Deep Learning

04-Deep Neural Networks

Marco Piastra

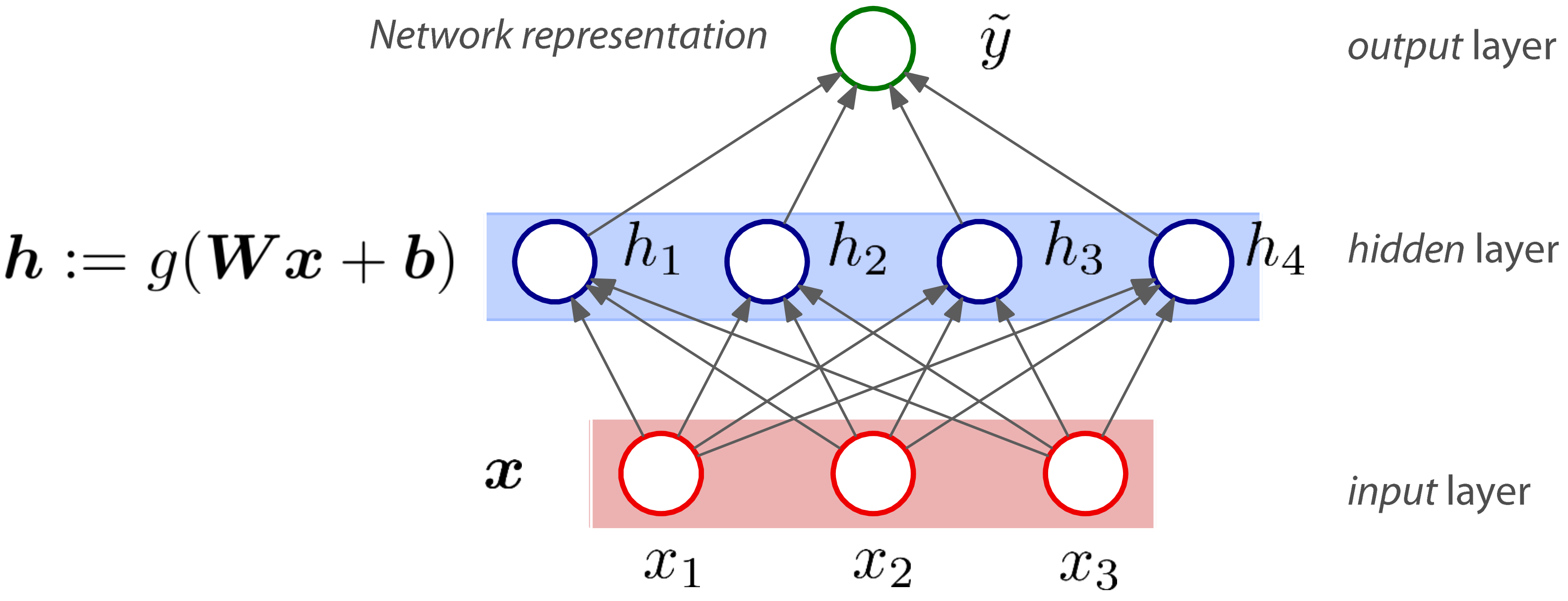
This presentation can be downloaded at:
http://vision.unipv.it/DL

- Approximating a target function

$$y = f^*(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d$$

Universal approximator: ***feed-forward neural network***

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b, \quad \boldsymbol{W} \in \mathbb{R}^{h \times d}, \ \boldsymbol{w}, \boldsymbol{b} \in \mathbb{R}^h, b \in \mathbb{R}$$



Feed-Forward Neural Network

- Approximating a target function

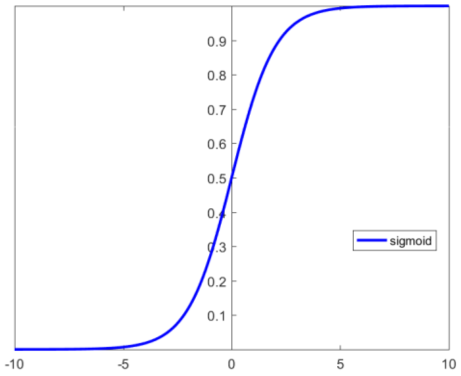
$$y = f^*(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d$$

Universal approximator: ***feed-forward neural network***

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b, \quad \boldsymbol{W} \in \mathbb{R}^{h \times d}, \ \boldsymbol{w}, \boldsymbol{b} \in \mathbb{R}^h, b \in \mathbb{R}$$

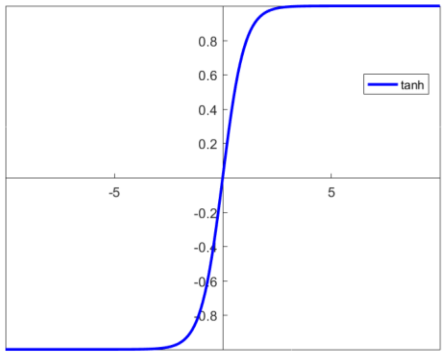
Popular choices for the non-linear function:

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



Sigmoid

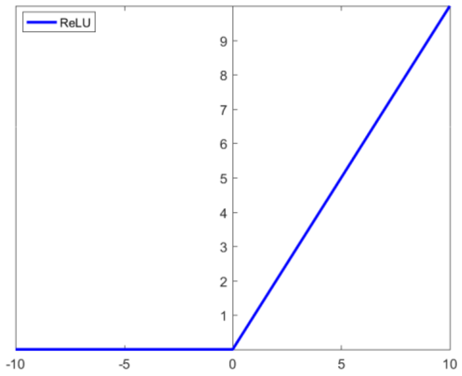
$$g(x) = \tanh(x)$$



Hyperbolic Tangent

this is somewhat special...

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



ReLU

Training Feed-Forward Neural Networks

- ***Stochastic*** **Gradient Descent (SGD)**
  1. Assign initial values to the four parameters $\boldsymbol{W}^{(0)},\ \boldsymbol{b}^{(0)},\ \boldsymbol{w}^{(0)},\ b^{(0)}$
  2. Pick up a data item $(\boldsymbol{x}^{(i)}, y^{(i)})$ from D with uniform probability and update the four parameters (with $\eta \ll 1.0, \quad \eta \to 0$ as iterations progress)

$$\Delta \boldsymbol{W} = -\eta \frac{\partial}{\partial \boldsymbol{W}} L(\tilde{y}^{(i)}, y^{(i)}) \qquad \Delta \boldsymbol{b} = -\eta \frac{\partial}{\partial \boldsymbol{b}} L(\tilde{y}^{(i)}, y^{(i)})$$

$$\Delta \boldsymbol{w} = -\eta \frac{\partial}{\partial \boldsymbol{w}} L(\tilde{y}^{(i)}, y^{(i)}) \qquad \Delta b = -\eta \frac{\partial}{\partial b} L(\tilde{y}^{(i)}, y^{(i)})$$

  3. Unless complete, return to step 2.

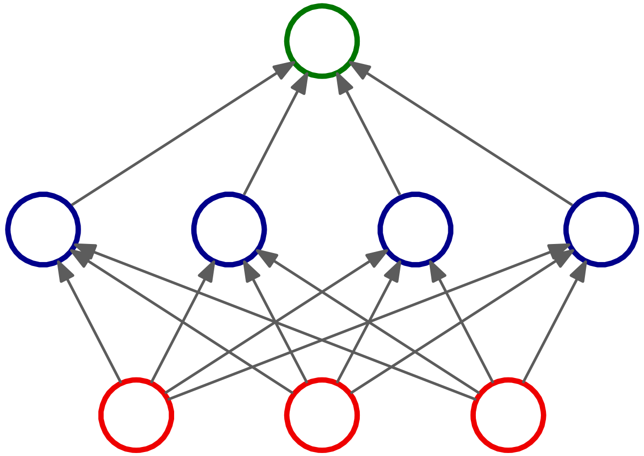
The Quest for Deeper Networks

Shallow vs. Deep Feed-Forward Neural Networks

- **Increasing network depth**

  A feed-forward neural network with one hidden layer

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}) + b$$

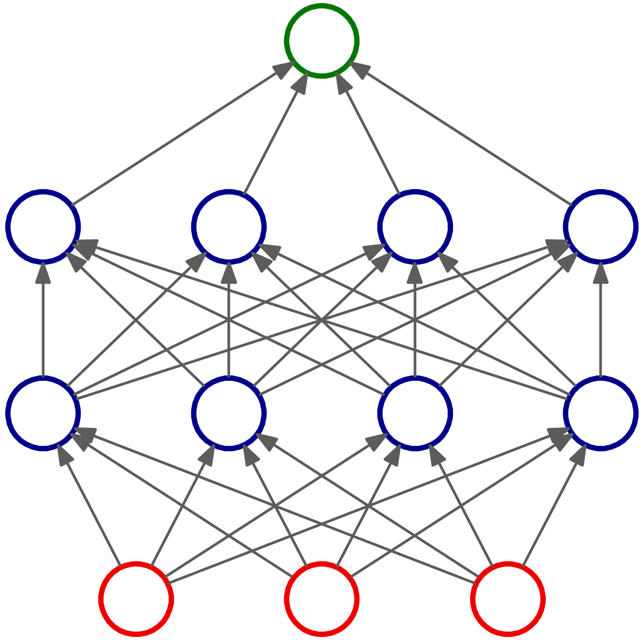


Shallow vs. Deep Feed-Forward Neural Networks

- **Increasing network depth**

A feed-forward neural network with two hidden layers

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}^{[2]} g(\boldsymbol{W}^{[1]} \boldsymbol{x} + \boldsymbol{b}^{[1]}) + \boldsymbol{b}^{[2]}) + b$$

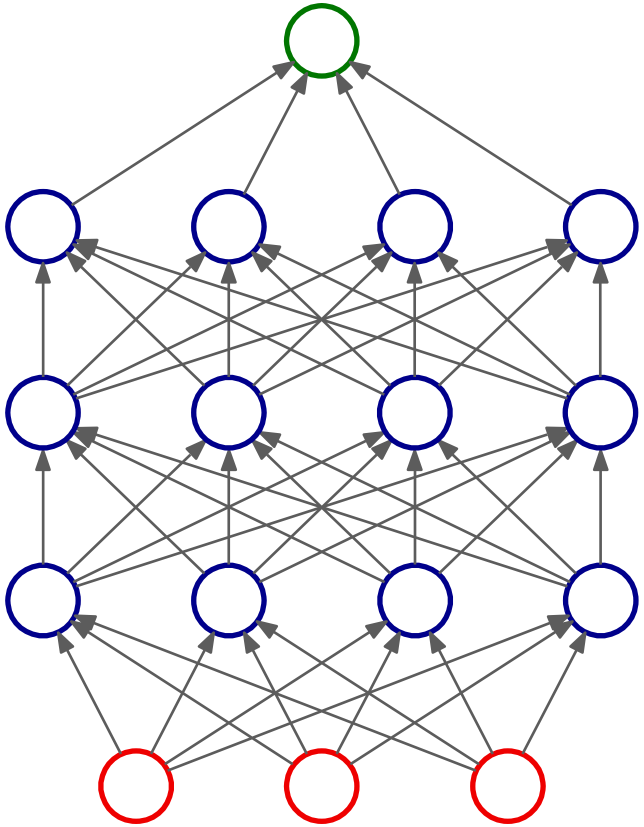


Shallow vs. Deep Feed-Forward Neural Networks

- **Increasing network depth**

  A feed-forward neural network with three hidden layers

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}^{[3]} g(\boldsymbol{W}^{[2]} g(\boldsymbol{W}^{[1]} \boldsymbol{x} + \boldsymbol{b}^{[1]}) + \boldsymbol{b}^{[2]}) + \boldsymbol{b}^{[3]}) + b$$



- **Increasing network depth**

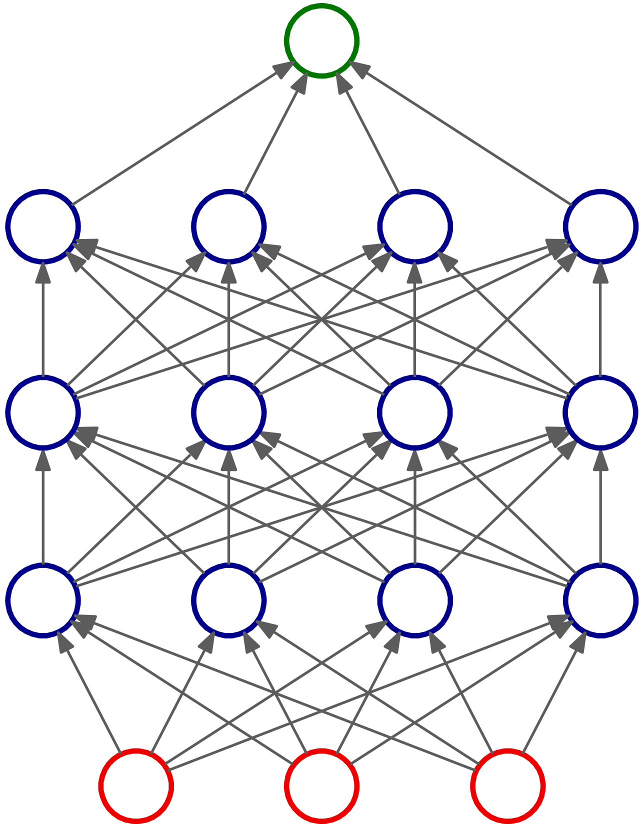
  A feed-forward neural network with three hidden layers

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}^{[3]} g(\boldsymbol{W}^{[2]} g(\boldsymbol{W}^{[1]} \boldsymbol{x} + \boldsymbol{b}^{[1]}) + \boldsymbol{b}^{[2]}) + \boldsymbol{b}^{[3]}) + b$$

OK, but what is there to gain from such increase in depth?

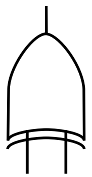
After all, the universal approximation theorem says that one layer is enough…

…and each layer brings in some extra complexity and further parameters.



Parity Circuits

A logical circuit whose output is 1 whenever the number of 1s in input is <u>*odd*</u>



XOR

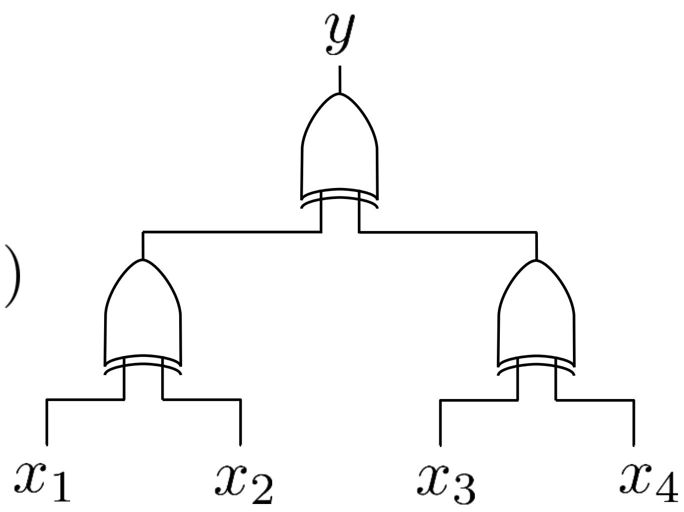
x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

For instance:

$$\boldsymbol{x} = [0, 1, 1, 0] \rightarrow y = 0$$

$$\boldsymbol{x} = [1, 1, 0, 1] \rightarrow y = 1$$

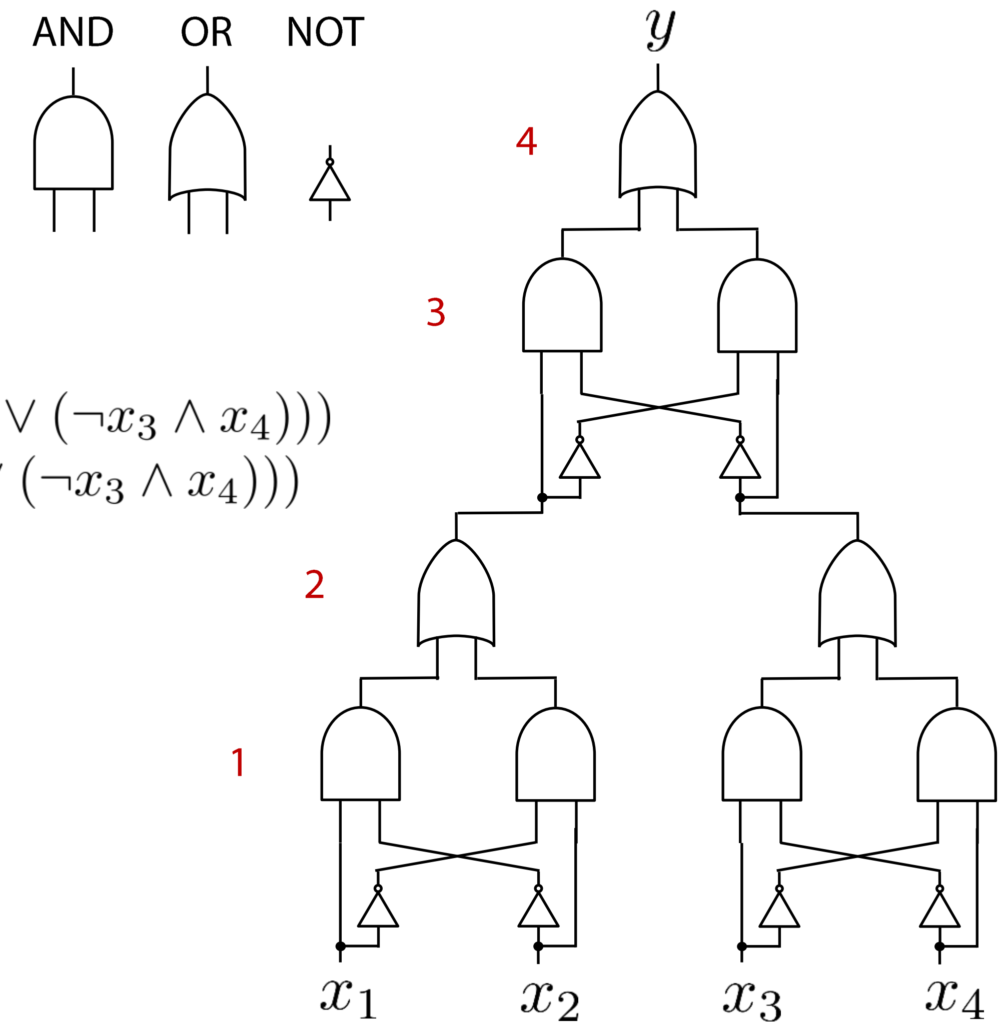
$$(x_1 \oplus x_2) \oplus (x_3 \oplus x_4)$$



This is an implementation using XOR components

Parity Circuits

An implementation of the same parity circuit using AND, OR and NOT



$$(((x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)) \wedge \neg((x_3 \wedge \neg x_4) \vee (\neg x_3 \wedge x_4)))$$
$$\vee (\neg((x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)) \wedge ((x_3 \wedge \neg x_4) \vee (\neg x_3 \wedge x_4)))$$

Note that, discounting NOTs, the depth of this circuit is 4

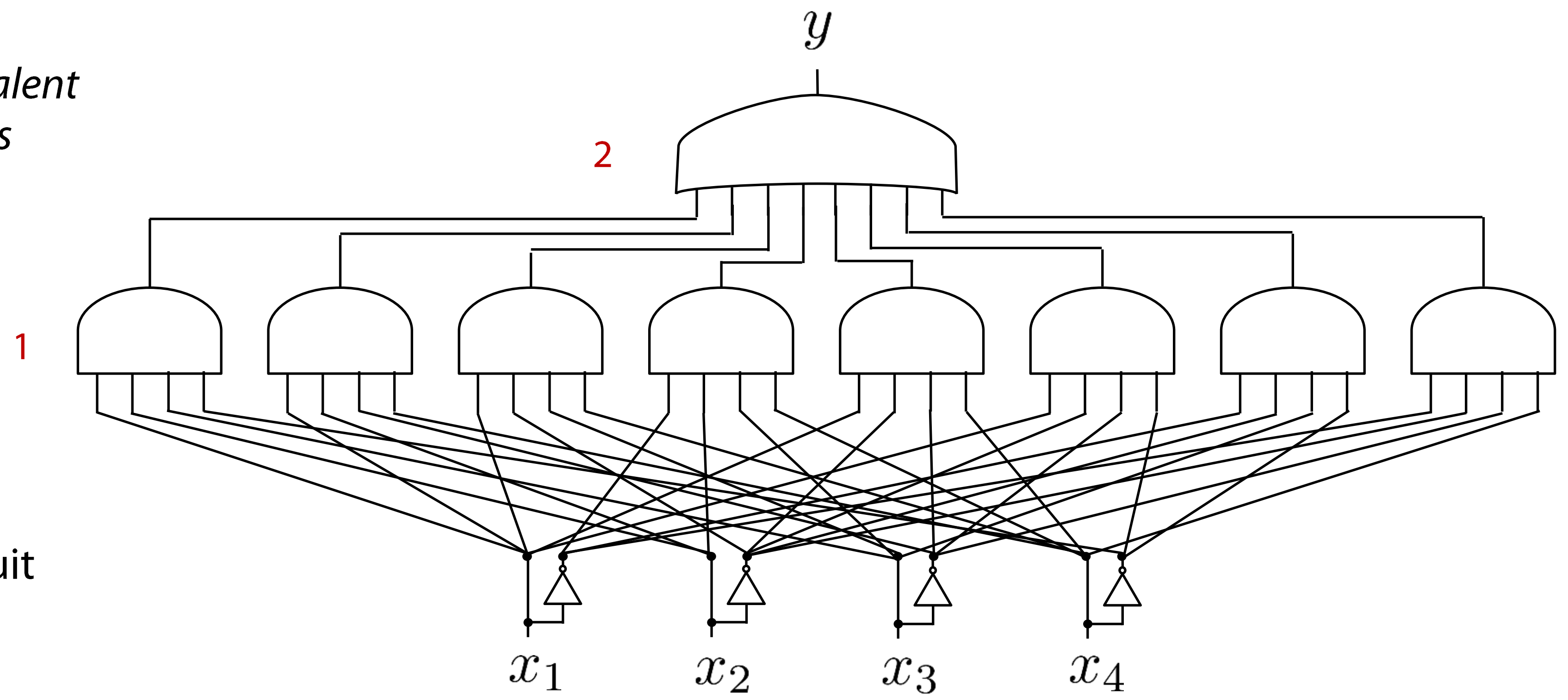
Parity Circuits

- Disjunctive Normal Form (DNF)

  Any logical formula
  can be expressed
  as an OR of ANDs
  of the inputs
  and their negations

$$
\begin{aligned}
&(x_1 \land x_2 \land x_3 \land \neg x_4) \lor (x_1 \land \neg x_2 \land x_3 \land x_4) \\
\lor\ &(x_1 \land x_2 \land \neg x_3 \land x_4) \lor (\neg x_1 \land x_2 \land x_3 \land x_4) \\
\lor\ &(x_1 \land \neg x_2 \land \neg x_3 \land \neg x_4) \lor (\neg x_1 \land x_2 \land \neg x_3 \land \neg x_4) \\
\lor\ &(\neg x_1 \land \neg x_2 \land x_3 \land \neg x_4) \lor (\neg x_1 \land \neg x_2 \land \neg x_3 \land x_4)
\end{aligned}
$$

  This circuit is equivalent
  to the previous ones



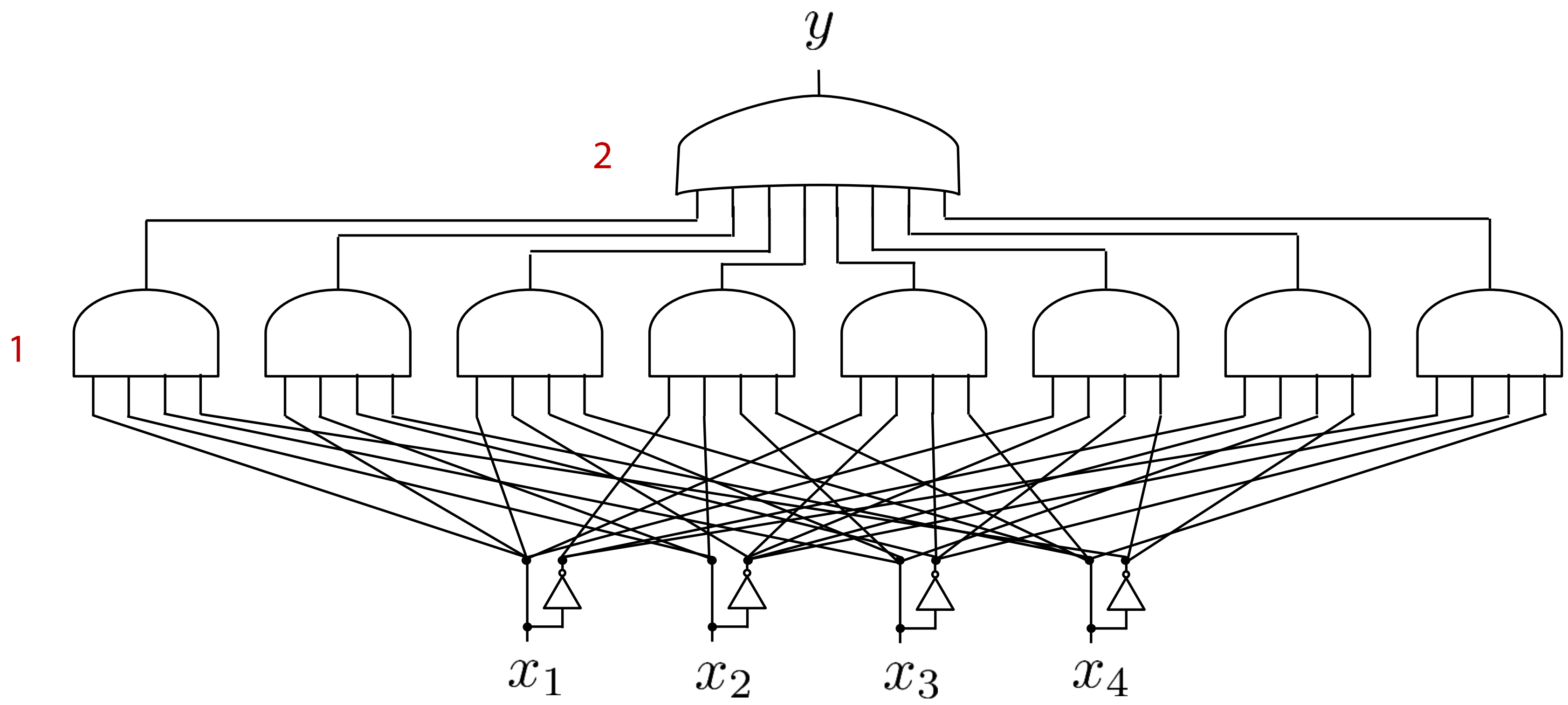
  Note that this circuit
  has depth 2

Parity Circuits

Any logical circuit can be re-implemented in *shallow* mode (i.e. with depth 2)

- Question

  Which way is better?
  (*deep* vs. *shallow*)



Parity Circuits

Any logical circuit can be re-implemented in *shallow* mode (i.e., with depth 2)

- Lower Bound (Hastad, 1986)

For the implementation of *parity circuits*
the number of AND, OR components required is

$$\Omega\left(\exp\left(d^{\frac{1}{k-1}}\right)\right)$$

d is the number of bits in input
k is the maximum depth allowed

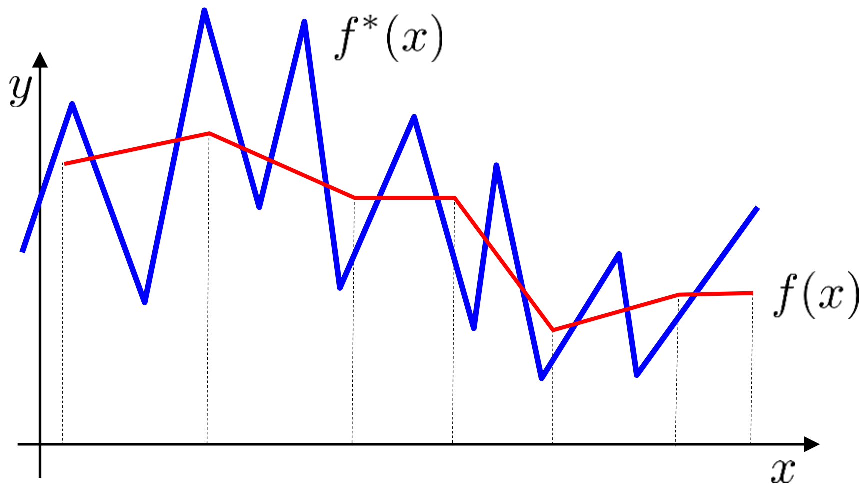
The above quantity becomes polynomial for

$$k = \frac{\log(d-1)}{\log\log(d-1) + \mathcal{O}(1)}$$

In English: there exists a threshold $k_{\min}(d)$ *beyond which*
an exponential number of components w.r.t. d *is no longer required*

Depth and piecewise linear functions

Example: a zig-zag target function:



Intuitively, the accuracy of the approximation depends on input space partitioning: unless we have a sufficient number of 'pieces' (i.e. regions in the partition)
the approximation will be inaccurate

Assume we want to use a deep neural network with ReLU

$$\tilde{y} = \boldsymbol{w} \cdot \max(0, \boldsymbol{W}^{[k]} \cdots \max(0, \boldsymbol{W}^{[1]} x + \boldsymbol{b}^{[1]}) \cdots + \boldsymbol{b}^{[k]}) + b$$

Depth and piecewise linear functions

Construct two scalar functions using ReLU and parameters

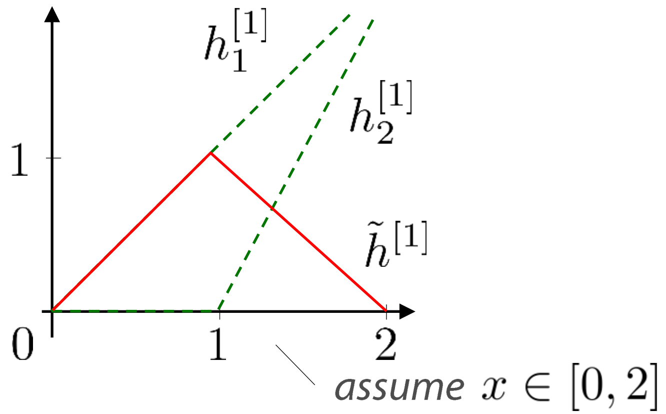
$$\tilde{h}^{[k]} := \boldsymbol{w}^{[k]} \cdot \max(0, \boldsymbol{h}^{[k]} x) \qquad \boldsymbol{h}^{[k]} := \max(0, \boldsymbol{W}^{[k]} x + \boldsymbol{b}^{[k]})$$

$$\boldsymbol{h}^{[1]} := [h_1^{[1]}, h_2^{[1]}]$$

$$h_1^{[1]} := \max(0, x)$$

$$h_2^{[1]} := \max(0, 2(x-1))$$

$$\tilde{h}^{[1]} := \max(0, x) - \max(0, 2(x-1))$$

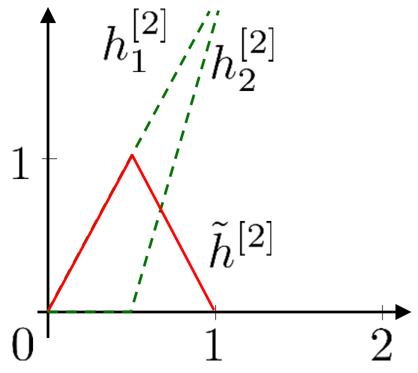


$$\boldsymbol{h}^{[2]} := [h_1^{[2]}, h_2^{[2]}]$$

$$h_1^{[2]} := \max(0, 2x)$$

$$h_2^{[2]} := \max(0, 4(x-1/2))$$

$$\tilde{h}^{[2]} := \max(0, 2x) - \max(0, 4(x-1/2))$$



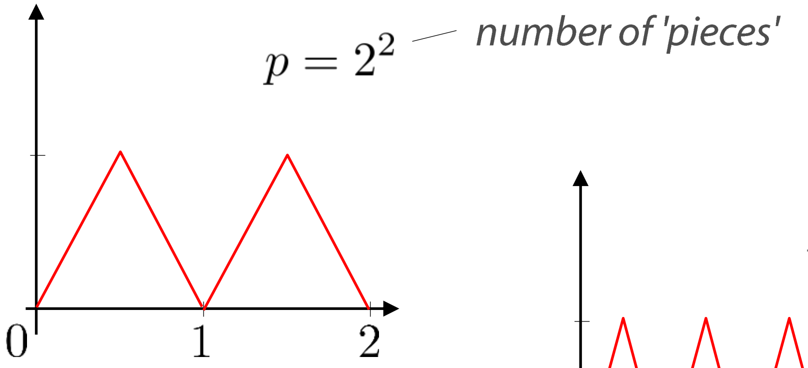
Depth and piecewise linear functions

Construct two scalar functions using ReLU plus parameters

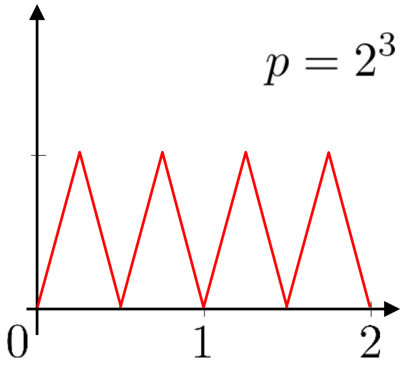
$$\tilde{h}^{[k]} := \boldsymbol{w}^{[k]} \cdot \max(0, \boldsymbol{h}^{[k]} x) \qquad \boldsymbol{h}^{[k]} := \max(0, \boldsymbol{W}^{[k]} x + \boldsymbol{b}^{[k]})$$

By nesting the two scalar functions:

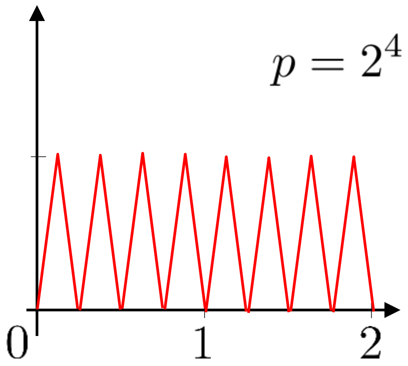
$\tilde{h}^{[2]}(\tilde{h}^{[1]}(x))$



$\tilde{h}^{[2]}(\tilde{h}^{[2]}(\tilde{h}^{[1]}(x)))$



$\tilde{h}^{[2]}(\tilde{h}^{[2]}(\tilde{h}^{[2]}(\tilde{h}^{[1]}(x))))$



Depth and piecewise linear functions

Deeper networks can make more 'pieces' with the same number of units

- **A lower bound that grows with depth** [Montufar et al. 2014]

For a network with <u>one</u> hidden layer of ReLU units of size h
the max number of pieces for the piecewise linear approximator is

$$p_{\max} = \sum_{i=0}^{d} \binom{h}{i} \leq h^d$$

input dimension

For a network with k hidden *layers* of ReLU units, each of size h,
the max number of such pieces is

$$p_{max} = \mathcal{O}(2^k), \quad p_{max} = \Omega\left(\left(\frac{h}{d}\right)^{(k-1)d} h^d\right)$$

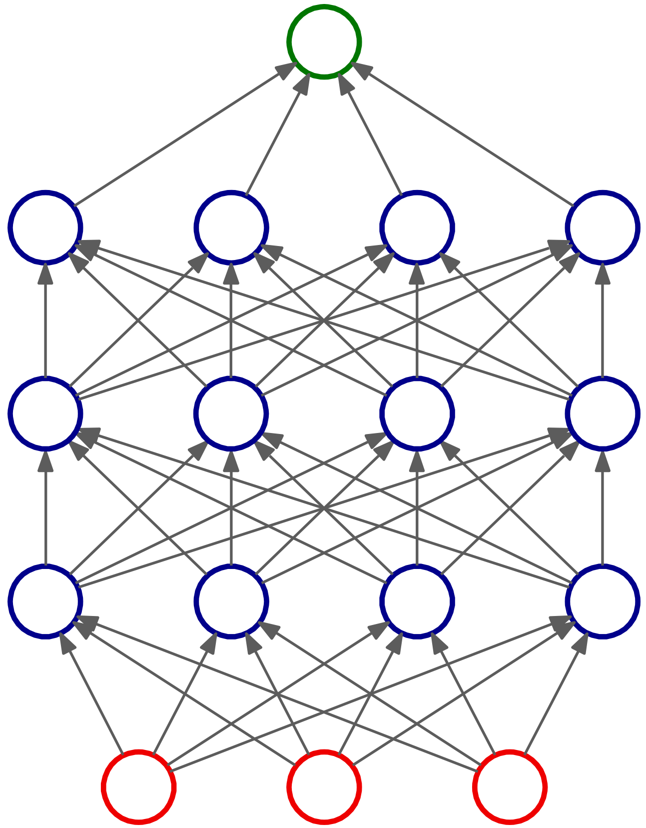
Moral: p_{max} *grows polynomially with layer size* h *but exponentially with depth* k

Layerwise differentiation

Generalizing Deep Feed-Forward Neural Networks

- **A feed-forward neural network with three hidden layers**

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}^{[3]} g(\boldsymbol{W}^{[2]} g(\boldsymbol{W}^{[1]} \boldsymbol{x} + \boldsymbol{b}^{[1]}) + \boldsymbol{b}^{[2]}) + \boldsymbol{b}^{[3]}) + b$$



Generalizing Deep Feed-Forward Neural Networks

- **A feed-forward neural network with three hidden layers**

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}^{[3]} g(\boldsymbol{W}^{[2]} g(\boldsymbol{W}^{[1]} \boldsymbol{x} + \boldsymbol{b}^{[1]}) + \boldsymbol{b}^{[2]}) + \boldsymbol{b}^{[3]}) + b$$

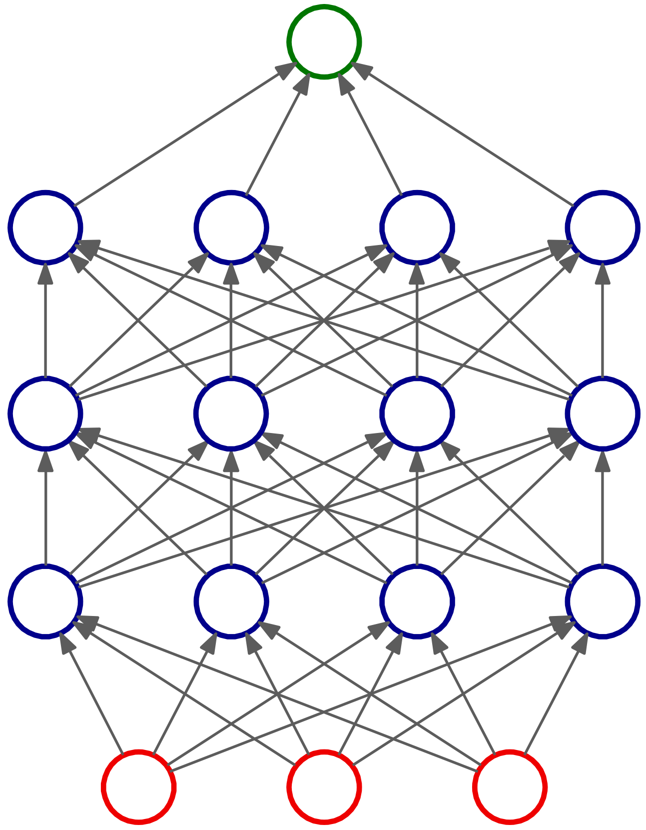
$$\tilde{y} := \boldsymbol{w} \cdot \boldsymbol{h}^{[3]} + b$$

$$\boldsymbol{h}^{[3]} := g(\boldsymbol{W}^{[3]} \boldsymbol{h}^{[2]} + \boldsymbol{b}^{[3]})$$

$$\boldsymbol{h}^{[2]} := g(\boldsymbol{W}^{[2]} \boldsymbol{h}^{[1]} + \boldsymbol{b}^{[2]})$$

$$\boldsymbol{h}^{[1]} := g(\boldsymbol{W}^{[1]} \boldsymbol{x} + \boldsymbol{b}^{[1]})$$

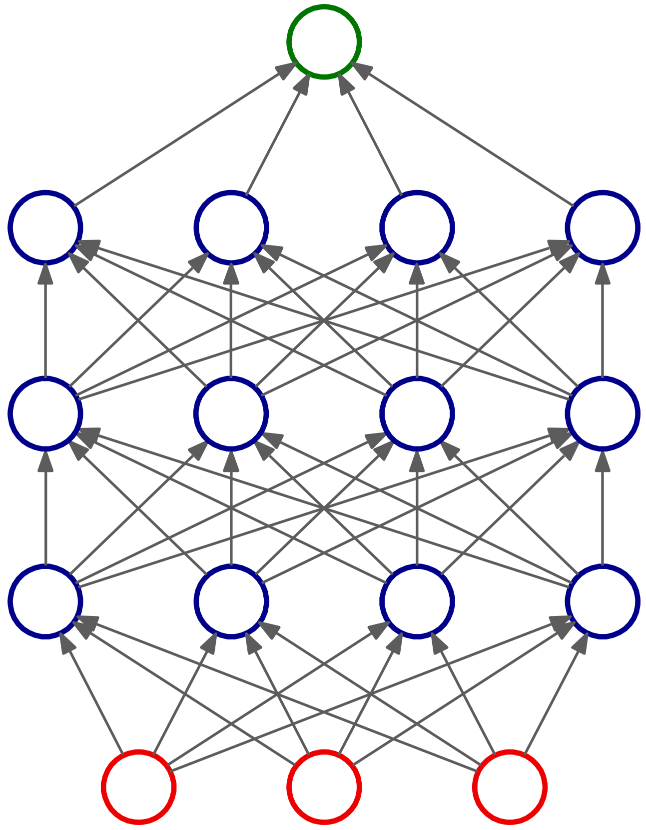
$$\boldsymbol{x}$$



- **A feed-forward neural network with three hidden layers**

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}^{[3]} g(\boldsymbol{W}^{[2]} g(\boldsymbol{W}^{[1]} \boldsymbol{x} + \boldsymbol{b}^{[1]}) + \boldsymbol{b}^{[2]}) + \boldsymbol{b}^{[3]}) + b$$

$\tilde{y}(\boldsymbol{h}^{[3]}, \boldsymbol{\vartheta}^{[\tilde{y}]})$	$\tilde{y} := \boldsymbol{w} \cdot \boldsymbol{h}^{[3]} + b$	
$\boldsymbol{h}^{[3]}(\boldsymbol{h}^{[2]}, \boldsymbol{\vartheta}^{[3]})$	$\boldsymbol{h}^{[3]} := g(\boldsymbol{W}^{[3]} \boldsymbol{h}^{[2]} + \boldsymbol{b}^{[3]})$	
$\boldsymbol{h}^{[2]}(\boldsymbol{h}^{[1]}, \boldsymbol{\vartheta}^{[2]})$	$\boldsymbol{h}^{[2]} := g(\boldsymbol{W}^{[2]} \boldsymbol{h}^{[1]} + \boldsymbol{b}^{[2]})$	
$\boldsymbol{h}^{[1]}(\boldsymbol{x}, \boldsymbol{\vartheta}^{[1]})$	$\boldsymbol{h}^{[1]} := g(\boldsymbol{W}^{[1]} \boldsymbol{x} + \boldsymbol{b}^{[1]})$	
$\boldsymbol{x}$	$\boldsymbol{x}$	



Generalizing Deep Feed-Forward Neural Networks

- **A feed-forward neural network with three hidden layers**

$$L(\tilde{y}, y) = (\tilde{y} - y)^2$$

$$\tilde{y}(\boldsymbol{h}^{[3]}, \boldsymbol{\vartheta}^{[\tilde{y}]})$$

$$\boldsymbol{h}^{[3]}(\boldsymbol{h}^{[2]}, \boldsymbol{\vartheta}^{[3]})$$

$$\boldsymbol{h}^{[2]}(\boldsymbol{h}^{[1]}, \boldsymbol{\vartheta}^{[2]})$$

$$\boldsymbol{h}^{[1]}(\boldsymbol{x}, \boldsymbol{\vartheta}^{[1]})$$

$$\boldsymbol{x}$$

Generalizing Deep Feed-Forward Neural Networks

- **Computing gradient** (layerwise)

$$L(\tilde{y}, y) = (\tilde{y} - y)^2 \qquad \frac{\partial}{\partial \boldsymbol{\vartheta}^{[\tilde{y}]}}(\tilde{y} - y)^2 = 2(\tilde{y} - y)\frac{\partial \tilde{y}}{\partial \boldsymbol{\vartheta}^{[\tilde{y}]}}$$

$$\tilde{y}(\boldsymbol{h}^{[3]}, \boldsymbol{\vartheta}^{[\tilde{y}]}) \qquad \frac{\partial \tilde{y}}{\partial \boldsymbol{\vartheta}^{[\tilde{y}]}}$$

$$\boldsymbol{h}^{[3]}(\boldsymbol{h}^{[2]}, \boldsymbol{\vartheta}^{[3]})$$

$$\boldsymbol{h}^{[2]}(\boldsymbol{h}^{[1]}, \boldsymbol{\vartheta}^{[2]})$$

$$\boldsymbol{h}^{[1]}(\boldsymbol{x}, \boldsymbol{\vartheta}^{[1]})$$

$$\boldsymbol{x}$$

Generalizing Deep Feed-Forward Neural Networks

- **Computing gradient** (layerwise)

$$L(\tilde{y}, y) = (\tilde{y} - y)^2 \qquad \frac{\partial}{\partial \boldsymbol{\vartheta}^{[3]}} (\tilde{y} - y)^2 = 2(\tilde{y} - y) \frac{\partial \tilde{y}}{\partial \boldsymbol{\vartheta}^{[3]}}$$

$$\tilde{y}(\boldsymbol{h}^{[3]}, \boldsymbol{\vartheta}^{[\tilde{y}]}) \qquad \frac{\partial \tilde{y}}{\partial \boldsymbol{\vartheta}^{[3]}} = \frac{\partial \tilde{y}}{\partial \boldsymbol{h}^{[3]}} \frac{\partial \boldsymbol{h}^{[3]}}{\partial \boldsymbol{\vartheta}^{[3]}}$$

$$\boldsymbol{h}^{[3]}(\boldsymbol{h}^{[2]}, \boldsymbol{\vartheta}^{[3]}) \qquad \frac{\partial \boldsymbol{h}^{[3]}}{\partial \boldsymbol{\vartheta}^{[3]}}$$

$$\boldsymbol{h}^{[2]}(\boldsymbol{h}^{[1]}, \boldsymbol{\vartheta}^{[2]})$$

$$\boldsymbol{h}^{[1]}(\boldsymbol{x}, \boldsymbol{\vartheta}^{[1]})$$

$$\boldsymbol{x}$$

- **Computing gradient** (layerwise)

$$L(\tilde{y}, y) = (\tilde{y} - y)^2 \qquad \frac{\partial}{\partial \boldsymbol{\vartheta}^{[2]}} (\tilde{y} - y)^2 = 2(\tilde{y} - y) \frac{\partial \tilde{y}}{\partial \boldsymbol{\vartheta}^{[2]}}$$

$$\tilde{y}(\boldsymbol{h}^{[3]}, \boldsymbol{\vartheta}^{[\tilde{y}]}) \qquad \frac{\partial \tilde{y}}{\partial \boldsymbol{\vartheta}^{[2]}} = \frac{\partial \tilde{y}}{\partial \boldsymbol{h}^{[3]}} \frac{\partial \boldsymbol{h}^{[3]}}{\partial \boldsymbol{\vartheta}^{[2]}}$$

$$\boldsymbol{h}^{[3]}(\boldsymbol{h}^{[2]}, \boldsymbol{\vartheta}^{[3]}) \qquad \frac{\partial \boldsymbol{h}^{[3]}}{\partial \boldsymbol{\vartheta}^{[2]}} = \frac{\partial \boldsymbol{h}^{[3]}}{\partial \boldsymbol{h}^{[2]}} \frac{\partial \boldsymbol{h}^{[2]}}{\partial \boldsymbol{\vartheta}^{[2]}}$$

$$\boldsymbol{h}^{[2]}(\boldsymbol{h}^{[1]}, \boldsymbol{\vartheta}^{[2]}) \qquad \frac{\partial \boldsymbol{h}^{[2]}}{\partial \boldsymbol{\vartheta}^{[2]}}$$

$$\boldsymbol{h}^{[1]}(\boldsymbol{x}, \boldsymbol{\vartheta}^{[1]})$$

$$\boldsymbol{x}$$

Generalizing Deep Feed-Forward Neural Networks

- **Computing gradient** (layerwise)

$$L(\tilde{y}, y) = (\tilde{y} - y)^2 \qquad \frac{\partial}{\partial \boldsymbol{\vartheta}^{[j]}} (\tilde{y} - y)^2 = 2(\tilde{y} - y) \frac{\partial \tilde{y}}{\partial \boldsymbol{\vartheta}^{[j]}}$$

$$\ldots \qquad \ldots$$

$$\boldsymbol{h}^{[i]}(\boldsymbol{h}^{[i-1]}, \boldsymbol{\vartheta}^{[i]}) \qquad \frac{\partial \boldsymbol{h}^{[i]}}{\partial \boldsymbol{\vartheta}^{[i]}}, \; j = i$$

$$\frac{\partial \boldsymbol{h}^{[i]}}{\partial \boldsymbol{\vartheta}^{[j]}} = \frac{\partial \boldsymbol{h}^{[i]}}{\partial \boldsymbol{h}^{[i-1]}} \frac{\partial \boldsymbol{h}^{[i-1]}}{\partial \boldsymbol{\vartheta}^{[j]}}, \; j < i$$

$$\ldots \qquad \ldots$$

Generalizing Deep Feed-Forward Neural Networks

- **Computing gradient** (layerwise)

$$L(\tilde{y}, y) = (\tilde{y} - y)^2 \qquad \frac{\partial}{\partial \boldsymbol{\vartheta}^{[j]}}(\tilde{y} - y)^2 = 2(\tilde{y} - y)\frac{\partial \tilde{y}}{\partial \boldsymbol{\vartheta}^{[j]}}$$

$$\cdots \qquad \cdots$$

$$\boldsymbol{h}^{[i]}(\boldsymbol{h}^{[i-1]}, \boldsymbol{\vartheta}^{[i]}) \qquad \boxed{\frac{\partial \boldsymbol{h}^{[i]}}{\partial \boldsymbol{\vartheta}^{[i]}}},\ j = i$$

$$\frac{\partial \boldsymbol{h}^{[i]}}{\partial \boldsymbol{\vartheta}^{[j]}} = \boxed{\frac{\partial \boldsymbol{h}^{[i]}}{\partial \boldsymbol{h}^{[i-1]}}} \frac{\partial \boldsymbol{h}^{[i-1]}}{\partial \boldsymbol{\vartheta}^{[j]}},\ j < i$$

Each layer
'needs to know'
just these two derivatives

$$\cdots \qquad \cdots$$

Function approximation (a.k.a. regression) vs. classification

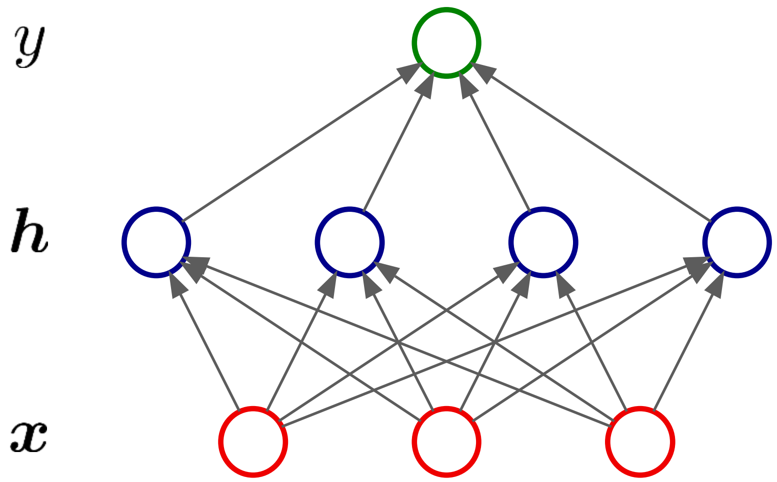
Classification: Softmax

- ***Function approximation*** *(a.k.a. regression)*

$$y = f^*(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d$$

Feed-forward neural network

$$\tilde{y} = \boldsymbol{w} \cdot g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b$$



- ***Classification***

$$y = f^*(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d, \quad y \in \{\text{class}_i\}_{i=1}^k$$

Feed-forward neural network with a *Softmax* layer

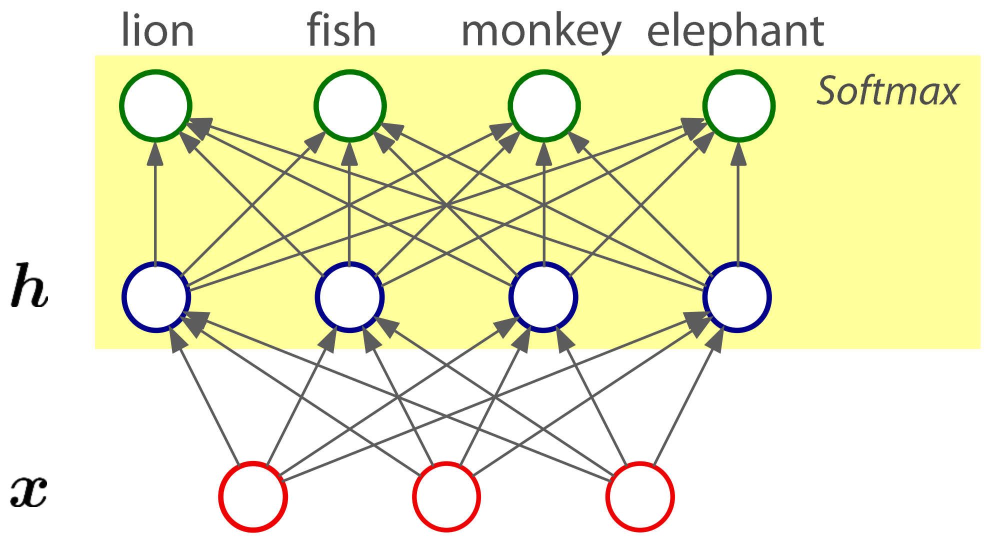
$$P(\tilde{y} = \text{class}_i \,|\, \boldsymbol{x}) := \frac{\exp(\boldsymbol{w}_i \cdot g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b_i)}{\sum_{j=1}^k \exp(\boldsymbol{w}_j \cdot g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) + b_j)}$$

From now on

$$P(\tilde{y} = \text{class}_i \,|\, \boldsymbol{x})$$

will be written as

$$P(\tilde{y} = i \,|\, \boldsymbol{x})$$



Classification: Softmax

- ***Classification***

$$y = f^*(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d, \quad y \in \{\text{class}_i\}_{i=1}^k$$

The *Softmax* layer can be rewritten as:

$$P(\tilde{y} = \text{class}_i \mid \boldsymbol{h}) := \frac{\exp(\boldsymbol{w}_i \cdot \boldsymbol{h} + b_i)}{\sum_{j=1}^k \exp(\boldsymbol{w}_j \cdot \boldsymbol{h} + b_j)}$$

where, in this case: $\boldsymbol{h} := g(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

(yet, more in general, $\boldsymbol{h}$ *can be anything)*

Classification: Softmax

- ***Softmax as a layer***

The entire *Softmax* layer can be rewritten as:

$$P((\tilde{y} = i)_1^k \,|\, \boldsymbol{h}) := \frac{\exp(\boldsymbol{W}_S \boldsymbol{h} + \boldsymbol{b}_S)}{\sum \exp(\boldsymbol{W}_S \boldsymbol{h} + \boldsymbol{b}_S)}$$

Probability distribution (a vector)

Sum of all components

where:

$$\boldsymbol{W}_S := \begin{bmatrix} - & \boldsymbol{w}_1 & - \\ & \vdots & \\ - & \boldsymbol{w}_k & - \end{bmatrix} \qquad \boldsymbol{b}_S := \begin{bmatrix} b_1 \\ \vdots \\ b_k \end{bmatrix}$$

The vector $\boldsymbol{W}_S \boldsymbol{h} + \boldsymbol{b}_S$ *is sometimes referred to as the* ***logit***

Classification: Softmax

- ***Cross-entropy* in general**

P and Q are probability distributions on a discrete random variable $y \in \{1, \cdots, k\}$

$$H(Q, P) := -\sum_{j=1}^{k} Q(y = j) \log P(\tilde{y} = j)$$

- ***As a loss function for Softmax***

Q in this case is the 'true' classification, i.e. the one in the dataset

$$Q(y = j) := \delta(y = j)$$

Kronecker delta

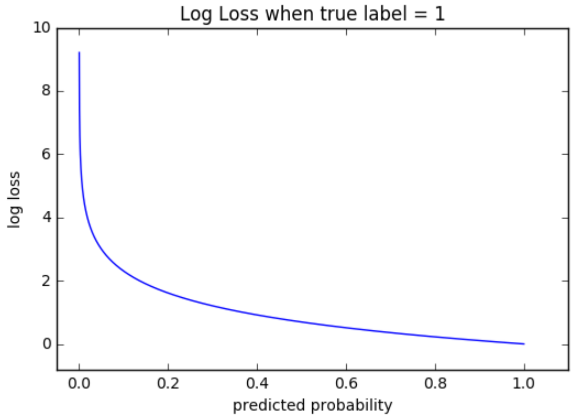
while P is the output of the Softmax layer

$$P(\tilde{y} = j \,|\, \boldsymbol{h})$$

Hence, the loss is:

$$L(\boldsymbol{h}^{(i)}, y^{(i)}) := -\sum_{j=1}^{k} \delta(y^{(i)} = j) \log P(\tilde{y} = j | \boldsymbol{h}^{(i)})$$

$$= -\log P(\tilde{y} = y^{(i)} | \boldsymbol{h}^{(i)})$$



Classification: Softmax

- ***Cross-entropy for Softmax***

$$L(\boldsymbol{h}^{(i)}, y^{(i)}) := -\sum_{j=1}^{k} \delta(y^{(i)} = j) \log P(\tilde{y} = j|\boldsymbol{h}^{(i)})$$

Expressing the loss function in vector form:

$$\boldsymbol{y} := \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}, \; y_j := \delta(y = j) \qquad \boldsymbol{p} := \begin{bmatrix} p_1 \\ \vdots \\ p_k \end{bmatrix}, \; p_j := P(\tilde{y} = j \,|\, \boldsymbol{h})$$

'one hot' representation

$$L(\boldsymbol{h}^{(i)}, \boldsymbol{y}^{(i)}) = -\,\boldsymbol{y}^{(i)} \cdot \log(\boldsymbol{p}^{(i)})$$

which implies that also the dataset has to be transformed in the 'one hot' representation

$$D := \{(\boldsymbol{x}^{(i)},\, y^{(i)})\}_{i=1}^{N} \qquad \Longrightarrow \qquad D := \{(\boldsymbol{x}^{(i)},\, \boldsymbol{y}^{(i)})\}_{i=1}^{N}$$

Classification: Softmax

- **Gradient of *Softmax*** (layerwise)

$$L(D) = \sum_{i=1}^{N} L(\boldsymbol{h}^{(i)}, \boldsymbol{y}^{(i)}) = -\sum_{i=1}^{N} \boldsymbol{y}^{(i)} \cdot \log(\boldsymbol{p}^{(i)})$$

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\vartheta}} L(D) &= \frac{\partial}{\partial \boldsymbol{\vartheta}} \sum_{i=1}^{N} L(\boldsymbol{h}^{(i)}, \boldsymbol{y}^{(i)}) = -\frac{\partial}{\partial \boldsymbol{\vartheta}} \sum_{i=1}^{N} \boldsymbol{y}^{(i)} \cdot \log(\boldsymbol{p}^{(i)}) \\ &= -\sum_{i=1}^{N} \boldsymbol{y}^{(i)} \cdot \frac{\partial}{\partial \boldsymbol{\vartheta}} \log(\boldsymbol{p}^{(i)})\end{aligned}$$

This is a matrix

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} \log(\boldsymbol{p}) = \begin{bmatrix} \frac{\partial}{\partial \vartheta_1} \log(p_1) & \dots & \frac{\partial}{\partial \vartheta_d} \log(p_1) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial \vartheta_1} \log(p_k) & \dots & \frac{\partial}{\partial \vartheta_d} \log(p_k) \end{bmatrix} = \begin{bmatrix} - & \frac{\partial}{\partial \boldsymbol{\vartheta}} \log(p_1) & - \\ & \vdots & \\ - & \frac{\partial}{\partial \boldsymbol{\vartheta}} \log(p_k) & - \end{bmatrix}$$

Classification: Softmax

- **Gradient of *Softmax*** (layerwise)

$$
\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\vartheta}} \log(p_j) &= \frac{\partial}{\partial \boldsymbol{\vartheta}} \log P(\tilde{y} = j \mid \boldsymbol{h}) \\
&= \frac{\partial}{\partial \boldsymbol{\vartheta}} \log \frac{\exp(\boldsymbol{w}_j \cdot \boldsymbol{h} + b_j)}{\sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l)} \\
&= \frac{\partial}{\partial \boldsymbol{\vartheta}} \left(\log \exp(\boldsymbol{w}_j \cdot \boldsymbol{h} + b_j) - \log \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) \right) \\
&= \frac{\partial}{\partial \boldsymbol{\vartheta}} (\boldsymbol{w}_j \cdot \boldsymbol{h} + b_j) - \frac{\partial}{\partial \boldsymbol{\vartheta}} \log \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l)
\end{aligned}
$$

Classification: Softmax

- **Gradient of *Softmax*** (layerwise)

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} \log(p_j) = \frac{\partial}{\partial \boldsymbol{\vartheta}} (\boldsymbol{w}_j \cdot \boldsymbol{h} + b_j) - \frac{\partial}{\partial \boldsymbol{\vartheta}} \log \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l)$$

Case 1: $\boldsymbol{\vartheta} = \boldsymbol{w}_r$ or $\boldsymbol{\vartheta} = b_r$

$$\frac{\partial \boldsymbol{h}^{[i]}}{\partial \boldsymbol{\vartheta}^{[i]}}$$

Case 2: $\boldsymbol{h}(\boldsymbol{\vartheta})$ i.e. $\boldsymbol{\vartheta}$ is a generic parameter on which $\boldsymbol{h}$ depends

$$\frac{\partial \boldsymbol{h}^{[i]}}{\partial \boldsymbol{\vartheta}^{[j]}}, \quad j < i$$

Let's compute the two contributions separately

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} (\boldsymbol{w}_j \cdot \boldsymbol{h} + b_j)$$

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} \log \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l)$$

Classification: Softmax

- **Gradient of *Softmax*** (layerwise)

$$\frac{\partial}{\partial \boldsymbol{\vartheta}}(\boldsymbol{w}_j \cdot \boldsymbol{h} + b_j)$$

Case 1: $\boldsymbol{\vartheta} = \boldsymbol{w}_r$ or $\boldsymbol{\vartheta} = b_r$

$$\frac{\partial}{\partial \boldsymbol{w}_r}(\boldsymbol{w}_j \cdot \boldsymbol{h} + b_j) = \begin{cases} \mathbf{0} & \text{if } r \neq j \\ \boldsymbol{h} & \text{otherwise} \end{cases}$$

$$\frac{\partial}{\partial b_r}(\boldsymbol{w}_j \cdot \boldsymbol{h} + b_j) = \begin{cases} 0 & \text{if } r \neq j \\ 1 & \text{otherwise} \end{cases}$$

Case 2: $\boldsymbol{h}(\boldsymbol{\vartheta})$ i.e. $\boldsymbol{\vartheta}$ is a generic parameter on which $\boldsymbol{h}$ depends

$$\frac{\partial}{\partial \boldsymbol{\vartheta}}(\boldsymbol{w}_j \cdot \boldsymbol{h} + b_j) = \boldsymbol{w}_j \cdot \frac{\partial}{\partial \boldsymbol{\vartheta}}\boldsymbol{h}$$

- **Gradient of *Softmax*** (layerwise)

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} \log \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l)$$

Case 1: $\boldsymbol{\vartheta} = \boldsymbol{w}_r$ or $\boldsymbol{\vartheta} = b_r$

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{w}_r} \log \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) &= \\
&= \frac{1}{\sum_{m=1}^{k} \exp(\boldsymbol{w}_m \cdot \boldsymbol{h} + b_m)} \frac{\partial}{\partial \boldsymbol{w}_r} \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) \\
&= \frac{1}{\sum_{m=1}^{k} \exp(\boldsymbol{w}_m \cdot \boldsymbol{h} + b_m)} \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) \frac{\partial}{\partial \boldsymbol{w}_r} (\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) \\
&= \frac{\exp(\boldsymbol{w}_r \cdot \boldsymbol{h} + b_r)}{\sum_{m=1}^{k} \exp(\boldsymbol{w}_m \cdot \boldsymbol{h} + b_m)} \boldsymbol{h} \quad = p_r \boldsymbol{h}
\end{aligned}$$

Classification: Softmax

- **Gradient of *Softmax*** (layerwise)

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} \log \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l)$$

Case 1: $\boldsymbol{\vartheta} = \boldsymbol{w}_r$ or $\boldsymbol{\vartheta} = b_r$

$$\begin{aligned}
\frac{\partial}{\partial b_r} \log \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) &= \\
&= \frac{1}{\sum_{m=1}^{k} \exp(\boldsymbol{w}_m \cdot \boldsymbol{h} + b_m)} \frac{\partial}{\partial b_r} \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) \\
&= \frac{1}{\sum_{m=1}^{k} \exp(\boldsymbol{w}_m \cdot \boldsymbol{h} + b_m)} \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) \frac{\partial}{\partial b_r} (\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) \\
&= \frac{\exp(\boldsymbol{w}_r \cdot \boldsymbol{h} + b_r)}{\sum_{m=1}^{k} \exp(\boldsymbol{w}_m \cdot \boldsymbol{h} + b_m)} \quad = p_r
\end{aligned}$$

- **Gradient of *Softmax*** (layerwise)

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} \log \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l)$$

Case 2: $\boldsymbol{h}(\boldsymbol{\vartheta})$ i.e. $\boldsymbol{\vartheta}$ is a generic parameter on which $\boldsymbol{h}$ depends

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} \log \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) =$$

$$= \frac{1}{\sum_{m=1}^{k} \exp(\boldsymbol{w}_m \cdot \boldsymbol{h} + b_m)} \frac{\partial}{\partial \boldsymbol{\vartheta}} \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l)$$

$$= \frac{1}{\sum_{m=1}^{k} \exp(\boldsymbol{w}_m \cdot \boldsymbol{h} + b_m)} \sum_{l=1}^{k} \exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l) \frac{\partial}{\partial \boldsymbol{\vartheta}} (\boldsymbol{w}_l \cdot \boldsymbol{h} + b_l)$$

$$= \sum_{l=1}^{k} \frac{\exp(\boldsymbol{w}_l \cdot \boldsymbol{h} + b_r)}{\sum_{m=1}^{k} \exp(\boldsymbol{w}_m \cdot \boldsymbol{h} + b_m)} \boldsymbol{w}_l^T \frac{\partial}{\partial \boldsymbol{\vartheta}} \boldsymbol{h} \quad = \left(\sum_{l=1}^{k} p_l \, \boldsymbol{w}_l^T \right) \frac{\partial}{\partial \boldsymbol{\vartheta}} \boldsymbol{h}$$