Artificial Neural Networks

Mihai Oltean

www.cs.ubbcluj.ro/~moltean

moltean@cs.ubbcluj.ro

STRUCTURE

- Why we need ANNs?
- Biological neurons and networks / brain
- Perceptron / Artificial Neuron
- Artificial Neural Networks
- Training ANNs
  - Perceptron' algorithm
  - BackPropagation
  - Evolutionary Training
- Examples

LEARNING PARADIGMS

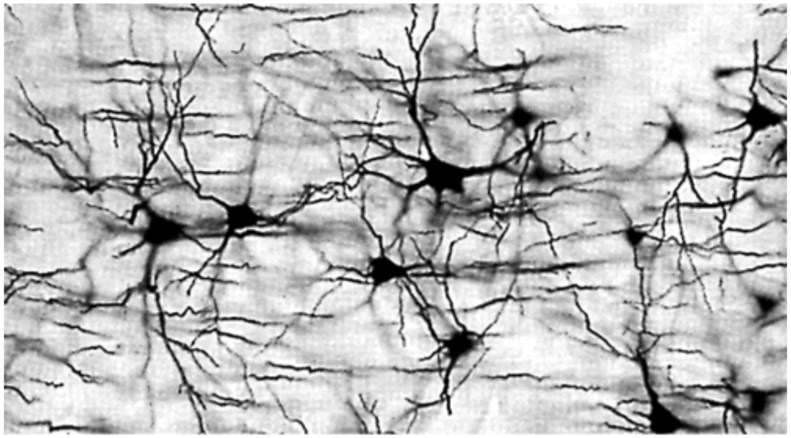
- **Supervised learning**
  - We have a set of example (x, f(x)).
- **Unsupervised learning**
  - In unsupervised learning we are given some data *x*, and the cost function to be minimized can be any function of the data *x*.
- **Reinforcement learning**
  - Data *x* is usually not given, but generated by an agent's interactions with the environment. At each point in time *t*, the agent performs an action y_t and the environment generates an observation x_t and an instantaneous cost c_t, according to some (usually unknown) dynamics. The aim is to discover a *policy* for selecting actions that minimizes some measure of a long-term cost, i.e. the expected cumulative cost. The environment's dynamics and the long-term cost for each policy are usually unknown, but can be estimated.

Why Artificial Neural Networks?

- Some tasks can be done easily (effortlessly) by humans but are hard by conventional paradigms on Von Neumann machine with algorithmic approach
  - Pattern recognition (old friends, hand-written characters, voice)
  - Content addressable recall
  - Approximate, common sense reasoning (driving, playing piano, baseball player)
- These tasks are often ill-defined, experience based, hard to apply logic

Human Brain

- 10.000.000.000 neurons
- 5.000 connections / neuron (in average)
- New connections between neurons can be developed during lifetime
- Small animals have fixed brain.



COMPARISON

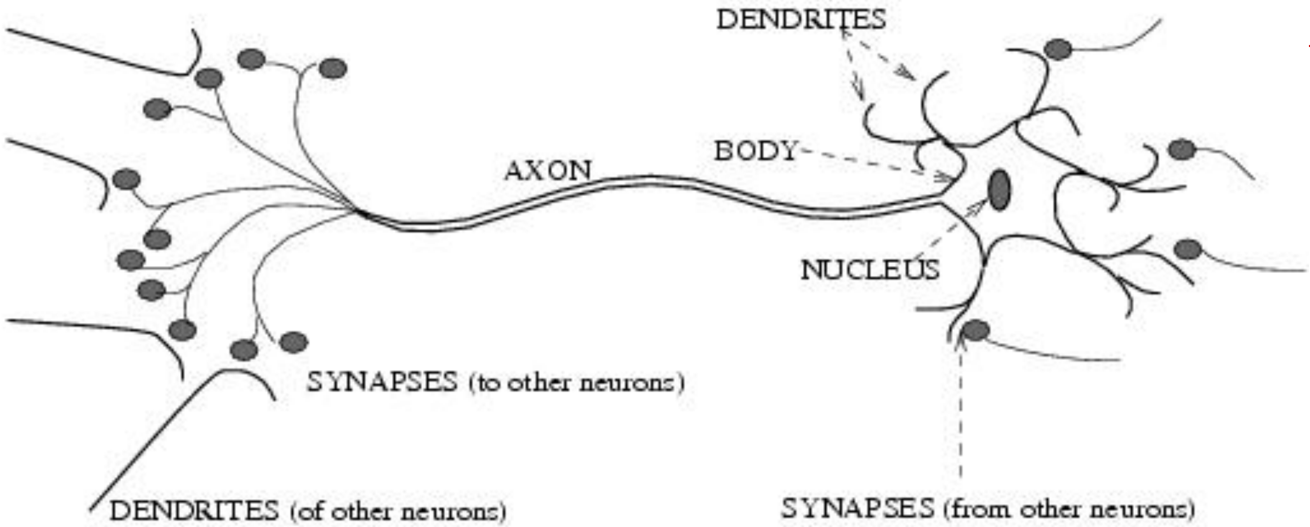
Modern Computers

- One or a few high speed (ns) processors with considerable computing power
- One or a few shared high speed buses for communication
- Sequential memory access by address
- Problem-solving knowledge is separated from the computing component
- Hard to be adaptive

Human Brain

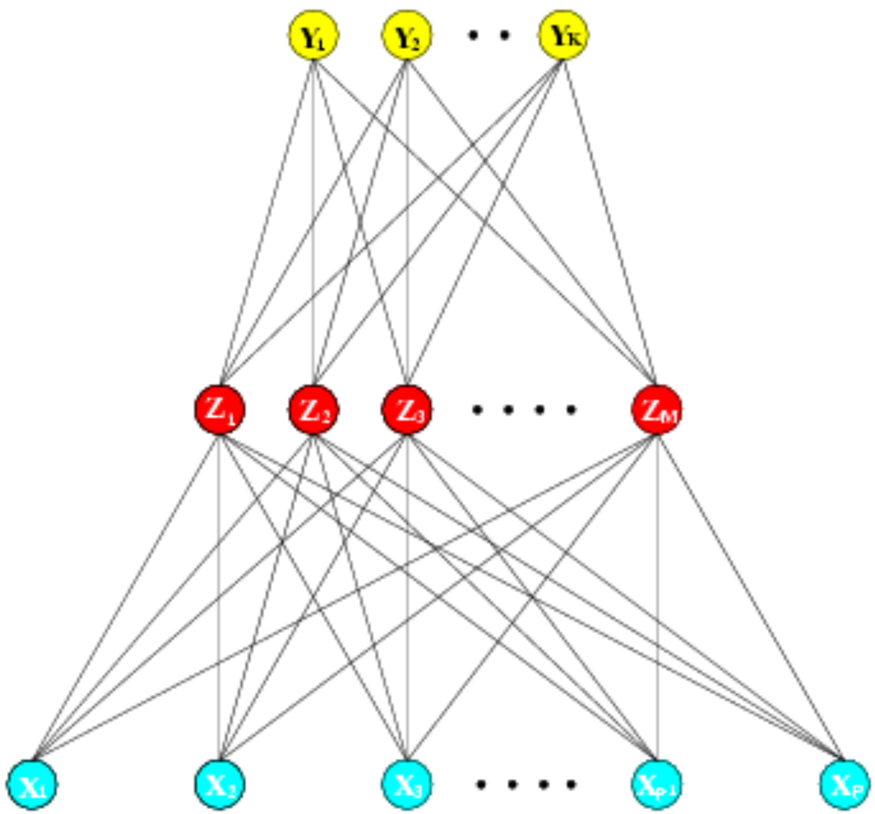
- Large # (10^{11}) of low speed processors (ms) with limited computing power
- Large # (10^{15}) of low speed connections
- Content addressable recall (CAM)
- Problem-solving knowledge resides in the connectivity of neurons
- Adaptation by changing the connectivity

• Biological neural activity



- Each neuron has a *body*, an *axon*, and many *dendrites*
  - Can be in one of the two states: *firing* and *rest.*
  - Neuron fires if the total incoming stimulus exceeds the threshold
- *Synapse*: thin gap between axon of one neuron and dendrite of another.
  - Signal exchange
  - Synaptic strength/efficiency

An example of ANN



Schematic of a single hidden layer, feed-forward neural network.

WHAT IS AN (ARTIFICIAL) NEURAL NETWORK?

- A set of **nodes** (units, neurons, processing elements)
  - Each node has input and output
  - Each node performs a simple computation by its **node function**
- **Weighted connections** between nodes
  - Connectivity gives the structure/architecture of the net
  - What can be computed by a NN is primarily determined by the connections and their weights
- *A very much simplified version of networks of neurons in animal nerve systems*

ANN

- **Nodes**
  - input
  - output
  - node function
- **Connections**
  - connection strength

Bio NN

- **Cell body**
  - signal from other neurons
  - firing frequency
  - firing mechanism
- **Synapses**
  - synaptic strength

Benefits of using ANNs

- Learning: ANN have the ability to learn based on the so called learning stage.
- Auto organization: a ANN creates its own representation of the data given in the learning process.
- Tolerance to faults: because ANN store redundant information, partial destruction of the neural network do not damage completely the network response.
- Flexibility: ANN can handle input data without important changes like noisy signals or others changes in the given input data (e.g. if the input data is an object, this can be a little different without problems to the ANN response).
- Real Time: ANN are parallel structures; if they are implemented in this way using computers or special hardware real time can be achieved.
- Scalability: An ANN can be easily ported to fit any problem from a particular problem area.
- Highly parallel, simple local computation (at neuron level) achieves global results as emerging property of the interaction (at network level)

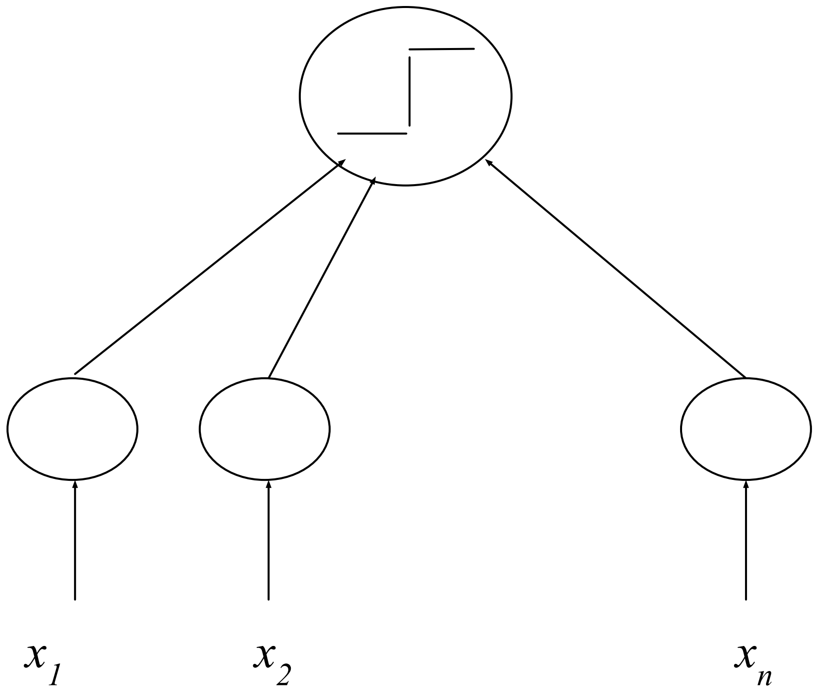
History of NN

- **Pitts & McCulloch (1943)**
  - First mathematical model of biological neurons
  - All Boolean operations can be implemented by these neuron-like nodes (with different threshold and excitatory/inhibitory connections).
  - Competitor to Von Neumann model for general purpose computing device
  - Origin of automata theory.
- **Hebb (1949)**
  - Hebbian rule of learning: increase the connection strength between neurons i and j whenever both i and j are activated.
  - Or increase the connection strength between nodes i and j whenever both nodes are simultaneously ON or OFF.

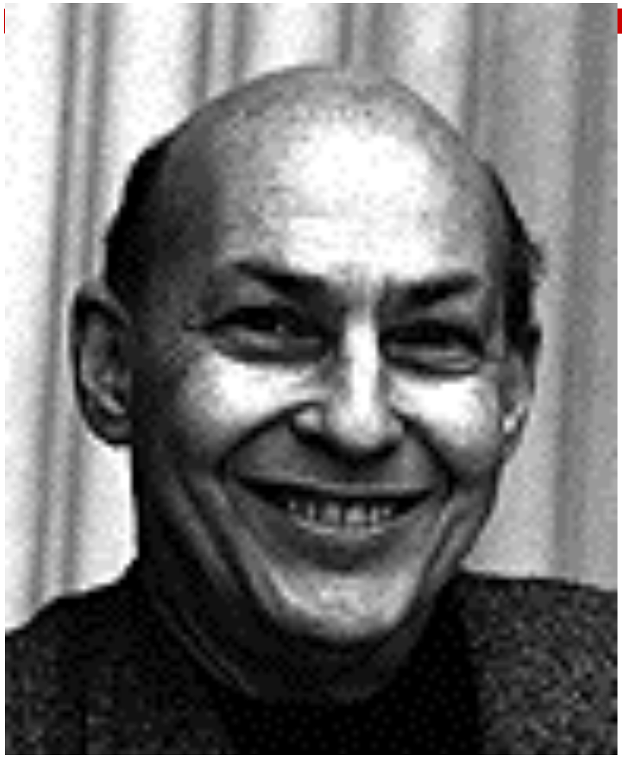
EARLY BOOMING (50's – EARLY 60's)

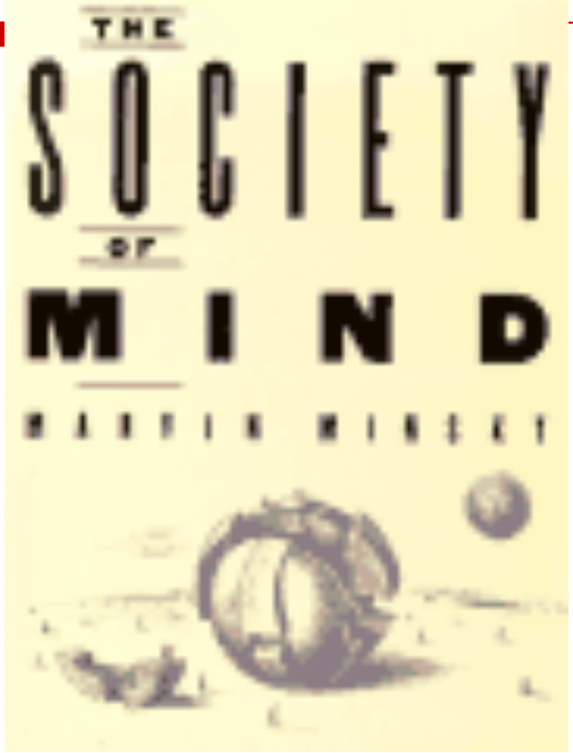
- Rosenblatt (1958)
  - Perceptron: network of threshold nodes for pattern classification
  - Perceptron learning rule
  - Perceptron convergence theorem: everything that can be represented by a perceptron can be learned
- Widow and Hoff (1960, 1962)
  - Learning rule based on gradient descent (with differentiable unit)
- Minsky's attempt to build a general purpose machine with Pitts/McCullock units

Perceptron with step function



Marvin Minsky (MIT AI Lab)

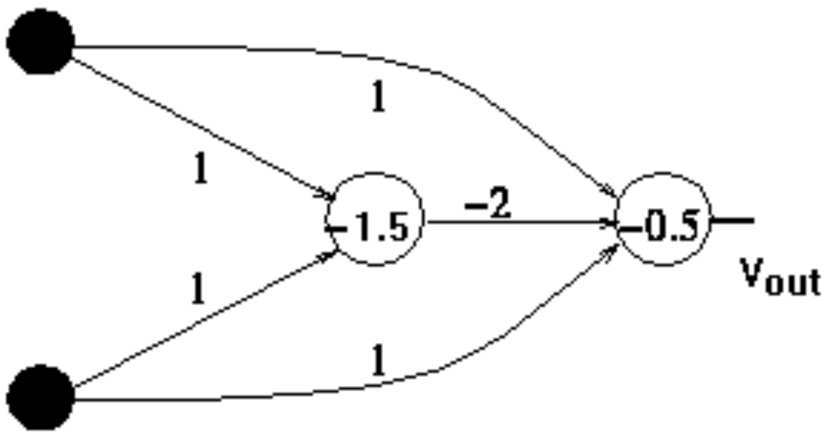




THE SETBACK (MID 60'S – LATE 70'S)

- Serious problems with perceptron model (Minsky's book 1969)
  - Single layer perceptrons cannot represent (learn) simple functions such as XOR
  - Multi-layer of non-linear units may have greater power but there is no learning rule for such nets
  - Scaling problem: connection weights may grow infinitely
- The first two problems overcame by latter effort in 80's, but the scaling problem persists
- Death of Rosenblatt (1964)
- Striving of Von Neumann machine and AI

mlp for xor



http://ecee.colorado.edu/~ecen4831/lectures/NNet3.html

Renewed enthusiasm and flourish (80's – present)

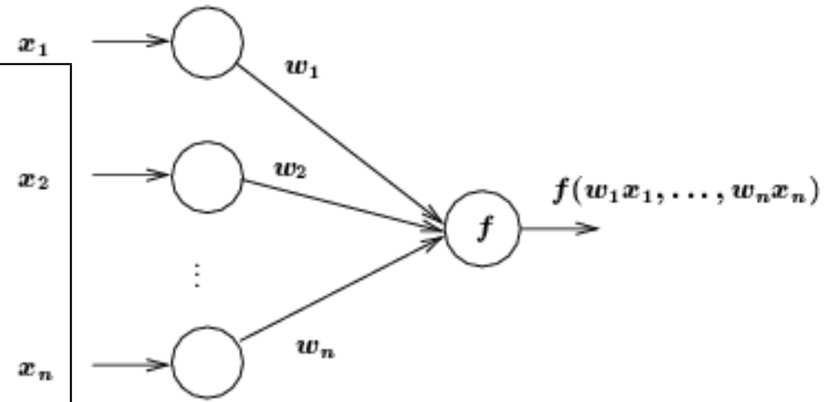
- New techniques
  - Backpropagation learning for multi-layer feed forward nets (with non-linear, differentiable node functions)
  - Thermodynamic models (Hopfield net, Boltzmann machine, etc.)
  - Unsupervised learning
- Impressive application (character recognition, speech recognition, text-to-speech transformation, process control, associative memory, etc.)
- Traditional approaches face difficult challenges
- Caution:
  - Don't underestimate difficulties and limitations
  - Poses more problems than solutions

ANN REQUIREMENTS

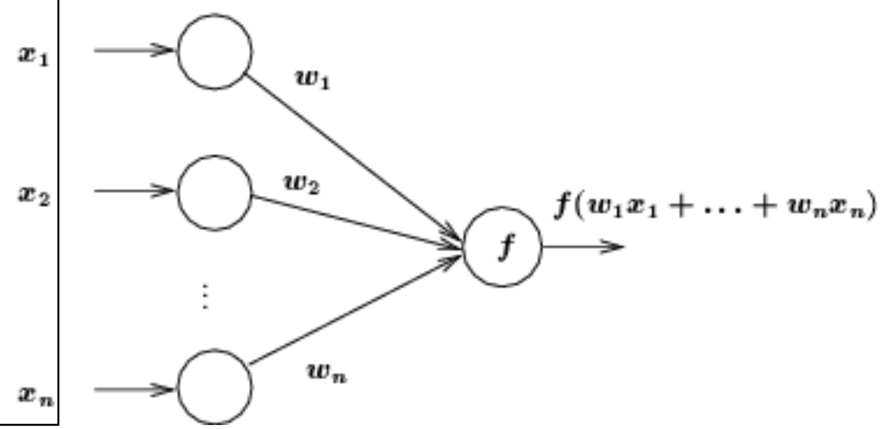
Structure
Activation function
Learning algorithm

Artificial Neuron Models

- Each node has one or more inputs from other nodes, and one output to other nodes
- Input/output values can be
  - Binary {0, 1}
  - Bipolar {-1, 1}
  - Continuous
- All inputs to one node come in at the same time and remain activated until the output is produced
- Weights associated with links



General neuron model



Weighted input summation

Node Function

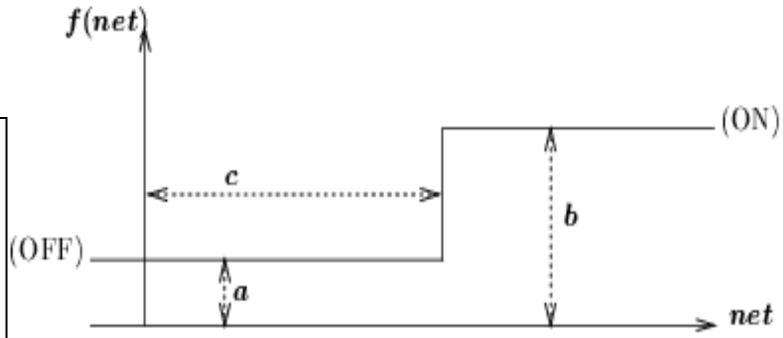
- Step (threshold) function

$$f(\text{net}) = \begin{cases} a & \text{if net} < c \\ b & \text{if net} > c \end{cases}$$

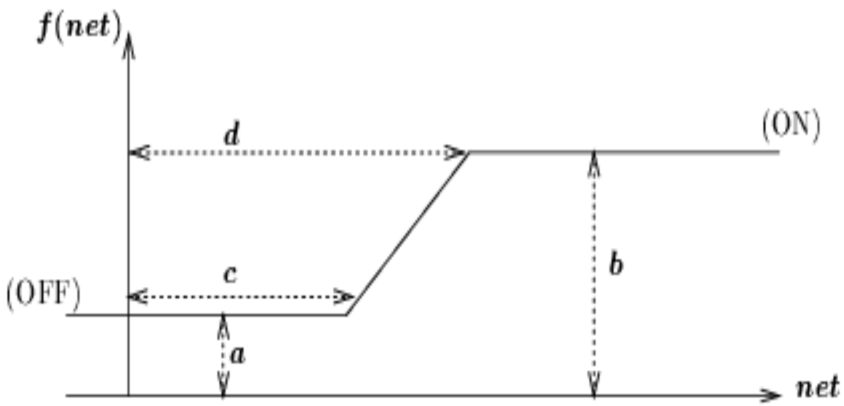
where c is called the threshold

- Ramp function

$$f(\text{net}) = \begin{cases} a & \text{if net} \le c \\ b & \text{if net} \ge d \\ a + \frac{(\text{net}-c)(b-a)}{(d-c)} & \text{otherwise} \end{cases}$$



Step function



Ramp function

Node Function (Sigmoid function)

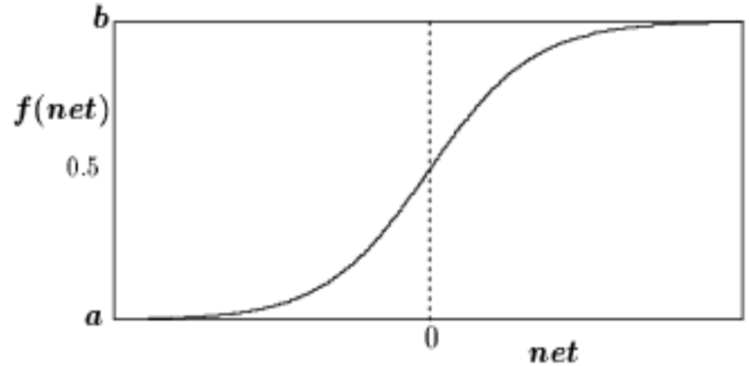
- S-shaped
- Continuous and everywhere differentiable
- Rotationally symmetric about some point (*net* = *c*)
- Asymptotically approach saturation points

$$\lim_{\text{net}\to-\infty} f(\text{net}) = a \qquad \lim_{\text{net}\to\infty} f(\text{net}) = b$$

- Examples:

$$f(\text{net}) = z + \frac{1}{1+\exp(-x\cdot\text{net}+y)}$$

$$f(\text{net}) = \tanh(x\cdot\text{net}-y)+z,$$



Sigmoid function

When $y = 0$ and $z = 0$:
$a = 0$, $b = 1$, $c = 0$.
When $y = 0$ and $z = -0.5$
$a = -0.5$, $b = 0.5$, $c = 0$.

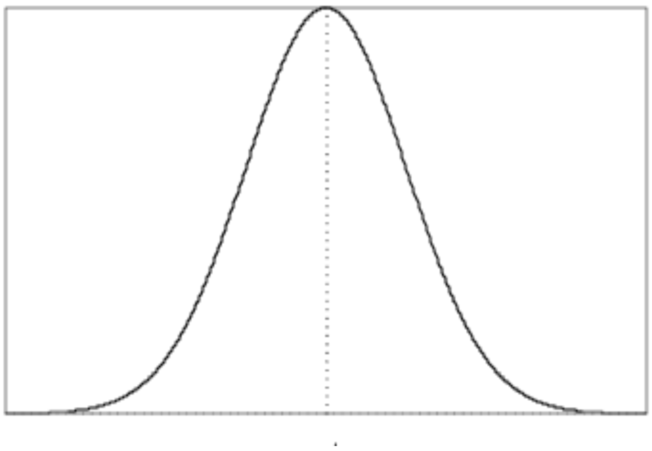
Larger x gives steeper curve

Node Function (Gaussian)

Bell-shaped (radial basis)

- Continuous
- *f*(*net*) asymptotically approaches 0 (or some constant) when |*net*| is large
- Single maximum
- Example:

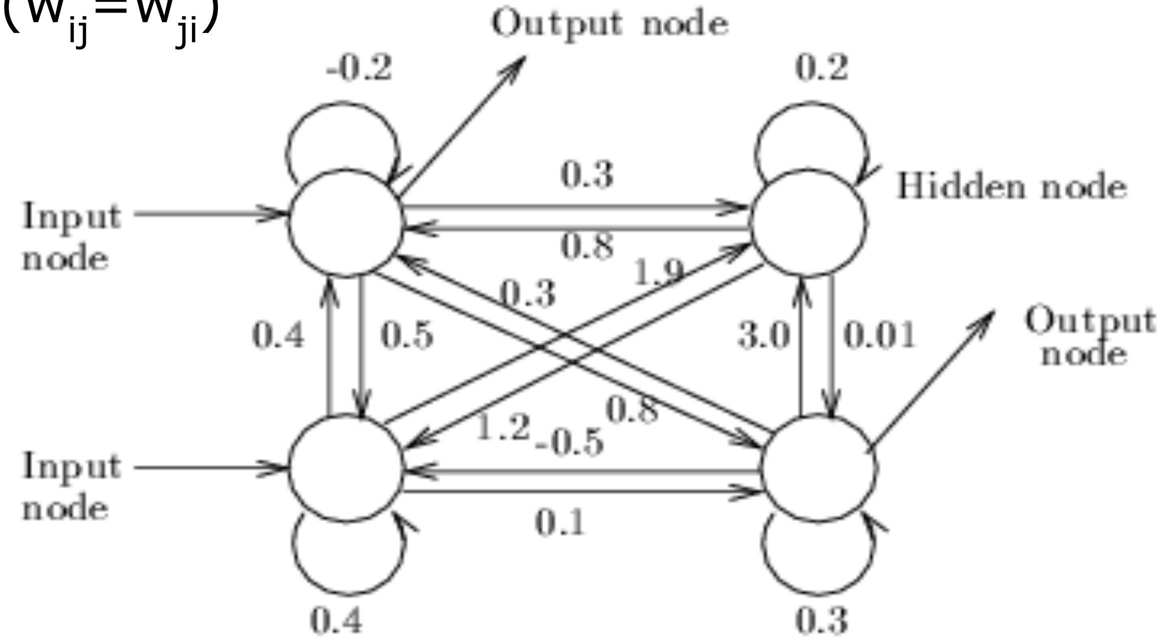
$$f(\text{net}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{\text{net}-\mu}{\sigma}\right)^2\right]$$



Gaussian function

Network Architecture (Asymmetric) Fully Connected Networks

- Every node is connected to every other node
- Connection may be excitatory (positive), inhibitory (negative), or irrelevant (0).
- Most general
- Symmetric fully connected nets: weights are symmetric ($w_{ij}=w_{ji}$)



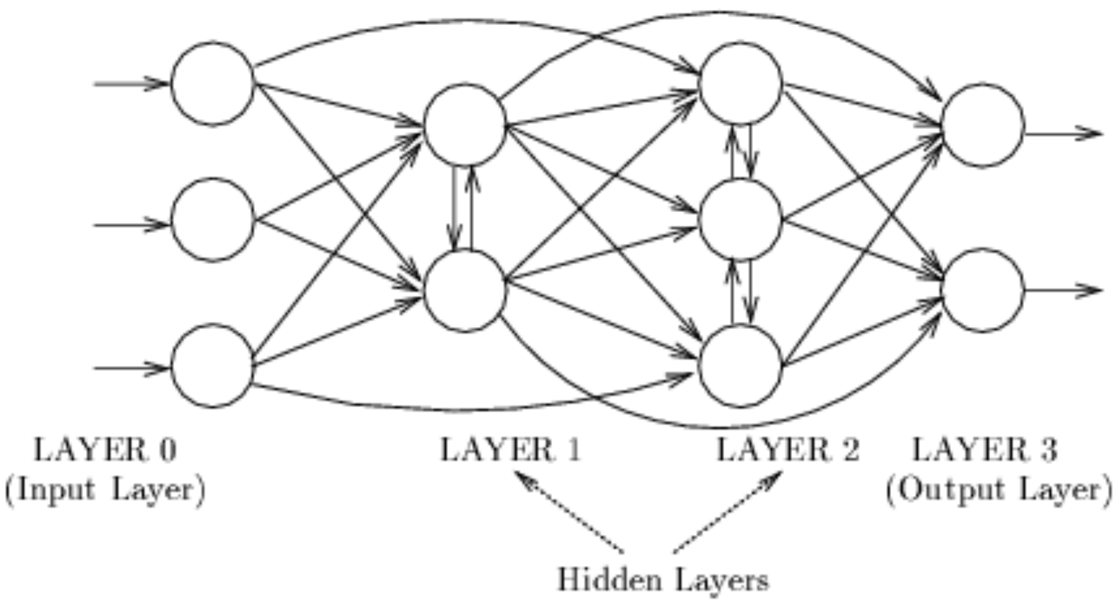
Input nodes: receive input from the environment

Output nodes: send signals to the environment

Hidden nodes: no direct interaction to the environment

Network Architecture
Layered Networks

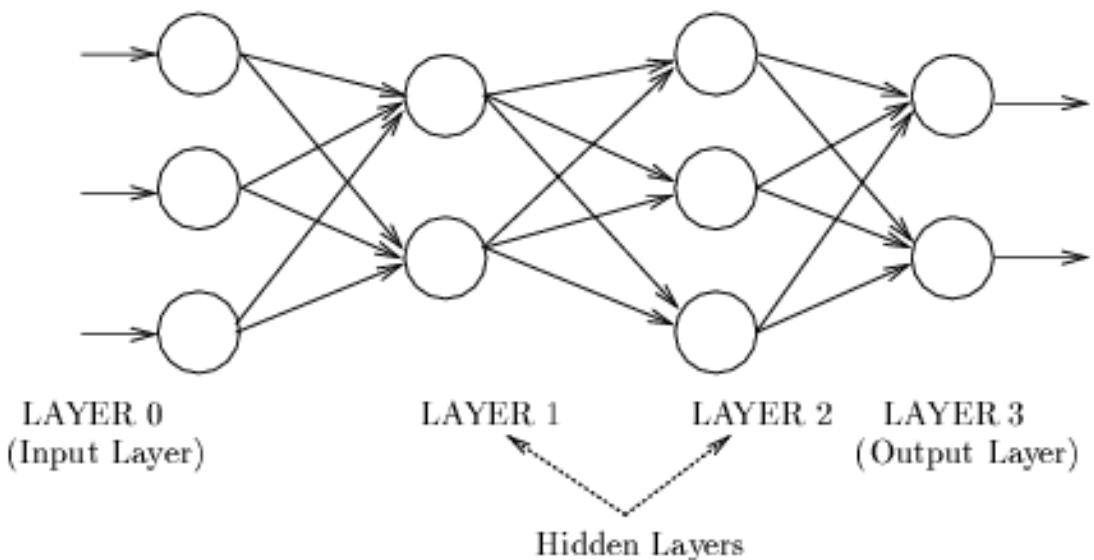
- Nodes are partitioned into subsets, called layers.
- No connections that lead from nodes in layer *j* to those in layer *k* if *j* > *k*.



- Inputs from the environment are applied to nodes in layer 0 (**input layer**).
- Nodes in input layer are placeholders with no computation occurring (i.e., their node functions are identity function)

Network Architecture Feedforward Networks

- A connection is allowed from a node in layer *i* only to nodes in layer *i* + 1.
- Most widely used architecture.



Conceptually, nodes at higher levels successively abstract features from preceding layers

What kind of problems can ANNs solve?

- Regression (function approximation)
- Classification

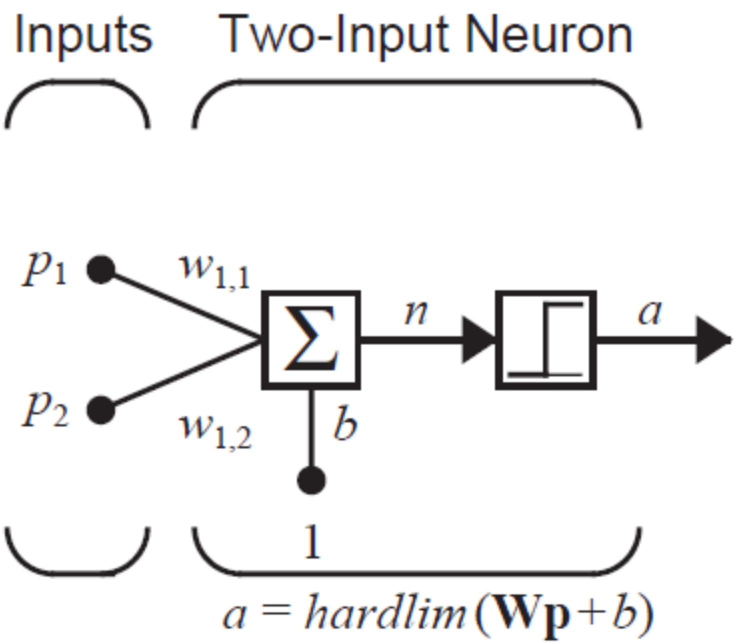
Training an ANN

- Perceptron’ algorithm
- Back-propagation
- Evolutionary
- ....

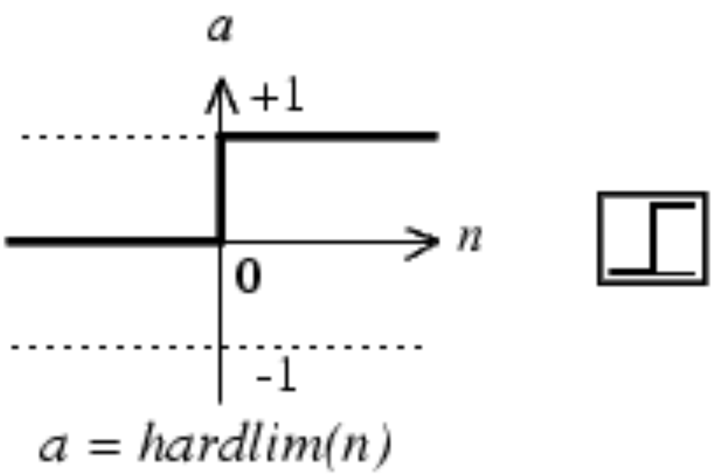
PERCEPTRON' ALGORITHM

- A linear classifier for classifying data X
- Parameters are adapted with an ad-hoc rule similar to stochastic steepest gradient descent.
- Can only perfectly classify a set of data for which different classes are linearly separable in the input space,
- It often fails completely for non-separable data.

EXAMPLE
TRAINING A SIMPLE PERCEPTRON FOR A CLASSIFICATION PROBLEM



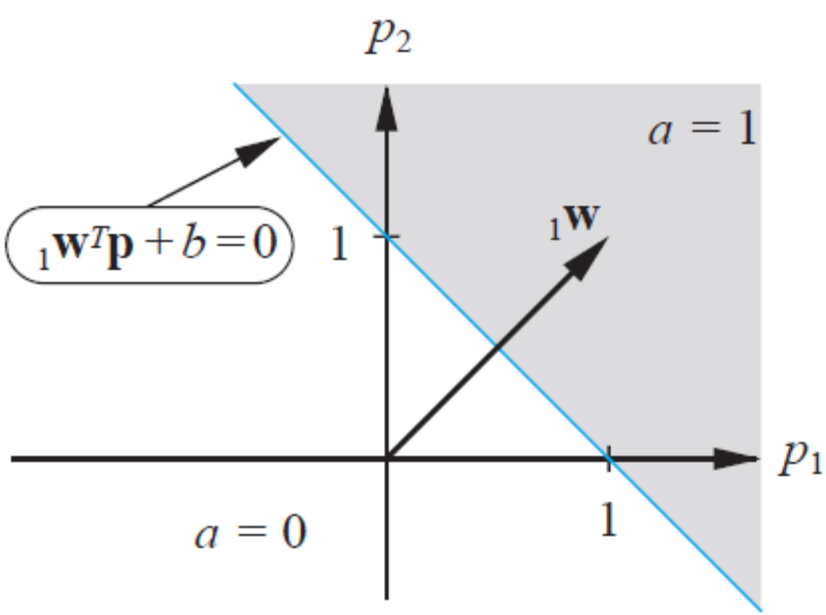
NODE FUNCTION



Hard-Limit Transfer Function

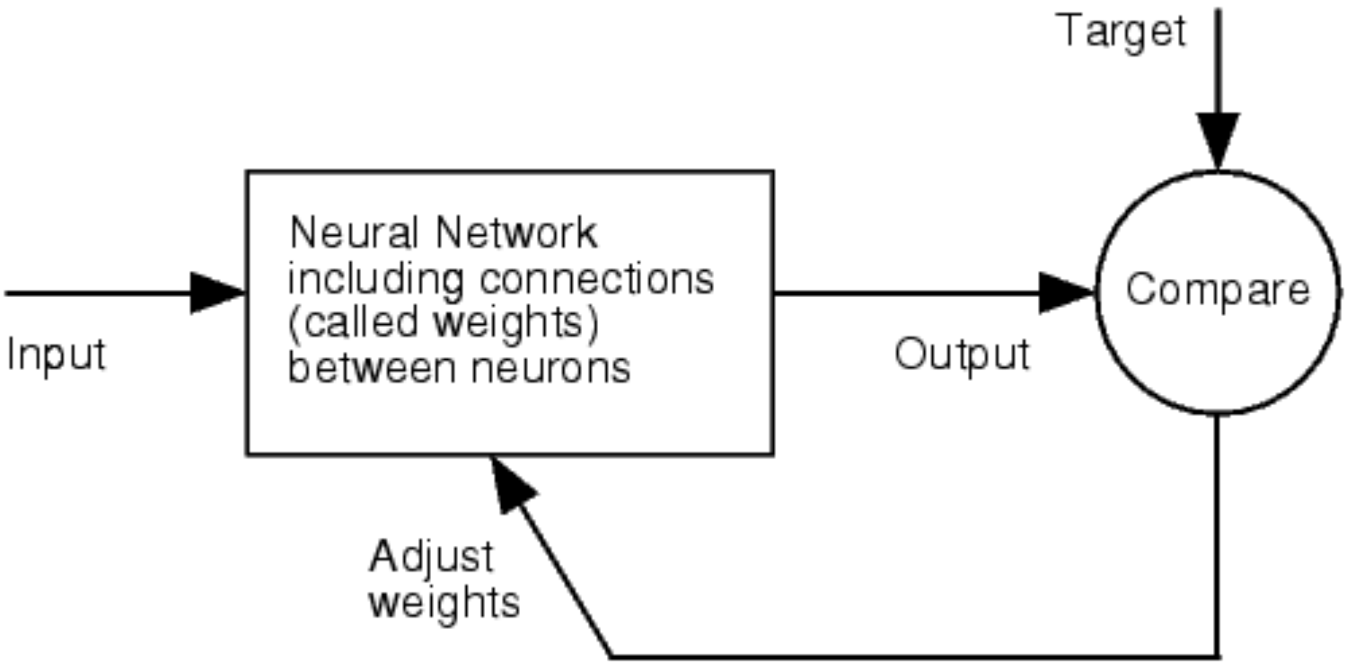
The perceptron produces:
=1 if the net input into the transfer function is equal to or greater than 0;
=otherwise it produces a 0.

Decision Boundary



$w_1 = 1$
$w_2 = 1$
$b = -1$

TRAINING ALGORITHM



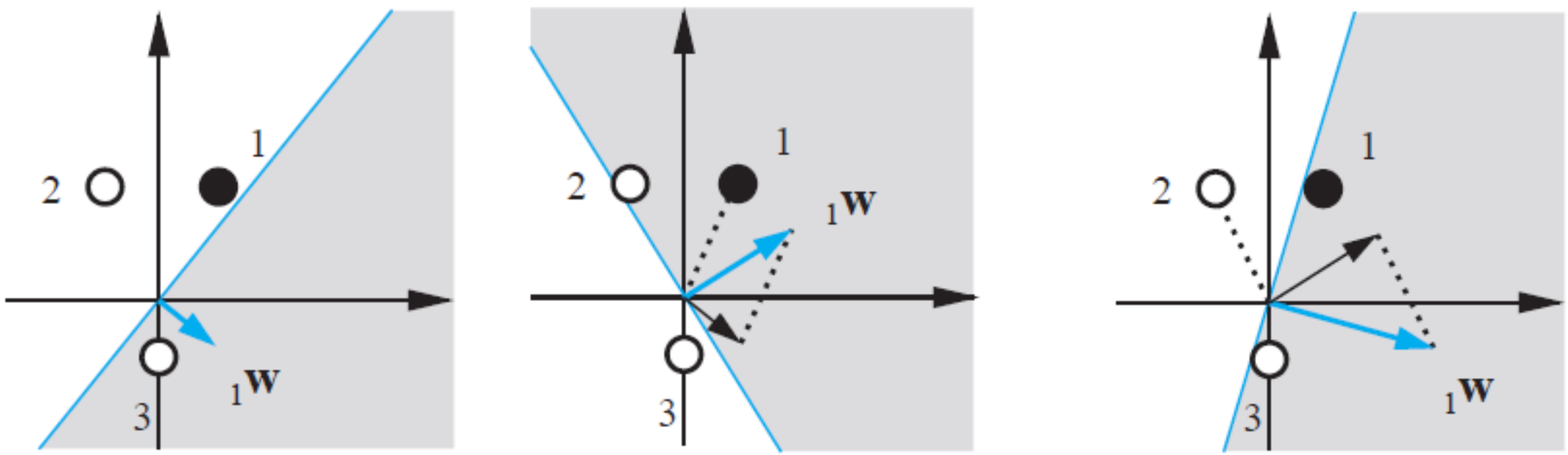
RULES FOR TRAINING (2)

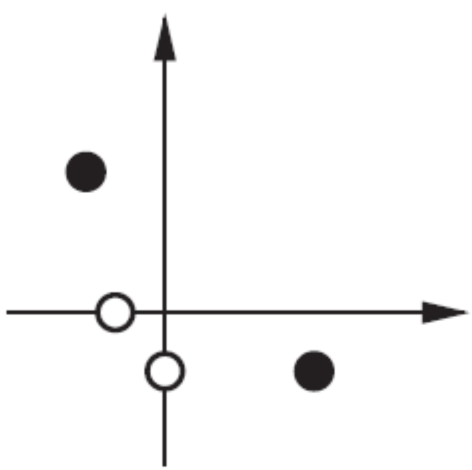
- If an input vector is presented and the output of the neuron is correct (**a** = **t**, and **e** = **t** - **a** = 0), then the weight vector **w** is not altered.
- If the neuron output is 0 and should have been 1 (**a** = 0 and **t** = 1, and **e** = **t** - **a** = 1), the input vector **p** is added to the weight vector **w**.

  This makes the weight vector point closer to the input vector, increasing the chance that the input vector will be classified as a 1 in the future.
- If the neuron output is 1 and should have been 0 (**a** = 1and **t** = 0, and **e** = **t** - **a** = -1), the input vector **p** is subtracted from the weight vector **w**.

  This makes the weight vector point farther away from the input vector, increasing the chance that the input vector is classified as a 0 in the future.

Rules for training





TRAINING DATA

$$\left\{\mathbf{p}_1 = \begin{bmatrix}2\\2\end{bmatrix}, t_1 = 0\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}1\\-2\end{bmatrix}, t_2 = 1\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}-2\\2\end{bmatrix}, t_3 = 0\right\} \quad \left\{\mathbf{p}_4 = \begin{bmatrix}-1\\1\end{bmatrix}, t_4 = 1\right\}$$

First step

$$\mathbf{W}(0) = \begin{bmatrix} 0 & 0 \end{bmatrix} \qquad b(0) = 0$$

$$a = hardlim(\mathbf{W}(0)\mathbf{p}_1 + b(0))$$

$$= hardlim\left(\begin{bmatrix} 0 & 0 \end{bmatrix}\begin{bmatrix} 2 \\ 2 \end{bmatrix} + 0\right) = hardlim(0) = 1$$

$$e = t_1 - a = 0 - 1 = -1$$

$$\Delta\mathbf{W} = e\mathbf{p}_1^T = (-1)\begin{bmatrix} 2 & 2 \end{bmatrix} = \begin{bmatrix} -2 & -2 \end{bmatrix}$$

$$\Delta b = e = (-1) = -1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T = \begin{bmatrix} 0 & 0 \end{bmatrix} + \begin{bmatrix} -2 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -2 \end{bmatrix} = \mathbf{W}(1)$$

$$b^{new} = b^{old} + e = 0 + (-1) = -1 = b(1)$$

SECOND STEP

$$a = hardlim(\mathbf{W}(1)\mathbf{p}_2 + b(1))$$

$$= hardlim\left(\begin{bmatrix}-2 & -2\end{bmatrix}\begin{bmatrix}1 \\ -2\end{bmatrix} - 1\right) = hardlim(1) = 1$$

Ok, no modification to the weights

The solution

$$\mathbf{W}(6) = \begin{bmatrix} -2 & -3 \end{bmatrix}$$

$$b(6) = 1$$

Back Propagation - Why

- Networks without hidden units are very limited in the input-output mappings they can model.
  - More layers of linear units do not help. Its still linear.
  - Fixed output non-linearities are not enough
- We need multiple layers of adaptive non-linear hidden units. This gives us a universal approximator. But how can we train such nets?
  - We need an efficient way of adapting all the weights, not just the last layer. This is hard. Learning the weights going into hidden units is equivalent to learning features.
  - Nobody is telling us directly what hidden units should do.

BACK PROPAGATION ALGORITHM

Initialize all weights to small random numbers.

Repeat

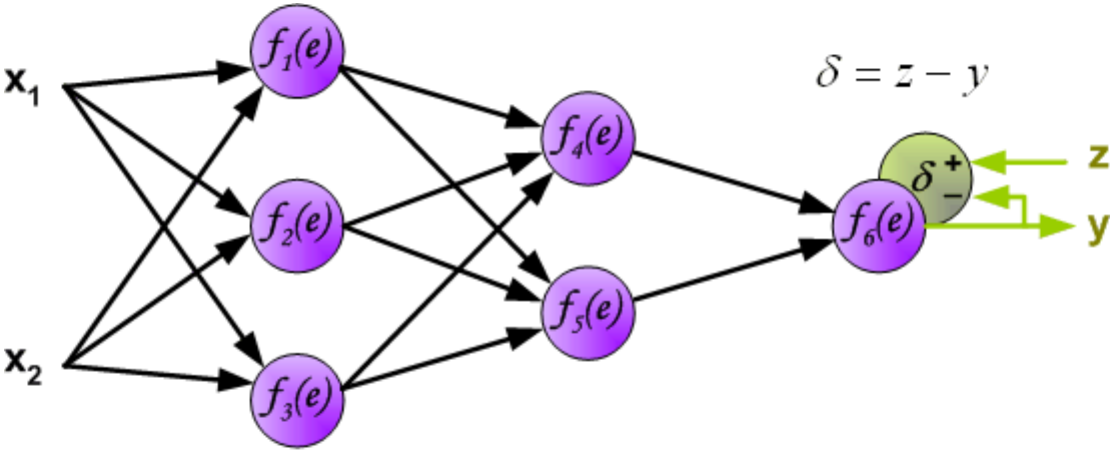
For each example X do

- Propagate example X forward through the network
- Propagate errors backward through the network

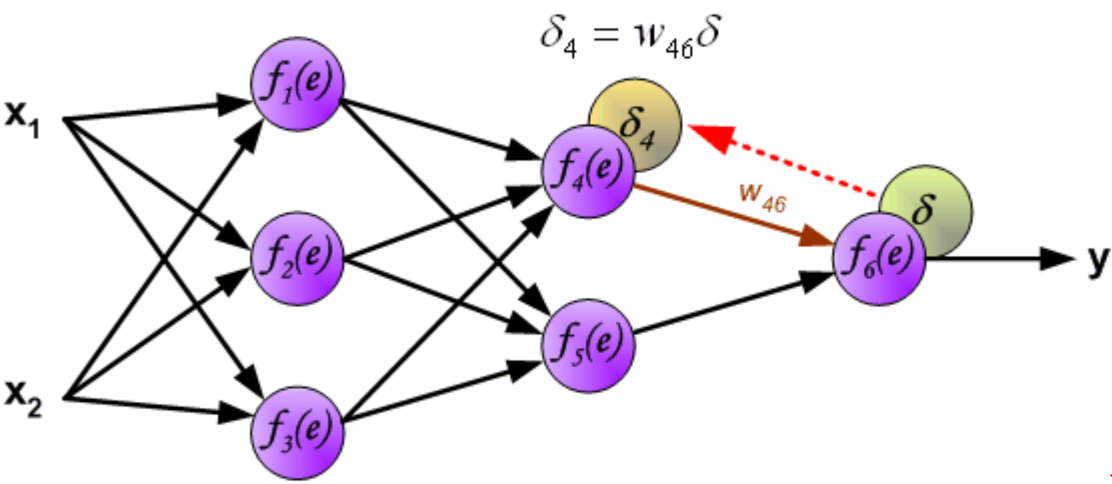
Until error is small

The idea behind backpropagation

- We don’t know what the hidden units ought to do, but we can compute how fast the error changes as we change a hidden activity.
  - Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities.
  - Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.
  - We can compute error derivatives for all the hidden units efficiently.
  - Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.



$\delta = z - y$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
δ
+
−
z
y



$\delta_4 = w_{46}\delta$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
δ_4
w_{46}
δ
y

Advantages and disadvantages of BP

- Reduced running time – smaller than evolutionary training (Strength)
- Cannot train activation functions that have no derivate – (Weak)
- Can easily fall into local optima– (Weak)
- Cannot discover new connections between neurons – (Weak)

Evolutionary Training

- A solution is a set of values for the weight of each connection between two neurons.
- Start with a set of random weights.
- Compute the quality of that NN.
- Repeat
  - Mutation / Crossover / Fitness
- Until some good values have been obtained

ADVANTAGES AND DISADVANTAGES

- Big running time – bigger than back-propagation (Weak)
- Can train any kind of activation function (including those that have no derivate) – (Strength)
- Can escape of local optima better than backpropagation – (Strength)
- Can discover new connections between neurons – (Strength)

Differences between ANNs and GP

GP discover the entire structure	ANNs discover only the weight.
Connection between nodes have weight 1	Connections between nodes have a weight
Within a node we can have any function	Within a node we have a predefined node function
A node always fires a value	A neuron does not always fires values.

Solving problems with ANNs

- Regression
- Classification

- (training similar with GP)
  - Some fitness cases are needed

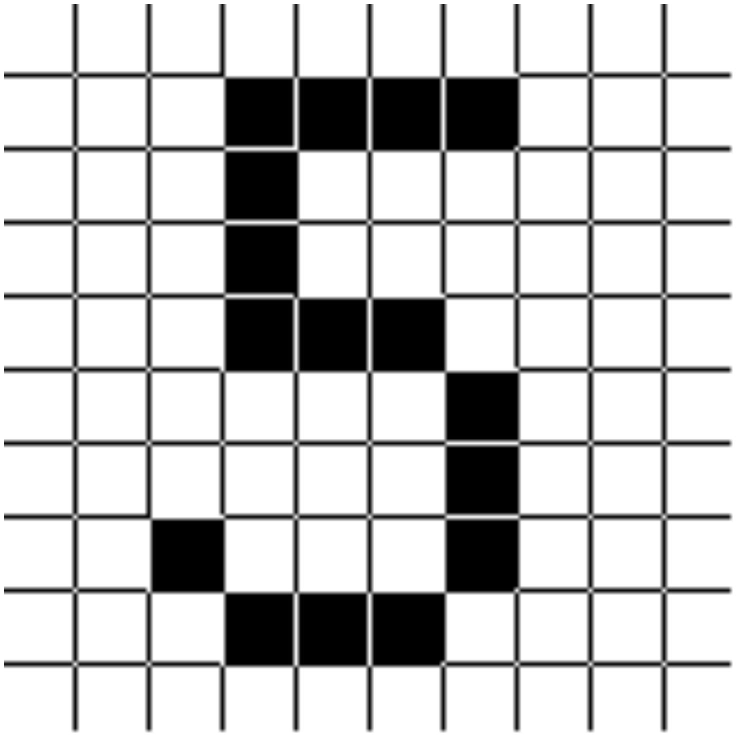
ANNs FOR DIGITS RECOGNITION

- Each digit is represented over a matrix of pixels (no more than 8x8)

- ANN with 64 inputs, 1 output and multiple hidden layers.

- Output between
  - [a1, a2) -> 0
  - [a2, a3) -> 1
  - …
  - [a10, a11) -> 9

- 10 training data
  - (00010101, 0)
  - (10101010, 1
  - …
  - (10101011, 9)



REFERENCES

- Martin T. Hagan, Howard B. Demuth, Mark H. Beale, Neural Network Design, PWS Publishing, 1996