Introduction to deep learning

Florian Ruppin -- *florian.ruppin@univ-lyon1.fr*

Université Claude Bernard Lyon 1, Institut de Physique des 2 Infinis de Lyon

May 16, 2024

Classification -- neural networks

Classification -- neural networks

Automatic classification -- *non-linear function*

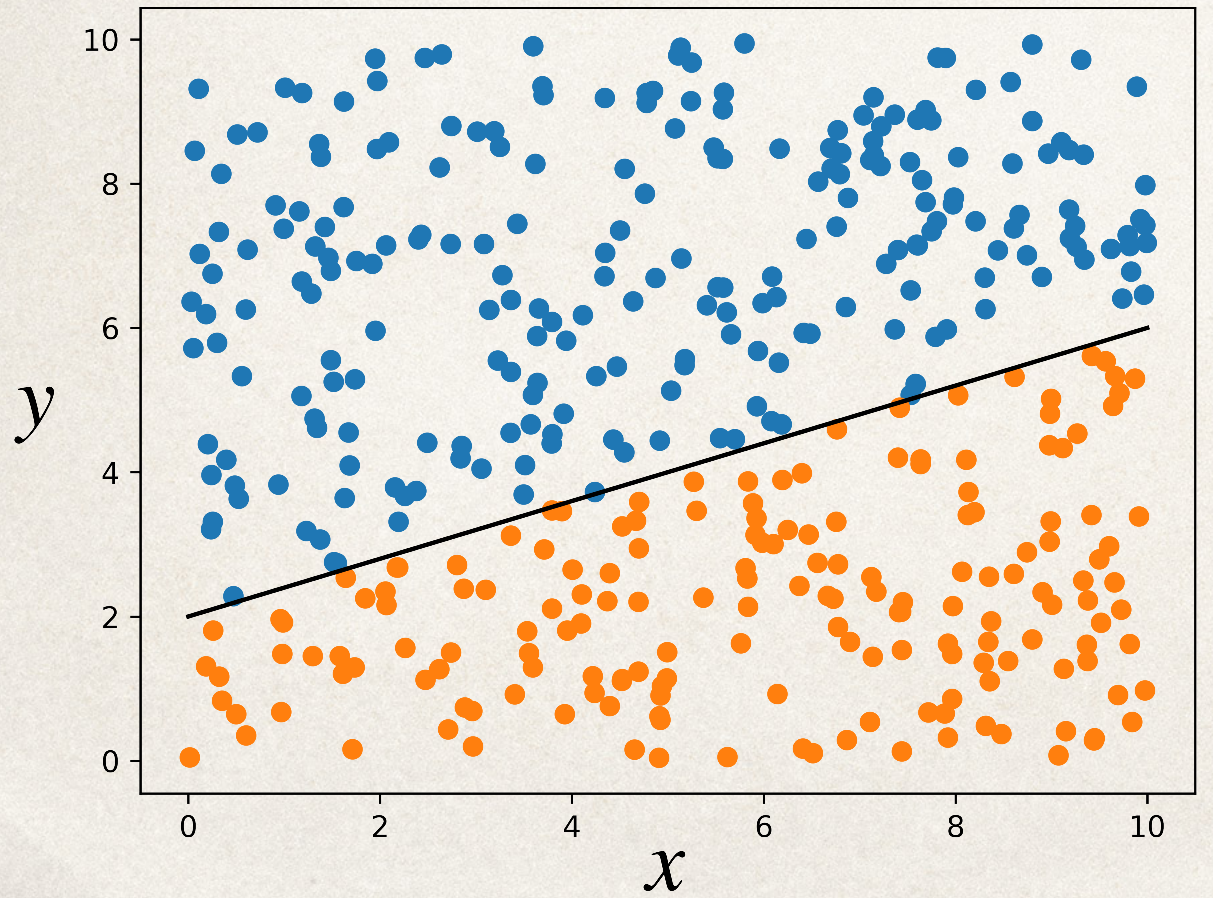
- Many fields require the use of algorithms to classify large datasets
  Examples: medicine, fundamental research *(physics, chemistry, biology, humanities, etc.)*, big-data company, etc.

- Simple cases: classification can be performed by a simple cutoff with respect to a linear function

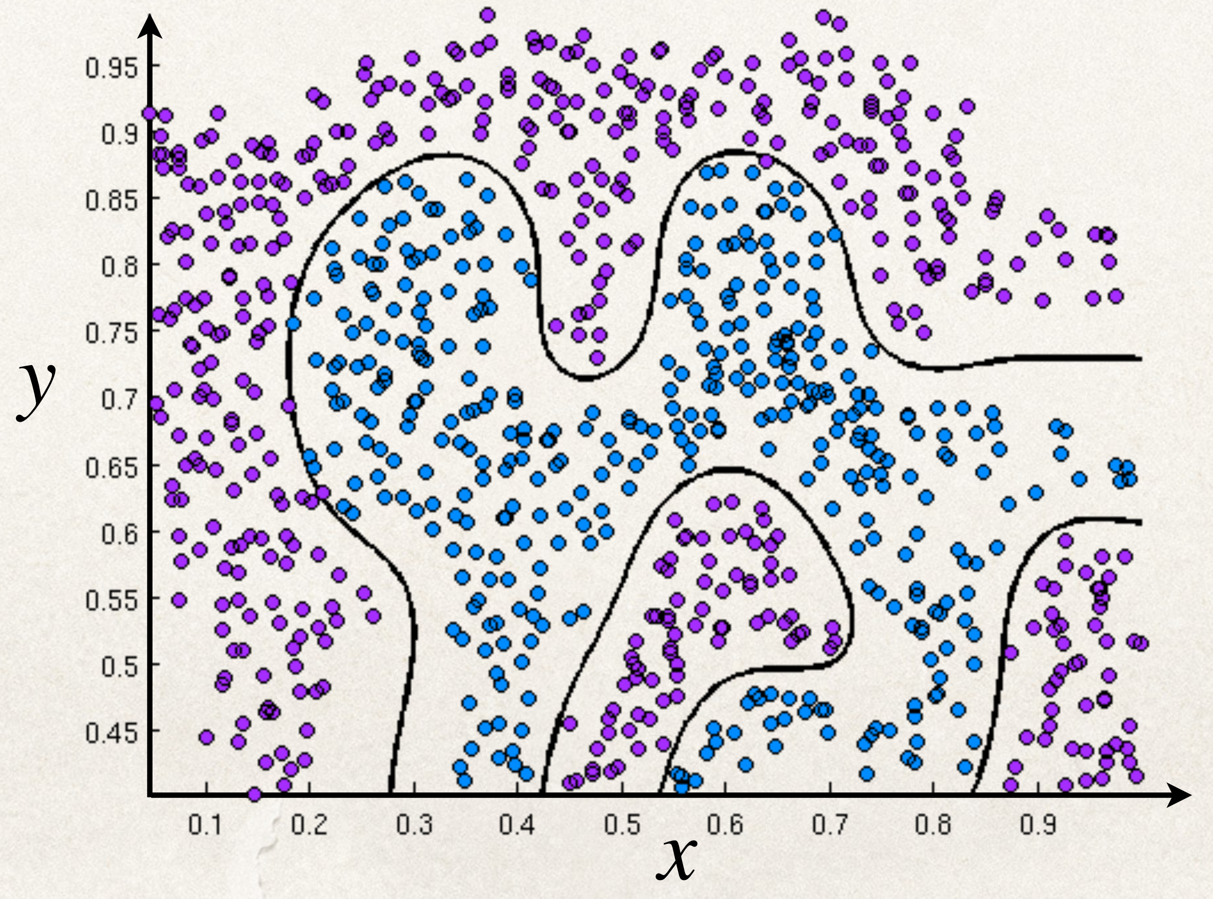
- Current problems: - the categories studied are defined by a large number of variables
  - the quality of the data is not sufficient to associate an unambiguous category *(contamination)*
  - the function for optimal data classification is highly non-linear

- Visual inspection relies on knowledge of a large number of properties *(noise + signal)*
  Disadvantages: often involves a single person *(errors / biases)* + cannot be considered for large data sets

Linear classification



1D non-linear classification



nD non-linear classification

Classification -- neural networks

Training a classifier -- *supervised or not*

Objective: We want to find the optimal function for separating our data into categories *(called class labels)*

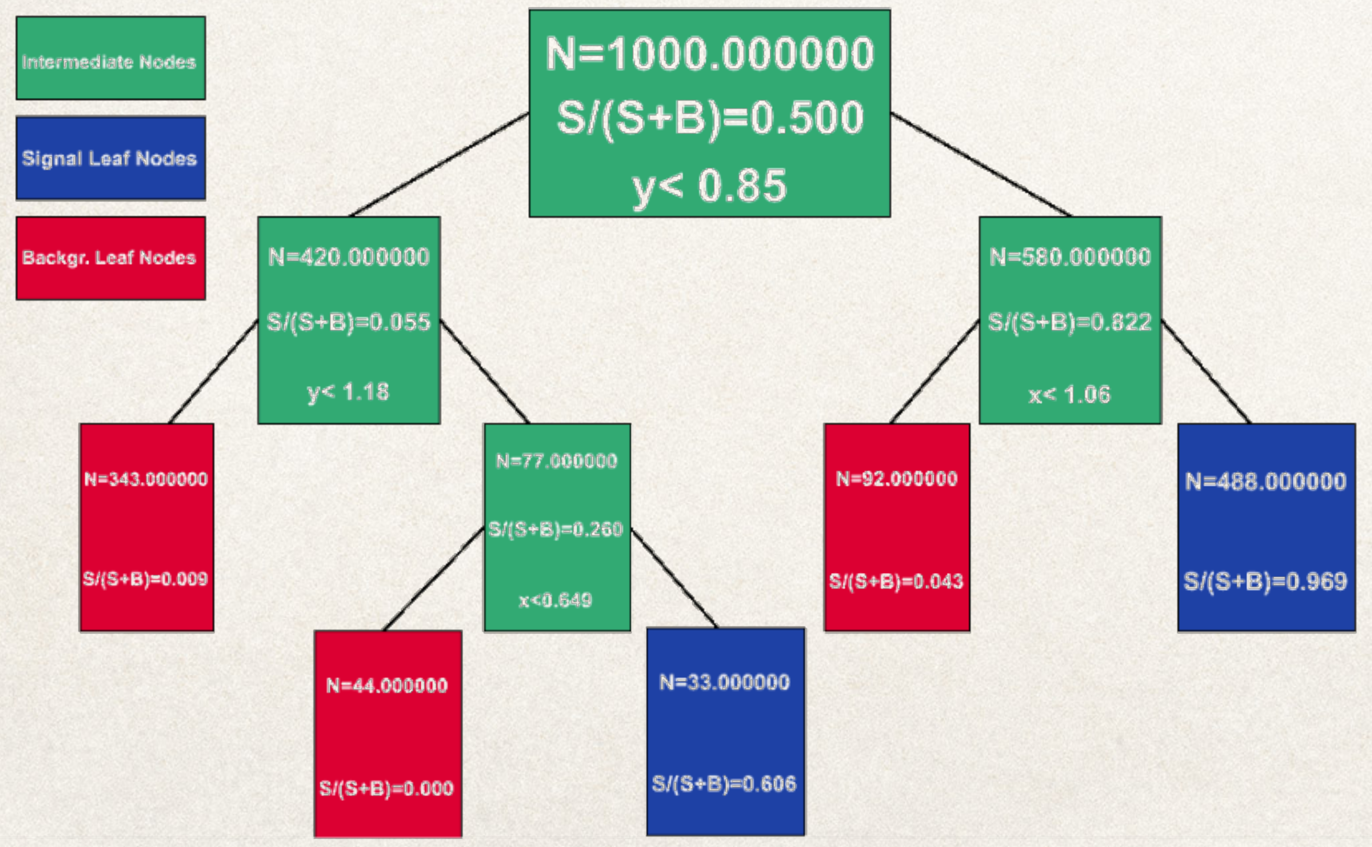
Supervised approach:

- Use of a training sample in which each data vector has a known class label
- Decision tree: we define a set of variables on which to apply successive cuts in order to achieve a minimum contamination rate for each class label
- Neural network: we define an architecture of interconnected neurons *(automata with a transfer function)* and exploit the potential complexity of the corresponding function *(synaptic plasticity)* to optimize our classification.

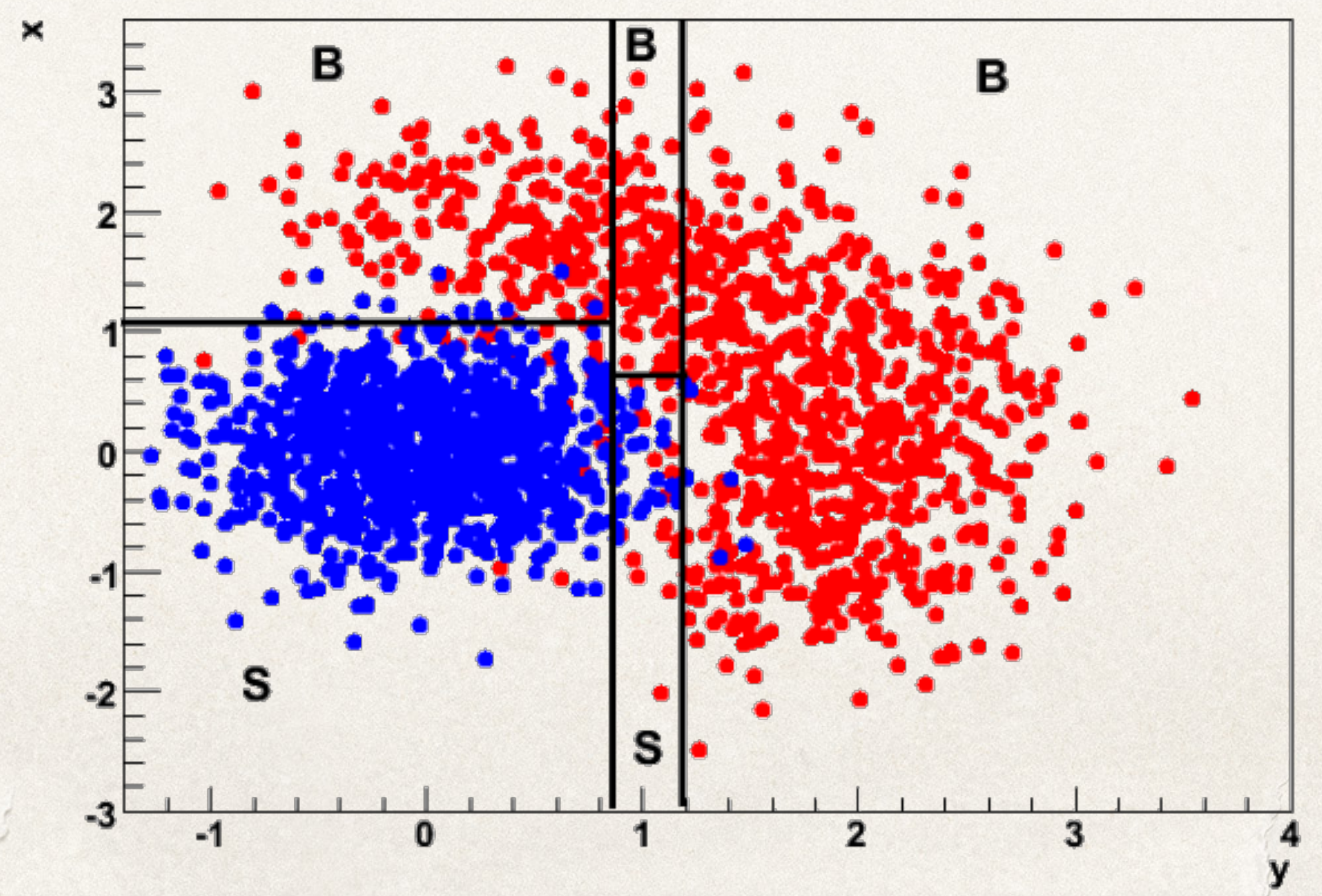
Unsupervised approach:

- The class labels of the training sample are not defined. The neural network groups together data with similar features

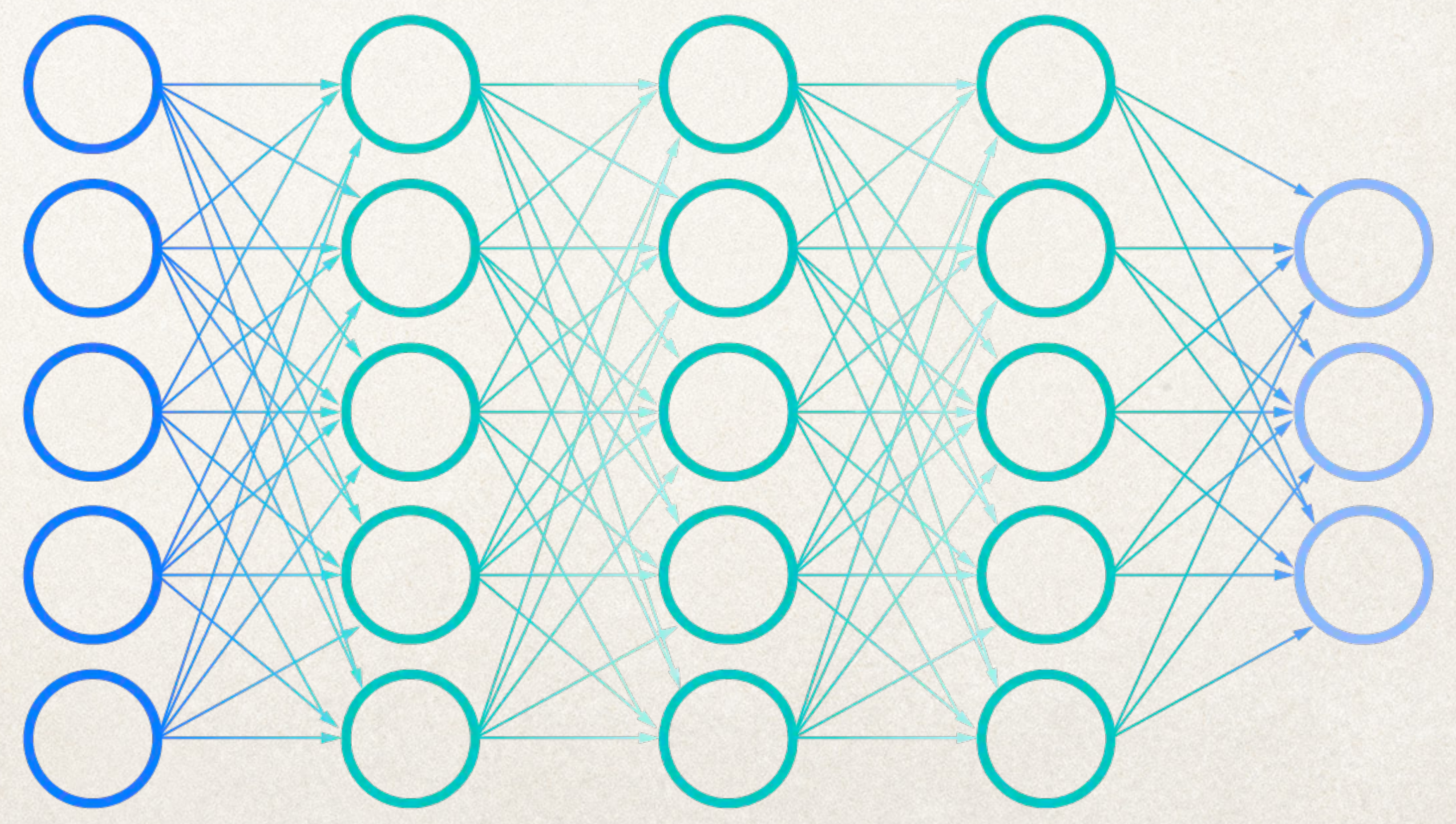
Decision tree



Associated cuts



Neural network

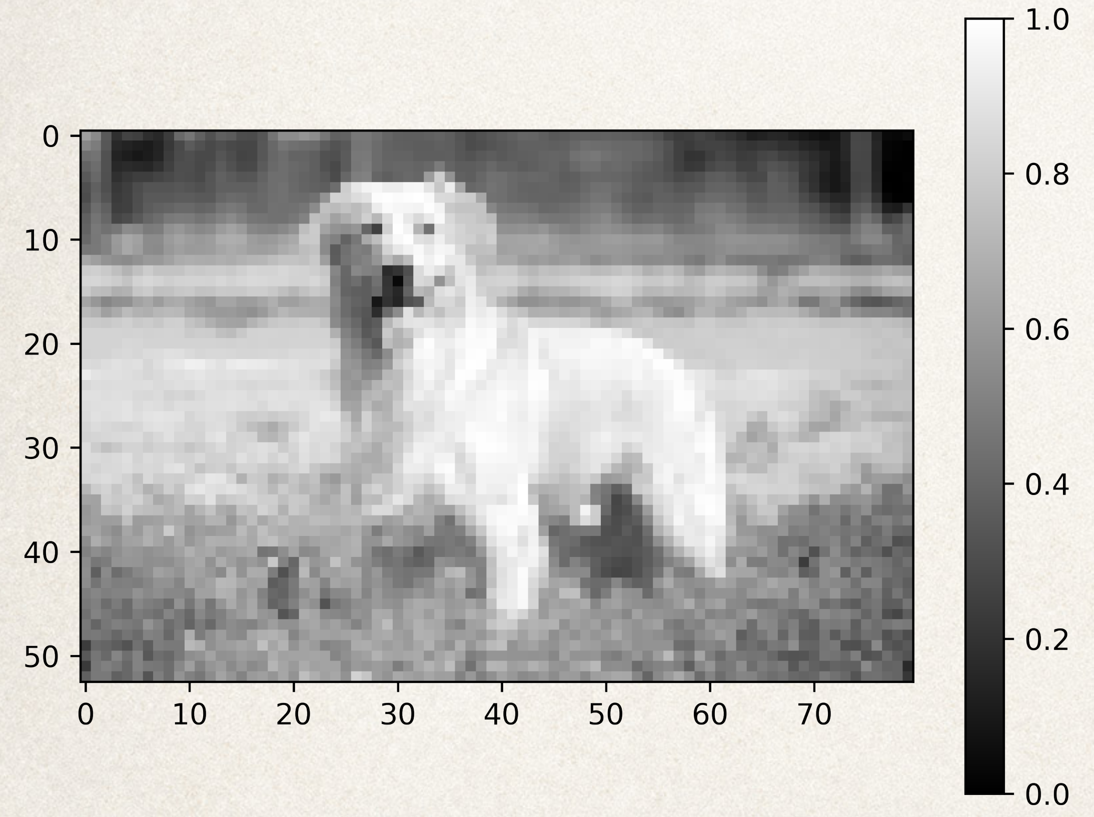


Classification -- neural networks

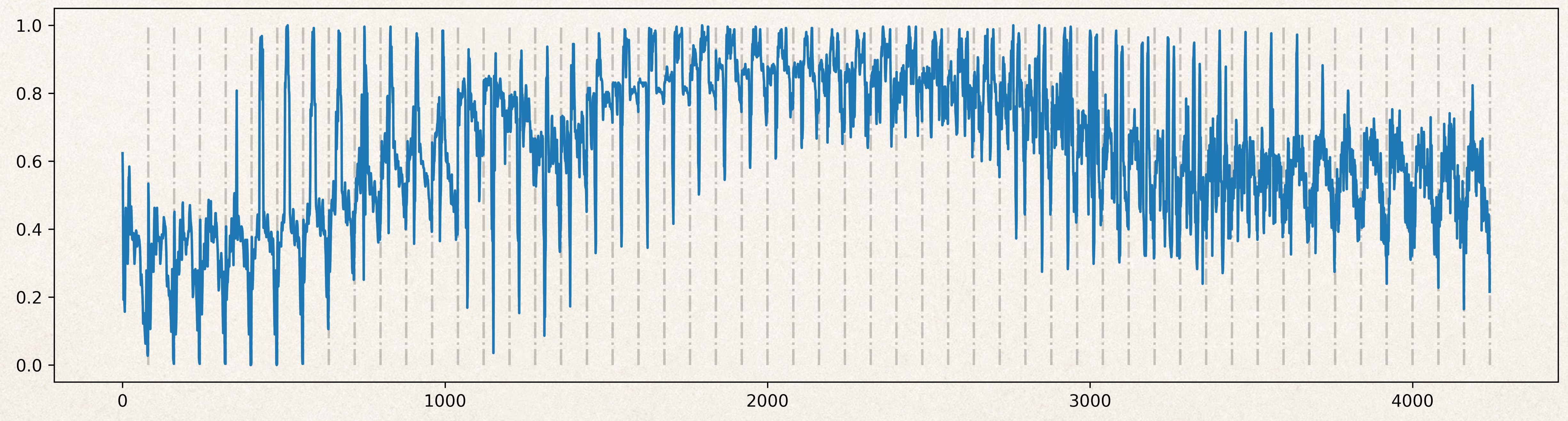
Neural network architecture -- *multilayer perceptron*

- Three types of layer: one input, one or more hidden layers, one output
- Input: the matrix corresponding to a data sample is unfolded as a vector *(each neuron contains a value)*
- Hidden layer: each neuron receives a linear combination of the outputs of all the neurons in the previous layer

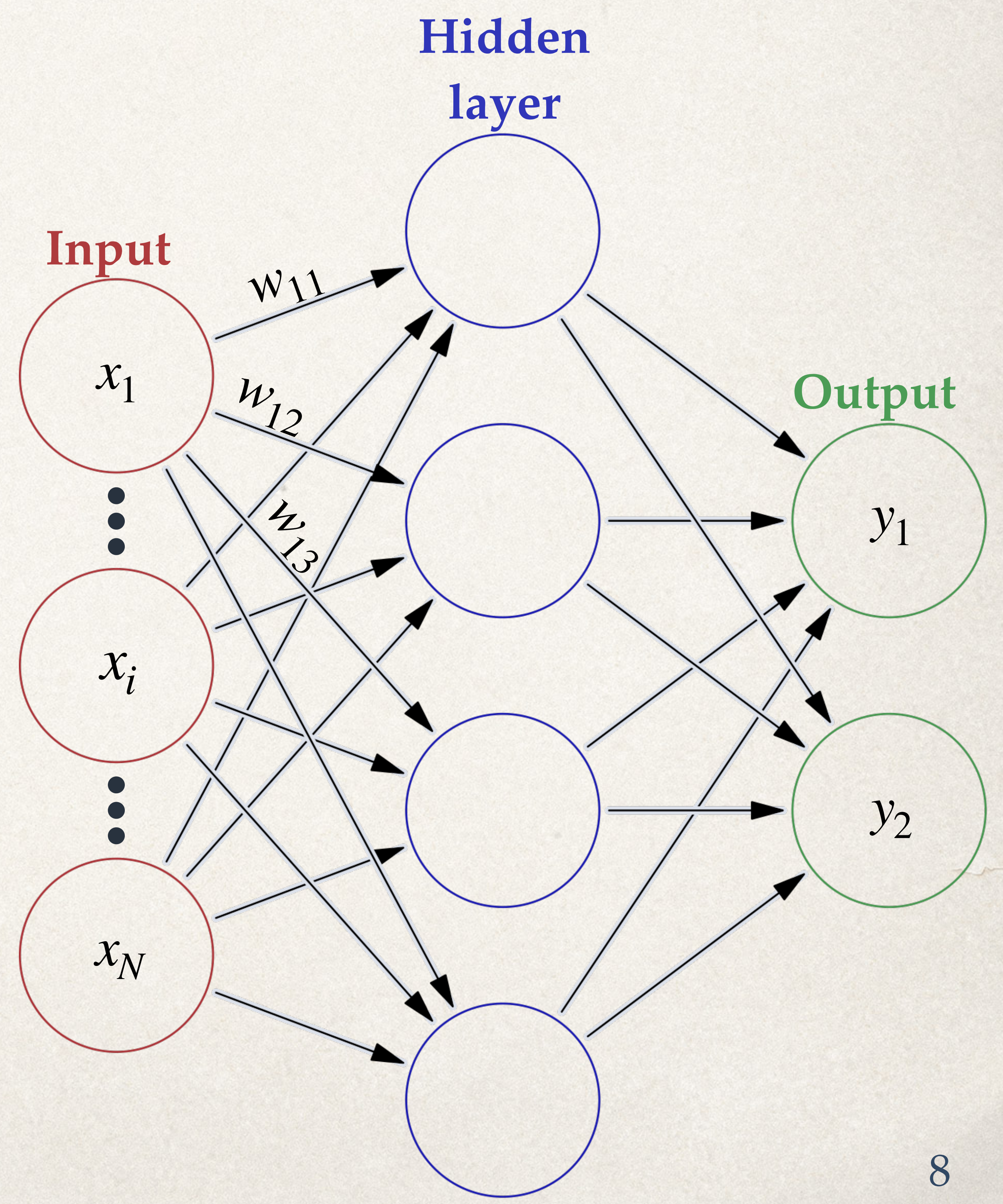
  $\longrightarrow$ Each pixel in the image is multiplied by a weight and summed to the other products
- Each hidden layer neuron applies a non-linear function to the linear input combination
- Output: each neuron in the output layer contains the probability of belonging to a category



Black and white image (53 x 80 x 1)



Associated vector (4240 values between 0 and 1)

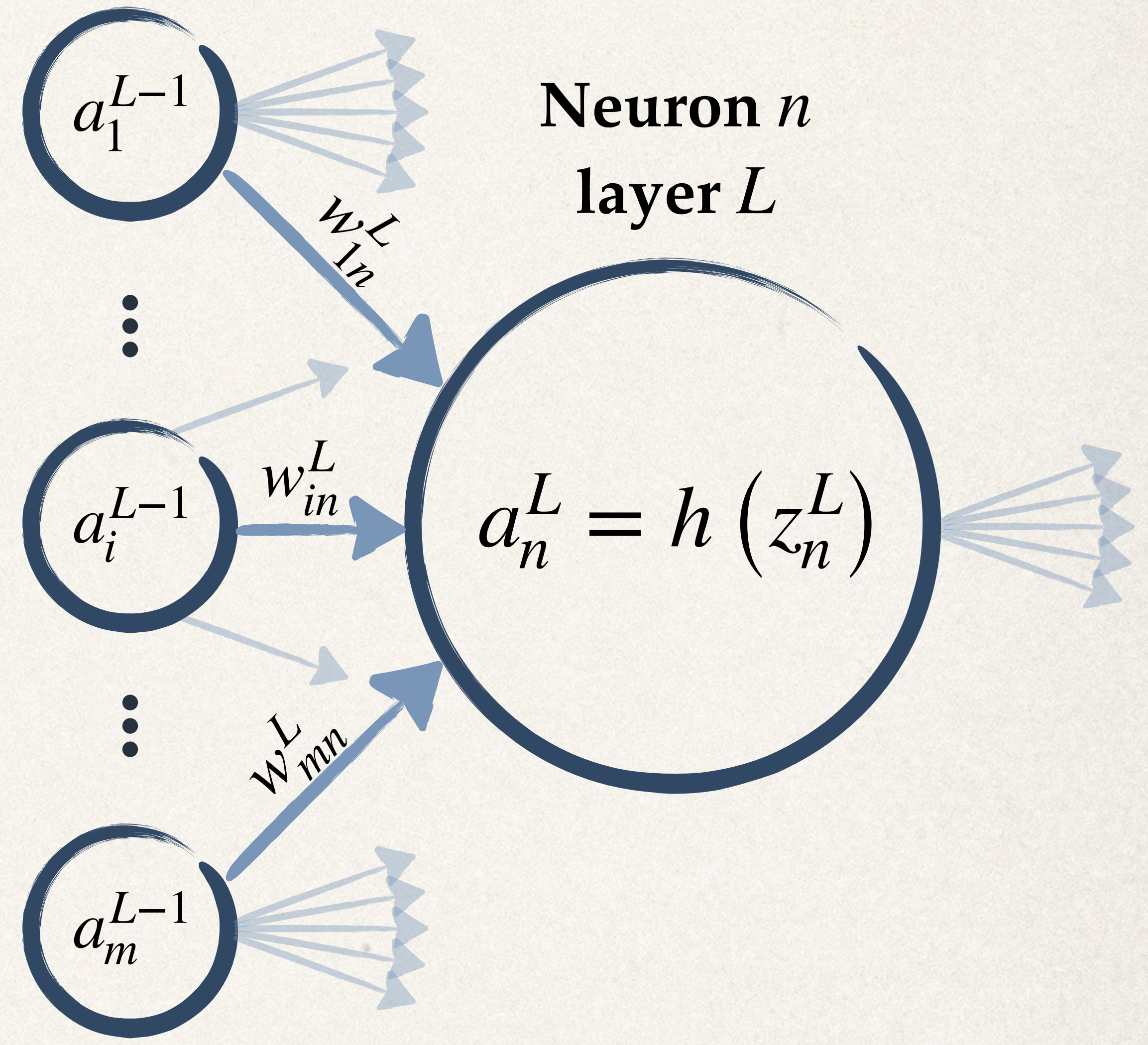


Classification -- neural networks

Forward propagation -- *notations*

Notations per neural network element:

- We call w^L_{mn} the weight associated with the link between neuron m of layer $L-1$ and neuron n of layer L
- The activation function is called h and the result of this function returned by neuron n of layer L is called a^L_n
- The bias associated with neuron n of layer L is called b^L_n
- We thus have: $a^L_n = h\left(\sum_i w^L_{in} a^{L-1}_i + b^L_n\right)$
- We call $z^L_n = \sum_i w^L_{in} a^{L-1}_i + b^L_n$ the linear combination at the input of neuron n of layer L.
  Thus, its output is simply given by $a^L_n = h(z^L_n)$



Matrix notation to generalize the above equations to all neurons in the same layer:

- We call W^L the matrix of weights from layer $L-1$ to layer L. It is therefore an $M \times N$ matrix where M and N are the number of neurons in layers $L-1$ and L. Thus, $w^L_{mn} = W^L[m,n]$
- We call b^L the bias vector of layer L. Thus, $b^L_n = b^L[n]$
- We call a^L the activation vector of layer L. Thus, $a^L_n = a^L[n]$
- We call z^L the vector of arguments of the activation function of layer L. Thus, $z^L_n = z^L[n]$

Forward propagation -- *activation functions*

- With these notations we can generalize the definition of z_n^L for each neuron of layer L:
  $z^L = (W^L)^T a^{L-1} + b^L$ and thus, $a^L = h(z^L)$

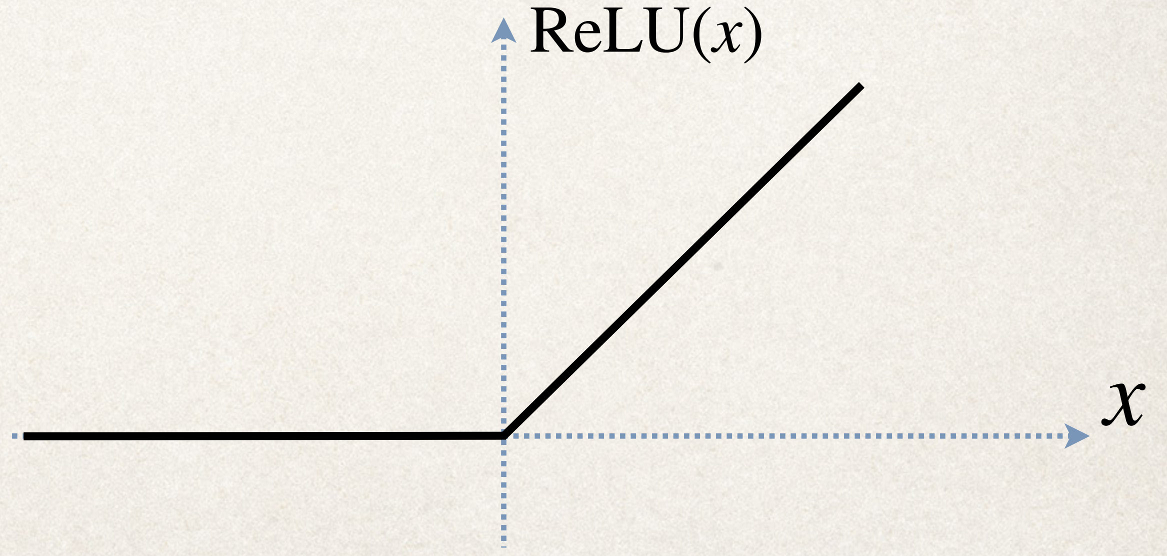
Note: We don't assign $L = 0$ to the input layer because then we'd have a weight matrix W^0 and a bias vector b^0 that aren't defined for the input layer, which has no prior weights or bias per neuron. The layer $L = 0$ is therefore the first hidden layer and we have to replace a^{L-1} by x, the data vector, in the previous equation.

- We call H the output layer ($L = H$). The layer $L = H - 1$ is therefore the last hidden layer and we thus have H hidden layers.

Activation functions:

- The activation function most commonly used in deep neural networks is ReLU *(rectified linear unit)* defined by:
  $f(x) = \max(0, x)$

  $\longrightarrow$ Intrinsically non-linear, obvious derivative, sparse activation

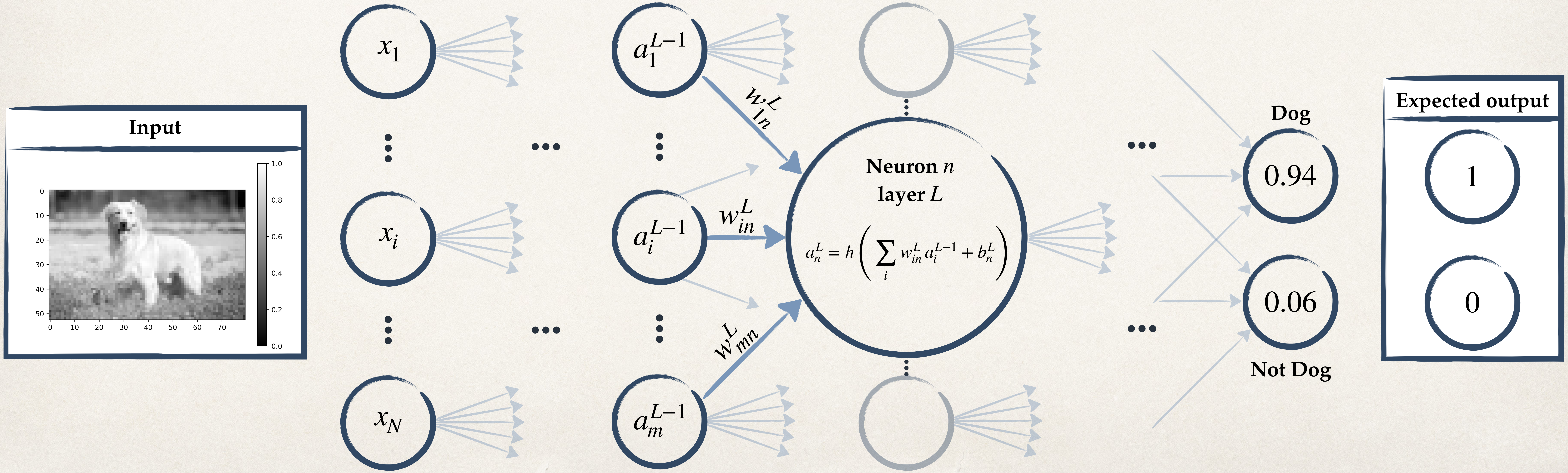


- The activation function of the output layer is often the softmax function defined by:
  $$\sigma(\vec{z})_j = \frac{e^{z_j}}{\sum_{i=1}^{N} e^{z_i}} \quad \text{for } j \in [1, \ldots, N]$$

  $\longrightarrow$ Discrete probability law on N different results

Forward propagation -- *principle*

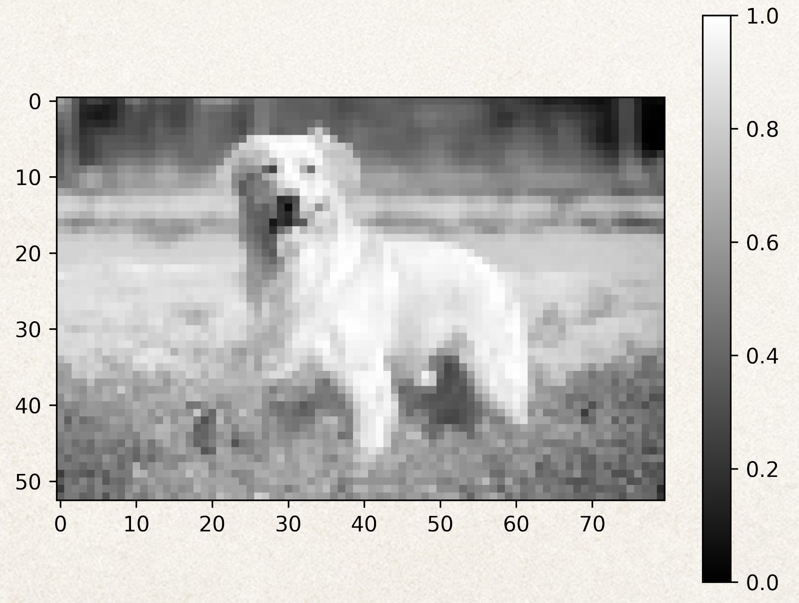
- For each pair (Input$_i$, Expected output$_i$)$_{i \in [1,\dots,D]}$, the network is run from left to right by performing the following operations
- Multiply input values by weights, sum results, add biases, apply activation function, pass to next neurons
- This propagation results in values between 0 and 1 inside the output neurons *(if the softmax function is used as output)*
- We therefore obtain a result in the neurons of the output layer that can be compared with the actual class label of the input data

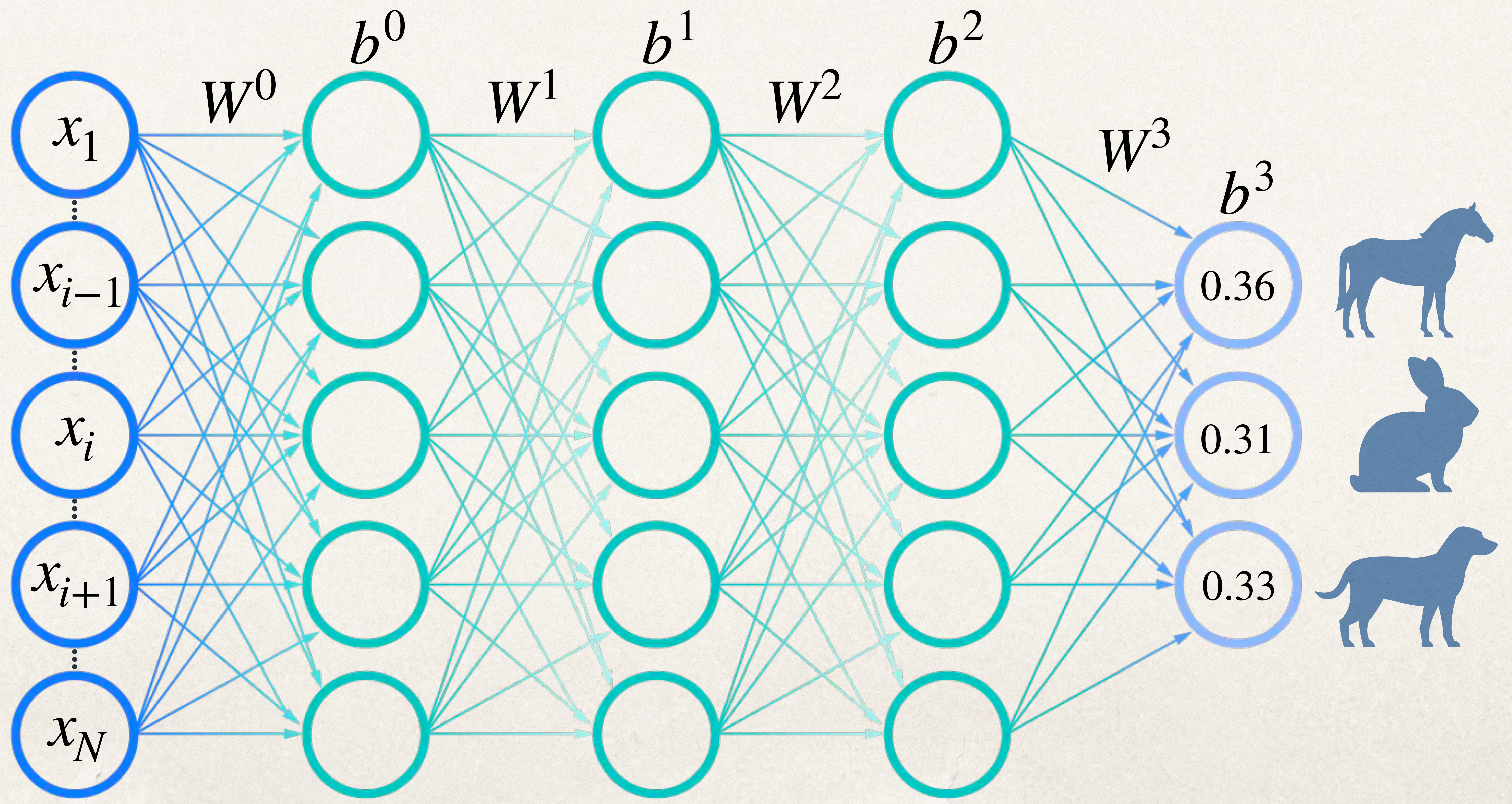


Classification -- neural networks

Backpropagation -- *principle*

- Initial state: weights and biases are randomly drawn $\longrightarrow$ output neurons have equivalent values
  Classification success rate is low *(random drawing of a class label)*
- We want to find the best weights and biases to maximize the classification success rate on the training sample
- In most cases, the number of free parameters of the problem is $\gg 10^4$ $\longrightarrow$ classical methods are not adapted
- Backpropagation: algorithm for updating the values of the weights and biases, taking into account the values obtained in the neurons of the output layer in the forward propagation step. The aim is to vary the weights and biases in the right direction to increase the success rate by successive iterations on the training sample.





Backpropagation -- *loss function*

- A function called the loss function is used to quantify the difference observed between the output of the neural network and the class label associated with the data under consideration

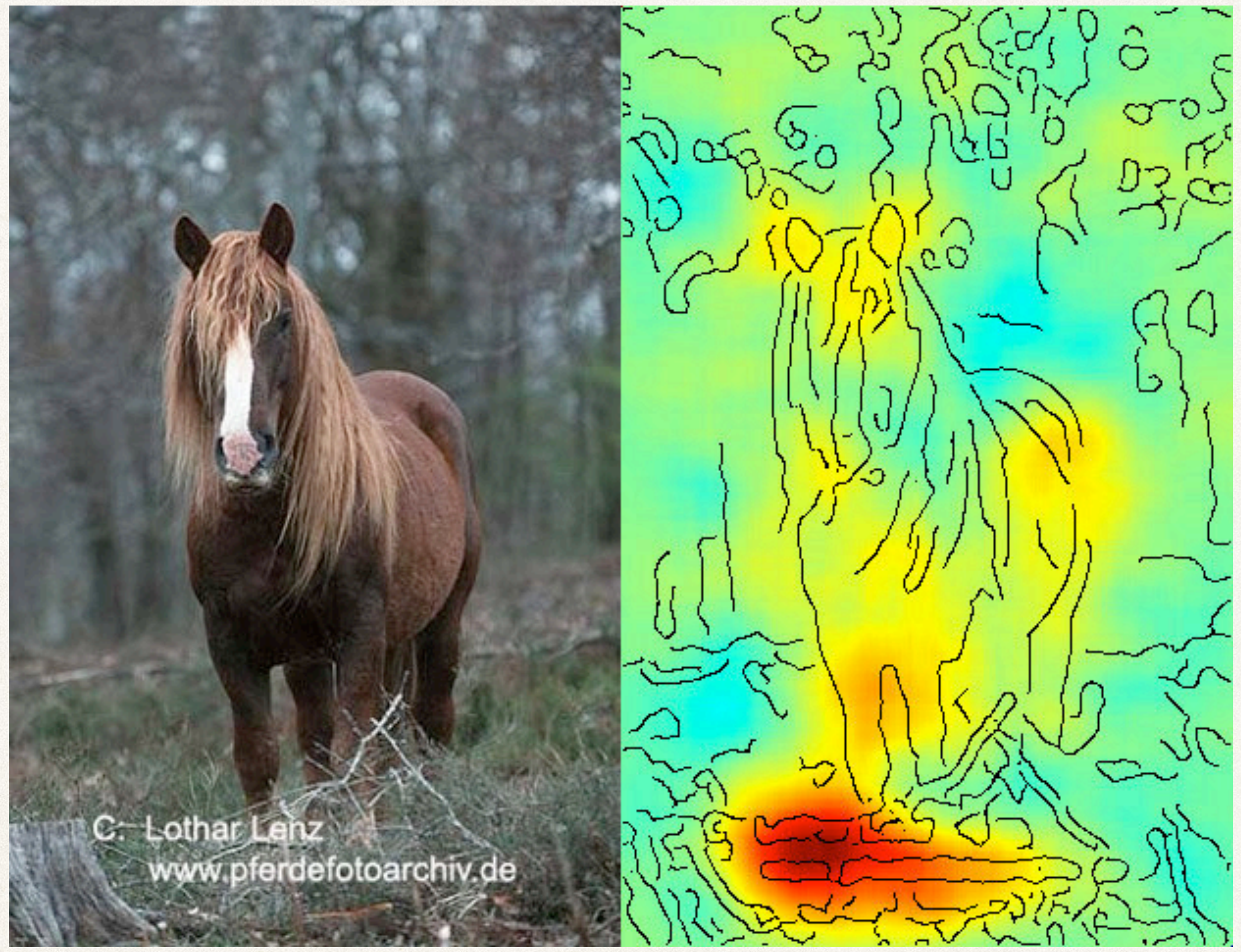
- The training database is a set of input-output pairs $(x_i, y_i)_{1 \leq i \leq D}$ where each x_i and y_i is a realisation of the random variables X and Y

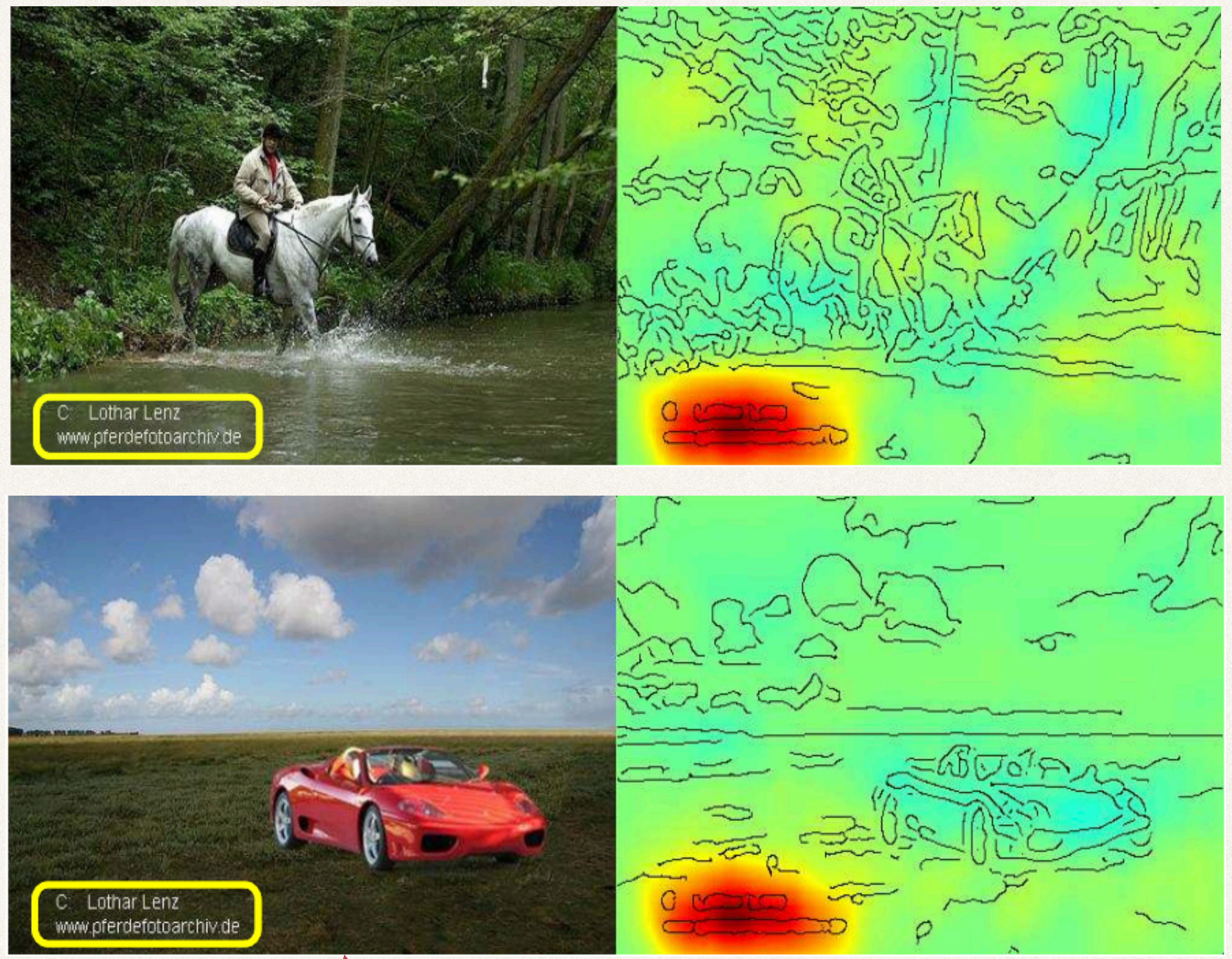
- The aim of supervised learning is to define a nonlinear function f that minimizes the deviation between the random variables $f(X)$ and Y. To define this deviation, we introduce a loss function J that quantifies the distance between a model prediction $f(x_i)$ and an expected output y_i

- To obtain the best function generalizing our data, we will minimize the empirical risk:
$$R_D(f) = \frac{1}{D} \sum_{i=1}^{D} J(y_i, f(x_i))$$
$\longrightarrow$ The mean of the deviations between network prediction and expected output is minimized for D training pairs

Backpropagation -- *representative data sample*

Lapuschkin, S., Wäldchen, S., Binder, A. *et al.*, *Nat Commun* **10**, 1096 (2019)





- The set of input-output pairs $(x_i, y_i)_{1 \leq i \leq D}$ must cover the whole distribution of the random variables X and Y

Backpropagation -- *proof*

- The free parameters of f are the weights W^L and biases b^L for each layer L. We will therefore vary W^L and b^L to minimise $R_D(f)$. As the function $R_D(f)$ is not analytic *(because f is not)*, we perform a numerical minimization via the iterative procedure:

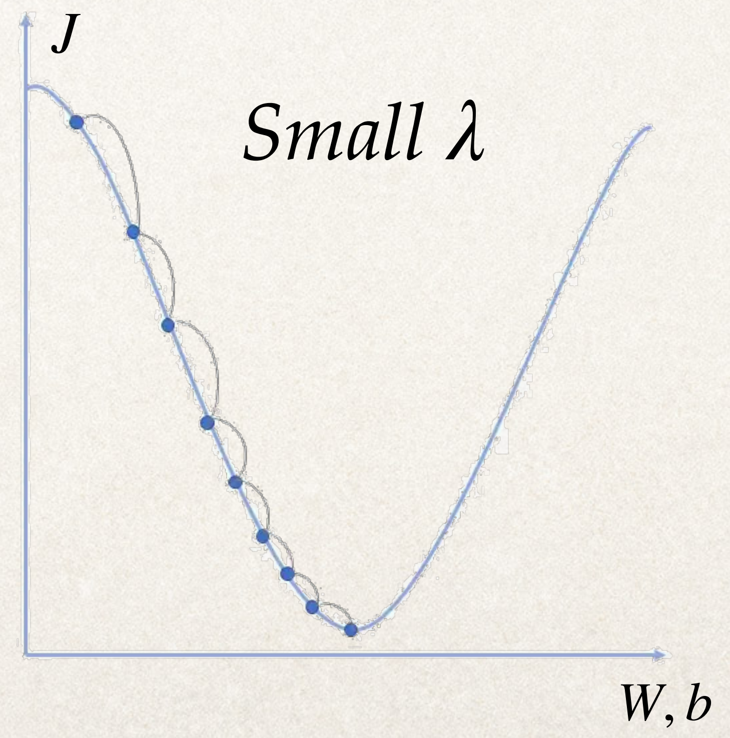
$$W^L = W^L - \lambda \frac{\partial R_D(f)}{\partial W^L} = W^L - \frac{\lambda}{D} \sum_{i=1}^{D} \frac{\partial J(y^i, a^{H_i})}{\partial W^L}$$

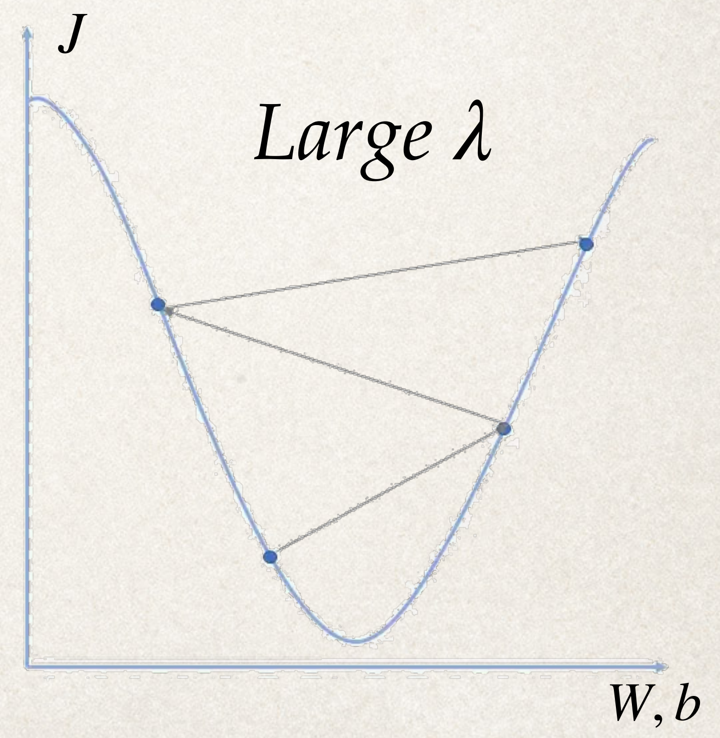
$$b^L = b^L - \lambda \frac{\partial R_D(f)}{\partial b^L} = b^L - \frac{\lambda}{D} \sum_{i=1}^{D} \frac{\partial J(y^i, a^{H_i})}{\partial b^L}$$

Gradient descent!

where the initial values of the W^L matrices and b^L vectors are randomly drawn and then updated at each training iteration *(called epoch)* to minimize $R_D(f)$. The parameter λ is called the learning rate. It is set by the user at the start of training phase.

- We therefore need to know the expressions of $\dfrac{\partial J}{\partial w^L_{mn}}$ and $\dfrac{\partial J}{\partial b^L_n}$ to train the neural network





Backpropagation -- *proof*

General case -- independent of the expression of *J*:

- $$\frac{\partial J}{\partial w_{mn}^L} = \frac{\partial J}{\partial z_n^L}\frac{\partial z_n^L}{\partial w_{mn}^L} = \frac{\partial J}{\partial z_n^L}\frac{\partial\left[\sum_i w_{in}^L a_i^{L-1} + b_n^L\right]}{\partial w_{mn}^L} = \frac{\partial J}{\partial z_n^L}\frac{\partial\left[\sum_i w_{in}^L a_i^{L-1}\right]}{\partial w_{mn}^L} = \frac{\partial J}{\partial z_n^L}\frac{\partial\left[w_{mn}^L a_m^{L-1}\right]}{\partial w_{mn}^L} = \frac{\partial J}{\partial z_n^L} a_m^{L-1} = \delta_n^L a_m^{L-1}$$

  where we define $\delta_n^L = \frac{\partial J}{\partial z_n^L}$ because it is the only element that depends on the expression of the loss function J

- In matrix form we have: $\boxed{\frac{\partial J}{\partial W^L} = a^{L-1}(\delta^L)^T}$

- $\frac{\partial J}{\partial b_n^L} = \frac{\partial J}{\partial z_n^L}\frac{\partial z_n^L}{\partial b_n^L} = \delta_n^L \frac{\partial b_n^L}{\partial b_n^L} = \delta_n^L \longrightarrow$ In vector form: $\boxed{\frac{\partial J}{\partial b^L} = \delta^L}$

- We now need to determine δ_n^L for a given loss function

Backpropagation -- *proof*

Loss function J: cross-entropy

- Consider the cross-entropy function which, for two discrete probability distributions p and q, is given by:
  $$J(p,q) = -\sum_x p(x)\ln[q(x)]$$

Link to maximum likelihood estimator:

- Classification problems: we wish to estimate the probability $q_\theta(X=i)$ that the random variable X has the value i for a set of parameters θ knowing the empirical probability $p(X=i)$ given by the frequency of occurrence of i in our training sample
- If we have D independent pairs in our training sample then the likelihood of the parameters θ in this data set is:
  $$\mathscr{L}(\theta) = \prod_{i\in X}\left(\text{estimated probability of i}\right)^{\text{(number of instances of i)}} = \prod_i q_\theta(X=i)^{D\,p(X=i)}$$
- Thus, the logarithm of this likelihood function is: $\frac{1}{D}\ln\left[\mathscr{L}(\theta)\right] = \frac{1}{D}\ln\left[\prod_i q_\theta(X=i)^{D\,p(X=i)}\right]$
  $$= \sum p(X=i)\ln\left[q_\theta(X=i)\right]$$
  $$= -J(p,q)$$

Maximizing the likelihood of finding W^L and b^L is equivalent to minimizing the cross-entropy J

Backpropagation -- *proof*

- We wish to estimate the distance between the output of the neurons of the last layer a_n^H and the expected value y_n

  We will therefore minimize $\frac{1}{D}\sum_{i=1}^{D} J(y^i, a^{H_i})$ where $J(y^i, a^{H_i}) = -\sum_n y_n^i \ln\left[a_n^{H_i}\right]$

- In a classification problem, each neuron in the output layer corresponds to a class label. It must therefore contain the probability that the input data belongs to a given class label. To achieve this, we use the softmax function as the activation function for each output neuron:

$$a_n^H = h^H(z_n^H) = \frac{e^{z_n^H}}{\sum_i e^{z_i^H}}$$

- Thus, $\delta_n^H = \frac{\partial J}{\partial z_n^H} = \sum_i \frac{\partial J}{\partial a_i^H}\frac{\partial a_i^H}{\partial z_n^H} = \sum_{i \neq n}\left(\frac{\partial J}{\partial a_i^H}\frac{\partial a_i^H}{\partial z_n^H}\right) + \frac{\partial J}{\partial a_n^H}\frac{\partial a_n^H}{\partial z_n^H}$

$$\left.\begin{array}{l} \text{With } \dfrac{\partial J}{\partial a_n^H} = \dfrac{\partial\left[-\sum_m y_m \ln(a_m^H)\right]}{\partial a_n^H} = -\dfrac{\partial\left[y_n \ln(a_n^H)\right]}{\partial a_n^H} = -\dfrac{y_n}{a_n^H} \\ \text{And } \dfrac{\partial a_n^H}{\partial z_n^H} = \dfrac{\partial\left[e^{z_n^H}/\left(\sum_i e^{z_i^H}\right)\right]}{\partial z_n^H} = \dfrac{e^{z_n^H}\left(\sum_i e^{z_i^H}\right) - e^{z_n^H}e^{z_n^H}}{\left(\sum_i e^{z_i^H}\right)^2} = \dfrac{e^{z_n^H}}{\sum_i e^{z_i^H}}\left(1 - \dfrac{e^{z_n^H}}{\sum_i e^{z_i^H}}\right) = a_n^H(1 - a_n^H) \end{array}\right\} \frac{\partial J}{\partial a_n^H}\frac{\partial a_n^H}{\partial z_n^H} = -y_n(1 - a_n^H)$$

Backpropagation -- *proof*

- Therefore, $\delta_n^H = \left[\sum_{i \neq n} \left(\frac{\partial J}{\partial a_i^H} \frac{\partial a_i^H}{\partial z_n^H} \right)\right] - y_n(1 - a_n^H)$

- Furthermore, $\frac{\partial J}{\partial a_i^H} = \frac{\partial \left[-\sum_m y_m \ln(a_m^H) \right]}{\partial a_i^H} = -\frac{y_i}{a_i^H}$ and $\frac{\partial a_i^H}{\partial z_n^H} = \frac{\partial \left[e^{z_i^H} / \left(\sum_m e^{z_m^H} \right) \right]}{\partial z_n^H} \underset{\uparrow}{=} \frac{-e^{z_i^H} e^{z_n^H}}{\left(\sum_i e^{z_i^H} \right)^2} = -a_i^H a_n^H$

  because $i \neq n$ so $\frac{\partial e^{z_i^H}}{\partial z_n^H} = 0$

- Then for $i \neq n$, $\frac{\partial J}{\partial a_i^H} \frac{\partial a_i^H}{\partial z_n^H} = -\frac{y_i}{a_i^H}(-a_i^H a_n^H) = y_i a_n^H$

- And finally, $\delta_n^H = \sum_{i \neq n} y_i a_n^H - y_n(1 - a_n^H) = \sum_{i \neq n} y_i a_n^H + y_n a_n^H - y_n = a_n^H \sum_i y_i - y_n$

  Now each x entry is associated with a single class label, so y is a vector containing only 0s except for one value, which is 1

  $\longrightarrow \sum_i y_i = 1$

- Therefore $\delta_n^H = a_n^H - y_n$ which is written in vector form: $\boxed{\delta^H = a^H - y}$

- We now know the expression of δ for the last layer. All that remains is to calculate a relationship between δ^L and δ^{L+1} to link this layer to the previous layers of the network *(backpropagation)*

Backpropagation -- *proof*

- In the general case, we have $\delta_n^L = \frac{\partial J}{\partial z_n^L} = \sum_m \frac{\partial J}{\partial z_m^{L+1}} \frac{\partial z_m^{L+1}}{\partial z_n^L} = \sum_m \delta_m^{L+1} \frac{\partial z_m^{L+1}}{\partial z_n^L}$ *We have to sum because every z_m in the $L+1$ layer is a function of z_n. The result $h(z_n)$ goes into every neuron in the $L+1$ layer.*

- In addition, $\frac{\partial z_m^{L+1}}{\partial z_n^L} = \frac{\partial \left[\sum_i w_{im}^{L+1} a_i^L + b_m^{L+1} \right]}{\partial z_n^L} = \frac{\partial \left[w_{nm}^{L+1} a_n^L \right]}{\partial z_n^L}$ *Because the only a_i^L that depends on z_n^L is $a_n^L = h(z_n^L)$*

- Then, $\frac{\partial z_m^{L+1}}{\partial z_n^L} = w_{nm}^{L+1} \frac{\partial a_n^L}{\partial z_n^L} = w_{nm}^{L+1}\, h'(z_n^L)$

- Finally, $\delta_n^L = \sum_m \delta_m^{L+1} w_{nm}^{L+1}\, h'(z_n^L) = h'(z_n^L) \sum_m \delta_m^{L+1} w_{nm}^{L+1}$

- We introduce the symbol $\odot$ to represent element-by-element multiplication between two vectors.
  For example: $(a_1, a_2, \ldots a_n) \odot (b_1, b_2, \ldots b_n) = (a_1 b_1, a_2 b_2, \ldots a_n b_n)$

$$\boxed{\text{In vector form we have } \delta^L = h'(z^L) \odot W^{L+1} \delta^{L+1}}$$

Backpropagation -- *overview of the training algorithm*

Summary:

1. $z^L = (W^L)^T a^{L-1} + b^L$ unless $L = 0$ then $z^L = (W^L)^T x + b^L$

2. $a^L = h(z^L)$

3. $\delta^L = h'(z^L) \odot W^{L+1}\delta^{L+1}$ unless $L = H$ then $\delta^H = a^H - y$

4. $\dfrac{\partial J}{\partial W^L} = a^{L-1}(\delta^L)^T$ unless $L = 0$ then $\dfrac{\partial J}{\partial W^L} = x\,(\delta^L)^T$ *(We replace a^{L-1} by x)*

5. $\dfrac{\partial J}{\partial b^L} = \delta^L$

Algorithm :

- Initialize the W^L and b^L with random values and calculate all z^L and a^L with (1) and (2) $\rightarrow$ *forward propagation*
- Compute the δ^L from layer H to 0 using (3)
- Simultaneously compute the $\dfrac{\partial J}{\partial W^L}$ and $\dfrac{\partial J}{\partial b^L}$ from layer H to 0 using (4) and (5)

$\}$ $\rightarrow$ *backpropagation*

- Update the W^L and b^L using the gradient descent method

Backpropagation -- *particular case: single hidden layer*

Study of a classifier with 10 output categories, a single hidden layer and D training pairs (x_i, y_i):

- Elements considered: inputs x_i, matrices W^0 and W^1, vectors b^0 and b^1, and outputs y_i

- Forward propagation: $z_i^0 = (W^0)^T x_i + b^0$ and $a_i^0 = \mathrm{ReLU}(z_i^0)$

  $z_i^1 = (W^1)^T a_i^0 + b^1$ and $a_i^1 = \mathrm{softmax}(z_i^1)$

- Backpropagation:
$$\left.\begin{aligned} \delta_i^1 &= a_i^1 - y_i \\ \delta_i^0 &= h'(z_i^0) \odot W^1 \delta_i^1 \end{aligned}\right\} \rightarrow \begin{aligned} \frac{\partial J_i}{\partial W^1} &= a_i^0(\delta_i^1)^T & \qquad \frac{\partial J_i}{\partial b^1} &= \delta_i^1 \\ \frac{\partial J_i}{\partial W^0} &= x_i(\delta_i^0)^T & \qquad \frac{\partial J_i}{\partial b^0} &= \delta_i^0 \end{aligned}$$

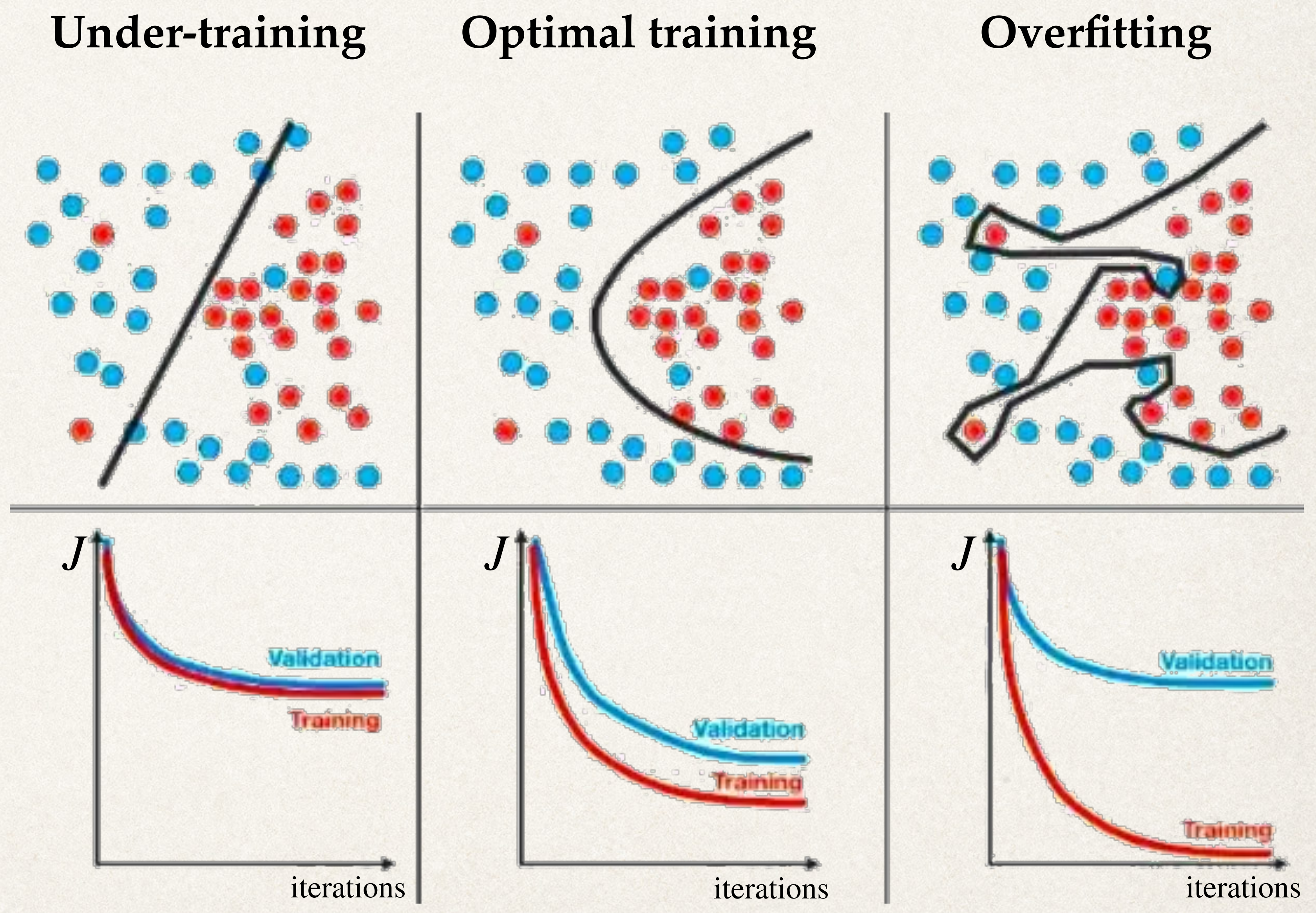
- Weight update:
$$W^1 = W^1 - \frac{\lambda}{D}\sum_{i=1}^{D}\frac{\partial J_i}{\partial W^1} \qquad b^1 = b^1 - \frac{\lambda}{D}\sum_{i=1}^{D}\frac{\partial J_i}{\partial b^1}$$
$$W^0 = W^0 - \frac{\lambda}{D}\sum_{i=1}^{D}\frac{\partial J_i}{\partial W^0} \qquad b^0 = b^0 - \frac{\lambda}{D}\sum_{i=1}^{D}\frac{\partial J_i}{\partial b^0}$$

Backpropagation -- *accuracy and overfitting*

- The accuracy of the neural network's predictions increases with each iteration by construction *(decrease in the loss function)*
- If too many iterations are performed to increase accuracy, the neural network will learn the particularities of the training sample $\longrightarrow$ less generalizable

Two samples: training + test

- We separate the available data into a training sample and a test sample
- At each iteration, the training sample is used to update the weight values, and the test sample is used to calculate the accuracy
- The accuracy calculated on the training sample is necessarily greater because the weights are optimized to maximize it
- If the accuracy calculated on the test sample starts to fall, the network is suffering from overfitting



Classification -- neural networks

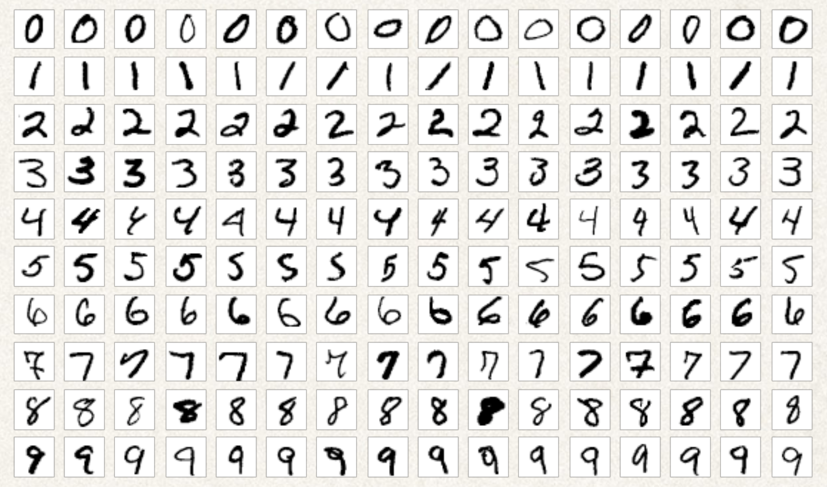
Hands-on session -- *introduction / goals*

- Implementation of a neural network for image classification:

  - Single hidden layer neural network: digit recognition

  - Impact of network architecture on accuracy

  - Analysis of overtraining



- Use of a dedicated neural network library

- Comparison of the tools developed during the session: - results

  - calculation time *(time library)*

  - CO_2 emissions *(CodeCarbon library)*

Hands-on session -- *python tools*

General information:

- All projects are based on the Python 3.9 language
- Projects involve libraries that are increasingly dedicated to the methods being studied

Special case of the neural network project:

- Single hidden layer neural network (architecture + training) only using NumPy
- Using TensorFlow: comparison and generalization to multiple hidden layers
- Analysis of the impact of architecture and learning rate on accuracy

Don't hesitate to have a look at the documentation of these libraries before the session!