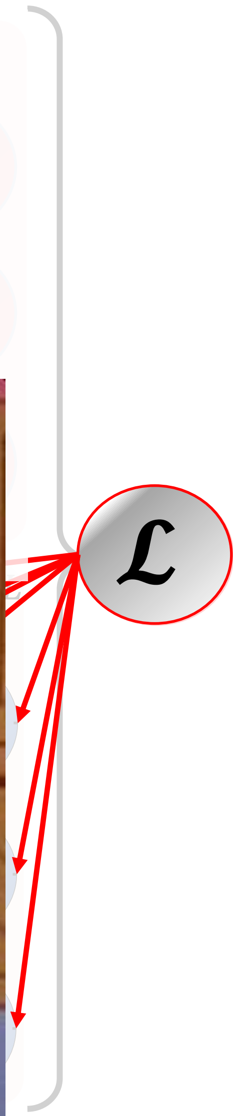
Multiply by activation derivative

No bias

Stopping criterion:

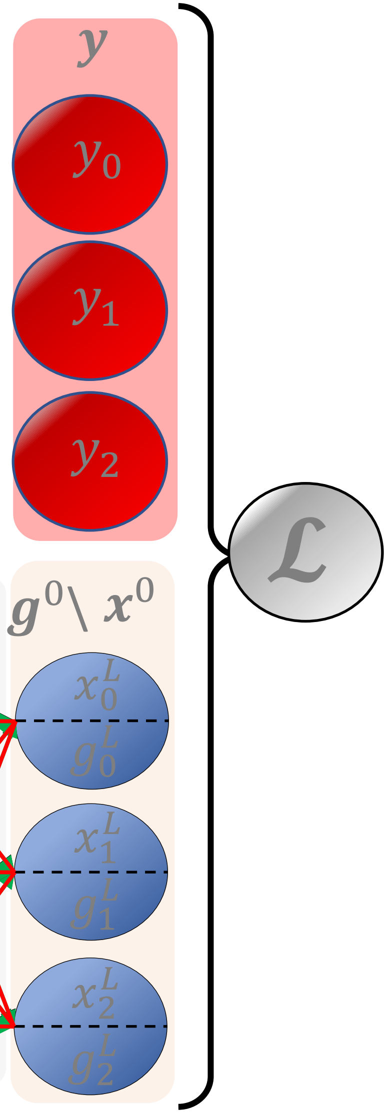
$$x_j^l = \sigma\left(\underbrace{\sum_i w_{ij}^l \cdot x_i^{l-1} + b_j}_{z_j^l}\right)$$

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

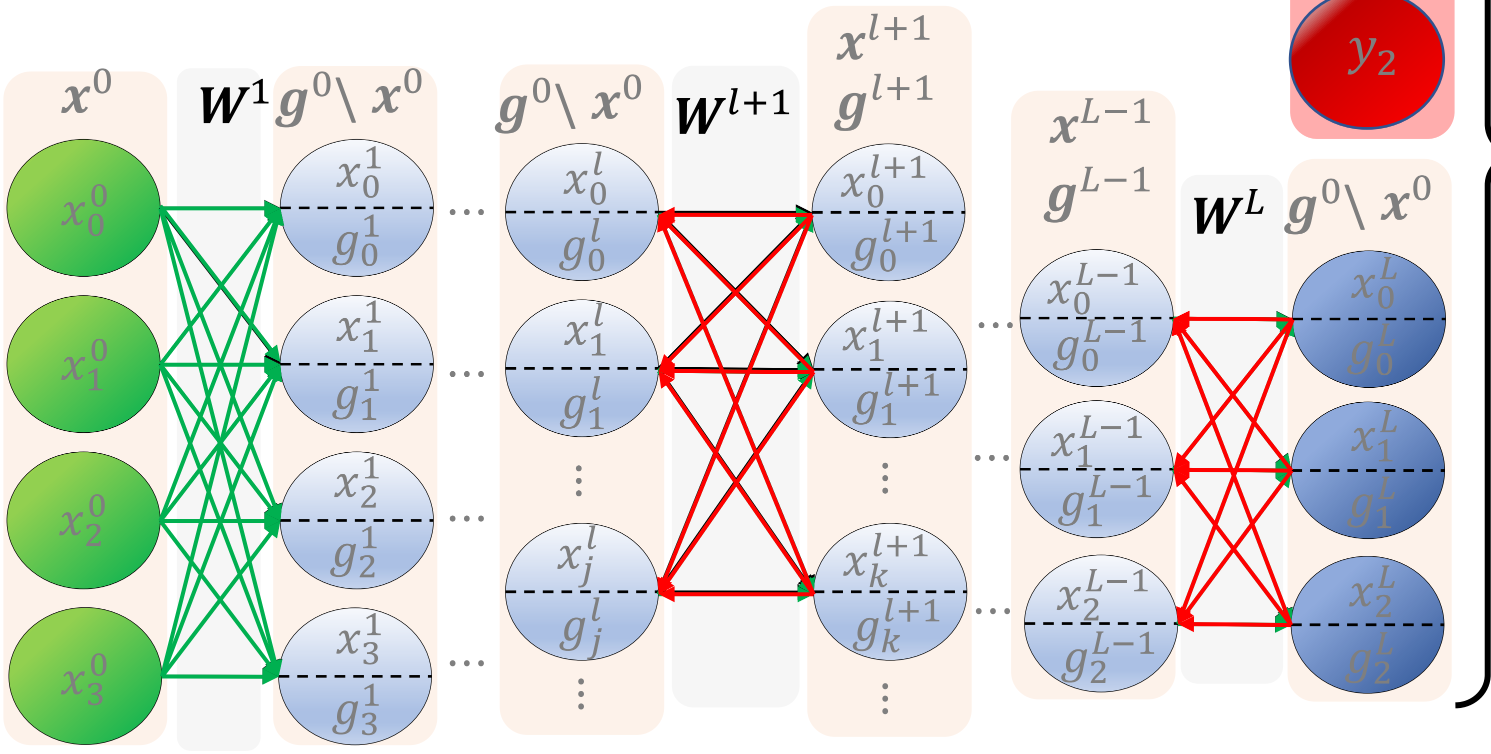


- Initialize weights
- Repeat until convergence:

1. Sample a batch from the data: $\{(\boldsymbol{x_i}, \boldsymbol{y_i}) \ldots\}$
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: $\boldsymbol{g}^L = \frac{\partial \mathcal{L}}{\partial x^L}$, $\boldsymbol{g}^l = \left(\boldsymbol{W}^{l+1}(\boldsymbol{\sigma'}_{\boldsymbol{l+1}} \circ \boldsymbol{g}^{l+1})\right)$
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: $\boldsymbol{W}^l := \boldsymbol{W}^l - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l}$, $\boldsymbol{b}^l := \boldsymbol{b}^l - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l}$



Vanilla Network Back Propagation



Let’s get more generic

Yann LeCun

January 5, 2018 ·

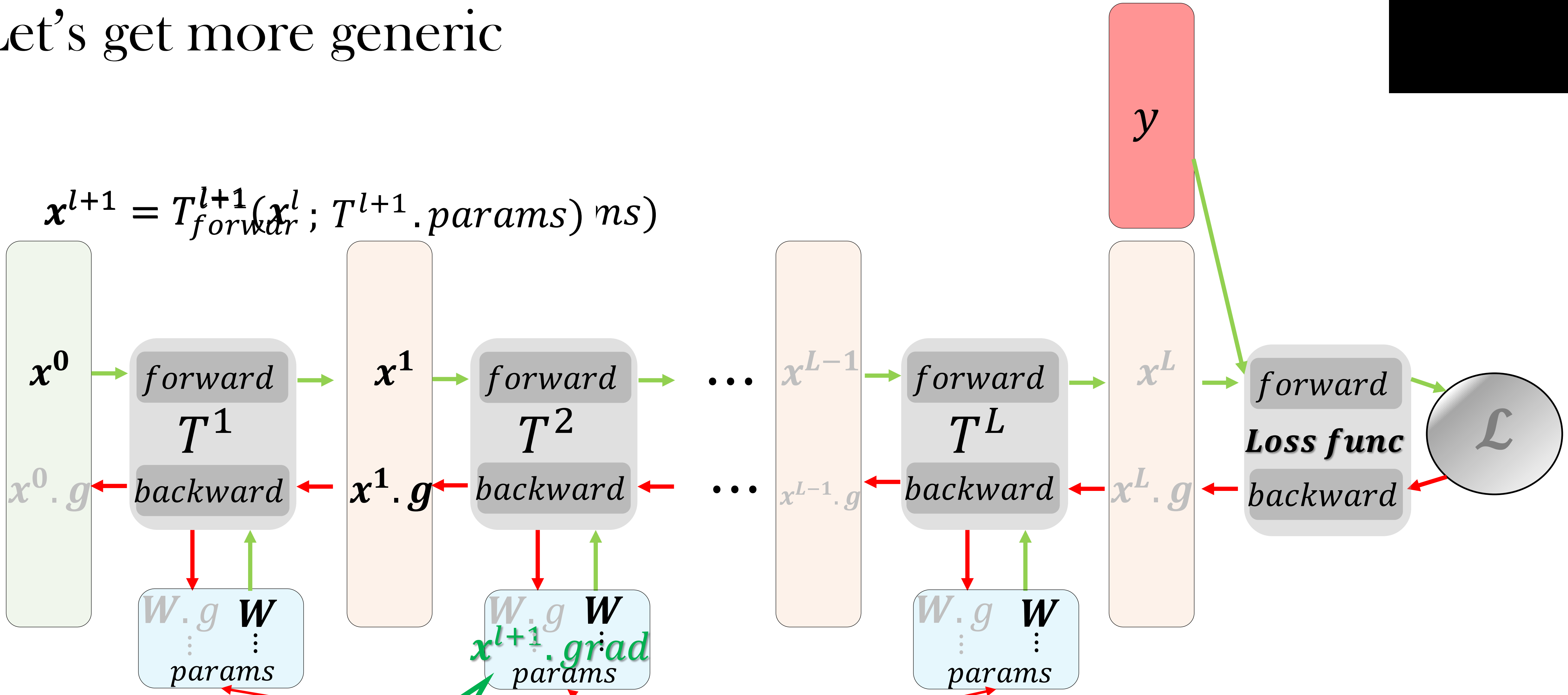
OK, Deep Learning has outlived its usefulness as a buzz-phrase. Deep Learning est mort. Vive Differentiable Programming!

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

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

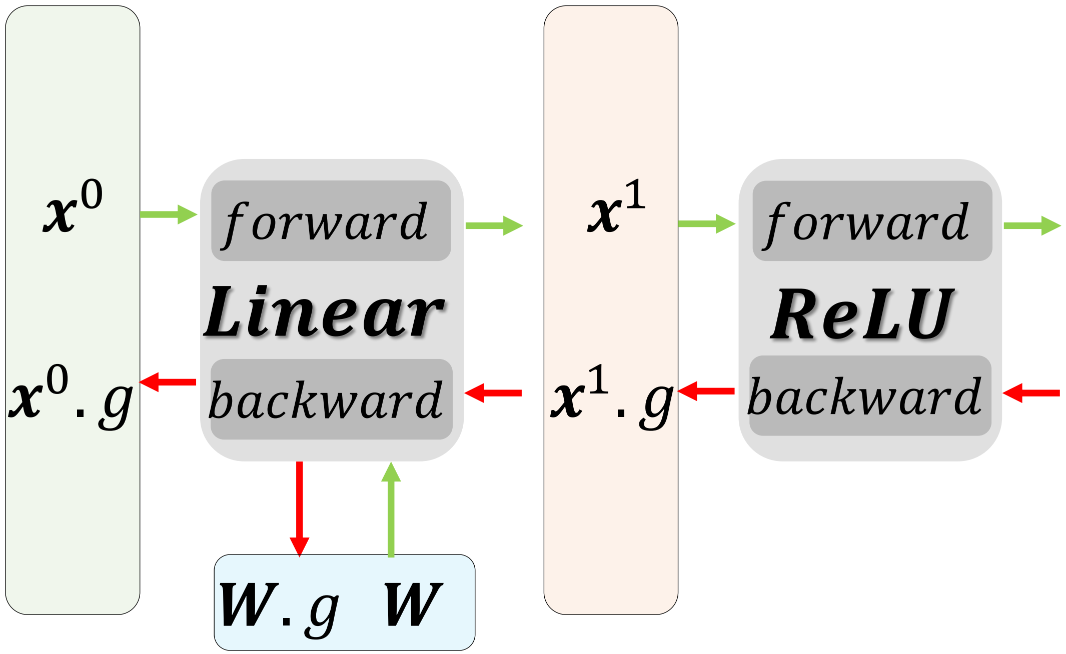


$$x^l.grad = \frac{d\mathcal{L}}{dx^l} = \frac{d\mathcal{L}}{dx^{l+1}} \cdot \frac{dx^{l+1}}{dx^l} = T^{l+1}_{backward}\left(x^{l+1}.grad;\quad T^{l+1}.params,\quad x^l\right)$$

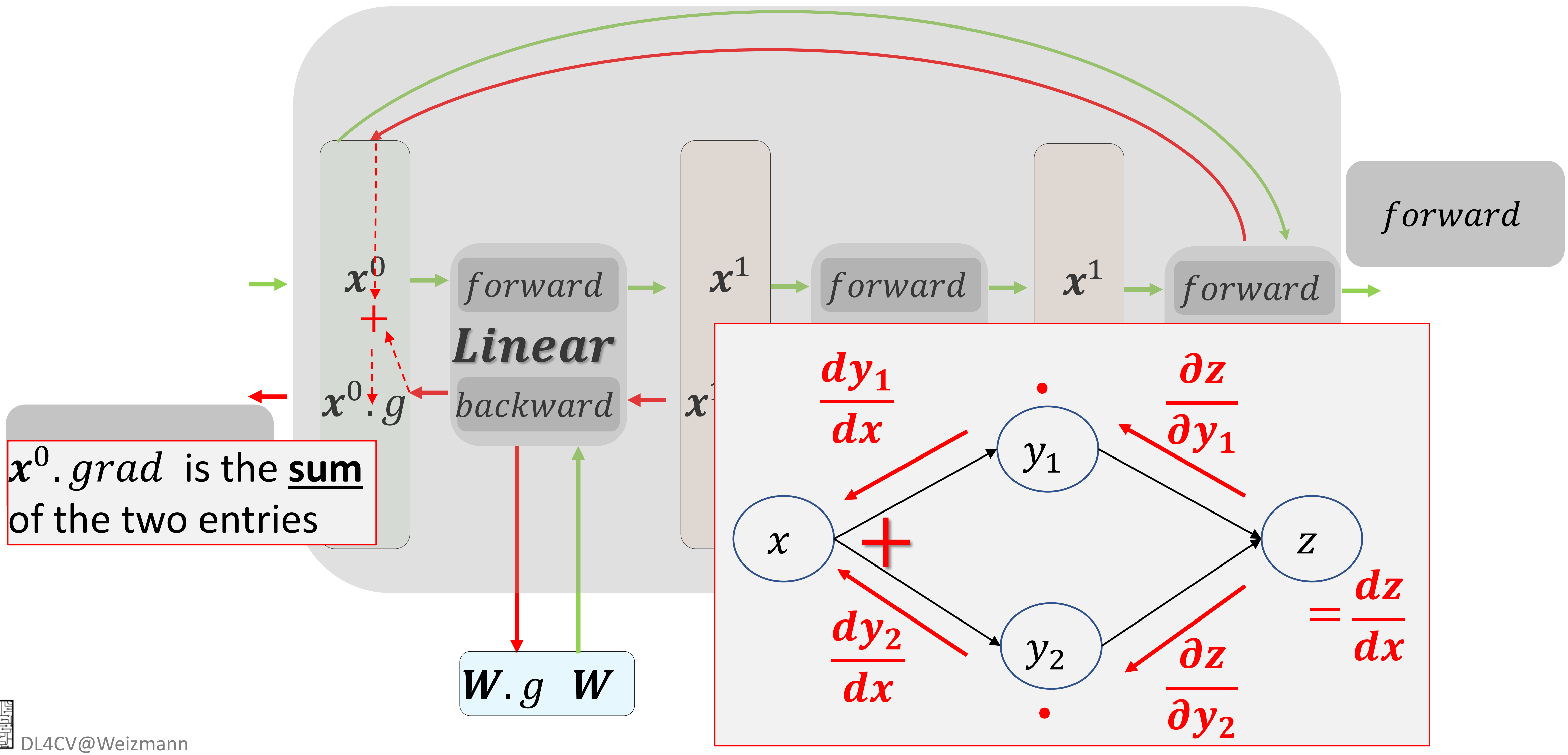
$$T^{l+1}.params.grad = \left(\ldots\right)$$

Use to update weights:
$$W := W - \alpha W.g$$

Example: Standard layer



BTW : You can backprop any DAG!
BTW2: Layers (NN modules) can be nested!



Yes you should understand b

Be creative,
but always watch your back(prop)!

…

http://playground.tensorflow.org

This week's tutorial:

Intro to PyTorch

Dana Joffe

Next week's lecture:

(Me
Again ☹)

Convolutional Neural Networks