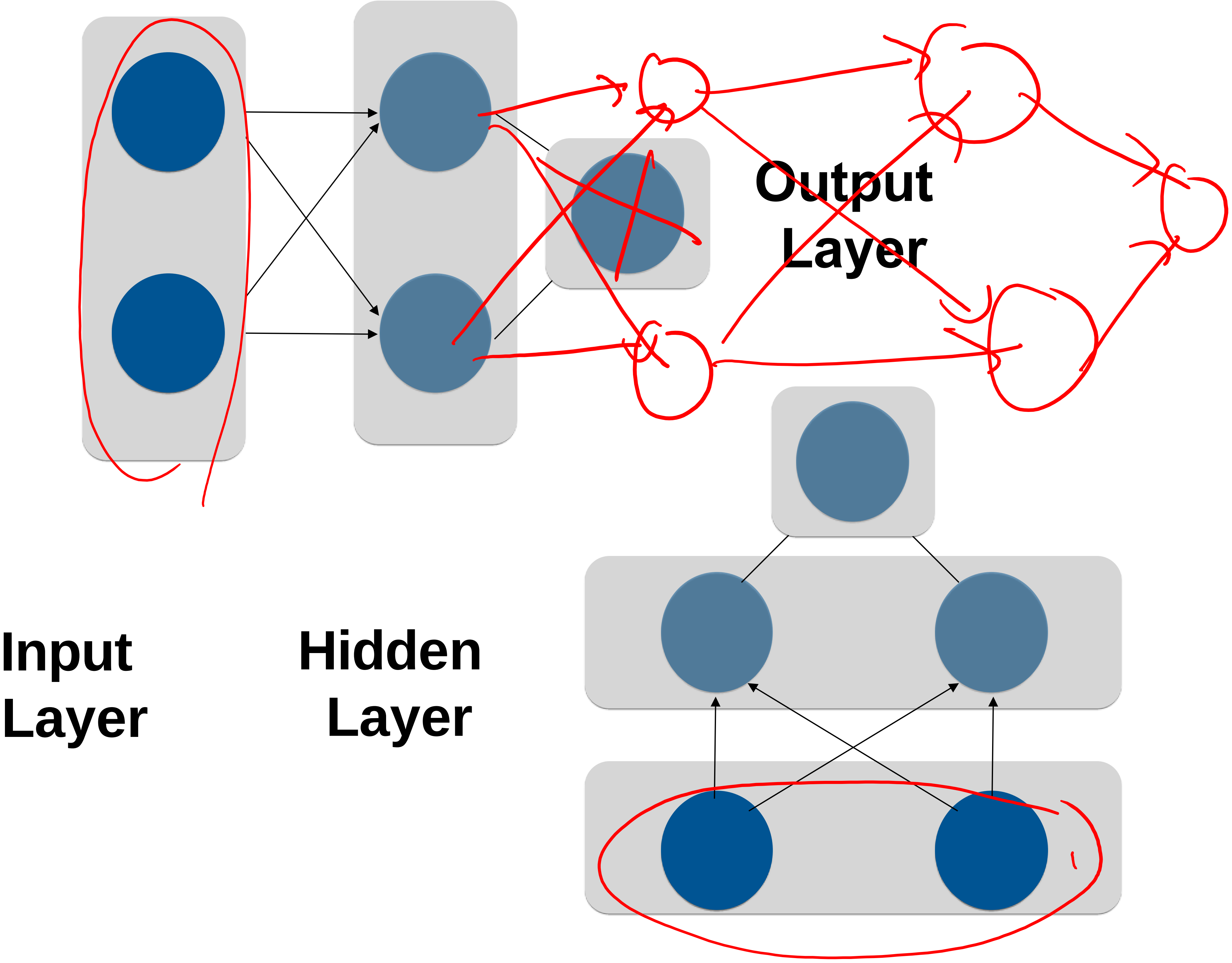
Introduction to Machine Learning (CS419M)

Lecture 17: Introduction to Neural Networks

Original author: Preethi Jyothi
Modified by: Sunita Sarawagi

Mar 16, 2018

Feed-forward Neural Network



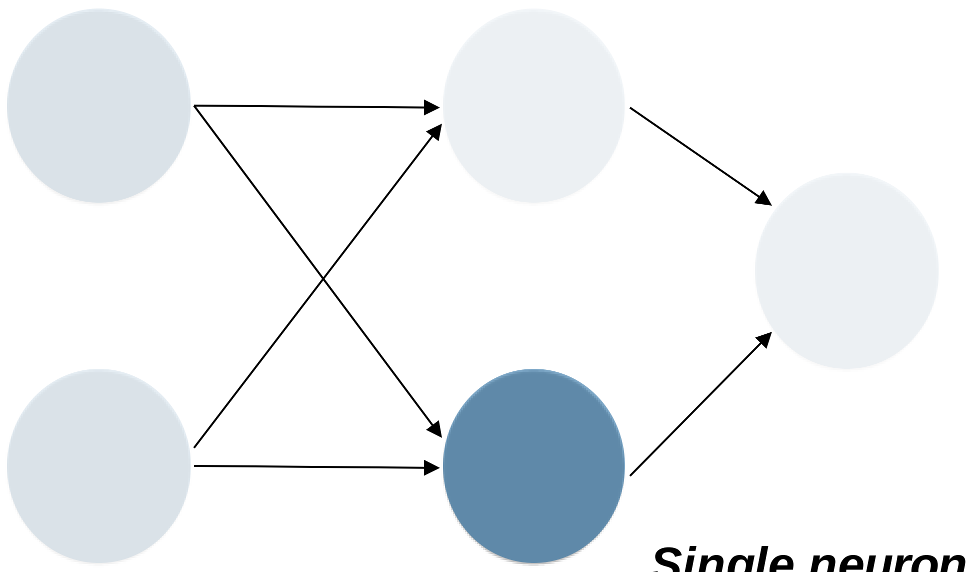
Output Layer

Input Layer

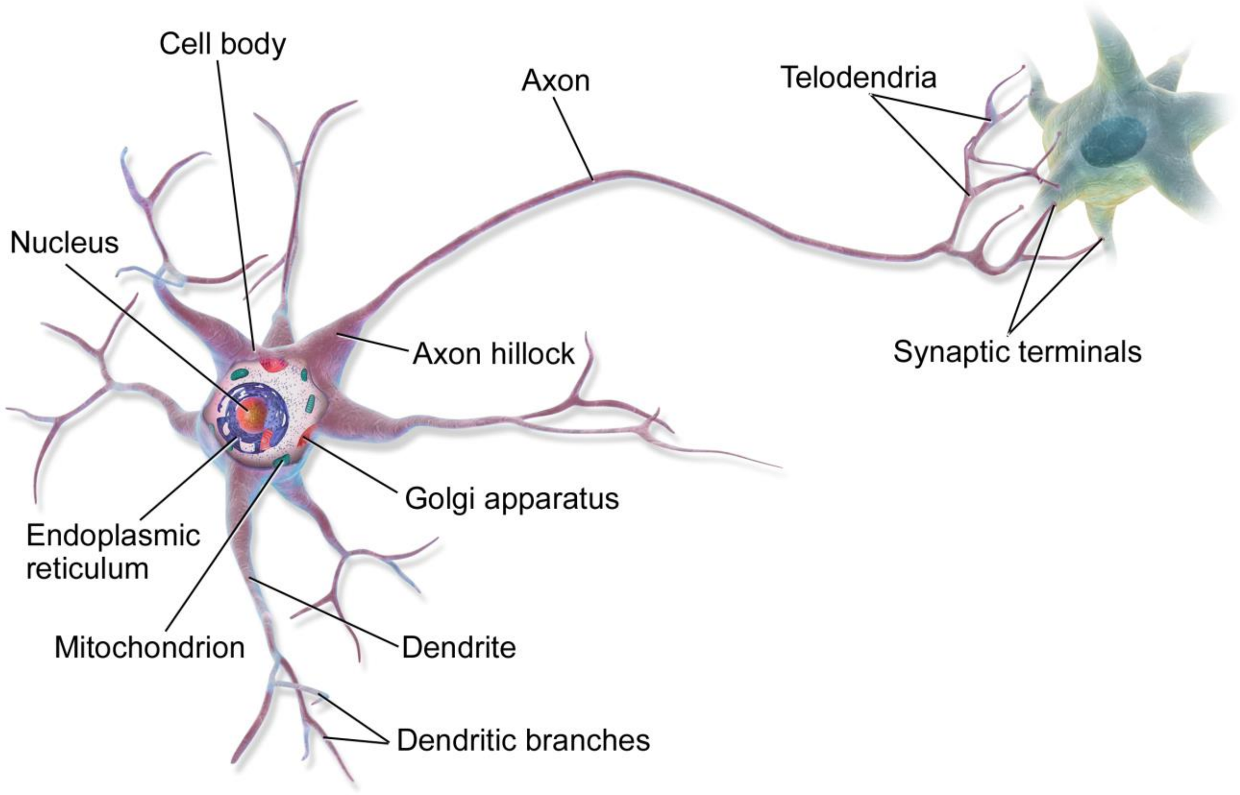
Hidden Layer

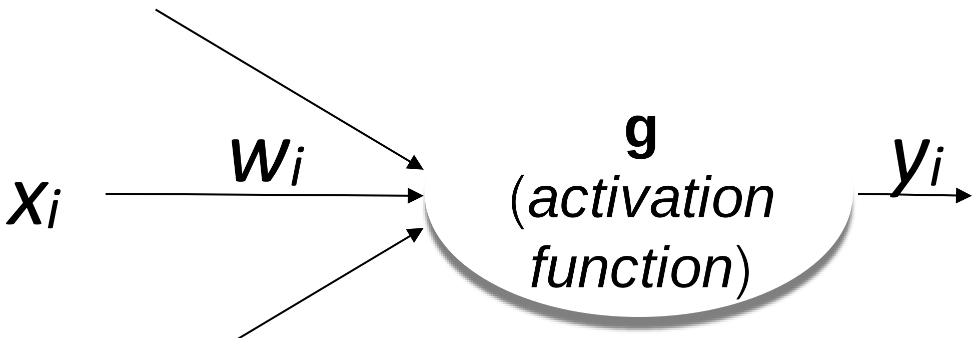
Feed-forward Neural Network

Brain Metaphor



Single neuron



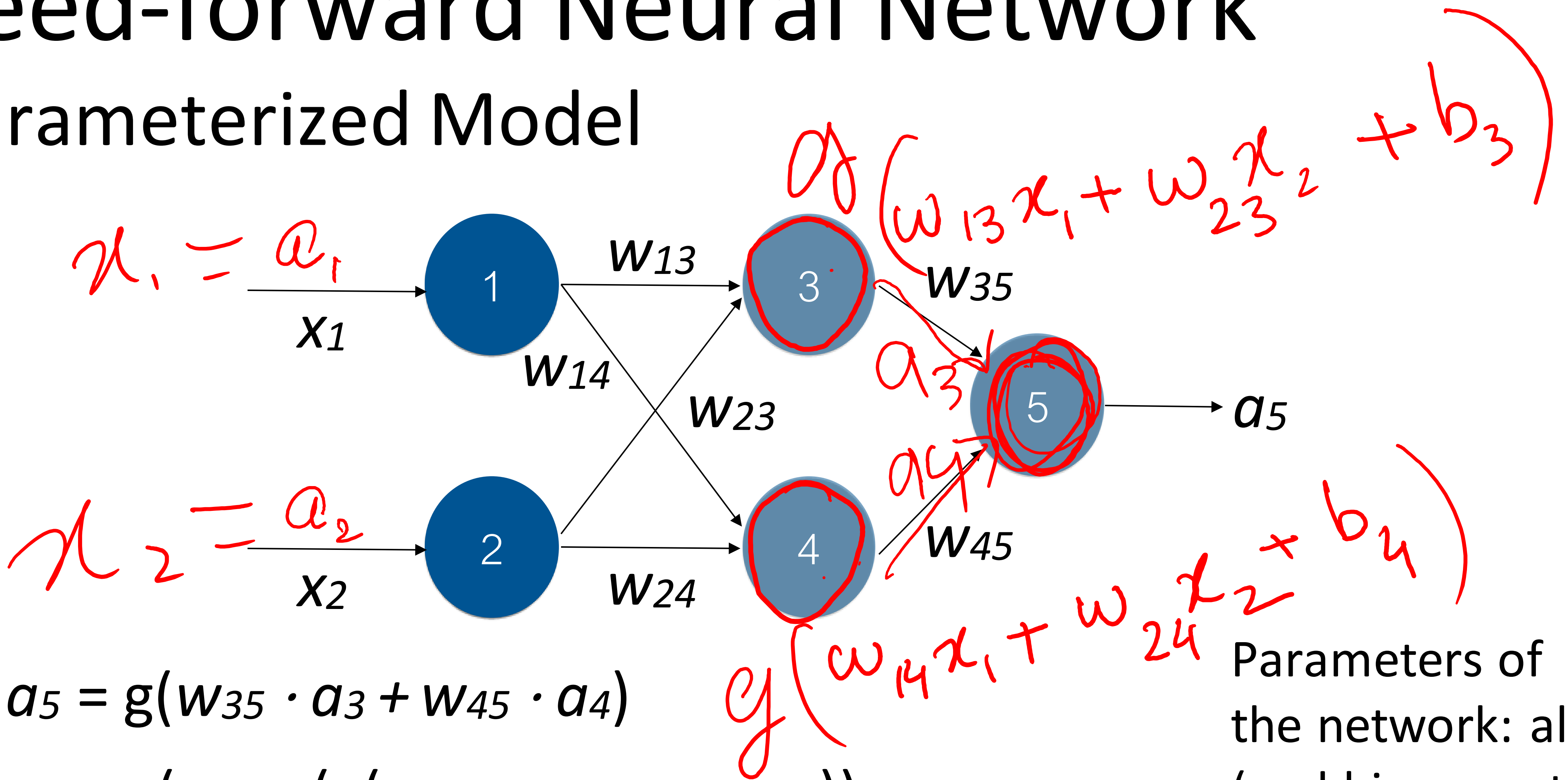


$$y_i = g(\Sigma_i\, w_i \cdot x_i)$$

Image from: https://upload.wikimedia.org/wikipedia/commons/1/10/Blausen_0657_MultipolarNeuron.png

Feed-forward Neural Network

Parameterized Model



$$a_5 = g(w_{35} \cdot a_3 + w_{45} \cdot a_4)$$

$$= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))$$

Parameters of the network: all w_{ij} (and biases not shown here)

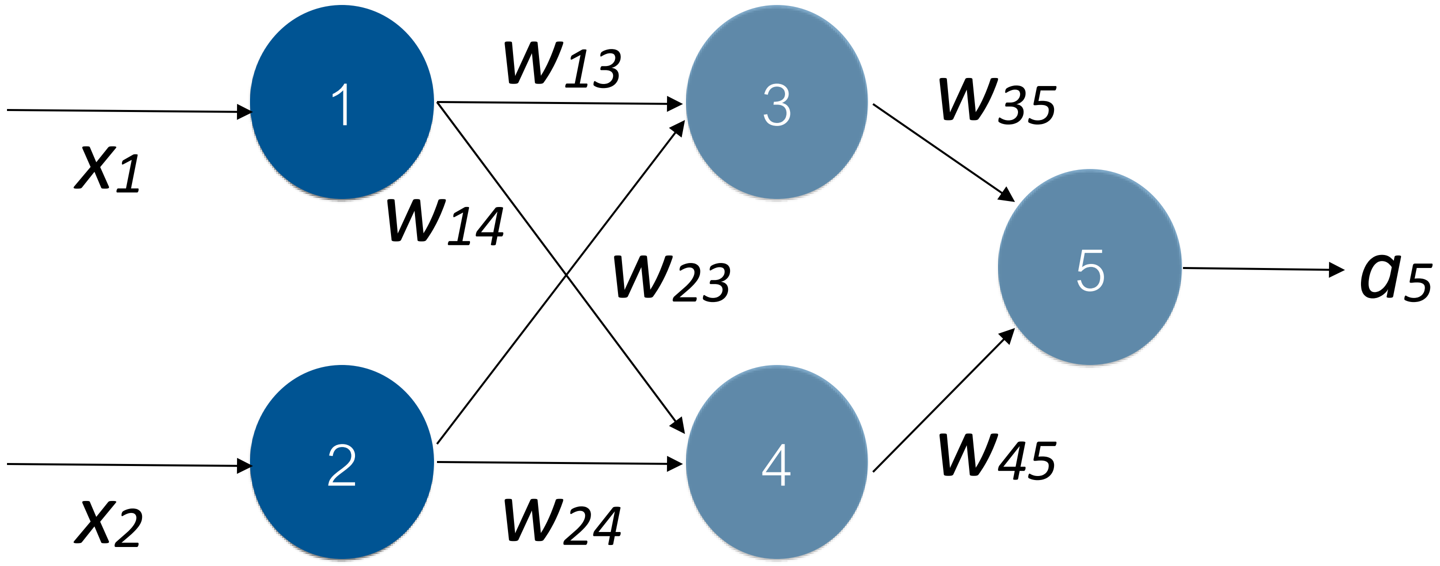
If **x** is a 2-dimensional vector and the layer above it is a 2-dimensional vector **h**, a fully-connected layer is associated with:

$$\mathbf{h} = \mathbf{x}\mathbf{W} + \mathbf{b}$$

where w_{ij} in **W** is the weight of the connection between i^{th} neuron in the input row and j^{th} neuron in the first hidden layer and **b** is the bias vector

Feed-forward Neural Network

Parameterized Model



$$\begin{aligned} a_5 &= g(w_{35} \cdot a_3 + w_{45} \cdot a_4) \\ &= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + \\ &\qquad w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2))) \end{aligned}$$

The simplest neural network is the perceptron:

$$\text{Perceptron}(x) = \mathbf{x}\mathbf{W} + \mathbf{b}$$

A 1-layer feedforward neural network has the form:

$$\text{MLP}(x) = g(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\, \mathbf{W}_2 + \mathbf{b}_2 \longleftarrow$$

Activation Functions (g)

- Cannot be linear: Will get back a linear classifier otherwise: Show.

$$\rightarrow g(xW_1 + b_1)W_2 + b_2$$

$$(xW_1 + b_1)W_2 + b_2$$

$$\begin{bmatrix} w_{11}x_1 + w_{13}x_2 \\ w_{12}x_1 + w_{14}x_2 \end{bmatrix}^T \begin{bmatrix} W_{21} \\ W_{22} \end{bmatrix}$$

$$= \boxed{w_{11}W_{21}}\,\underline{x_1} + \boxed{w_{13}W_{21}}\,\underline{x_2} + \underline{\quad\quad} \quad \underline{\quad\quad}$$

$g(z) = z$

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T$

$b_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$b_2 = 0$

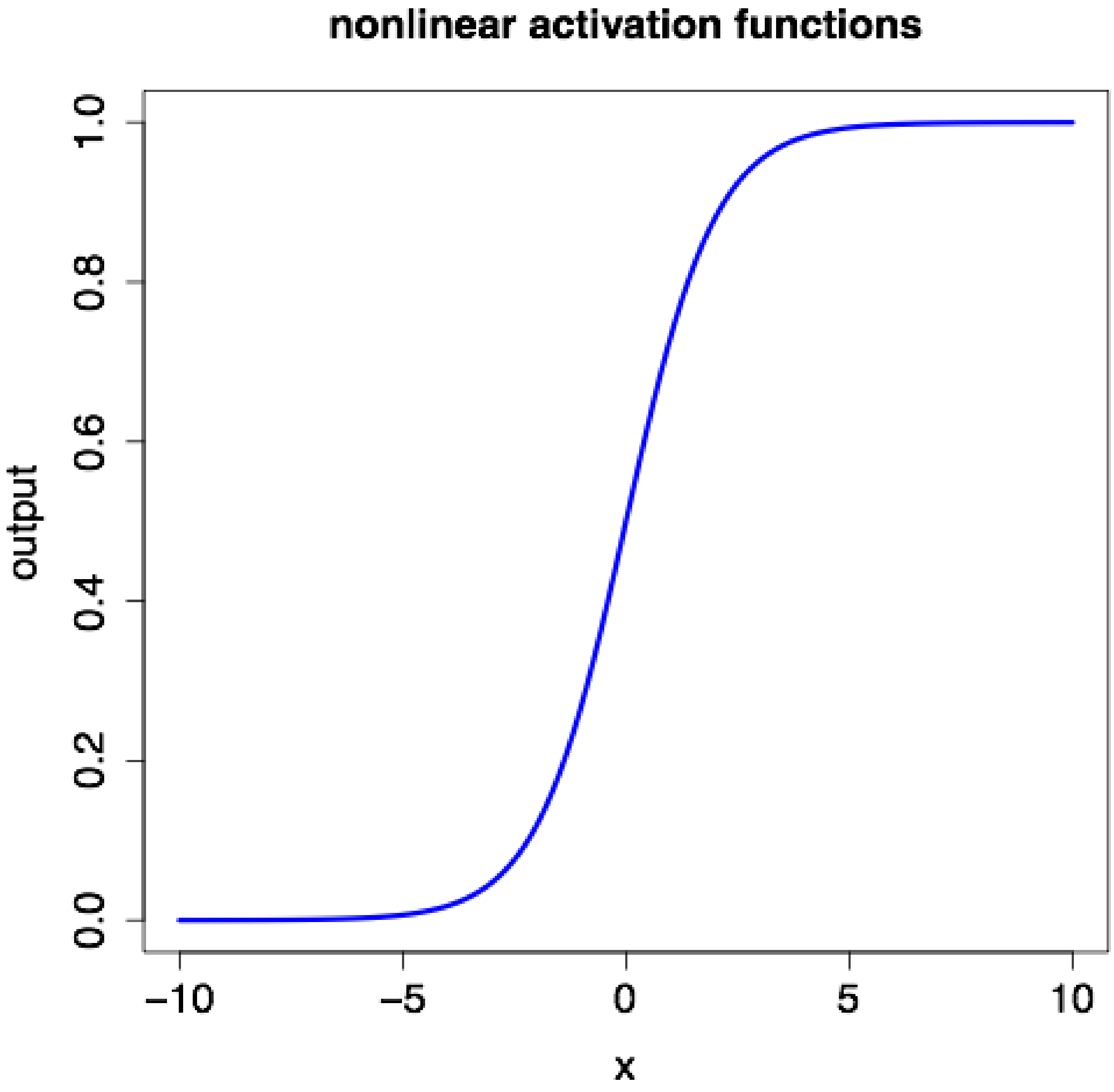
$W_2 = \begin{bmatrix} w_{21} \\ w_{22} \end{bmatrix}$

$W_1 = \begin{bmatrix} w_{11} & w_{12} \\ w_{13} & w_{14} \end{bmatrix}$

- Want a function that is efficient to compute, easy to optimizer (informative gradient), almost linear

Common Activation Functions (g)

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



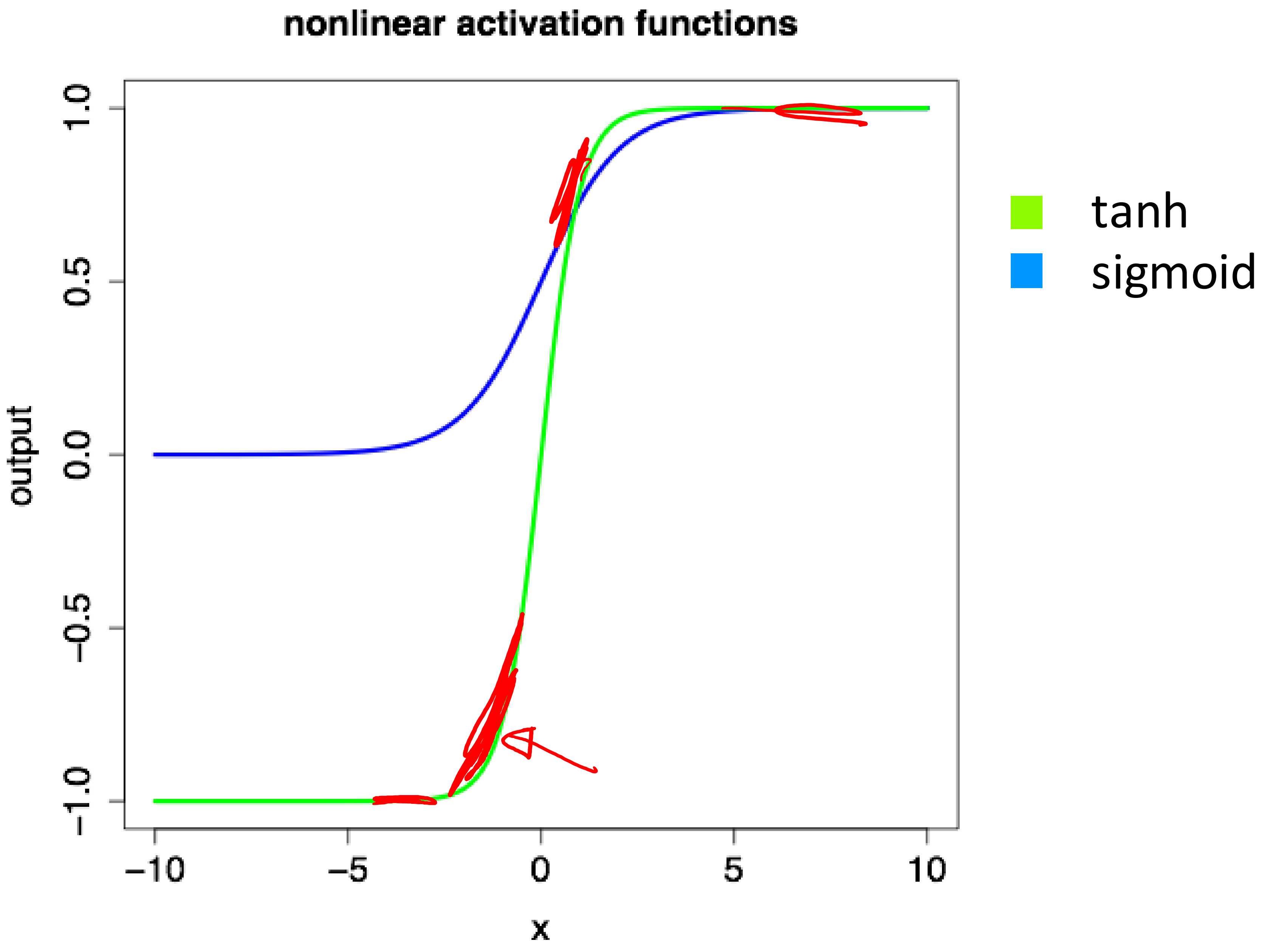
2

🟦 sigmoid

Common Activation Functions (g)

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

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

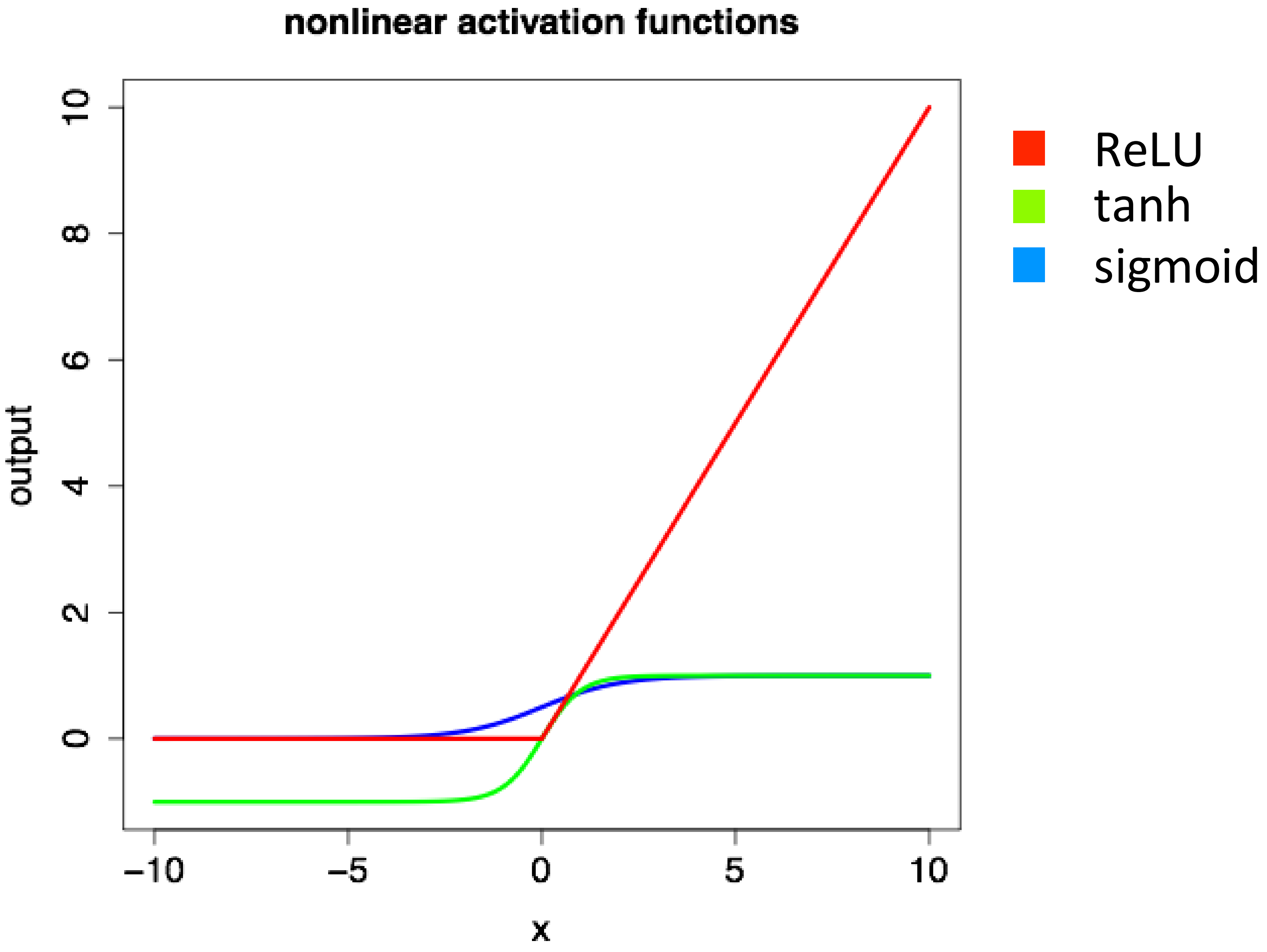


Common Activation Functions (g)

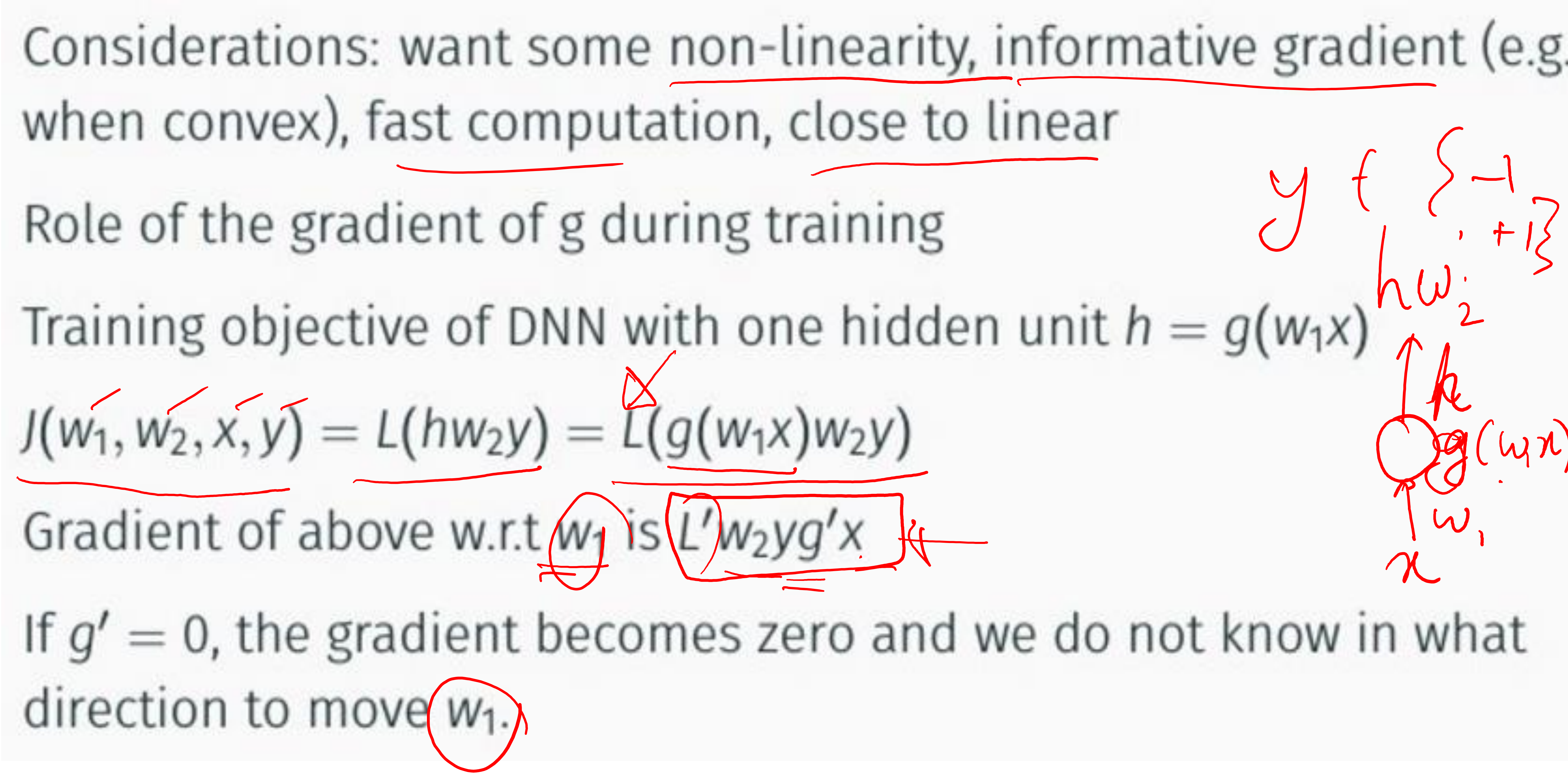
Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

Rectified Linear Unit (ReLU): $\mathrm{RELU}(x) = \max(0, x)$



Choosing g()



Considerations: want some non-linearity, informative gradient (e.g. when convex), fast computation, close to linear

Role of the gradient of g during training

Training objective of DNN with one hidden unit $h = g(w_1x)$

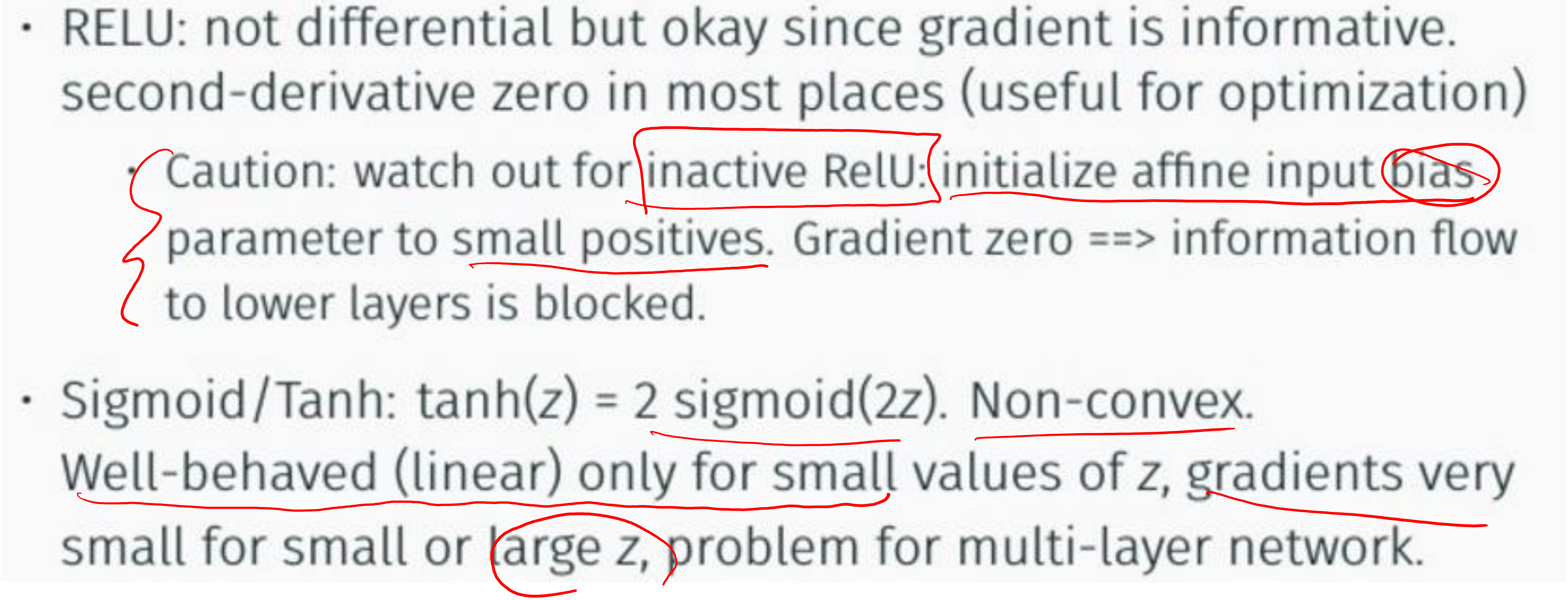
$$J(w_1, w_2, x, y) = L(hw_2y) = L(g(w_1x)w_2y)$$

Gradient of above w.r.t w_1 is $\boxed{L'w_2yg'x}$

If $g' = 0$, the gradient becomes zero and we do not know in what direction to move w_1.

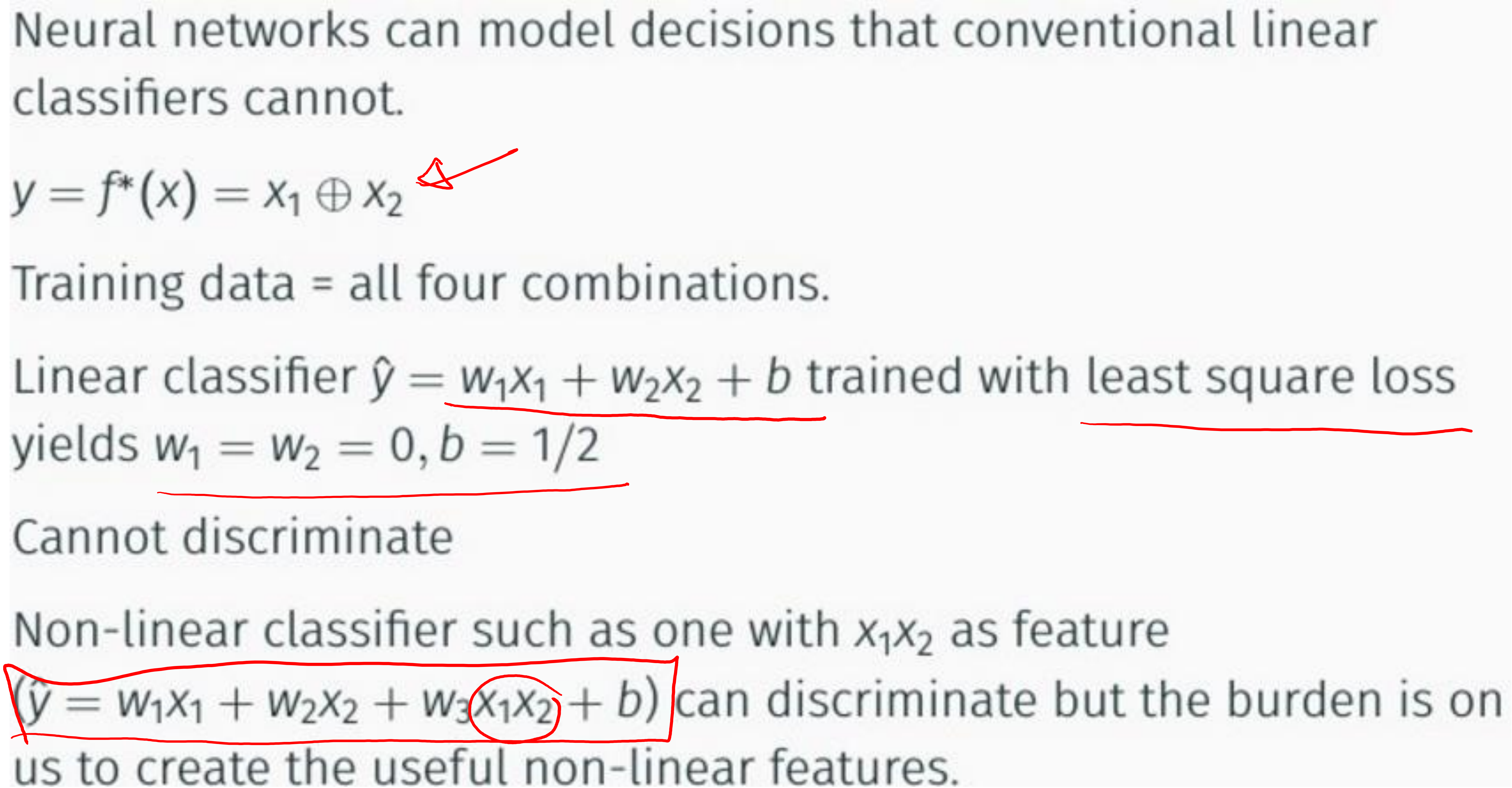
$y \in \{-1, +1\}$

Choosing g()



- RELU: not differential but okay since gradient is informative. second-derivative zero in most places (useful for optimization)
  - Caution: watch out for inactive RelU: initialize affine input bias parameter to small positives. Gradient zero ==> information flow to lower layers is blocked.
- Sigmoid/Tanh: tanh(z) = 2 sigmoid(2z). Non-convex. Well-behaved (linear) only for small values of z, gradients very small for small or large z, problem for multi-layer network.

Example XOR



Neural networks can model decisions that conventional linear classifiers cannot.

$y = f^*(x) = x_1 \oplus x_2$

Training data = all four combinations.

Linear classifier $\hat{y} = \underline{w_1x_1 + w_2x_2 + b}$ trained with least square loss yields $\underline{w_1 = w_2 = 0, b = 1/2}$

Cannot discriminate

Non-linear classifier such as one with x_1x_2 as feature $\boxed{(\hat{y} = w_1x_1 + w_2x_2 + w_3\boxed{x_1x_2} + b)}$ can discriminate but the burden is on us to create the useful non-linear features.

Example: XOR

A generic ~~two~~ one layer neural network with ReLU:

$$y = f(x) = \underbrace{W^2 \max(0, W^1x + b^1)}_{g(W^1x+b^1)} + b^2$$

Role of non-linear transform.

$$W^1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$b^1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}^T, W^2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b^2 = 0$$

$$[h_1, h_2]$$

$$= g\left([x_1, x_2]\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + [0 \ -1]\right)$$

$$\max([x_1+x_2 \quad x_1+x_2-1], [0, 0])$$

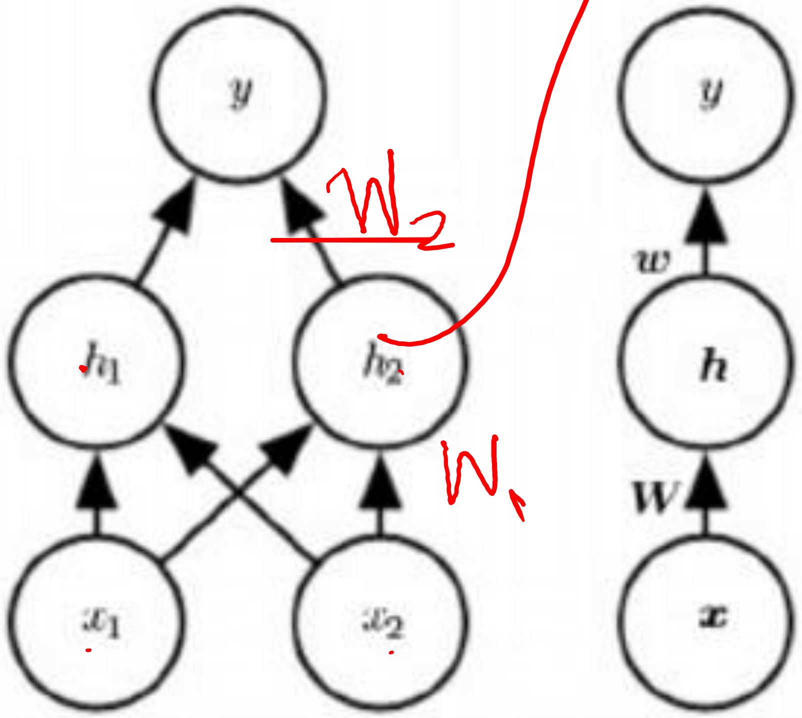
$$[x_1, x_2] = [0, 0]$$

$$\rightarrow [0, 0]$$

$$W^2 h + b^2 = 0$$

$$[x_1, x_2] = [1, 1]$$

$$2 - 2 = 0$$

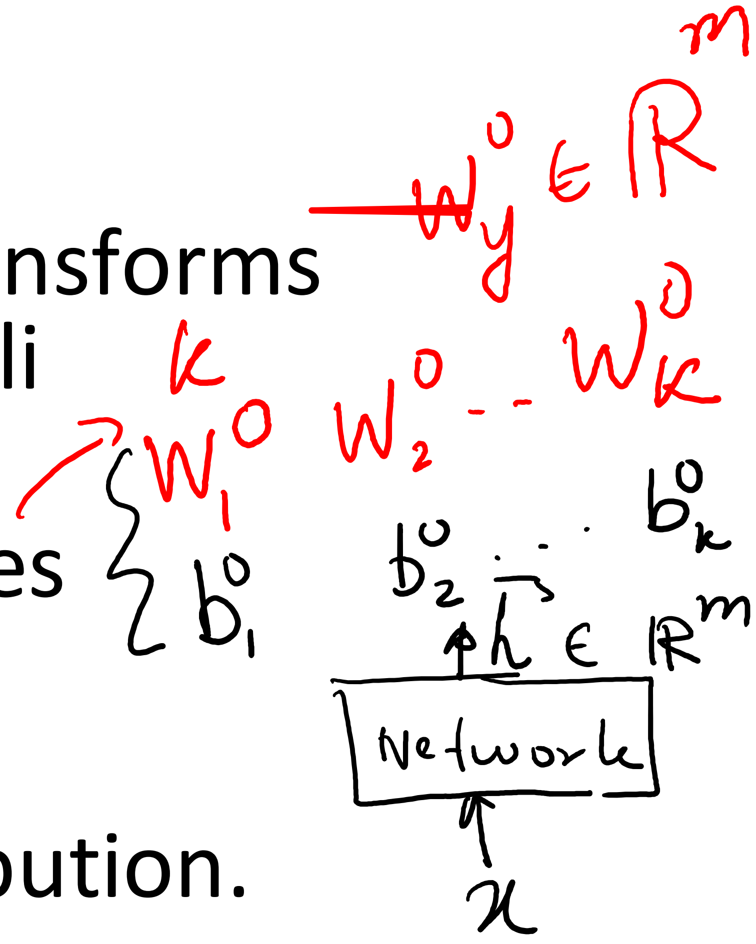


Output layers

- Depends on the output type
  - Binary class labels: sigmoid function transforms arbitrary reals to probability of Bernoulli
  - Multi-class class labels: softmax provides multinomial probabilities softmax –

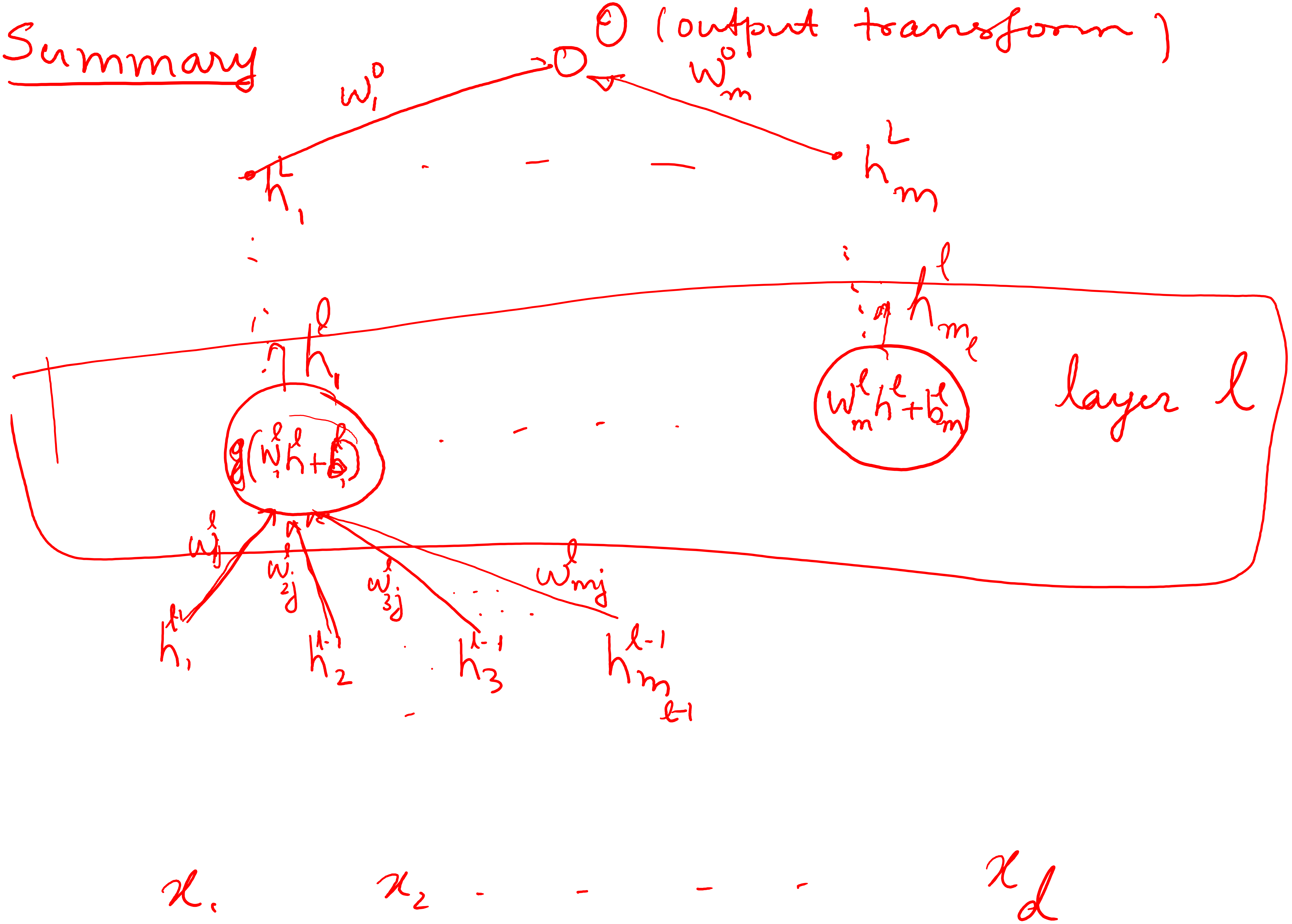
$$P(y|x) = e^{w_y h + b_y} / \sum_{y'} e^{w_{y'} h + b_{y'}}$$

  - Real: output is mean of Gaussian distribution.



- Advantage of all of above: Probability distribution over output. Maximum likelihood training loss is convex in parameters of outer-most layer.
- Similar to conventional training.

Summary


Θ (output transform)
O
w_1^o
w_m^o
h_1^L
h_m^L
h_1^l
$h_{m_l}^l$
$g(w_1^l h^l + b_1^l)$
$w_m^l h^l + b_m^l$
layer l
w_{1j}^l
w_{2j}^l
w_{3j}^l
w_{mj}^l
h_1^{l-1}
h_2^{l-1}
h_3^{l-1}
$h_{m_{l-1}}^{l-1}$
x_1
x_2
x_d

Training a Feed-forward network

- To train a neural network, define a loss function $L(y,\tilde{y})$:
  a function of the true output y and the predicted output $\tilde{y}$

- $L(y,\tilde{y})$ assigns a non-negative numerical score to the neural network's output, $\tilde{y}$

- The parameters of the network are set to minimise L over the training examples (i.e. a sum of losses over different training samples)

- L is typically minimised using a *gradient-based method*

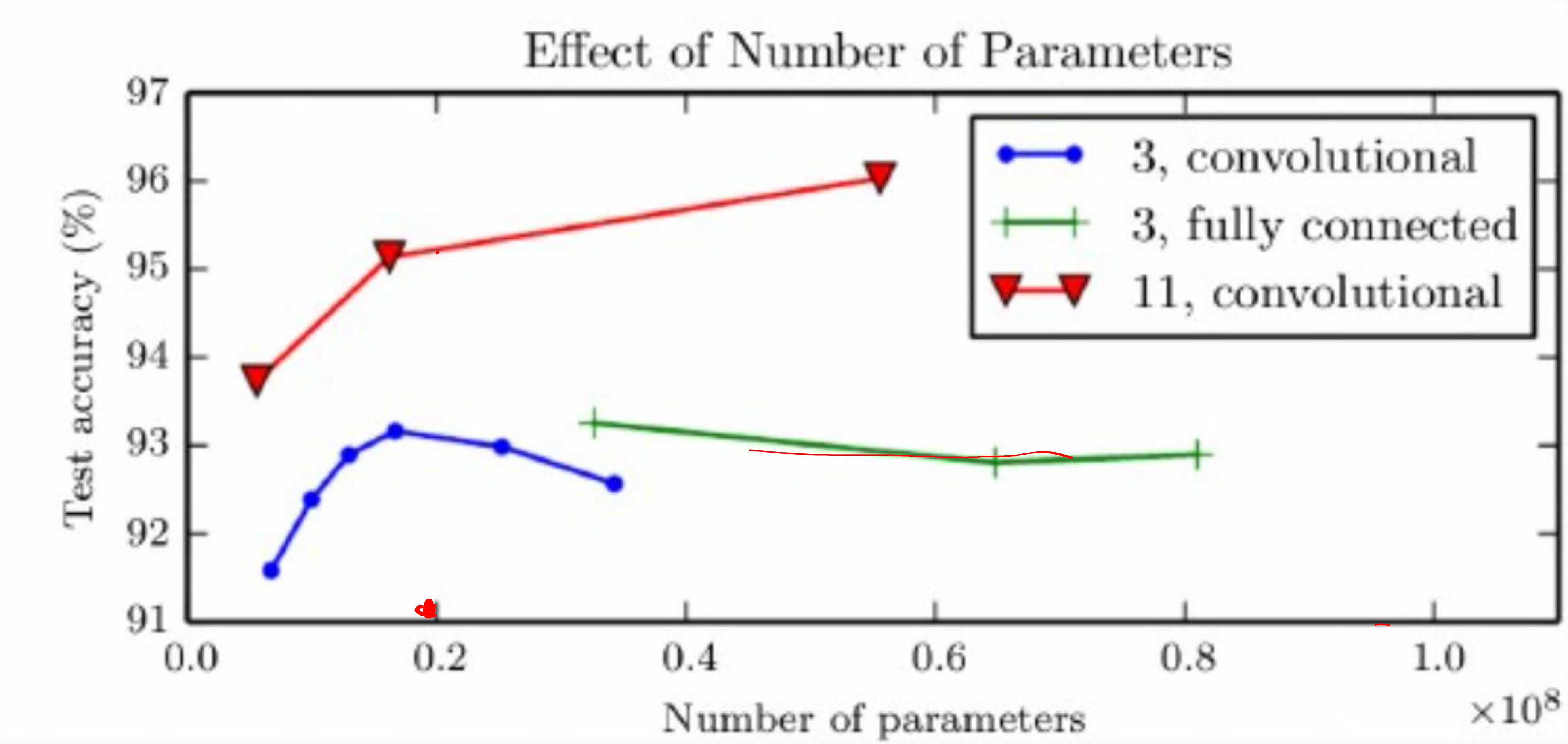
Network Architecture

Choosing the number of layers and width of the network and connection between layer

- **Universal approximation theorem:** A network with one hidden layer (sigmoid type activation) can approximate any continuous function from a closed and bounded set given enough hidden units.
- Proof also extended to work for RELU activations.
- Not useful in practice:
  - number of hidden units required may be exponentially large,
  - the parameters of the network may not be easily learnable: might overfit on a wrong function.

Effect of depth

- Many functions can be efficiently represented with multiple hidden layers but require exponential width with single hidden layer
- The number of linear regions carved out via d inputs, l+1 depth, $m =$ c units per hidden layer is $O\left(C(c, d)^{dl} c^{d}\right)$
- Empirically too, larger depth leads to better generalization and lower error.



Stochastic Gradient Descent (SGD)

SGD Algorithm

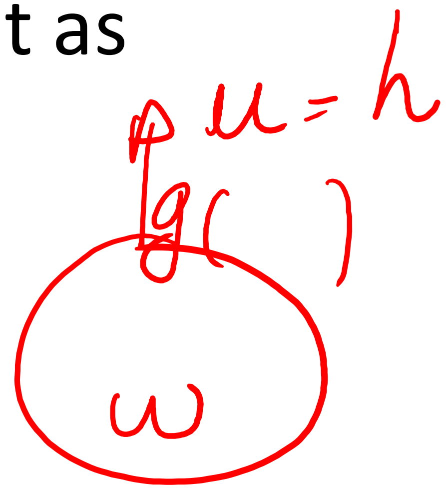
```
Inputs:
    Function NN(x; θ), Training examples, $x_1 … x_n$ and    → N
    outputs, $y_1 … y_n$ and Loss function L.

do until stopping criterion
    Pick a training example $x_i, y_i$    → pick a batch of b ~~examples~~
    Compute the loss $L(NN(x_i; θ), y_i)$
    Compute gradient of L, $\nabla L$ with respect to θ
    $θ \leftarrow θ - η \nabla L$
                  ↑ learning rate
done

Return: θ
```

Training a Neural Network

Define the **Loss function** to be minimised as a node L

Goal: Learn weights for the neural network which minimise L

Gradient Descent: Find $\partial L/\partial w$ for every weight w, and update it as
$w \leftarrow w - \eta\, \partial L/\partial w$



How do we efficiently compute $\partial L/\partial w$ for all w?

Will compute $\partial L/\partial u$ for every node u in the network!

easy local

$\partial L/\partial w = \boxed{\partial L/\partial u} \cdot \boxed{\partial u/\partial w}$ where u is the node which uses w

compute recursively

Training a Neural Network

New goal: compute $\partial L/\partial u$ for every node $\overset{h}{u}$ in the network

Simple algorithm: Backpropagation

Key fact: Chain rule of differentiation

If L can be written as a function of variables $v_1, \ldots, v_n$, which in turn depend (partially) on another variable u, then

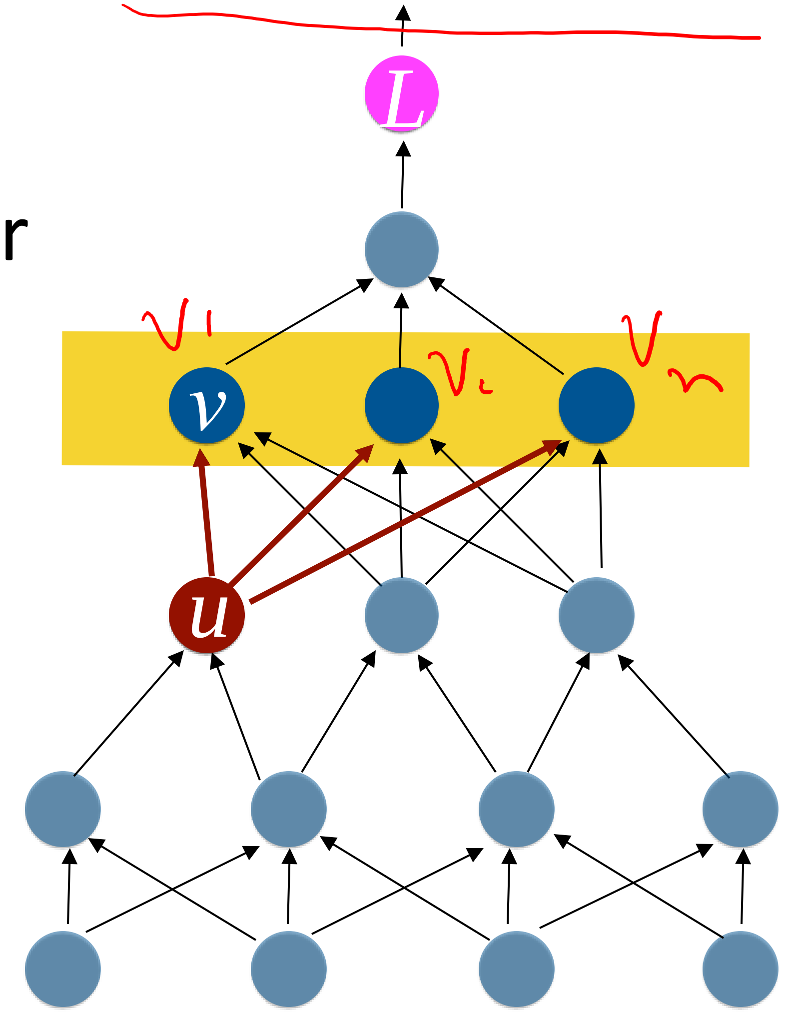
$$\partial L/\partial u = \Sigma_i\, \partial L/\partial v_i \cdot \partial v_i/\partial u$$

Backpropagation

If L can be written as a function of variables $v_1,\ldots, v_n$, which in turn depend (partially) on another variable u, then

$$\partial L/\partial u = \Sigma_i\ \partial L/\partial v_i \cdot \partial v_i/\partial u$$

Consider $v_1,\ldots, v_n$ as the layer above u, $\Gamma(u)$



Then, the chain rule gives

$$\partial L/\partial u = \Sigma_{v\ \in \Gamma(u)}\ \partial L/\partial v \cdot \partial v/\partial u$$

Backpropagation

$$\partial L/\partial u = \underline{\Sigma_{v \in \Gamma(u)} \, \partial L/\partial v \cdot \partial v/\partial u}$$

Backpropagation

Base case: $\underline{\partial L/\partial L = 1}$

For each u (top to bottom):

For each $v \in \Gamma(u)$:

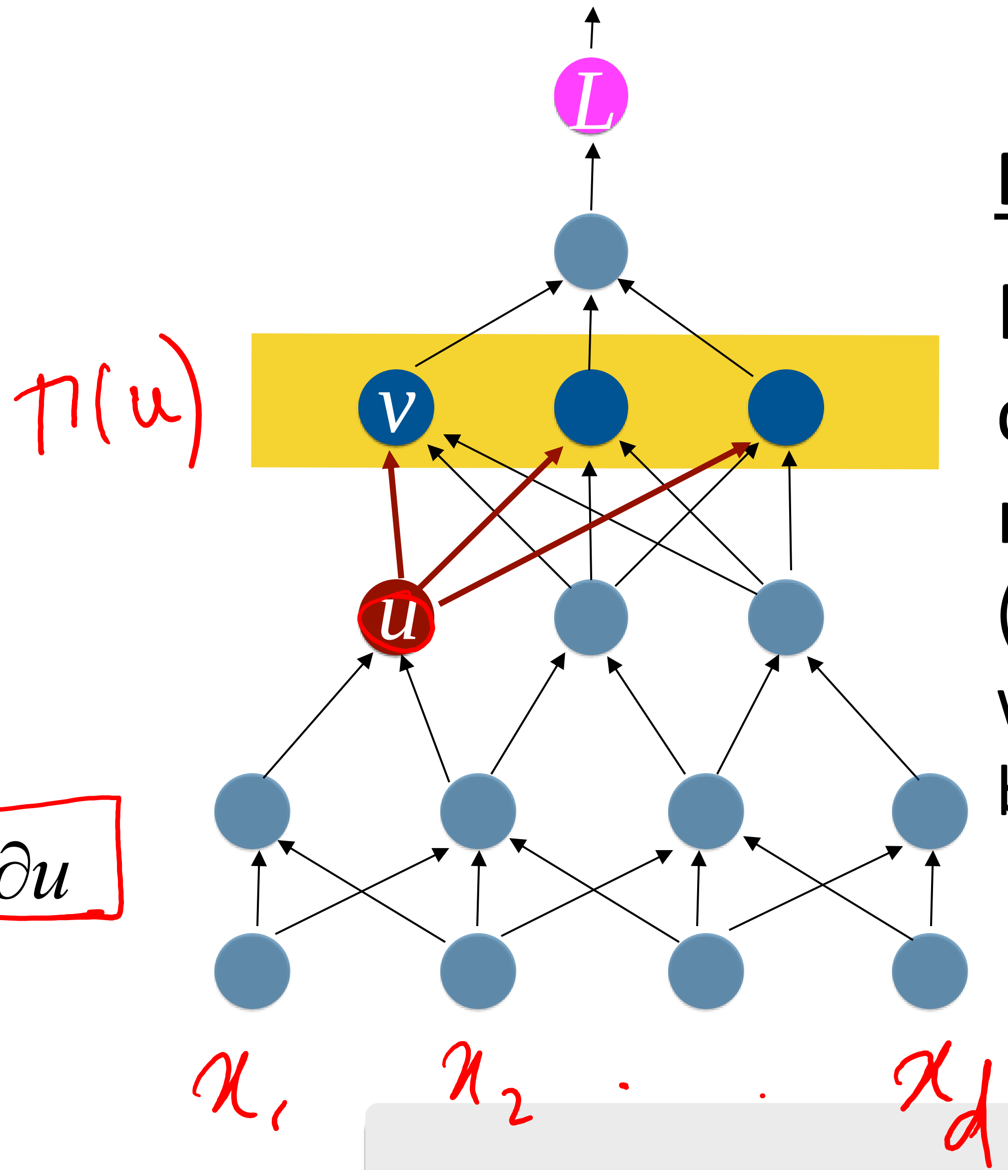
Inductively, have computed $\underline{\partial L/\partial v}$

Directly compute $\boxed{\partial v/\partial u}$

Compute $\underline{\partial L/\partial u}$

Compute $\underline{\partial L/\partial w}$

where $\underline{\partial L/\partial w = \partial L/\partial u \cdot \partial u/\partial w}$

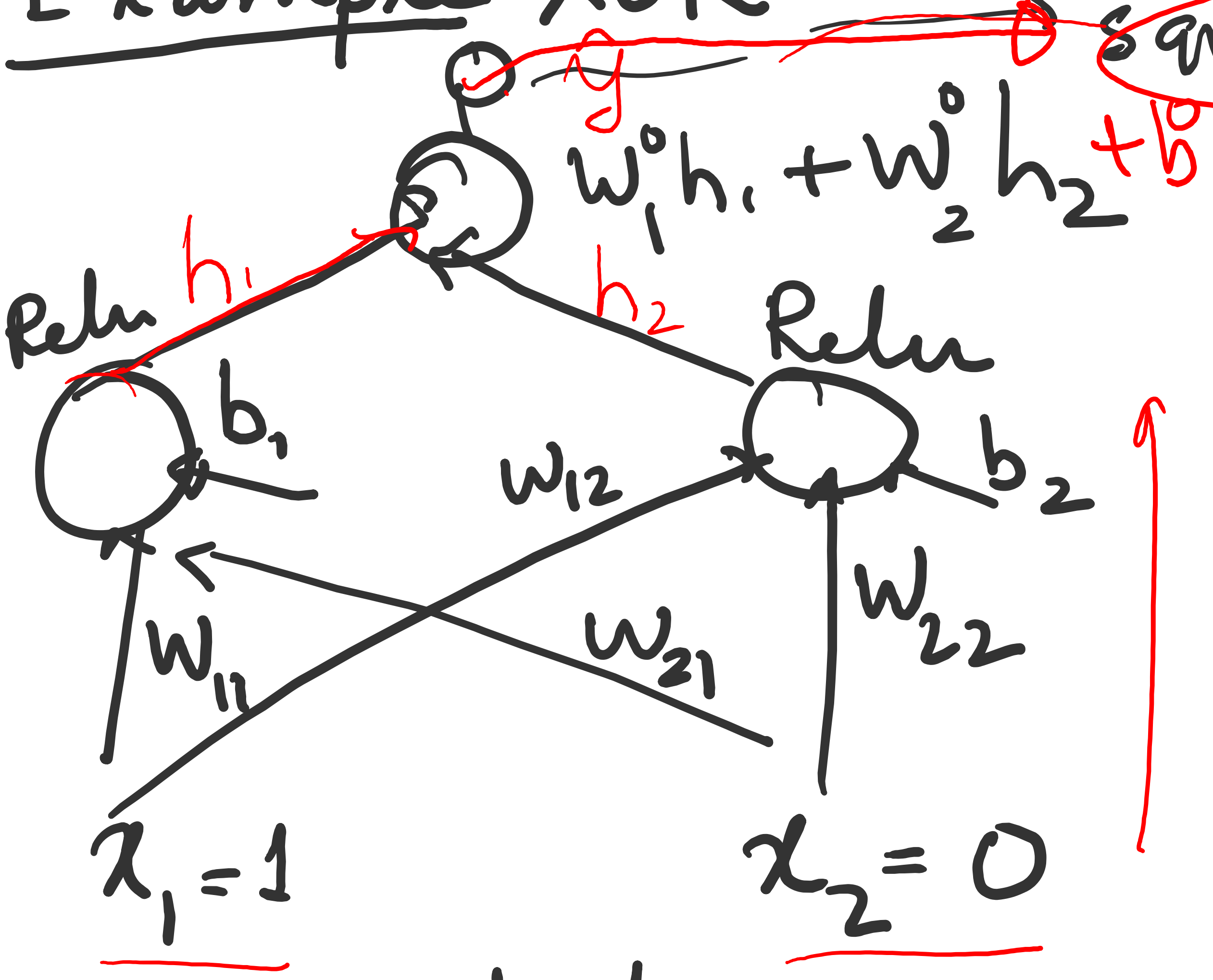


Where values computed in the forward pass are needed

Forward Pass

First, in a forward pass, compute values of all nodes given an input (The values of each node will be needed during backprop)

Example XOR



$y = 1$

$$\frac{\partial L}{\partial w^0_1} = \boxed{\frac{\partial L}{\partial \hat y}} \frac{\partial \hat y}{\partial w^0_1} = -2h_1 = -2$$

$$w^0_1 = 0 - \eta(-2) = 2\eta$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial h_1} \boxed{\frac{\partial h_1}{\partial w_{11}}} = 0 \cdot 1 \cdot x_1$$

Initially: all $W^s = 0$

all $b^s = \cancel{1}\ 0$

$b^0 = 0$, $b_1 = 1$, $b_2 = -1$

Fwd:

$h^1 = 1$, $h^2 = \cancel{1}\ 0$

$\hat y = 0 = w^0_1 h_1 + w^0_2 h_2 + b^0$

$L = \text{loss}: (y - \hat y)^2 = 1.$

Bwd

$$\frac{\partial L}{\partial \hat y} = -\cancel{1}\,2 \qquad \boxed{\frac{\partial L}{\partial w^0_1} \cancel{= -1}\, h_1}$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial \hat y} \frac{\partial \hat y}{\partial h_1} = -\cancel{1}\,2 \cdot w^0_1 = 0$$

$$h_1 = \text{Relu}(w_{11} x_1 + w_{21} x_2 + b_1)$$