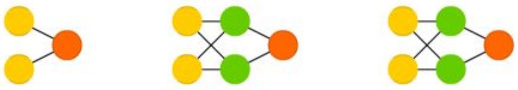
From logistic regression to neural networks

Binary classification problems

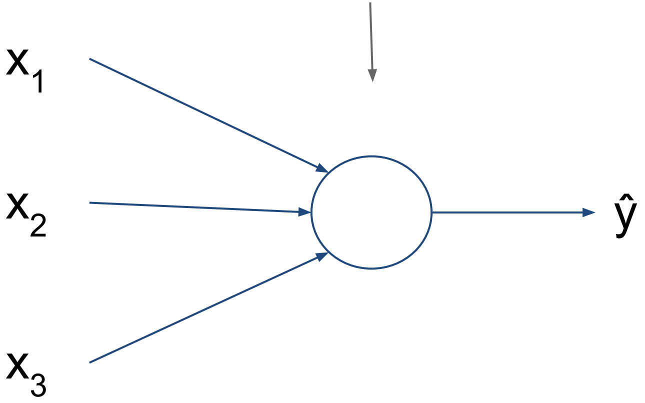
Filippo Biscarini
Senior Scientist
CNR, Milan (Italy)

Nelson Nazzicari
Research fellow
CREA, Lodi (Italy)

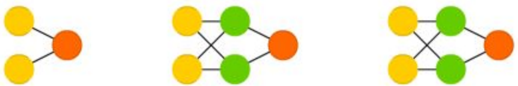


Logistic regression as a neural network

$$\underbrace{z = w \cdot x + b \longrightarrow \hat{y} = \sigma(z)}$$

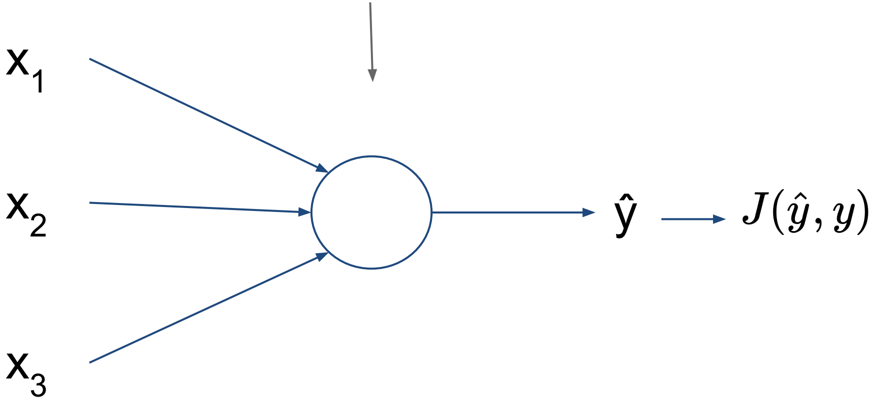


- input variables (features): x1, x2, x3
- calculations in the unit/neuron

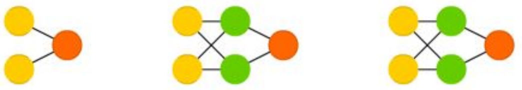


Logistic regression as a neural network

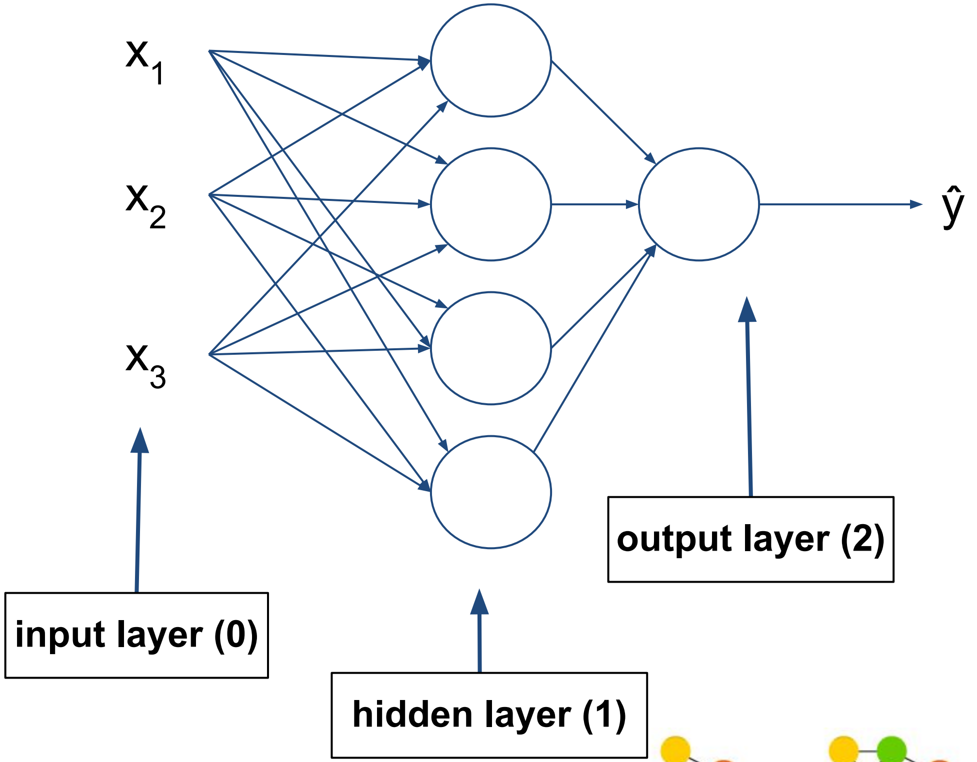
$$z = w \cdot x + b \longrightarrow \hat{y} = \sigma(z)$$

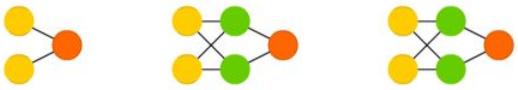


- input variables (features): x1, x2, x3
- calculations in the unit/neuron
- forward and back propagation
- this is just one single neuron!



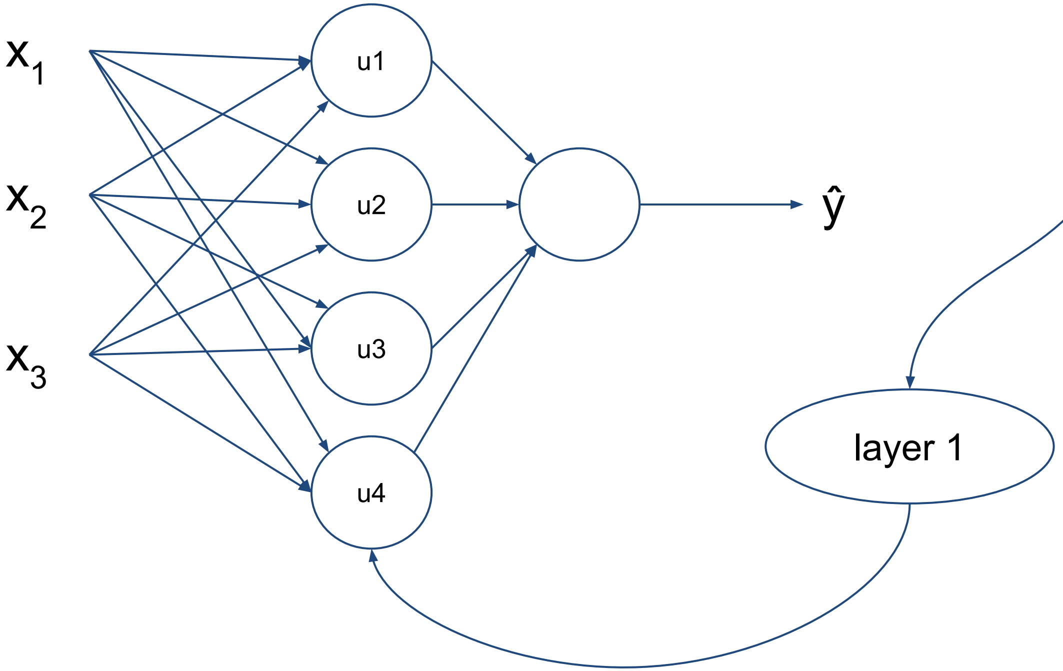
Logistic regression as a neural network





- <u>two layers</u>:
  - **1 hidden layer, 4 nodes**
  - **1 output layer, 1 node**
- logistic regression is performed in each node
- each node in the hidden layer receives all input variables
- the node in the output layer receives all outputs (**activations**) from the hidden layer nodes

Logistic regression as a neural network



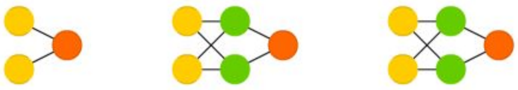
n observations, **m** features, **u** units

$$\mathbf{Z}^{[1]}_{(n,u)} = \mathbf{X}_{(n,m)} \cdot \mathbf{W}'^{[1]}_{(m,u)} + \mathbf{b}^{[1]}_{(1,u)}$$

$$\mathbf{A}^{[1]}_{(n,u)} = \sigma(\mathbf{Z}^{[1]})$$

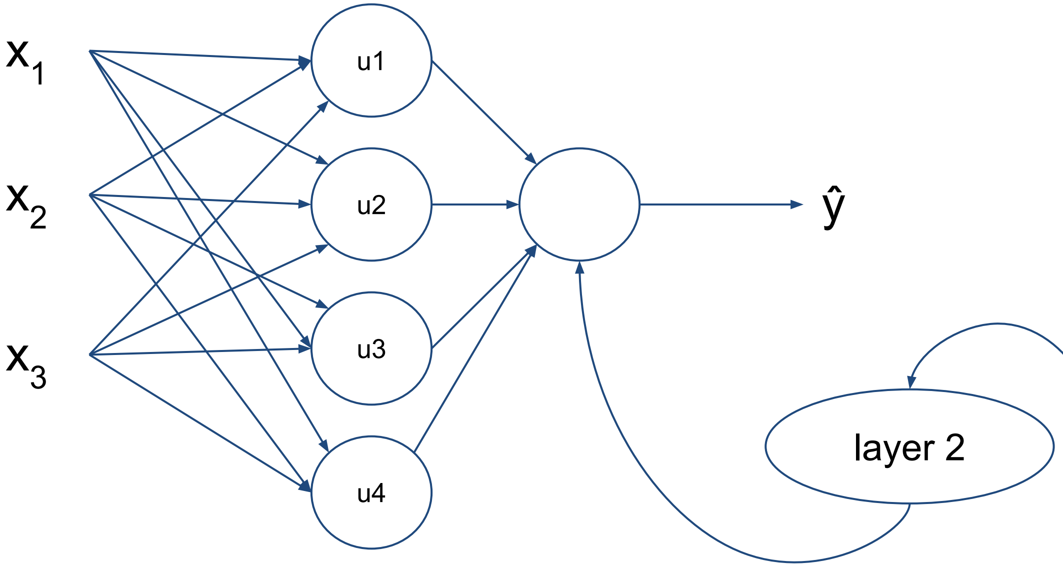
original input data matrix **X**

A for activation (output of the hidden layer)



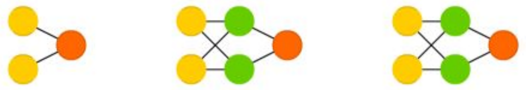
Logistic regression as a neural network

n observations, **m** features, **u** units



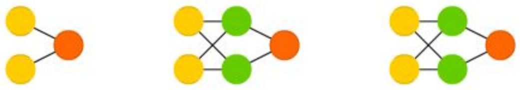
$$\left\{\begin{aligned} \mathbf{Z}^{[1]}_{(n,u)} &= \mathbf{X}_{(n,m)} \cdot \mathbf{W'}^{[1]}_{(m,u)} + \mathbf{b}^{[1]}_{(1,u)} \\ \mathbf{A}^{[1]}_{(n,u)} &= \sigma(\mathbf{Z}^{[1]}) \end{aligned}\right.$$

$$\left\{\begin{aligned} \mathbf{Z}^{[2]}_{(n,1)} &= \mathbf{A}^{[1]}_{(n,u)} \cdot \mathbf{W'}^{[2]}_{(u,1)} + \mathbf{b}^{[2]}_{(1,1)} \\ \hat{\mathbf{y}}_{(n,1)} &= \sigma(\mathbf{Z}^{[2]}) \end{aligned}\right.$$



Take away messages

- You can build a neural network (NN) for binary classification
- NNs are logistic regression repeated several times! → n. of nodes/units, n. of layers
- probably a bit of an overkill to use NNs in place of a simple logistic regression model → used for illustration
- however, when you have many observations and many features (big data), NN will do the job

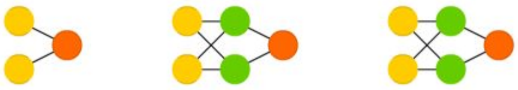


Let’s go deep

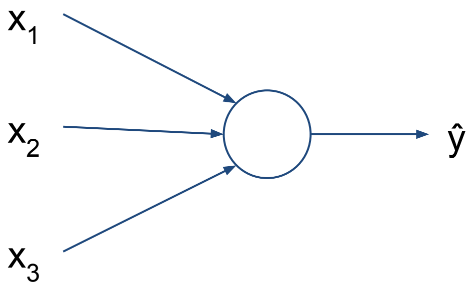
From NNs to deep learning

Filippo Biscarini
Senior Scientist
CNR, Milan (Italy)

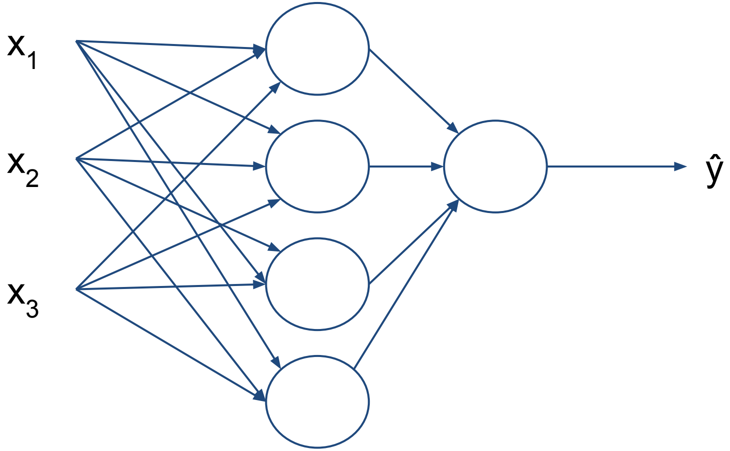
Nelson Nazzicari
Research fellow
CREA, Lodi (Italy)



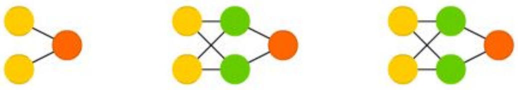
It’s a matter of layers



logistic regression
(1 layer)

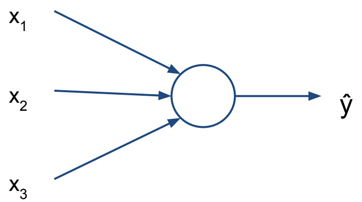


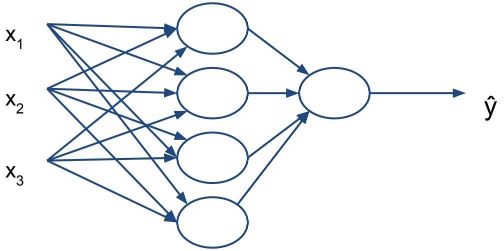
shallow NN
(2 layers)

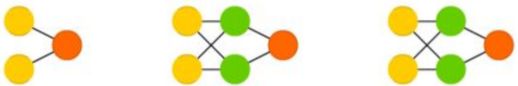


It’s a matter of layers

- we had a (cursory) look at the calculations involved in neural networks models
- Lots of details, but it’s **not a black box**!

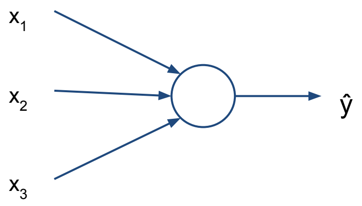


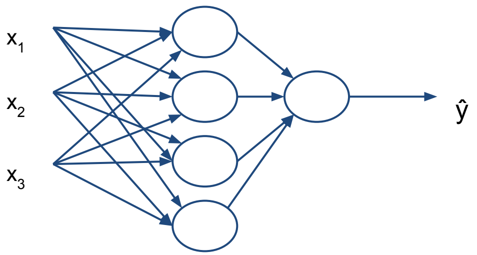




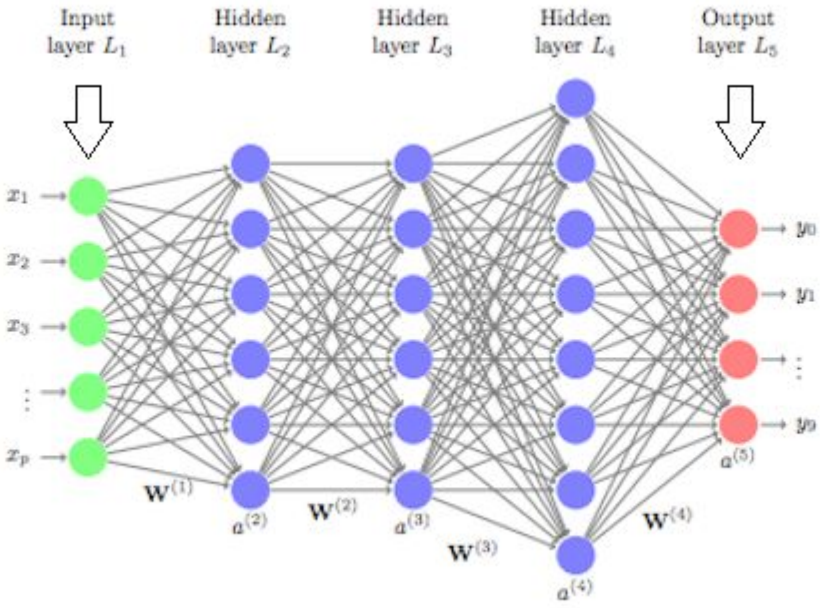
It’s a matter of layers

- we had a (cursory) look at the calculations involved in neural networks models
- Lots of details, but it’s **not a black box**!
- however, when you go deep (more layers), the magic gets back in the play!



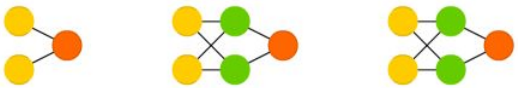


It’s a matter of layers

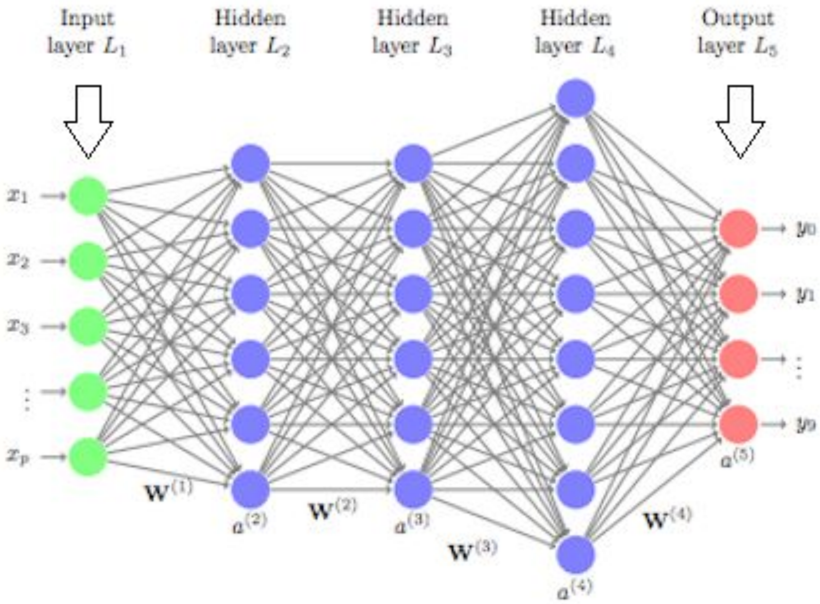


Source: University of Cincinnati

- 4 layers (“deep” NN)
- n. of layers in a deep learning model: hyperparameter to tune (one of many)

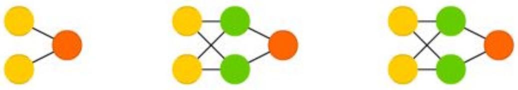


Layers matter!

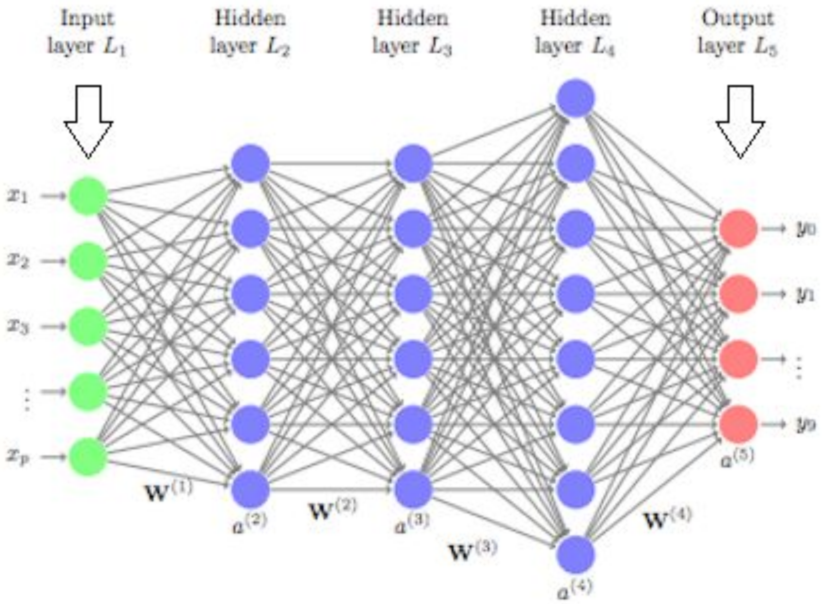


Source: University of Cincinnati

- 4 layers ("deep" NN)
- n. of layers in a deep learning model: hyperparameter to tune (one of many)
- research in the AI/machine learning communities has shown that there are functions that deep NN can learn which can not be learnt by shallower models



Forward propagation

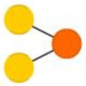


Source: University of Cincinnati

- L: n. of layers (l in 1 to L)
- X: matrix of input features → $A^{[0]}$

$$\begin{cases} \mathbf{Z}^{[l]} = \mathbf{A}^{[l-1]} \cdot \mathbf{W}'^{[l]} + \mathbf{b}^{[l]} \\ \mathbf{A}^{[l]} = g^{[l]}(\mathbf{Z}^{[l]}) \end{cases}$$

- for each layer
- iterate over n. of layers (unavoidable for loop)

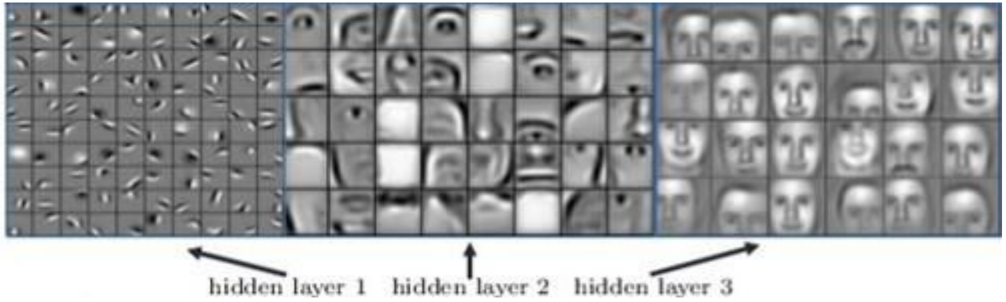


Compositional representation

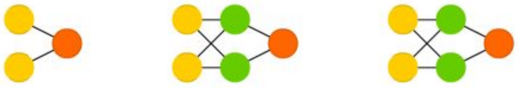
- why do deep NNs work better?
- **each layer** focuses on **one representation** of the data
- representations are then **combined** to get the final result → **compositional representation**

<u>Simplified example from face recognition</u>:

- Layer 1 → edges
- Layer 2 → pieces of faces
- Layer 3 → pieces



Source: https://medium.com/@fenjiro/face-id-deep-learning-for-face-recognition-324b50d916d1

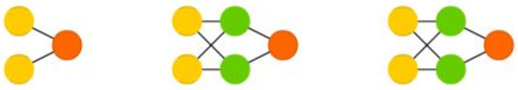


Compositional representation

- works also with other types of data
- e.g. speech recognition: i) high/low soundwaves; ii) combinations of soundwaves into phonemes; iii) combination of phonemes into words; iv) from words to sentences
- relatively simple functions of the input data in the first layers → **progressively more complex functions** of the data in the later layers

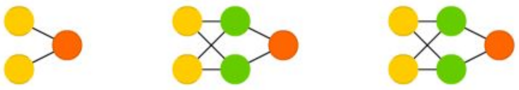
Depth vs width

- deep learning works by stacking together multiple (many) hidden layers with relatively few nodes
- alternatively, one could use a shallow but very wide (many nodes) neural network
- **depth is more efficient than width**: shallow NN require exponentially more hidden units (nodes) compared to deep NN:
    - e.g.: XOR of x features (x_1 XOR x_2 XOR … XOR x_n)
    - deep NN → $O(\log(n))$
    - wide NN → $O(2^n)$



Hyperparameters

- we saw that there are many ingredients that make up a deep learning model (and many still yet to come) → deep learning has **many hyperparameters**:
    - learning rate α
    - n. of hidden layers
    - n. of nodes (total, each layer)
    - activation function
    - and many more (mini-batch size, NN architecture, regularization etc.)
- to be fine-tuned (→ cross-validation)



Neural networks models

- demonstration 04b

  → code_04b_keras_shallow_neural_networks.ipynb