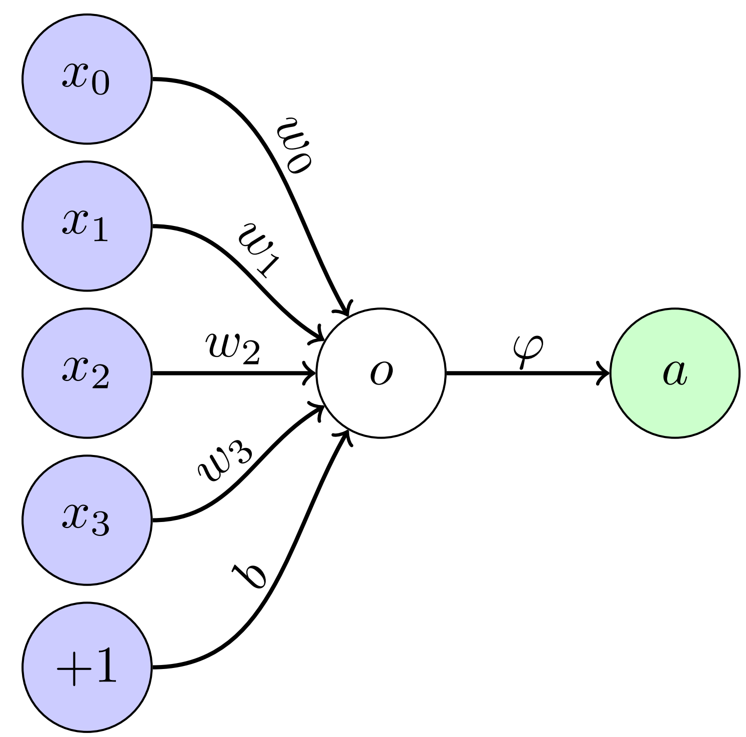
Deep Learning
2. Multi Layer Perceptrons

A course @EDHEC
by Romain Tavenard (Prof. @Univ. Rennes 2)

Limitations of the Perceptron

- Input: $x = \{x_0, \ldots, x_D\}$
- Output: $a = \varphi(\sum_j w_j x_j + b)$

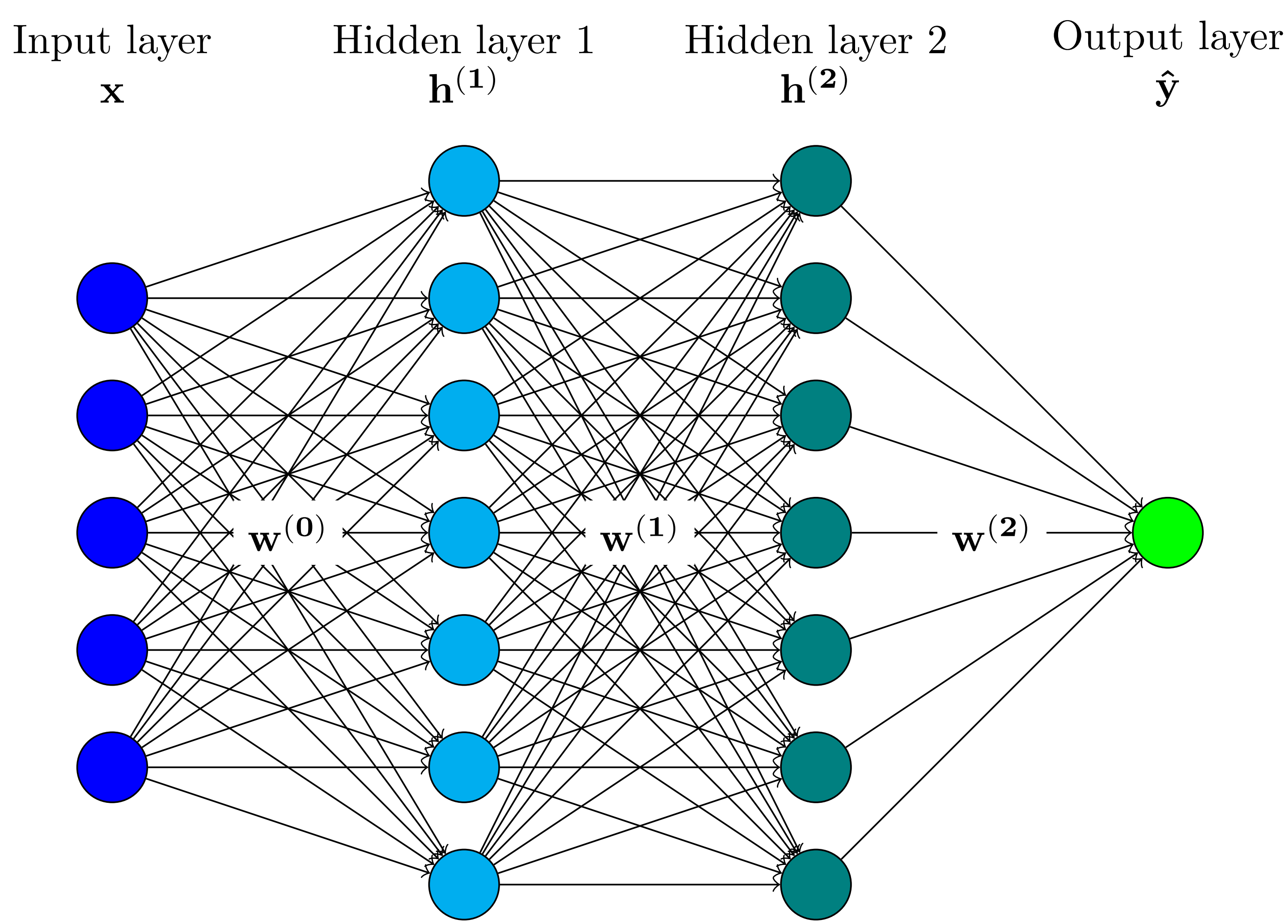


- Can do
  - Linear regression
  - Linear boundaries for classification
  - Single output

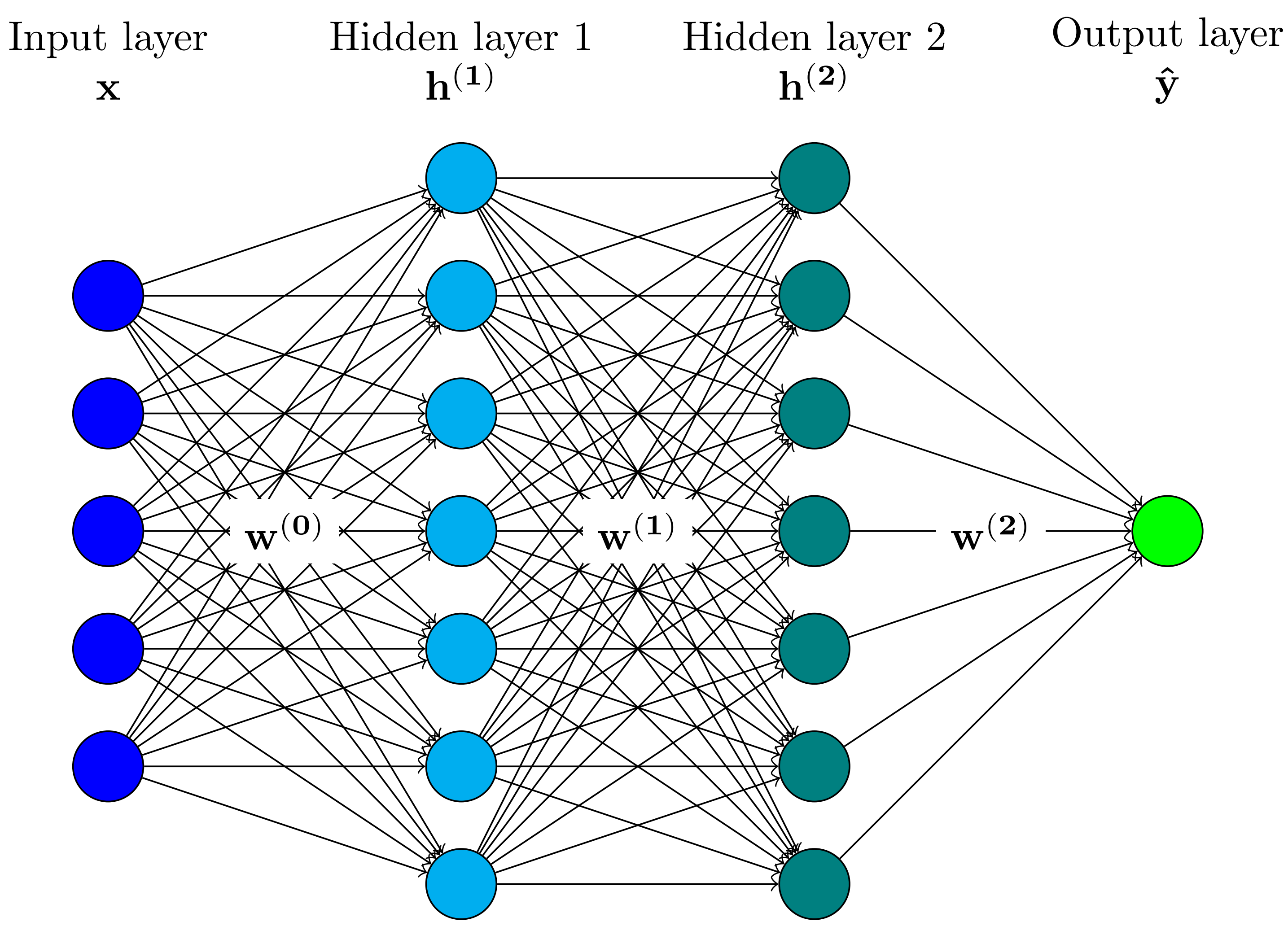
Multi-Layer Perceptron (MLP) model (Rumelhart, Hinton & Williams, 1985)

Definition

A Multilayer perceptron is an acyclic graph of neurons, where neurons are structured in successive layers, beginning by an input layer and finishing with an output layer.



Multi-Layer Perceptron (MLP) model
(Rumelhart, Hinton & Williams, 1985)



$$\hat{y} = \varphi_{\text{out}} \left(\sum_i w_i^{(2)} h_i^{(2)} + b^{(2)} \right)$$

$$\forall i, h_i^{(2)} = \varphi \left(\sum_j w_{ij}^{(1)} h_j^{(1)} + b_i^{(1)} \right)$$

$$\forall i, h_i^{(1)} = \varphi \left(\sum_j w_{ij}^{(0)} x_j + b_i^{(0)} \right)$$

Why introduce hidden layer(s)? Universal approximation theorem (Cybenko, 1989)

- Under reasonable assumptions on the activation function to be used*
- For any continuous function on a compact g and any precision threshold ε
- There exists a 1-hidden-layer MLP with a finite number of neurons that can approximate g at level ε

*Non-constant, bounded, monotonically increasing, continuous
also holds for some unbounded functions like ReLU, cf. [Somoda & Murata, 2015]

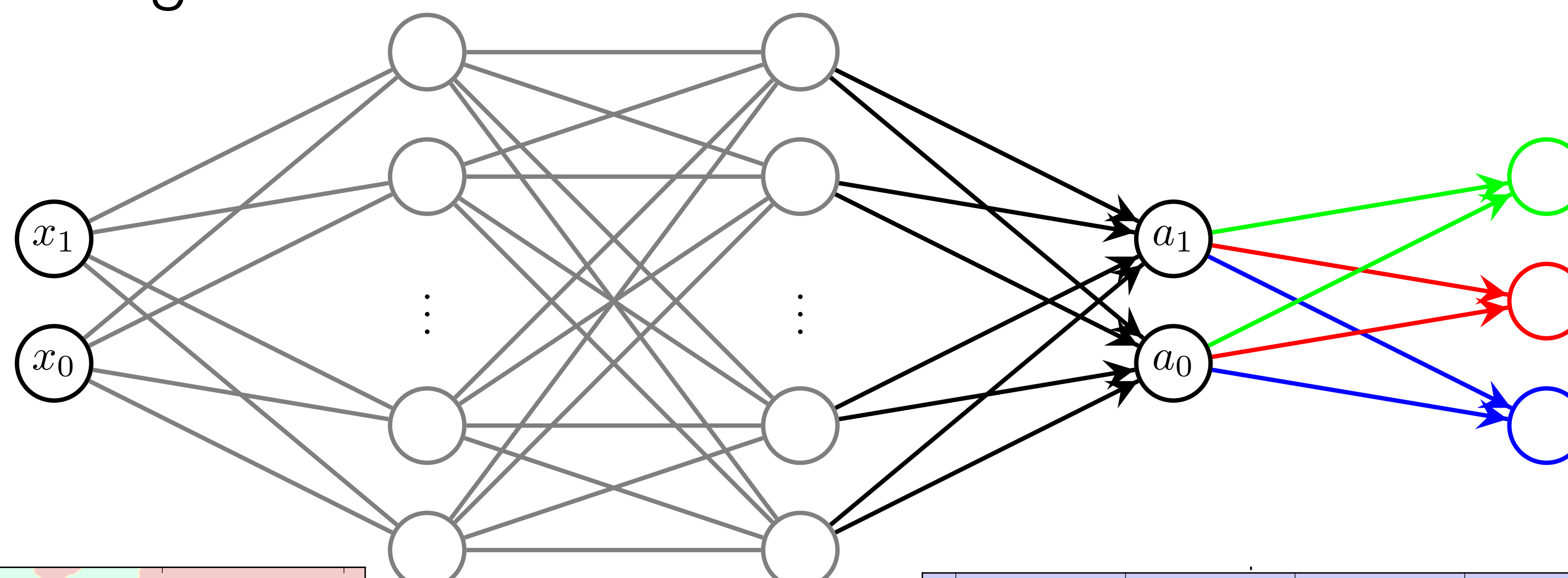
Why introduce hidden layer(s)? In practice

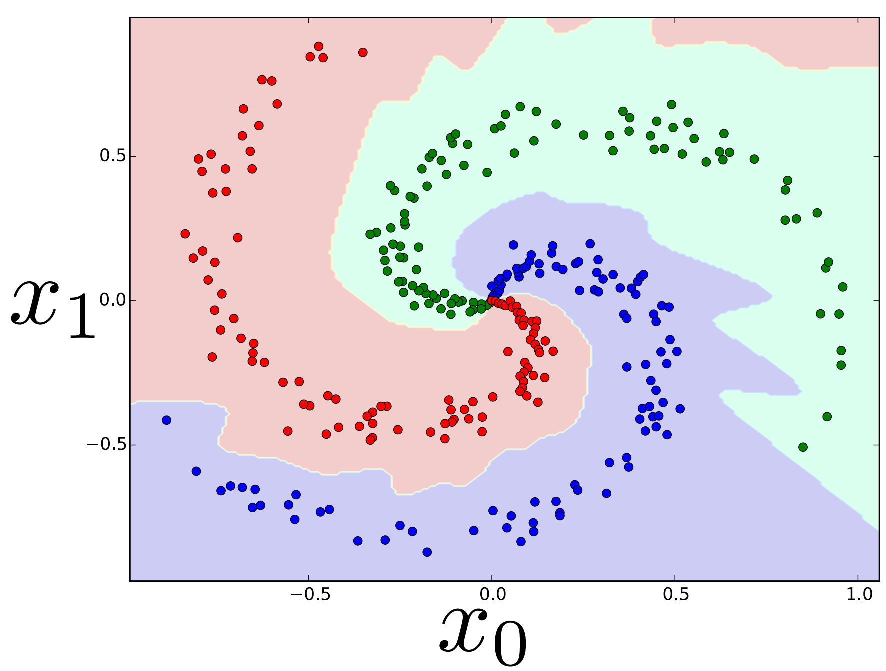
- Universal Approximation Theorem:
  - A single hidden layer is sufficient in theory
  - The number of neurons in this layer is not given

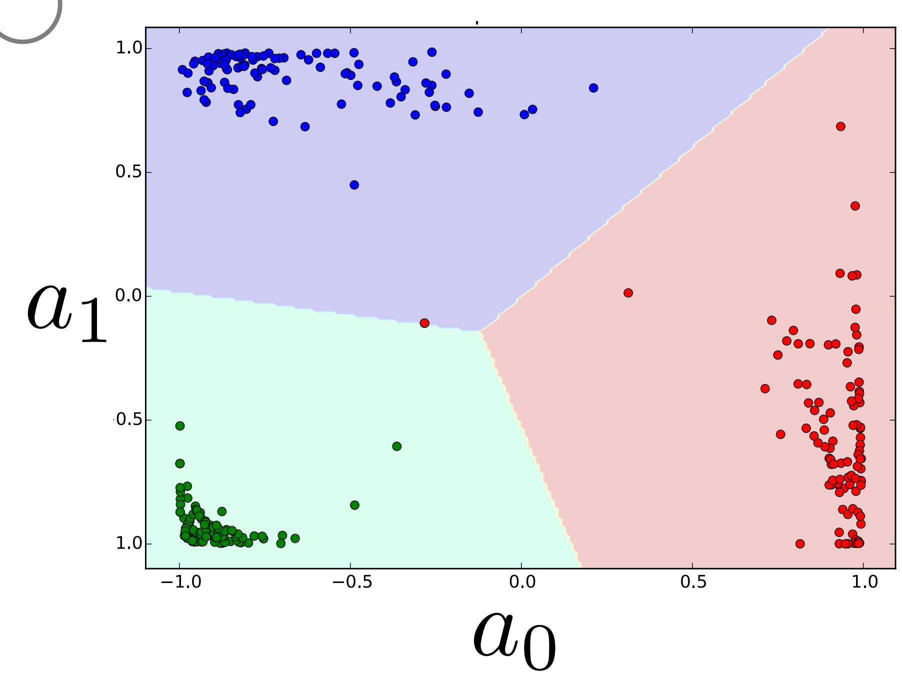
- In practice, for a given approximation level
  - Stacking more layers requires less parameters
  - ⚠ Very deep networks suffer from specific problems too (discussed later in the course)
  - 2-3 hidden layers is a good start for an MLP

End-to-end learning

- Classification using MLP
  - Hidden layers: non-linear transformations
  - Last layer: logistic regression
- Example

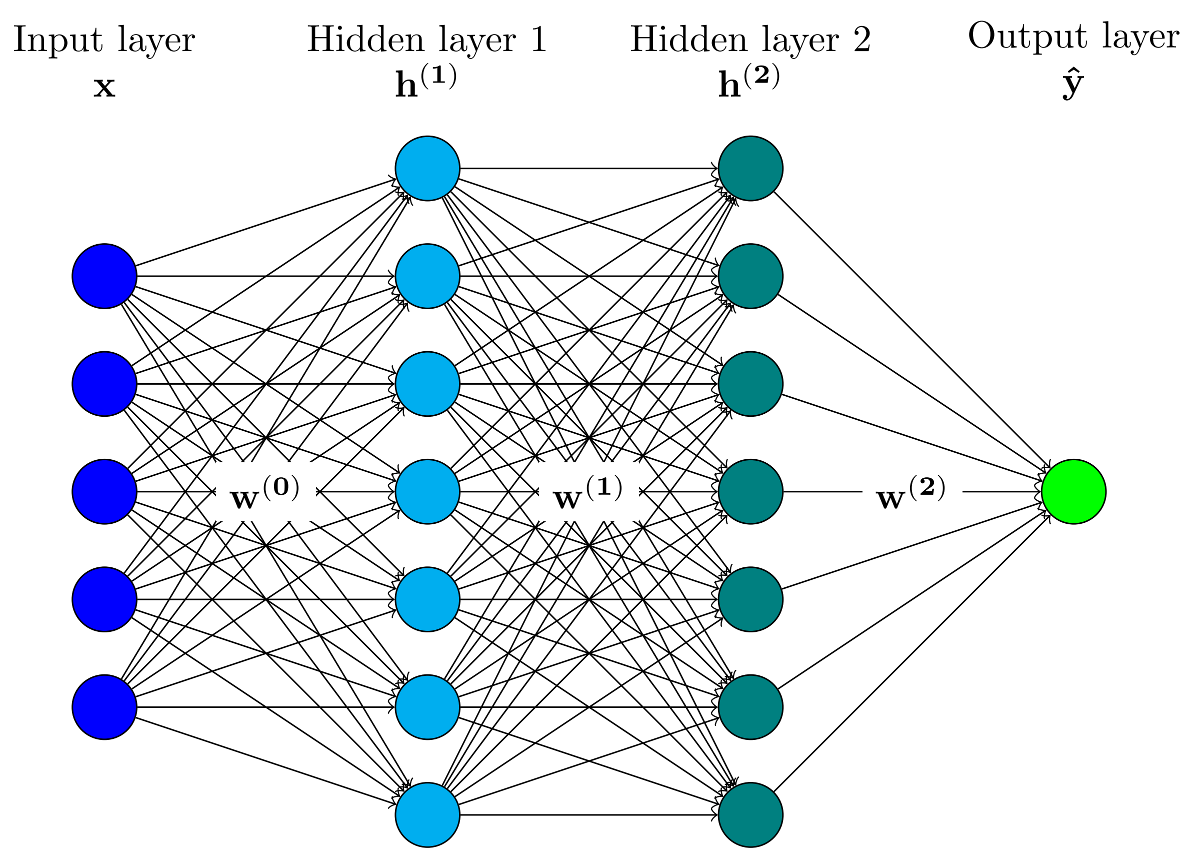






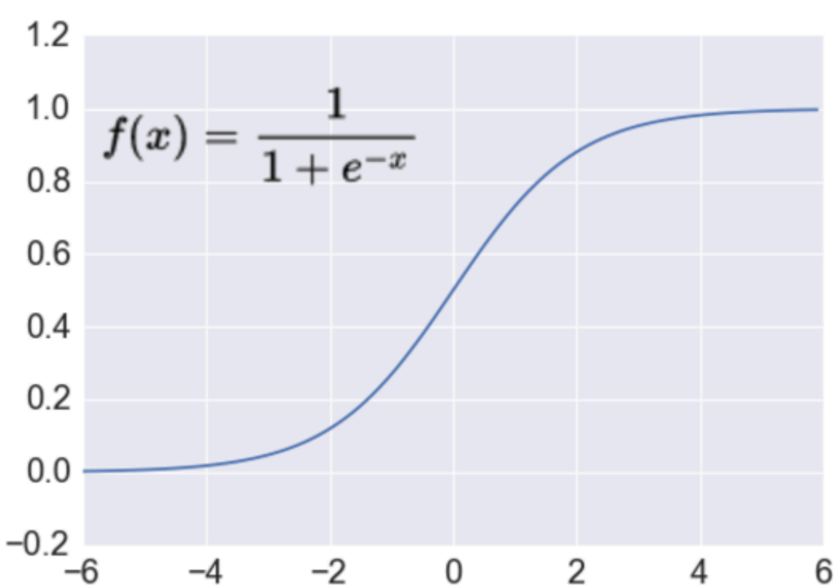
Building blocks of neural networks
Input/Output layers

- Given a dataset (X, y)
- Constraints on model structure:
  - Input layer dimension is the number of features in X
  - Output layer has as many units as columns in y

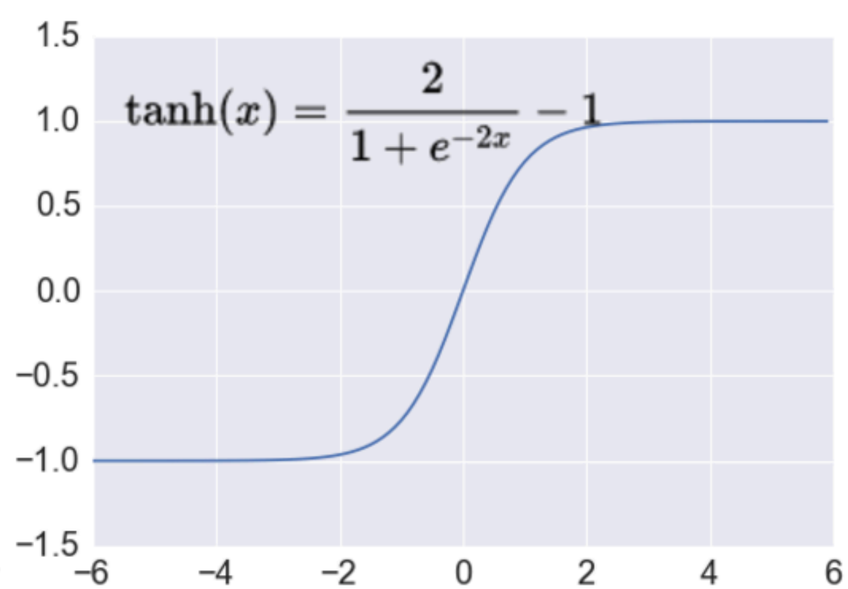


Building blocks of neural networks
Activation functions

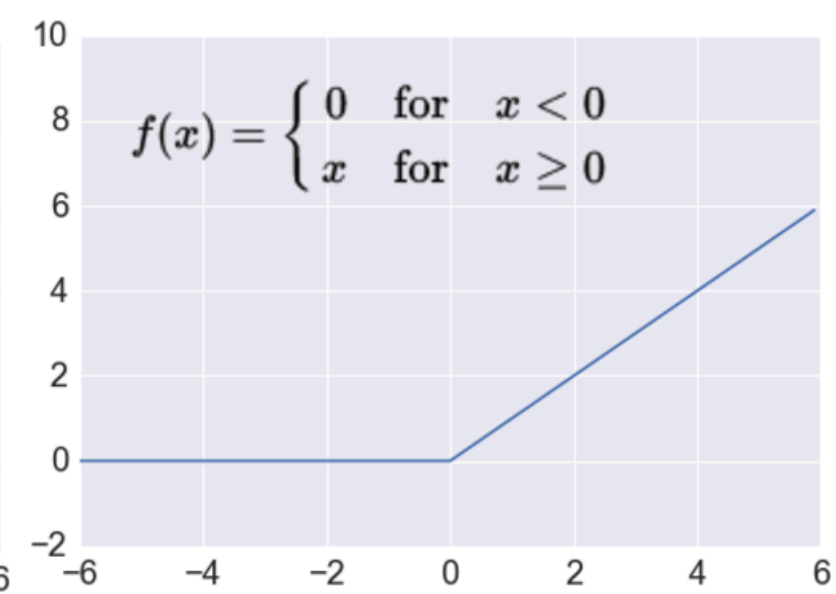
- Important features
  - φ should be differentiable almost everywhere
  - Non-linearities
  - Some linear regime
- Examples



sigmoid



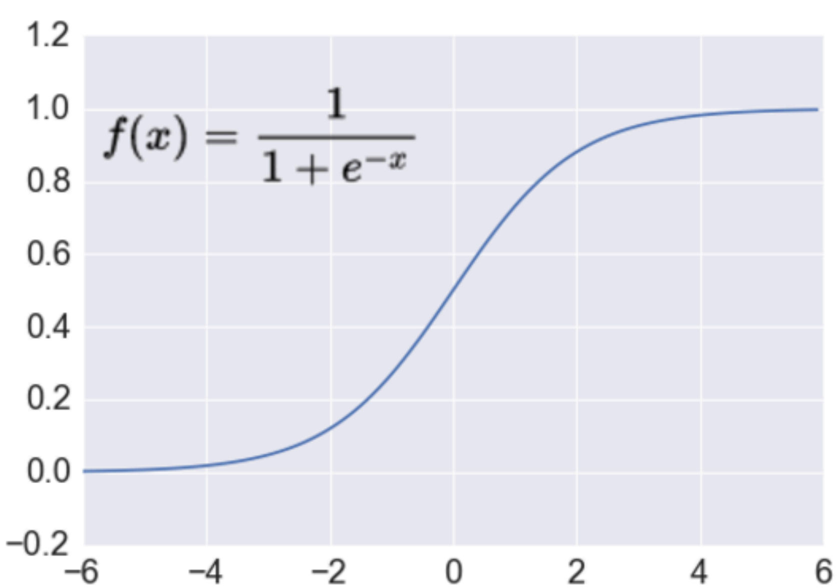
tanh



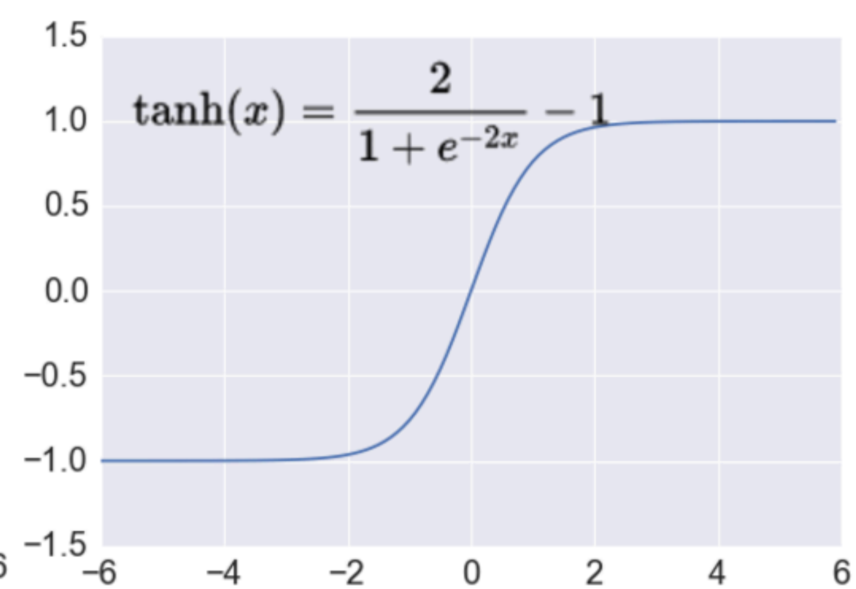
ReLU

Building blocks of neural networks
Activation functions: the reign of ReLU

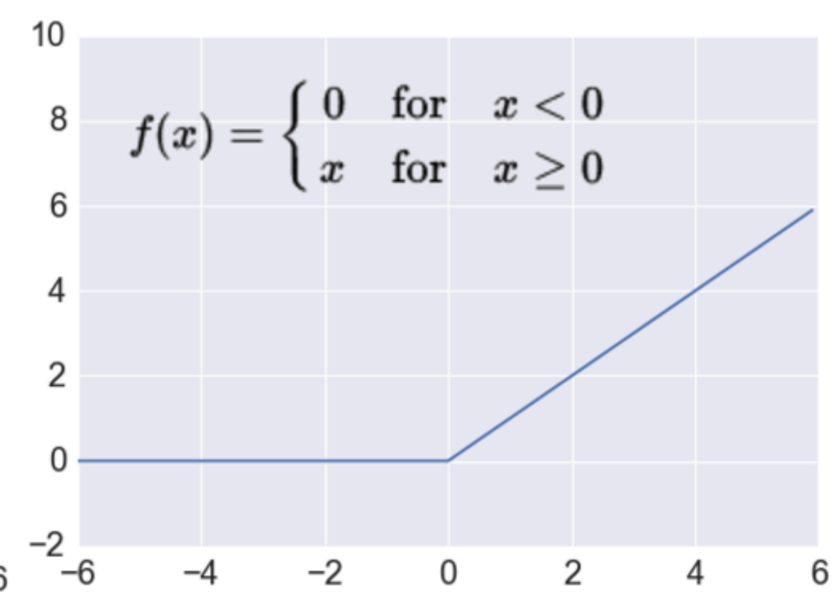
- ReLU has become the default choice **for internal layers** over time
- 2 main reasons:
  - cheap to compute (both ReLU and its derivative)
  - vanishing gradients phenomenon (more on that later)



sigmoid



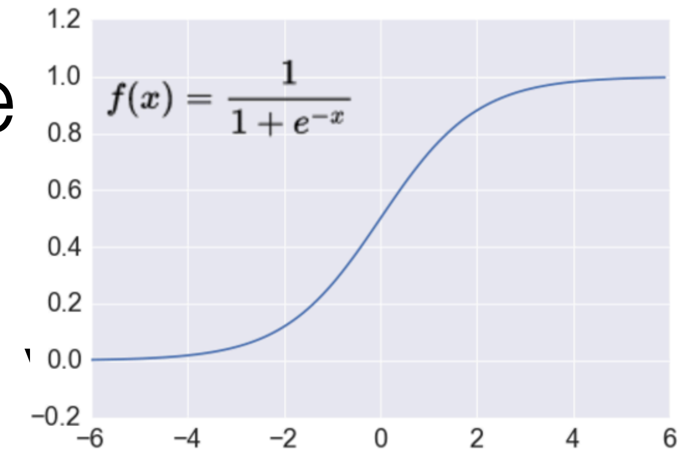
tanh



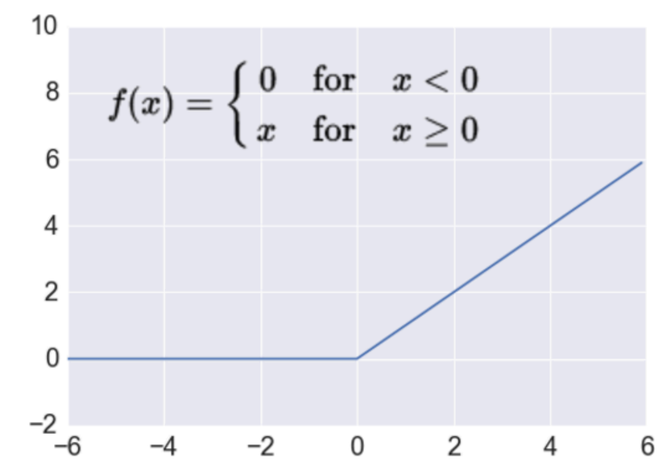
ReLU

Building blocks of neural networks
Activation functions: the case of the output layer

- Output activation functions drive the values:
    - identity ("linear" in keras): any real
    - ReLU:
      any positive value
    - sigmoid:
      any value in [0, 1]
    - softmax:
      >0 and sums to 1
      (across output neurons)



sigmoid



ReLU

$$\text{soft-max}(o)_i = \frac{e^{o_i}}{\sum_j e^{o_j}}$$