Neural Networks: Introduction & Matlab Examples

Ricardo de Castro
Technische Universität München
Lehrstuhl für Elektrische Antriebssysteme
Agenda

• **Introduction & Motivation**

• **Single-Layer Neural Network**
  • Fundamentals: neuron, activation function and layer
  • Matlab example: constructing & evaluating NN
  • Learning algorithms
    • Batch solution: least-squares
    • Online solution: LMS
  • Matlab example: online system identification with NN

• **Multi-Layer Neural Network**
  • Network architecture
  • Learning algorithm: backpropagation
  • Matlab example: nonlinear fitting with noise
  • Overfitting & regularization

• **Case Study**
  • Matlab example: MPC solution via Neural Networks
References


Chapters 2,3, 10 and 11

(aka Deep Learning Toolbox)
Some Problems…

**Computer vision**

-symptoms: fever, headache, blood pressure, …

**Medical Diagnosis**

-biochemical analysis, age, height, …

**Finance (e.g. prediction)**

-stockPrice[k], stockPrice[k-1], …

**Control**

-(e.g., prediction / system identification)

-u[k], u[k-1],… u[k-N], y[k], y[k-1], …, y[k-M]

-y[k+1]

-u = control input, y=output, k=time index

How to build a system that can **learn** these tasks?
Motivation Example: Credit Approval

Application Information

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>23 years</td>
</tr>
<tr>
<td>gender</td>
<td>male</td>
</tr>
<tr>
<td>annual salary</td>
<td>$30,000</td>
</tr>
<tr>
<td>years in residence</td>
<td>1 year</td>
</tr>
<tr>
<td>years in job</td>
<td>1 year</td>
</tr>
<tr>
<td>current debt</td>
<td>$15,000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- credit approved
- credit rejected

Adapted from “Learning from Data, a Short Course”
Learning Framework

Formalism:

- Input: \( p = [\text{age, gender, annual salary, ...}] \in \mathbb{R}^R \) (customer application)
- Output: \( y \in \{0, 1\} \) (0 → credit rejected, 1 → credit approved)
- Ideal function: \( y = h(p) \) (\( h=\)ideal credit approval function)
- Data: \( (p_1, y_1), (p_2, y_2), ..., (p_N, y_N) \) (historical records)
- Candidate model: \( \hat{y} = g(p) \) (formula to be learned from data, e.g. Neural Network)

Remarks:
- Ideal (credit approval) function is unknown,
- ..., but banks have massive amounts of data (customer info, default history, etc...)

Adapted from “Learning from Data, a Short Course”
Learning Framework (II)

**Input** \( p \) → **Unknown Function** \( y = h(p) \) → **Output** \( y \)

**Data Set** \( D = \{(p_1, y_1), \ldots, (p_N, y_N)\} \)

**Candidate Model** (e.g. Neural Net) \( \hat{y} = g(p, \theta) \)

**Learning Algorithm**

Adapt \( \theta \) s.t. \( y \approx \hat{y} \)

**Main goal:** learn \( h \) (i.e. \( g \approx h \)) **using data** \( D \)

Adapted from “Learning from Data, a Short Course”
Classification vs Regression

**Classification Problems**
Output $\rightarrow$ discrete categories

**Regression Problems**
Output $\rightarrow$ continuous variable

\[
\begin{bmatrix}
stockPrice(k) \\
stockPrice(k - 1) \\
\vdots \\
stockPrice(N - 1)
\end{bmatrix}
\rightarrow y \in [0, \infty)
\]
Neuron (Single Input)

\[ a = f(wp + b) \]

Input
- \( p \in \mathbb{R} \)

Parameters
- weight: \( w \in \mathbb{R} \)
- bias: \( b \in \mathbb{R} \)

Net Input
- \( n \in \mathbb{R} \)

Output
- \( a \in \mathbb{R} \)

Activation/Transfer Function
- \( f: \mathbb{R} \rightarrow \mathbb{R} \)
Neuron: Activation Functions

**Linear**

\[ f(n) = n \]

Matlab: `purelin`

**Hard Limit**

\[ f(n) = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases} \]

Matlab: `hardlim`

**Log-Sigmoid**

\[ f(n) = \frac{1}{1 + e^{-n}} \]

Matlab: `logsig`

**Hyperbolic Tangent Sigmoid**

\[ f(n) = \frac{e^n - e^{-n}}{e^n + e^{-n}} \]

Matlab: `tansig`
Neuron: Multiple Inputs

**element-wise representation**

\[ a = f(n) \]

Net input: \( n = w_{1,1}p_1 + w_{1,2}p_2 + \cdots + w_{1,R}p_R + b \)

Weights: \( w_{1,1}, w_{1,2}, \ldots, w_{1,R} \)

Bias: \( b \)

Inputs: \( p_1, p_2, \ldots, p_R \)

Number of inputs: \( R \)

**vector representation**

\[ a = f(Wp + b) \]

Net input: \( n = Wp + b \)

Weight vector: 
\[
W = \begin{bmatrix}
    w_{1,1} & w_{1,2} & \cdots & w_{1,R}
\end{bmatrix}
\]

Bias: \( b \)

Inputs: \( p = [p_1 \ p_2 \ \cdots \ p_R]^T \)
Single-Layer Neural Network (NN)

**element-wise representation**

\[ a_1 = f(w_{1,1} p_1 + w_{1,2} p_2 + \cdots w_{1,R} p_R + b_1) \]

\[ a_2 = f(w_{2,1} p_1 + w_{2,2} p_2 + \cdots w_{2,R} p_R + b_2) \]

\[ \vdots \]

\[ a_S = f(w_{S,1} p_1 + w_{S,2} p_2 + \cdots w_{S,R} p_R + b_S) \]

**Notation:**

\( w_{i,j} = \text{weight connecting neuron } i \text{ with input } j \)

**Remark:** each of the \( R \) inputs is connected to a neuron through a weight \( w_{i,j} \)

Number of inputs = \( R \)

Number of outputs = \( S \)
Single-Layer Neural Network (NN)

vector representation

\[ a = f(Wp + b) \]

\[ W = \begin{bmatrix}
  w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\
  w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\
  \vdots & \vdots & \ddots & \vdots \\
  w_{S,1} & w_{S,2} & \cdots & w_{S,R}
\end{bmatrix} \]

\[ p = \begin{bmatrix}
  p_1 \\
  p_2 \\
  \vdots \\
  p_R
\end{bmatrix} \quad b = \begin{bmatrix}
  b_1 \\
  b_2 \\
  \vdots \\
  b_S
\end{bmatrix} \quad a = \begin{bmatrix}
  a_1 \\
  a_2 \\
  \vdots \\
  a_S
\end{bmatrix} \]

\[ R = \text{number of inputs} \]
\[ S = \text{number of outputs} \]
Create and Evaluate Single-Layer NN (I)

Matlab Code

\[ P = \begin{bmatrix} 1 \\ 2 \end{bmatrix}; \quad \text{% data set - input (R=2)} \]
\[ Y = 1; \quad \text{% data set - output (S=1)} \]

\[ \text{net} = \text{linearlayer}; \]
\[ \text{% Creates a single layer of linear neurons} \]
\[ \text{% OUTPUT: net = neural network object} \]

\[ \text{net} = \text{configure}(\text{net}, P, Y); \]
\[ \text{% Configures network dimensions (inputs, outputs, weight,} \]
\[ \text{% bias, etc) to match the size of data set (P,Y)} \]
\[ \text{% INPUT} \]
\[ \text{net= neural network object} \]
\[ \text{P = [R-by-N] matrix with data set (input)} \]
\[ \text{Y = [S-by-N] matrix with data set (output)} \]
\[ \text{% OUTPUT} \]
\[ \text{net= configured neural network object} \]
Create and Evaluate Single-Layer NN (II)

Matlab Code

% net = Neural network object

net.layers{1}.transferFcn = 'hardlim';

net.IW{1,1} = W;   % set weights W
net.b{1} = b;      % set bias b

A = sim(net, P)
% Simulates neural network
% INPUT
% net= neural network object
% P = [R-by-N] input data
% OUTPUT
% A= [S-by-N] neural network output
Matlab Example

Consider a two-input network with one neuron and the following parameters:

\[ p = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad w = \begin{bmatrix} 3 & 2 \end{bmatrix} \quad b = 1.2 \]

Write a Matlab script that computes the output of the network for the following activation functions:

a) Hard limit
b) Linear
c) Log-Sigmoid

**Answers:**

a) 1.000  
b) 0.200  
c) 0.197
Learning Algorithm: Problem Setup

**Data set:** \( D = \{(p_1, y_1), ..., (p_N, y_N)\} \),
- vector input: \( p_i \in \mathbb{R}^R \)
- scalar output: \( y \in \mathbb{R} \)

**Single-layer linear NN:** \( a = Wp + b \)
- parameters \( \theta = [W \ b]^T \)

**Problem:** \( \min_{\theta} E(\theta) \)
- \( \theta^* = [W^* \ b^*]^T \)
  = optimal NN parameters

**Error function:**
\[
E(\theta) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - a_i(\theta))^2
\]
- \( a_i(\theta) = Wp_i + b \)
Learning Algorithm: Batch Solution

**Solution 1:**

\[
\min_{\theta} E(\theta) = \min \frac{1}{2N} \sum_{i=1}^{N} (y_i - a_i(\theta))^2
\]

A. vector representation

\[
\min_{\theta} E(\theta) = \min \frac{1}{2N} (Y - \Phi \theta)^T (Y - \Phi \theta)
\]

B. first-order optimality condition

\[
\nabla E(\theta) = 0
\]

\[
\theta^* = (\Phi^T \Phi)^{-1} \Phi^T Y
\]

Least-squares Solution

Problem: solution might be costly to compute for large data sets
Learning Algorithm: On-line Solution

Solution 2:
\[
\min_{\theta} E(\theta) = \min_{\theta} \frac{1}{2N} \sum_{i=1}^{N} (y_i - a_i(\theta))^2
\]

A. Split error function
\[
\min_{\theta} E(\theta) = \min_{\theta} \frac{1}{N} \sum_{i=1}^{N} E_i
\]

B. Sequential gradient descent
\[
\theta^{(\tau+1)} = \theta^{(\tau)} - \alpha \nabla E_i
\]

C. Compute gradient
\[
\nabla E_i(\theta) = -(y_i - \theta^T [p_i]) [p_i]
\]

\[
\theta^{(\tau+1)} = \theta^{(\tau)} + \alpha (y_i - [p_i 1] \theta^{(\tau)}) [p_i]
\]

aka least-mean squares (LMS) algorithm

Remarks:
- data points are processed one at a time (useful for real-time applications)
- learning rate needs to be chosen with care to ensure algorithm convergence [Hagan, Chap. 10]
On-line Learning Algorithm: Matlab API

Matlab Code

```matlab
%net = Neural network object

net.IW{1,1} = ...; % set initial weights
net.b{1}    = ...; % set initial bias

% setup learning algorithm
net.inputWeights{1,1}.learnFcn = 'learnwh';
net.biases{1}.learnFcn = 'learnwh';

net.inputWeights{1,1}.learnParam.lr = alpha;
net.biases{1,1}.learnParam.lr = alpha;
net.trainFcn = 'trains'; %

[net] = adapt(net,p,y); %
% INPUT
% net= neural network object
% p = [R-by-1] data point- input
% y = [S-by-1] data point- output
% OUTPUT
% net= updated neural network object (with new weights and bias)
```

Remark: simplified API for function `adapt()`
Online System Identification via NN (I)

Diagram:
- **Plant** with control input $u$ generating output $y$.
- **Neural Network** with weights $W$, bias $b$, and activation function $f$, feeding into the plant.
- **Learning Algorithm** receiving NN output $a$.
Online System Identification via NN (II)

Consider the following discrete-time model of a plant

\[ y[k] = u[k] + 1.8 u[k - 1] + 0.9 u[k - 2] \]

**Goal:** design and train a single-layer NN capable of approximating the output of this plant

- **Proposed structure** for the NN
  - Single layer
  - Inputs: \( p = [u[k] \quad u[k - 1] \quad u[k - 2]]^T \)
  - Activation function: linear

Write a Matlab script that:

a) **generates a data set** for the problem
   - assume: \( u[k] = \sin(2\pi f T_s k), \ T_s = 1s, \ f = 0.02Hz, \ k = 1,2,3,...,150 \)

b) **plots** data set

c) **creates the NN** and define initial weights and bias: \( W = \begin{bmatrix} 0 & -10 & 0 \end{bmatrix}, \ b = -2 \)

d) **adapts** the weight and bias of the NN in order to approximate the plant’s output \( y[k] \)
   - option 1: via `adapt(.)`; option 2: manual implementation of the LMS algorithm
   - learning rate \( \alpha = 0.2 \)

e) **plots** i) plant’s output, NN’s output; ii) difference between plant’s output and NN’s output
Online System Identification via NN (III)

Results:

a) Training Data

b) Fitting Results

Final weights/bias

\[ \mathbf{W} = [6.23 \quad -5.06 \quad 3.58], \quad b = 0.0883 \]
Multi-Layer Neural Network (I)

Example: 2-layer NN

\[ R = \text{number of inputs} \]

Element-wise representation

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of neurons</td>
<td>( S^1 )</td>
</tr>
<tr>
<td>Outputs</td>
<td>( a_1^1, a_2^1, ..., a_{S^1}^1 )</td>
</tr>
<tr>
<td>Weights</td>
<td>( w_{ij}^1 ) ( i = 1, ..., S^1; j = 1, ..., R )</td>
</tr>
<tr>
<td>Bias</td>
<td>( b_j^1, j = 1, ..., S^1 )</td>
</tr>
<tr>
<td>Activation function</td>
<td>( f^1 )</td>
</tr>
</tbody>
</table>
Multi-Layer Neural Network (II)

**Example: 2-layer NN**

Vector representation

<table>
<thead>
<tr>
<th></th>
<th>Layer 1</th>
<th>Layer 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of neurons</strong></td>
<td>$S^1$</td>
<td>$S^2$</td>
</tr>
<tr>
<td><strong>Outputs</strong></td>
<td>$a^1$</td>
<td>$a^2$</td>
</tr>
<tr>
<td><strong>Weights</strong></td>
<td>$W^1$</td>
<td>$W^2$</td>
</tr>
<tr>
<td><strong>Bias</strong></td>
<td>$b^1$</td>
<td>$b^2$</td>
</tr>
<tr>
<td><strong>Activation function</strong></td>
<td>$f^1$</td>
<td>$f^2$</td>
</tr>
</tbody>
</table>
Multi-Layer Neural Network (III)

**Hidden layer**

\[
a^1 = f^1(W^1p + b^1)
\]

**Output layer**

\[
a^2 = f^2(W^2a^1 + b^2)
\]

\[
a^2 = f^2(W^2 f^1(W^1p + b^1) + b^2)
\]

**Remarks:**

- Layer 1 → Hidden/Input layer
- Layer 2 → Output layer
- \( R, S^2 \) → defined by **data set** (input and output dimensions)
- \( S^1, f^1, f^2 \) → defined by the **designer**
- \( W^1, W^2, b^1, b^2 \) → parameters defined by the **learning algorithm**
Learning Algorithm: Problem Setup

Data set: \( D = \{(p_1, y_1), ..., (p_N, y_N)\} \),
- vector input: \( p_i \in \mathbb{R}^R \)
- scalar output: \( y \in \mathbb{R} \)

Two-layer NN:
\[
\alpha = g(p, \theta) = f^2(f^1(W^1p + b^1) + b^2)
\]
parameters \( \theta = (W^1, W^2, b^1, b^2) \)

Problem: \( \min_{\theta} E(\theta) \)

Error function:
\[
E(\theta) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - a_i(\theta))^2
\]

\( a_i(\theta) = g(p_i, \theta) \)

\( \theta^* = \text{optimal NN parameters} \)
Learning Algorithm: Backpropagation

Solution:
\[
\min_{\theta} E(\theta) = \min \frac{1}{2N} \sum_{i=1}^{N} (y_i - a_i(\theta))^2
\]

A. Split error function
\[
\min_{\theta} E(\theta) = \min \frac{1}{N} \sum_{i=1}^{N} E_i
\]

B. Sequential gradient descent
\[
\theta^{(\tau+1)} = \theta^{(\tau)} - \alpha \nabla E_i
\]

C. Compute gradient
\[
\nabla E_i(\theta) = -(y_i - a_i) \frac{\partial g(p_i, \theta)}{\partial \theta}
\]

\[
g(p_i, \theta) = f^2(W^2 f^1(W^1 p + b^1) + b^2)
\]

Backpropagation algorithm
compute gradients of \(g(p_i, \theta)\) using chain rule
(see [Hagan, Chap. 11] for details)
Matlab API: Create a 2-layer NN

Matlab Code

\[ P = \begin{bmatrix} 1 \\ 2 \end{bmatrix}; \] % data set - input (R=2)
\[ Y = 1; \] % data set - output

\[ \text{hiddensize} = [10]; \]
\[ \text{net} = \text{feedforwardnet}(\text{hiddensize}); \]
% Creates a multi-layer neuron network
% INPUT
% \text{hiddensize} = row vector with one or more
% hidden layer sizes
% OUTPUT: \text{net} = neural network object

\[ \text{net} = \text{configure}(\text{net}, P, Y); \]
% configures network dimensions (inputs, outputs, weight,
% bias, etc) to match the size of data set (P,Y)
Matlab Code

% define activation function, % weight and bias of Layer 1
net.layers{1}.transferFcn = 'logsig';
net.IW{1,1} = ...; % set W1
net.b{1} = ...; % set b1

% define activation function, % weight and bias of Layer 2
net.layers{2}.transferFcn = 'purelin';
net.LW{2,1} = ...; % set W2
net.b{2} = ...; % set b2

% Remark: net.b{i} = bias of layer i; net.LW{i,j} = weights connecting layer i with layer j
Matlab API: Training Algorithm

Matlab Code

```
net.trainFcn = 'traingd';  % Gradient descent backpropagation
net.trainFcn = 'trainlm'; % Levenberg-Marquardt backpropagation (default)
net.trainParam.lr = 0.01;   % learning rate (\alpha)
net.trainParam.epochs = 1000;  % Maximum number of epochs to train
net.trainParam.goal = 0;    % performance goal (stop criteria)

net.trainParam.min_grad = 1e-5; % Minimum performance gradient (stop criteria)
```

Data Set

Learning/Training Algorithm

\[ W^1, b^1, W^2, b^2 \]
Matlab API: Training Algorithm

Matlab Code

```matlab
net.trainParam.showWindow = 0; % deactivate interactive GUI
net.trainParam.showCommandLine=1; % Generate command-line output
net.divideFcn = ''; % use entire data set for training

[net]=train(net, P,Y);
% train trains a network net according to net.trainFcn and net.trainParam.
% INPUT
%    net= neural network object
%    P = [R-by-N] data set - input
%    Y = [S-by-N] data set - output
% OUTPUT
%    net = (trained) neural network output
```
Consider the following system

\[ D = \{ (p_i, y_i) \}, \text{ where } p_i \in \{-1, -0.9, -0.8, ..., 1\} \] is the input for the system

Write a Matlab script that:

a) **generates** an artificial data set for this problem, i.e.

\[ h(p) = 1 + \sin(\pi p) \]

b) **creates** a neural network (NN) with 2 layers;
   a) hidden layer with 10 neurons and activation function=logistic sigmoid
   b) output layer with 1 neuron and activation function=linear

c) **trains** the weights of the NN to fit the data set \( D \)

d) **plots**
   - training results, i.e. the data set \( D \) and the output of the trained NN
   - testing results, i.e. the data set \( D \), the (true) function and trained NN evaluated in the domain \( p_i \in \{-1, -0.99, -0.98, -0.97, ..., 1\} \)
Matlab Example: Nonlinear Fitting (II)

Perfect fitting of training data

We might be fitting noise, instead of the true signal \( (h) \)
Overfitting

Problem: previous example shown poor generalization, i.e.
  • NN fits very well the training data set,
  • … but produces large error when faced with new inputs

Solution*: Regularization
  • Idea: penalize large values of NN parameters (weights/bias)
    \[
    \min_{\theta} (1 - \lambda)E(\theta) + \lambda \|\theta\|^2
    \]
    \[
    \theta = (W^1, W^2, b^1, W^2, \ldots)
    \]
  • \(\lambda \in [0,1]\) → controls the importance of regularization vs fitting
  • larger \(\lambda\) → smoother fitting of neural network

Matlab Code

```matlab
net.performParam.regularization = 0.1; % sets value of lambda(\(\lambda\))
```

*Additional solutions: early stopping, reduce number of neurons (see Hagen, Chap. 13, for details)
Matlab Example: Nonlinear Fitting (III)

(Continuation of previous problem)

\[ p \rightarrow h(p) \rightarrow n \rightarrow y \]

- input: \( p \in [-1,1] \)
- output: \( y \in \mathbb{R} \)
- function: \( h(p) = 1 + \sin(\pi p) \)
- random noise: \( n \sim N(0, \sigma^2) \)

Gaussian distribution with zero mean and variance \( \sigma^2 = 0.2 \)

e) adapt the previous Matlab script in order to train neural network with regularization
   • use \( \lambda = 10^{-5} \)

f) Plot training and testing results for both cases
Matlab Example: Nonlinear Fitting (IV)

\( \lambda = 0 \)

\( \lambda = 10^{-5} \)
Matlab API: Auxiliary Commands

Matlab Code

gensim(net);
% net = neural network object

Generation of Simulink block for neural network simulation

view(net);
Generation of a graphical view
MPC-NN: Motivation

Consider the following discrete-time model of a mechanical system

\[
x[k + 1] = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \quad y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x
\]

\[
\begin{align*}
\{ u & = \text{acceleration } [\text{m/s}^2] \\
x & = [x_1 \ x_2]^T, \quad x_1 = \text{position}[\text{m}] \\
\{ -1 & \leq u \leq 1, \quad T_s = 1\text{s} \\
-1 & \leq x_2[k] \leq 1 \}
\end{align*}
\]

**Previous lecture:** the MPC Toolbox was employed to design a controller that brings the state of this system to the origin while fulfilling input and state constraints.

**Issue:** the computational time of the MPC is too high due to the numerical optimization routines

**Task:** design a NN that learns the MPC’s control law, enabling us to quickly compute the optimal control action
MPC-NN: Design Approach

1) Design MPC Controller

2) Design NN

3) Replace MPC with NN
Write a Matlab script that:

a) **generates data set** $D = \{(x_j, u_j)\}_{j=1}^{N_D}$ where

$$x_j \in \mathbb{R}^2 = \text{state sampled from the grid } \{-10, -9.8, -9.6, \ldots, 10\} \times \{-1, -0.9, -0.8, \ldots, 1\}$$

$$u_j = u_j^*(x_j) = \text{MPC's optimal control action with initial state } x_j$$

$N_D = \text{number of samples}$

b) **plots training data**

c) **trains a neural network** to learn the MPC's control law using the following settings

- 2 layers
- hidden layer: 20 neurons, "logsig" activation function
- output layer: 1 neuron, linear activation function

d) **plots neural network's fitting of training data**

e) **exports trained network** to Simulink and simulates the closed-loop response of plant with initial state $x[0] = [10 \quad 0]^T$

*Matlab implementation of MPC controller: i) previous lecture; or ii) https://tinyurl.com/tsfvdyp
MPC-NN: Matlab Example (II)

b) training data
d) NN fitting
MPC-NN: Matlab Example (III)

e) Plant response with NN control
Summary

Introduction
- Data sets, learning algorithms, candidate models

Single-Layer Neural Network
- Fundamentals: neuron, activation function, layer
- Matlab example: constructing & evaluating NN
- Learning algorithms
  - Batch solution: least-squares
  - Online solution: LMS

Multi-Layer Neural Network
- Network architecture
- Learning algorithm: backpropagation
- Overfitting & regularization

Key Matlab Functions
- linearlayer()
- configure()
- net.layer{i}.transferFcn=...
- sim()
- adapt()
- feedforwardnet()
- train()
- net.performParam.
  regularization=...
Consider the following system

\[ p \rightarrow h(p) \rightarrow n \rightarrow y \]

Write a Matlab script that:

- **a)** generates an artificial data set for this problem, i.e.
  \[ D = \{ (p_i, y_i) \} \]
  where \( p_i \) is an input sampled from the grid \( \{-1, -0.9, \ldots, 1\} \times \{-1, -0.9, \ldots, 1\} \)

- **b)** plots data set (hint use `plot3`)

- **c)** creates a neural network (NN) with 2 layers:
  - a) hidden layer with 50 neurons and activation function= hyperbolic tangent sigmoid
  - b) output layer with 1 neuron and activation function=linear

- **d)** trains the weights and bias of the NN to fit the data set \( D \)
  - Use all data for training; number of epochs = 2000

- **e)** plots training results, i.e. the data set \( D \) and the output of the trained NN

- **f)** adds a regularization term to the training (\( \lambda = 0.5 \)) and plots results

- input: \( p = [p_1, p_2]^T \in [-1, 1] \times [-1, 1] \)
- output: \( y \in \mathbb{R} \)
- function: \( h(p) = 10p_1^3 + 3p_2 + 7p_1p_2 \)
- random noise: \( n \sim N(0, \sigma^2), \sigma^2 = 2 \)
Homework: Expected Results

\[ \lambda = 0.0 \]

\[ \lambda = 0.5 \]