Advanced Introduction to Machine Learning, CMU-10715

Perceptron, Multilayer Perceptron

Barnabás Póczos, Sept 17

Contents

- History of Artificial Neural Networks
- Definitions: Perceptron, MLP
- Representation questions
- Perceptron algorithm
- Backpropagation algorithm

Short History

- **Progression (1943-1960)**
  - First mathematical model of neurons
    - Pitts & McCulloch (1943)
  - Beginning of artificial neural networks
  - Perceptron, Rosenblatt (1958)
    - A single layer neuron for classification
    - Perceptron learning rule
    - Perceptron convergence theorem

- **Degression (1960-1980)**
  - Perceptron can’t even learn the XOR function
  - We don’t know how to train MLP
  - 1969 Backpropagation… but not much attention…

Short History

- **Progression (1980-)**
  - 1986 Backpropagation reinvented:
    - Rumelhart, Hinton, Williams:
      Learning representations by back-propagating errors. Nature, 323, 533—536, 1986
  - Successful applications:
    - Character recognition, autonomous cars,...
  - **Open questions**: Overfitting? Network structure? Neuron number? Layer number? Bad local minimum points? When to stop training?
  - Hopfield nets (1982), Boltzmann machines,...

Short History

- **Degression (1993-)**

  - SVM: Vapnik and his co-workers developed the Support Vector Machine (1993). It is a shallow architecture.
  - SVM almost kills the ANN research.
  - Training deeper networks consistently yields poor results.
  - Exception: deep convolutional neural networks, Yann LeCun 1998. (discriminative model)

Short History

Progression (2006-)

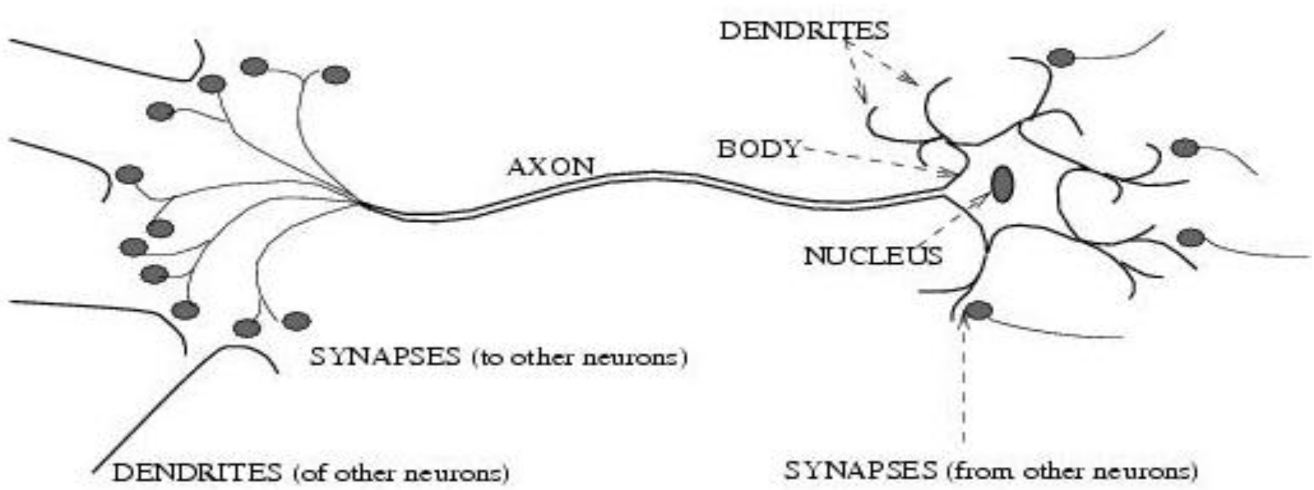
Deep Belief Networks (DBN)

- Hinton, G. E, Osindero, S., and Teh, Y. W. (2006).
  A fast learning algorithm for deep belief nets.
  Neural Computation, 18:1527-1554.
- Generative graphical model
- Based on restrictive Boltzmann machines
- Can be trained efficiently

Deep Autoencoder based networks

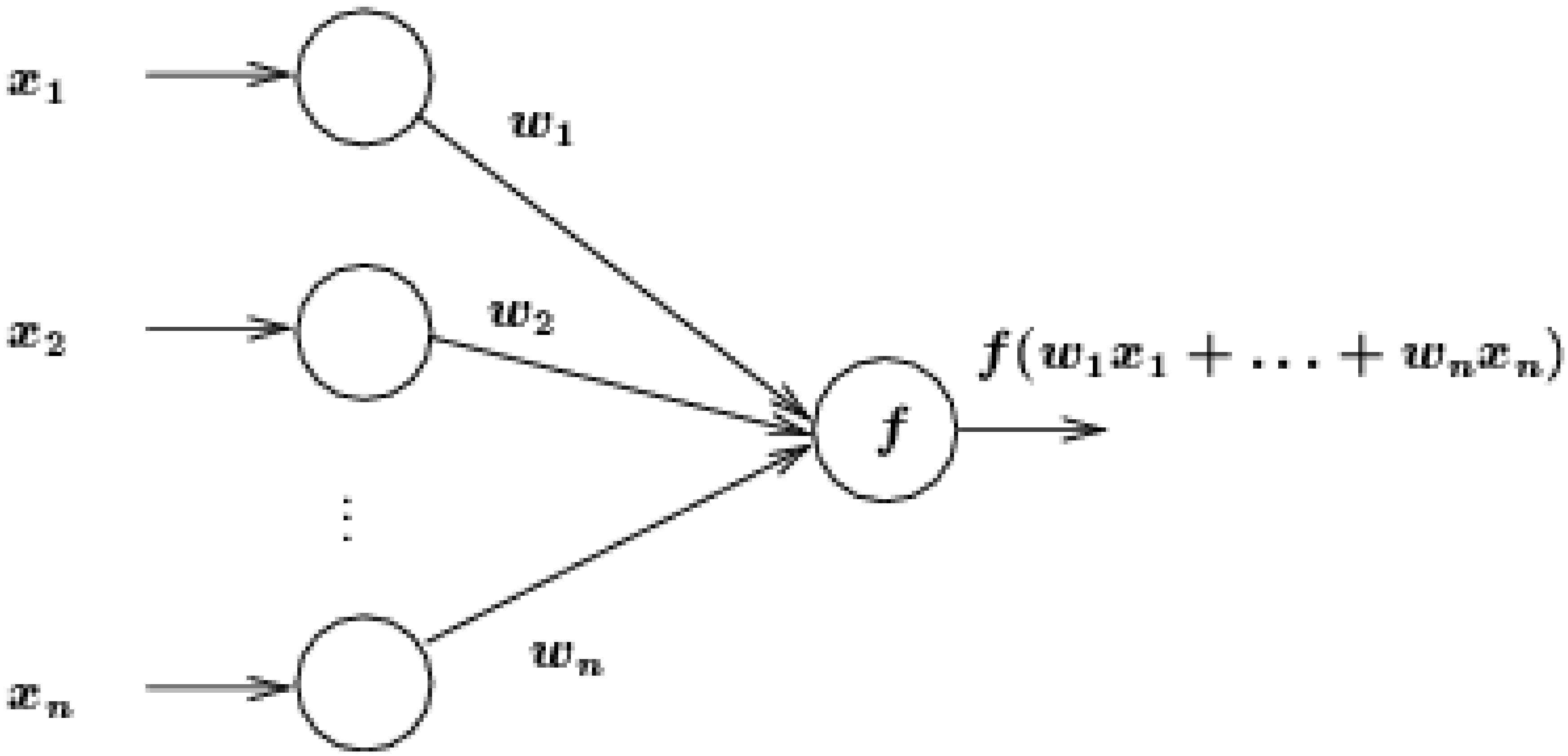
Bengio, Y., Lamblin, P., Popovici, P., Larochelle, H. (2007).
Greedy Layer-Wise Training of Deep Networks,
Advances in Neural Information Processing Systems 19

The Neuron



– Each neuron has a body, axon, and many dendrites
– A neuron can fire or rest
– If the sum of weighted inputs larger than a threshold, then the neuron fires.
– Synapses: The gap between the axon and other neuron's dendrites. It determines the weights in the sum.

The Mathematical Model of a Neuron



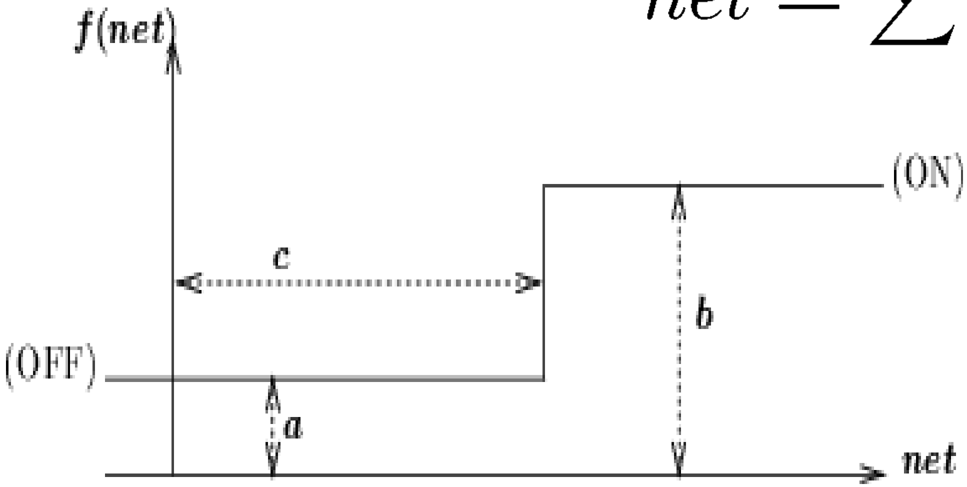
$$y = f(w_1x_1 + \ldots + w_nx_n)$$

Typical activation functions

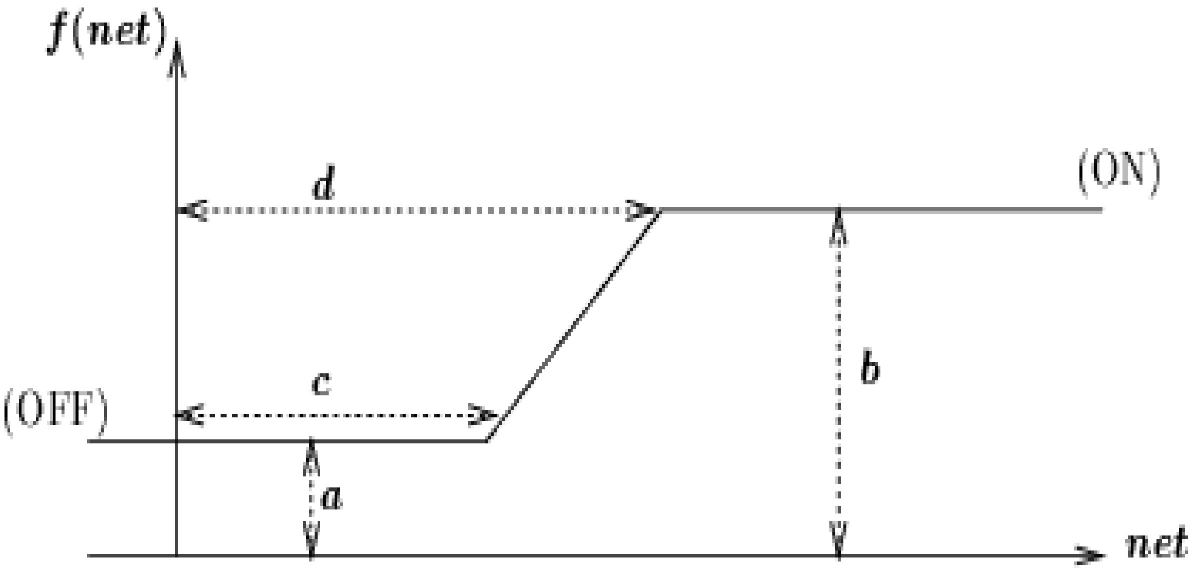
- Identity function

$$net = \sum w_i x_i$$

- Threshold function (perceptron)



- Ramp function



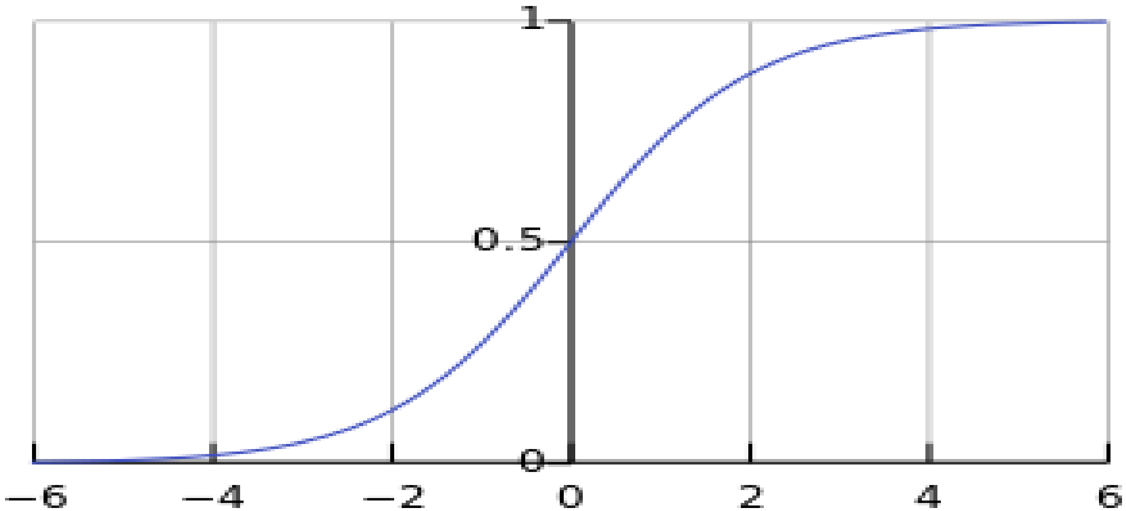
Typical activation functions

- Logistic function

$$y(x) = (1 + e^{-x})^{-1}$$

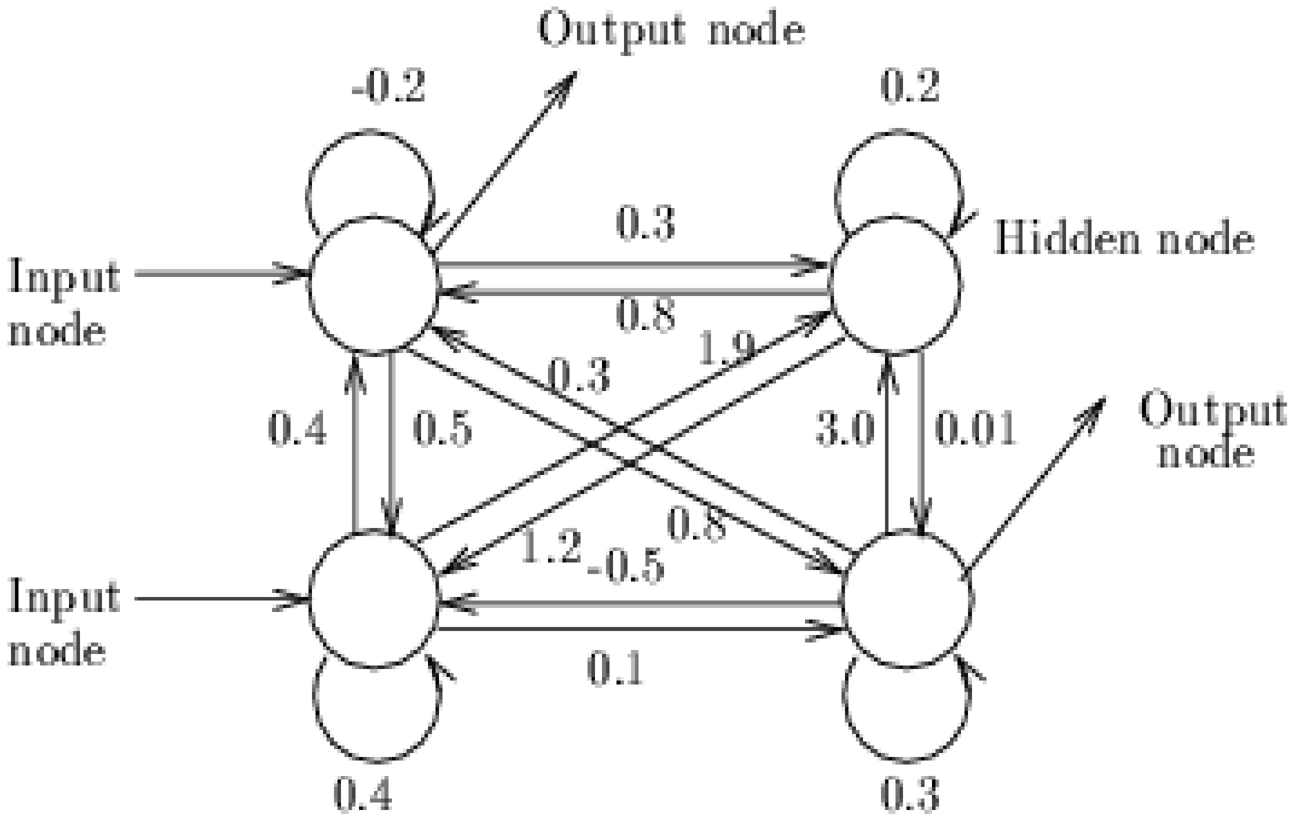
- Hiperbolic tangent function

$$y(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



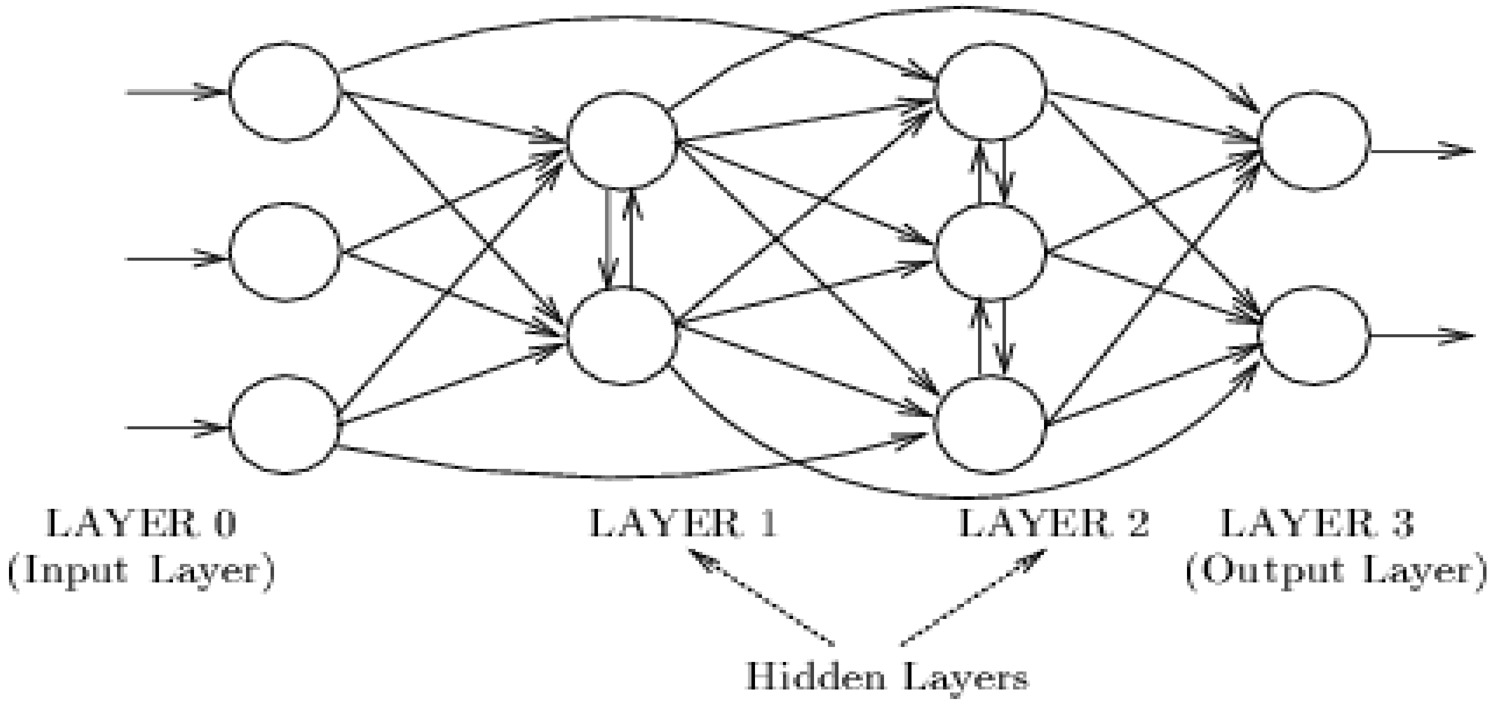
Structure of Neural Networks

Fully Connected Neural Network



Input neurons, Hidden neurons, Output neurons

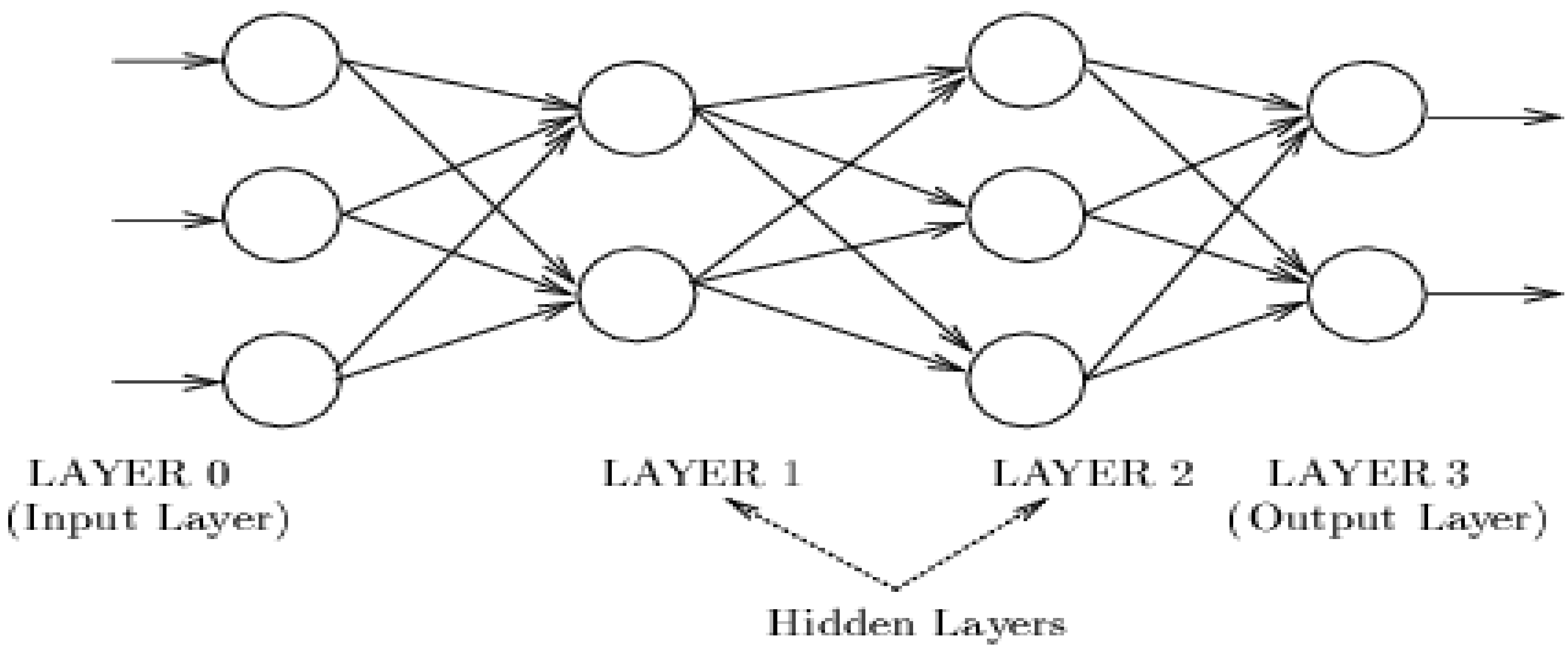
Layers



Convention: The input layer is Layer 0.

Feedforward neural networks

- Connections only between Layer i and Layer i+1
- **= Multilayer perceptron**
- The most popular architecture.

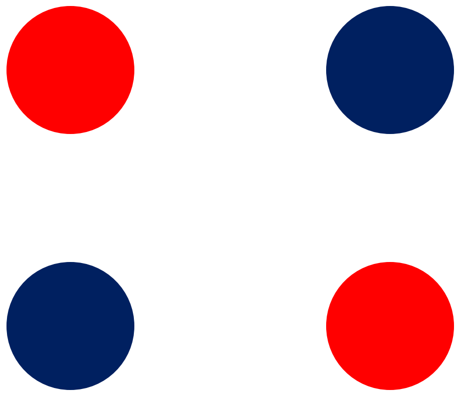


Recurrent NN: there are connections backwards too.

What functions can multi-layer perceptrons represent?

Perceptrons cannot represent the XOR function

f(0,0)=1, f(1,1)=1, f(0,1)=0, f(1,0)=0



$$f(x_1, x_2) = sgn(w_1 x_1 + w_2 x_2 + w_0). \quad w_0, w_1, w_2 =?.$$

What functions can **multilayer** perceptrons represent?

Hilbert's 13th Problem

"Solve 7-th degree equation using continuous functions of two parameters."

Related conjecture:

Let f be a function of 3 arguments such that

$$f(a,b,c) = x, \text{ where } x^7 + ax^3 + bx^2 + cx + 1 = 0.$$

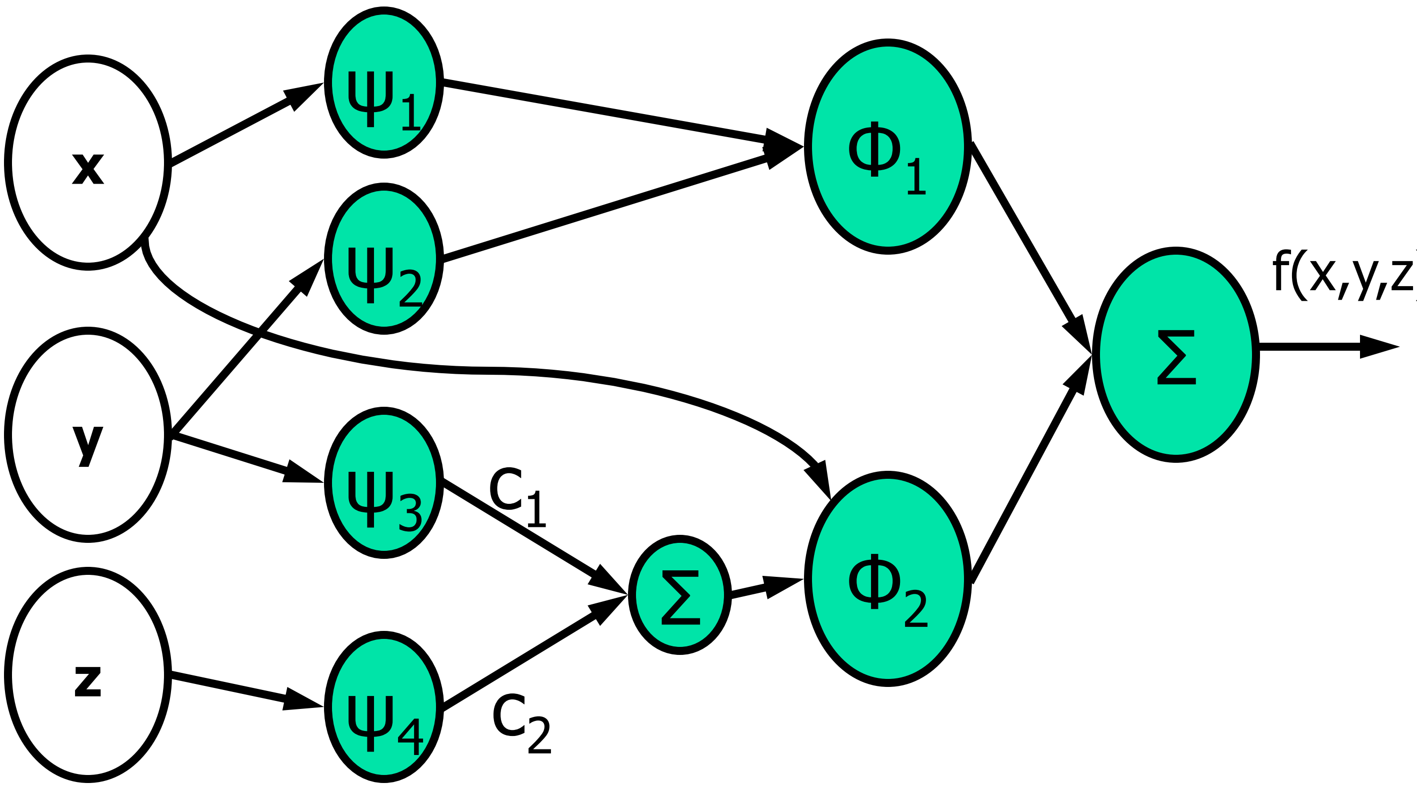
Prove that f cannot be rewritten as a composition of finitely many functions of two arguments.

Another rewritten form:

Prove that there is a nonlinear continuous system of three variables that cannot be decomposed with finitely many functions of two variables.

Function decompositions

$$f(x,y,z)=\Phi_1(\psi_1(x),\ \psi_2(y))+\Phi_2(c_1\psi_3(y)+c_2\psi_4(z),x)$$



Function decompositions

1957, Arnold disproves Hilbert's conjecture.

Let $f : [0,1]^N \to \mathbb{R}$ be an arbitrary continuous function.

Then there exisist $N(2N+1)$ functions ψ_{pq}, s.t.

$\psi_{pq} : [0,1] \to \mathbb{R}$, $p = 1, 2 \ldots N$, $q = 0, 1, \ldots 2N$,

$\star$ they are monotone increasing

$\star$ don't depend on f (only on N)

and there exisist $2N+1$ functions ϕ_q^f:

$\phi_q^f : \mathbb{R} \to \mathbb{R}$, $q = 0, 1, 2 \ldots 2N$, they can depend on f, s.t.

$$f(x_1, \ldots, x_N) = \sum_{q=0}^{2N} \phi_q^f \left(\sum_{p=1}^{N} \psi_{pq}(x_p) \right)$$

Function decompositions

Corollary:

Any $f : [0,1]^N \to \mathbb{R}$ function can be represented exactly with an MLP of two hidden layers.

$$f(x_1, \ldots, x_N) = \sum_{q=0}^{2N} \phi_q^f \left(\sum_{p=1}^{N} \psi_{pq}(x_p) \right)$$

Issues: This statement is not constructive.

For a given *N* we don't know ψ_{pq}

and for a given *N* and *f*, we don't know ϕ_q^f

Universal Approximators

Kur Hornik, Maxwell Stinchcombe and Halber White: "Multilayer feedforward networks are universal approximators", Neural Networks, Vol:2(3), 359-366, 1989

Definition: $\Sigma^N(g)$ neural network with 1 hidden layer:

$$\Sigma^N(g) = \left\{ f : \Re^N \to \Re \,\middle|\, f(x_1, \ldots, x_N) = \sum_{i=1}^{M} c_i g(a_i^T x + b_i) \right\}$$

Theorem:

If $\delta > 0$, g arbitrary sigmoid function, f is continuous on a compact set A, then

$$\exists \hat{f} \in \Sigma^N(g), \text{such that} \left\| f(x) - \hat{f}(x) \right\| < \delta \; \forall x \in A \text{ esetén}$$

Universal Approximators

Theorem: (Blum & Li, 1991)

SgnNet²(x,w) with two hidden layers and sgn activation function is uniformly dense in L².

Definition:

$$\operatorname{sgn} Net^{(2)}(\bar{x},\bar{w}) = \sum_i w_i^{(3)} \operatorname{sgn}\left\{\sum_j w_{ij}^{(2)} \operatorname{sgn}\left\{\sum_l w_{jl}^{(1)} x_l\right\}\right\}$$

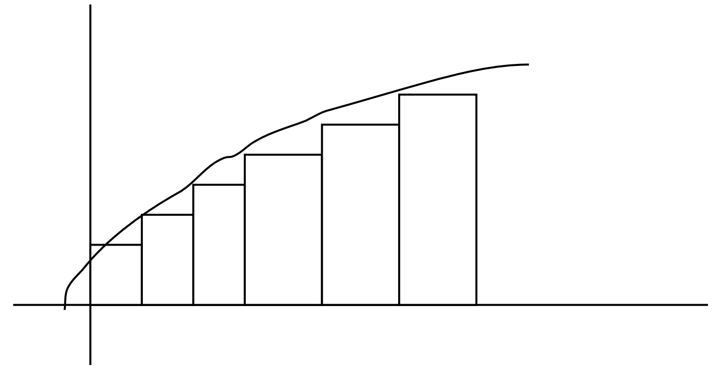
Formal statement:

If $f(\bar{x}) \in L^2$, i.e. $\sqrt{\int_X f^2(\bar{x})dx} < \infty$, and $\varepsilon > 0$, then $\exists \bar{w}$, such that

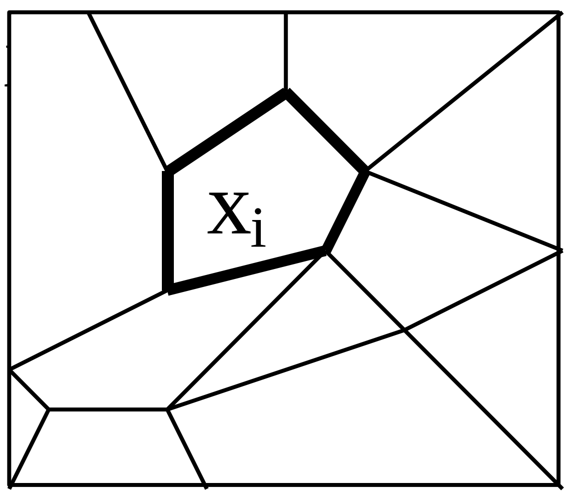
$$\int \ldots \int \left(f(\bar{x}) - \sum_i w_i^{(3)} \operatorname{sgn}\left\{\sum_j w_{ij}^{(2)} \operatorname{sgn}\left\{\sum_l w_{jl}^{(1)} x_l\right\}\right\}\right)^2 dx_1 \ldots dx_n < \varepsilon$$

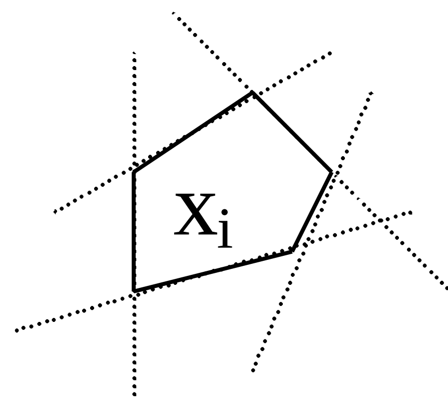
Proof

Integral approximation in 1-dim:



Integral approximation in 2-dim:

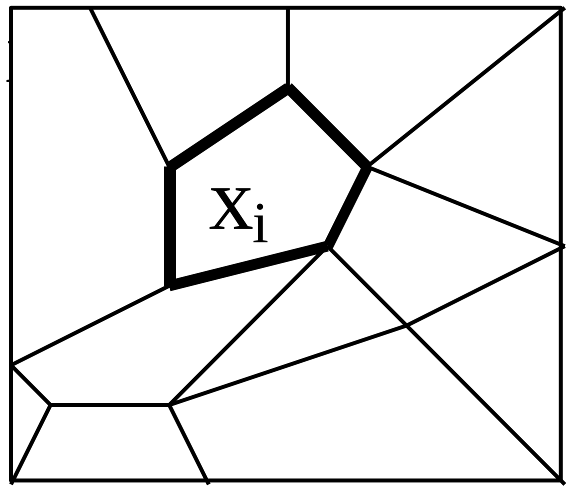


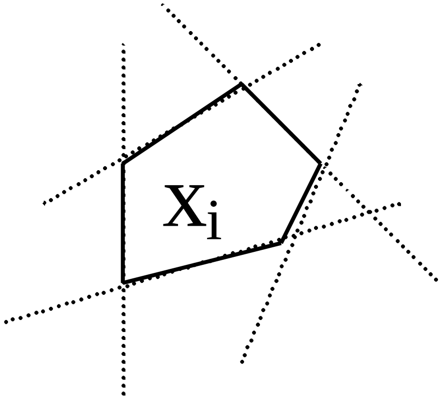


$$\bigcup_i X_i = X \qquad X_i \bigcap X_j = \varnothing$$

$$\int \ldots \int \left| f(x) - \sum_i \alpha_i I_{X_i}(x) \right|^2 d(x) \; < \varepsilon$$

Proof





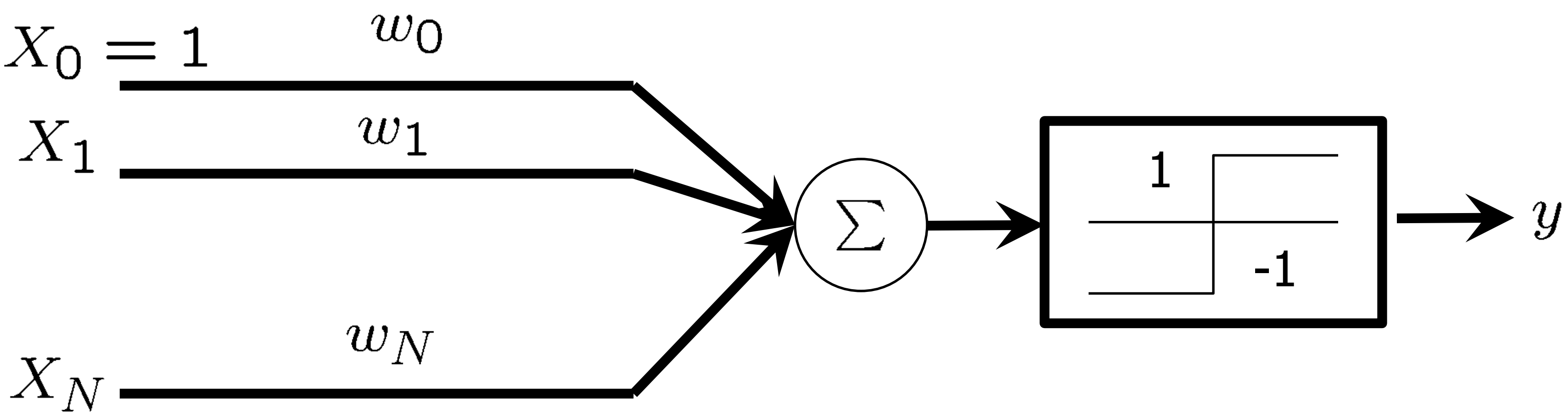
The indicator function of X_i polygon can be learned by this neural network:

$$\operatorname{sgn}\left\{\sum_i a_i \operatorname{sgn}\left\{\sum_j b_{ij} x_j\right\}\right\} = \begin{cases} 1 \text{ if x is in } X_i \\ -1 \text{ otherwise} \end{cases}$$

The weighted linear combination of these indicator functions will be a good approximation of the original function *f*

LEARNING
The Perceptron Algorithm

The Perceptron



$$y = sgn(\mathbf{w}^T\mathbf{x})$$

The training set

Let the training set be

$$X^1 = \{\mathbf{x}_k | \mathbf{x}_k \in \text{Class I}\}$$

$$X^2 = \{\mathbf{x}_k | \mathbf{x}_k \in \text{Class II}\}$$

Assume that the classes are linearly separable.

Let $\mathbf{w}^*$ be the normal vector of the separating hyperplane.

$$\mathbf{w}^{*T}\mathbf{x} > 0 \text{ if } \mathbf{x} \in X^1$$

$$\mathbf{w}^{*T}\mathbf{x} < 0 \text{ if } \mathbf{x} \in X^2$$

The perceptron algorithm

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \mu(y(k) - \widehat{y}(k))\mathbf{x}(k) \tag{1}$$

$$\boxed{\mathbf{w}(k) = \mathbf{w}(k-1) + \mu\varepsilon(k)\mathbf{x}(k)} \tag{2}$$

- This is an LMS algoritm. We change $\mathbf{w}(k-1)$ with $\pm\mathbf{x}(k)$
- $\mu > 0$ learning rate. It doesn't need to go to zero!
- If $y(k), \widehat{y}(k) \in \{-1, 1\} \Rightarrow \varepsilon(k) \in \{0, 2, -2\}$
- If $y(k), \widehat{y}(k) \in \{0, 1\} \Rightarrow \varepsilon(k) \in \{0, 1, -1\}$

The perceptron algorithm

1., If $k = 1$, let $\mathbf{w}(0)$ be arbitrary.

2., Let $\mathbf{x}(k) \in X^1 \bigcup X^2$ be a training point misclassified by $\mathbf{w}(k-1)$

3., If there is no such vector $\Rightarrow$ 5.

4., If $\exists$ a misclassified vector $\Rightarrow \begin{cases} \alpha(k) = \mu(y(k) - \widehat{y}(k)) \\ \mathbf{w}(k) = \mathbf{w}(k-1) + \alpha(k)\mathbf{x}(k) \\ k = k+1 \\ \text{Back to 2-re} \end{cases}$

5., END

Perceptron convergence theorem

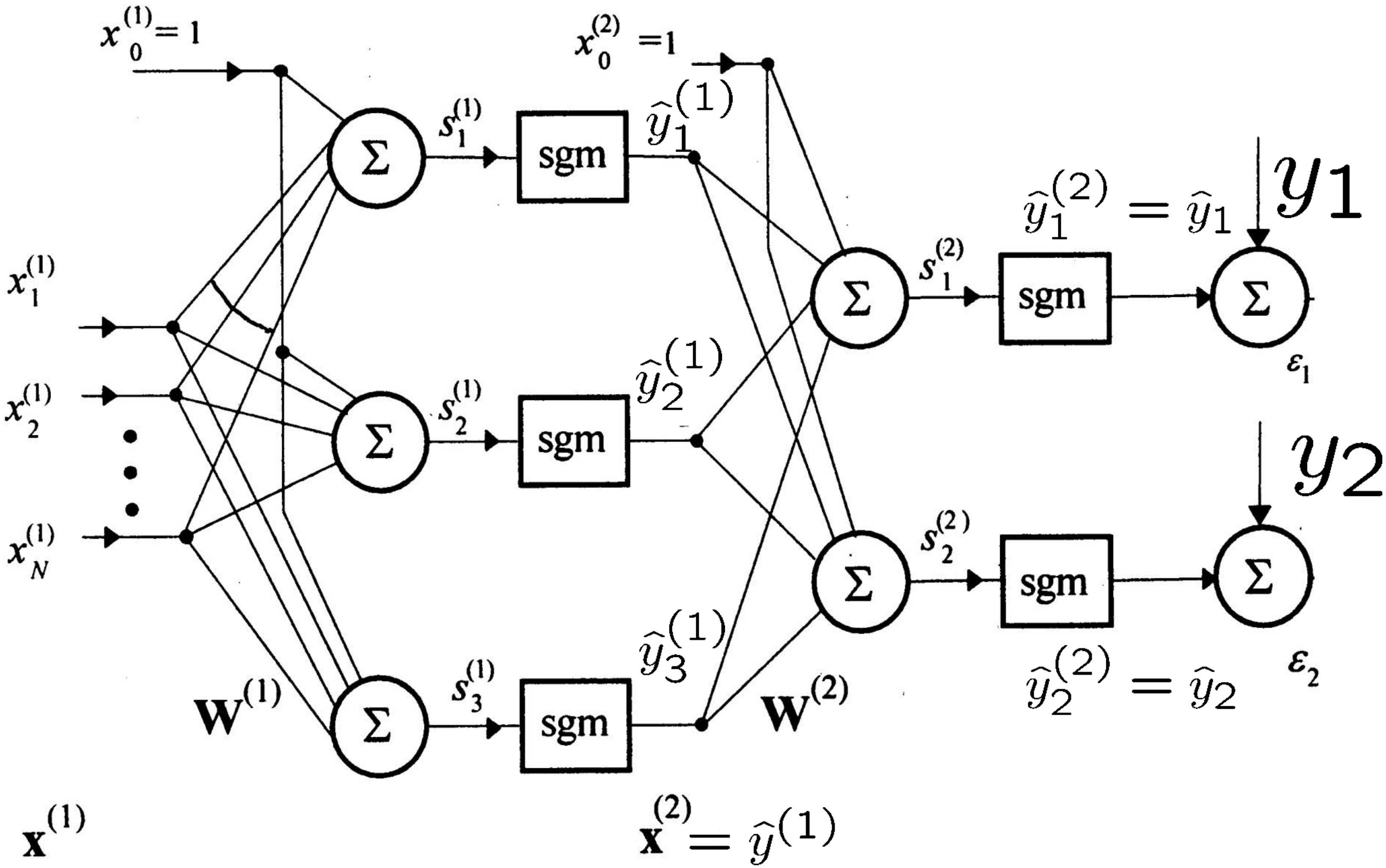
Theorem:

If the training samples were linearly separable, then the algorithm finds a separating hyperplane in finite steps.

The upper bound on the number of steps is "independent" from the number of training samples, but depends on the dimension.

Proof: [homework**]**

Multilayer Perceptron



The gradient of the error

The current error:

$$\varepsilon^2 = \varepsilon_1^2 + \varepsilon_2^2 = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \tag{1}$$

More generally:

$$\varepsilon^2 = \sum_{p=1}^{N_L} \varepsilon_p^2 = \sum_{p=1}^{N_L} (\hat{y}_p - y_p)^2 \tag{2}$$

We want to calculate

$$\frac{\partial \varepsilon(k)^2}{\partial W_{ij}^l(k)} = ?$$

Notation

- $W_{ij}^{l}(k)$: At time step k, the stength of connection from neuron j on layer $l-1$ to neuron i on layer l.
  $(i = 1 \ldots N_l,\ j = 1 \ldots N_{l-1})$

- $s_i^l(k)$: The summed input of neuron i on layer l before function f at time step k $(i = 1 \ldots N_l)$.

- $\mathbf{x}^l(k) \in \mathbb{R}^{N_{l-1}}$: The input of layer l at time step k

- $\hat{\mathbf{y}}^l(k) \in \mathbb{R}^{N_l}$: The output of layer l at time step k

- $N_1, N_2, \ldots, N_l, \ldots N_L$: Number of nurons in layers $1, 2, \ldots, l, \ldots, L$

Some observations

$$\mathbf{x}^l = \hat{\mathbf{y}}^{l-1} \in \mathbb{R}^{N_{l-1}} \tag{1}$$

$$s_i^l = \mathbf{W}_{i\cdot}^l \hat{\mathbf{y}}^{l-1} = \sum_{j=1}^{N_{l-1}} W_{ij}^l \mathbf{x}_j^l = \sum_{j=1}^{N_{l-1}} W_{ij}^l \underbrace{f(s_j^{l-1})}_{\hat{y}_j^{l-1}} \tag{2}$$

$$s_j^{l+1} = \sum_{i=1}^{N_l} W_{ji}^{l+1} f(s_i^l) \tag{3}$$

The backpropagated error

Introduce the notation

$$\boxed{\delta_i^l(k) = \frac{-\partial \varepsilon^2(k)}{\partial s_i^l(k)} = -\sum_{p=1}^{N_L} \frac{\partial \varepsilon_p^2(k)}{\partial s_i^l(k)}} \qquad (1)$$

where $i = 1..N_l$

As a special case, we have that

$$\boxed{\delta_i^L(k) = -\sum_{p=1}^{N_L} \frac{-\partial (y_p(k) - f(s_p^L(k)))^2}{\partial s_i^L(k)} = 2\varepsilon_i(k) f'(s_i^L(k))} \qquad (2)$$

The backpropagated error

Lemma

$\delta_i^l(k)$ can be calculated from $\{\delta_1^{l+1}(k), \ldots, \delta_{N_{l+1}}^{l+1}(k)\}$ using Backward recursion.

$$\delta_i^l(k) = -\sum_{p=1}^{N_L} \frac{\partial \varepsilon_p^2}{\partial s_i^l} = \sum_{p=1}^{N_L} \sum_{j=1}^{N_{l+1}} -\frac{\partial \varepsilon_p^2}{\partial s_j^{l+1}} \underbrace{\frac{\partial s_j^{l+1}}{\partial s_i^l}}_{W_{ji}^{l+1} f'(s_i^l)} \quad (1)$$

$$= \sum_{j=1}^{N_{l+1}} \underbrace{\sum_{p=1}^{N_L} -\frac{\partial \varepsilon_p^2}{\partial s_j^{l+1}}}_{\delta_j^{l+1}} W_{ji}^{l+1} f'(s_i^l) \quad (2)$$

The backpropagated error

Therefore,

$$\delta_i^l(k) = \left(\sum_{j=1}^{N_{l+1}} \delta_j^{l+1}(k) W_{ji}^{l+1}(k) \right) f'(s_i^l(k))$$

where $\delta_i^l(k)$ is the backpropagated error.

Now using that

$$s_i^l(k) = \sum_{j=1}^{N_{l-1}} W_{ij}^l(k) x_j^l(k) \qquad (1)$$

The backpropagation algorithm

$$\frac{\partial \varepsilon(k)^2}{\partial W_{ij}^l(k)} = \underbrace{\frac{\partial \varepsilon(k)^2}{\partial s_i^l(k)}}_{-\delta_i^l(k)} \underbrace{\frac{\partial s_i^l(k)}{\partial W_{ij}^l(k)}}_{x_j^l(k)} = -\delta_i^l(k) x_j^l(k) \tag{1}$$

The Backpropagation algorithm:

$$\boxed{W_{ij}^l(k+1) = W_{ij}^l(k) + \mu \delta_i^l(k) x_j^l(k)} \tag{2}$$

In vector form:

$$\boxed{\mathbf{W}_{i\cdot}^l(k+1) = \mathbf{W}_{i\cdot}^l(k) + \mu \delta_i^l(k) \mathbf{x}_{\cdot}^l(k)} \tag{3}$$