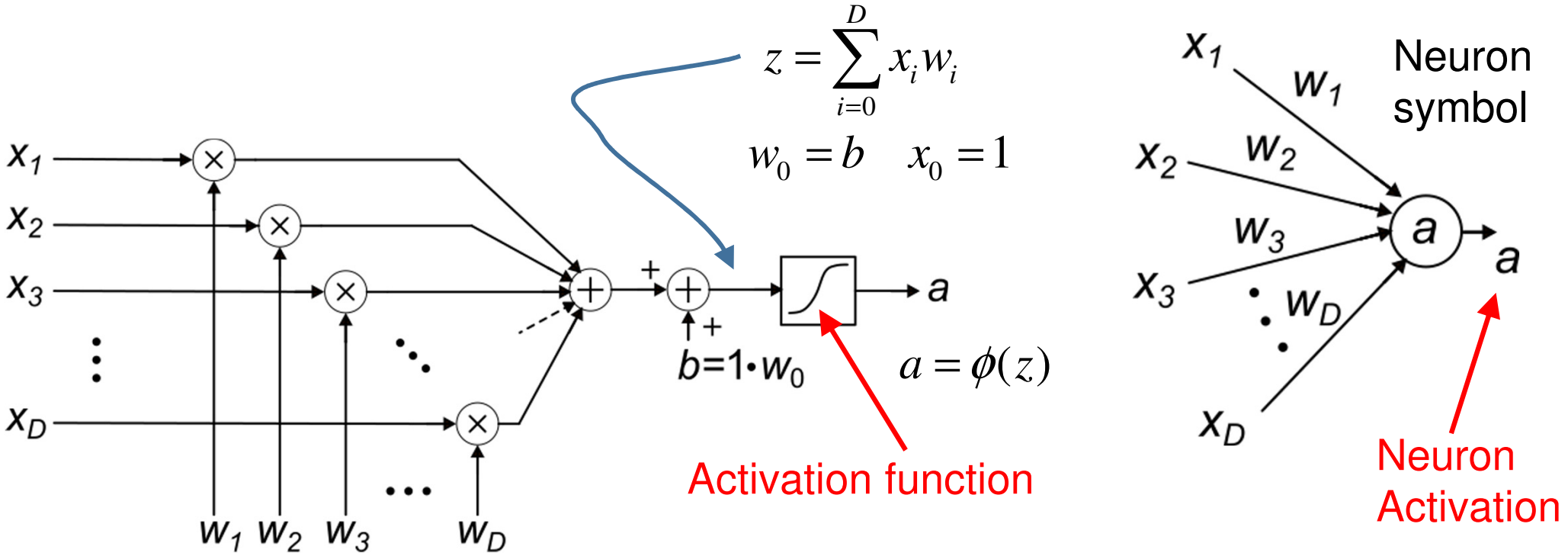
Artificial Neural Networks

The perceptron implements a "single layer" ANN and can solve only binary classification problems for linearly separable data

An ANN uses more perceptrons to solve a much wider variety of problems

In an ANN, the perceptrons are slightly modified to produce a continuous output (instead of a two-value output). The modified perceptrons are called neurons

From the perceptron to the Neuron

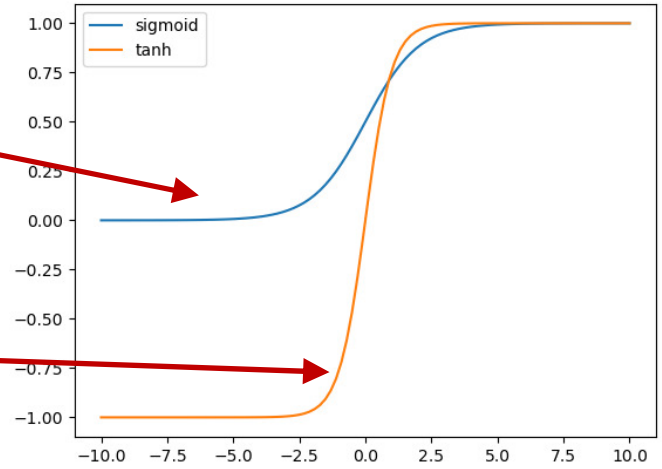


In the neuron, the output step function is substituted by a continuous function, called "activation function" $\phi(z)$. Different activation functions can be used, resulting in different training times. The only important requirement to enable solution of the widest variety of problems is that the activation function is non-linear.

Common activation functions

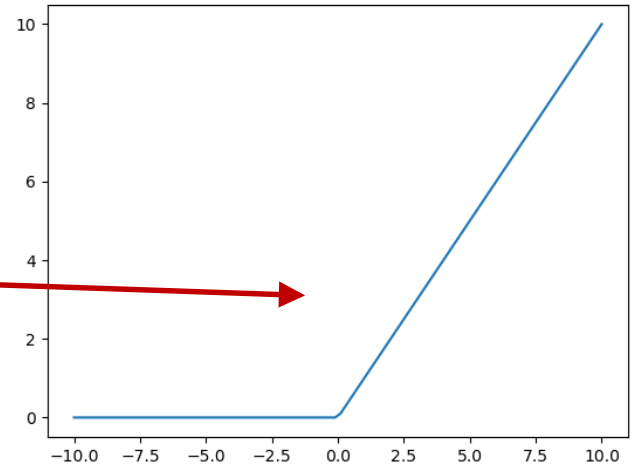
sigmoid $$\phi(z) = \frac{1}{1+e^{-z}}$$

tanh $$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

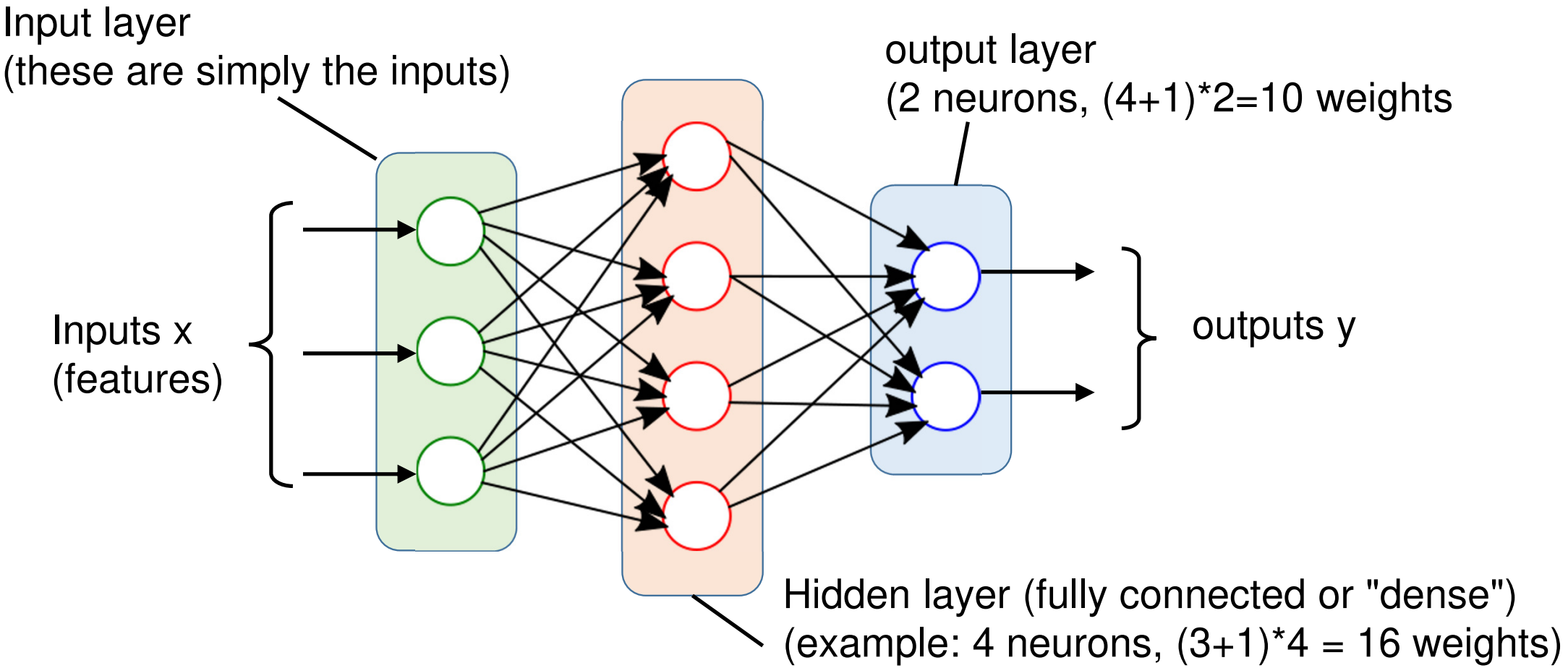


relu (rectified linear unit)

$$\phi(z) = \begin{cases} 0 & for \quad z < 0 \\ z & for \quad z \geq 0 \end{cases}$$



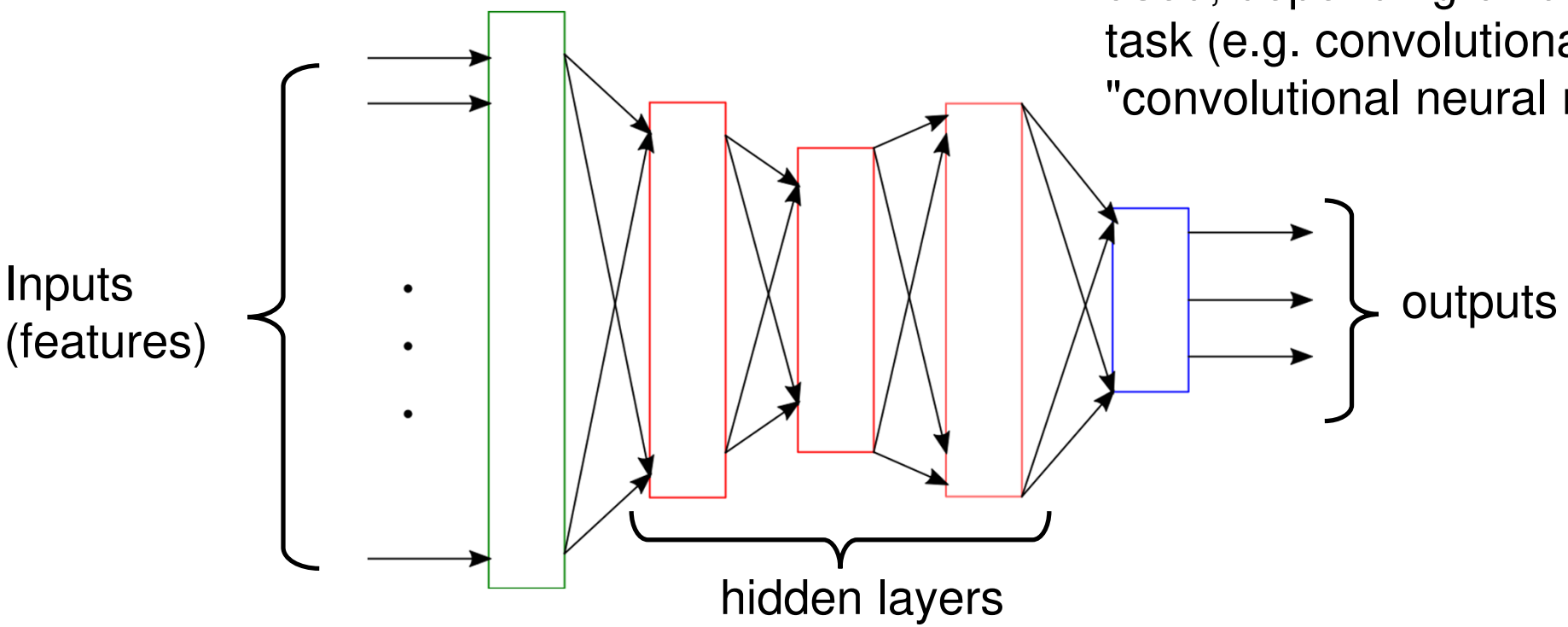
Multi layer neural networks



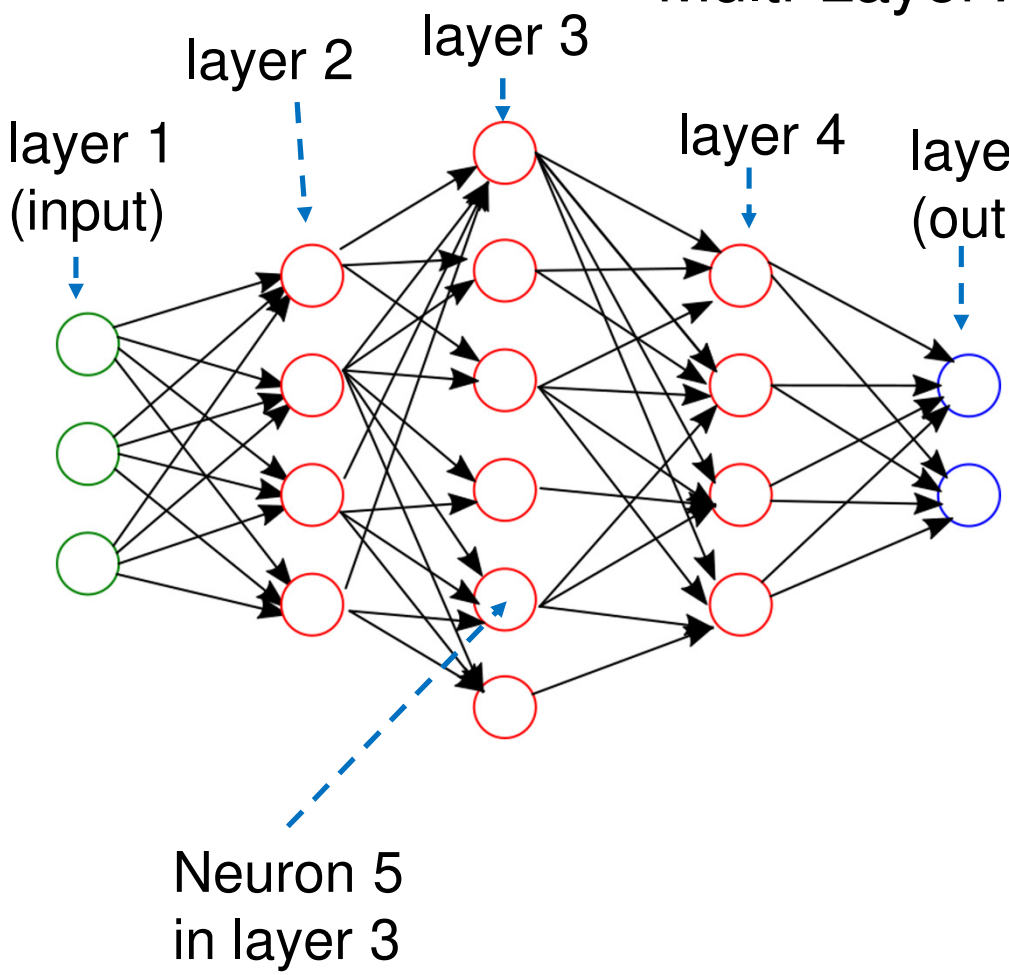
Deep Learning

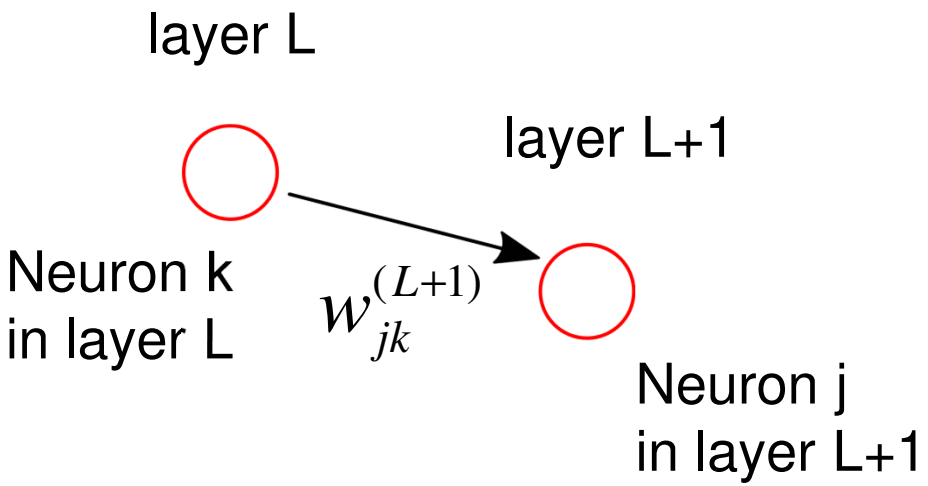
Deep learning indicate machine learning approaches based on ANNs formed by more than one hidden layer.

Specialized hidden layers can be used, depending on the particular task (e.g. convolutional layers in "convolutional neural networks")



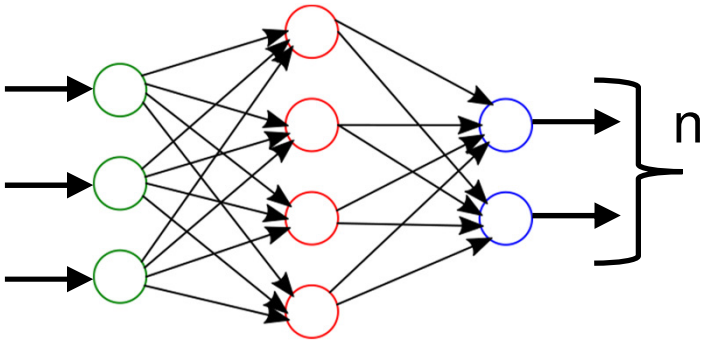
Multi-Layer ANN: naming conventions





Weight determination: cost (loss) function

Goal of learning: to minimize a cost function (or loss function) that satisfy the condition:
-) The smaller the cost function value, the best is the accuracy



$$C_x = \frac{1}{2n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2$$

Cost calculated for a single example x

$$C = \frac{1}{N}\sum_{k=1}^{N} C_{x(k)}$$

Cost calculated over a batch of N examples x

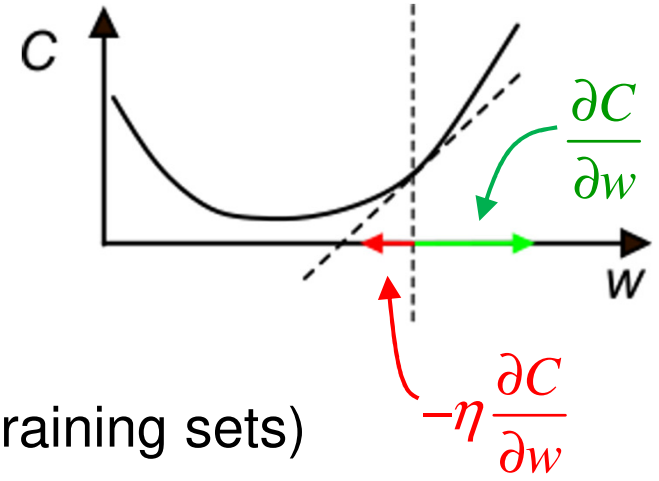
This is just an example of cost function: "quadratic loss"

Depending on the type of problem, different cost functions can be used

Training algorithm

During training, at each update step, all the weights (including bias) are updated according to the following expression:

$$w_{new} = w_{old} - \eta \frac{\partial C}{\partial w}$$



Where C can be calculated:

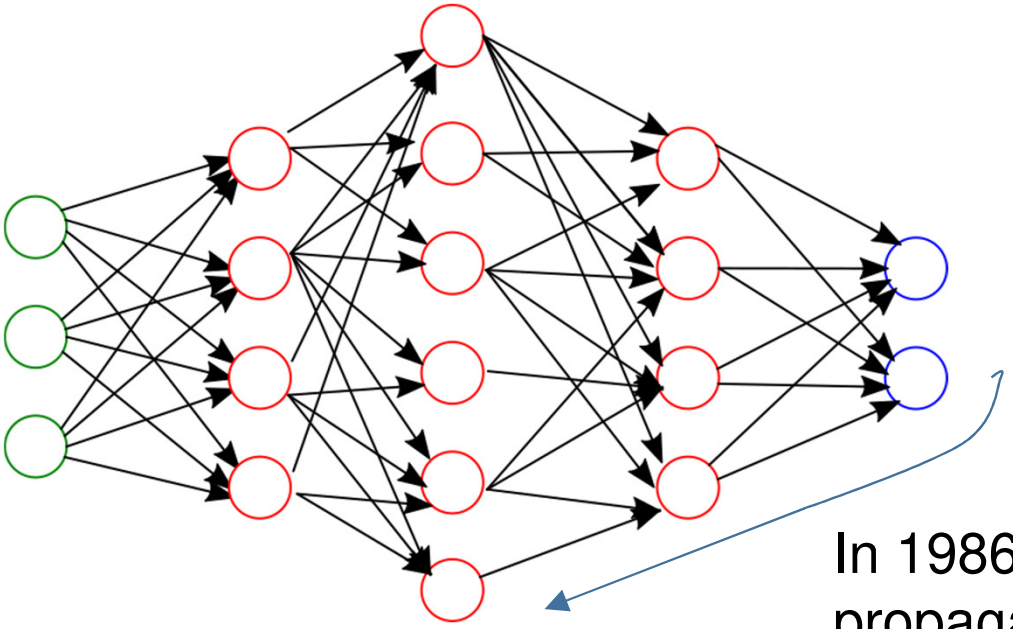
1) Over the whole training set (not feasible for large training sets)

2) Over small batches ("minibatch") of examples randomly taken from the training set

3) Over each single example of the training set

} Stochastic gradient descend

Mention to the backpropagation algorithm



$$w_{new} = w_{old} - \eta \frac{\partial C}{\partial w}$$

In a complex ANN, calculation of the partial derivative of C with respect to every weights can be a computationally hard task

In 1986*, an efficient algorithm called back-propagation that determines all partial derivatives proceeding from the output back to the input was proposed and it is now the most popular method.

*Learning representations by back-propagating errors, D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Nature, 323: 6088, pages 533–536, 1986),

Excercise: an ANN for the classification of the "IRIS" database

The Iris database:

features (**x**)

Label (y)

The iris flower

Example	Sepal Length	Sepal Width	Petal Length	Petal Width	Type
1					
2					
3					
4					

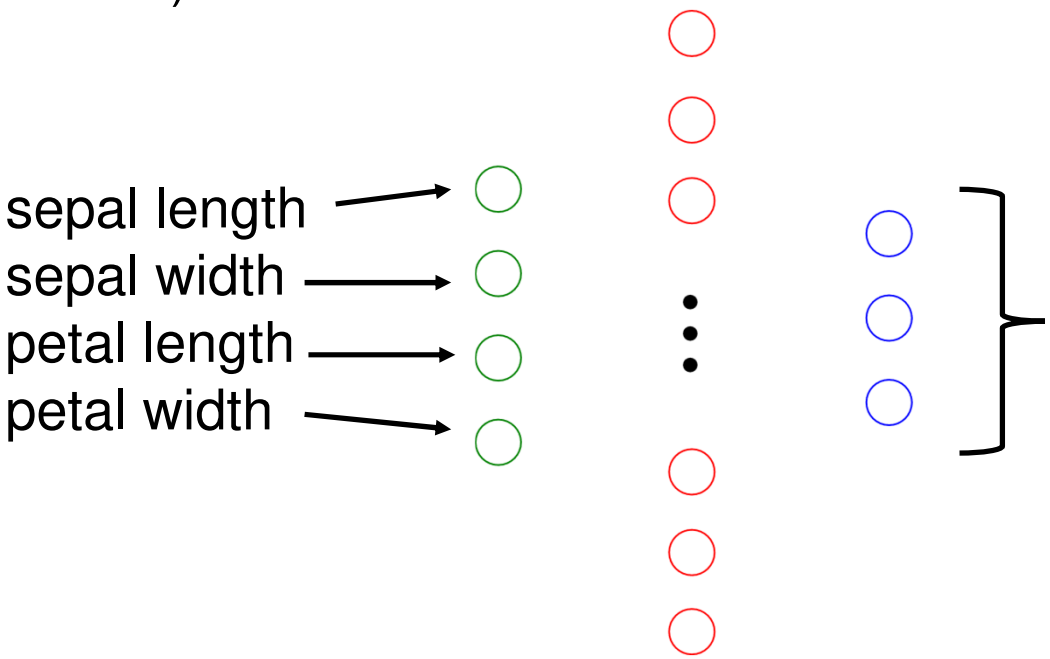
150 examples

Three types:
0: Setosa
1: Versicolor
2: Virginica

Multiclass classification ANN

input layer
(4 neurons)

output layer (3 neurons)



"one hot" representation

$$\begin{pmatrix}1\\0\\0\end{pmatrix}\quad\begin{pmatrix}0\\1\\0\end{pmatrix}\quad\begin{pmatrix}0\\0\\1\end{pmatrix}$$

0 1 2

The three output of the network will be real number that represent the probability that the input belongs to the corresponding class

hiddel layer (16 neurons)

Tensorflow and Keras

Tensorflow: **Tensors:** corresponds to arrays in numpy
Tensorflow includes very effective functions to manipulate tensors

Tensorflow.data **Datasets:** combine data (for example tensors) in a way that is specialized for machine learning operations. Situations where the training data are so many that cannot be loaded completely in RAM are efficiently handled by Database related functions

Tensorflow.keras Include classes and functions that allows easy creation and training of **ANNs**

Bibliography

S. Raschka and V. Mirjalili, "Python Machine Learning " , Third Edition