1BM56 / 1MG56
Business Intelligence

Lecture 9, 2015-2016
Introduction to neural networks

dr. Anna Wilbik

Pav. D17, a.m.wilbik@tue.nl

Technische Universiteit **Eindhoven** University of Technology

Where innovation starts

Outline

- Introduction
- Neuron
- Multilayer perceptron (MLP)
- Back propagation algorithm

Why neural networks?

- (Neuro-)Biology / (Neuro-)Physiology / Psychology:
  - Exploit similarity to real (biological) neural networks.
  - Build models to understand nerve and brain operation by simulation.
- Computer Science / Engineering / Economics
  - Mimic certain cognitive capabilities of human beings.
  - Solve learning/adaptation, prediction, and optimization problems.

Why neural networks?

- Inductive Reasoning.
  - Given input and output data (training examples), we construct the rules.
- Computation is collective, asynchronous, and parallel.
- Memory is distributed, internalized, short term and content addressable.
- Fault tolerant, redundancy, and sharing of responsibilities.
- Inexact.
- Dynamic connectivity

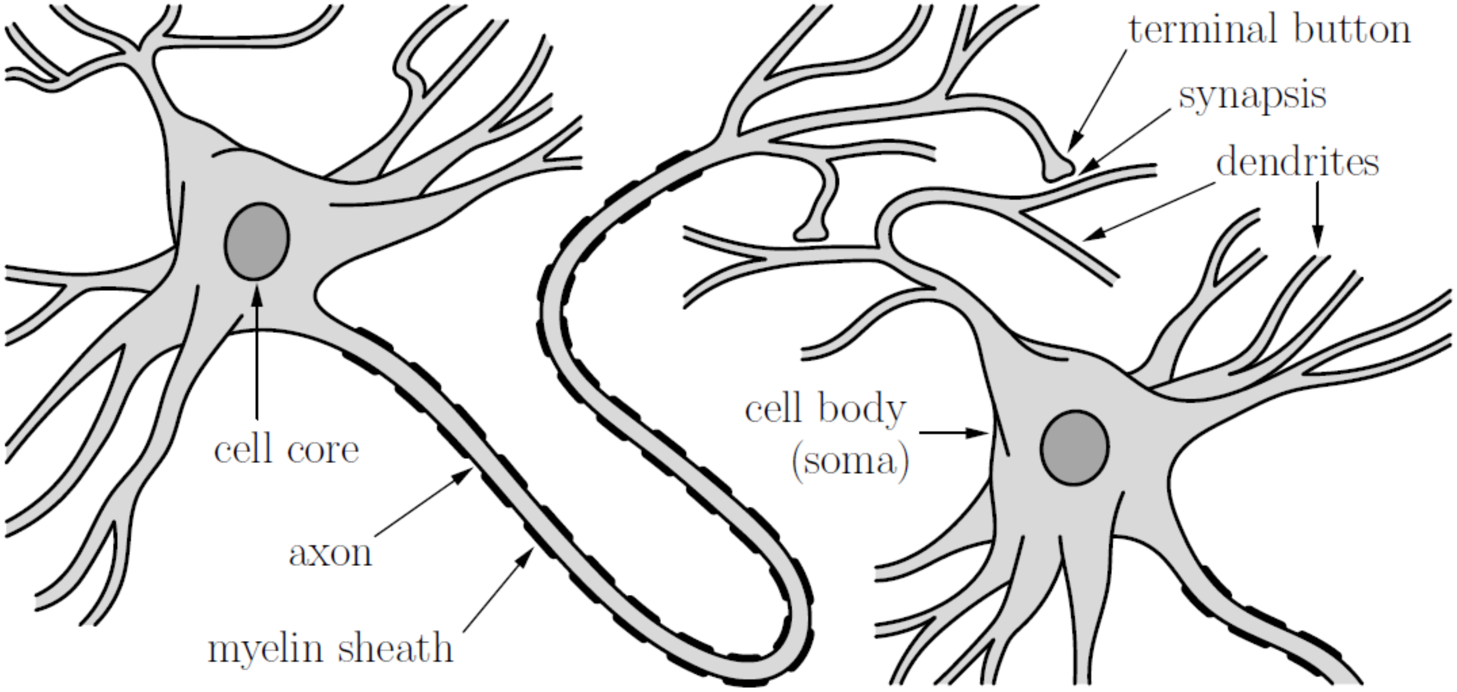
Applications (1)

- classification
  - marketing: consumer spending pattern classification
  - defense: radar and sonar image classification
  - agriculture & fishing: fruit and catch grading
  - medicine: ultrasound and electrocardiogram image classification, EEGs, medical diagnosis
- recognition and identification
  - general computing: speech, vision and handwriting recognition
  - finance: signature verification and bank note verification

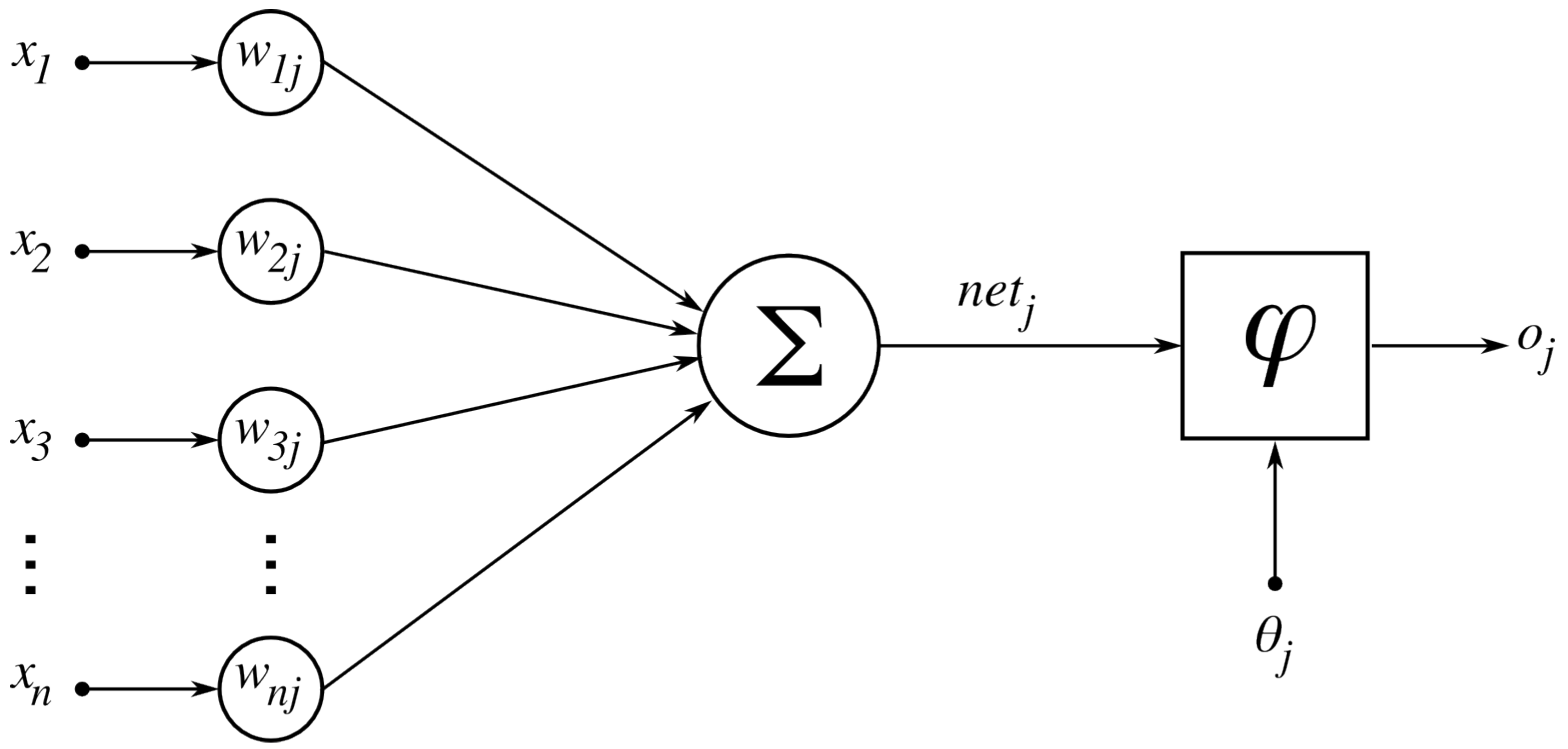
Applications (2)

- assessment
  - engineering: product inspection monitoring and control
  - defense: target tracking
  - security: motion detection, surveillance image analysis and fingerprint matching
- forecasting and prediction
  - finance: foreign exchange rate and stock market forecasting
  - agriculture: crop yield forecasting
  - marketing: sales forecasting
  - meteorology: weather prediction

Biological Background



Artificial Neuron (Perceptron)



Artificial Neuron

Each neuron u possesses:

- the network inputs
- the activation
- the output

Each neuron u may possesses an external input ex_u .

Each neuron u possesses three functions:

- the network input function f_{net}
- the activation function f_{act}
- the output function f_{out}

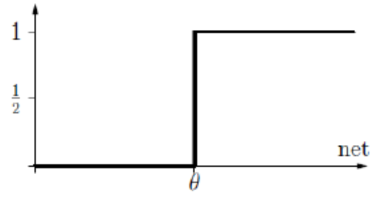
which are used to compute the values of the state variables.

TU/e Technische Universiteit Eindhoven University of Technology

Activation functions (1)

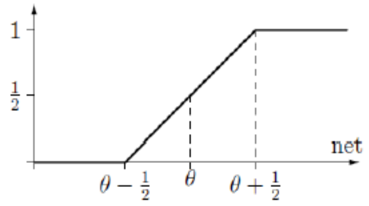
- **step function**

$$f_{act}(net, \theta) = \begin{cases} 1 & \text{if } net \geq \theta \\ 0 & \text{otherwise} \end{cases}$$



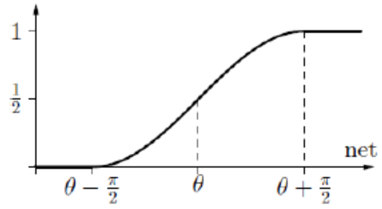
- **semi-linear function**

$$f_{act}(net, \theta) = \begin{cases} 1 & \text{if } net \geq \theta + \frac{1}{2} \\ 0 & \text{if } net < \theta - \frac{1}{2} \\ (net - \theta) + \frac{1}{2} & \text{otherwise} \end{cases}$$



- **sine until saturation**

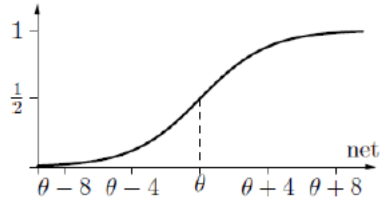
$$f_{act}(net, \theta) = \begin{cases} 1 & \text{if } net \geq \theta + \frac{\pi}{2} \\ 0 & \text{if } net < \theta - \frac{\pi}{2} \\ \frac{\sin(net - \theta) + 1}{2} & \text{otherwise} \end{cases}$$



Activation functions (2)

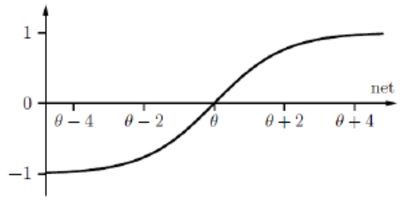
- **logistic function**

$$f_{act}(net, \theta) = \frac{1}{1 + e^{-(net-\theta)}}$$

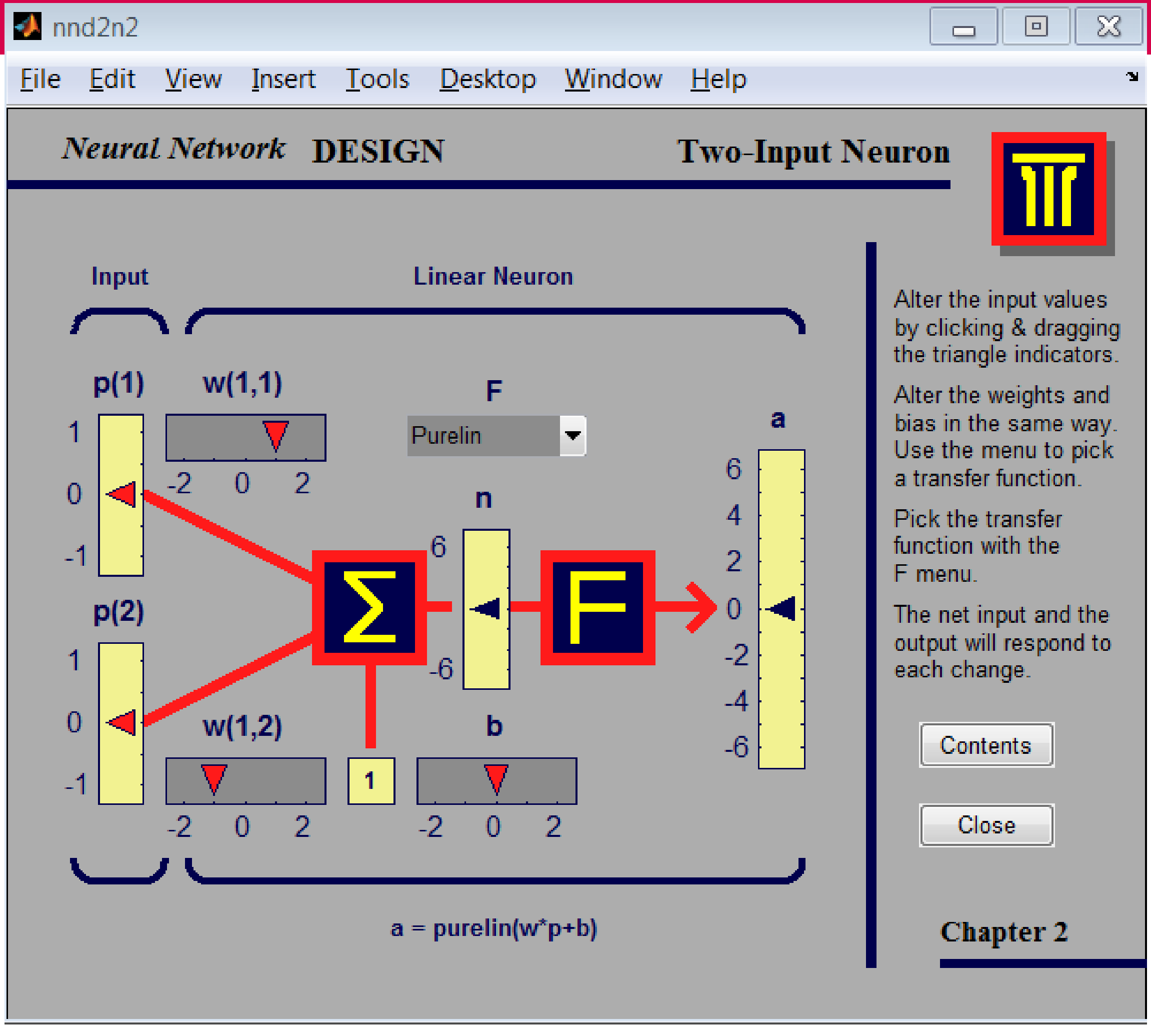


- **tangens hyperbolicus**

$$f_{act}(net, \theta) = \frac{2}{1 + e^{-2(net-\theta)}-1}$$



Demo – MATLAB: nnd2n2



Alter the input values by clicking & dragging the triangle indicators.

Alter the weights and bias in the same way. Use the menu to pick a transfer function.

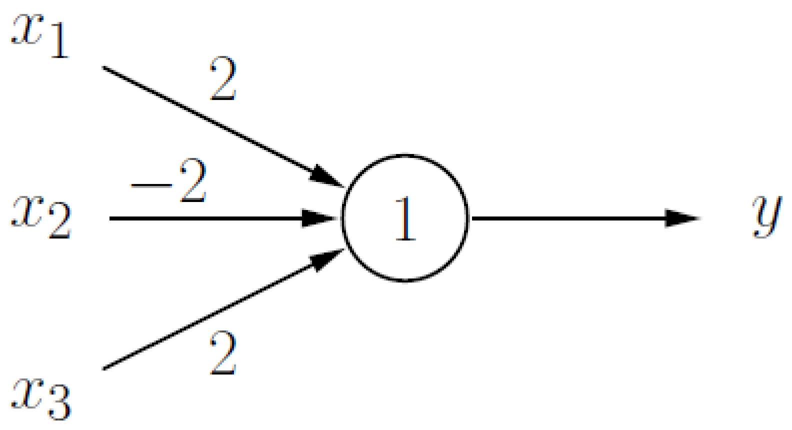
Pick the transfer function with the F menu.

The net input and the output will respond to each change.

a = purelin(w*p+b)

Exercise 1

Compute the outputs for the artificial neuron shown below. Assume that inputs are either 0 and 1.
As activation function use step function with θ=1.
Input function is weighted sum of inputs. Weights are depicted on the connections. Output function is identity.

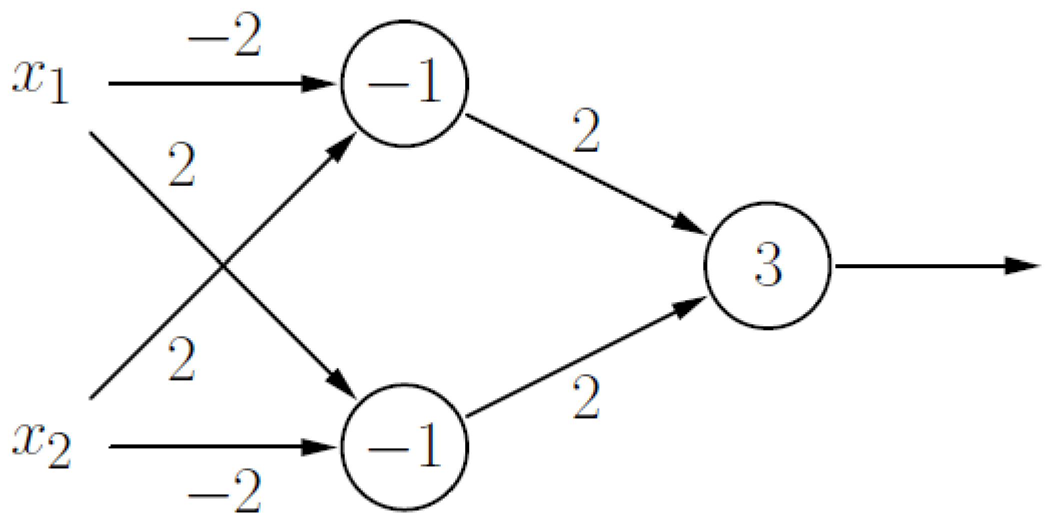


Exercise 1 - solution

x_1	x_2	x_3	$\Sigma_i w_i x_i$	y
0	0	0	0	0
1	0	0	2	1
0	1	0	-2	0
1	1	0	0	0
0	0	1	2	1
1	0	1	4	1
0	1	1	0	0
1	1	1	2	1

Exercise 2

Compute the outputs for the small net shown below. Assume that inputs are either 0 and 1. Use step function with θ values shown in the circles. Input function is weighted sum of inputs. Weights are depicted on the connections. Output function is identity.



Technische Universiteit
Eindhoven
University of Technology

Exercise 2 - solution

x_1	x_2	u_1: $\Sigma_i w_i x_i$	y_1	u_2: $\Sigma_i w_i x_i$	y_2	u_3: $\Sigma_i w_i x_i$	y_3
0	0	0	1	0	1	4	1
1	0	-2	0	2	1	2	0
0	1	2	1	-2	0	2	0
1	1	0	1	0	1	4	1

Neural Networks

An (artificial) neural network is a (directed) graph $G = (U, C)$

- whose nodes $u \in U$ are called **neurons** or units and
- whose edges $c \in C$ are called **connections**.

The set U of nodes is partitioned into:

- the set U_{in} of input neurons,
- the set U_{out} of output neurons,
- the set U_{hidden} of hidden neurons.

Remark: a node can be input neuron and output neuron simultaneously, but not a hidden neuron.

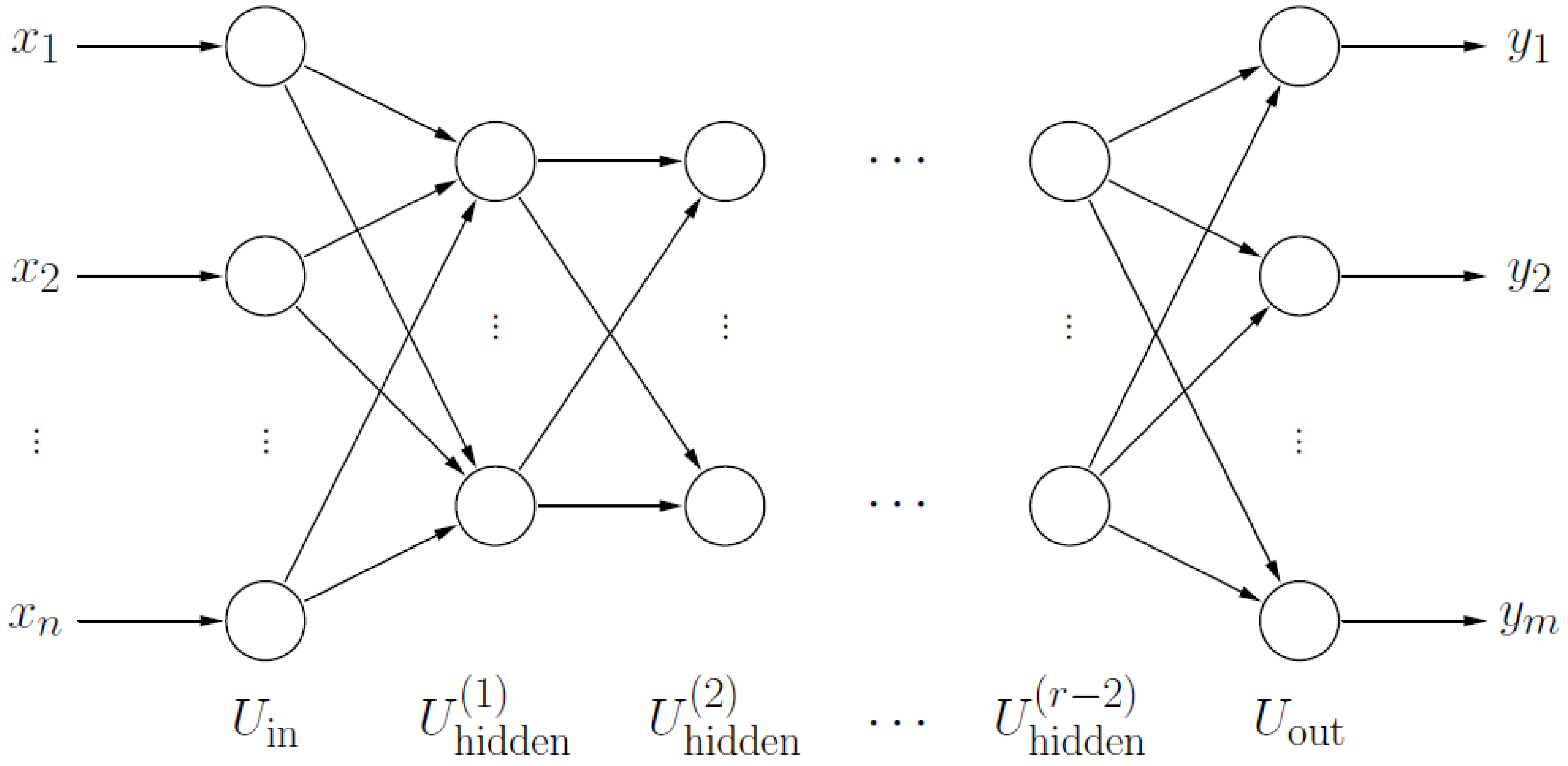
Each connection $(v,u) \in C$ possesses a weight w_{uv}

Neural Networks

- architecture and weights determine what a network does:
- different possibilities:
  - feed forward networks (MLP)
  - recurrent networks (Elman networks, Hopfield networks)
  - Self Organizing Maps (SOMs)
  - unstructured networks

Multilayer Perceptron (MLP)

- **feed forward network with strictly layered structure**



x_1
x_2
x_n
y_1
y_2
y_m
U_{in}
$U^{(1)}_{\text{hidden}}$
$U^{(2)}_{\text{hidden}}$
$U^{(r-2)}_{\text{hidden}}$
U_{out}

Multilayer Perceptron (MLP)

- The network input function of each hidden neuron and of each output neuron is the weighted sum of its inputs, i.e.

$$\forall u \in U_{hidden} \cup U_{out} : f_{net}^{(u)}(\vec{w}_u; \vec{in}_u) = \vec{w}_u \cdot \vec{in}_u = \sum_{v \in pred(u)} w_{uv} \cdot in_v$$

- The activation function of each hidden neuron is logistic function (in general case, it may be an unipolar sigmoid function, i.e. a monotonously increasing function)
- The output function is the identity (in general case, it may be a linear function)

Technische Universiteit **Eindhoven** University of Technology

Applications for MLP

- Supervised Learning (Input/Output Mapping):
  - Classification (discrete outputs)
  - Regression (numeric outputs)
- Reinforcement Learning (Output is not perfectly known)
  - Control (e.g., autonomous driving);
  - Game Playing (e.g., checkers);

How to build a MLP? (architecture)

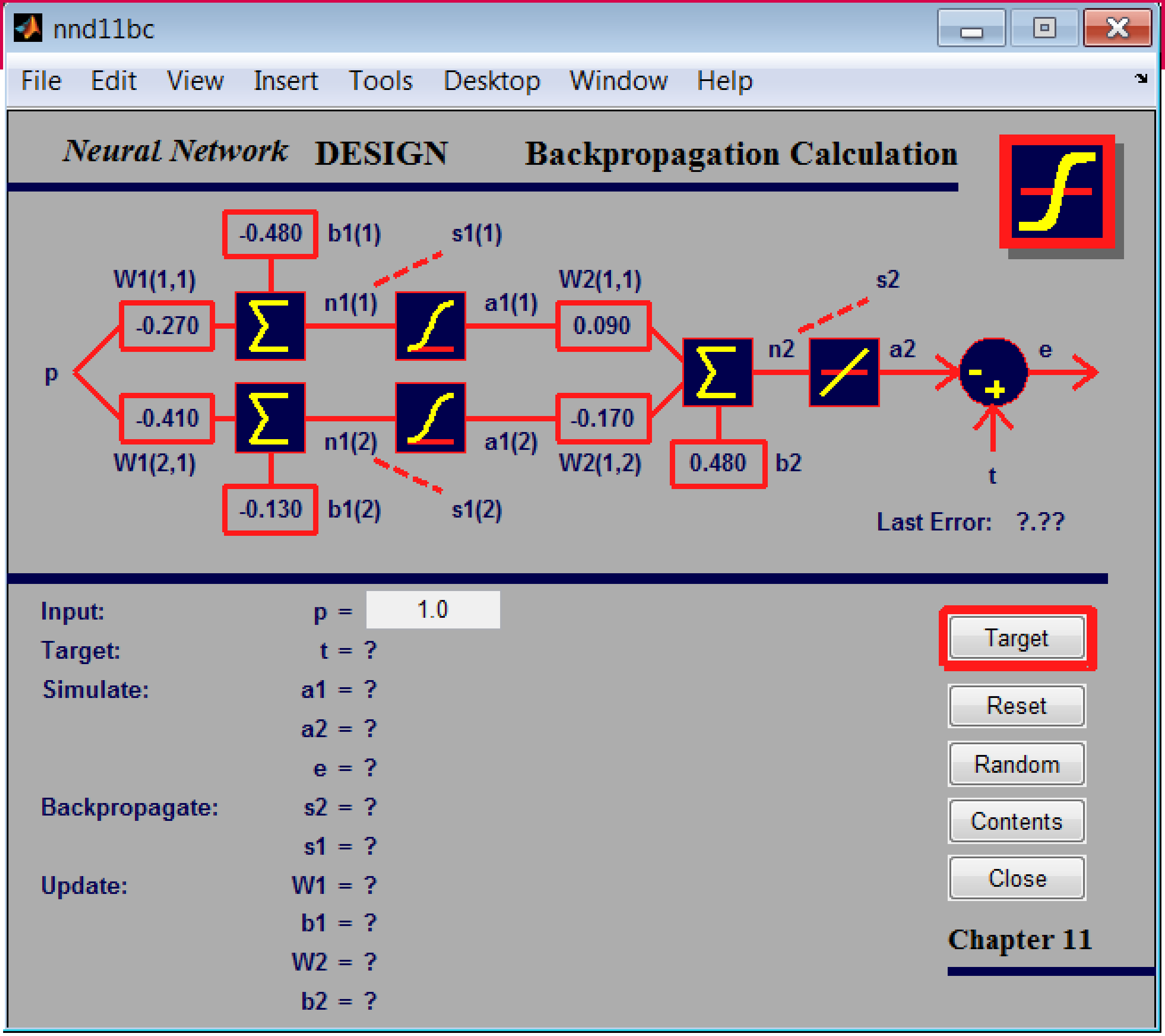
- number of nodes in input layer – depend on the data
- number of nodes in output layer – depend on the number of classes (better to have one node per class then use coding)
- number of hidden layers?
  - any continuous function mapping can be learned with one hidden layer
  - discontinuous complex functions can be learned with more hidden layers
- how many nodes in hidden layers?
  - too few hidden units will generally leave high training and generalization errors due to under-fitting
  - too many hidden units will result in low training errors, but will make the training unnecessarily slow, and will often result in poor generalization (error on test set) due to over-fitting
- trail and error!

How to build a MLP? (weights)

Use the data!

- backpropagation method
- genetic algorithms
- ...

Backpropagation method (BP)



Input: p = 1.0
Target: t = ?
Simulate: a1 = ?
a2 = ?
e = ?
Backpropagate: s2 = ?
s1 = ?
Update: W1 = ?
b1 = ?
W2 = ?
b2 = ?

MATLAB: nnd11bc

TU/e Technische Universiteit Eindhoven University of Technology

Backpropagation method (BP)

- **adjust the network weights w_{ij} in order to minimize the sum-squared error function**

$$e = \sum_{i=1}^{n} \sum_{k} (out_k^i - y_k^i)^2$$

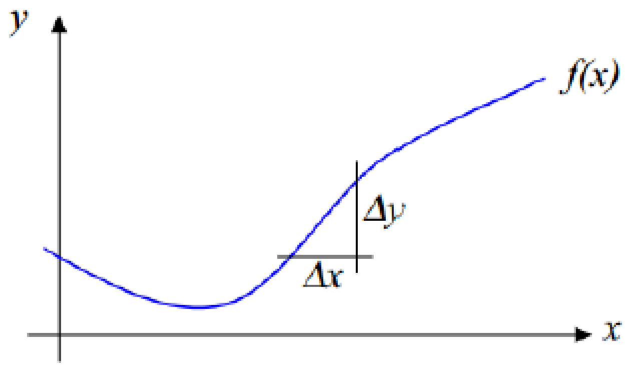
- **the aim of learning is to minimize this error by adjusting the weights w_{ij}. Typically we make a series of small adjustments to the weights $w_{ij} \rightarrow w_{ij} + \Delta w_{ij}$ until the error is “small enough”.**
- **A systematic procedure for doing this requires the knowledge of how the error varies as we change the weights w_{ij}, i.e. the gradient of e with respect to w_{ij} .**

Backpropagation method (BP)

1. Forward pass
   - Input is applied to the network and propagated to the output
   - Synaptic weights stay frozen
   - Based on the desired response, error signal is calculated
2. Backward pass
   - Error signal is propagated backwards from output to input
   - Synaptic weights are adjusted according to the error gradient

Gradient descent minimization

The gradient of f(x) at a particular value of x, as we change x can be approximated by Δy/Δx – partial derivative of f(x) with respect to x.



Change the value of x to minimize f(x).

What we need to do depends on the gradient of f(x).

- If $\frac{\partial f}{\partial x} > 0$ then f(x) increases as x increases – decrease x
- If $\frac{\partial f}{\partial x} < 0$ then f(x) increases as x decreases – increase x
- If $\frac{\partial f}{\partial x} = 0$ then f(x) is at a maximum or minimum – do not change x

We can decrease f(x) by changing x by the amount:

$$\Delta x = x_{new} - x_{old} = -\eta \frac{\partial f}{\partial x}$$

Backpropagation method (BP)

- **Correction of weights is proportional to partial derivative of the error**
- **Update rule:**
- **On-line learning:** $\Delta w_{uk} = -\eta \delta_u in_k$
- **Off-line (batch) learning:** $\Delta w_{uk} = -\eta \sum_{i=1}^{N} \delta_u in_k$
  - **neuron u is an output neuron:**

$$\delta_u = (y_u - out_u)\, out_u\, (1 - out_u)$$

  - **neuron u is a hidden neuron:**

$$\delta_u = \left(\sum_{s \succ (u)} \delta_s w_{su} \right) out_u (1 - out_u)$$

Technische Universiteit Eindhoven University of Technology

Summary of backpropagation algorithm

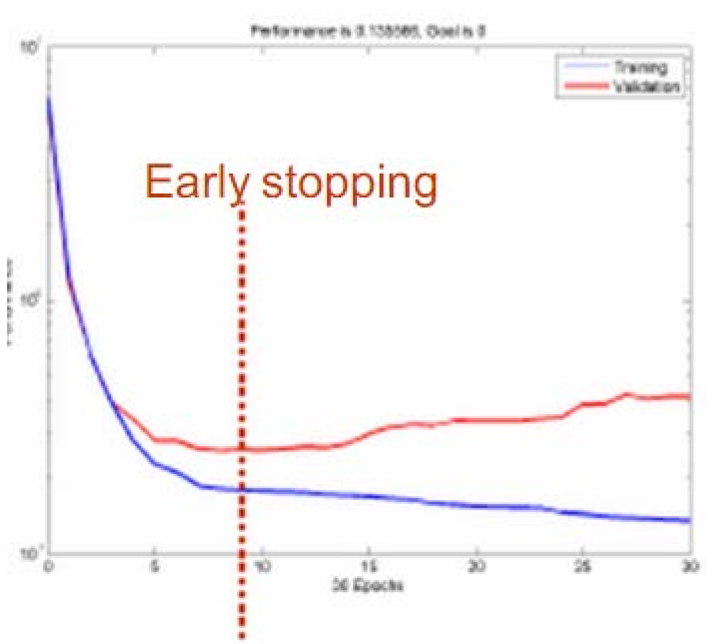
1. **Initialization**
   - Pick weights and biases from the uniform distribution with zero mean and variance that induces local fields between the linear and saturated parts of logistic function
2. **Presentation of training samples**
   - For each sample from the epoch, perform forward pass and backward pass
3. **Forward pass**
   - Propagate training sample from network input to the output
   - Calculate the error signal
4. **Backward pass**
   - Recursive computation of local gradients from output layer toward input layer
   - Adaptation of synaptic weights according to generalized delta rule
5. **Iteration**
   - Iterate steps 2-4 until stopping criterion is met

Stopping criteria

1. Gradient vector
   - Euclidean norm of the gradient vector reaches a sufficiently small gradient
2. Output error
   - Output error is small enough
   - Rate of change in the average squared error per epoch is sufficiently small
3. Generalization performance
   - Generalization performance has peaked or is adequate
4. Max number of iterations
   - We are out of time ...

Improving generalization

- Keep the network small (avoid overfitting)
- Early stopping
  - Data divided intro 3 sets:
    - Training set
    - Validation set – used for early stopping, when the error starts to increase
    - Test set - used for final estimation of network performance and for comparison of various models

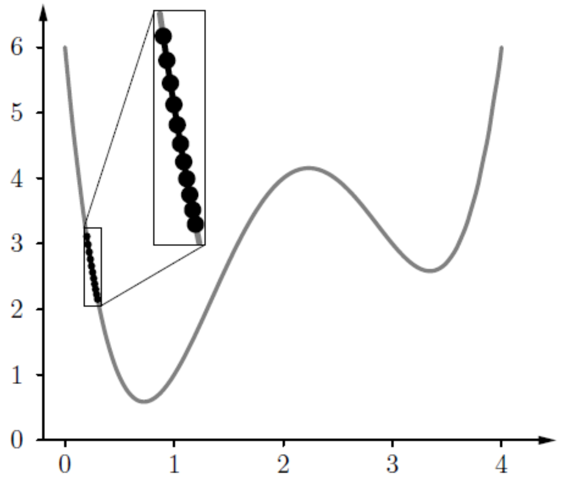


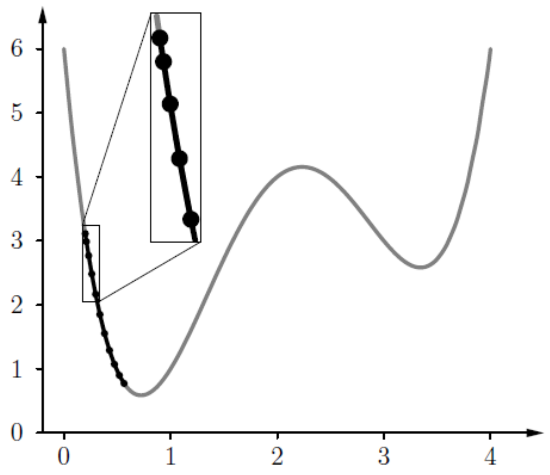
Deficiencies of backpropagation

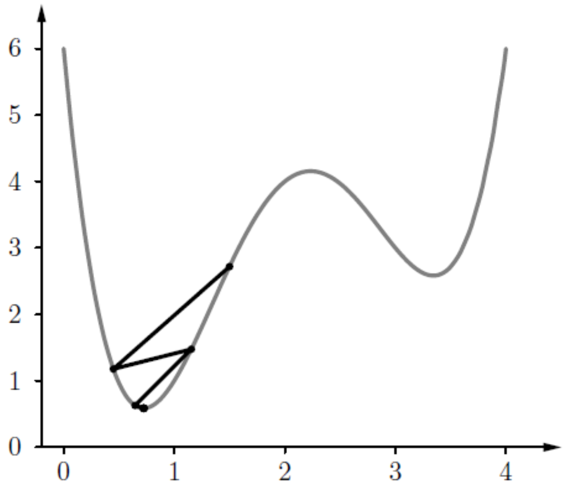
- **Long training process**
  - **possibly due to non-optimum learning rate (advanced algorithms address this problem)**
- **Network paralysis**
  - **combination of sigmoidal activation and very large weights can decrease gradients almost to zero $\rightarrow$ training is almost stopped**
- **Local minima**
  - **error surface of a complex network can be very complex, with many hills and valleys**
  - **Gradient methods can get trapped in local minima**
  - **Solutions: probabilistic learning methods (simulated annealing, ...)**

Speeding up backpropagation

- **Steepest descent**
- **Momentum rule**
- **Variable learning rate**

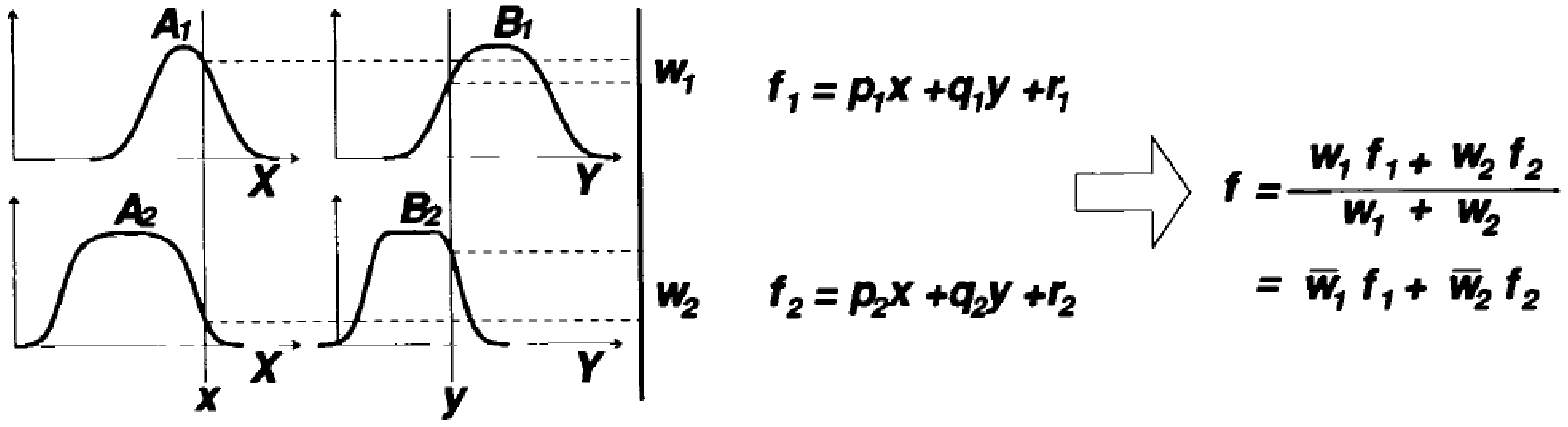






Technische Universiteit
Eindhoven
University of Technology

NN and FIS (ANFIS)

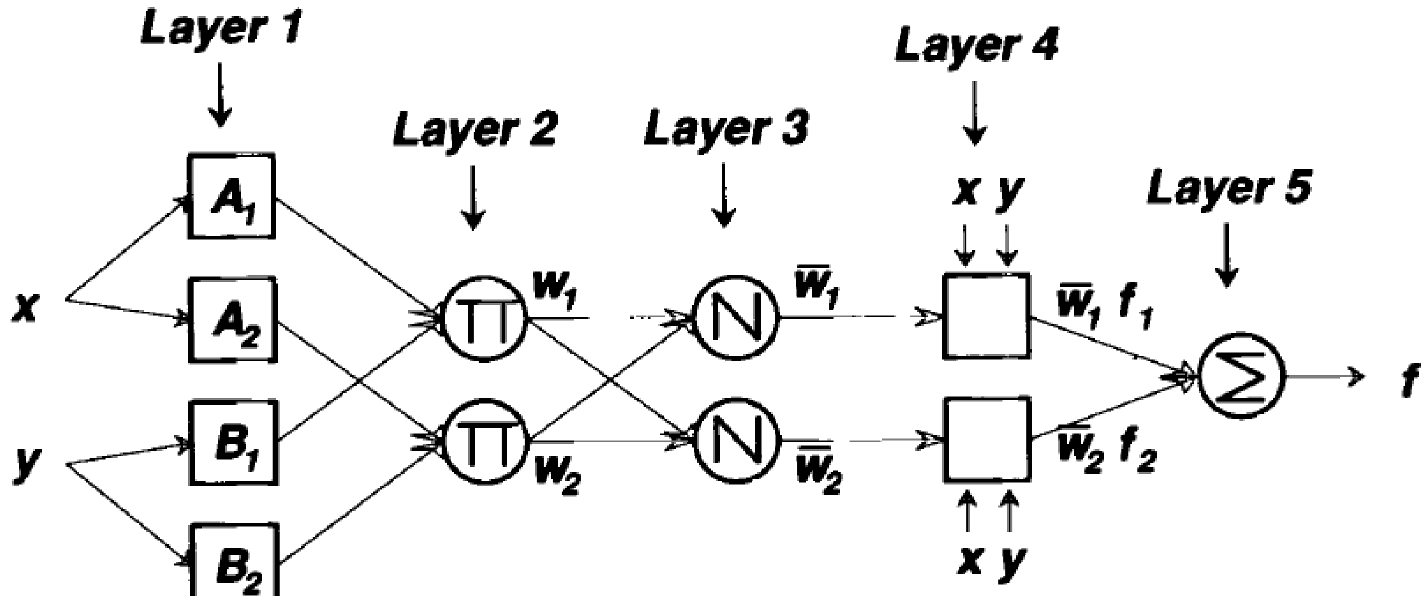


$$f_1 = p_1x + q_1y + r_1$$

$$f_2 = p_2x + q_2y + r_2$$

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} = \overline{w}_1 f_1 + \overline{w}_2 f_2$$

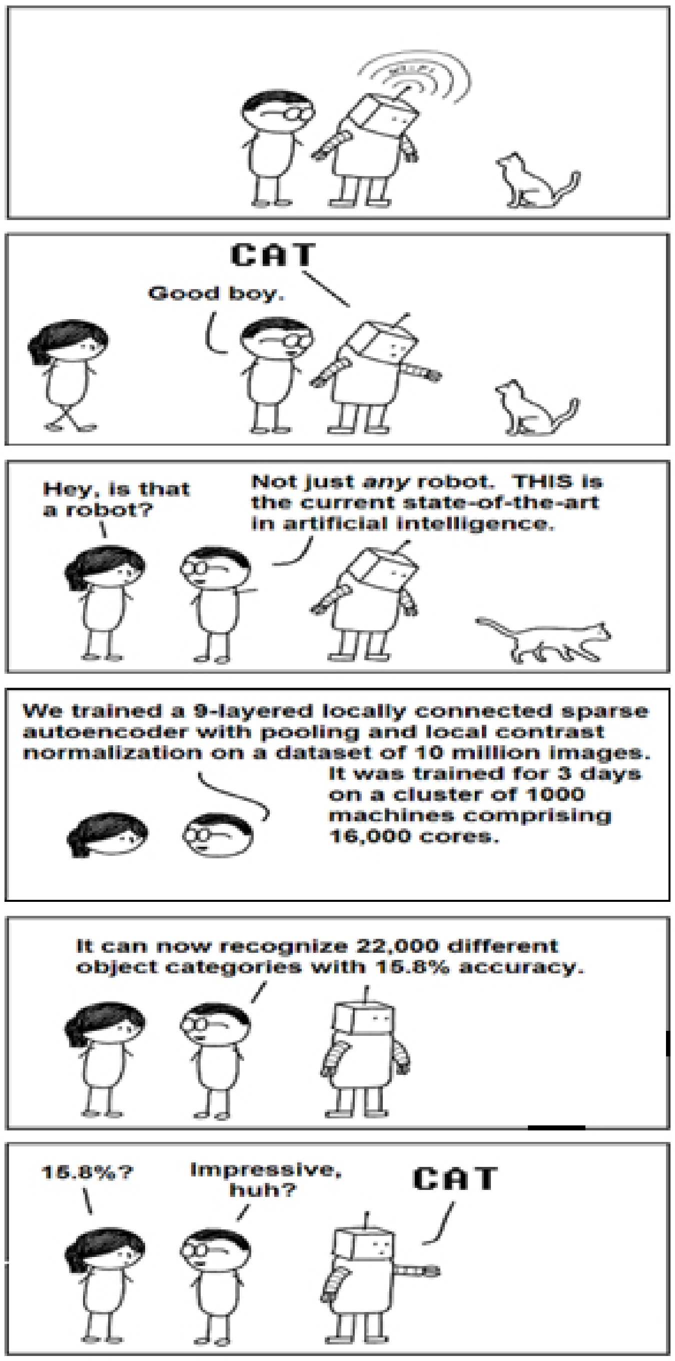
(a)



(b)

Summary

- Artificial neural networks – to mimic certain cognitive capabilities of human beings
- Neuron
- Multilayer perceptron (MLP)
- Back propagation algorithm
- MATLAB demo: nnd


WI-FI
CAT
Good boy.
Hey, is that a robot?
Not just *any* robot. THIS is the current state-of-the-art in artificial intelligence.
We trained a 9-layered locally connected sparse autoencoder with pooling and local contrast normalization on a dataset of 10 million images.
It was trained for 3 days on a cluster of 1000 machines comprising 16,000 cores.
It can now recognize 22,000 different object categories with 15.8% accuracy.
15.8%?
Impressive, huh?
CAT