Artificial Intelligence

Neural Networks

Lesson 5:
Multi-Layer Perceptrons (MLPs)

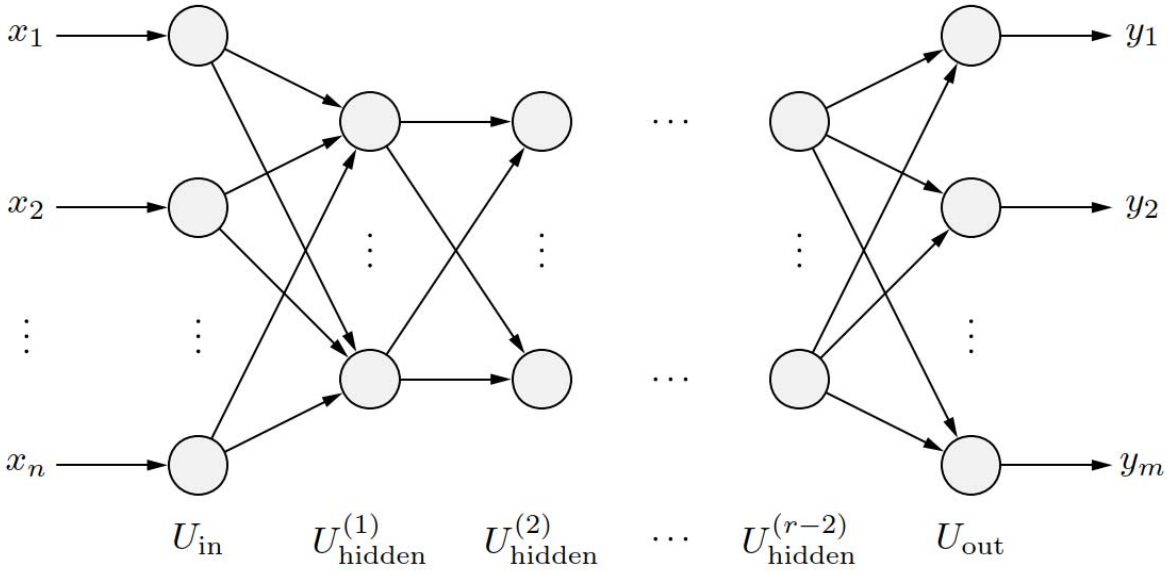
Vincenzo Piuri

Università degli Studi di Milano

Contents

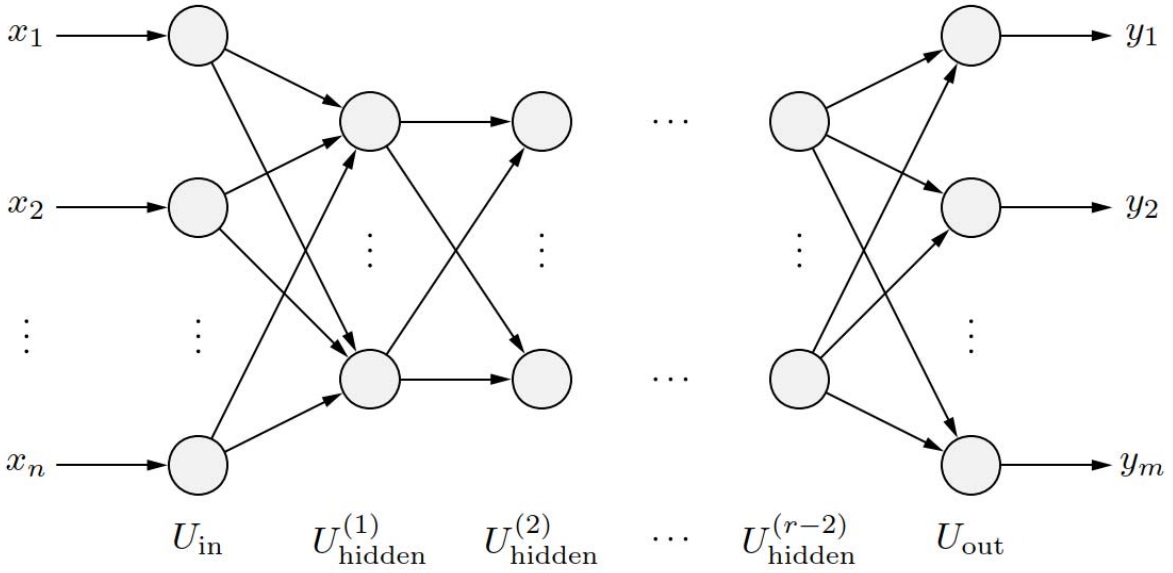
- Multi-Layer Perceptrons
- Sigmoid activation functions
- Weight matrices
- Biimplication
- Function approximation

Multi-Layer Perceptrons (1)



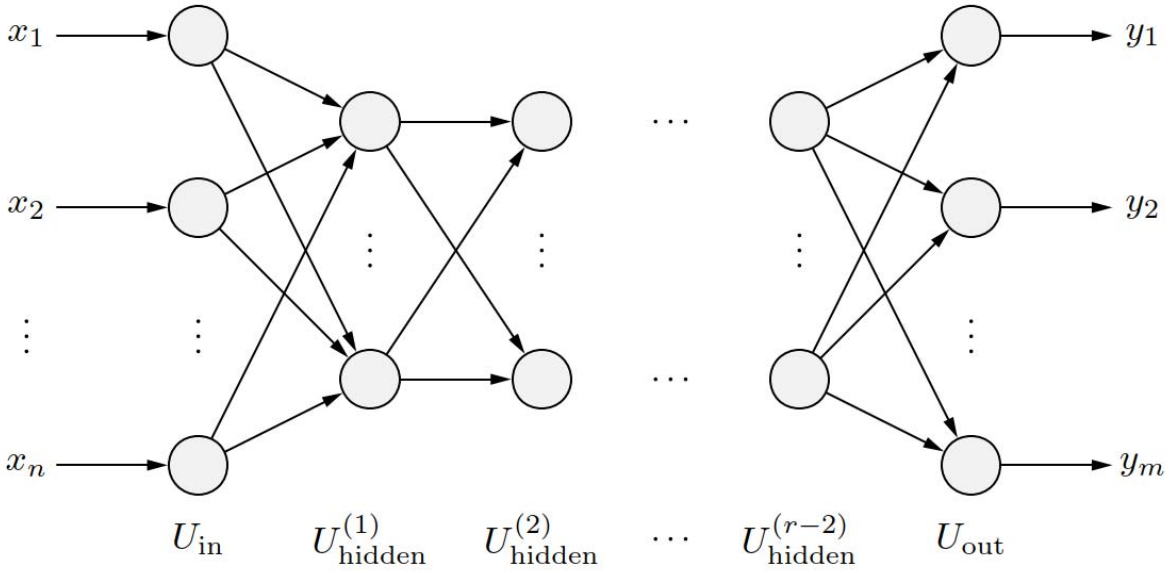
- An **r-layer perceptron** is a Feed-Forward Network with a strictly layered structure

Multi-Layer Perceptrons (2)



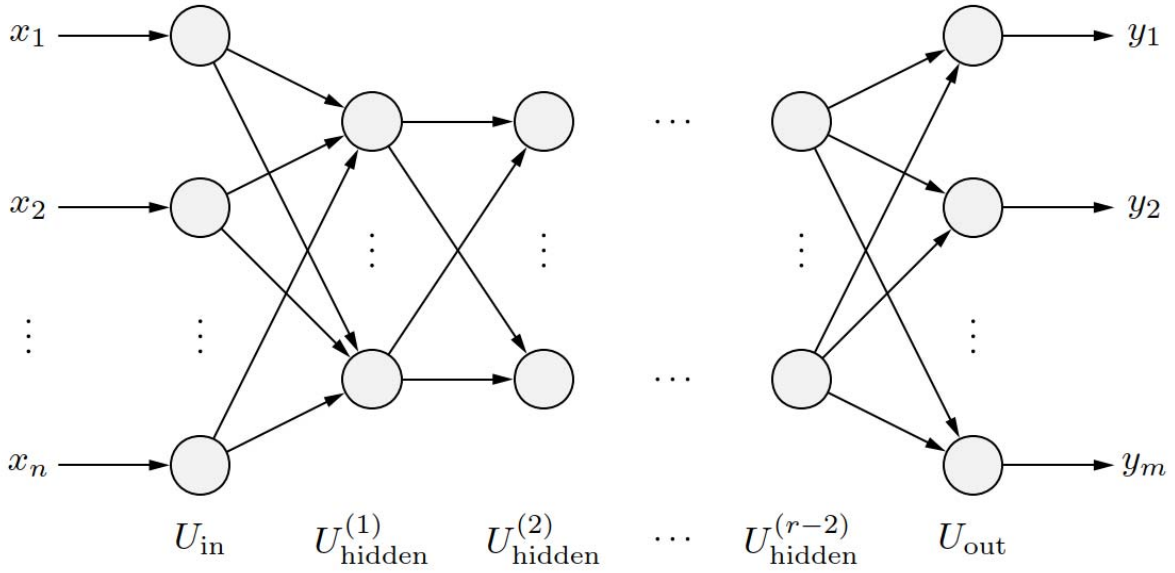
- The network input function of each hidden neuron and of each output neuron is the **weighted sum** of its inputs
  - $f_{net}^{(u)}(\overrightarrow{w_u}, \overrightarrow{in_u},) = \overrightarrow{w_u^T}\overrightarrow{in_u}, = \sum_{v \in pred(u)} w_{uv} out_v$

Multi-Layer Perceptrons (3)



- The activation function of each hidden neuron is a so-called **sigmoid function**, that is, a monotonically non-decreasing function

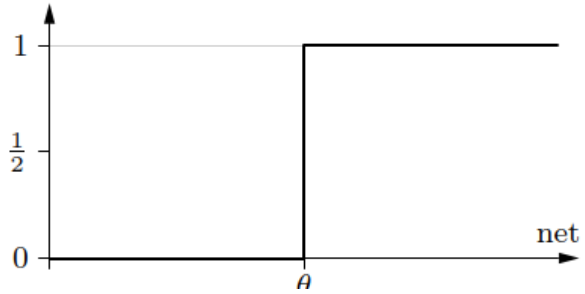
Multi-Layer Perceptrons (4)



- The activation function of each output neuron is either also a sigmoid function or a **linear function**

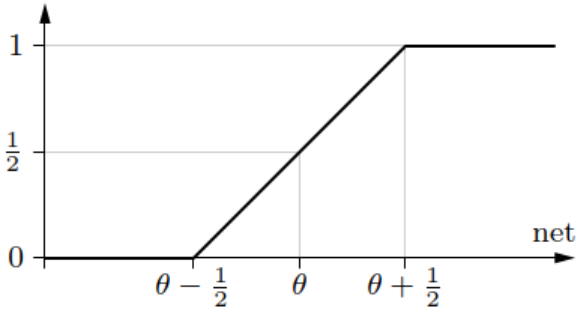
Sigmoid Activation Functions (1)

- Step function $f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{if net} \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$



- Semi-linear function

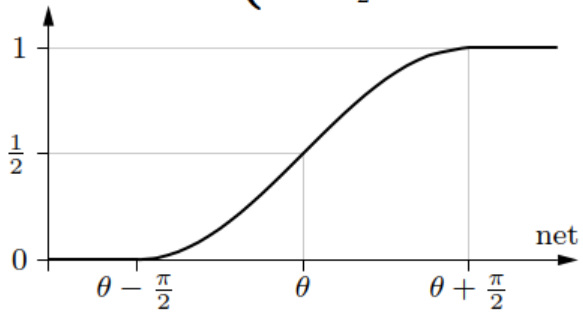
$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{if net} > \theta + \frac{1}{2}, \\ 0, & \text{if net} < \theta - \frac{1}{2}, \\ (\text{net} - \theta) + \frac{1}{2}, & \text{otherwise.} \end{cases}$$



Sigmoid Activation Functions (2)

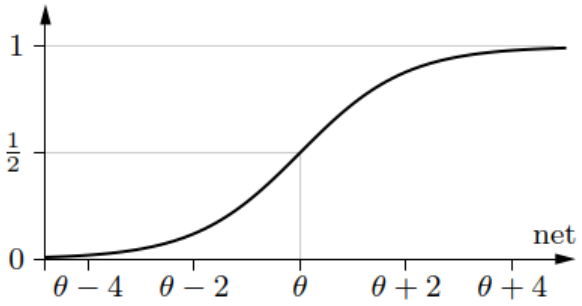
- Sine until saturation

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{if net} > \theta + \frac{\pi}{2}, \\ 0, & \text{if net} < \theta - \frac{\pi}{2}, \\ \frac{\sin(\text{net} - \theta) + 1}{2}, & \text{otherwis} \end{cases}$$



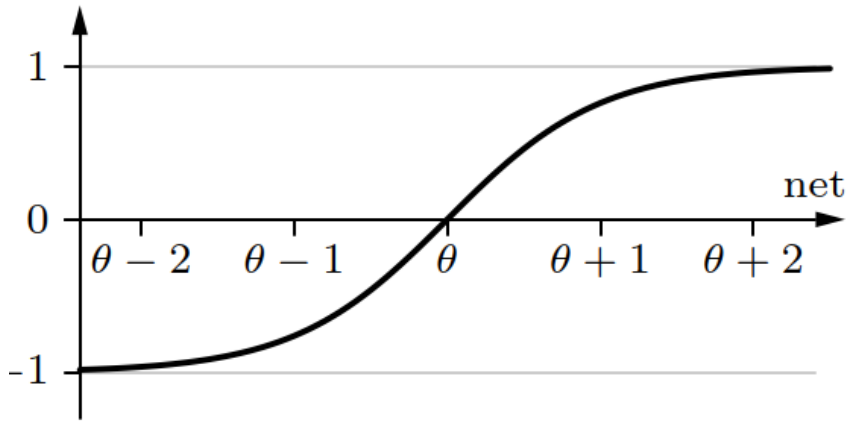
- Logistic function

$$f_{\text{act}}(\text{net}, \theta) = \frac{1}{1 + e^{-(\text{net} - \theta)}}$$



Sigmoid Activation Functions (3)

Sometimes **bipolar** sigmoid functions are used (ranging from $-1\ to\ +1$), like the hyperbolic tangent (*tangens hyperbolicus*).



Weight Matrices

Let $U_1 = \{v_1, \ldots, v_m\}$ and $U_2 = \{u_1, \ldots, u_n\}$ be the neurons of two consecutive layers of a multi-layer perceptron.

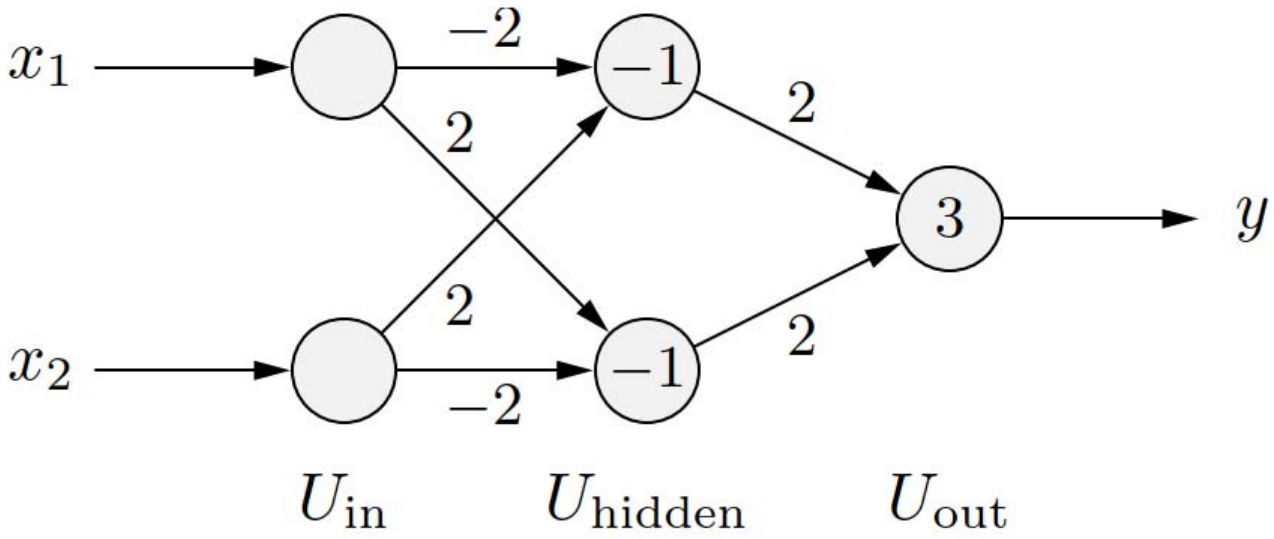
- Their connection weights are represented by an $n \times m$ matrix

$$\mathbf{W} = \begin{pmatrix} w_{u_1 v_1} & w_{u_1 v_2} & \cdots & w_{u_1 v_m} \\ w_{u_2 v_1} & w_{u_2 v_2} & \cdots & w_{u_2 v_m} \\ \vdots & \vdots & & \vdots \\ w_{u_n v_1} & w_{u_n v_2} & \cdots & w_{u_n v_m} \end{pmatrix},$$

- The computation of the network input can be written as

$$\overrightarrow{net_{U_2}} = \mathbf{W}\ \overrightarrow{in_{U_2}} = \mathbf{W}\overrightarrow{out_{U_1}}$$

Biimplication



$$\mathbf{W}_1 = \begin{pmatrix} -2 & 2 \\ 2 & -2 \end{pmatrix} \quad \text{and} \quad \mathbf{W}_2 = \begin{pmatrix} 2 & 2 \end{pmatrix}$$

- Additional input neurons compared to using TLU

Function Approximation (1)

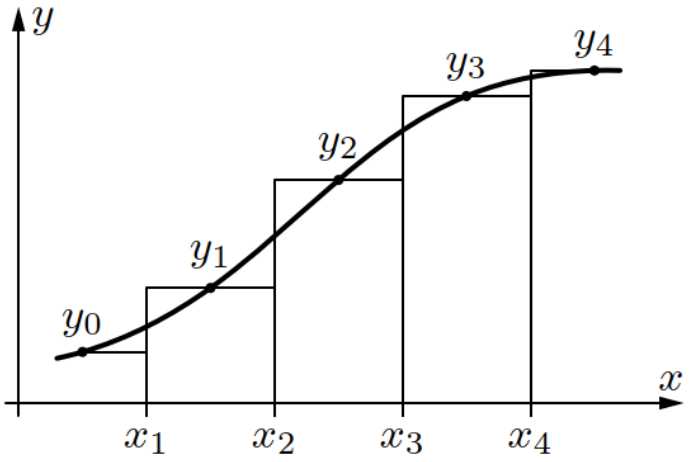
- *Up to now*: representing and learning Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$.

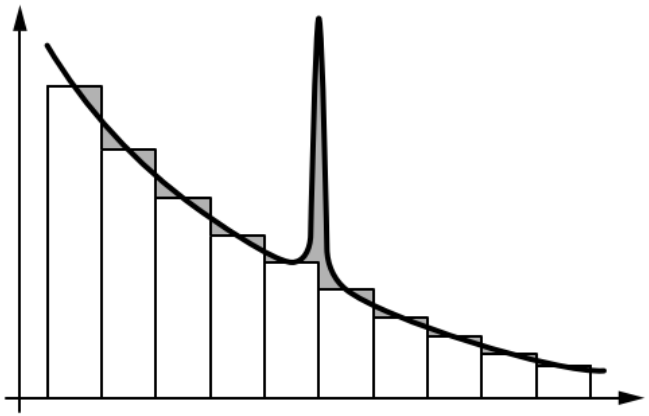
- *Now*: representing and learning real-valued functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$.

Function Approximation (2)

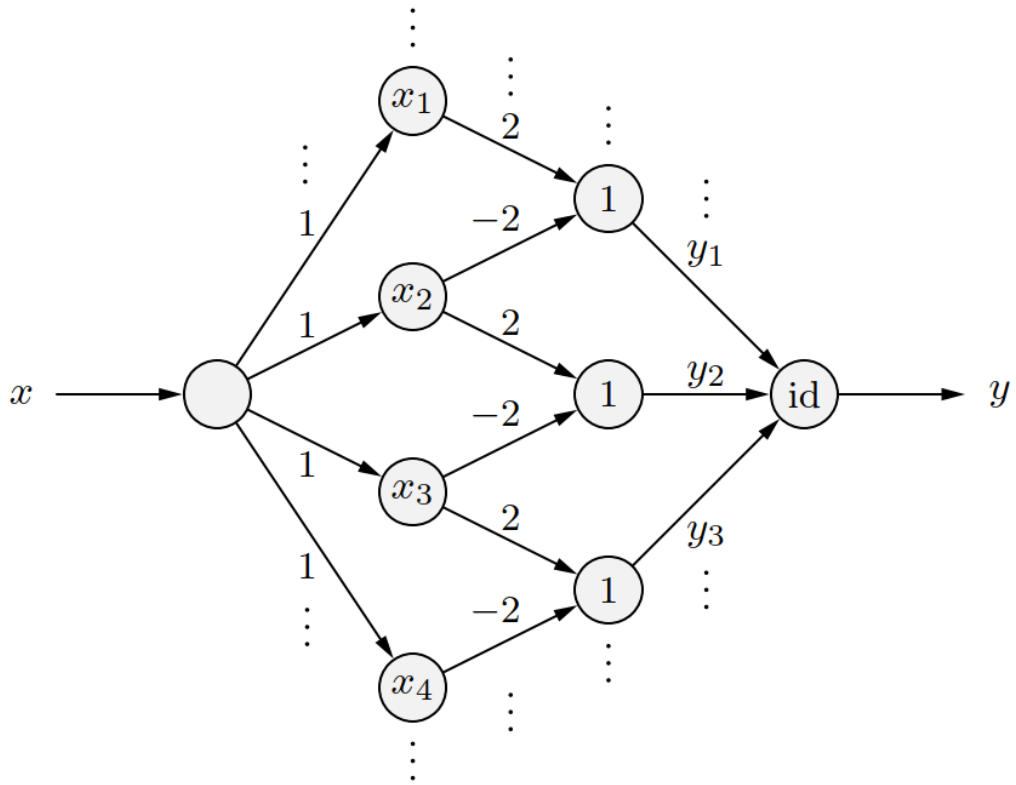
Function approximation

- Approximate a given function by a step function.
- Construct a neural network that computes the step function.
- Error is measured as the area between the functions



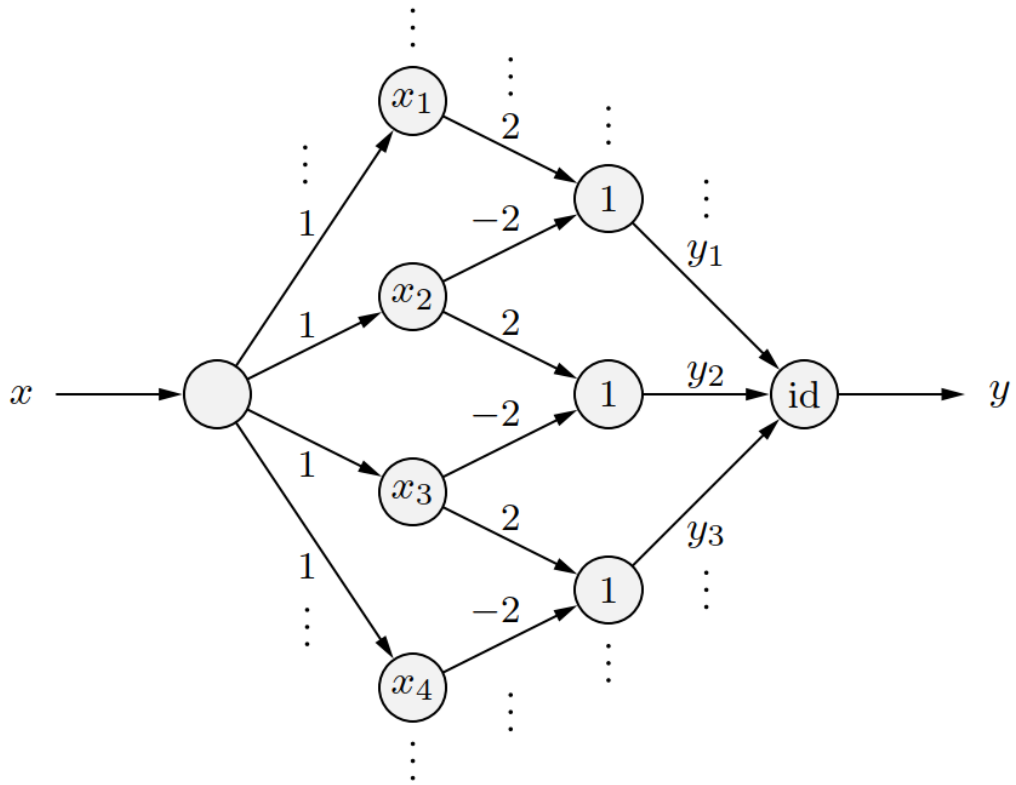


Function Approximation (3)



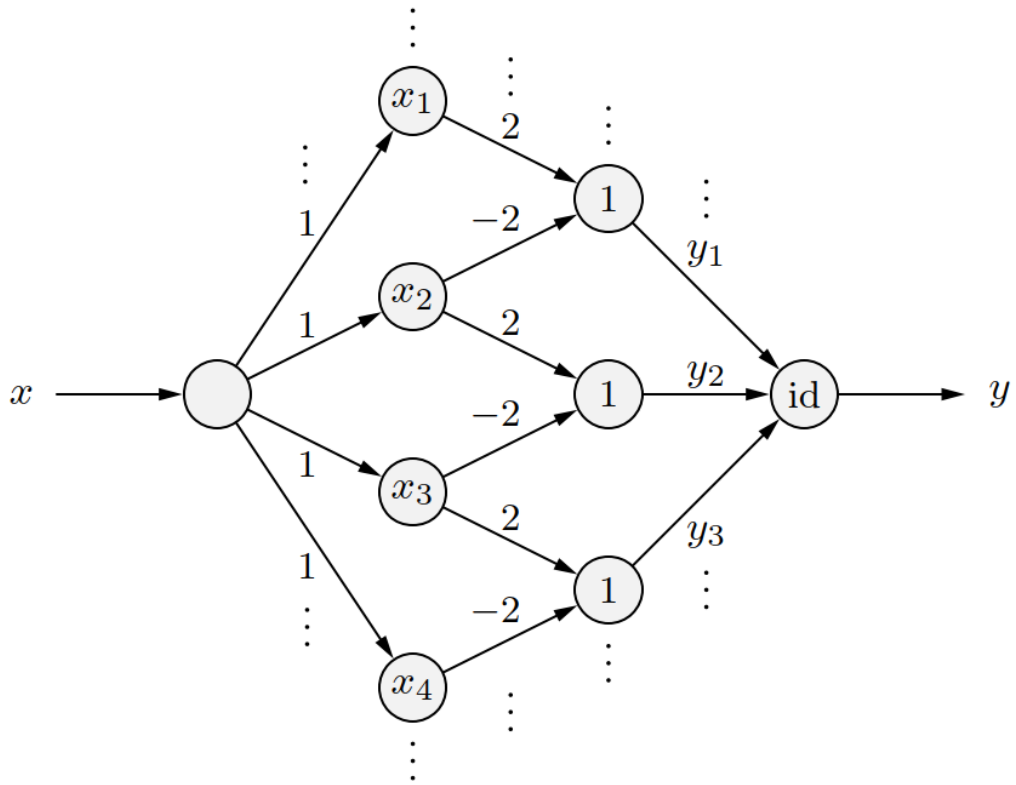
- Any Riemann-integrable function can be approximated with arbitrary accuracy by a *four-layer* perceptron.

Function Approximation (4)



- With a *three-layer* perceptron any continuous function can be approximated with arbitrary accuracy

Function Approximation (5)



- It is not the shape of the activation function, but the layered structure of the feedforward network that renders multi-layer perceptrons **universal approximators**

Function Approximation (6)