

# From logistic regression to neural networks

## Binary classification problems

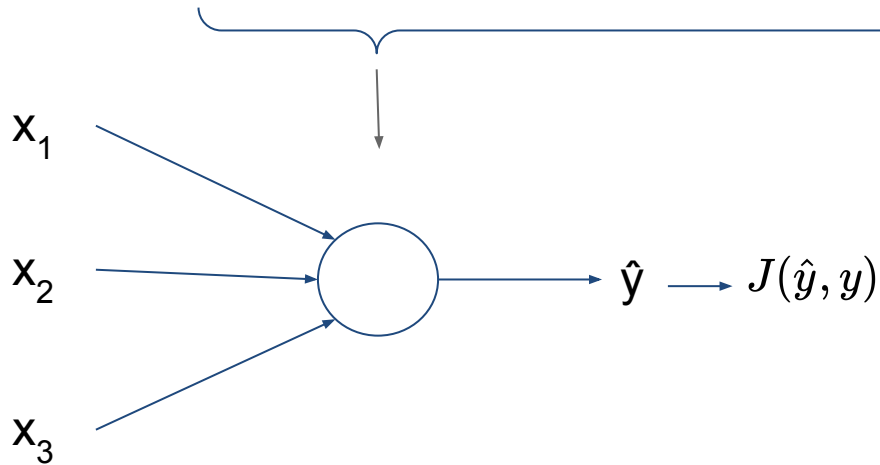
Filippo Biscarini  
Senior Scientist  
CNR, Milan (Italy)

Nelson Nazzicari  
Senior Scientist  
CREA, Lodi (Italy)



# Logistic regression as a neural network

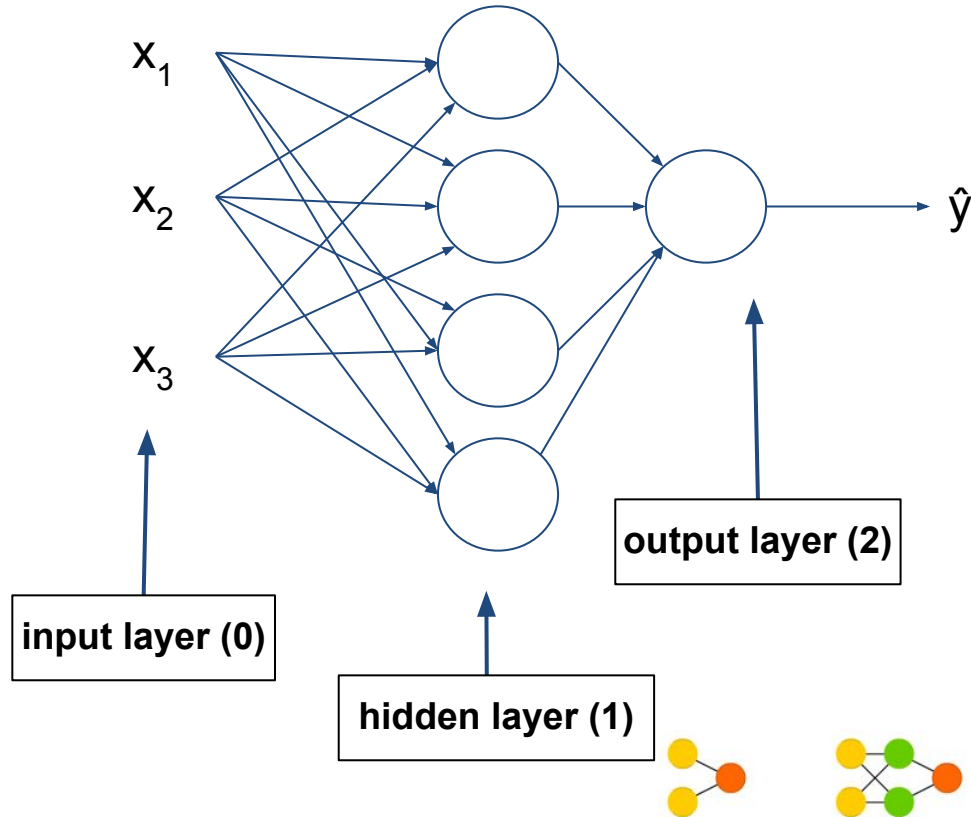
$$z = w \cdot x + b \longrightarrow \hat{y} = \sigma(z)$$



- input variables (features):  $x_1$ ,  $x_2$ ,  $x_3$
- calculations in the unit/neuron
- forward and back propagation
- **this is just one single neuron!**



# Logistic regression as a neural network



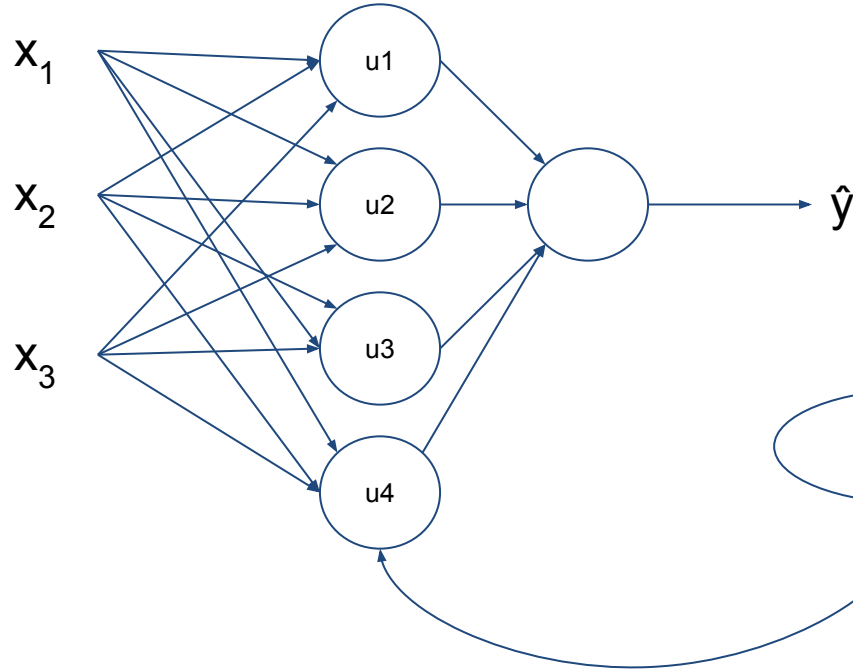
- two layers:
  - 1 hidden layer, 4 nodes
  - 1 output layer, 1 node
- logistic regression is performed in each node
- each node in the hidden layer receives all input variables
- the node in the output layer receives all outputs (**activations**) from the hidden layer nodes

**Q: what differs between one logistic regression model and the other?**



# Logistic regression as a neural network

$n$  observations,  $m$  features,  $u$  units



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

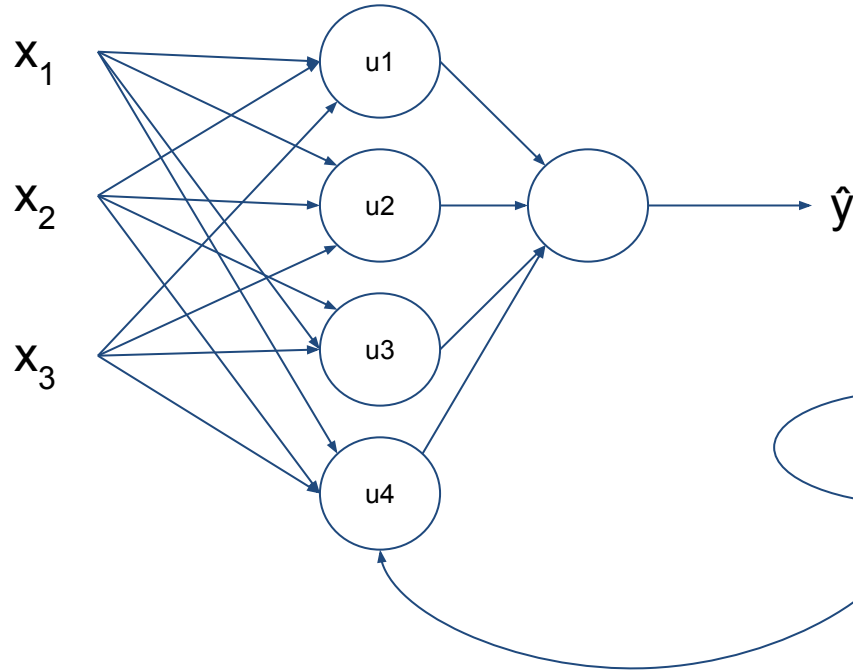
original input data matrix  $\mathbf{X}$

$\mathbf{A}$  for activation  
(output of the hidden layer)



# Logistic regression as a neural network

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



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

original input data matrix **X**

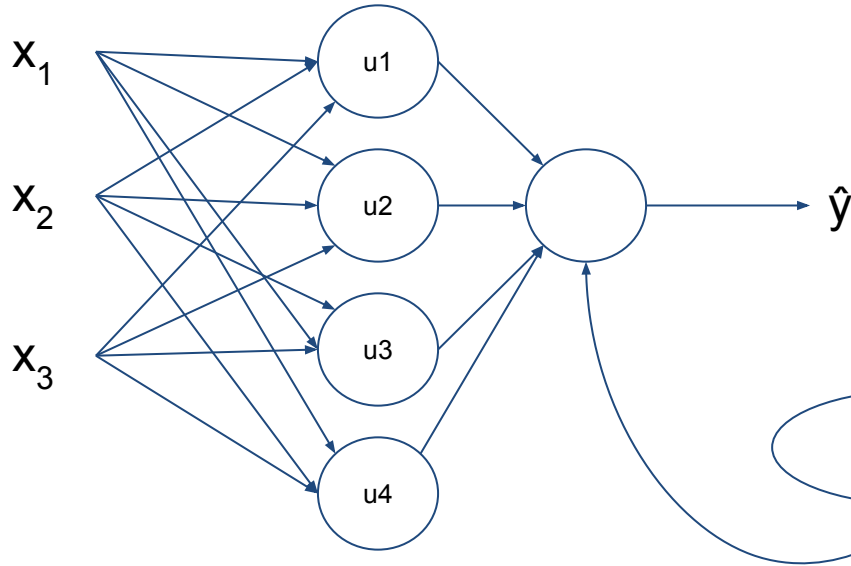
**A** for activation  
(output of the hidden layer)

What do we have in matrix **A**?



# Logistic regression as a neural network

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



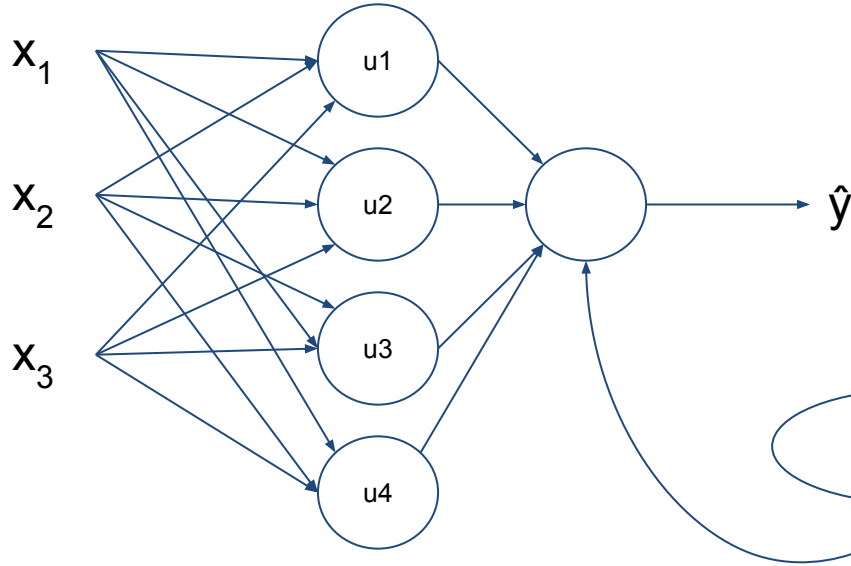
$$\begin{cases} \mathbf{Z}_{(n,u)}^{[1]} = \mathbf{X}_{(n,m)} \cdot \mathbf{W}_{(m,u)}'^{[1]} + \mathbf{b}_{(1,u)}^{[1]} \\ \mathbf{A}_{(n,u)}^{[1]} = \sigma(\mathbf{Z}^{[1]}) \end{cases}$$

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



# Logistic regression as a neural network

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



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

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

What do we have in vector  $\hat{\mathbf{y}}$ ?



# Take away messages

- You can build a neural network (NN) for binary classification
- NNs are logistic regressions 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

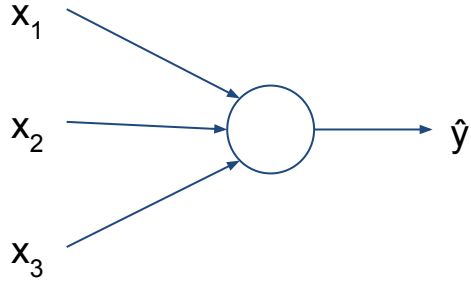
## From NNs to deep learning

Filippo Biscarini  
Senior Scientist  
CNR, Milan (Italy)

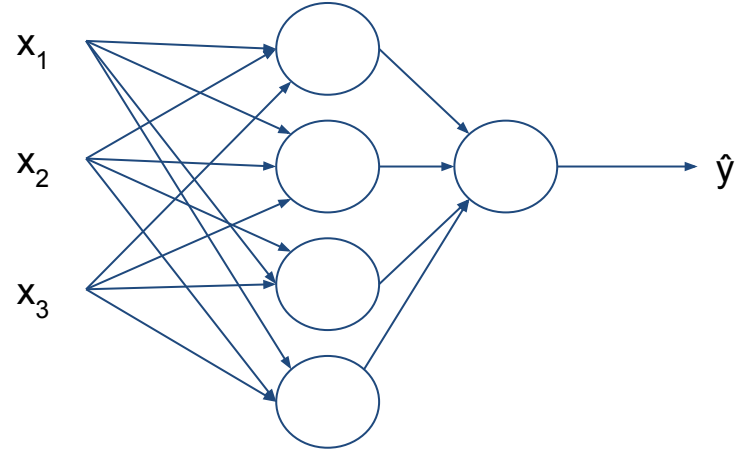
Nelson Nazzicari  
Senior Scientist  
CREA, Lodi (Italy)



# It's a matter of layers



**logistic regression**  
(1 layer)

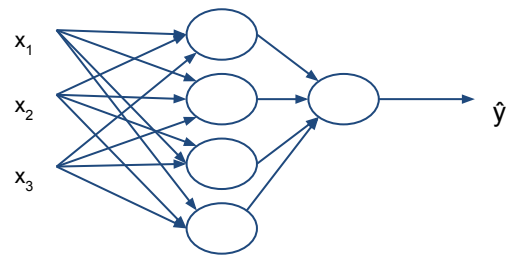
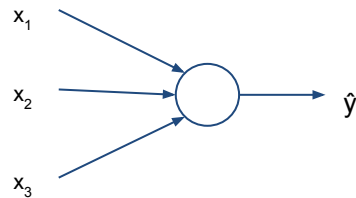


**shallow NN**  
(2 layers)



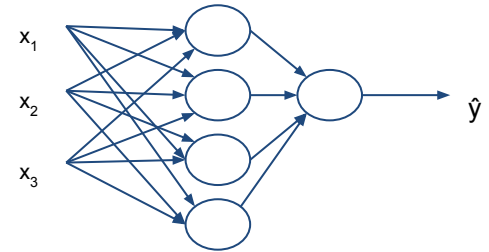
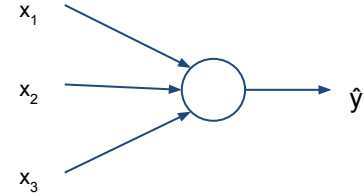
# It's a matter of layers

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

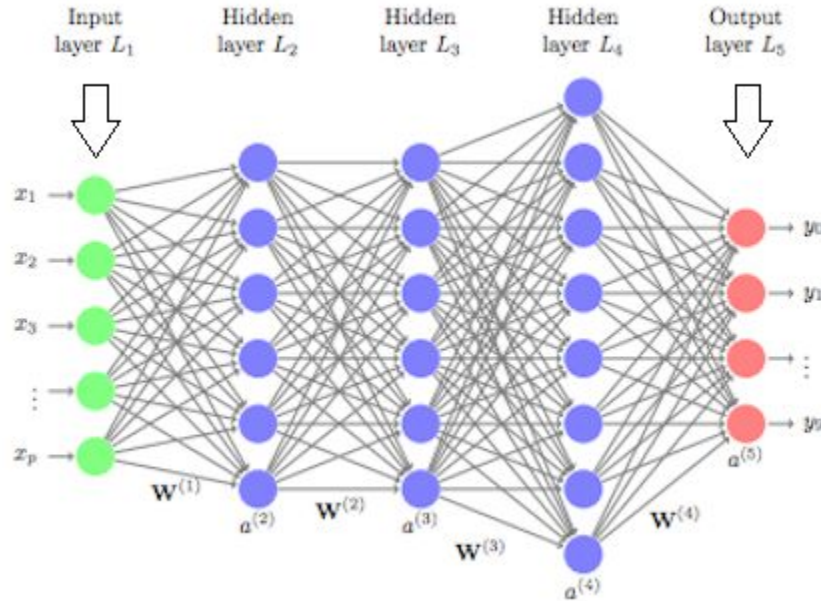


# It's a matter of layers

- 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

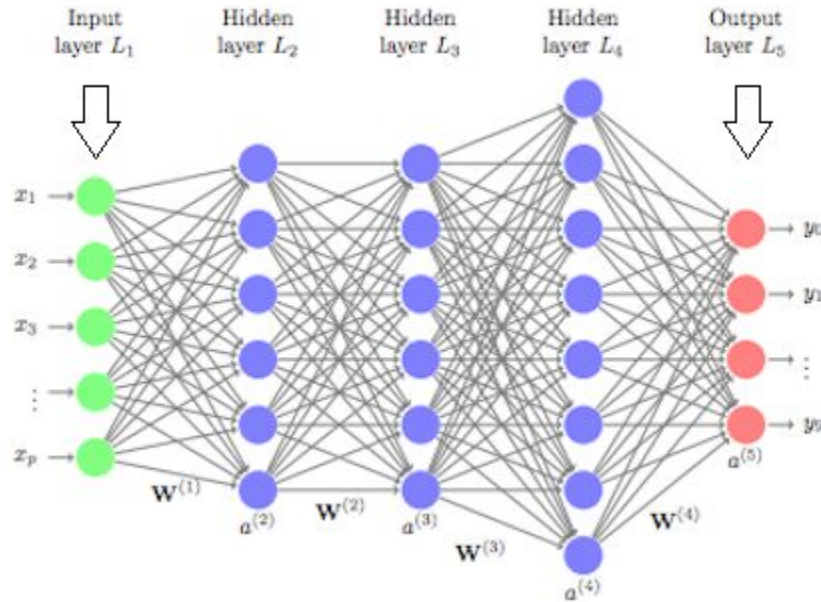


- 4 layers (“deep” NN)
- n. of layers in a deep learning model: hyperparameter to tune (one of many)

Source: University of Cincinnati



# Layers matter!

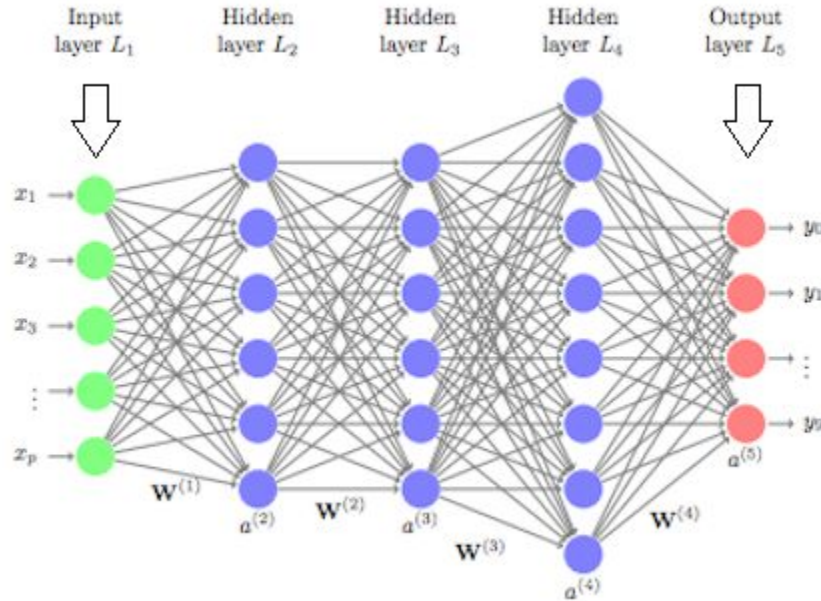


Source: University of Cincinnati

- 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 (1 in 1 to  $L$ )
- $X$ : matrix of input features  $\rightarrow A^{[0]}$

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

- for each layer
- iterate over n. of layers (unavoidable for loop)
  - N. nodes  $\rightarrow$  memory
  - N. layers  $\rightarrow$  CPU



# Compositional representation

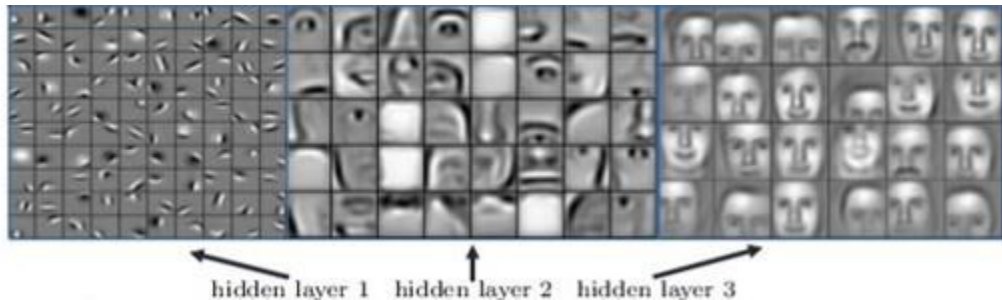
- why do deep NNs work better?
- **each layer** focuses on **one representation** of the data
- representations are then **combined** to get the final result → **compositional representation**

Simplified example from face recognition:

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



[mind you: this is just intuition]



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:
  - deep NN  $\rightarrow O(\log(n))$
  - wide NN  $\rightarrow O(2^n)$
- deep NN can learn functions that can not be learnt by shallow networks



# 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  $\alpha$
  - 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

- lab 5

→ day2\_code02 keras shallow neural networks.ipynb



# Neural networks models: recap

- students' (collaborative) exercise n. 1 (write your own code, in groups or independently)

→ `day2_code03_neural_networks_[EXERCISE].ipynb`

