Development of Complex Curricula for Molecular Bionics and Infobionics Programs within a consortial* framework**

Consortium leader

PETER PAZMANY CATHOLIC UNIVERSITY

Consortium members

SEMMELWEIS UNIVERSITY, DIALOG CAMPUS PUBLISHER

The Project has been realised with the support of the European Union and has been co-financed by the European Social Fund ***

**Molekuláris bionika és Infobionika Szakok tananyagának komplex fejlesztése konzorciumi keretben

***A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Digital- and Neural Based Signal Processing & Kiloprocessor Arrays

Digitális- neurális-, és kiloprocesszoros architektúrákon alapuló jelfeldolgozás

Feedforward Neural Networks

Előrecsatolt neurális hálózatok

J. Levendovszky, A. Oláh, K. Tornai

Contents

- Introduction – topology
- Representation capability
- Blum and Li construction
- Generalization capabilities
- Bias variance dilemma
- Learning
- Applications

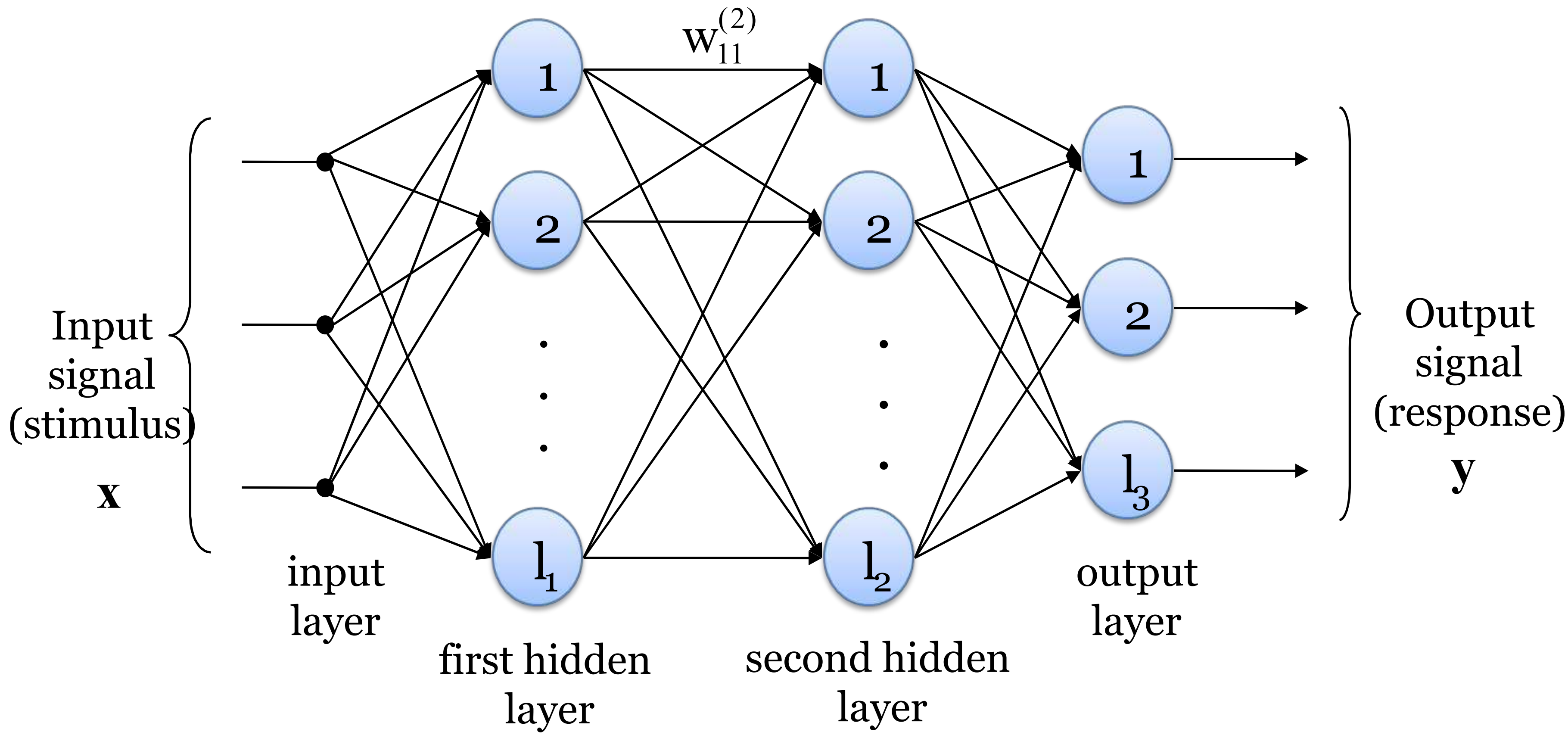
Introduction – FFNN

- Multilayer neural network
  - Input layer
  - Intermediate (hidden) layers
  - Output layer
  - The outputs are the inputs of the following layer
- Multiple inputs, multiple outputs
- Each layer contains a number of nonlinear perceptrons

Introduction – FFNN

- Feed Forward Neural Networks are used for
  - Classification
    - Supervised learning for classification
    - Given inputs and class labels
  - Approximation
    - Arbitrary function with arbitrary precision
  - Prediction
    - „What is the next element in the future of given time series?”

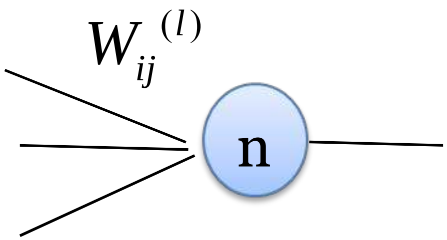
Topology



Input signal (stimulus)
x
input layer
1
2
l_1
first hidden layer
$w_{11}^{(2)}$
1
2
l_2
second hidden layer
1
2
l_3
output layer
Output signal (response)
y

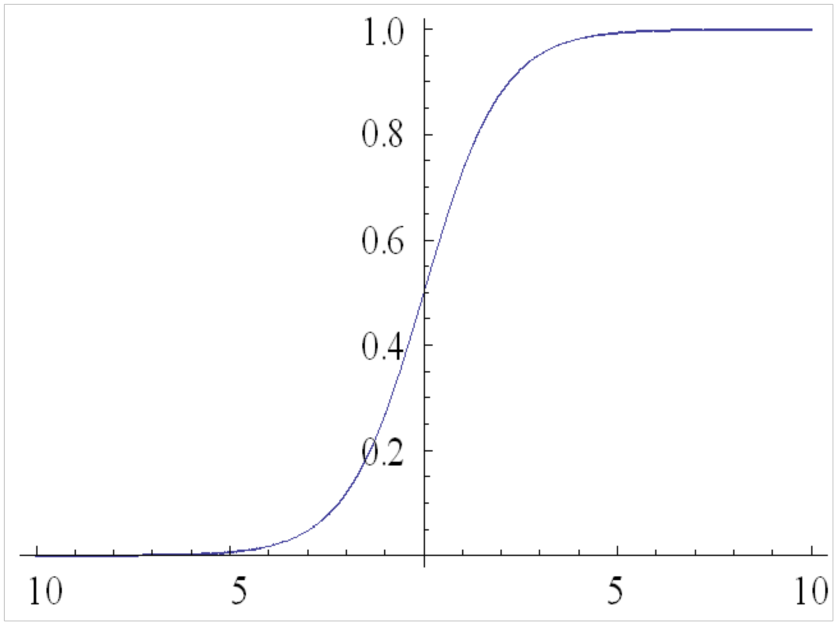
Topology

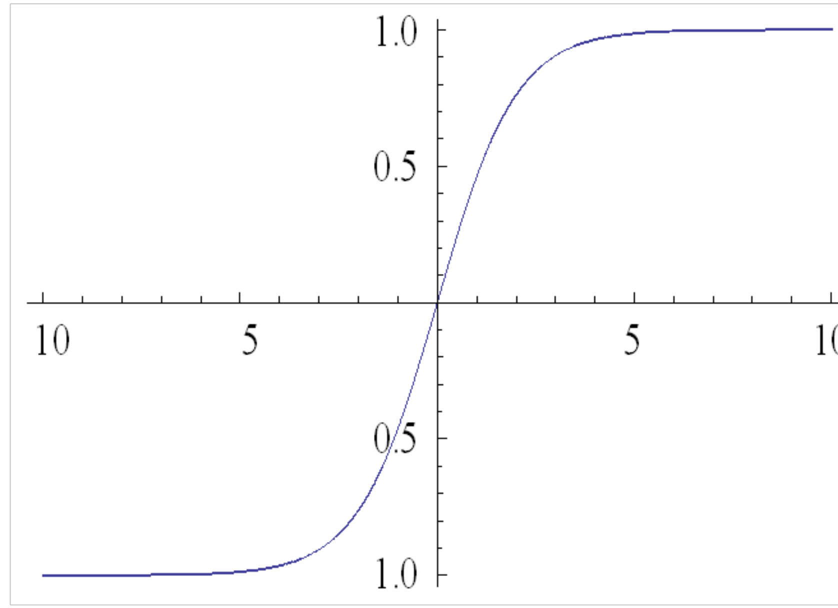
- Each cell
  - Weights
    - l^{th} layer
    - i^{th} neuron in the l^{th} layer
    - From the j^{th} neuron of the $(l\text{-}1)^{th}$ layer
  - Nonlinear activation function (logistic function, biologically motivated)



$$\phi(u) = \frac{1}{1+e^{-u}} \qquad \phi(u) = \frac{2}{1+e^{-\lambda u}} - 1$$

Activation functions





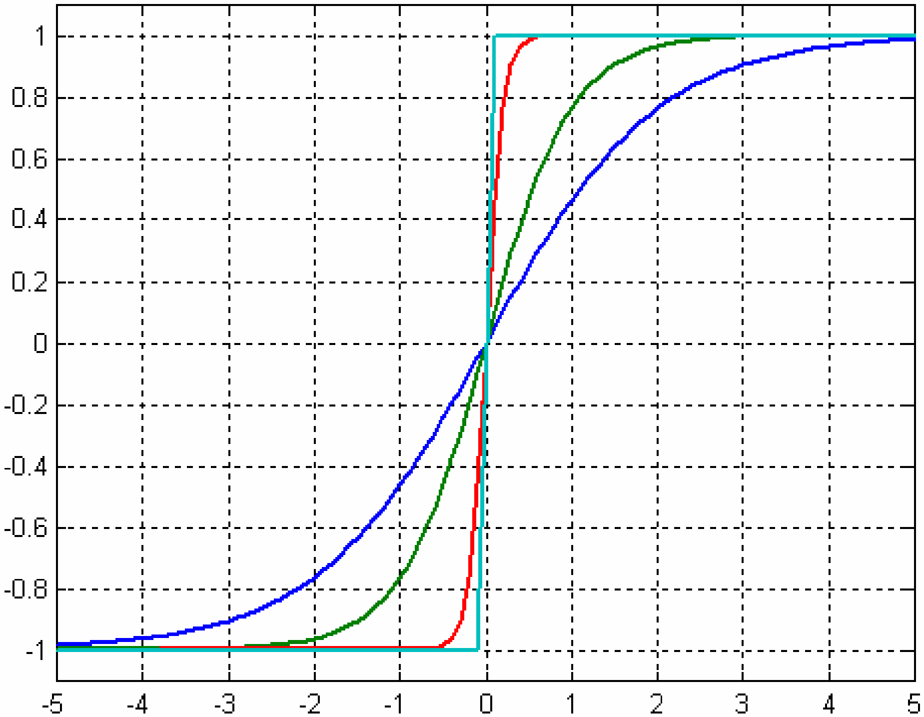
$$\phi(u) = \frac{1}{1+e^{-\gamma u}}$$

$$\phi(u) = \frac{2}{1+e^{-\lambda u}} - 1$$

Activation functions

- The parameter of the sigmoid function may be different as it can be seen on the figure

$$\phi(u) = \frac{2}{1+e^{-\lambda u}} - 1$$



FFNN – mode of operation

- 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

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

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

FFNN – weights

- The free parameters, called weights
  - Can be changed in course of adaptation (learning) process in order to „tune” the network for performing a special task
  - This learning procedure will be discussed later

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

FFNN – questions

1. Representation
   – How many different tasks 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

FFNN in operation

- The neural network works as follows
  - the network should be created by the specification
  - the weights of the network are set so the error of the network should be minimal
  - The weights are set by the training sequence

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

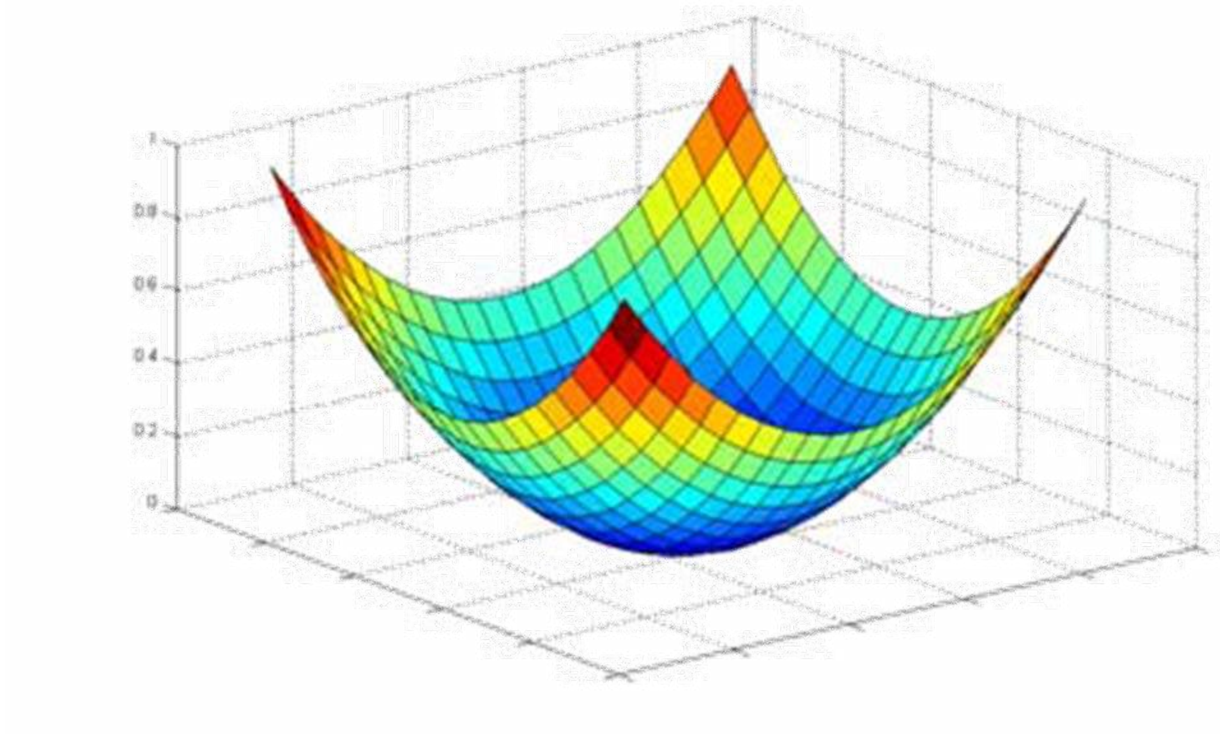
  - The learning is lead through the error function, which determines the adaptation of the weights of the neural network on the error surface

FFNN – in operation

- most cases the error function is chosen to be the square error
- the adaptation of weights can be done by different methods
  - usually the gradient descent method is used
- In simple problems the error function is a quadratic function
  - It has only one minimum, so the convergence to the global optima can be assured.

FFNN – Example of error function

- Possible quadratic error surface
  - The learning task is to find the global minimum



Representation

- In the following the representation capability of the FFNN will be discussed.
- We seek the $\mathcal{F}$ function space where the FFNN approximation is uniformly dense

$$Net(\mathbf{w},\mathbf{x}) \in \mathcal{NN}$$

$$\mathcal{NN} \subseteq_D \mathcal{F}$$

$$F(\mathbf{x}) \in \mathcal{F}$$

- ($\subseteq_D$ symbol denotes the fact that the $\mathcal{NN}$ is uniformly dense in $\mathcal{F}$.

Representation

- In this function space every function can be arbitrarily approximated with FFNN

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

- The notation || || defines a norm used in $\mathcal{F}$ space
- For example error computed as follows in L^p

$$\int_{\mathbf{X}} \cdots \int \left(F(\mathbf{x}) - Net(\mathbf{x}, \mathbf{w}) \right)^p \mathbf{d}x, \ldots \mathbf{d}x_N < \varepsilon$$

Representation – Theorem 1

Theorem (Harnik, Stinchambe, White 1989)

- The FFNN-s are uniformly dense in the L^p space

  $$\mathcal{NN} \subseteq_D L^p$$

- Recall:

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

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

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

Representation – Theorem 1

Theorem (Harnik, Stinchambe, White 1989)

- In other words every function in L^p can be represented arbitrarily closely approximation by a neural net
- More precisely for each $F(x) \in L^p$

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

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

Representation – Theorem 1

Theorem (Harnik, Stinchambe, White 1989)

- Since L^p is a rather large space, the theorem implies that almost any engineering task can be solved by a one-layer neural network
- The proof of theorem heavily draws from functional analysis and is based on the Hahn-Banach theorem.
- Since it is out of the focus of the course this proof will not be presented here.

Representation – Blum and Li theorem

Theorem (Blum and Li)

- The FFNN-s are uniformly dense in the L^2 space

  $\mathcal{NN} \subseteq_D L^2$

- In other words:
  - For each $F(x) \in L^2$

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

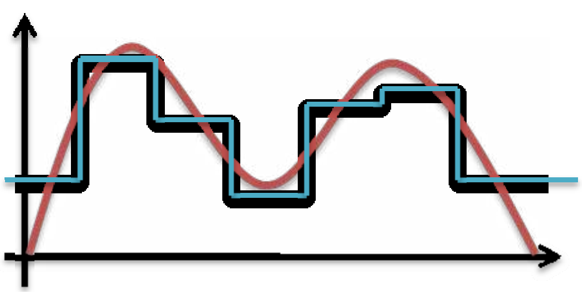
    $\int_{\mathbf{X}} \cdots \int \left(F(\mathbf{x}) - Net(\mathbf{x}, \mathbf{w})\right)^2 \mathbf{d}x, \ldots \mathbf{d}x_N < \varepsilon$

Representation – Blum and Li theorem

- Theorem: $\forall F(\mathbf{x}) \in \mathcal{F}, \varepsilon > 0 \rightarrow \exists \mathbf{w} : \int \cdots_{\mathbf{X}} \int \left(F(\mathbf{x}) - Net(\mathbf{x}, \mathbf{w}) \right)^2 \mathbf{d}x, \ldots \mathbf{d}x_N < \varepsilon$
- Proof:
  - Using the step functions: S
  - From elementary integral theory it is clear that S is uniformly dense in L^1 , namely every function in L^1 can be approximated by an appropriate step function (figure)

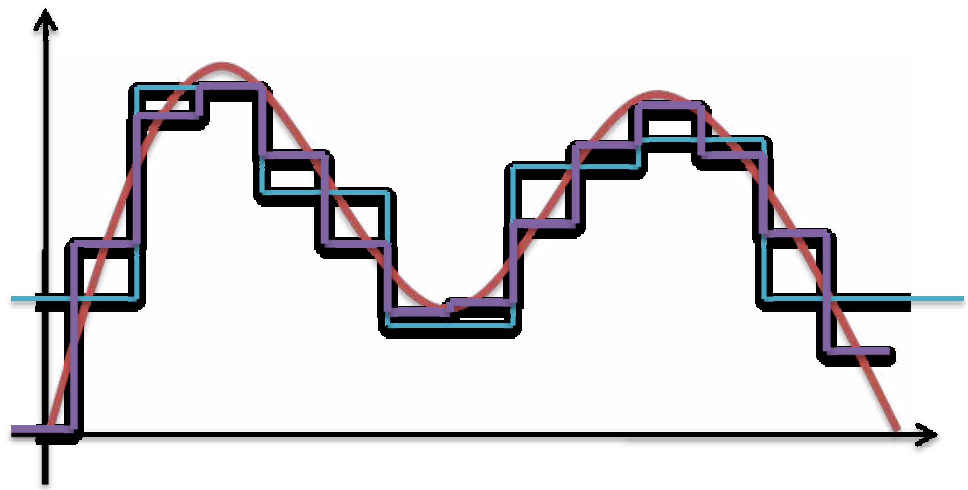
$$S \subseteq_D L^1 \subseteq_D L^2$$

$$S := \left\{ s(\mathbf{x}) : s(\mathbf{x}) = \sum_i a_i I(x_i) \right\}$$



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

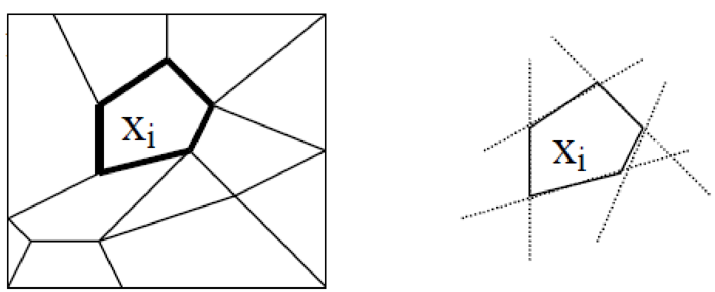


$$I(X) = \begin{cases} 1 & \text{if } \mathbf{x} \in X \\ 0 & \text{else} \end{cases}$$

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

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 theorem

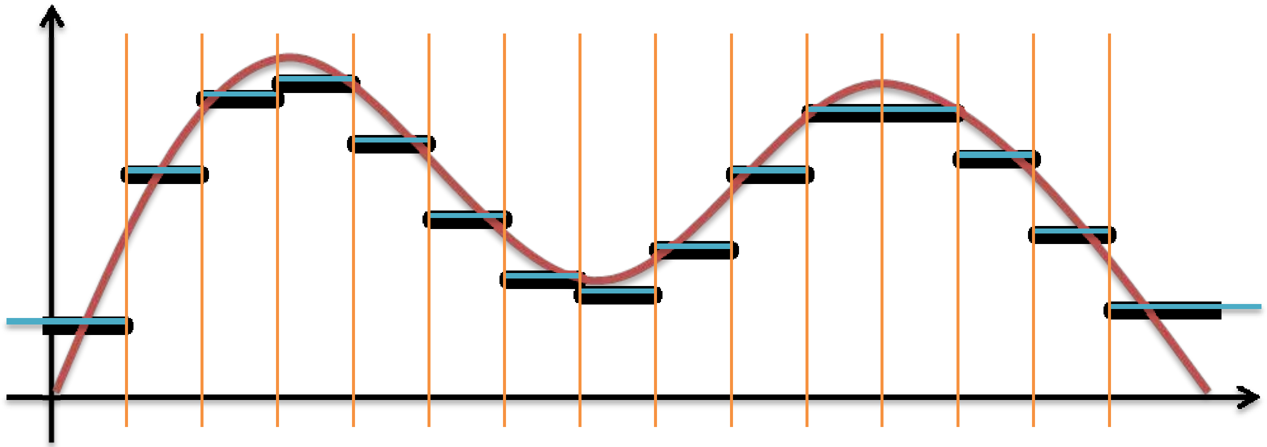
- Now since every partition can be represented by a corresponding

$$\operatorname{sgn}\left\{\sum_i a_j \operatorname{sgn}\left\{\sum_j b_{ij} x_j\right\}\right\}$$

- Therefore whole *F(x)* function can be approximated by the FFNN
- In the following slides a constructive approximation method will be introduced

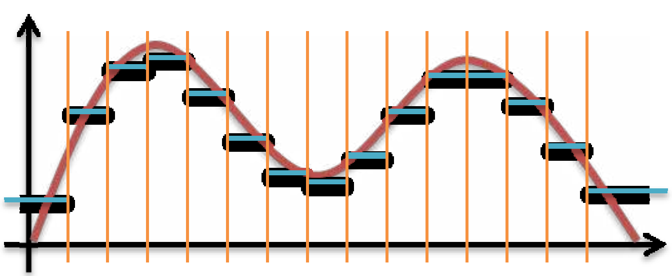
Blum and Li construction

- The Blum and Li construction is based on the „LEGO" principle
- The approximation of the F function is based on its step function
    - Let us have a step function with *n* number of steps



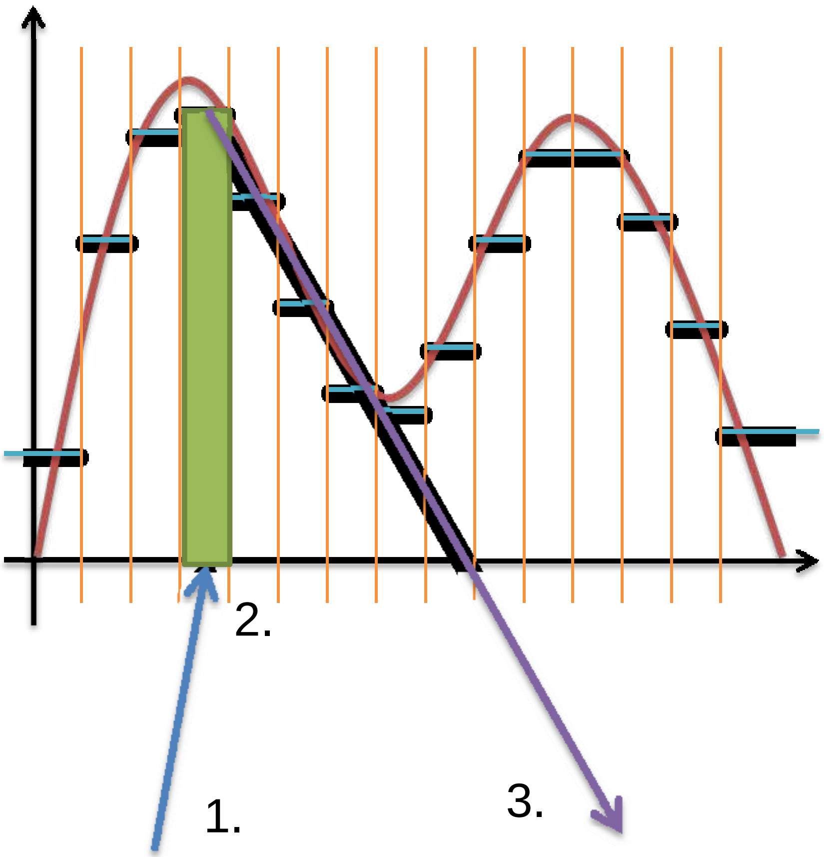
Blum and Li construction

- 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



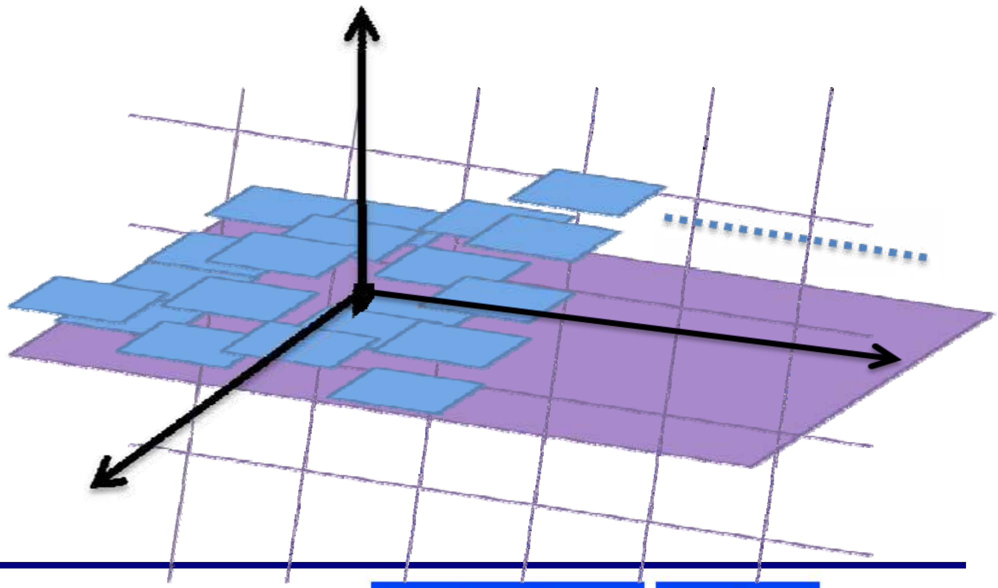
Blum and Li construction

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)



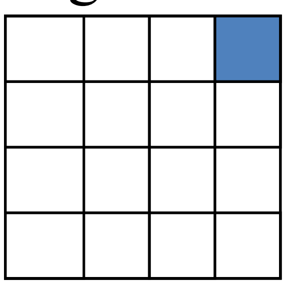
Blum and Li construction

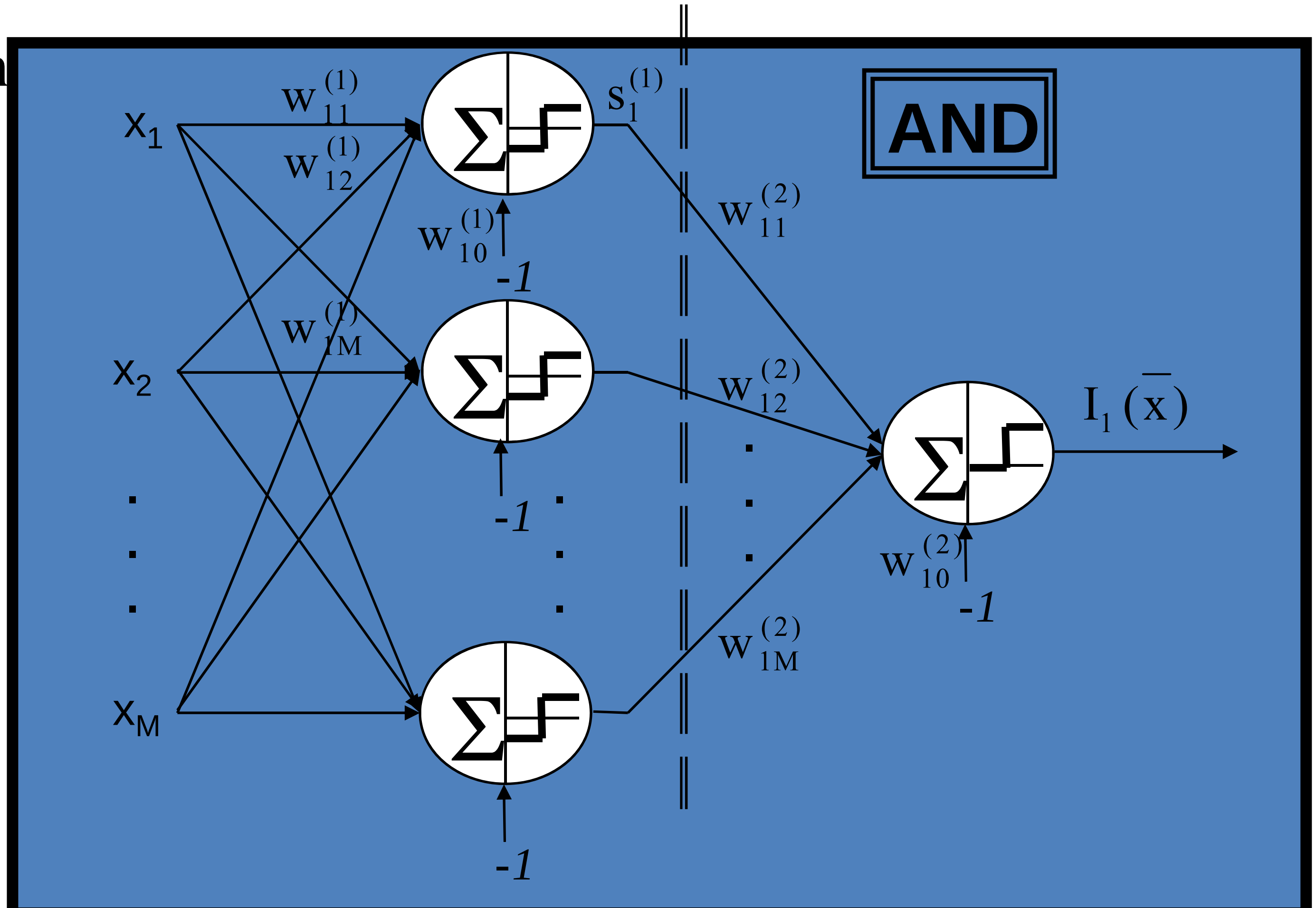
- This construction …
  - … has no dimensional limits
  - … has no equidistance restrictions on tiles (partitions)
  - … can be further fined, and the approximation can be any precise
- 2 dimensional example
  - The tiles are the top of the columns for each approximation cell



Blum and Li construction

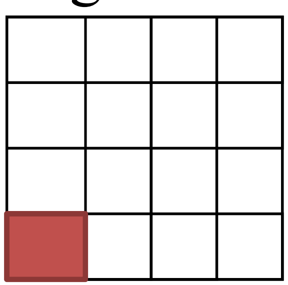
- Construction for one particular region
- The output is I_1 if we are in this region

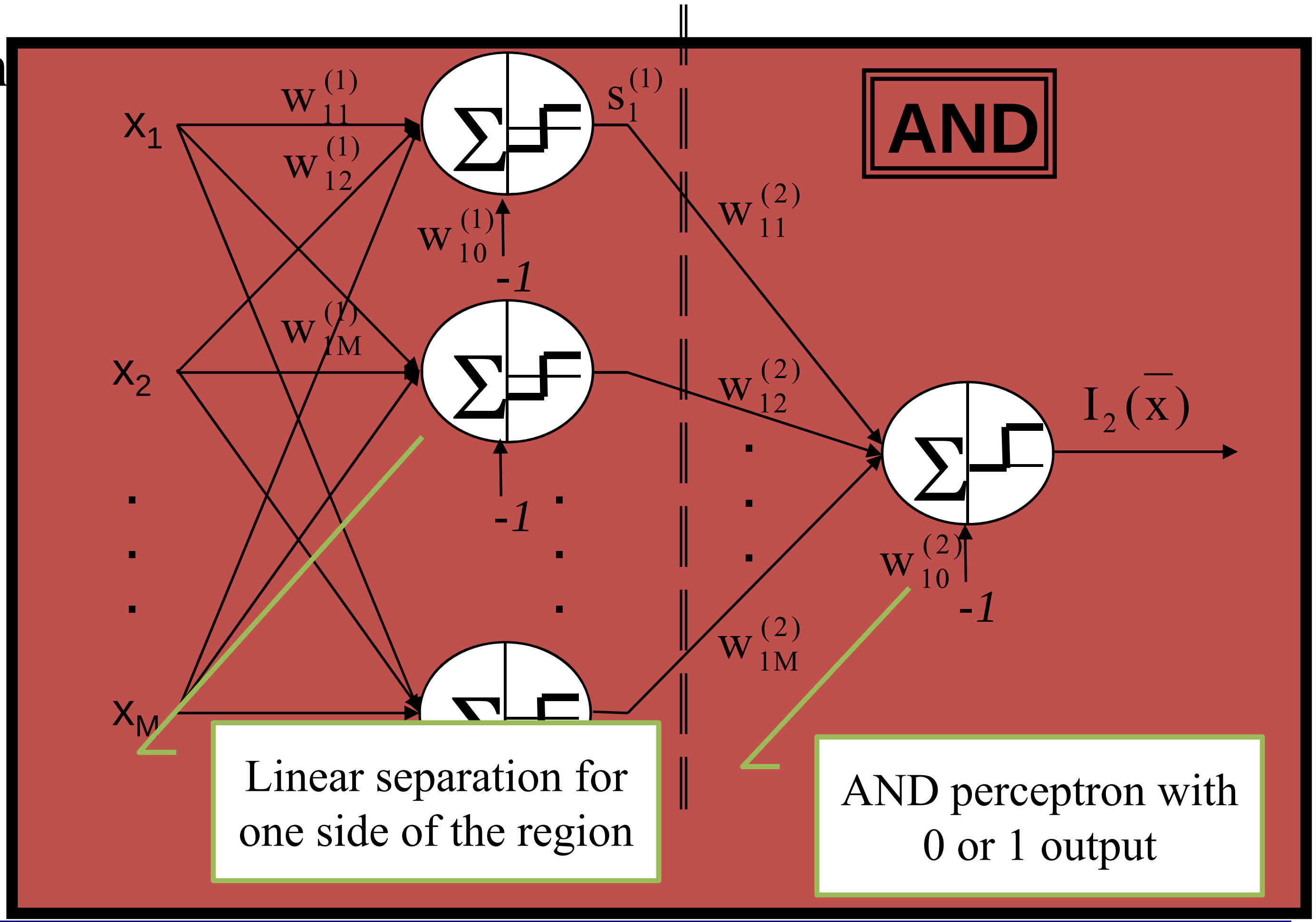




Blum and Li construction

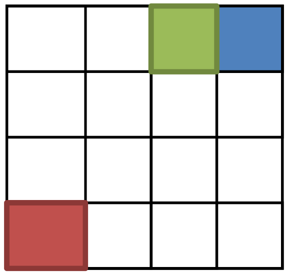
- Construction for one particular region
- The output is I_2 if we are in this region

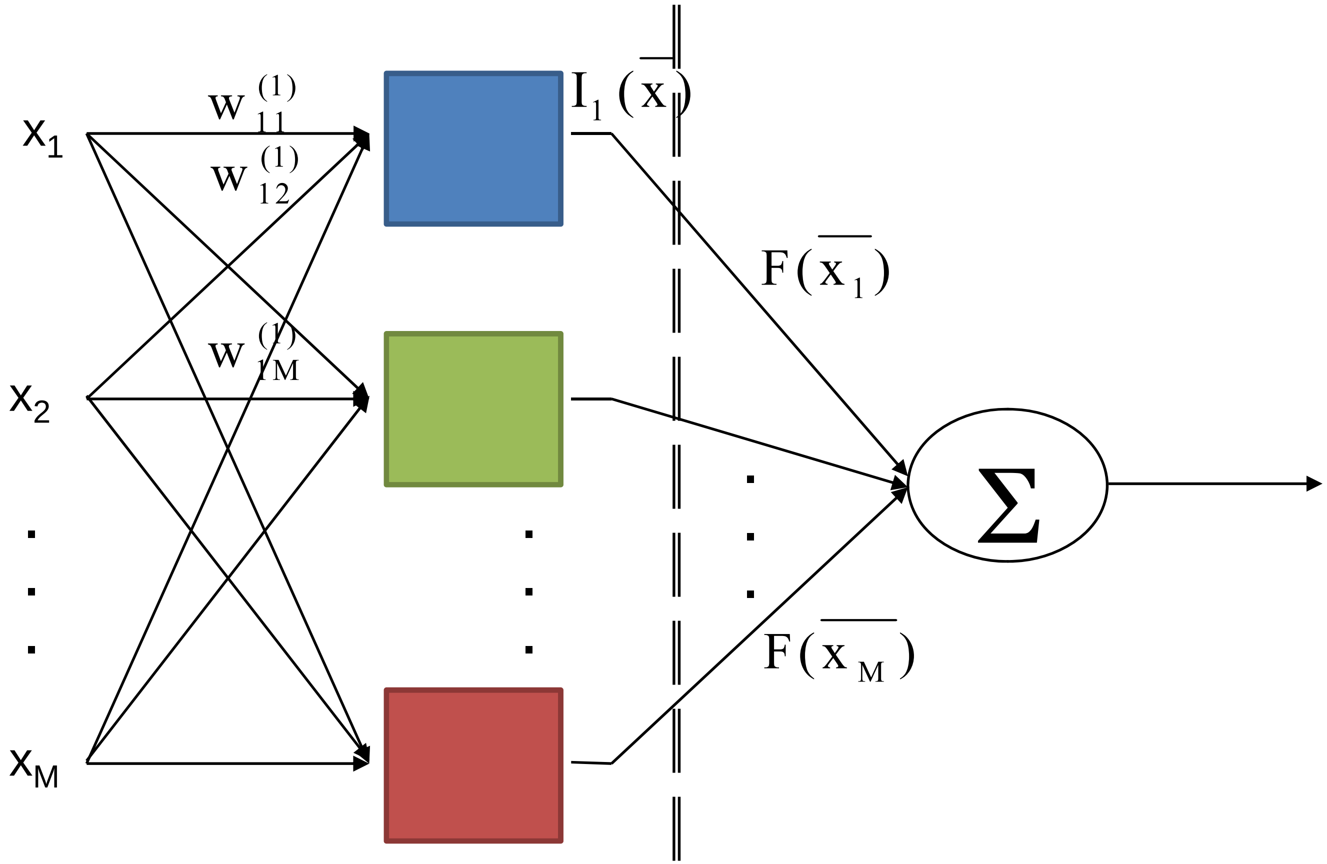




Blum and Li construction

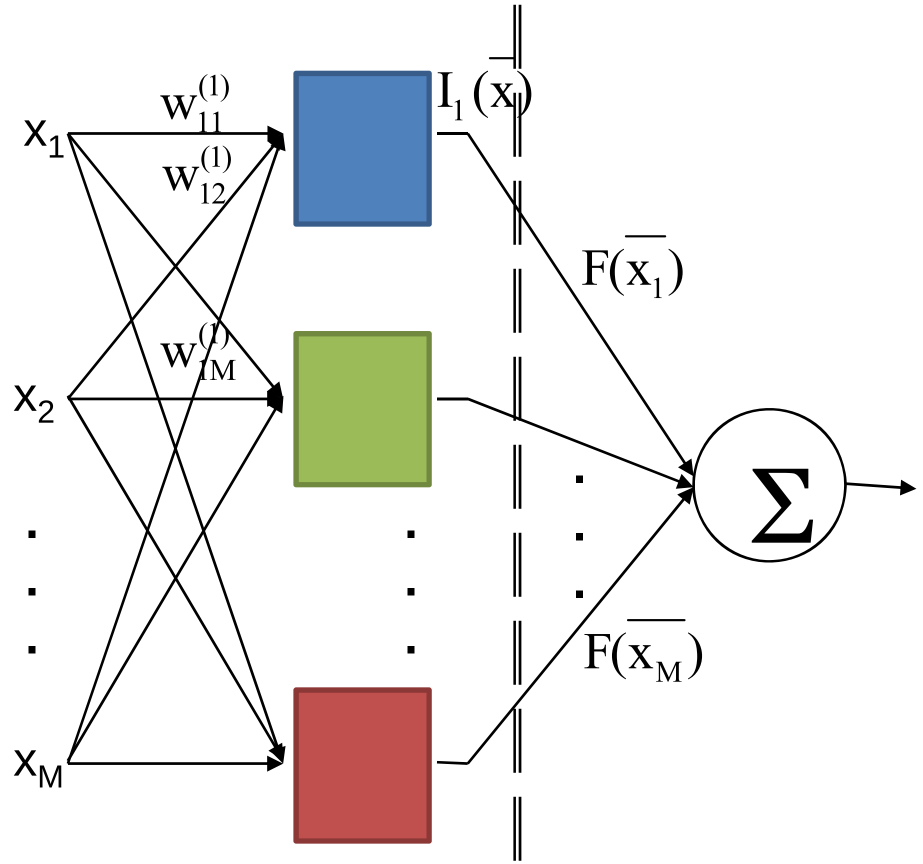
- Each region is being approximated by a block specified above





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
  - The output of blocks marked with different colors is zero or one as the input is in the specified region,
  - Thus the approximation for the whole domain of the original F function is done by FFNN

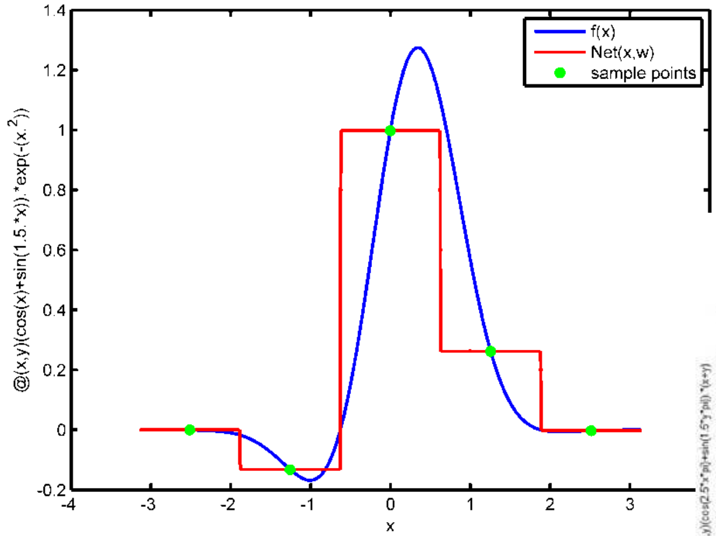


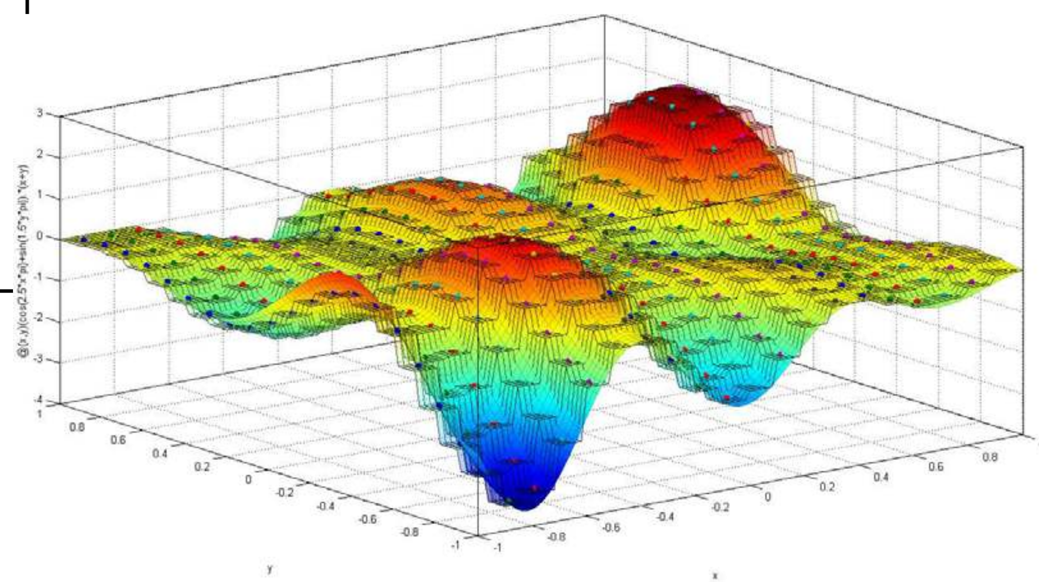
Blum and Li construction

- Minimizing the number of neurons
  - We do not have to represent a hyper plane more than once
  - size of FFNN ~ max||grad F||
- If F has an input, where F is very sensitive, meaning that the changing of F is very fast (the derivative is large), than we have to define the number of regions according to the derivative.

Blum and Li examples

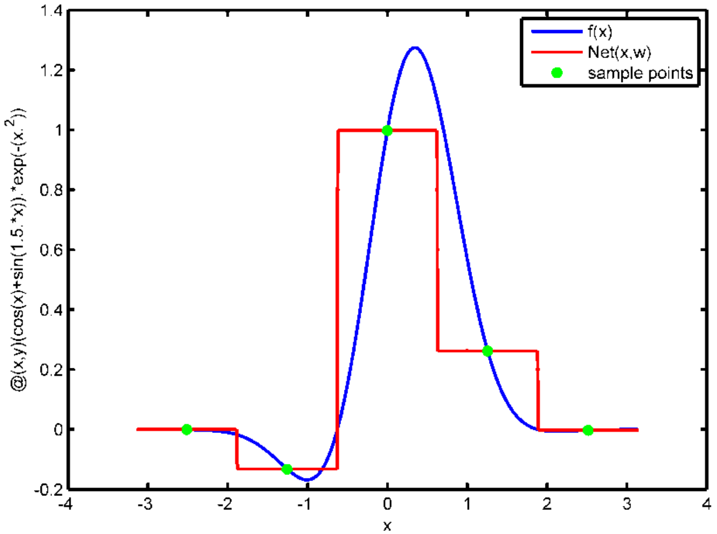
- 2D example and 3D example





Blum and Li examples

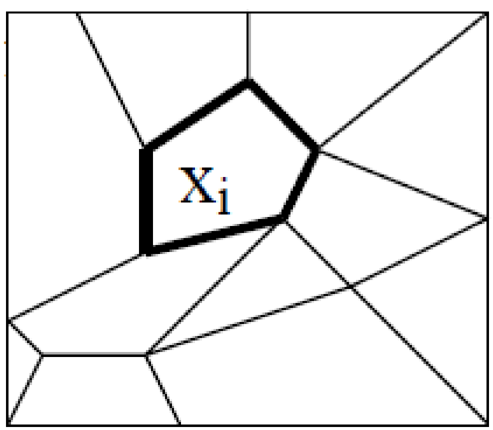
- Weights – separator neurons
  1. [-1,875 -1]
  2. [-1.875 +1], [-0.625 -1]
  3. [-0.625 +1], [0.625 -1]
  4. [0.625 +1], [1.875 -1]
  5. [1.875 +1]

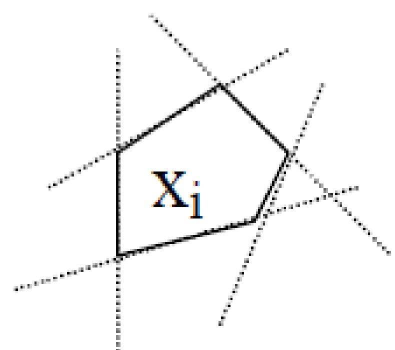


- AND neurons: [0.5 1] or [1.5 1 1]
- Linear neuron in output layer:
  - Weights: [0, -0.18, 1, 0.24, 0.01]

Blum and Li in general

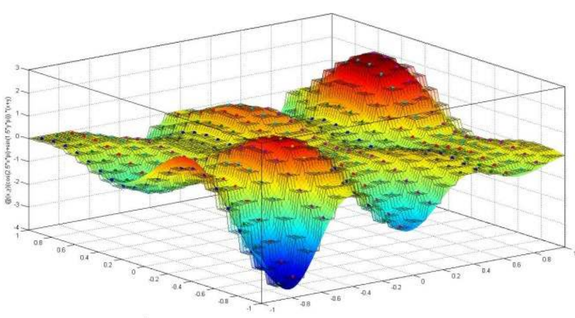
- The partitioning of the domain may be arbitrary
- Let us consider the 2D plane as the domain of the F function
- The following partitioning is possible to be used:





Blum and Li – problems

- The Blum and Li construction is a good approximator as shown previously, but it has its 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
  - Optimal case 3003 neurons are needed
  - (non-optimal: ~4 Million)
  - Smoother approximation needs more
  - We are after to find a less complicated architecture



Learning

- The Blum and Li construction is not always applicable, therefore we seek a solution which trains the neural network for an arbitrary function, then this function can be approximated by the neural network
  - The F function is partially known
  - The F function behaves as a black box
- The task is to find a w which minimize the difference between the F and the network:

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

Learning

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

- This minimization task is not possibly done
  - Complete information is needed about F(x)
- Weak learning in incomplete environment, instead of using F(x)
- A training set is being constructed of observations

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

Learning

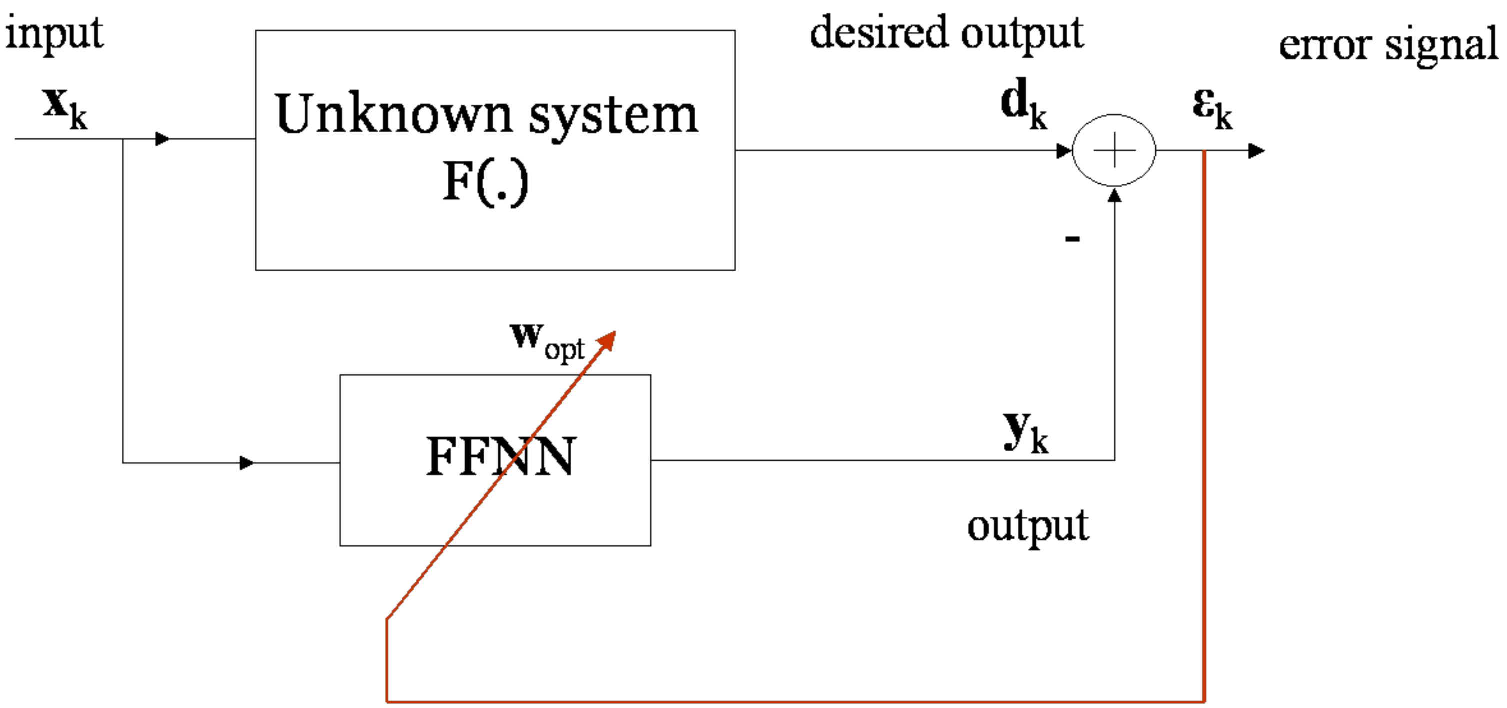
$$\mathbf{w}_{\text{opt}} : \min_{\mathbf{w}} \left\| \text{F}(\mathbf{x}) - \text{Net}(\mathbf{x},\mathbf{w}) \right\|^2 = \min_{\mathbf{w}} \int .. \int \left(\text{F}(\mathbf{x}) - \text{Net}(\mathbf{x},\mathbf{w}) \right)^2 dx_1 \ldots dx_N$$

- The error of the network (the square of difference between the output and the desired output) is minimal
  - The approximation is the best achievable
- We cannot do this due to the limited information on F, instead of we seek:

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

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

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

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

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

Statistical learning theory

- Empirical error

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

- Theoretical error

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

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

Statistical learning theory

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

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

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

Because it is ~ constant due to the uniformity

$$\frac{1}{|X|}\int_X \dots \int \left(\mathrm{F}(\mathbf{x}) - \mathrm{Net}\left(\mathbf{x},\mathbf{w}\right)\right)^2 dx_1 \dots dx_N \sim$$

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

Statistical learning theory

- Therefore

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

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

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

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

- The question is, how to set K to have

$$\left\|\mathbf{w}_{\mathrm{opt}}-\mathbf{w}^{(K)}_{\mathrm{opt}}\right\|\le\epsilon$$

Bias – variance dilemma

- Size of the $\mathcal{NN} \leftrightarrow$ size of training set, K
  - The size of the neural network is the number of weights
  - K is the size of the training set
- Let us investigate the difference:

$$\mathrm{E}\left(d_k - Net\left(\mathbf{x}_k, \mathbf{w}_{opt}^{(k)}\right)\right)^2$$

  - Where $\mathbf{w}_{opt}^{(k)}$ is obtained by minimizing the empirical error R_{emp}

Bias – variance dilemma

- One can write then (adding and subtracting the same term)

$$\mathrm{E}\left(d_k - Net\left(\mathbf{x}_k, \mathbf{w}_{opt}{}^{(k)}\right)\right)^2 =$$

$$= \mathrm{E}\left(d - Net(\mathbf{x}, \mathbf{w}) + Net(\mathbf{x}, \mathbf{w}) - Net\left(\mathbf{x}_k, \mathbf{w}_{opt}{}^{(k)}\right)\right)^2$$

- Therefore

$$= \mathrm{E}\left(d_k - Net\left(\mathbf{x}_k, \mathbf{w}_{opt}\right)\right)^2 + \mathrm{E}\left[Net\left(\mathbf{x}_k, \mathbf{w}_{opt}\right) - Net\left(\mathbf{x}_k, \mathbf{w}_{opt}{}^{(k)}\right)^2\right]$$

  - This expected value should be zero

Bias – variance dilemma

- Remarks
  - The other terms in the expression above become zero
  - The first term in the expression above is the approximation error between $F(\mathbf{x})$ and $Net(\mathbf{x},\mathbf{w})$
  - The second term is the error resulting from the finite training set
  - One can choose between the following options
    - either minimizing the first term (which is referred to as bias) with a relatively large size network, but in this case with a limited size training set the weights cannot be trained correctly by learning, so the second term will be large

Bias – variance dilemma

- Second option
  - minimizing the second term (called variance) which needs small size network. However the size of the training set the should be large, invoking the first term large
- Conclusion
  - there is a dilemma between bias and variance
  - This gives rise to the question, how to set the size of the training set which strikes a good balance between the bias and variance.

$$\underbrace{\mathrm{E}\left(d_k - Net\left(\mathbf{x}_k, \mathbf{w}_{opt}\right)\right)^2}_{BIAS} + \underbrace{\mathrm{E}\left[Net\left(\mathbf{x}_k, \mathbf{w}_{opt}\right) - Net\left(\mathbf{x}_k, \mathbf{w}_{opt}^{(k)}\right)^2\right]}_{VARIANCE}$$

VC dimension

- *Question*: how to set the size of the training set which strikes a good balance between the bias and variance.
- We know the theoretical and empirical error
  The question is, what is the probability of that the difference of these errors are greater than a given constant

$$P\left(\left|R_{th}(\mathbf{w}_{opt}) - R_{emp}\left(\mathbf{w}_{opt}{}^{(k)}\right)\right| \geq \epsilon\right)$$

- Furthermore this probability must be minimized

$$P\left(\left|R_{th}(\mathbf{w}_{opt}) - R_{emp}\left(\mathbf{w}_{opt}{}^{(k)}\right)\right| > \epsilon\right) \leq \Psi(\epsilon, K)$$

VC dimension

- We seek this function $\Psi(\epsilon, K)$
- Replacing the optimal weight vector:

$$P\left(\left|\min_{\mathbf{w}} R_{th}(\mathbf{w}) - \min_{\mathbf{w}} R_{emp}(\mathbf{w})\right| > \epsilon\right) \leq \Psi(\epsilon, K)$$

- To have such result, we have to introduce a more stronger bound on the convergence, called uniform convergence

VC dimension

- Uniform convergence

$$\forall \epsilon > 0, \forall \alpha > 0, \mathbf{w} \in \mathbf{W}$$

$$P\left(\sup_{\mathbf{w} \in W} \left| R_{th}(\mathbf{w}) - R_{emp}(\mathbf{w}) \right| > \epsilon \right) < \alpha$$

- Which enforces that for all other **w**

$$P\left(\left| R_{th}(\mathbf{w}) - R_{emp}(\mathbf{w}) \right| > \epsilon \right) < \alpha$$

VC dimension

- If this uniform convergence holds then the necessary size of learning set can be estimated
- Vapnik and Chervonenkis pioneered the work in revealing such bounds and the basic parameter of this bound is called VC dimension to honor their achievements
- Following slides will discuss this VC dimension

VC dimension

- Let us assume that we are given by a *Net*(**x**,**w**), what we use for binary classification
- VC dimension is related to the classification "power" of *Net*(**x**,**w**).
- More precisely, given the set of dichotomies expanded by *Net*(**x**,**w**) as

$$F := \left\{ \begin{array}{l} Net\left(\mathbf{x},\mathbf{w}\right), \mathbf{w} \in W : Net\left(\mathbf{x},\mathbf{w}\right) = 1 \;\; \text{if}\, x \in X^{(1)} \\ Net\left(\mathbf{x},\mathbf{w}\right) = 0 \;\text{if}\, x \in X^{-(0)} \\ X^{(1)} \bigcup X^{(0)} = X \,, X^{(1)} \bigcap X^{(0)} = 0 \end{array} \right\}$$

VC dimension

- The VC dimension of *Net*(**x**,**w**). is defined as the number of possible dichotomies expressed by *Net*(**x**,**w**)
- For example let us consider the following elementary network *Net*(**x**,**w**)= $\mathrm{sgn}\{\mathbf{w}^T\mathbf{x} - b\}$
  - Its VC dimension is $N+1$
  - If $N = 2$ only $2 + 1 = 3$ points can be separated on a 2D plane.
  - (As we have seen at the investigation of the capacity of one perceptron)

VC dimension

- VC dimension in general
  - Consider the following theoretical and empirical errors, and given relations

$$R_{th}(\mathbf{w}_{opt}{}^{(k)}) \geq R_{th}\left(\mathbf{w}_{opt}\right)$$

$$R_{emp}(\mathbf{w}_{opt}) \geq R_{emp}\left(\mathbf{w}_{opt}{}^{(k)}\right)$$

  - We also know

$$R_{emp}(\mathbf{w}_{opt}) - R_{th}\left(\mathbf{w}_{opt}\right) < \epsilon$$

$$R_{th}(\mathbf{w}_{opt}{}^{(k)}) - R_{emp}\left(\mathbf{w}_{opt}{}^{(k)}\right) < \epsilon$$

VC dimension

- Therefore

$$R_{emp}(\mathbf{w}_{opt}) - R_{th}\left(\mathbf{w}_{opt}\right) \leq R_{th}(\mathbf{w}_{opt}{}^{(k)}) - R_{emp}\left(\mathbf{w}_{opt}{}^{(k)}\right) \leq \epsilon$$

- Vapnik states the following

$$P\left(\sup_{\mathbf{w}\in W}\left|R_{th}(\mathbf{w}) - R_{emp}\left(\mathbf{w}\right)\right| > \epsilon\right) < \left(\frac{2ek}{V_c}\right)^{V_c} e^{-\epsilon^2 K}$$

- Combining

$$P\left(\sup_{\mathbf{w}\in W}\left|R_{th}(\mathbf{w}_{opt}{}^{(k)}) - R_{emp}\left(\mathbf{w}_{opt}{}^{(k)}\right)\right| > 2\epsilon\right) < \left(\frac{2ek}{V_c}\right)^{V_c} e^{-\epsilon^2 K}$$

VC dimension

- VC dimension result

$$P\left(\sup_{\mathbf{w}\in W}\left|R_{th}(\mathbf{w}_{opt}{}^{(k)})-R_{emp}\left(\mathbf{w}_{opt}{}^{(k)}\right)\right|>2\epsilon\right)<\alpha$$

- To set the constant properly

$$\alpha=\left(\frac{2ek}{V_c}\right)^{V_c}e^{-\epsilon^2K}$$

- Therefore the optimal size of training set is driven by the Vc dimension

VC dimension

- Value of the *Vc* parameter
  - If we apply hard nonlinearity in the neural network

$$Vc = O(W \log_2 W)$$

  - If we apply soft nonlinearity

$$Vc = O(W^2)$$

  - Where the W is the number of weights in the neural network

Learning – in practice

- Learning based on the training set:

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

- 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

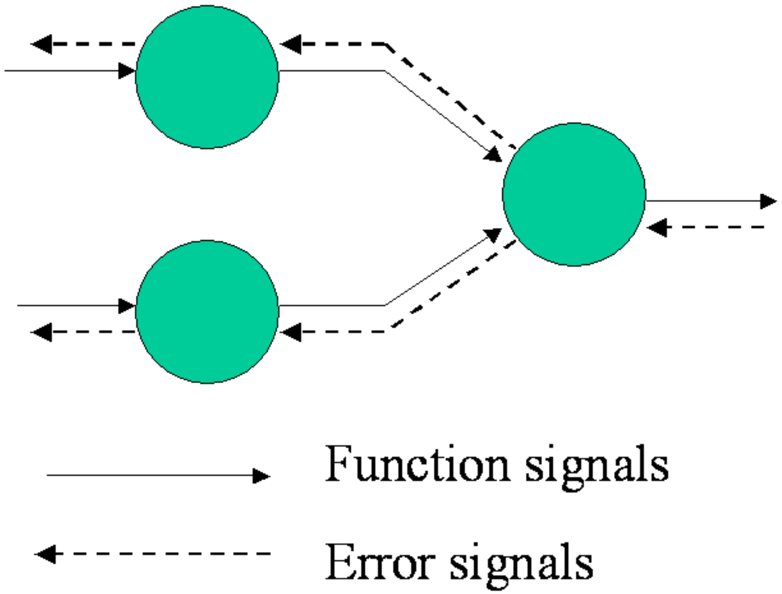
- Newton method

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

- In each step using the learning set we modify the weights of the neurons in layers in order to minimize the error
- To do this the empirical error of the actual neuron is computed and the gradient of this error is used to modify the weight

Learning

- The Rosenblatt algorithm is inapplicable, while we do not know the error and desired output in the hidden layers of the FFNN
- 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

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

Sequential back propagation

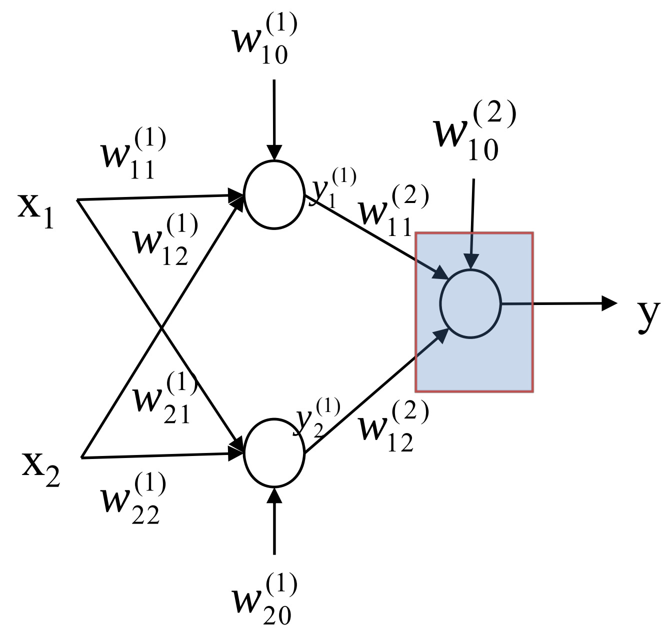
- Consider the following FFNN
  - Error function $E = \left(d(\mathbf{x}) - y(\mathbf{x})\right)^2$
  - Adapting the bias of neuron in hidden layer

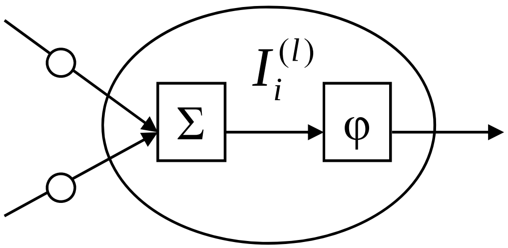
$$\frac{\partial R_{emp}}{\partial w_{10}^{(2)}} = \frac{\partial R_{emp}}{\partial y} \frac{\partial y}{\partial I_1^{(2)}} \frac{\partial I_1^{(2)}}{\partial w_{10}^{(2)}}$$

  - Where the empirical error is

$$\frac{\partial R_{emp}}{\partial y} = -2\left(d(\mathbf{x}) - y(\mathbf{x})\right); \quad \frac{\partial I_1^{(2)}}{\partial w_{10}^{(2)}} = -1$$

$$\frac{\partial y}{\partial I_1^{(2)}} = \frac{\partial \phi\left(I_1^{(2)}\right)}{\partial I_1^{(2)}} = \phi'\left(I_1^{(2)}\right) = ?$$





Sequential back propagation

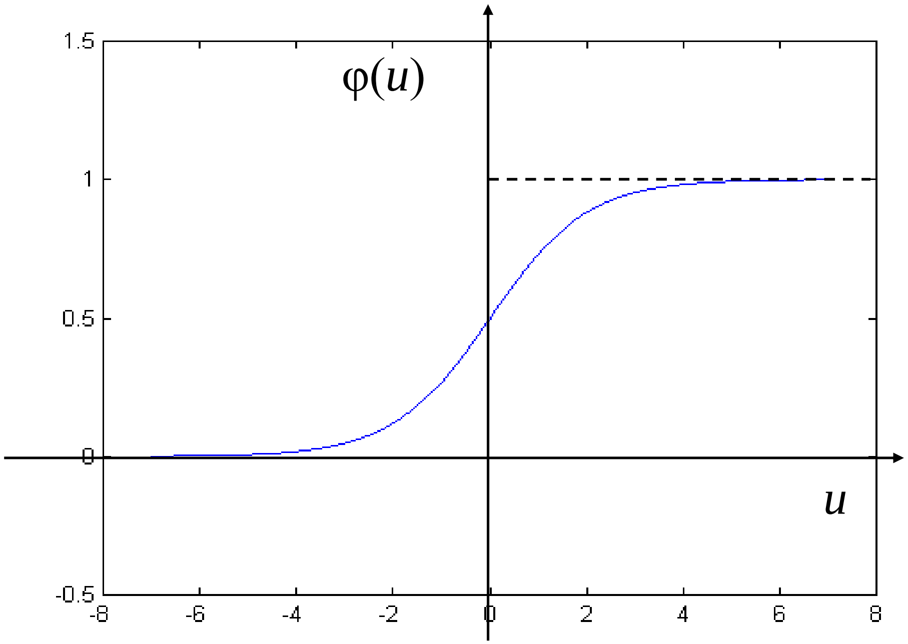
- Activation function $$\phi(u) = \frac{1}{1+e^{-u}}$$
- The derivative of this function

$$\phi'(u) = \frac{\partial}{\partial u}\frac{1}{1+e^{-u}} =$$

$$= \frac{1}{\left(1+e^{-u}\right)^2}\left(e^{-u}\right) =$$

$$= \frac{1}{1+e^{-u}}\frac{e^{-u}}{1+e^{-u}} =$$

$$= \phi(u)\left(1-\phi(u)\right)$$



Sequential back propagation

- Using the previous result of the derivative of activation function

$$\frac{\partial y}{\partial I_1^{(2)}} = \frac{\partial \phi\left(I_1^{(2)}\right)}{\partial I_1^{(2)}} = \phi'\left(I_1^{(2)}\right) = y\left(1-y\right)$$

- Modifying the weight

$$\frac{\partial R_{emp}}{\partial w_{10}^{(2)}} = \frac{\partial R_{emp}}{\partial y}\frac{\partial y}{\partial I_1^{(2)}}\frac{\partial I_1^{(2)}}{\partial w_{10}^{(2)}} = 2\left(d-y\right)y\left(1-y\right)$$

$$\Delta w_{10}^{(2)} = -\eta \frac{\partial R_{emp}}{\partial w_{10}^{(2)}}$$

$$w_{10}^{(2)}(k+1) = w_{10}^{(2)}(k) + \Delta w_{10}^{(2)}(k) = w_{10}^{(2)}(k) - \eta \cdot 2\left(d-y\right)y\left(1-y\right)$$

Sequential back propagation

- Adapting the weights of the neuron in output layer

$$\frac{\partial R_{emp}}{\partial w_{11}^{(2)}} = \frac{\partial R_{emp}}{\partial y} \frac{\partial y}{\partial I_1^{(2)}} \frac{\partial I_1^{(2)}}{\partial w_{11}^{(2)}} = -2(d-y)\,y(1-y)\,y_1^{(1)}$$

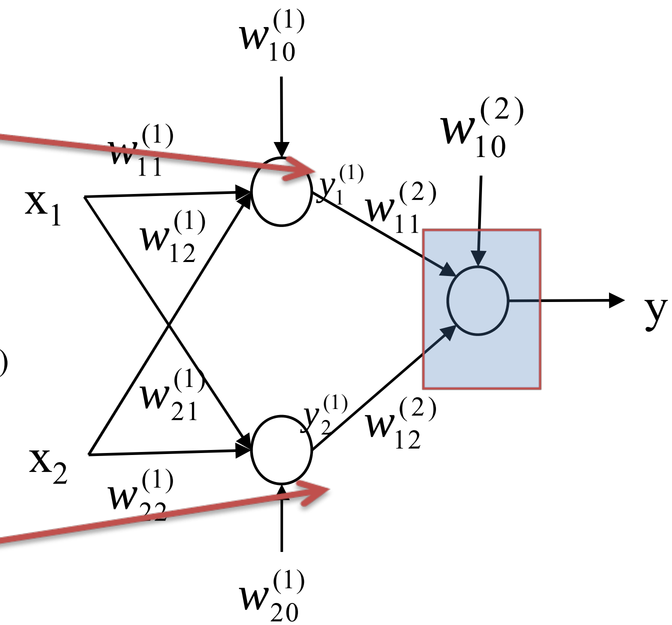
$$w_{11}^{(2)}(k+1) = w_{11}^{(2)}(k) + \Delta w_{11}^{(2)}(k) =$$

$$= w_{11}^{(2)}(k) + \eta \cdot 2(d-y)\,y(1-y)\,y_1^{(1)}$$

$$\frac{\partial R_{emp}}{\partial w_{12}^{(2)}} = \frac{\partial R_{emp}}{\partial y} \frac{\partial y}{\partial I_1^{(2)}} \frac{\partial I_1^{(2)}}{\partial w_{12}^{(2)}} = -2(d-y)\,y(1-y)\,y_2^{(1)}$$

$$w_{12}^{(2)}(k+1) = w_{12}^{(2)}(k) + \Delta w_{12}^{(2)}(k) =$$

$$= w_{12}^{(2)}(k) + \eta \cdot 2(d-y)\,y(1-y)\,y_2^{(1)}$$



Sequential back propagation

- Adapting the weights of the neuron in hidden layer

$$\frac{\partial R_{emp}}{\partial w_{10}^{(1)}} = \frac{\partial R_{emp}}{\partial y}\frac{\partial y}{\partial I_1^{(2)}}\frac{\partial I_1^{(2)}}{\partial y_1^{(1)}}\frac{\partial y_1^{(1)}}{\partial I_1^{(1)}}\frac{\partial I_1^{(1)}}{\partial w_{10}^{(1)}} =$$

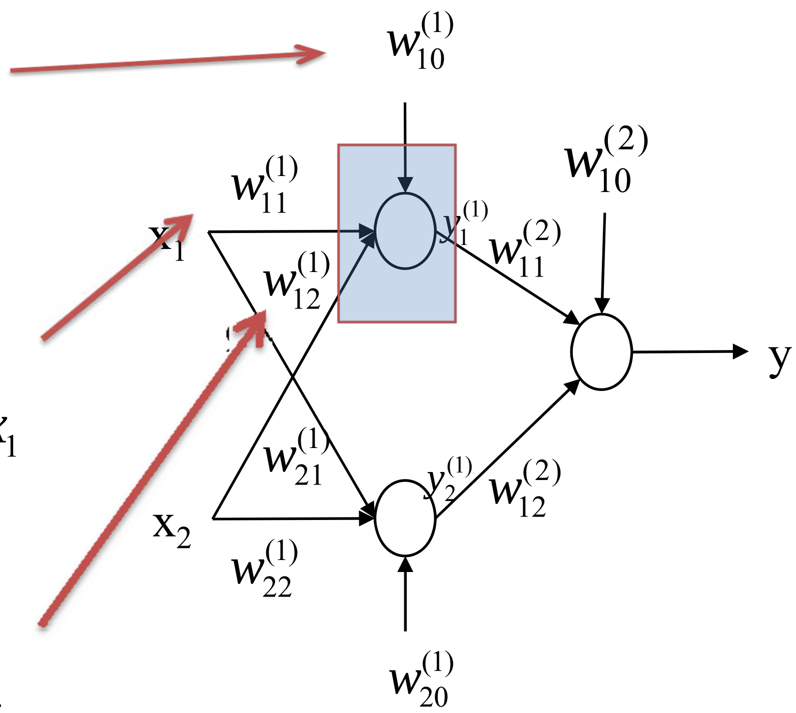
$$= -2(d-y)y(1-y)w_{11}^{(2)}y_1^{(1)}\left(1-y_1^{(1)}\right)\cdot -1$$

$$\frac{\partial R_{emp}}{\partial w_{11}^{(1)}} = \frac{\partial R_{emp}}{\partial y}\frac{\partial y}{\partial I_1^{(2)}}\frac{\partial I_1^{(2)}}{\partial y_1^{(1)}}\frac{\partial y_1^{(1)}}{\partial I_1^{(1)}}\frac{\partial I_1^{(1)}}{\partial w_{11}^{(1)}} =$$

$$= -2(d-y)y(1-y)w_{11}^{(2)}y_1^{(1)}\left(1-y_1^{(1)}\right)\cdot x_1$$

$$\frac{\partial R_{emp}}{\partial w_{12}^{(1)}} = \frac{\partial R_{emp}}{\partial y}\frac{\partial y}{\partial I_1^{(2)}}\frac{\partial I_1^{(2)}}{\partial y_1^{(1)}}\frac{\partial y_1^{(1)}}{\partial I_1^{(1)}}\frac{\partial I_1^{(1)}}{\partial w_{12}^{(1)}} =$$

$$= -2(d-y)y(1-y)w_{11}^{(2)}y_1^{(1)}\left(1-y_1^{(1)}\right)\cdot x_2$$



Sequential back propagation

- Adapting the weights of the neuron in hidden layer

$$\frac{\partial R_{emp}}{\partial w_{20}^{(1)}} = \frac{\partial R_{emp}}{\partial y} \frac{\partial y}{\partial I_1^{(2)}} \frac{\partial I_1^{(2)}}{\partial y_2^{(1)}} \frac{\partial y_2^{(1)}}{\partial I_2^{(1)}} \frac{\partial I_2^{(1)}}{\partial w_{20}^{(1)}} =$$

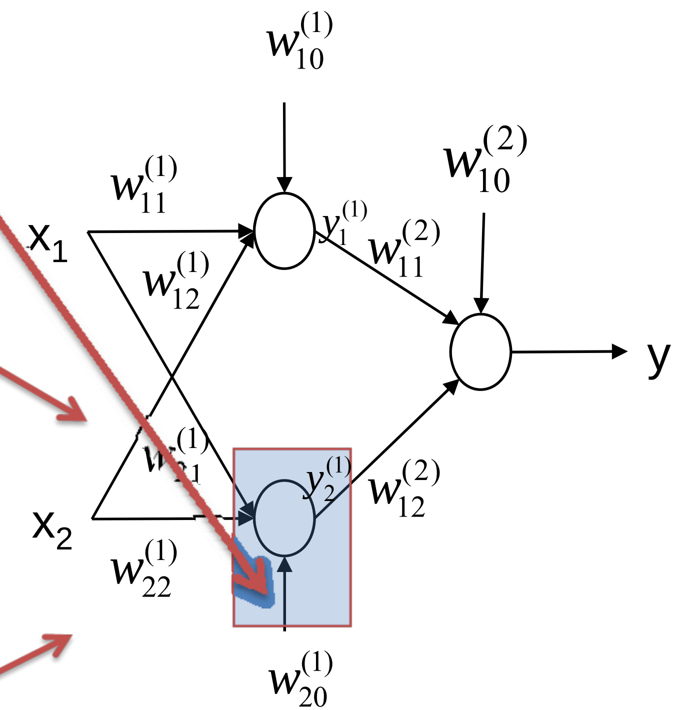
$$= -2(d-y)\,y(1-y)\,w_{21}^{(2)} y_2^{(1)} \left(1-y_2^{(1)}\right)\cdot -1$$

$$\frac{\partial R_{emp}}{\partial w_{21}^{(1)}} = \frac{\partial R_{emp}}{\partial y} \frac{\partial y}{\partial I_1^{(2)}} \frac{\partial I_1^{(2)}}{\partial y_2^{(1)}} \frac{\partial y_2^{(1)}}{\partial I_2^{(1)}} \frac{\partial I_2^{(1)}}{\partial w_{21}^{(1)}} =$$

$$= -2(d-y)\,y(1-y)\,w_{21}^{(2)} y_2^{(1)} \left(1-y_2^{(1)}\right)\cdot x_1$$

$$\frac{\partial R_{emp}}{\partial w_{22}^{(1)}} = \frac{\partial R_{emp}}{\partial y} \frac{\partial y}{\partial I_1^{(2)}} \frac{\partial I_1^{(2)}}{\partial y_2^{(1)}} \frac{\partial y_2^{(1)}}{\partial I_2^{(1)}} \frac{\partial I_2^{(1)}}{\partial w_{22}^{(1)}} =$$

$$= -2(d-y)\,y(1-y)\,w_{21}^{(2)} y_2^{(1)} \left(1-y_2^{(1)}\right)\cdot x_2$$



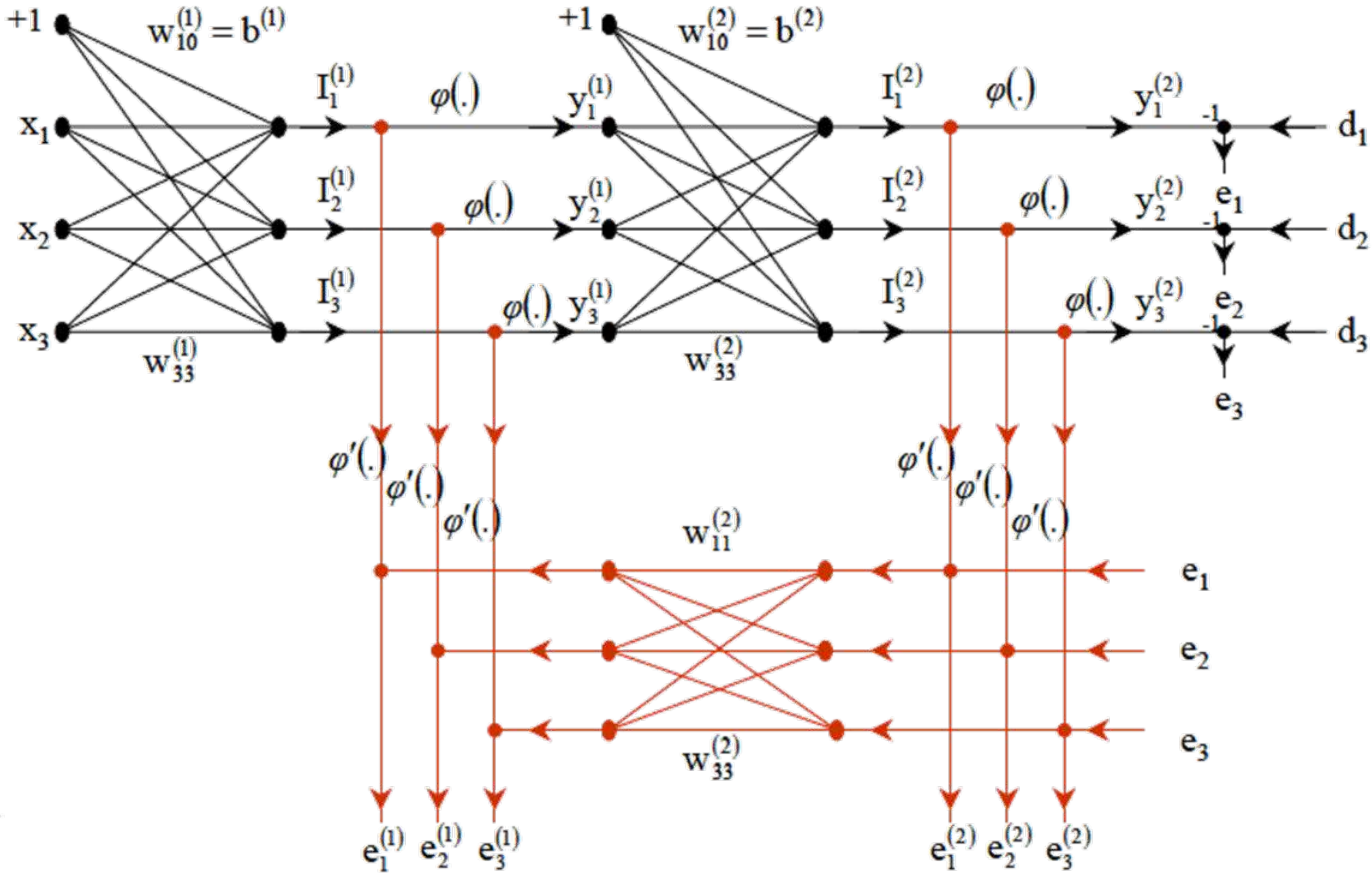
Steps of learning

1. Initialization
   - Setting up the initial **w** weights, usually random numbers
2. Assembling the training set
   - The training set has pairs of inputs and desired outputs
3. Propagating the signal
   - Compute the outputs for all neurons in the network
4. Back propagating the error and updating the weights

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

5. Repeating the 3. and 4. steps for a new sample

Propagation and back propagation



Numerical example – step 1 & 2

- Consider the following problem, initial states:

$\eta=1$

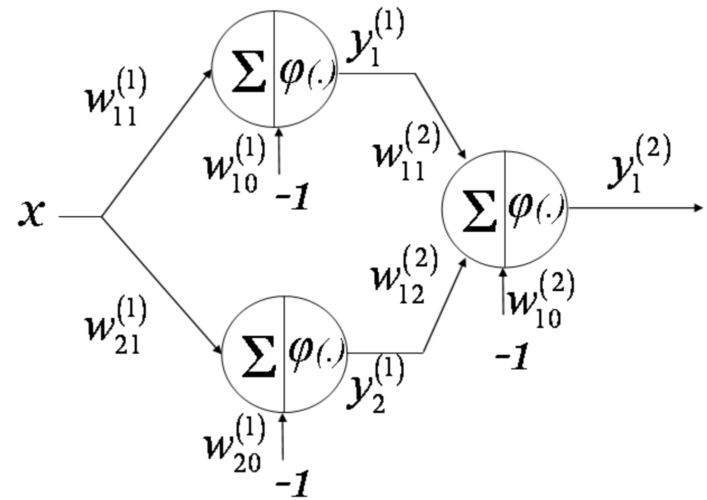
$$w_{11}^{(1)}=-0.3$$

$$w_{10}^{(1)}=w_{20}^{(1)}=w_{10}^{(2)}=0.5$$

$$w_{21}^{(1)}=0.6$$

$$w_{11}^{(2)}=0.5$$

$$w_{12}^{(2)}=0.4$$

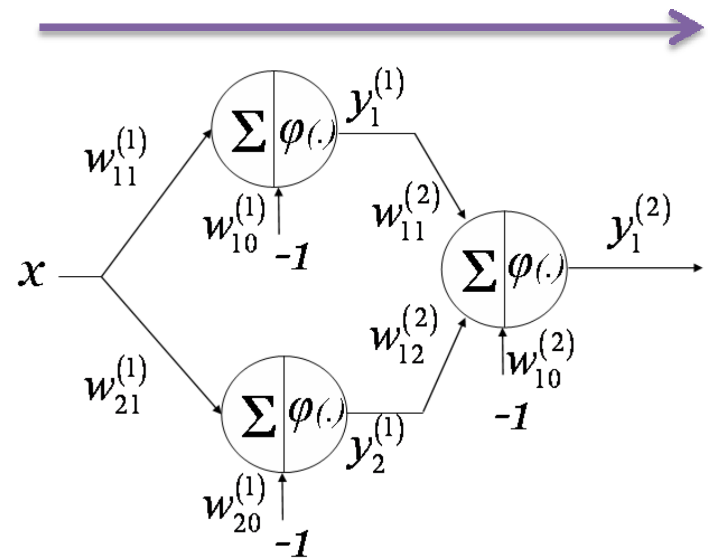


$$\varphi(u)=\frac{1}{1+e^{-\alpha u}},\alpha=2$$

$$\tau^{(3)}=\{(1,0.1),(2,0.5),(3,0.9)\}$$

Numerical example – step 3

- Propagating the signal



k=1

$$\tau^{(3)} = \{(1,0.1),(2,0.5),(3,0.9)\}$$

$$\mathbf{x}_1 = 1 \quad d = 0.1$$

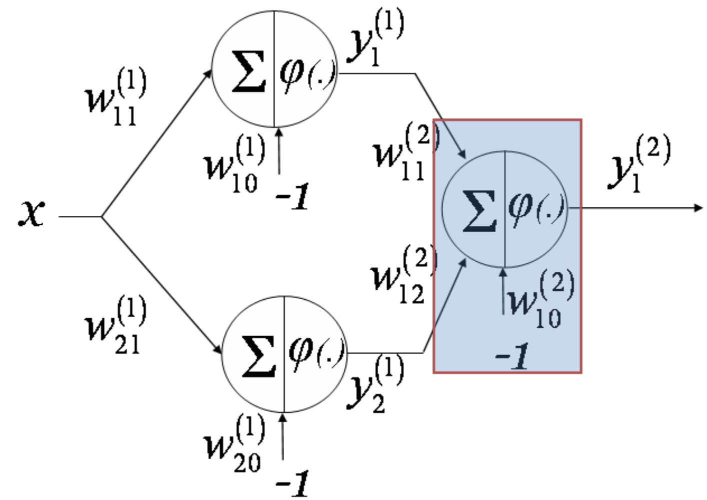
$$y_1^{(1)} = \frac{1}{1+e^{1.6}} = 0.1680$$

$$y_2^{(1)} = \frac{1}{1+e^{-0.2}} = 0.5498$$

$$y_1^{(2)} = \frac{1}{1+e^{-2(0.5\cdot 0.1680+0.4\cdot 0.5498-0.5)}} = 0.4032$$

Numerical example – step 4

- Back propagating, and updating
- Output layer



$$\Delta w_{10}^{(2)}(k) = -\eta \cdot 2(d-y)y(1-y)\alpha$$
$$= -\eta \cdot 2(0.1-0.4032)0.4032(1-0.4032)\cdot 2$$
$$= 0.2918$$

$$\Delta w_{11}^{(2)}(k) = -\eta \cdot -2(d-y)y(1-y)\cdot 2\cdot y_1^{(1)} =$$
$$= \eta \cdot 2(0.1-0.4032)0.4032(1-0.4032)\cdot 2\cdot 0.1680 = -0.0490$$

$$\Delta w_{12}^{(2)}(k) = -\eta \cdot -2(d-y)y(1-y)\cdot 2\cdot y_2^{(1)} =$$
$$= \eta \cdot 2(0.1-0.4032)0.4032(1-0.4032)\cdot 2\cdot 0.5498 = -0.1604$$

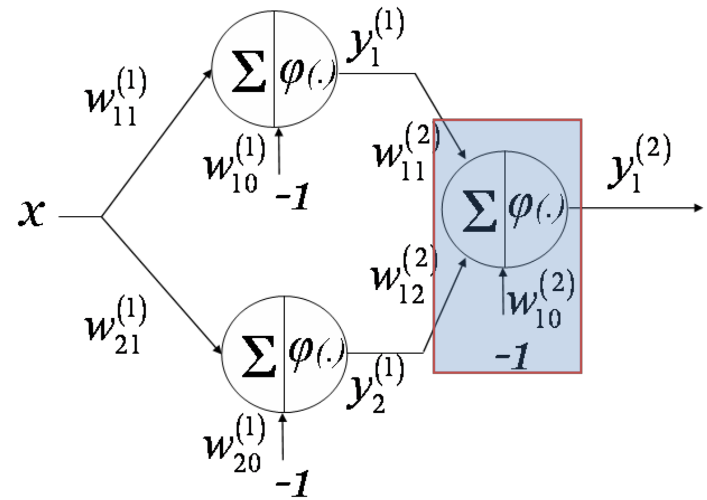
Numerical example – step 4

- Back propagating, and updating
- Output layer - Updating

$$w_{10}^{(2)}(1) = w_{10}^{(2)}(0) + \Delta w_{10}^{(2)}(0) =$$
$$= 0.5 + 0.2918 = 0.7918$$

$$w_{11}^{(2)}(1) = w_{11}^{(2)}(0) + \Delta w_{11}^{(2)}(0) =$$
$$= 0.5 - 0.0490 = 0.4510$$

$$w_{12}^{(2)}(1) = w_{12}^{(2)}(0) + \Delta w_{12}^{(2)}(0) =$$
$$= 0.4 - 0.1604 = 0.2396$$



Numerical example – step 4

$$\Delta w_{10}^{(1)}(k) = -\eta - 2(d-y)y \cdot 2(1-y)w_{11}^{(2)}y_1^{(1)} \cdot 2\left(1-y_1^{(1)}\right) \cdot -1 =$$

$$-\eta \cdot 2(0.1-0.4032)0.4032(1-0.4032)0.5 \cdot 0.1680(1-0.1680) \cdot 4 = 0.0408$$

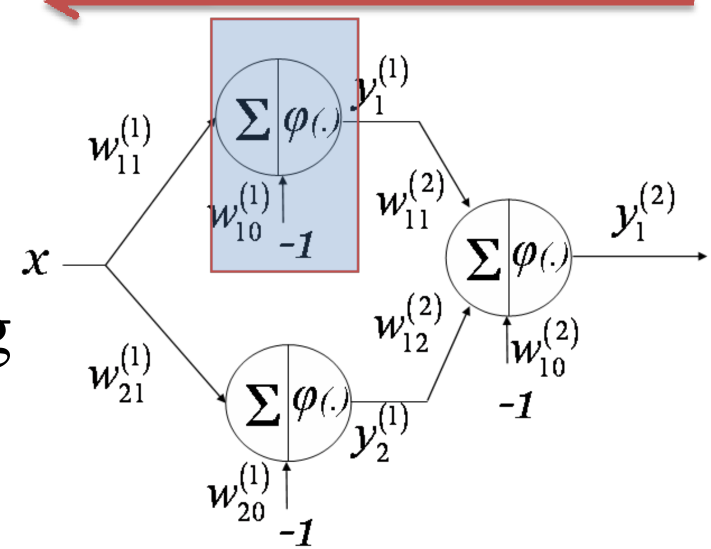
$$\Delta w_{11}^{(1)}(k) = -\eta - 2(d-y)y \cdot 2(1-y)w_{11}^{(2)}y_1^{(1)} \cdot 2\left(1-y_1^{(1)}\right) \cdot x =$$

$$= \eta \cdot 2(0.1-0.4032)0.4032(1-0.4032)0.5 \cdot 0.1680(1-0.1680) \cdot 1 \cdot 4 = -0.0408$$

$$w_{10}^{(1)}(1) = 0.5 + 0.0408 = 0.5408$$

$$w_{11}^{(1)}(1) = -0.3 - 0.0408 = -0.3408$$

- Back propagating, and updating
- Hidden layer - Updating



Numerical example – step 4

$$\Delta w_{20}^{(1)}(k) = -\eta - 2(d-y)\,y\cdot 2(1-y)\,w_{12}^{(2)} y_2^{(1)}\cdot 2\left(1-y_2^{(1)}\right)\cdot -1 =$$

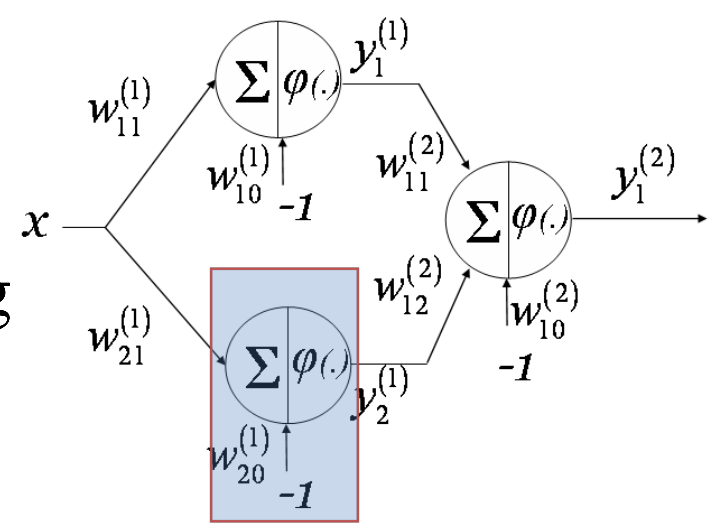
$$= -\eta\cdot 2(0.1-0.4032)0.4032(1-0.4032)0.4\cdot 0.5498(1-0.5498)\cdot 4 = 0.0576$$

$$\Delta w_{21}^{(1)}(k) = -\eta - 2(d-y)\,y\cdot 2(1-y)\,w_{12}^{(2)} y_2^{(1)}\cdot 2\left(1-y_2^{(1)}\right)\cdot x =$$

$$= \eta\cdot 2(0.1-0.4032)0.4032(1-0.4032)0.4\cdot 0.5498(1-0.5498)\cdot 1\cdot 4 = -0.0576$$

$$w_{20}^{(1)}(1) = 0.5 + 0.0576 = 0.5576$$

$$w_{21}^{(1)}(1) = 0.6 - 0.0576 = 0.5424$$



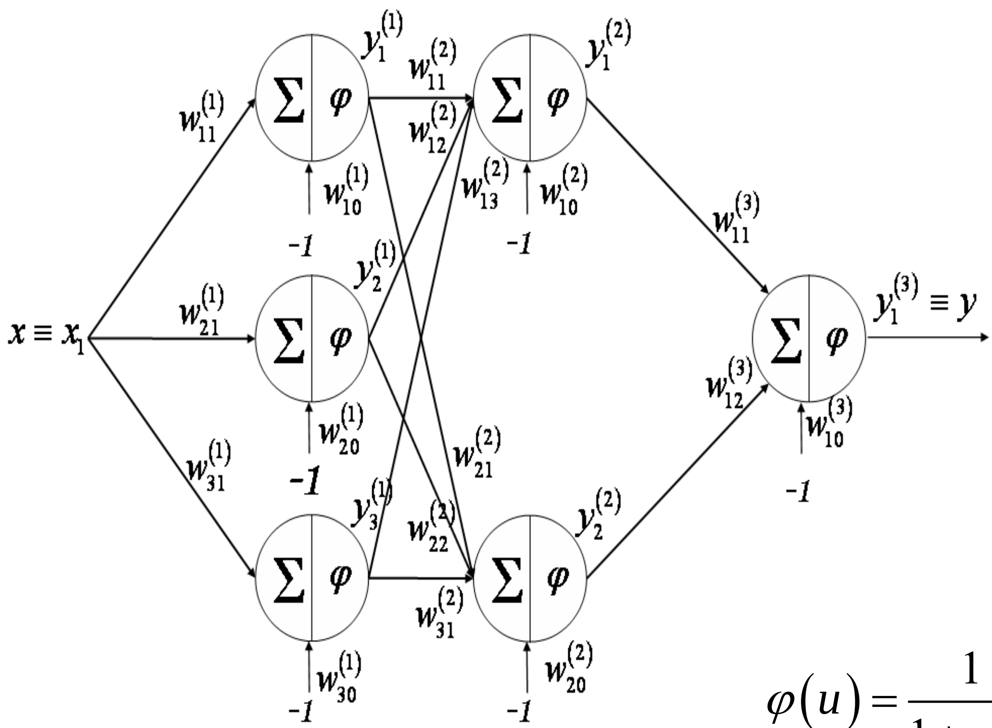
- Back propagating, and updating
- Hidden layer - Updating

Numerical example – step 4

- This must be repeated for the other samples in the training set, until a pre-defined stopping criteria is reached.
- This criteria can be
  - A limit of steps
  - A pre-defined level of empirical error
  - When the weight does not change
  - …

Example of errors

- Consider the following problem, initial states:



$w_{10}^{(3)}(0) = -1$ $\quad w_{10}^{(2)}(0) = 0$

$w_{11}^{(3)}(0) = 1$ $\quad w_{11}^{(2)}(0) = 1$

$w_{12}^{(3)}(0) = 1$ $\quad w_{12}^{(2)}(0) = -0.5$

$w_{10}^{(1)}(0) = 1$ $\quad w_{13}^{(2)}(0) = 0.5$

$w_{20}^{(1)}(0) = 0$ $\quad w_{20}^{(2)}(0) = 2$

$w_{30}^{(1)}(0) = -1$ $\quad w_{21}^{(2)}(0) = 1$

$w_{11}^{(1)}(0) = 1$ $\quad w_{22}^{(2)}(0) = 0.5$

$w_{21}^{(1)}(0) = 1$ $\quad w_{23}^{(2)}(0) = -2$

$w_{31}^{(1)}(0) = 1$

$$\varphi(u) = \frac{1}{1+e^{-u}} \qquad \tau^{(3)} = \{(-1, 0.5), (0, 0.3), (1, 1)\}$$

Example of errors

- The structure of back propagated errors:

$$e_3^{(1)} = \left[\underbrace{w_{13}^{(2)} w_{11}^{(3)} \underbrace{\left(d - y_1^{(3)}\right) y_1^{(3)} \left(1 - y_1^{(3)}\right)}_{e_1^{(3)}} y_1^{(2)} \left(1 - y_1^{(2)}\right)}_{e_1^{(2)}} + \underbrace{w_{23}^{(2)} w_{12}^{(3)} \underbrace{\left(d - y_1^{(3)}\right) y_1^{(3)} \left(1 - y_1^{(3)}\right)}_{e_1^{(3)}} y_2^{(2)} \left(1 - y_2^{(2)}\right)}_{e_2^{(2)}} \right] y_3^{(1)} \left(1 - y_3^{(1)}\right)$$

- Simplified

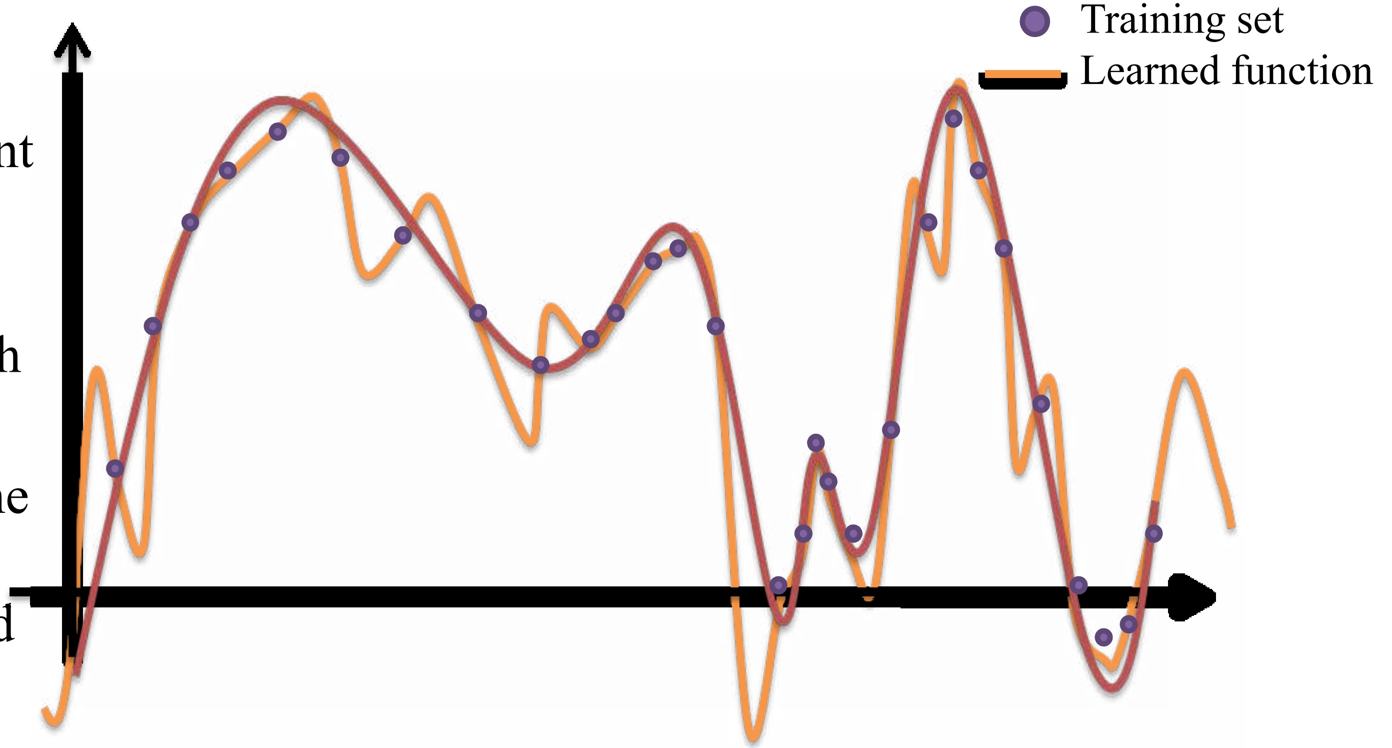
$$e_3^{(1)} = \left[\underbrace{w_{13}^{(2)} w_{11}^{(3)} \underbrace{\left(d - y_1^{(3)}\right) 1}_{e_1^{(3)}} y_1^{(2)} \left(1 - y_1^{(2)}\right)}_{e_1^{(2)}} + \underbrace{w_{23}^{(2)} w_{12}^{(3)} \underbrace{\left(d - y_1^{(3)}\right) 1}_{e_1^{(3)}} y_2^{(2)} \left(1 - y_2^{(2)}\right)}_{e_2^{(2)}} \right] y_3^{(1)} \left(1 - y_3^{(1)}\right)$$

Learning issues

- The speed of learning or the quality of approximation is not always the best reachable
  - It is possible to improve the result with other (better) w weights or other neural structure
- The *Vc* dimension must be considered when the size of the network and the training set is planned
  - It is very possible that the FFNN is being over trained
  - On the elements of the training set the output of the network is errorless, but on other inputs the error is huge
  - Consider the following figure

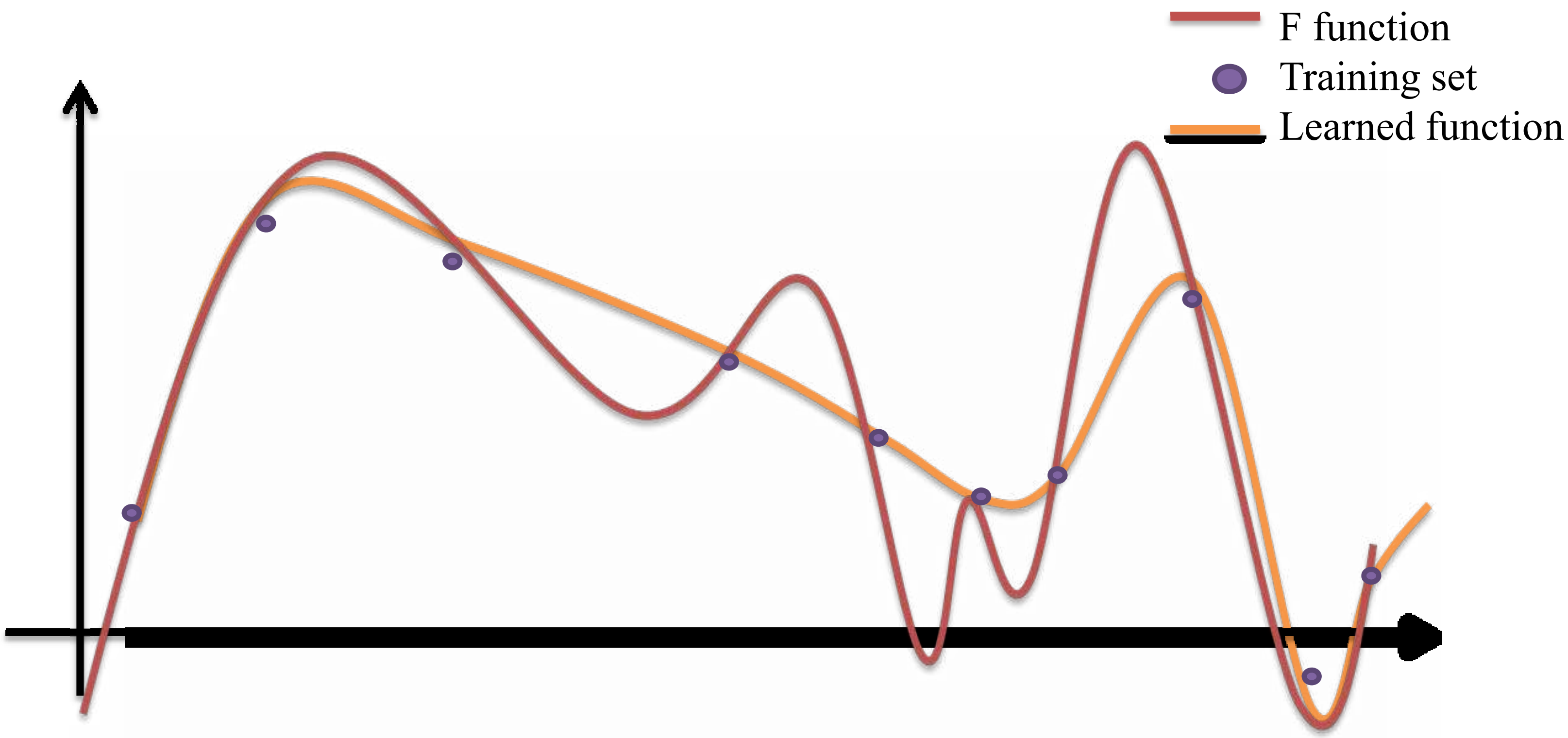
Learning issues

- Example: over trained network
  - The error of the network in case where the input is a training point is almost zero.
  - In other cases the error is much bigger.
  - Therefore not the goal F function has been learned by the network.



Learning issues

- Example: under trained network



Improvements of learning

- Preprocessing the input and post processing the output
  - Normalizing
  - Altering the statistical properties of the input
    - Type of distribution
    - Range of data → mapping
- Use of different nonlinearity (even linearity)
  - Using different activation functions in different layers
- Use of different learning parameters

Improvements of learning

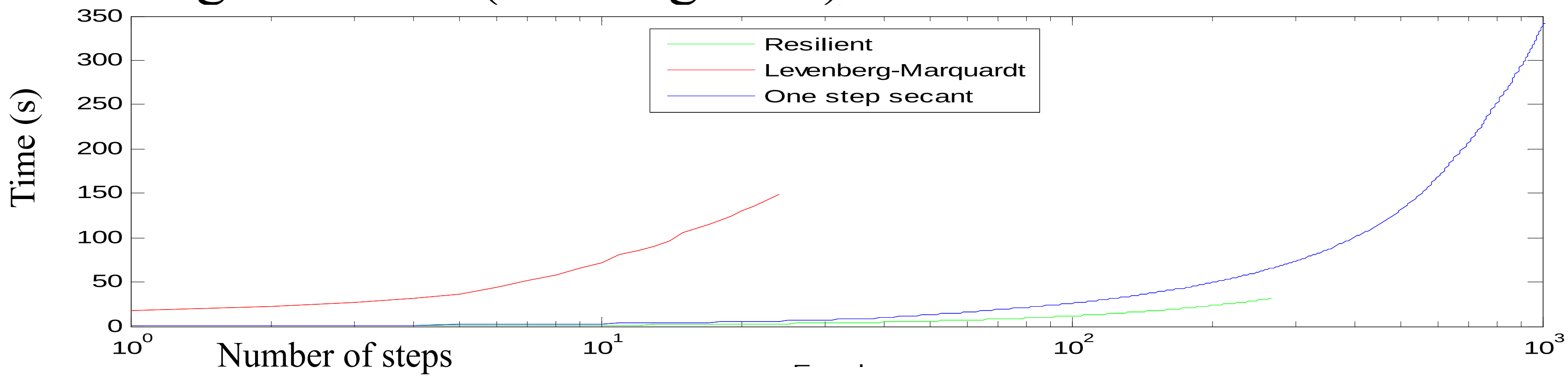
- Altering the initialization method
  - Not to use random numbers during initialization
- Improved version of learning algorithms
  - Resilient Back propagation
  - Levenberg Marquadt algorithm
  - Momentum methods
- Partition the training set into
  - Learning set – to train the network
  - Validation set – to validate the learned weights
  - Testing set – to evaluate the FFNN

Comparison of two learning methods

- Resilient back propagation rule

$$\Delta w_{ij}(n) = \begin{cases} -\Delta_{ij}(n) & \text{if } \dfrac{\delta E(n)}{\delta w_{ij}} > 0 \\ +\Delta_{ij}(n) & \text{if } \dfrac{\delta E(n)}{\delta w_{ij}} < 0 \\ 0 & \text{else} \end{cases} \qquad \Delta_{ij}(n) = \begin{cases} \eta^{+} \cdot \Delta_{ij}(n) & \text{if } \dfrac{\delta E(n-1)}{\delta w_{ij}} \dfrac{\delta E(n)}{\delta w_{ij}} > 0 \\ \eta^{-} \cdot \Delta_{ij}(n) & \text{if } \dfrac{\delta E(n-1)}{\delta w_{ij}} \dfrac{\delta E(n)}{\delta w_{ij}} < 0 \\ \Delta_{ij}(n-1) & \text{else} \end{cases}$$

- Convergence time (learning time)



Applications of FFNN - Introducing

- Pattern (character) recognition
  - Given: samples and indices
  - Input: noisy sample
  - Output: index of stored sample
- Time series prediction
  - FFNN is able to predict the new value of time series when historical data is available and the FFNN is trained on the historical data
  - Example: power consumption, currency exchange rates

Applications of FFNN - Introducing

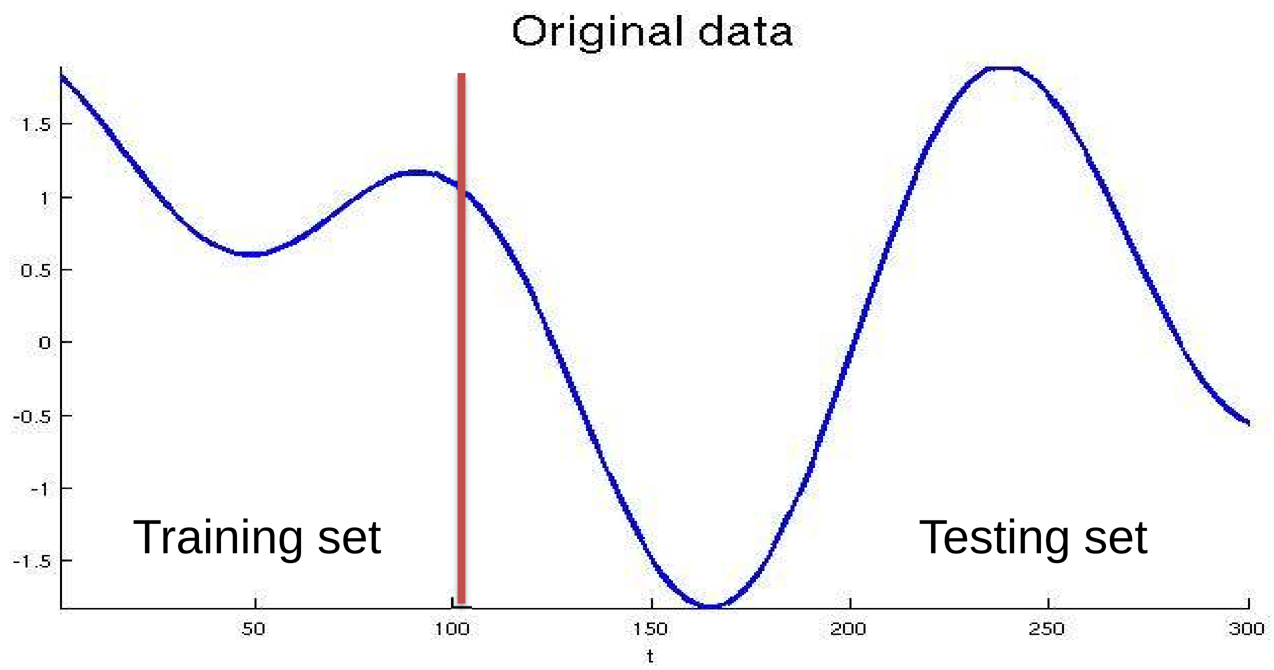
- Telecommunication
  - Signal detection task
  - Given channel and noisy symbols arrived through this channel
  - The task is to decide what symbol has been sent over the channel
- Call admission control
  - In packet switched networks
  - To provide maximal throughput and avoid overflow of the network

Applications of FFNN – Time series prediction

- The task is the following:
  - We know the history of a time series till the current time instant
  - We would like to estimate the next few element of this time series
- In order to solve this task using the FFNN a training set must be assembled
  - This training set contains a n length vector containing the values from i to $i+n$ from the time series as input and the $i+n+1$ of the time series as desired output
  - Running i from 1 to N-n-1, where N is the length of the time series the training set is constructed easily

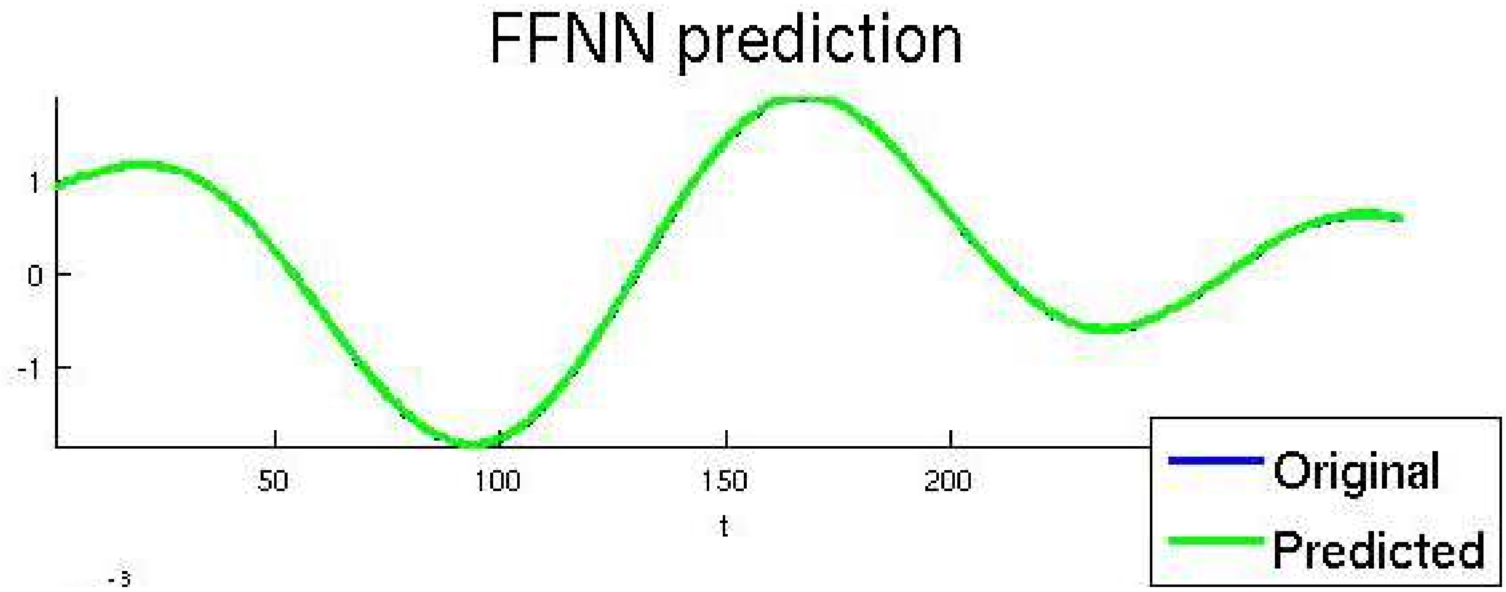
Applications of FFNN – Time series prediction

- For example take the following simple function as the time series



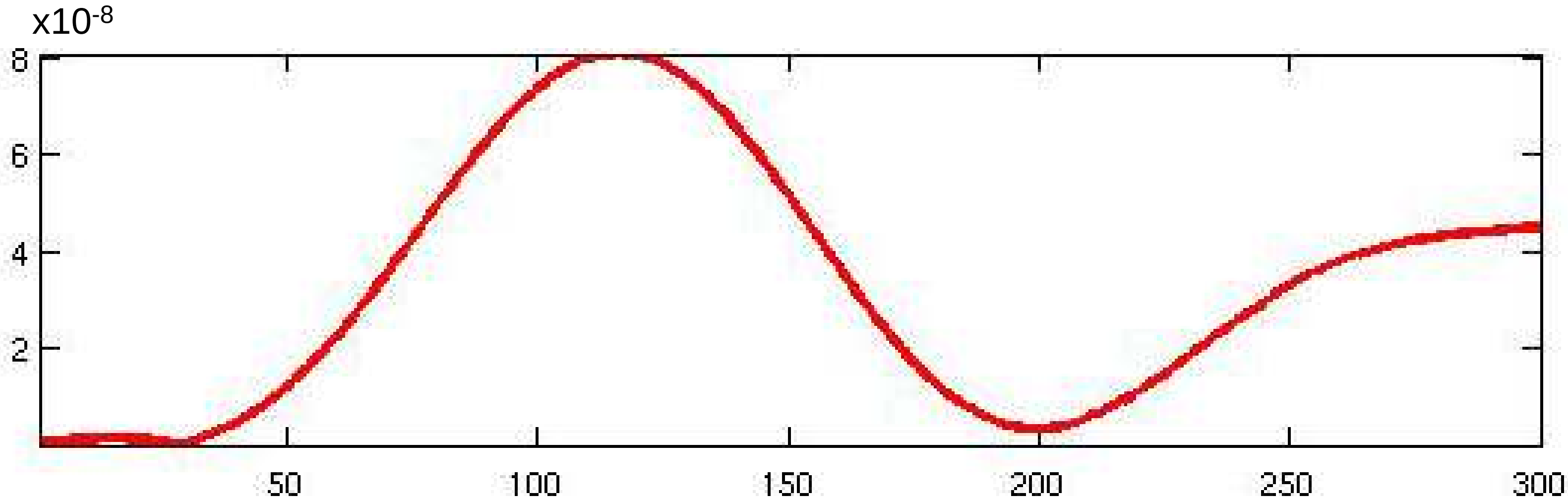
Applications of FFNN – Time series prediction

- The predicted time series
  - The precision of prediction is very high



Applications of FFNN – Time series prediction

- Error of prediction and real time series
  - This function was learned by the FFNN
  - The information if the training set was generalized and the future values of the time series was predicted well by the FFNN

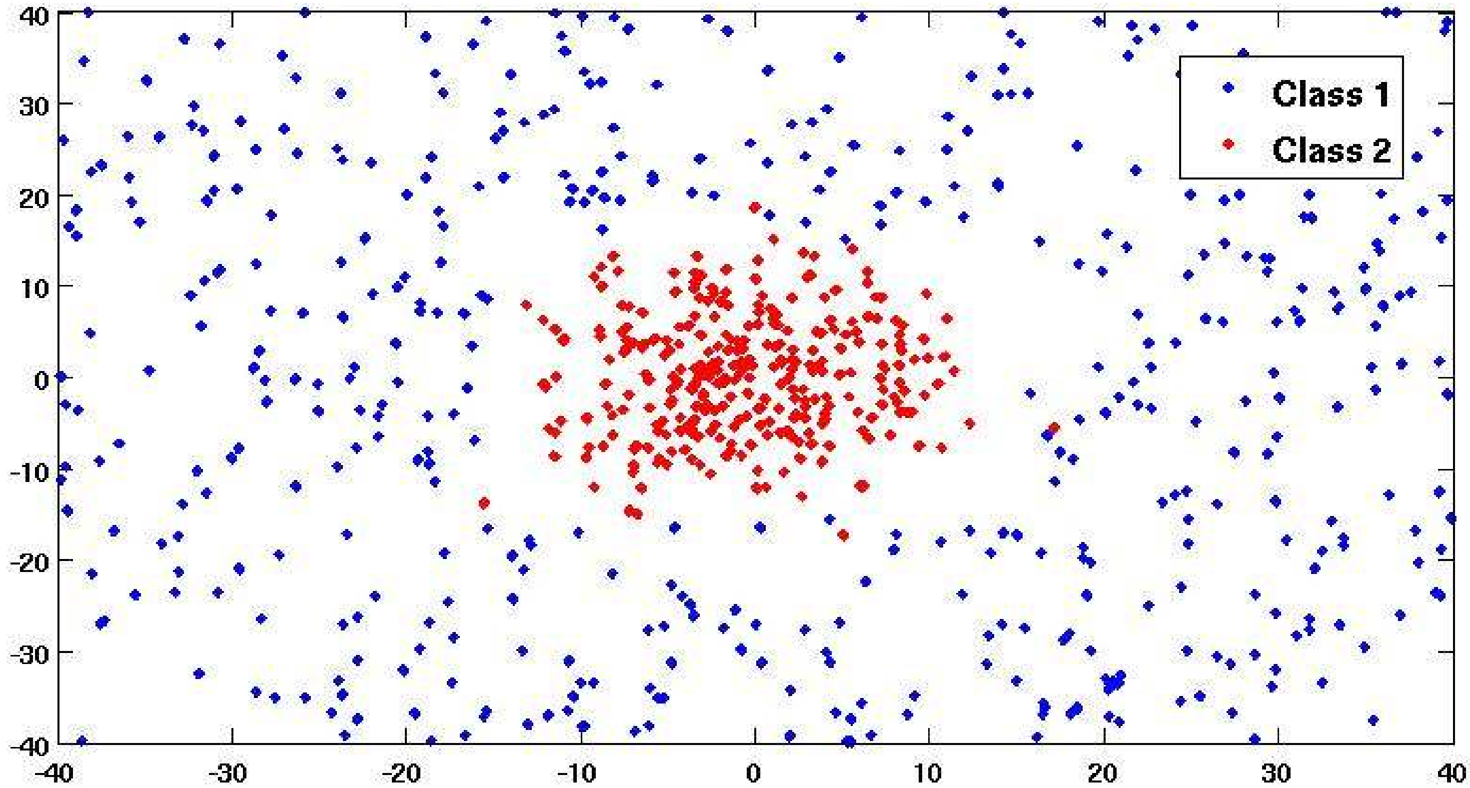


Applications of FFNN – Classification

- The classification example task is the following:
  - Classification with two classes
  - A data set is given with vectors, the two classes are not defined explicitly
  - The information which vector belongs to the first class and which vector belongs to the second class is available
  - The training set is constructed from the previous information
    - Vector as input and +1 or –1 as the classification data

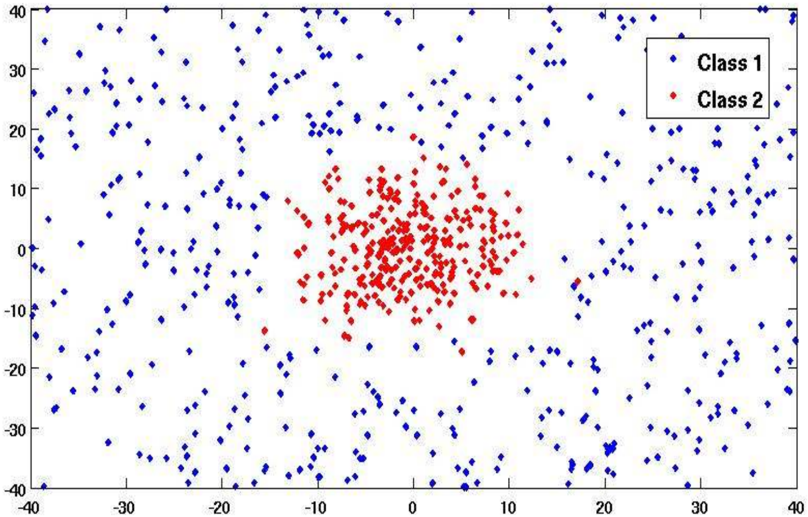
Applications of FFNN – Classification

- Training set in 2D space with two classes



Applications of FFNN – Classification

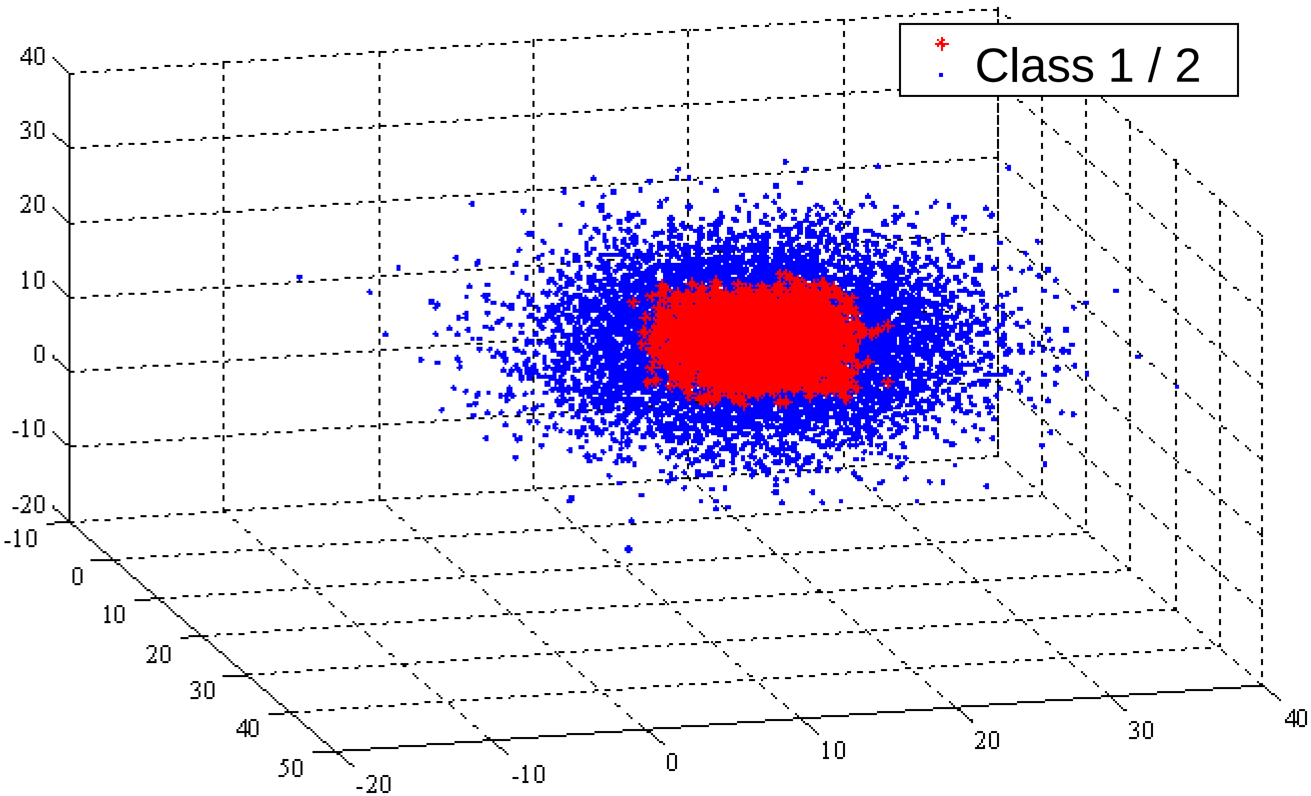
- Training set in 2D space
  - The classes may not be separable with hyper plane
  - The edges of classes are soft edges and there is no explicit rule



  - For example: a circle with center at 0,0 and with radius 7.
  - This information (where the edges are between the two classes) should be learned and generalized by the FFNN

Applications of FFNN – Classification

- Real classification by FFNN in a 3D example

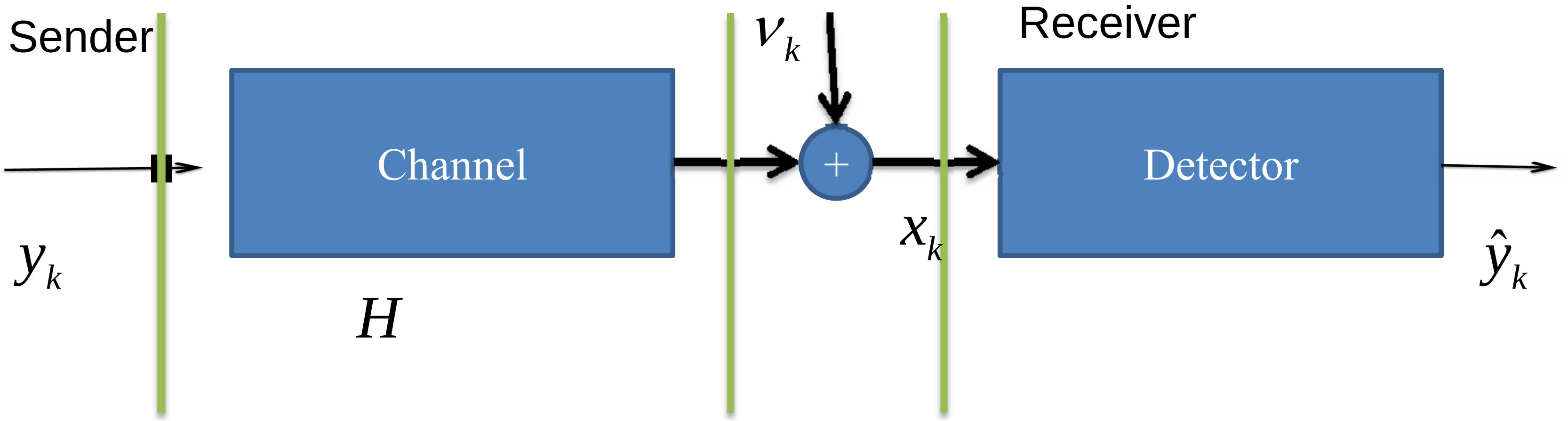


Applications of FFNN – Signal detection

- Signal detection in a wireless network
  - Given channel with defined noise
  - There is no information about the noise
    - No parameters, only observations
  - The sender transmits its symbols through this noisy channel
  - The receiver detects these symbols with noise
  - The task is to determine which symbols has been sent through this channel
  - To solve this task we can use FFNN as detector

Applications of FFNN – Signal detection

- There is no information about the noise
  - No parameters, only observations
  - This observation may be used as the training set for the network



Applications of FFNN – Signal detection, example

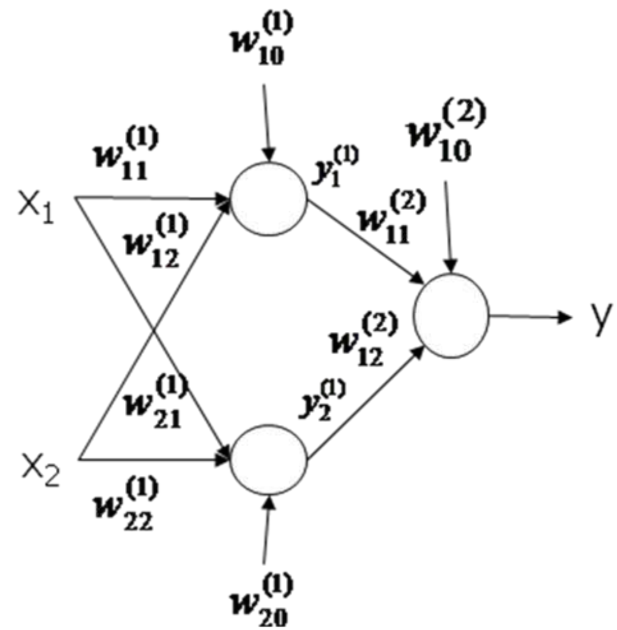
- Let us have the following impulse response from the channel (after channel identification $\mathbf{h} = [1 \quad 0.5]^T$
- The following training set can constructed from observations
  - Sent symbols $y_k = 1, -1, 1, -1$
  - Received values $x_k = 0.9, -0.1, 0.2, -0.7$
  - Training set (example, using two symbols as input)

$$\tau^{(3)} := \{([-0.7 \quad 0.2], -1); ([0.2 \quad -0.1], 1); ([-0.1 \quad 0.9], -1)\}$$

Applications of FFNN – Signal detection, example

- Structure of the FFNN
- Activation function for the output neuron should be the following

$$y = \varphi\left(I_1^{(2)}\right) = \frac{2}{1+e^{-\alpha I_1^{(2)}}} - 1 = \mathrm{th}\left(\frac{\alpha I_1^{(2)}}{2}\right)$$



- Because a differentiable function is needed, but it has to be very similar to the sign function in order to obtain –1 or +1 response of the neural network

Summary

- The architecture of the Feed forward Neural Network has been introduced
- The representation capability of the FFNN is the following

  $$\mathcal{NN} \subseteq_D L^p$$

- Blum and Li construction – LEGO principle
  - Constructive algorithm to approximate arbitrary function
- Back propagation algorithm
  - Training set, iterative algorithm to obtain information from the training set
- Bias-Variance dilemma, VC dimension
- Applications