Neural Networks

(P-ITEEA-0011)

Multilayer Perceptron
Back-propagation algorithm

Akos Zarandy
Lecture 3
September 23, 2018

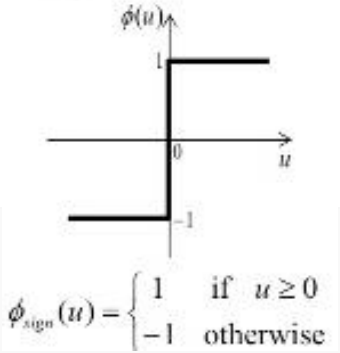
Contents

- Recall
  - Single-layer perceptron and its learning method
- Multilayer perceptron
  - Topology
  - Operation
- Representation
- Blum and Li construction
- Learning
  - Back-propagation

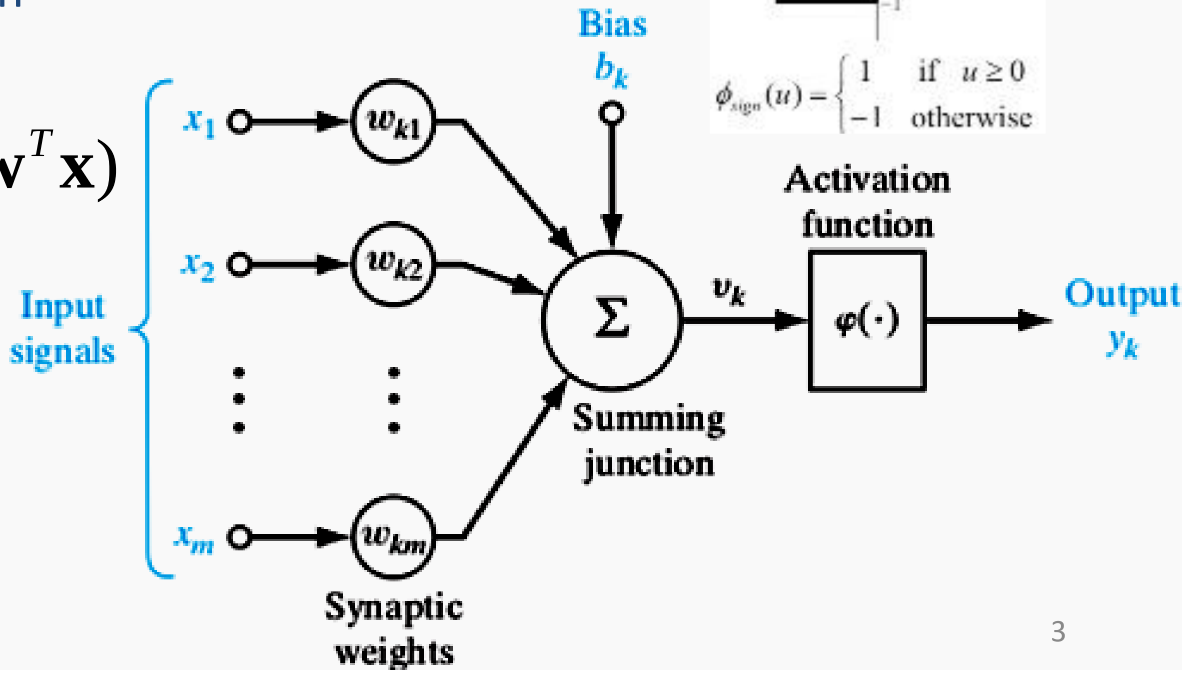
Single-layer Perceptron

- Receives input through its synapsis (x_i)
- Synapsis are weighted (w_i) (including bias)
- A weighted sum is calculated
- Nonlinear activation function

sign function



$$y_k = \varphi\left(\sum_{i=0}^{m} w_{ki} x_i\right) = \varphi(\mathbf{w}^T \mathbf{x})$$



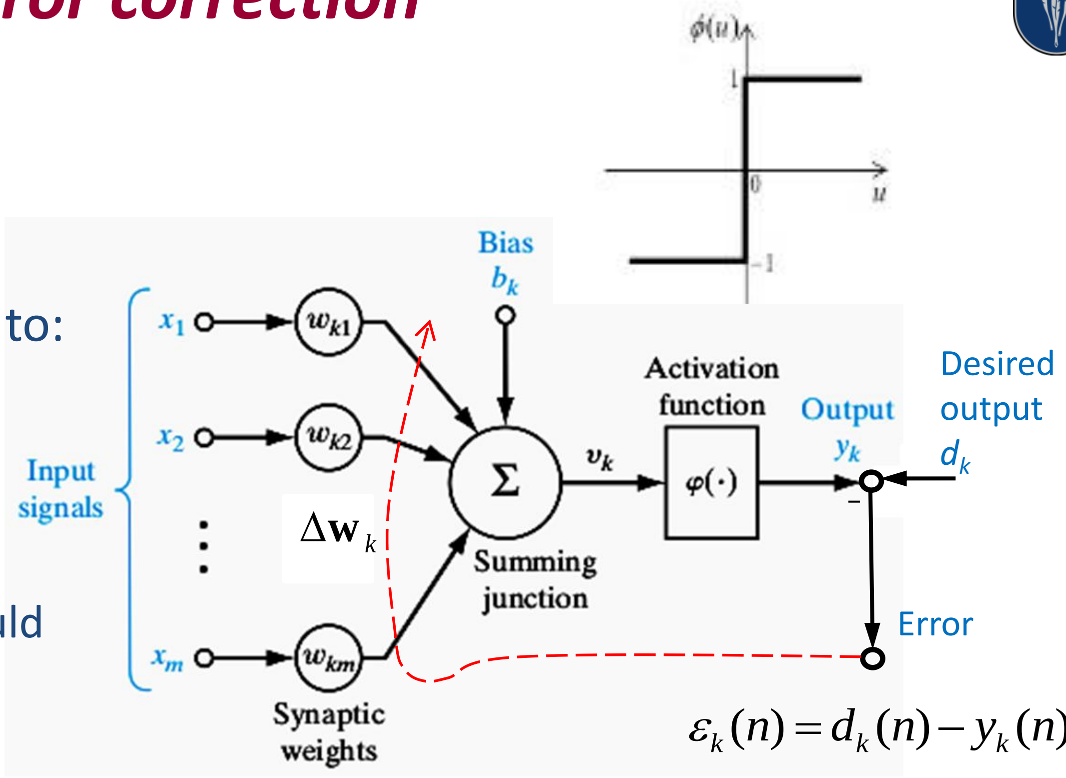
x_i : input vector
w_{ki} : weight coefficient vector
v_k : weigthted sum
b_k : bias value of neuron k
o_k : output value of neuron k

Single-layer perceptron training: *Error correction*

- Apply the input vector (x_i)
- Calculate the output
- If output is false
- Modify the weights according to:

$$\Delta \mathbf{w}_k = \eta \, \varepsilon_k(n) \, \mathbf{x}(n)$$

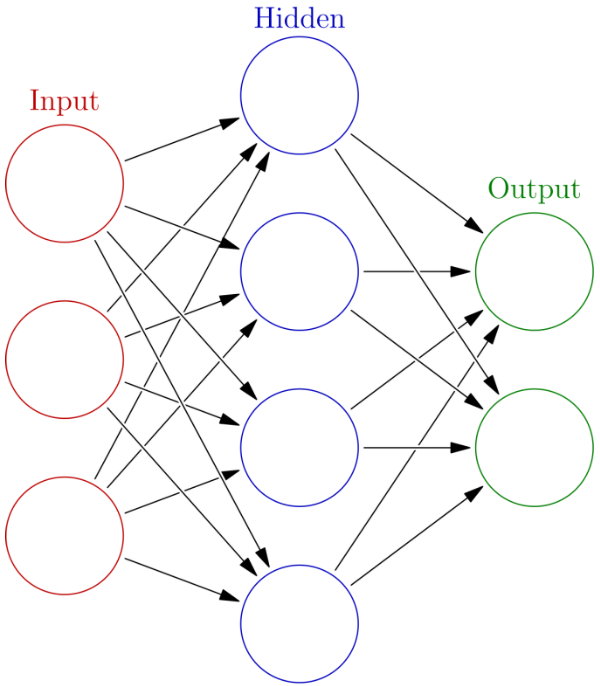
- Operation:
  - When error is positive the contribution of $w_{ki}x_i$ should be increased
- Convergence is proven in case of linearly separable task



$$\varepsilon_k(n) = d_k(n) - y_k(n)$$

Multilayer perceptron

- Different names of Multilayer perceptron
  - Feed forward neural networks (FFNN)
  - Fully connected neural networks
- Multilayer neural network
  - Input layer
  - Hidden layers
  - Output layer
  - The outputs are the inputs of the following layer
  - Many hidden layers → deep network
- Multiple inputs, multiple outputs

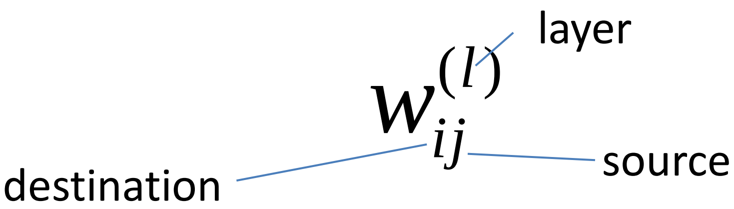


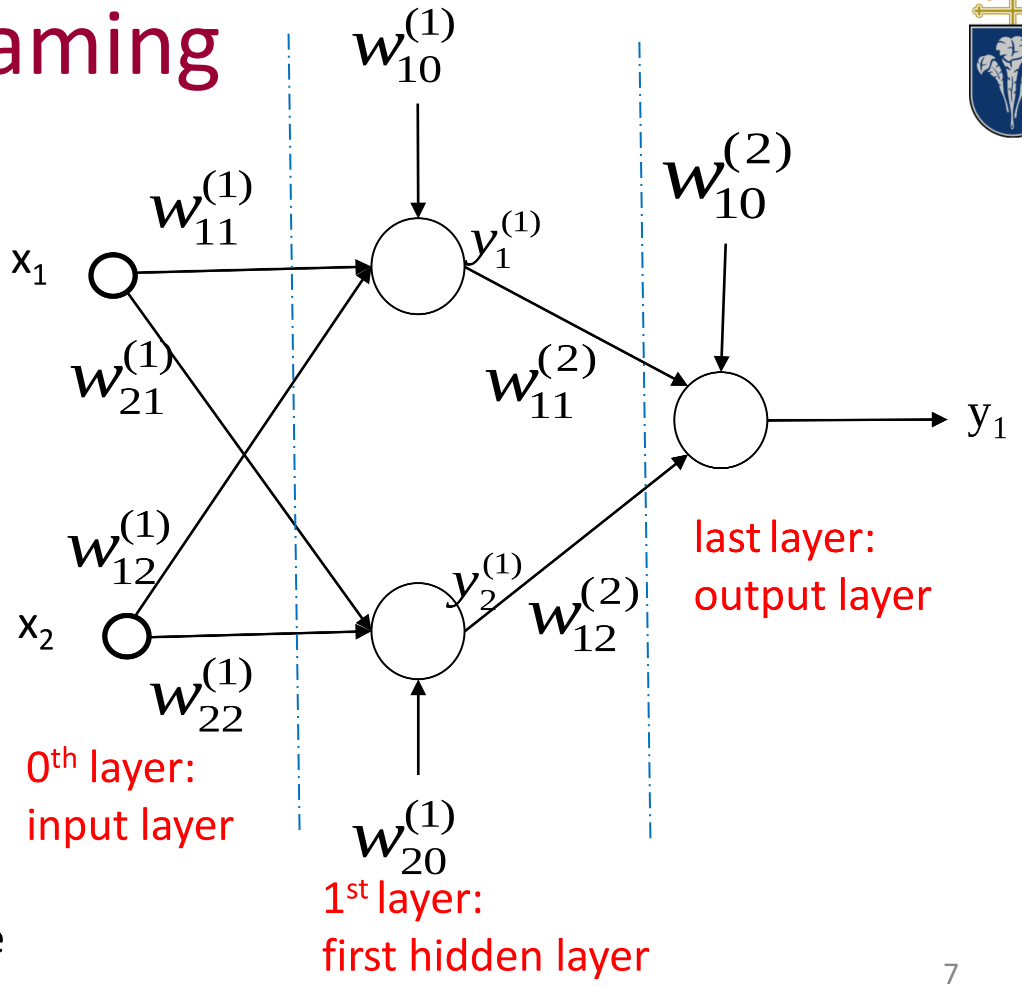
Multilayer perceptron

- Multilayer perceptrons are used for
  - Classification
    - Supervised learning for classification
    - Given inputs and class labels
  - Approximation
    - Approximate an arbitrary function with arbitrary precision
  - Prediction
    - „What is the next element in the future of given time series?”
    - Stock market, currency exchange

Topology and naming

- Weights: $w_{ij}^{(l)}$
  - Arrives to the l^{th} layer
  - Comes from the j^{th} neuron from the $(l\text{-}1)^{\text{th}}$ layer
  - Arrives to the i^{th} neuron of the l^{th} layer

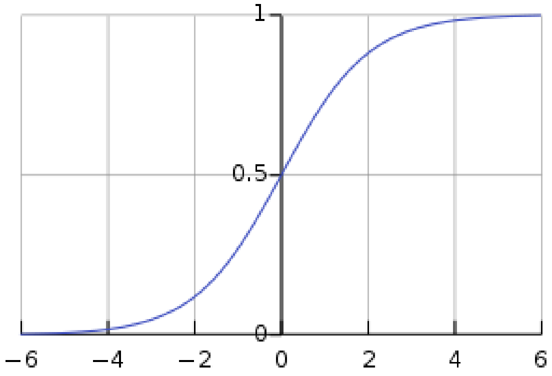




Activation function options

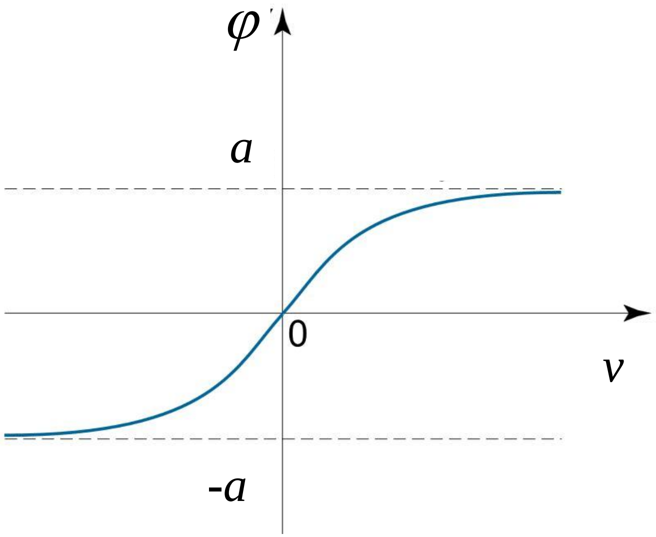
- Sigmoid function
  - Continuous
  - Continuous differentiable
  - We will use this

$$\varphi(v) = \frac{1}{1+e^{-\lambda v}}$$

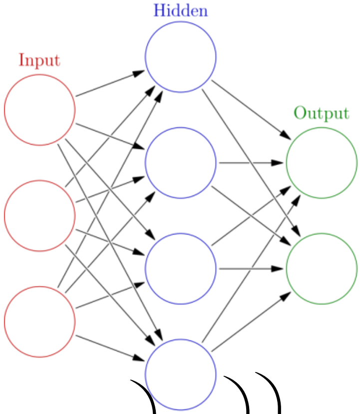


- Hyperbolic tangent function
  - Continuous
  - Continuous differentiable
  - $a, b > 0$

$$\varphi(v) = a\tanh(bv)$$



Operation



- Signal flow through the network progresses left to right

- The output of the network:

$$Net(\mathbf{W},\mathbf{x}) = y = \phi\left(\sum_{i=1}^{n^L} w_i^{(L)} \cdot \phi\left(\sum_{j=1}^{n^{L-1}} w_{ij}^{(L-1)} \cdot \ldots \cdot \phi\left(\sum_{m=1}^{n^1} w_{km}^{(1)} x_m\right) \ldots\right)\right)$$

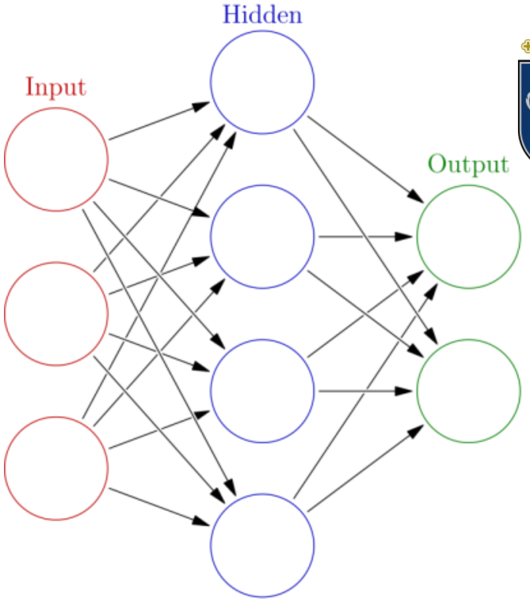
- Where

$$\mathbf{W} = \left(w_{1,0}^{(1)}, w_{1,1}^{(1)}, w_{1,2}^{(1)}, \ldots, w_{1,0}^{(2)}, w_{1,1}^{(2)}, \ldots w_{1,0}^{(L)}, \ldots\right)$$

$$\phi(v) = \frac{1}{1+e^{-\lambda v}} \qquad \phi, \varphi \text{ are the same lower case Phi}$$

- Number of layers: L, neurons in l^{th} layer: n^l

Questions



- When solving engineering task by FFNN we are faced with the following questions:

1. Representation
   - What kind of functions can be represented by an FFNN?
2. Learning
   - How to set up the weights to solve a specific given task?
3. Generalization
   - If only limited knowledge is available about the task which is to be solved, then how the FFNN is going to generalize this knowledge?

Representation

- Can it approximate a function?
  Can it approximate all the function? With what precision?

$$\left.\begin{matrix}\forall F(\mathbf{x}) \in \mathcal{F} \\ \varepsilon > 0\end{matrix}\right\} \rightarrow \exists \mathbf{w} : \left\| F(\mathbf{x}) - Net(\mathbf{x}, \mathbf{w}) \right\| < \varepsilon$$

- The notation || || defines a norm used in $\mathcal{F}$ space

$$\int \mathrm{L}_{\mathbf{x}} \int \left(F(\mathbf{x}) - Net(\mathbf{x}, \mathbf{w}) \right)^{p} \mathbf{d}x, \ldots \mathbf{d}x_{N} < \varepsilon$$

- For example error computed as follows in L^p

Representation – Theorem 1

- Theorem (Harnik, Stinchambe, White 1989)
- Every function in L^p can be represented arbitrarily closely approximation by a neural net
- More precisely for each

$$F(x) \in L^p$$

$$\forall \varepsilon > 0, \exists \mathbf{w}$$

$$\int \mathrm{L}_{\mathbf{x}} \int \left(F(\mathbf{x}) - Net(\mathbf{x}, \mathbf{w}) \right)^p \mathbf{d}x, \ldots \mathbf{d}x_N < \varepsilon$$

- Since it is out of the focus of the course this proof will not be presented here

Recall:

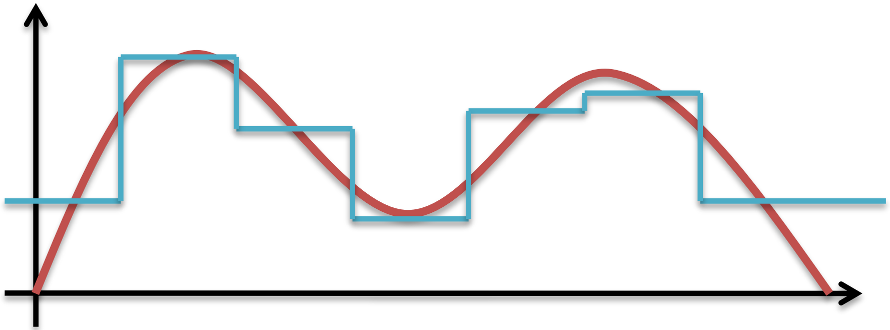
$$L^1 : \int \mathrm{L}_{\mathbf{x}} \int \left(F(x) \right) \mathbf{d}x, \ldots \mathbf{d}x_N < \infty$$

$$L^2 : \int \mathrm{L}_{\mathbf{x}} \int \left(F(x) \right)^2 \mathbf{d}x, \ldots \mathbf{d}x_N < \infty$$

$$L^p : \int \mathrm{L}_{\mathbf{x}} \int \left(F(x) \right)^p \mathbf{d}x, \ldots \mathbf{d}x_N < \infty$$

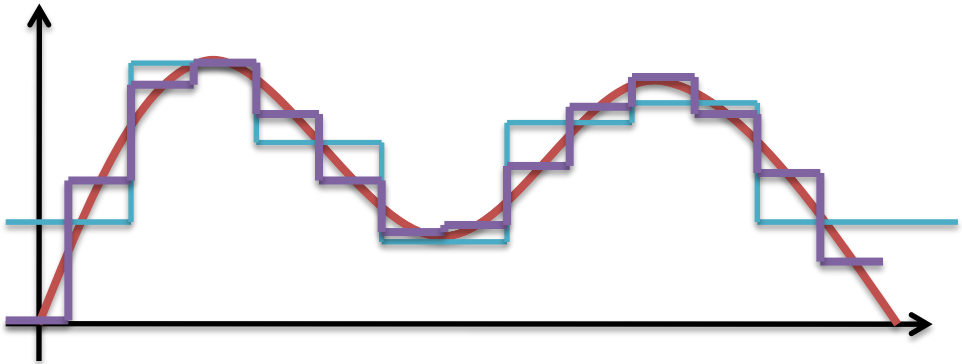
Representation – Blum and Li theorem

- Theorem: $F(x) \in L^2$
  $\forall \varepsilon > 0, \exists \mathbf{w}$
- Proof: $\int_{\mathbf{x}}^{\mathbf{L}} \int \left(F(\mathbf{x}) - Net(\mathbf{x}, \mathbf{w})\right)^2 \mathbf{d}x, \ldots \mathbf{d}x_N < \varepsilon$
  - Using the step functions: S
  - From elementary integral theory it is clear every function can be approximated by appropriate step function sequence



Representation – Blum and Li theorem

- This step function can have arbitrary narrow steps
- For example each step could be divided into two sub-steps
- Therefore we can synthetize

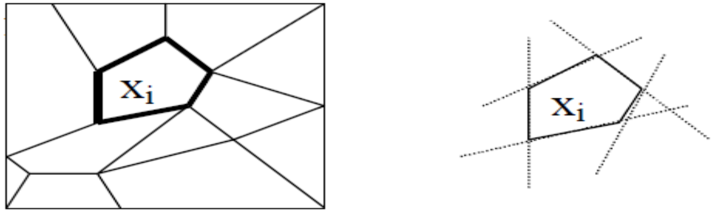


$$I(X) = \begin{cases} 1 & \text{if } \mathbf{x} \in X \\ 0 & \text{else} \end{cases}$$

$$F(x) \cong \underbrace{\sum_i F(x_i) I(x_i)}_{s(x)}$$

Representation – Blum and Li theorem

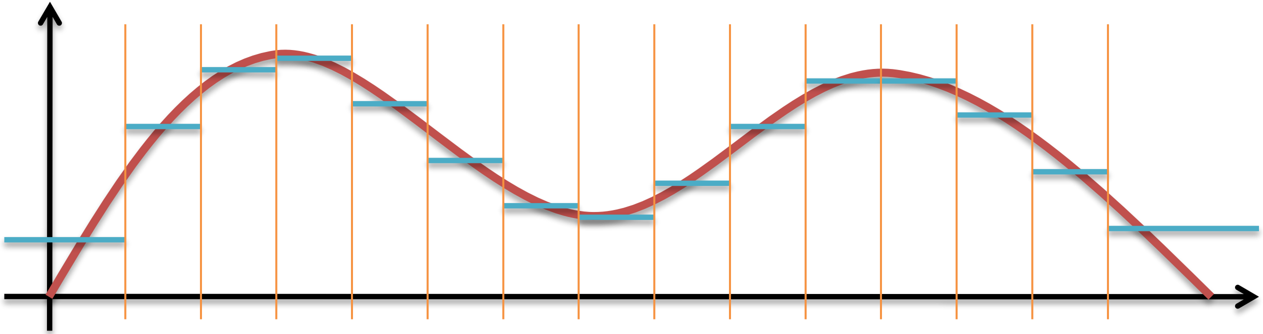
- These steps partition the domain of the function
- One partition can be easily represented by small neural network
  - In two dimension the following figure gives an example



  - The borders of the partition are hyper planes which could represented by one perceptron

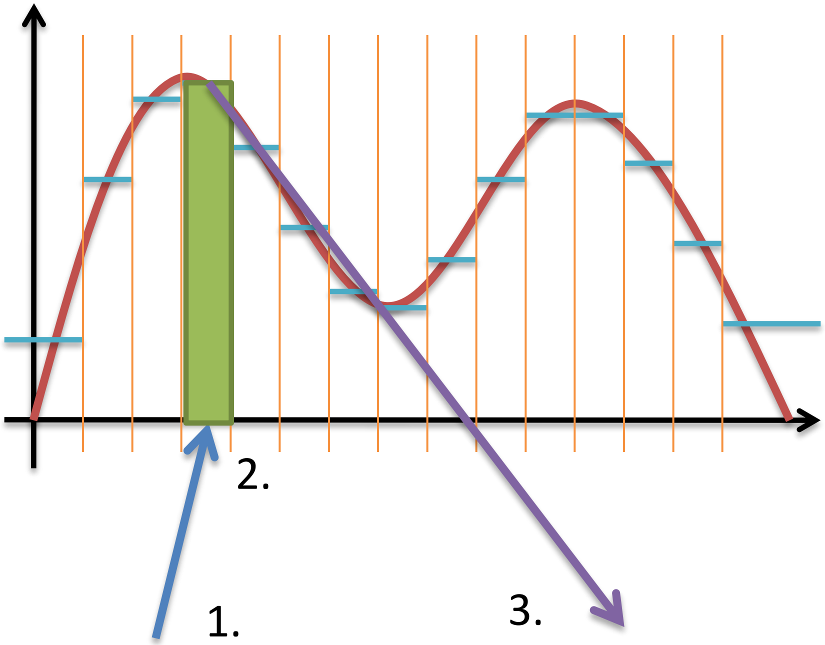
Representation – Blum and Li construction

- The Blum and Li construction is based on the „LEGO” principle
- The approximation of the F function is based on its step functions
- This step function partitions the domain of the original F function
- For each partition there is a neuron responsible for approximation the „step”
- If the input of the FFNN (x) falls into a given range the appropriate approximator neuron has to be selected
- The output of the network should be this selected value



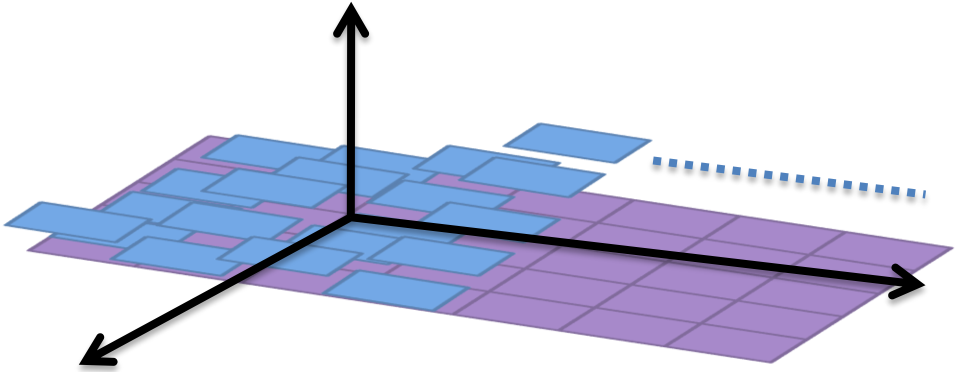
Representation – Blum and Li construction

1. Incoming arbitrary *x* value
2. The appropriate interval will be selected
3. The response of the network is the response of selected neuron (approximator)



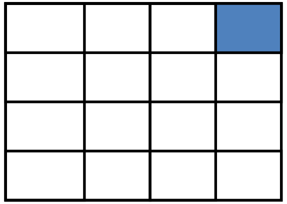
Representation – Blum and Li construction

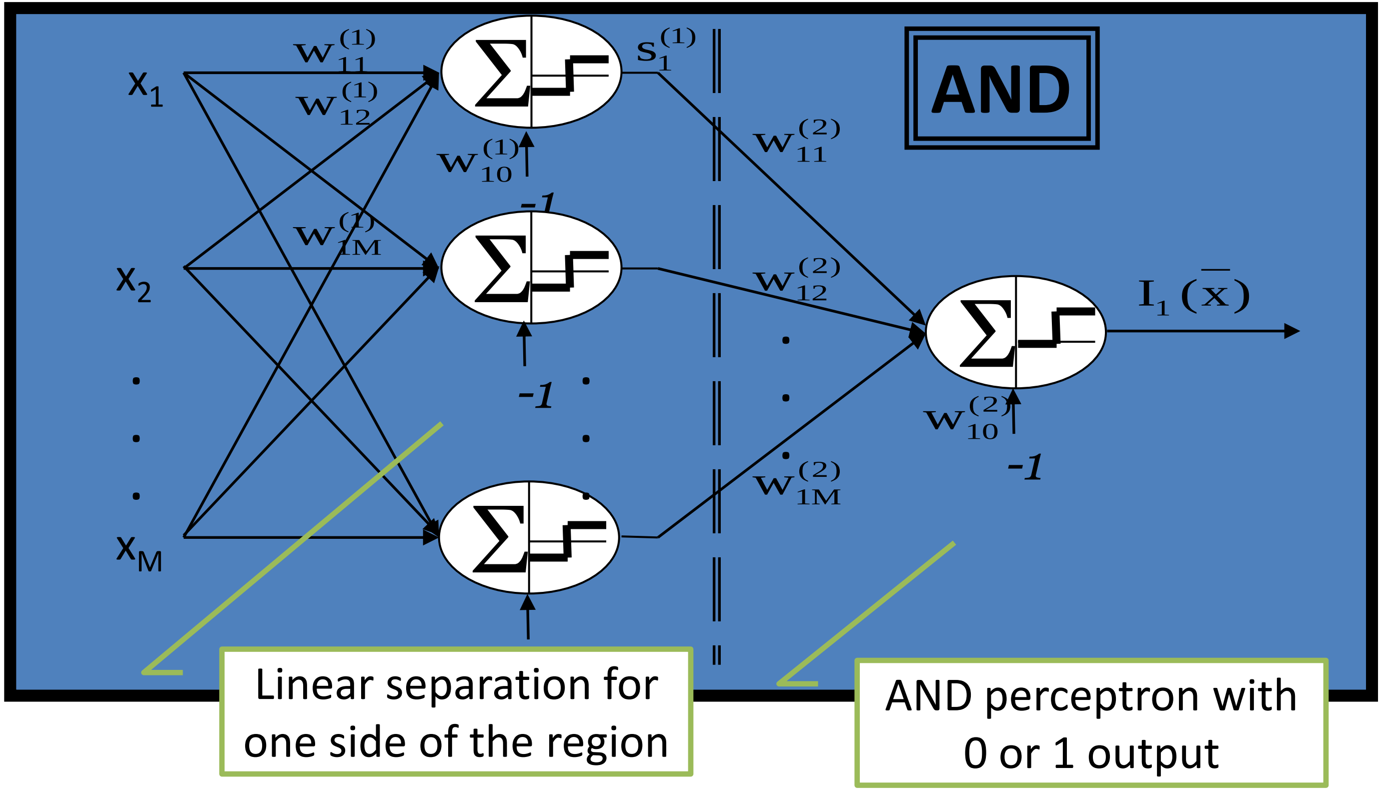
- This construction …
  - … has no dimensional limits
  - … has no equidistance restrictions on tiles (partitions)
  - … can be further fined, and the approximation can be any precise
- 2 dimensional example
  - The tiles are the top of the columns for each approximation cell



Representation – Blum and Li construction

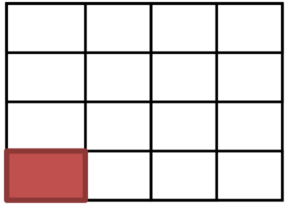
- Construction for one particular region
- The output is I_1 if we are in this region

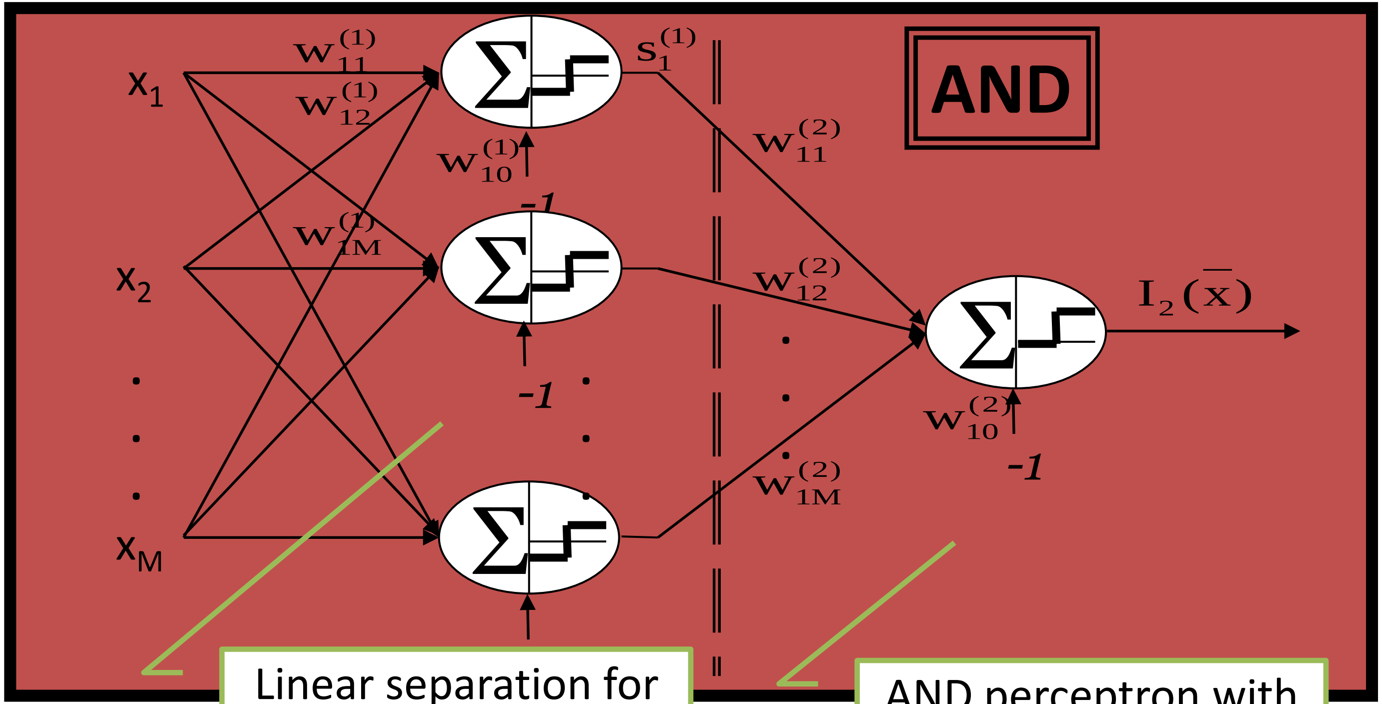




Representation – Blum and Li construction

- Construction for one particular region
- The output is I_2 if we are in this region



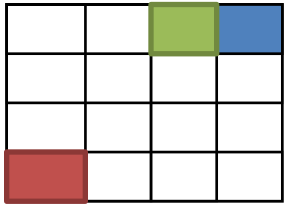


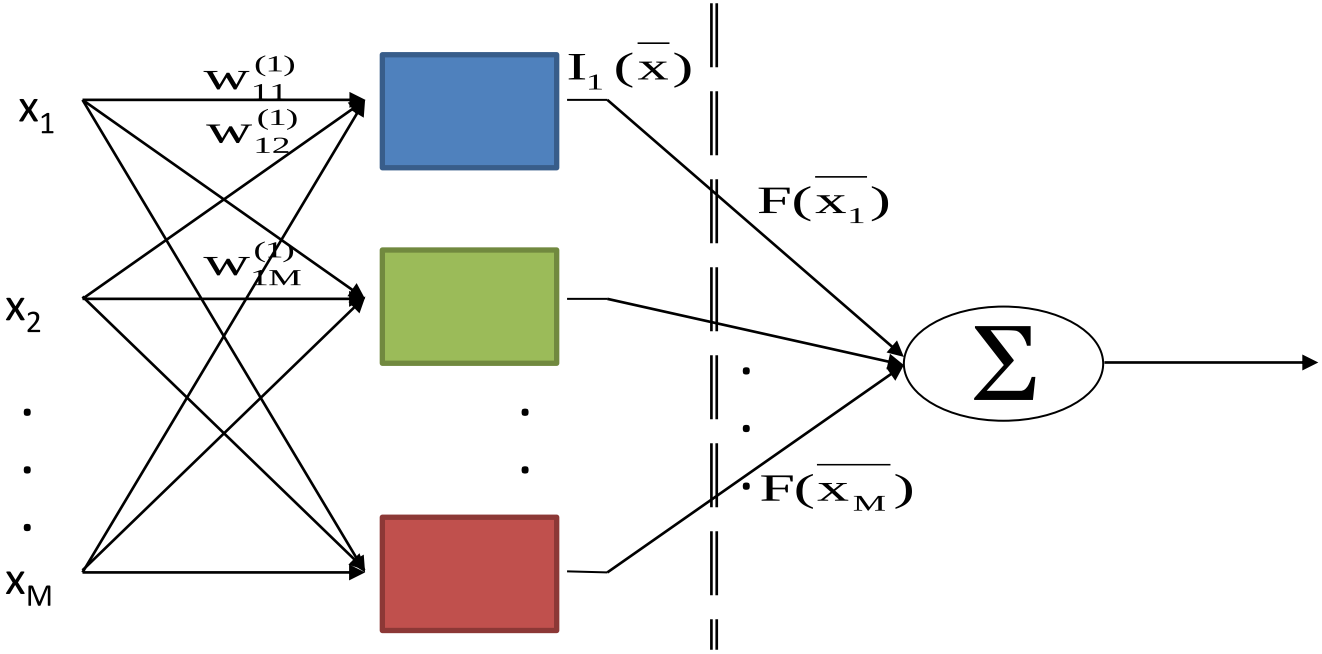
Linear separation for one side of the region

AND perceptron with 0 or 1 output

Representation – Blum and Li construction

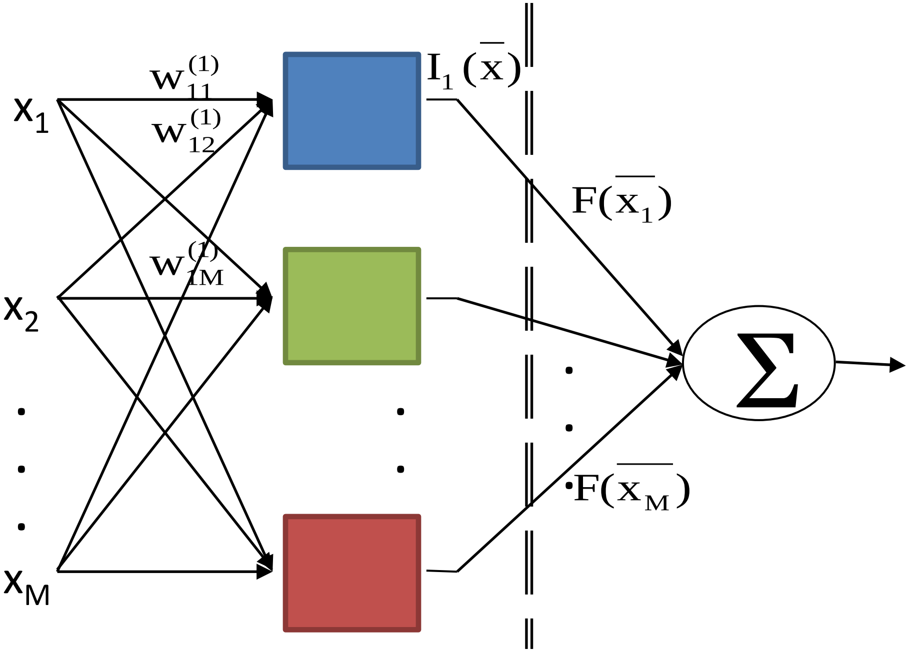
- Each region is being approximated by a block specified above





Representation – Blum and Li construction

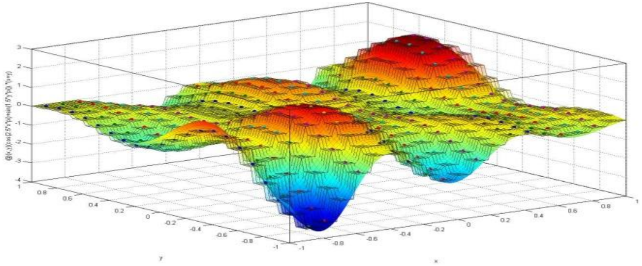
- Third layer
  - This neuron has linear activation function
  - The weights of this neuron are the approximation values of the F function
  - Thus the approximation for the whole domain of the original F function is done by FFNN

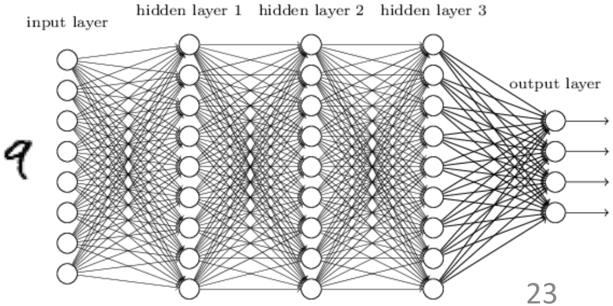


Blum and Li – Limitations

- The size of the FFNN constructed via this method is quite big
- Consider the task on the picture, where let us have 1000 by 1000 cell to approximate the function
- General case:

  ~2 Million neurons are needed
- Smoother approximation needs more
- We are after to find a less complicated architecture





Learning

$$\mathbf{w}_{\text{opt}} : \min_{\mathbf{w}} \left\| \mathrm{F}(\mathbf{x}) - \mathrm{Net}(\mathbf{x},\mathbf{w}) \right\|^2 = \min_{\mathbf{w}} \int .. \int \left(\mathrm{F}(\mathbf{x}) - \mathrm{Net}(\mathbf{x},\mathbf{w}) \right)^2 dx_1 ... dx_N$$

- Nor minimization task neither construction is possible most cases
  - Complete information would be needed about F(*x*), however it is typically unknown
- Weak learning in incomplete environment, instead of using F(*x*)

$$\tau^{(K)} = \left\{ \left(\mathbf{x}_k, d_k \right); k = 1, ..., K \right\}$$

- A training set is being constructed of observations

Learning

- Rather than minimizing the error function

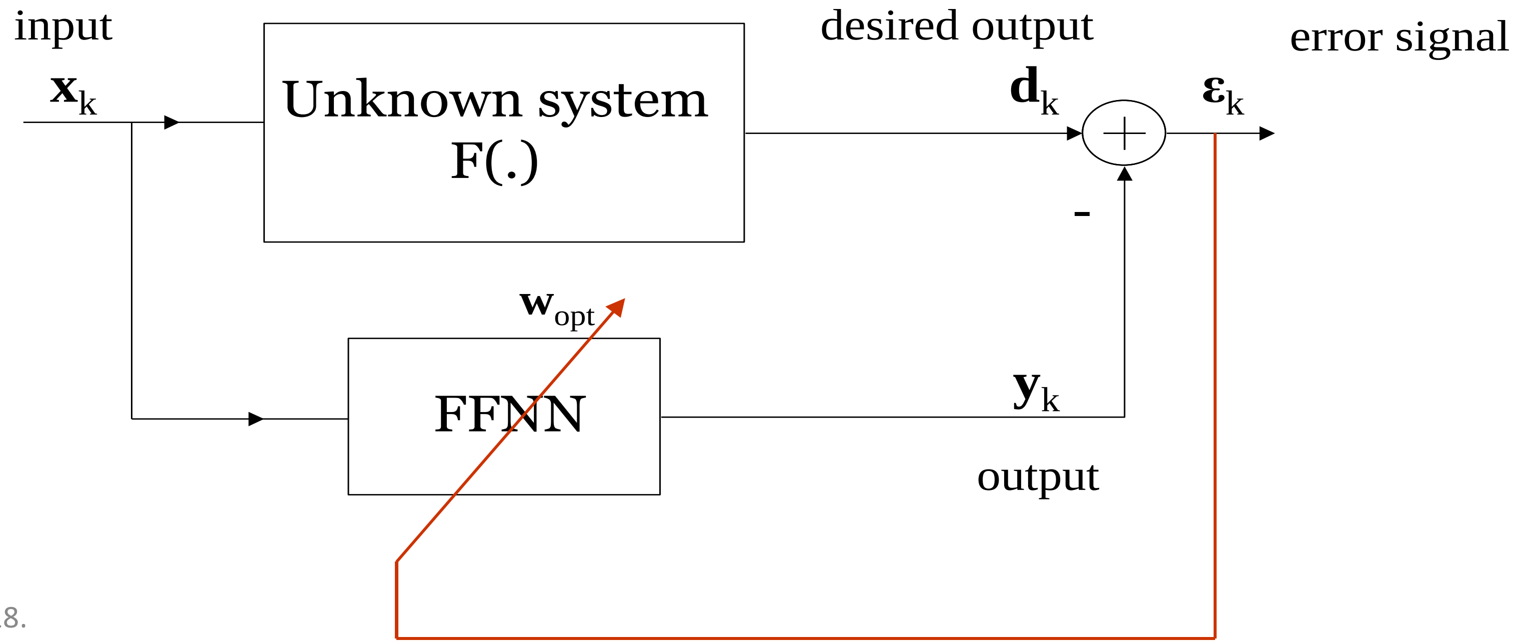
$$\mathbf{w}_{\text{opt}} : \min_{\mathbf{w}} \left\| \mathrm{F}(\mathbf{x}) - \mathrm{Net}(\mathbf{x},\mathbf{w}) \right\|^2 = \min_{\mathbf{w}} \int .. \int \left(\mathrm{F}(\mathbf{x}) - \mathrm{Net}(\mathbf{x},\mathbf{w}) \right)^2 dx_1 ... dx_N$$

- The approximation is the best achievable
  - F function is known in a limited positions (training set)

$$\mathbf{w}_{\text{opt}}^{(K)} : \min_{\mathbf{w}} \frac{1}{K} \sum_{k=1}^{K} \left(d_k - Net(\mathbf{x}_k, \mathbf{w}) \right)^2$$

Learning

$$\mathbf{w}_{opt} : \min_{\mathbf{w}} \left\| F(\mathbf{x}) - Net(\mathbf{x},\mathbf{w}) \right\|^2 = \min_{\mathbf{w}} \int .. \int \left(F(\mathbf{x}) - Net(\mathbf{x},\mathbf{w}) \right)^2 dx_1 ... dx_N$$



Learning

- The questions are the following
  - What is the relationship of these optimal weights

$$\mathbf{w}_{\text{opt}} \overset{???}{\Leftrightarrow} \mathbf{w}_{\text{opt}}^{(K)}$$

$$\mathbf{w}_{\text{opt}}^{(K)} : \min_{\mathbf{w}} \frac{1}{K} \sum_{k=1}^{K} \left(d_k - Net\left(\mathbf{x}_k, \mathbf{w}\right)\right)^2$$

  - How this new objective function should be minimized as quickly as possible

Statistical learning theory

- Empirical error

$$R_{emp}(\mathbf{w}) = \frac{1}{K}\sum_{k=1}^{K}\left(d_k - Net(\mathbf{x}_k, \mathbf{w})\right)^2$$

- Theoretical error

$$\left\|\mathrm{F}(\mathbf{x}) - \mathrm{Net}(\mathbf{x},\mathbf{w})\right\|^2 = \int_X \dots \int \left(\mathrm{F}(\mathbf{x}) - \mathrm{Net}(\mathbf{x},\mathbf{w})\right)^2 dx_1 \dots dx_N$$

- Let us have $\mathbf{x}_k$ random variables subject to uniform distribution

Statistical learning theory

- $\mathbf{x}_k$ random variable, where d=F($\mathbf{x}$)

$$\lim_{k\to\infty} = \frac{1}{K}\sum_{k=1}^{K}\left(d_k - Net\left(\mathbf{x}_k, \mathbf{w}\right)\right)^2 = \mathrm{E}\left(d - Net(\mathbf{x}, \mathbf{w})\right)^2 =$$

$$\int \underset{X}{\cdots} \int \left(\mathrm{F}(\mathbf{x}) - \mathrm{Net}\left(\mathbf{x}, \mathbf{w}\right)\right)^2 p(\mathbf{x}) dx_1 \ldots dx_N =$$

Because it is ~ constant due to the uniformity

$$\frac{1}{|X|}\int \underset{X}{\cdots} \int \left(\mathrm{F}(\mathbf{x}) - \mathrm{Net}\left(\mathbf{x}, \mathbf{w}\right)\right)^2 dx_1 \ldots dx_N \; :$$

$$\int \underset{X}{\cdots} \int \left(\mathrm{F}(\mathbf{x}) - \mathrm{Net}\left(\mathbf{x}, \mathbf{w}\right)\right)^2 dx_1 \ldots dx_N$$

Statistical learning theory

- Therefore

$$\underset{K\to\infty}{\text{l.i.m.}}\,\mathbf{w}_{\text{opt}} = \mathbf{w}_{\text{opt}}^{(K)}$$

- Where l.i.m. means: lim in mean

$$\lim_{K\to\infty} R_{emp}(\mathbf{w}) = R_{th}(\mathbf{w})$$

$$\lim_{K\to\infty} \frac{1}{K}\sum_{k=1}^{K}\left(d_k - Net(\mathbf{x}_k,\mathbf{w})\right)^2 = \int_X \dots \int \left(\mathrm{F}(\mathbf{x}) - \mathrm{Net}(\mathbf{x},\mathbf{w})\right)^2 dx_1 \dots dx_N$$

Weak learning is satifactory!

Learning – in practice

- Learning based on the training set:

$$\tau^{(K)} = \{(\mathbf{x}_k, d_k); k = 1, \dots, K\}$$

- Minimize the empirical error function (R_{emp})

$$\mathbf{w}_{\text{opt}}^{(K)} : \min_{\mathbf{w}} \frac{1}{K} \sum_{k=1}^{K} \underbrace{\left(d_k - Net(\mathbf{x}_k, \mathbf{w})\right)^2}_{E_k} = \min_{\mathbf{w}} R_{emp}(\mathbf{w})$$

- Learning is a multivariate optimization task

Learning – Newton method

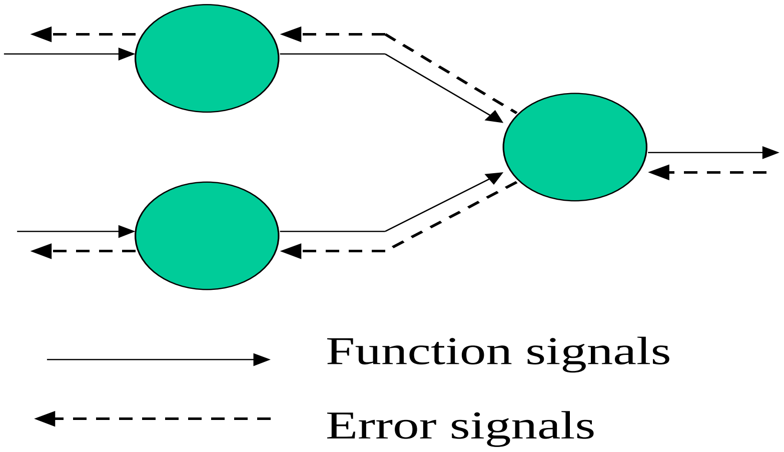
- First order gradient based optimization method:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \eta \cdot \underset{\mathbf{w}}{\mathrm{grad}} \left\{ R_{\mathrm{emp}} \left(\mathbf{w}(k) \right) \right\}$$

- Iterative method
  - Each step modify the weights
  - To reduce the error
- The empirical error of the actual neuron is computed
- The gradient of this error is used to modify the weight

Learning

- The Rosenblatt algorithm is inapplicable,
  - the error and desired output in the hidden layers of the FFNN **<u>is unknown</u>**
- Someway the error of the whole network has to be distributed to the internal neurons, in a feedback way



Forward propagation of function signals and back-propagation of errors signals

Sequential back propagation

- Adapting the weights of the FFNN

$$w_{ij}^{(l)}(k+1) = w_{ij}^{(l)}(k) + \Delta w_{ij}^{(l)}(k)$$

$$\Delta w_{ij}^{(l)}(k) = ?$$

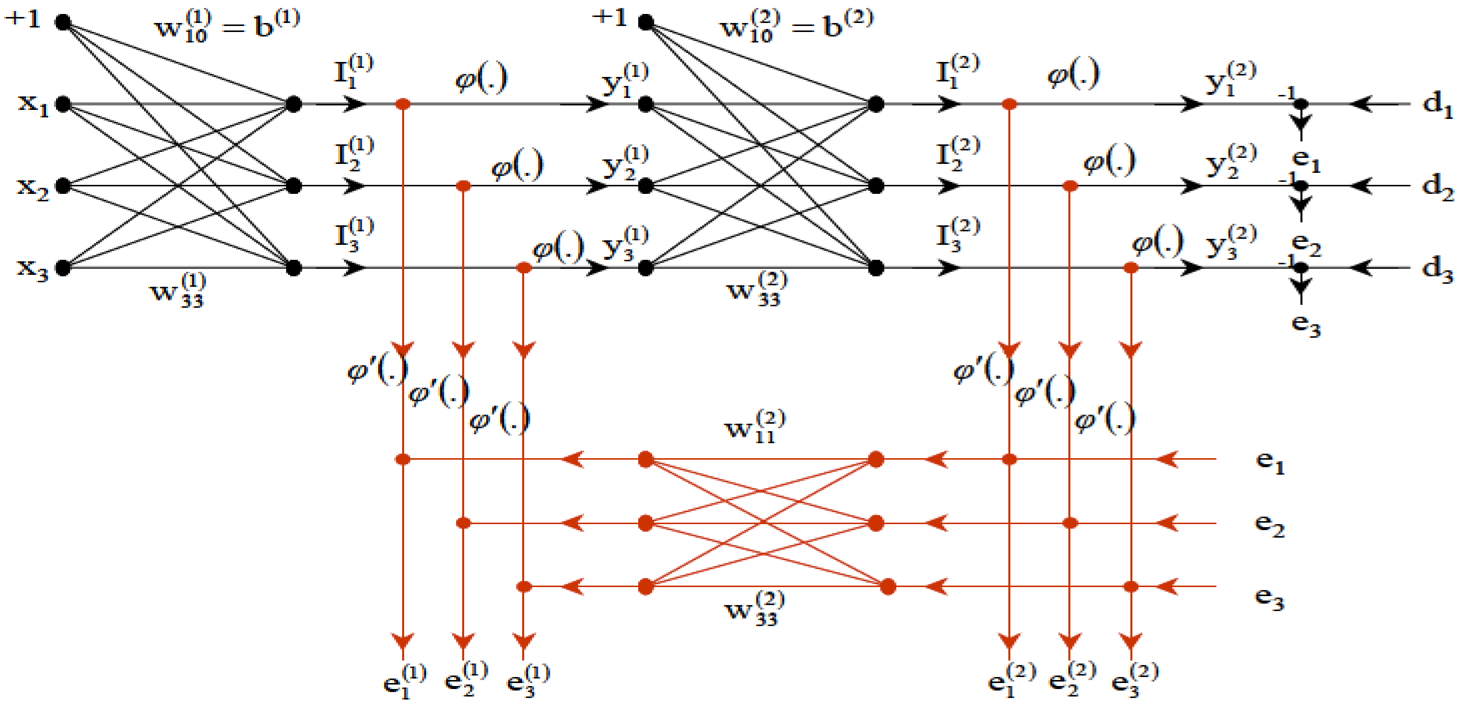
- The weights are modified towards the differential of the error function:

$$\Delta w_{ij}^{(l)} = -\eta \frac{\partial R_{emp}}{\partial w_{ij}^{(l)}}$$

- The elements of the training set adapted by the FFNN sequentially

$$R_{emp} = R_{emp}(y(\mathbf{x}), d)$$

Propagation and back propagation



Sequential or batch training mode

- Sequential mode:
  - For each training vector both the forward and the backward propagation is done one after the other
  - Weights are updated after each input
- Batch mode:
  - All the training vectors are applied, and the total error of the training set is calculated
  - The weight updates are calculated with the accumulated error

Literature

- *Simon Haykin:* **Neural Networks and Learning Machines**
- *Back propagation: Page 129-141*
- http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf