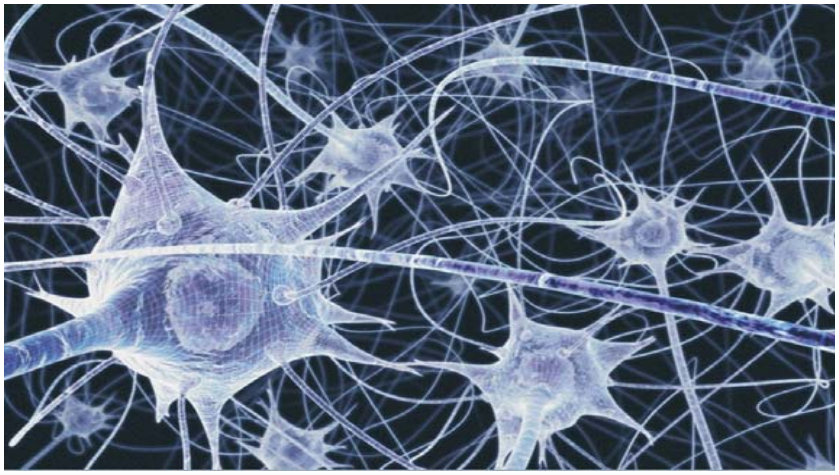
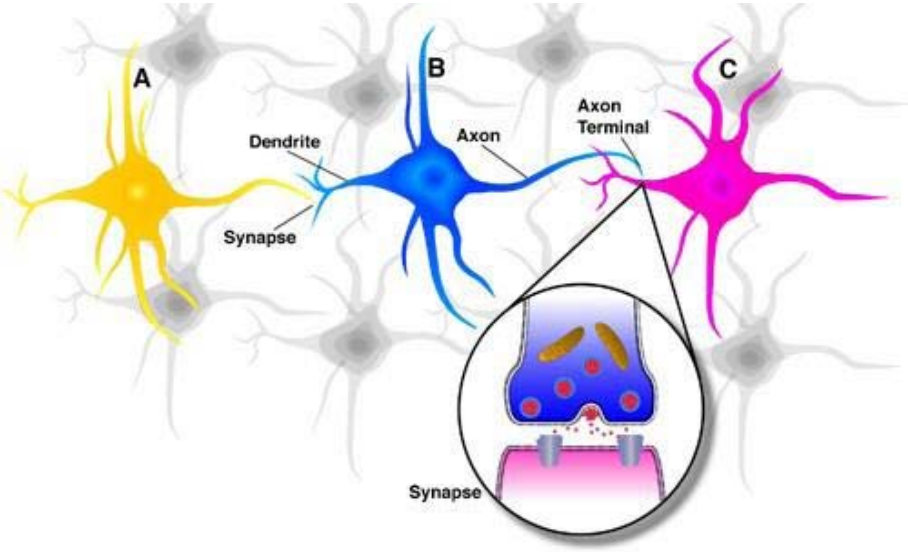
Introduction to Artificial Neural Networks (ANN)

Biological nervous systems

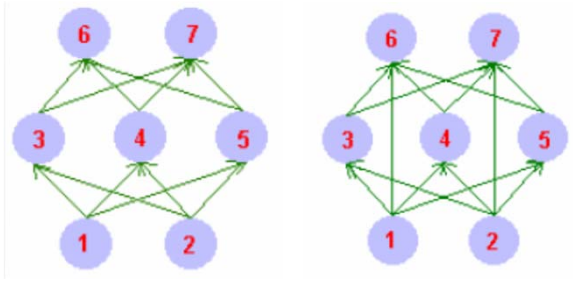
Ability to respond (response) to signals from the environment (stimulus)



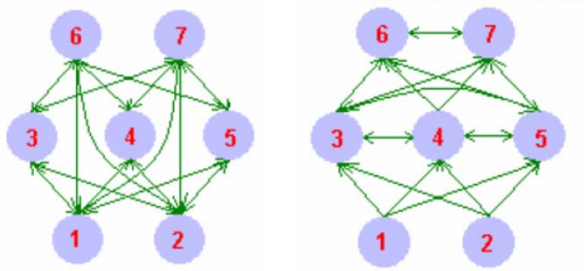


What are Artificial Neural Networks?

- Mathematical replicas of biological nervous systems:
  Receive inputs ⇒ Process data (through learning process) ⇒ Provide outputs
- An ANN is made up of a large number of artificial neurons and an associated network structure.
  - Neurons are defined as mathematical functions that filter the signal through the network.
  - Neurons are arranged in layers. Each neuron is connected to other neurons via weighted connections.
- Different type of ANN classified depending on their Structure, Neuron link, Neurons and layers density, Activate functions etc.:
  - Feed Forward Neural Network
  - Multilayer Perceptron
  - Recurrent Neural Network
  - Convolutional Neural Network
  - Radial Basis Functional Neural Network
  - LSTM – Long Short-Term Memory
  - Sequence to Sequence Models
  - Modular Neural Network
  - …



Feedforward
(forward link)

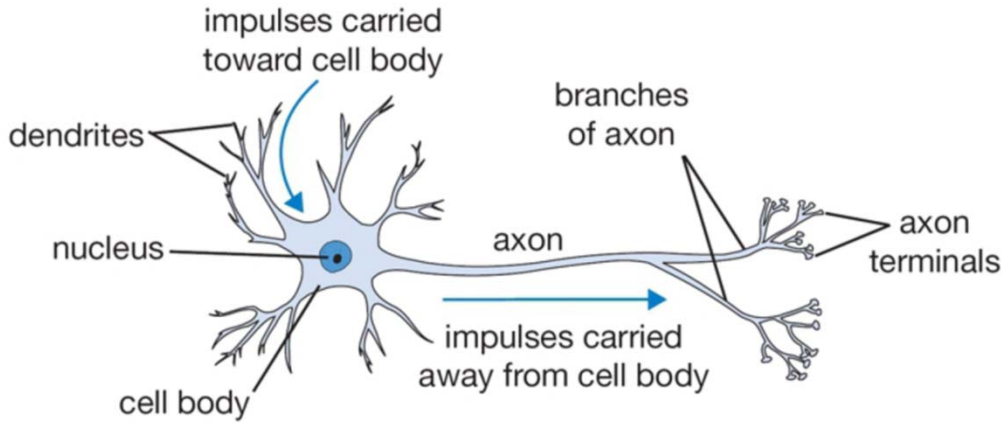


Recurrent
(with feedback)

Functioning of a neuron

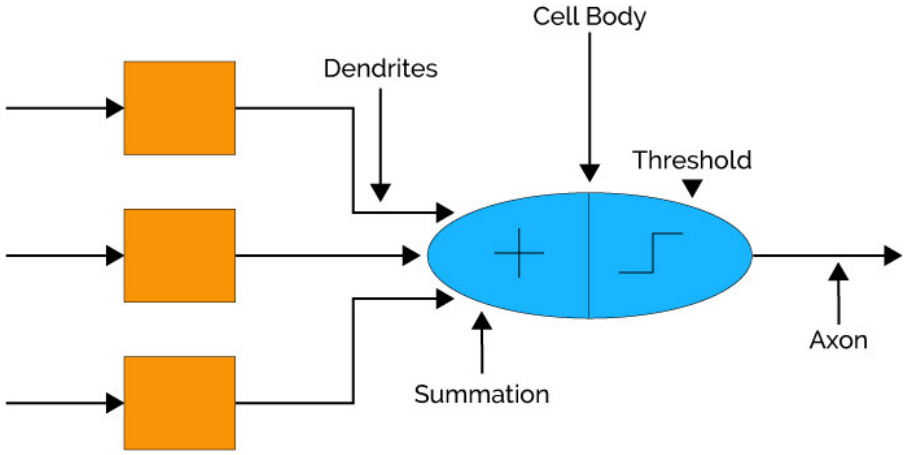
Biological Neuron:

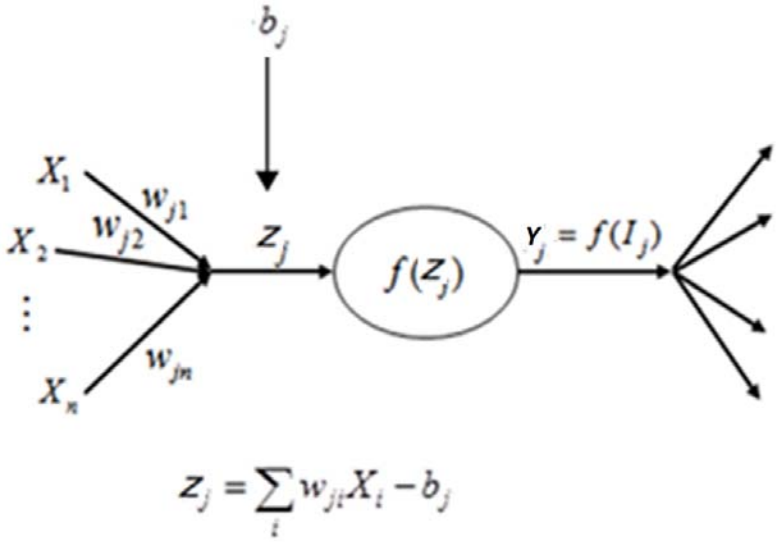
- Signals are received via the dendrites and forwarded to the cell body and added up.
- If the signals have exceeded a certain threshold value, the cell nucleus is activated, the signals are analyzed, evaluated and finally forwarded via the axon and then transmitted through the synapses to the dendrites of the next neurons.
- The biological learning process takes place through the adaptation of the connections (synapses) between the nerve cells.



Artificial Neuron (or Node / Perceptron / Processing element / Unit):

- Idea is to use a simplified (mathematical) model of a biological neuron: a neuron receives its weighted inputs, which are combined and passed through a transfer or activation function to produce the output.
- The weights and thresholds are adjusted and defined by the learning or training process.

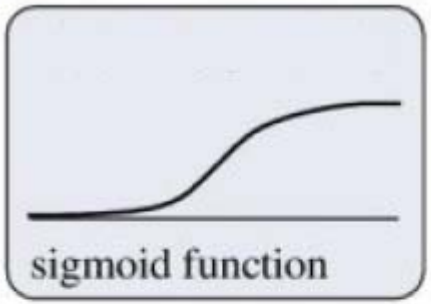




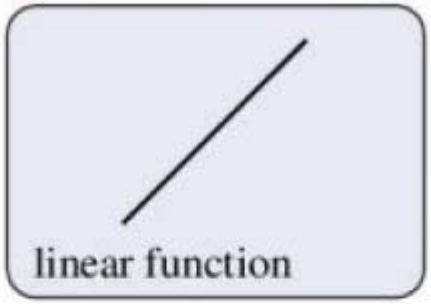
$$z_j = \sum_i w_{ji} X_i - b_j$$

Commonly used activation functions

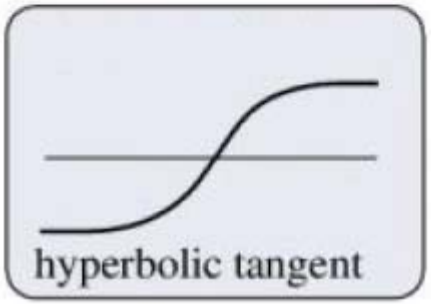
$$f(Z_j) = \sigma(Z_j) = \frac{1}{1+e^{-Z_j}}$$



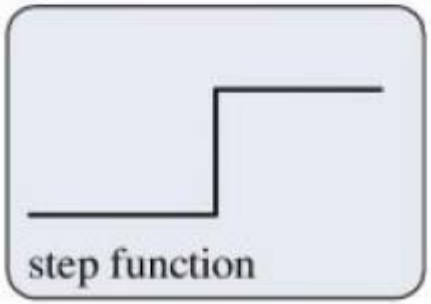
$$f(Z_j) = Z_j$$



$$f(Z_j) = \tanh(Z_j) = \frac{e^{Z_j} - e^{-Z_j}}{e^{Z_j} + e^{-Z_j}}$$

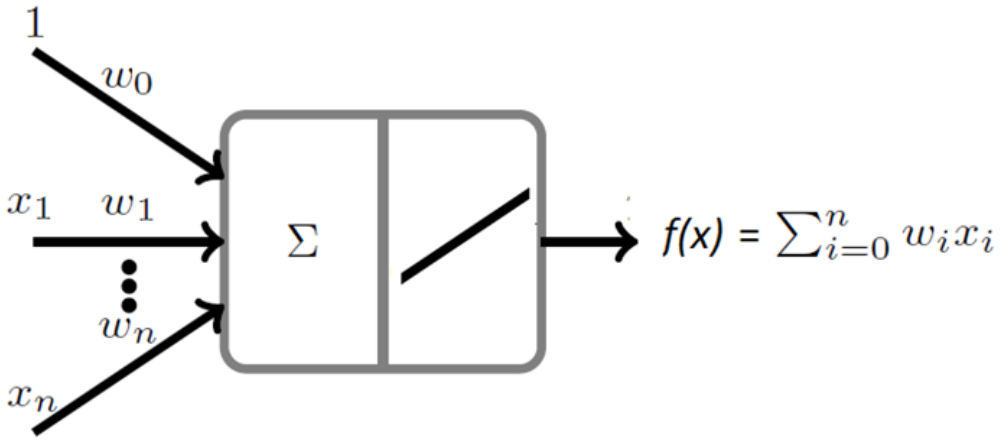


$$f(Z_j) = \begin{cases} 1 & \text{if} \quad Z_j \geq 0 \\ 0 & \text{if} \quad Z_j < 0 \end{cases}$$



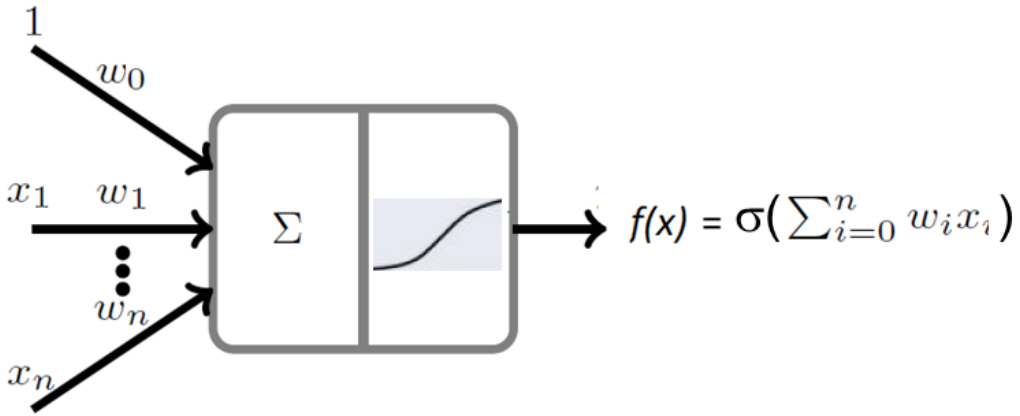
Linear Regression is a simplified network with

- only one processing unit, and
- a linear activation function.



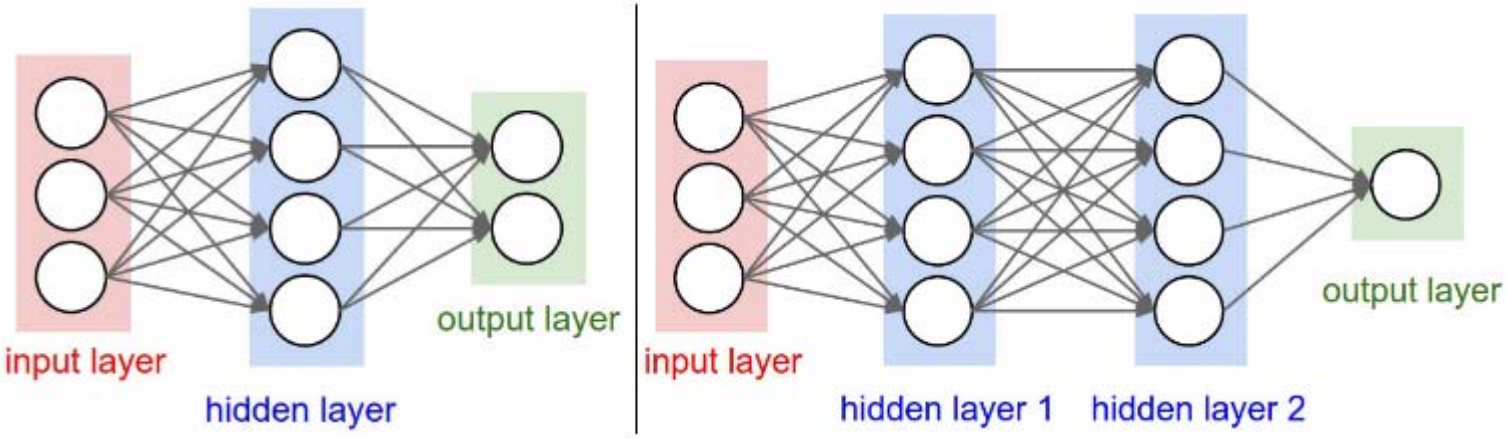
Logistic Regression is a simplified network with

- only one processing unit, and
- a logistic/sigmoid activation function.



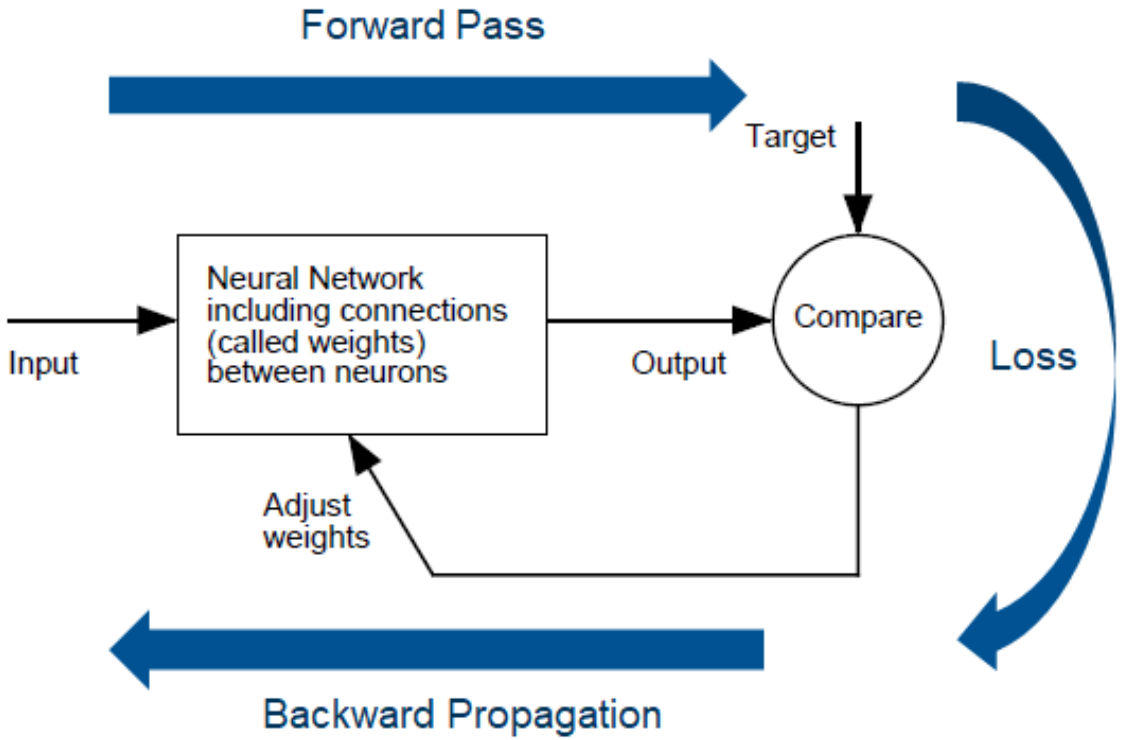
Multilayer Perceptron (MLP)

- A MLP has at least one or more hidden layers.
- Every single node is connected to all neurons in the next layer (fully connected layers).
- Input data travels in one direction only, passing through neural nodes and exiting through output nodes (feedforward ANN):
  - Inputs are multiplied with weights, fed to the activation function and produced outputs, which are used as inputs for nodes in the next layer.

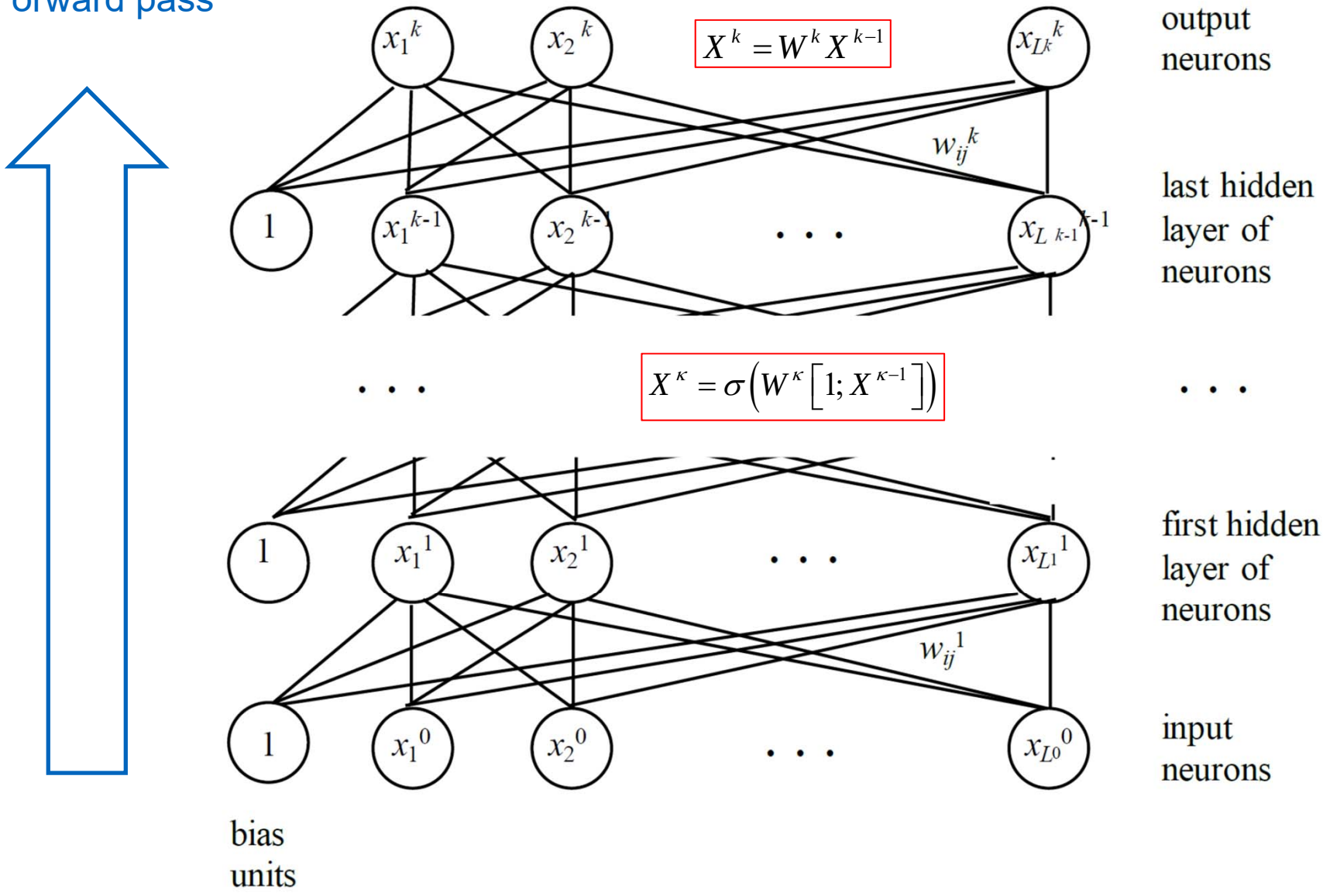


MLPs are used for the supervised learning of vectorial input-output tasks.

- Labeled data set (X,Y)
- Forward pass: define $F_W(X)$
- Cost function: E(W)
- Error Backpropagation: adjust W to reduce E(W).



Forward pass



Cost Function

- Cost function is defined as the quadratic loss:

$$E(W) = \left\| F_W(X) - Y \right\|^2 = \sum_{i=1}^{m} \left(F_W(X^{(i)}) - Y^{(i)} \right)^2 := \sum_{i=1}^{m} \varepsilon^2$$

- During the forward pass, the following quantity is computed and stored (before applying the activation function) for each hidden and output unit x^{κ}_i

$$a_i^{\kappa} = \sum_{j=0}^{L^{\kappa-1}} w_{ij}^{\kappa} x_j^{\kappa-1}$$

- We apply the chain rule for partial derivative of the cost function:

$$\frac{\partial E(W)}{\partial w_{ij}^{\kappa}} = \frac{\partial E(W)}{\partial a_i^{\kappa}} \frac{\partial a_i^{\kappa}}{\partial w_{ij}^{\kappa}} = \frac{\partial E(W)}{\partial a_i^{\kappa}} x_j^{\kappa-1}$$

- We define $\delta_i^{\kappa} := \frac{\partial E(W)}{\partial a_i^{\kappa}} \quad \Rightarrow \frac{\partial E(W)}{\partial w_{ij}^{\kappa}} = \delta_i^{\kappa} x_j^{\kappa-1}$

$\Rightarrow$ in order to calculate the gradient of the cost function, we only need to compute the values of δ^{κ}_i for each hidden and output unit.

Backward propagation

- Compute the values of δ^k_i for each output unit:

$$\delta_i^k = \frac{\partial E(W)}{\partial a_i^k} = \frac{\partial E(W)}{\partial F_W\left(X^{(i)}\right)} = \frac{\partial \left\| F_W\left(X^{(i)}\right) - Y^{(i)} \right\|^2}{\partial F_W\left(X^{(i)}\right)} = 2\left(F_W\left(X^{(i)}\right) - Y^{(i)}\right)$$

- Compute the values of δ^κ_i for each hidden unit:

  Since the only path by which a^κ_i can affect E(W) is through the potentials $a^{\kappa+1}_i$ of the next higher layer, we have

$$\delta_i^\kappa = \frac{\partial E(W)}{\partial a_i^\kappa} = \sum_{l=1}^{L^{\kappa+1}} \frac{\partial E(W)}{\partial a_l^{\kappa+1}} \frac{\partial a_l^{\kappa+1}}{\partial a_i^\kappa} = \sum_{l=1}^{L^{\kappa+1}} \delta_l^{\kappa+1} \frac{\partial\left(\sum_{j=0}^{L^\kappa} w_{lj}^{\kappa+1}\sigma\left(a_j^\kappa\right)\right)}{\partial a_i^\kappa}$$

$$= \sum_{l=1}^{L^{\kappa+1}} \delta_l^{\kappa+1} \frac{\partial\left(w_{li}^{\kappa+1}\sigma\left(a_i^\kappa\right)\right)}{\partial a_i^\kappa} = \frac{\partial\left(\sigma\left(a_i^\kappa\right)\right)}{\partial a_i^\kappa} \sum_{l=1}^{L^{\kappa+1}} \delta_l^{\kappa+1} w_{li}^{\kappa+1}$$

$$= \sigma\left(a_i^\kappa\right)\left[1-\sigma\left(a_i^\kappa\right)\right] \sum_{l=1}^{L^{\kappa+1}} \delta_l^{\kappa+1} w_{li}^{\kappa+1}$$

$\Rightarrow$ This formula describes how the δ^κ_i in a hidden layer can be computed by "back-propagating" the $\delta^{\kappa+1}_i$ from the next higher layer. The formula can be used to compute all δ's, starting from the output layer, and then working backwards through the network in the backward pass of the algorithm.

Training function / methods

- Gradient descent:

$$E(W) = \sum_{i=1}^{m} \left(F_W \left(X^{(i)} \right) - Y^{(i)} \right)^2 = \sum_{i=1}^{m} \varepsilon_i^2$$

$$W^{(k+1)} = W^{(k)} + \alpha \underbrace{\nabla W^{(k)}}_{(\delta W)}$$

- Levenberg-Marquardt's Algorithm:

$$(\delta W) = -\left[(\nabla W)^T (\nabla W) + \mu I \right]^{-1} (\nabla W)^T \varepsilon$$

I - Identity matrix,
μ - Marquardt-Parameter

- Model performance depends on:
  - ✓ Initialization of weights and thresholds
  - ✓ Network architecture: inputs, number of layers, number of hidden neurons, activation functions
  - ✓ Training method
  - ✓ etc.

MATLAB training functions

`trainFcn` — Training function name
`'trainlm'` (default) | `'trainbr'` | `'trainbfg'` | `'trainrp'` | `'trainscg'` | ...

Training function name, specified as one of the following.

Training Function	Algorithm
`'trainlm'`	Levenberg-Marquardt
`'trainbr'`	Bayesian Regularization
`'trainbfg'`	BFGS Quasi-Newton
`'trainrp'`	Resilient Backpropagation
`'trainscg'`	Scaled Conjugate Gradient
`'traincgb'`	Conjugate Gradient with Powell/Beale Restarts
`'traincgf'`	Fletcher-Powell Conjugate Gradient
`'traincgp'`	Polak-Ribiére Conjugate Gradient
`'trainoss'`	One Step Secant
`'traingdx'`	Variable Learning Rate Gradient Descent
`'traingdm'`	Gradient Descent with Momentum
`'traingd'`	Gradient Descent

Exercise 4.1

- Apply MPL network to find a non-linear relationship between a single input and a single output.
- Data set: *example_4_1.txt* (with 701 examples):
  - *X* - input
  - *Y* - output.
- Uncompleted MATLAB-scripts
  - *Exercise_4_1.m* - Script that steps you through this exercise
  - *PlotData.m* - Function to display the dataset

- Your task: to complete these MATLAB-scripts using the suitable MATLAB functions and run the program for different model parameters.

Homework 4: (deadline for submission is 14th June 2022)

- Extend the MATLAB script of Exercise 4.1 to approximate a mapping function from the input variable X to output variable Y using both MPL network and linear regression method.
- Print, plot and analyze the obtained results.