Introduction to Neural Networks

AE4-350 Bio-inspired intelligence and learning for aerospace applications

Erik-Jan van Kampen
Delft University of Technology

May 1, 2019

Outline

Learning objectives

After this lecture, the student can:

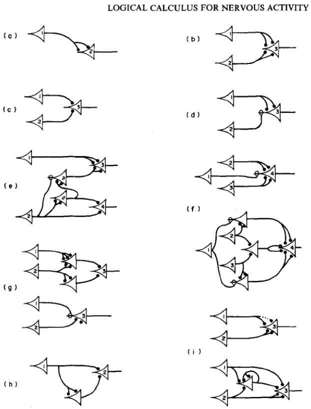
- Explain the relationship between biological neural networks and artificial neural networks.
- Describe the types of neural networks and their applications.
- Design and construct his/her own artificial neural network for a given task.

NASA Dryden Flight Research Center Photo Collection
http://www.dfrc.nasa.gov/Gallery/Photo/index.html
NASA Photo: EC03–0231–2 Date: August 27, 2003 Photo By: Jim Ross

NASA Dryden's highly–modified F–15B aircraft, tail number 837, serves as an Intelligent Flight Control System (IFCS) research testbed aircraft.

Introduction

- First appearance of the artificial neural network in the late 1800's but the modern era started in 1943 with the work of McCulloch and Pitts.



Introduction

- Artificial neurons are a mathematical representation of the neurons in the human brain.

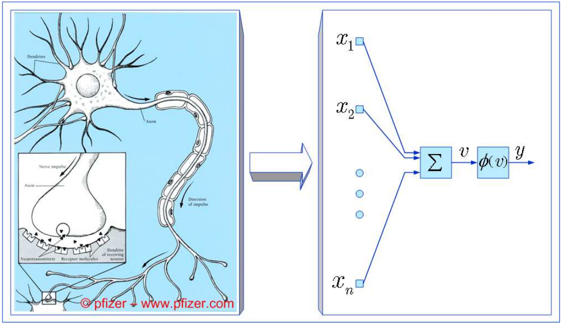


Figure: Biological neuron and the mathematical equivalent.

Introduction

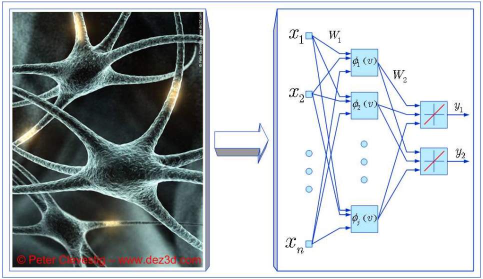


Figure: Mathematical representation of a network of neurons

- Initially the use of neural networks was limited due to difficulties in setting the weights
- Introduction of the error back-propagation learning method (1986, Rumelhart, Hinton, and Williams) created an efficient way in defining weights and gave a new incentive to the field on neural networks.

Introduction

- In the last decades neural networks have been used in a lot of different applications:
    - Pattern recognition
    - Speech recognition
    - Adaptive control
    - Stock market forecasting
    - Weather forecasting
    - Aerodynamic model identification
    - Games: checkers, backgammon, chess, ...
    - ...
- The method of incorporating neural networks for system identification and control can vary...

Introduction

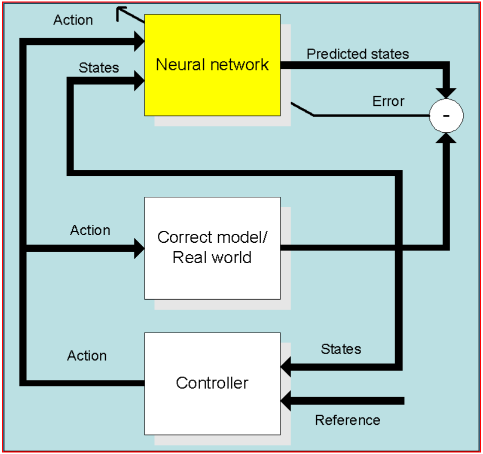


Figure: Neural networks for complete system identification

Introduction

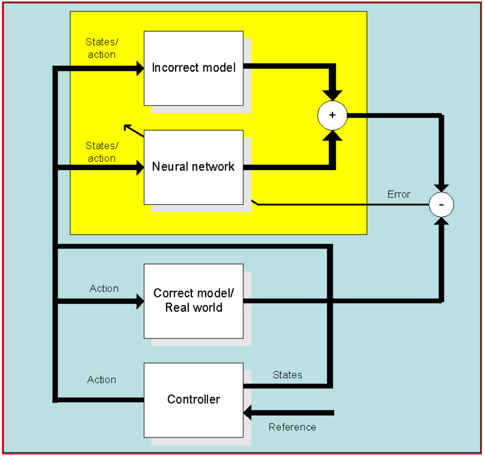


Figure: Neural networks for partial system identification. The neural network is used to correct a partial model or a linear model. This is one way of incorporating knowledge in system identification.

Introduction

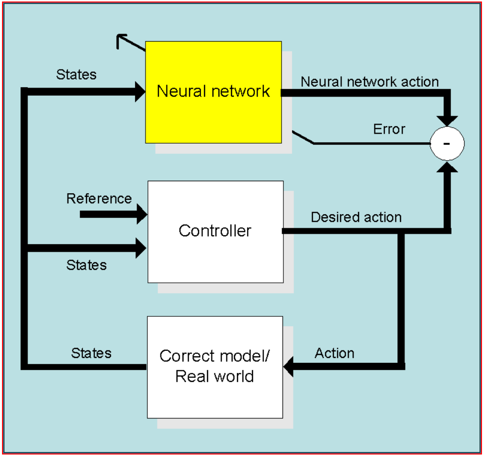


Figure: Neural networks for control. Training of the network is performed using a supervisor. The supervisor provides the error in the control output. With this error the network weights are updated.

Introduction

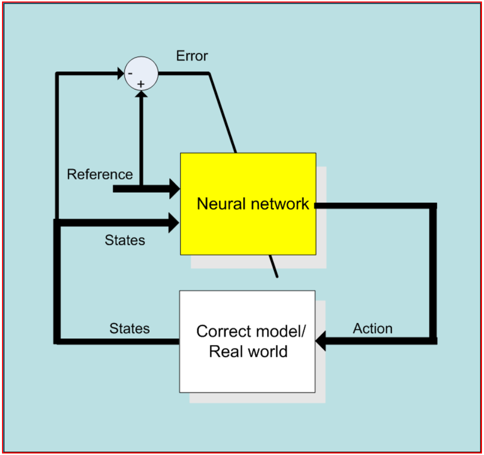
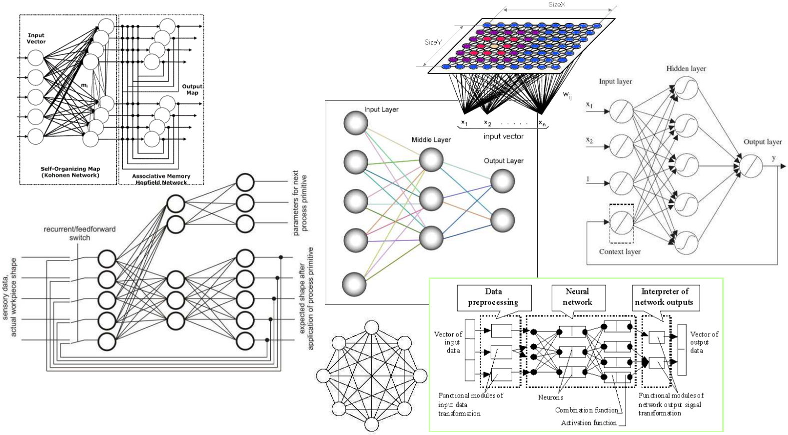


Figure: Neural networks for control. Unsupervised learning of the neural network. The network uses the difference between the reference and the state to adapt its weights, i.e. adapt the control law.

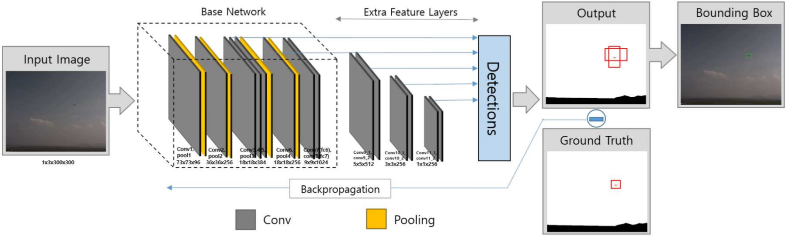
Introduction

- Many different network structures exist today ranging from simple to extremely complex networks.



Introduction

- Aircraft Detection using Deep Convolutional Neural Network for Small Unmanned Aircraft Systems. Sunyou Hwang, Jaehyun Lee, Heemin Shin, Sungwook Cho and David Hyunchul Shim. AIAA SciTech conference 2018.



Introduction

Introduction

- All neural network types are designed for a specific task: there is not a single network type which is optimal for all tasks.
- If you are given a task ...
  - which type of neural network is the most optimal?
  - what activation function must one use?
  - how many neurons are needed?
  - how are the neurons linked between layers?
  - which input and outputs are required?
  - what training algorithm must be used?
  - how can one collect the required training/testing data?
  - ...

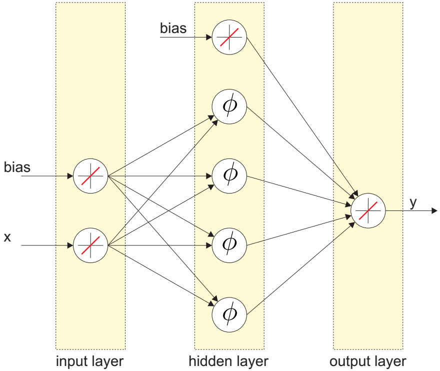
Introduction

Approach to achieve the learning goals:

- Static input-output mappings.
- Neural networks from a visual perspective (solving a puzzle).
- Linking the visual perspective to the mathematical representation and network definition.
- Discussion on training the networks offline.
- Discussion on training the networks online.
- Introduction into more complex neural networks.
- Viewing some applications of neural networks.

Visual inspection of neural networks

- Neural network design and optimization is much like puzzling.
- This will be demonstrated for a simple neural network



using a MATLAB demo...

Visual perspective on NN's

The demonstration has shown ...

- that a neural network output is a collection of weighted activation functions.
- the placement and shaping of the neurons can be difficult.
- activation function can be any desired function.
- the performance of the network depends on the chosen activation functions although almost perfect fits can be made with enough neurons.
- for any IO mapping there is an optimal set of activation functions such that the neural network contains the lowest amount of adaptable parameters to obtain a particular cost function value.

Visual perspective on NN's

Important questions

- How can one determine the optimal set of activation functions?
- What can one do if no information is available regarding the IO mapping?

Theorem on neural networks:

Cybenko (1989):

A feedforward neural net with at least one hidden layer with sigmoidal activation functions can approximate any continuous nonlinear function $\mathbb{R}^p \rightarrow \mathbb{R}^n$ *arbitrarily well on a compact set, provided that sufficient number of hidden neurons are available.*

Visual perspective on NN's

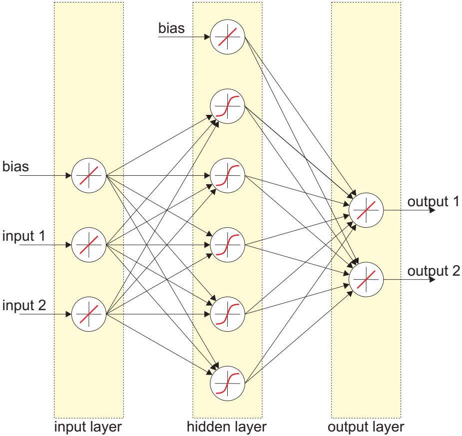
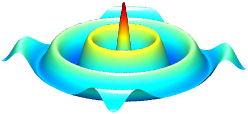


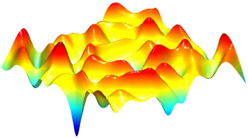
Figure: General structure of a feedforward neural network with a single hidden layer. Summation of inputs per neuron is performed inside the neuron before the activation function is applied.

Visual perspective on NN's

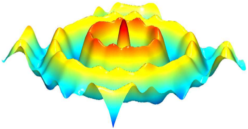
Demonstration of approximation power:



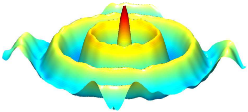
(a) Real IO mapping



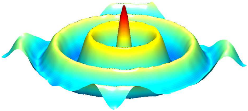
(b) NN 25 neurons



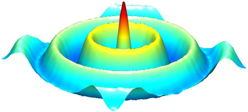
(c) NN 50 neurons



(d) NN 75 neurons



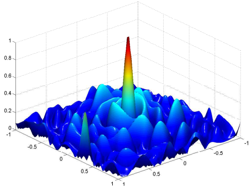
(e) NN 100 neurons



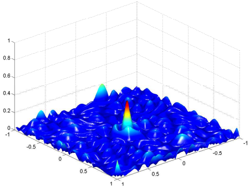
(f) NN 200 neurons

Visual perspective on NN's

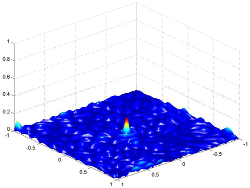
Demonstration of approximation power: error plots



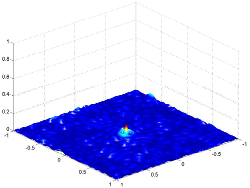
(a) NN 25 neurons



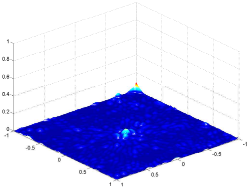
(b) NN 50 neurons



(c) NN 75 neurons



(d) NN 100 neurons



(e) NN 200 neurons

Mathematical description

General mathematical description of a single hidden layer feedforward neural network:

$$
\begin{aligned}
y_k &= \phi_k\left(v_k\right); \\
v_k &= \sum_j w_{jk} y_j = \sum_j w_{jk} \phi_j\left(v_j\right) \\
v_j &= \sum_i w_{ij} y_i = \sum_i w_{ij} \phi_i\left(v_i\right)
\end{aligned}
$$

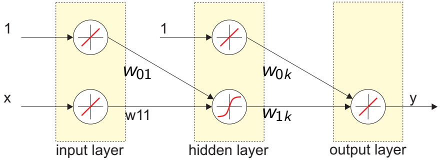
Simplified mathematical description:

$$
\left.
\begin{aligned}
y_k &= v_k \\
v_k &= \sum_j w_{jk} \phi_j\left(v_j\right) \\
v_j &= \sum_i w_{ij} x_i
\end{aligned}
\right\} y_k = \sum_j w_{jk} \phi_j \left(\sum_i w_{ij} x_i \right)
$$

Mathematical description

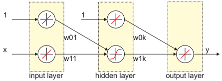
- The input weights w_{ij} and output weights w_{jk} determine the position and the shape of the activation function.
- This can be shown by a simple demonstration for a single hidden neuron neural network...

$$y = w_{1k}\phi_j\left(w_{11}x + w_{01}\right) + w_{0k}$$

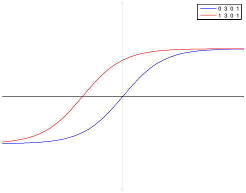


Worksheet!

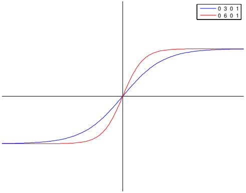
Mathematical description



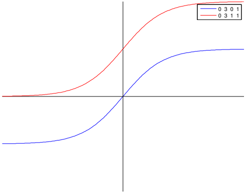
(a) FF structure



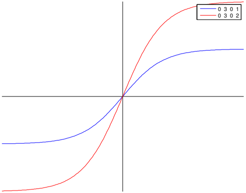
(b) Adaptation of w_{01}



(c) Adaptation of w_{11}



(d) Adaptation of w_{0k}



(e) Adaptation of w_{1k}

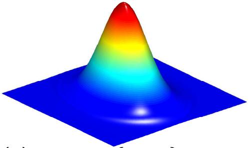
Mathematical description

- The feedforward neural networks with sigmoidal activation functions can approximate any function.
- It is one of the most popular networks.
- Another very popular type of network is the radial basis function neural network:

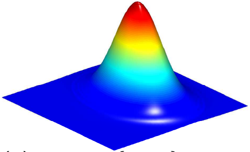
$$\left.\begin{aligned} y_k &= v_k \\ v_k &= \sum_j w_{jk}\phi_j\left(v_j\right) \\ v_j &= \sum_i w_{ij}\left(x_i - c_{ij}\right)^2 \end{aligned}\right\} y_k = \sum_j w_{jk}\phi_j\left(\sum_i w_{ij}\left(x_i - c_{ij}\right)^2\right)$$

The activation function looks like ...

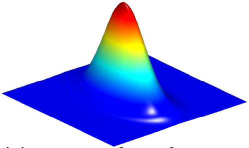
Mathematical description



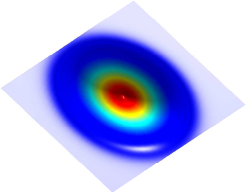
(a) $W_{ij} = \{10, 5\}, C_{ij} = \{0, 0\}$



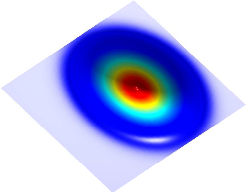
(b) $W_{ij} = \{10, 5\}, C_{ij} = \{1, 0\}$



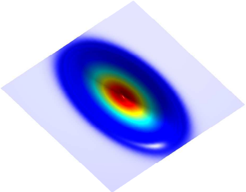
(c) $W_{ij} = \{20, 5\}, C_{ij} = \{0, 0\}$



(d) Top view



(e) Top view



(f) Top view

Mathematical description

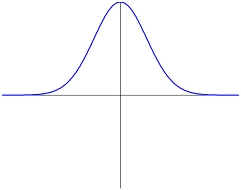
RBF network

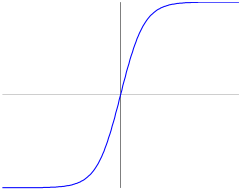
- Local character
- Easy to model local complexity
- Bad generalization?
- High computation load
- Easy optimization

$\Longleftrightarrow$

FF network

- Global character
- Easy to model global trends
- Good generalization?
- Low computation load
- Difficult optimization





Mathematical description

Summary

- Two commonly used neural networks introduced
- Neural network output is a summation of activation functions
- Characteristics of IO mapping should be reflected by the activation functions
- Shape of the activation functions adjusted by network weights
- Information stored in the network weights
- IO mapping complexity together with choice in activation function determines the required number of neurons.

- In the following slides, static IO mappings in combination with standard feedforward networks and RBF networks are considered.
- Next step: training networks ...

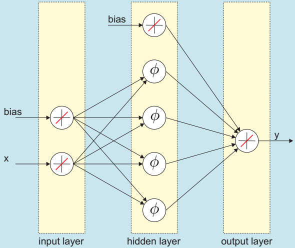
Training networks

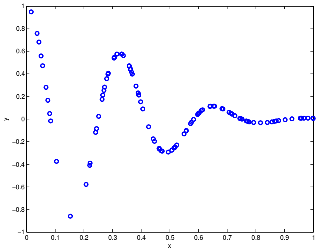
Start-off point

- Single hidden layer feedforward neural network (fixed structure)

$$y_k = \sum_j w_{jk} \phi_j \left(\sum_i w_{ij} x_i \right)$$

- Static IO mapping, fixed set of data (x, d).





Training networks

Aspects off training neural networks:

- Network structure
- Network weight initialization
- IO data pre-processing
- Adaptation rules
- Stopping conditions

For now consider ...

- Random initialization
- No data pre-processing

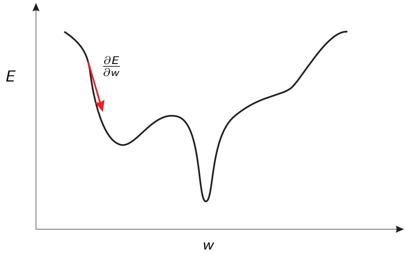
Next: error back propagation...

Error-back propagation

- Network performance measured with a squared cost function:

$$E = \frac{1}{2}\sum_q \sum_k \left(d_{k,q} - y_{k,q}\right)^2$$

- The error-back propagation approach tries to minimize the cost function by traveling down the slope:



Error-back propagation

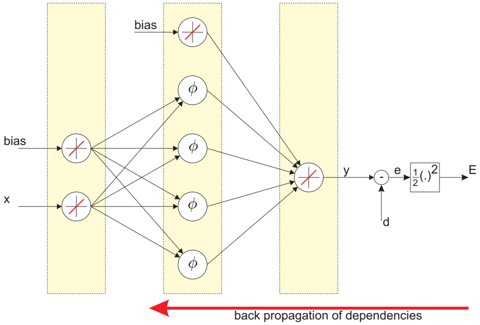
- Updates performed using partial derivatives: step taken in the negative gradient direction.
- Update law for the network weights given by:

$$w_{t+1} = w_t + \Delta w; \;\; \Delta w = -\eta \frac{\partial E}{\partial w_t}$$

- Update law depends on ...
  1. ... partial derivatives
  2. ... learning rate parameter η
- Partial derivative can be computed in a structured way using the chain rule: back-propagation of the errors

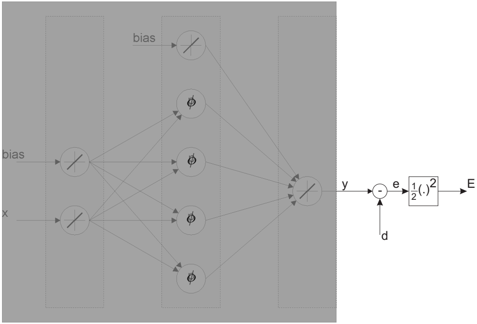
Error-back propagation

- First a forward computation step is performed to compute the neuron inputs (v) and outputs (y) (consider a single IO data point)
- Then the output errors are computed (e_k)
- Finally the cost function dependencies are propagated from right to left...



Error-back propagation

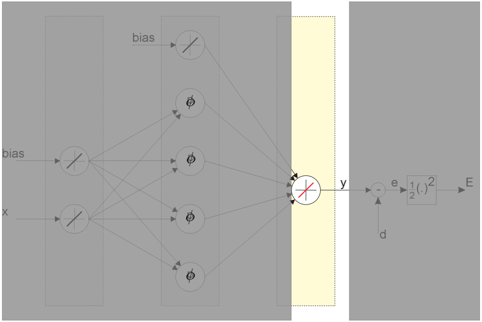
- Step 1: dependencies w.r.t. network outputs



$$
\begin{aligned}
E &= \sum_k \frac{1}{2}\left(d_k - y_k\right)^2 \\
\frac{\partial E}{\partial y_k} &= \frac{\partial E}{\partial e_k}\frac{\partial e_k}{\partial y_k} = e_{k,q}\cdot - 1
\end{aligned}
$$

Error-back propagation

- Step 2: dependencies w.r.t. neuron inputs

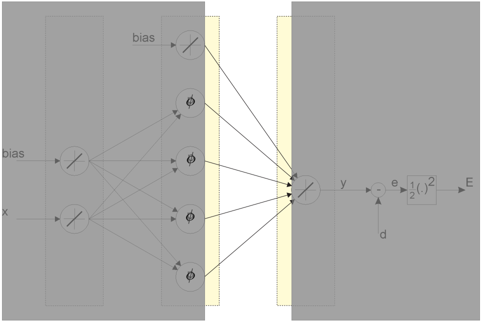


$$
\begin{aligned}
y_k &= \phi_k\left(v_k\right) \\
\frac{\partial y_k}{\partial v_k} &= \frac{\partial \phi_k}{\partial v_k}
\end{aligned}
$$

linear activation function $\rightarrow \frac{\partial y_k}{\partial v_k} = 1$

Error-back propagation

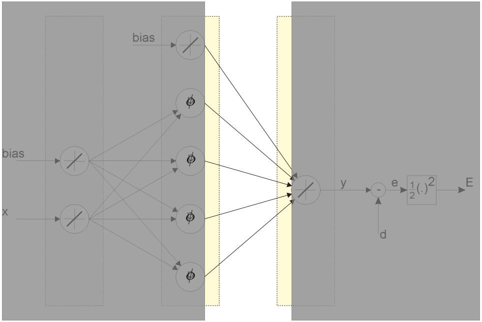
- End goal 1: dependencies w.r.t. output weights



$$v_k = \sum_j w_{jk} y_j \qquad \frac{\partial v_k}{\partial w_{jk}} = y_j \quad \rightarrow \frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial w_{jk}} = e_{k,q}. -1.\frac{\partial \phi_k}{\partial v_k}.y_j$$

Error-back propagation

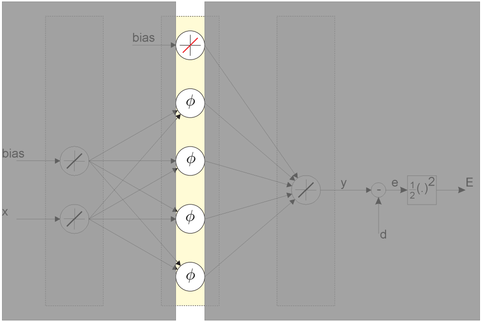
- Step 3: dependencies w.r.t. neuron output



$$\begin{aligned} v_k &= \sum_j w_{jk} y_j \\ \frac{\partial v_k}{\partial y_j} &= w_{jk} \end{aligned} \quad \rightarrow \frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial v_k} \frac{\partial v_k}{\partial y_j}$$

Error-back propagation

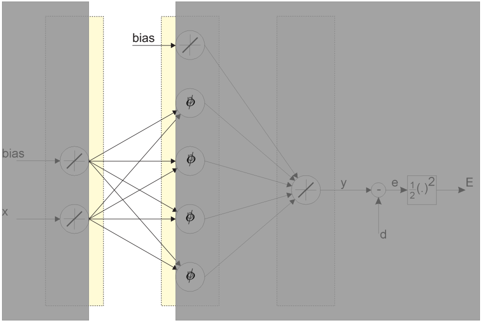
- Step 4: dependencies w.r.t. neuron input



$$\begin{aligned} y_j &= \phi_j\left(v_j\right) \\ \frac{\partial y_j}{\partial v_j} &= \frac{\partial \phi_j}{\partial v_j} \end{aligned}$$

Error-back propagation

- End goal 2: dependencies w.r.t. input weights



$$
\begin{aligned}
v_j &= \sum_i w_{ij} y_i \\
\frac{\partial v_j}{\partial w_{ij}} &= y_i
\end{aligned}
\quad \rightarrow \quad
\begin{aligned}
\frac{\partial E}{\partial w_{ij}} &= \sum_k \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial v_k} \frac{\partial v_k}{\partial y_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial w_{ij}} \\
&= \sum_k e_{k,q} . -1 . \frac{\partial \phi_k}{\partial v_k} . w_{jk} . \frac{\partial \phi_j}{\partial v_j} . y_i
\end{aligned}
$$

Error-back propagation

- Complete partial derivatives (for all data points):
  1. Output weights:

$$\frac{\partial E}{\partial w_{jk}} = \sum_q \frac{\partial E}{\partial e_{k,q}} \frac{\partial e_{k,q}}{\partial y_{k,q}} \frac{\partial y_{k,q}}{\partial w_{jk}} = \sum_q e_{k,q}. -1. y_j$$

  2. Input weights:

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \sum_q \left[\sum_k \left\{ \frac{\partial E}{\partial e_{k,q}} \frac{\partial e_{k,q}}{\partial y_{k,q}} \frac{\partial y_{k,q}}{\partial y_{j,q}} \frac{\partial y_{j,q}}{\partial v_{j,q}} \frac{\partial v_{j,q}}{\partial w_{ij}} \right\} \right] \\ &= \sum_q \sum_k e_{k,q}. -1. w_{jk} \frac{\partial \phi_{j,q}}{\partial v_{j,q}} x_i \end{aligned}$$

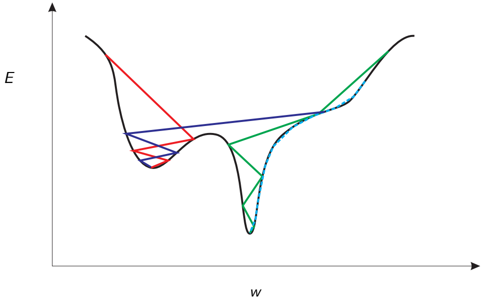
- If multiple layers are present than use the same method.

Error-back propagation

- Now that the derivatives are known the update of the weights can be performed:

$$w_{t+1} = w_t + \Delta w;\ \Delta w = -\eta \frac{\partial E}{\partial w_t}$$

- The learning rate is used to determine the step size
- Together with the initialization point they determine the final approximation performance:



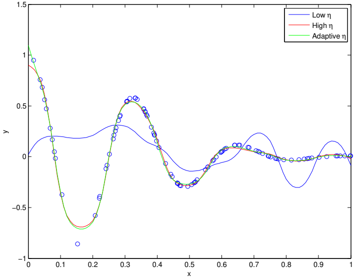
Error-back propagation

Basic learning algorithm: fixed learning rate

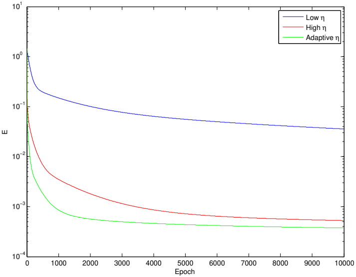
- Initialization
- LOOP (number of epochs)
  - Compute cost function value E_t and weight update ΔW for current set of weights W_t
  - Perform update of weights and compute new cost function value E_{t+1}
  - If $E_{t+1} < E_t$ then accept changes else stop loop
- END LOOP

- Small learning rate $\rightarrow$ slow convergence to local minimum (derivatives zero)
- Large learning rate $\rightarrow$ fast convergence, but end may not be a true minimum (derivatives non-zero)

Error-back propagation



(a) Approximation



(b) Performance

Figure: Influence of the learning rate on the performance time history during training

Error-back propagation

Learning algorithm: adaptive learning rate

- Initialization
- LOOP (number of epochs)
  - Compute cost function value E_t and weight update ΔW for current set of weights W_t and η_t
  - Perform update of weights and compute new cost function value E_{t+1}
  - If $E_{t+1} < E_t$ then accept changes and increase learning rate $\eta_{t+1} = \eta_t * \alpha$ else do not accept changes and decrease learning rate $\eta_{t+1} = \eta_t * \alpha^{-1}$.
  - Stop loop if partial derivatives are (nearly) zero.
- END LOOP

Error-back propagation

- The previous training algorithms are all first order gradient-descent algorithms (a.k.a. steepest descent)
- There are second order methods available of which the most commonly used is the Levenberg-Marquardt method:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \left(\mathbf{J}^T\mathbf{J} - \mu\mathbf{I}\right)^{-1}\mathbf{J}\mathbf{e}; \quad \mathbf{J} = \frac{\partial \mathbf{y}}{\partial \mathbf{w_t}}$$

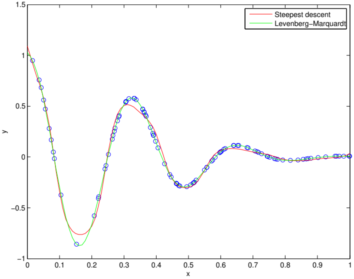
- If μ is larger then the LM method reduces to a first order steepest-descent method:

$$\mathbf{w}_{t+1} \approx \mathbf{w}_t - \left(\mu\mathbf{I}\right)^{-1}\mathbf{J}\mathbf{e} = \mathbf{w}_t - \mu^{-1}\mathbf{J}\mathbf{e}$$

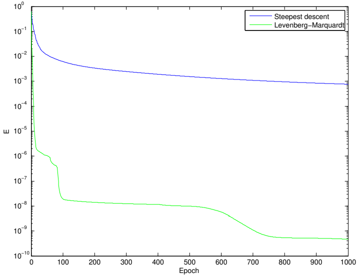
- If μ is small then the LM method reduces to a pure Newton step:

$$\mathbf{w}_{t+1} \approx \mathbf{w}_t - \left(\mathbf{J}^T\mathbf{J}\right)^{-1}\mathbf{J}\mathbf{e}$$

Error-back propagation



(a) Approximation



(b) Performance

Figure: Influence of the training algorithm on the performance time history during training

Levenberg-Marquardt is much more efficient and shows a faster convergence.

Error-back propagation

1 order method

- Low computation cost
- Numerically stable
- Slower convergence
- Gentle weight updates
- Easy implementation

$\Longleftrightarrow$

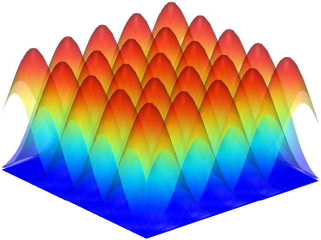
2 order method

- Higher computation cost
- Possibly unstable
- Fast convergence
- Aggressive weight updates
- More difficult implementation

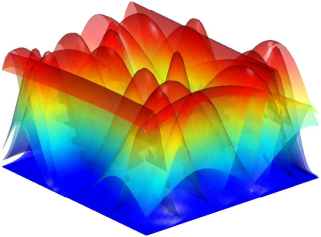
Initialization

- As mentioned previously, the initialization of the weights has a large influence
- For feedforward neural networks with sigmoidal activation function random initialization is used: small weight values means that the first order derivative is non-zero in the entire input space.
- For the RBF neural networks one can do the same but also uniform distribution can be performed (see next slide).
- Optimal initialization requires knowledge of the true IO mapping.
- Common practice is to perform the training several time, each time with different initial weights.

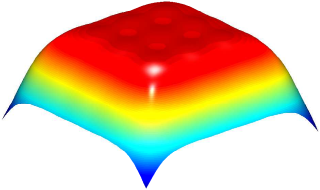
Initialization



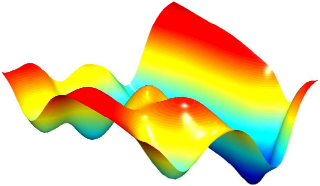
(a) Uniform



(b) Random



(c) IO mapping



(d) IO mapping

Neural network training

Summary

- Several basic training algorithms explained
- Error-back propagation algorithms most commonly used
- Fast convergence, aggressive steps vs slower convergence, gently steps
- Initialization has a large influence on the final performance

- Next step: defining your neural network in a structured way ...

Defining your network

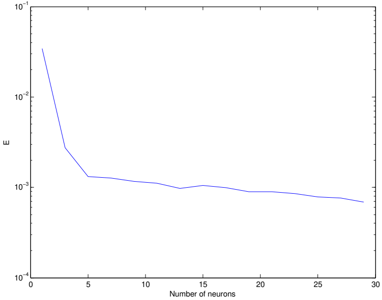
- Start-off point:
  - Input and output are set
  - Activation functions are set
  - Static IO mapping
  - IO data available

Question

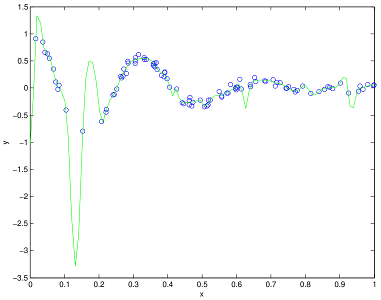
How many neurons do you need for optimal performance?

- Approach: Iteration based on performance
  - Perform training phase for a given number of neurons and several weight initializations.
  - Increase the number of neurons and if the performance increase is large enough: add more neurons.
- Pitfalls: Over-fitting, loss of generalization
- An example ...

Defining your network



(a) Performance

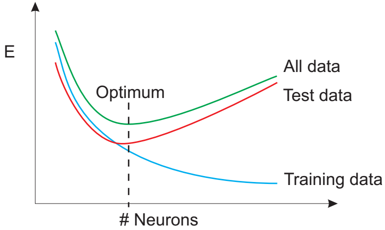


(b) Output for 30 hidden neurons

Figure: Example of the over-fitting problem

Defining your network

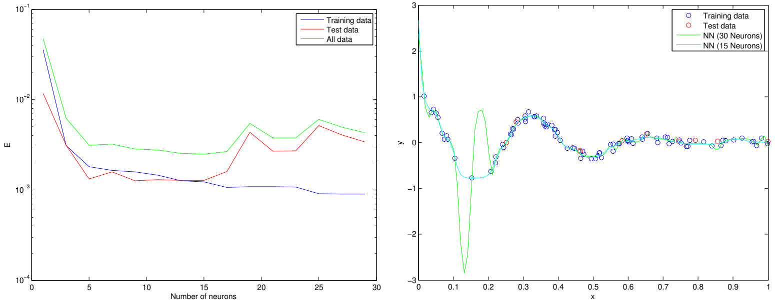
- Common approach to prevent over-fitting: separation of training and test data
- Training algorithm stops if the cost function value, using both data sets, is non-decreasing!



- Common ratios between training and test data: 90/10, 80/20.
- The test data will pose an upper limit on the number of neurons!
- An example ...

Defining your network

- 90% training data, 10% test data, per network 10 initializations:



(a) Performance (b) Output for two networks

Figure: Example of using test data to prevent over-fitting

Defining your network

- For RBF networks there are more possibilities with optimization since the effects per neuron are local
- A lot of pruning and allocation methods exist
- Principle of training/test data still applies
- Routines are available in the MATLAB neural network toolbox

Mathematical description

Summary up to now

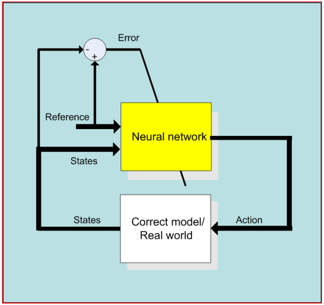
- Covered all basic aspects of off-line training for static IO mappings:
  - Activation function selection
  - Optimal number of neurons
  - Weight initialization
  - Training algorithms
- Two most commonly used networks treated: FFNN and RBFNN
- Creating the optimal network requires knowledge of the IO mapping
- No universal way of finding the optimal network

What remains to be treated:

- On-line training
- Discussion on more complex neural networks
- Interval analysis
- Assignment

On-line training

- Required when the IO mapping is time-varying:
  - Aerodynamic model identification in case of failures
  - UAV motor dynamics
  - Reinforcement learning
- On-line training usually required in a closed loop system identification setting



On-line training

- Problems with on-line training:
  - Gathering enough IO data
  - Loss of information during training
  - Problems of network optimization (activation functions, number of neurons, structure)
  - Defining inputs
  - Convergence issues
- Approach taken in the following slides
  - Random initialized neural network (mimic previous IO mapping)
  - Sweep through input space to gather IO data
  - Sequential training

On-line training

- Example of the problems with on-line training ... (movie)
- Problems with FFNN: global behavior of activation functions
- Problem can be reduced by using:
  - Time-window with batch updating
  - Gentle updates: steepest descent instead of Levenberg-Marquardt
  - Anti-recency points ... (movie)
  - Store previous IO data points
  - Activation function with a local character like RBF ... (movie)

On-line training

Summary

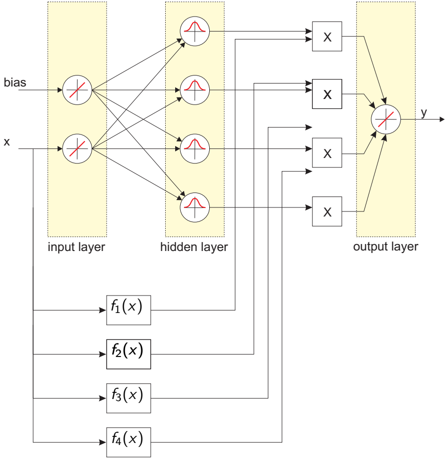
- When dealing with off-line identification neural networks are very suitable
- Optimization of neural networks is a non-linear optimization problem making neural networks less suitable for on-line system identification.
- Neural networks with ‘local’ activation functions such as RBF perform better in online learning.
- There will always be a trade-off between remembering previously learned information and learning new information.

Advanced neural network structures

- Theory shows that all IO mappings can be approximated to any desired accuracy with a single hidden layer feedforward neural network with sigmoidal activation functions.
- Practice has shown that the required number of neurons can be huge.
- Other activation functions are better suited or other network definitions are better suited.
- One very powerful group of networks uses the combination of local activation functions and polynomials...

Advanced neural network structures

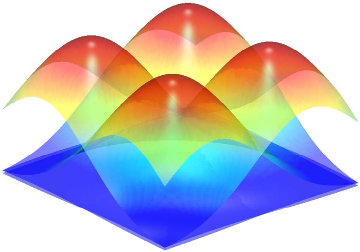
- Example of network layout:



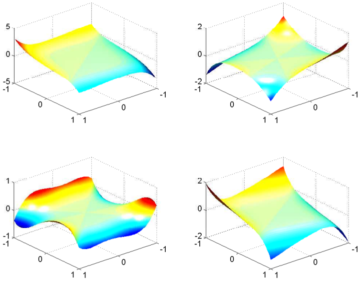
- Multiple layouts possible: most variations in the definition of the RBF neurons.

Advanced neural network structures

- The RBF neurons are used to defined sub-domains in the input-space.
- The polynomials approximate the IO mapping on each sub-domain.

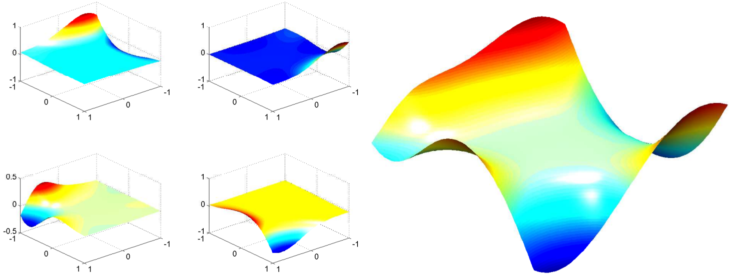


(a) RBF neurons



(b) Polynomials

Advanced neural network structures



(a) Output per neuron (b) Total network output

- That the IO mapping is continuous up to any degree (gradient descent methods applicable).
- RBF neurons ‘blend’ the polynomials between sub-domains.
- Local complexity can be adjusted by setting polynomial degree per neuron.

Advanced neural network structures

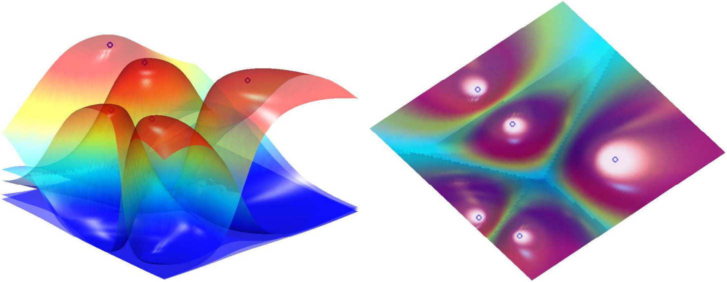
- Optimization of RBF neurons can be made based on allocation and pruning techniques.
- Optimization of polynomial order can be made based on performance on the sub-domains.
- Network comparable to fuzzy neural networks or multivariate splines (lectures of dr.ir. Coen de Visser)
- Note that the optimization of the polynomial coefficients is a convex optimization problem once the RBF weights are fixed.

Advanced neural network structures

- Other variation possible: Use of more elaborate RBF neuron definitions

$$u_t = \frac{1}{\sum_j \left(\frac{\|\mathbf{x}-\mathbf{v}_t\|}{\|\mathbf{x}-\mathbf{v}_j\|} \right)^{\frac{2}{m-1}}}$$

- More complex sub-domain definitions



(a) 3D view

(b) Top view

Figure: Output per neuron (or Degree of Membership functions)

Advanced training algorithms

- Neural network training (partially) non-linear optimization problem.
- Outcome of gradient-based algorithms dependent on weight initialization.
- More complex non-linear optimization algorithms are available to 'solve' the problem
  - Line search methods
  - Advanced gradient methods (better convergence, higher probability of optimal solution)
  - Genetic algorithms
  - etc...
- Previous methods can never give the guarantee that the global optimum or *all* global optima are found!
- Methods that do have this guarantee
  - Lipschitz bound based algorithms
  - Interval analysis based algorithms

Important notes on neural networks

- Each task has a unique network definition which is optimal (least amount of free parameters).
- Most complex networks can be broken down into the elements described in this lecture.
- Many learning algorithms available: start with gradient descent methods.
- Neural networks less suited for on-line applications, but if applied be aware of the recency effect (usually less neurons is better).
- When working with neural networks: if knowledge is available then use it the define you network and try to keep the network as small as possible! Less neurons means faster learning and fewer local minima!

Introduction to Neural Networks

AE4-350 Bio-inspired intelligence and learning for aerospace applications

Erik-Jan van Kampen
Delft University of Technology
May 1, 2019