III. Non-linear regression

1) Articifical Neural Neworks

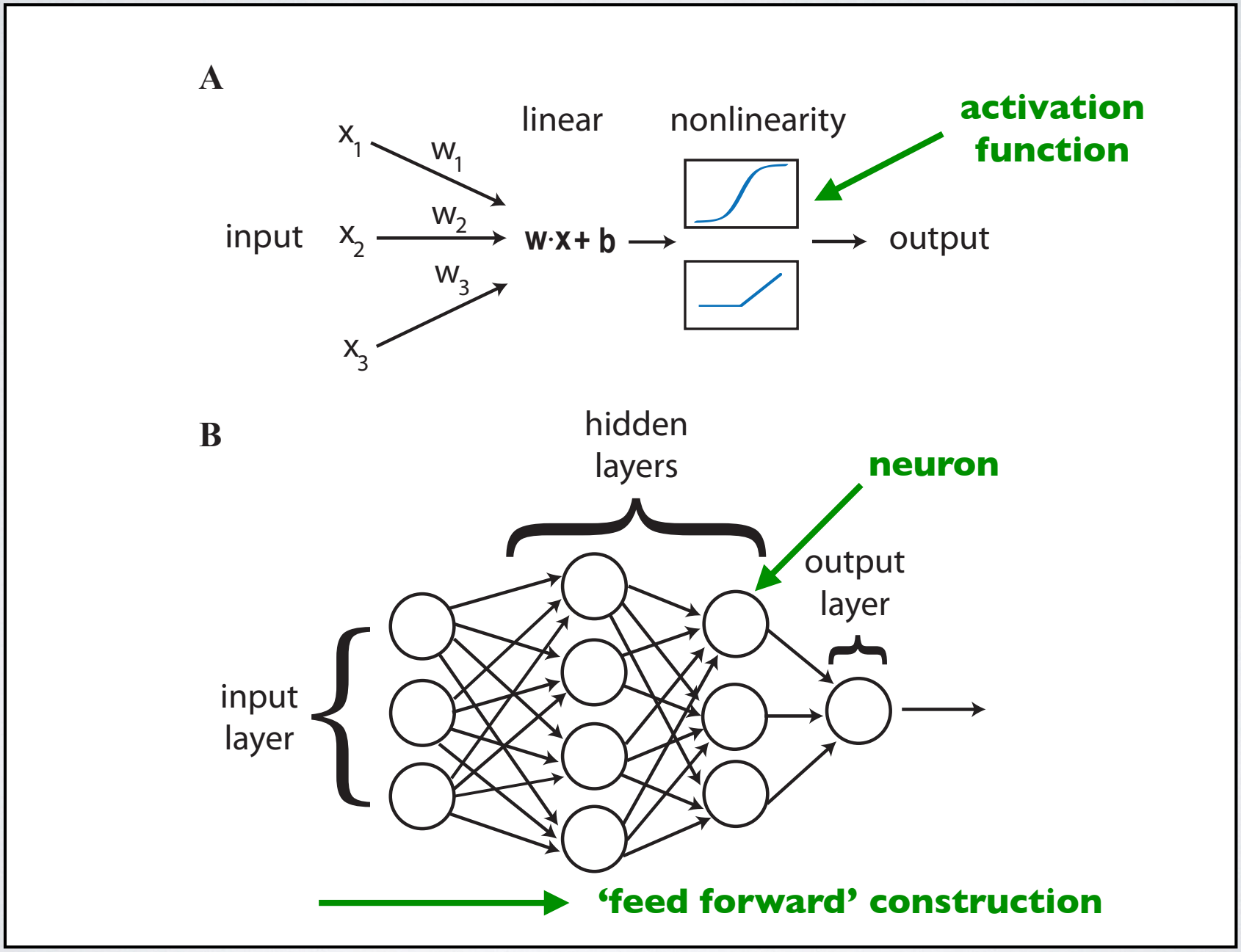
- 1.1) Definition and Motivation
- 1.2) The Perceptron
- 1.3) General models and Hidden Layers

2) Training

- 2.1) The loss function
- 2.2) Backpropagation
- 2.3) Efficient training algorithms

1)

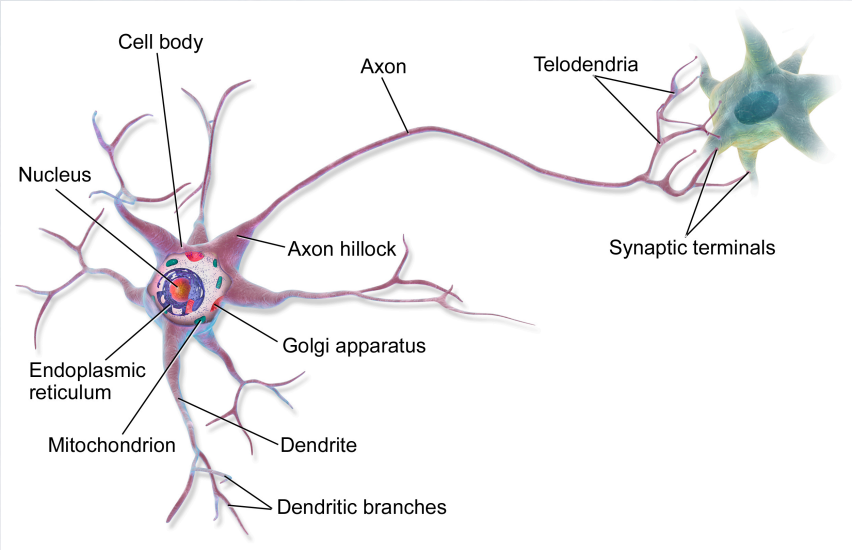
Artifical neural networks: architecture



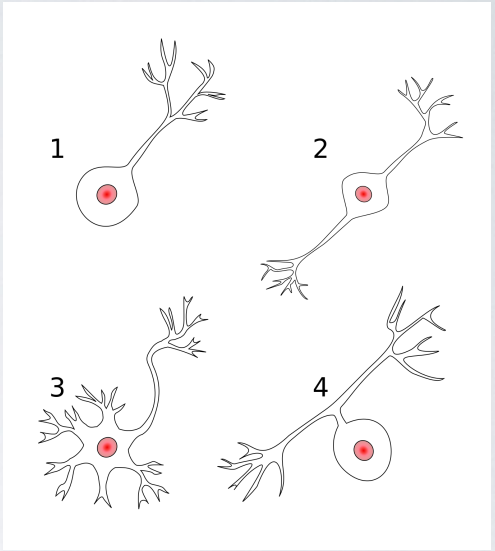
1.1)

Inspiration: how the brain works

neuron



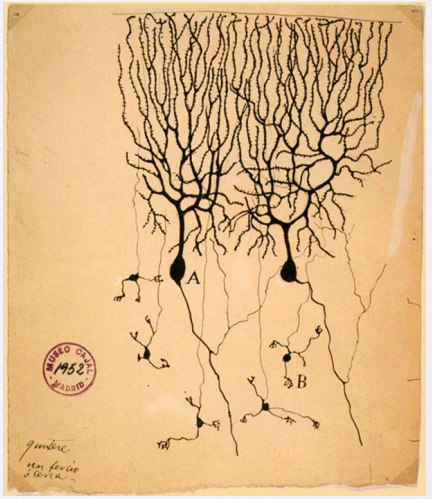
(word neuron introduced in 1891)



different types of neurons

connected system of neurons in the brain

billions of connected neurons



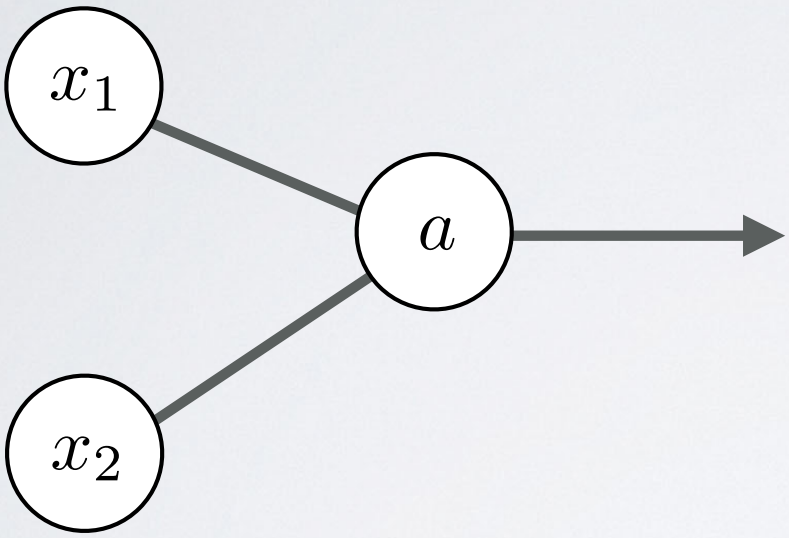
sketch by Ramon y Cahal (1899)

1.2)

The perceptron

or 'McCulloch-Pitts Neuron' (1943)

The most basic architecture we can consider is:



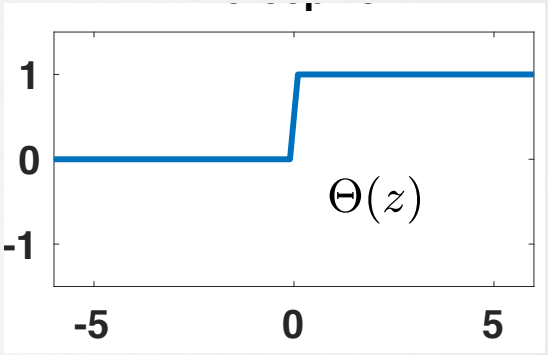
model parameters

$$a(x_1, x_2; \{w_1, w_2, b\}; \sigma) = \sigma(w_1 x_1 + w_2 x_2 + b)$$

weights

biases

where we take $\sigma(z) = \Theta(z)$



first implementaion Rosenblatt (1958)

1.3)

The perceptron is not enough!

[see also notes and Mathematica example]

- the perceptron is a linear classifier
- fails to model simple logic gates such as XOR
- additional layers were introduced to solve this problem

jargon and acronyms

MLP - Multi Layer Perceptron

ANN - Artificial Neural Network

MLP = Feed-forward ANN

common usage: NN = ANN

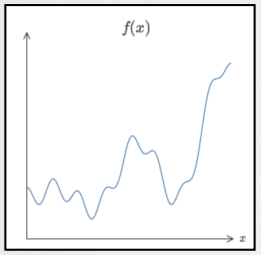
1.3)

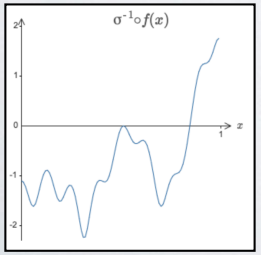
Universal functions

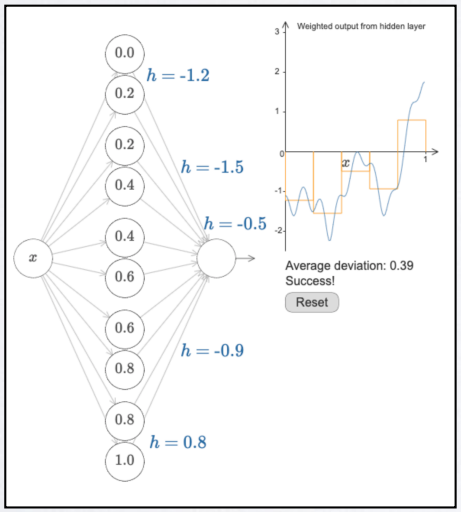
one can show thay a multi-layer neural network with a single hidden layer (infinite depth) is a universal (continuous) function *approximator*

more on the proof from
Nielsen, Neural Networks and Deep Learning (2015)

http://neuralnetworksanddeeplearning.com/chap4.html







counting model parameters

input	hidden	output	parameters
1	1	1	3
1	2	1	7
2	3	1	15
1	n	1	3n+1

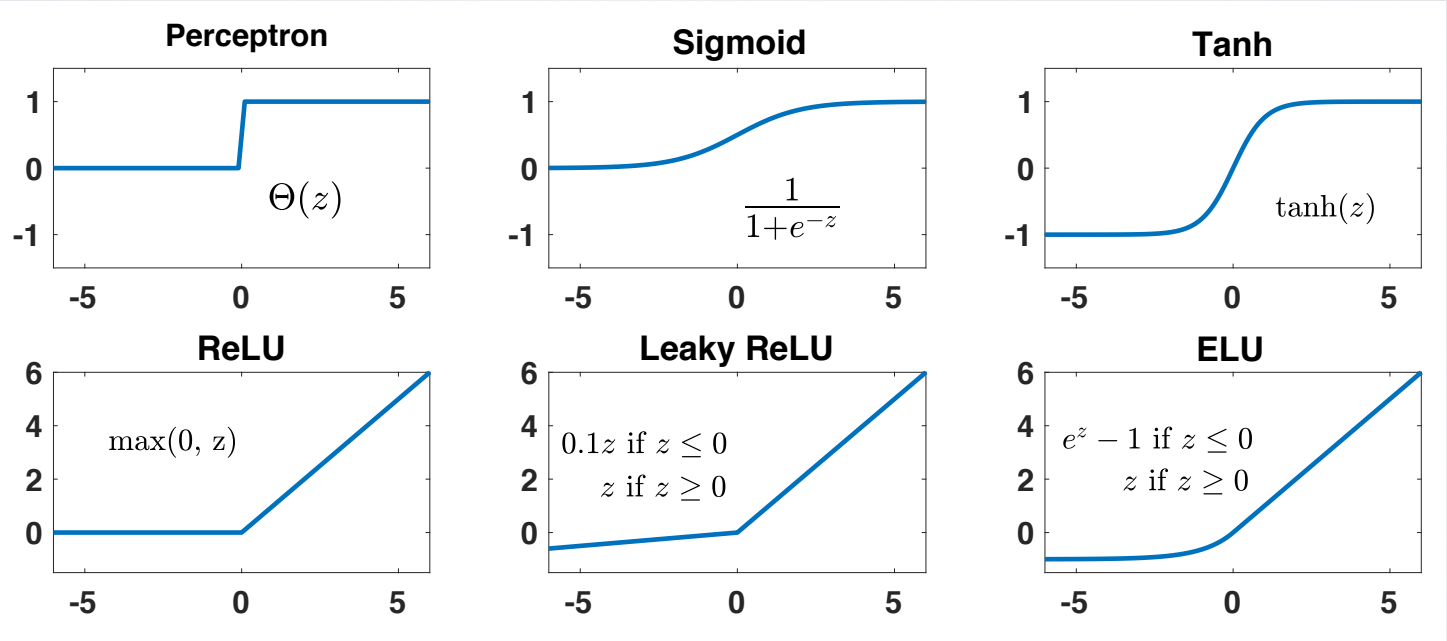
for L layers each with n_l neurons

$$\sum_{l=2}^{L}(n_{l-1}+1)n_l$$

1.3)

Types of activation functions

depending on the type of problem, the choice of activation function is very important



jargon and acronyms

ReLU - Rectified Linear Units

ELU - Exponential Linear Units

Sigmoid* = Logistic function

* technically the 'sigmoid' is general shape of curve of which the Logistic functions and hyberbolic tangent are examples

2)

Artifical neural networks: Training

Given a data set $\{x_i, y_i\}$ we may consider a

1. Initialise parameters
2. Compute gradient against data
3. Modify parameters
4. Iterate steps 1-4 until suitable approximation to data is found

← i.e. use gradient descent

question

what could affect the performance?

e.g. definition of 'suitable', number of iterations, format of input data…

2.1)

The Cost or Loss function

(also see notes)

We need a differentialble measure of the fit quality

$f(\{x\})$ the function we are trying to approximate

$\hat{f}(\{x\},\{p\})$ the model we are trying to train

$f(\{x_k\}) = f_k$ the training data (ignore errors for now)

$$E_k(\{p\}) = (f_k - \hat{f}(\{x_k\},\{p\})^2$$

Cost/Loss function
e.g. MSE function

$$\langle E(\{p\})\rangle = \frac{1}{n}\sum_{k=1}^{n} E_k(\{p\})$$

MSE function for the data set

2.1)

Gradient descent

(also earlier parts of the course from prof. Nocera)

Given the form of the cost/loss function we can compute

$$\{v_t\} = \langle \nabla_p E(\{p_t\}) \rangle$$
$$\{p_{t+1}\} = \{p_t\} - \eta\{v_t\}$$

for each iteration step t.

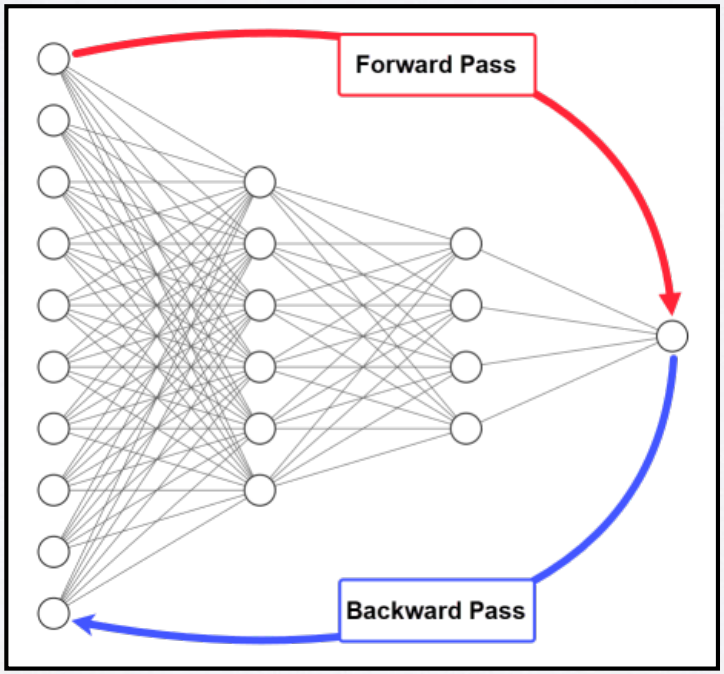
The improvements of *stochastic gradient descent* and *adaptive gradient descent* (e.g. ADAM) are essential for large data sets

2.2)

Backpropagation

(see notes)

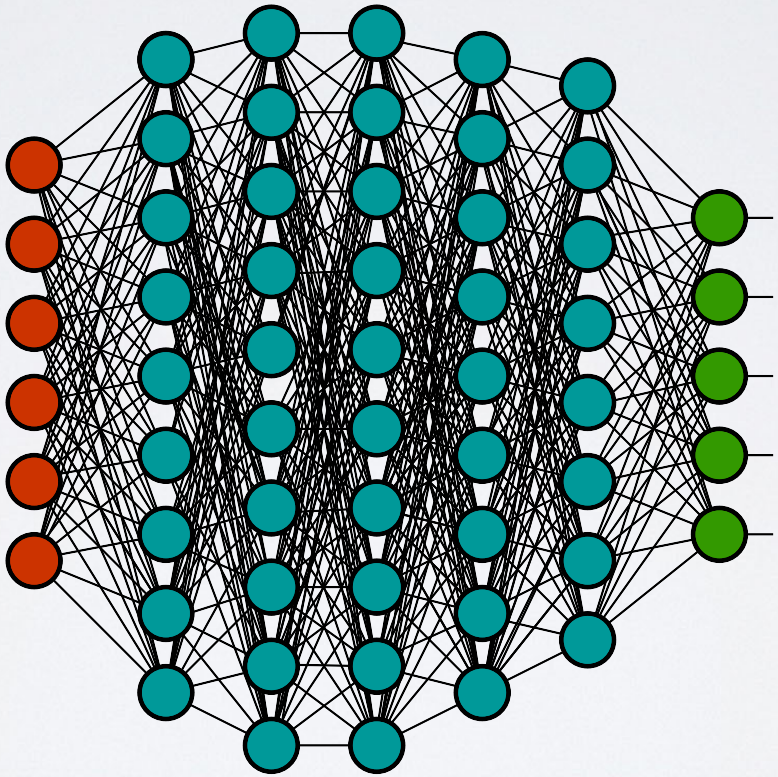
Introduced in various forms from 60's-80's, *backpropagation* is and efficient method to compute the gradient of a multilayer ANN using the differentiable property of the model



2.3)

Efficient Training Methods

The simple examples we have seen so far are
not scalable for deep neural networks (DNN)



DNNs are simply ANNs with many layers and many neurons per layer

2.3)

Efficient Training Methods (Regularisation)

- Parameter initialisation
- Cross-validation: Early stopping
- Hyperoptimisation
- Batch normalisation
- Dropout