**Experiment 5 — Introduction to MathWorks Neural Network Toolbox & Simulink Demo**

**Aim**

To learn basic MATLAB NN apps (nnstart, nntool) and how to use a MATLAB-trained network inside Simulink.

**Theory (short)**

MATLAB provides GUI apps and functions to create, train and simulate neural networks. Simulink lets you place a trained network into a block diagram for system-level simulation.

**Software / Files**

* MATLAB + Neural Network Toolbox
* Simulink
* Example dataset: built-in simplefit\_dataset or instructor CSV plant\_data.mat

**Pre-lab checklist**

* Open MATLAB, run nnstart to see GUI.
* In Command Window type which simplefit\_dataset to ensure dataset availability.

**Step-by-step (student follow and execute each numbered step)**

**A. Quick training in MATLAB (command line)**

1. Open MATLAB. In Command Window run:
2. clear; close all; clc
3. [x,t] = simplefit\_dataset; % built-in dataset: x (1xN), t (1xN)
4. Create a simple feedforward net and train:
5. hiddenSize = 10;
6. net = fitnet(hiddenSize); % create network
7. net.divideParam.trainRatio = 0.7;
8. net.divideParam.valRatio = 0.15;
9. net.divideParam.testRatio = 0.15;
10. [net,tr] = train(net,x,t); % train
11. y = net(x); % simulate on training data
12. perf = perform(net,t,y); % compute performance (MSE)
13. view(net); % visualise network architecture
14. figure; plotperform(tr); % training performance
15. figure; plotregression(t,y);% regression plot
16. Save network:
17. save('exp5\_net.mat','net','tr');

**B. Prepare data for Simulink**

1. Convert inputs to a timeseries for Simulink:
2. N = length(x);
3. Ts = 0.1; % sample time
4. tvec = (0:N-1)\*Ts;
5. P\_ts = timeseries(x', tvec'); % Nx1 timeseries

(Note: x' transposes to column vector which From Workspace expects.)

**C. Build Simulink demo model**

1. In MATLAB type simulink → click **Create Model** → Save as exp5\_nn\_demo.slx.
2. From Library Browser add:
   * **From Workspace** (Sources) — set *Variable name* = P\_ts
   * **Neural Network** block (search in library: type “neural” in search) — drag it
   * **Scope** (Sinks)
   * **To Workspace** (Sinks) — set *Save format* = Array, *Variable name* = Y\_sim
3. Connect blocks: From Workspace → Neural Network → Scope and To Workspace.
4. Double-click Neural Network block → set the *Network* parameter to net (use exact variable name from workspace). Set sample time to Ts or -1 for inherited.
5. Set model stop time to tvec(end) and run the model (press Run).
6. Check Scope and Y\_sim in the MATLAB workspace. Compare Y\_sim with y from MATLAB training:
7. figure; plot(tvec, y, '-o', tvec, Y\_sim, '-x'); legend('MATLAB y','Simulink Y\\_sim')

**Observation Table (fill in during the experiment)**

| **Sample No.** | **Input x** | **Target t** | **MATLAB Output y** | **Simulink Output Y\_sim** | **Error (t - y)** | **Remarks** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| ... |  |  |  |  |  |  |

**Result (student writes)**

(Write final MSE, observations comparing MATLAB & Simulink outputs, any mismatch reasons)

**Viva Questions**

1. What is nnstart used for?
2. How do you get a trained network into Simulink?
3. Why might Simulink output differ from direct MATLAB output?

**Precautions / Hints**

* Ensure net exists in base workspace (run whos).
* If Simulink errors about dimensions, use timeseries or reshape arrays.
* If Neural Network block not found, search the Simulink library search bar.

**Experiment 6 — ANN Implementation (Command Line, supervised learning)**

**Aim**

Implement a feedforward ANN for function fitting. Train, test and analyze performance.

**Theory (brief)**

Feedforward networks learn a mapping from inputs to targets using backpropagation to minimize error.

**Pre-lab**

* Use simplefit\_dataset or load instructor data\_fit.mat that contains x and t.

**Step-by-step (student execute)**

1. Create a new script file lab6\_ann.m in your lab folder and copy the code below. Run the script.
2. % lab6\_ann.m
3. clear; close all; clc
4. % Load data
5. [x,t] = simplefit\_dataset; % x: 1xN, t: 1xN
6. % Create network
7. hiddenSize = 10;
8. net = fitnet(hiddenSize);
9. % Division of data
10. net.divideParam.trainRatio = 0.7;
11. net.divideParam.valRatio = 0.15;
12. net.divideParam.testRatio = 0.15;
13. % Training function (default trainlm usually)
14. % net.trainFcn = 'trainlm'; % Levenberg-Marquardt
15. % Train network
16. [net,tr] = train(net,x,t);
17. % Evaluate
18. y = net(x);
19. perf = perform(net,t,y);
20. fprintf('Final performance (MSE): %.6f\\n', perf);
21. % Plots
22. figure; plotperform(tr);
23. figure; plotregression(t,y);
24. figure; ploterrhist(t-y);
25. % Save
26. save('exp6\_net.mat','net','tr');
27. **Run script**: Press Run or type lab6\_ann in Command Window.
28. Note down:
    * Final MSE printed in Command Window.
    * Regression plot R-value (displayed on regression plot).
    * Error histogram shape (normal vs skewed).

**Student tasks (write in report)**

* Vary hiddenSize = 5, 10, 20 and record MSE & training times.
* Change data division ratios and note effect on test performance.

**Observation Table (example)**

| **hiddenSize** | **Train Time (s)** | **Train MSE** | **Val MSE** | **Test MSE** | **R (regression)** | **Remarks** |
| --- | --- | --- | --- | --- | --- | --- |
| 5 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |

**Viva**

1. Explain the role of hidden layers.
2. What does plotperform show?
3. How to avoid overfitting in ANN?

**Experiment 7 — NN Tool GUI (nntool) Implementation**

**Aim**

Use GUI (nntool) to import data, create, train and export a neural network without writing initial code.

**Theory (short)**

nntool and nnstart provide user-friendly GUI for common NN tasks; useful for learning and quick prototyping.

**Step-by-step (student follow)**

1. In MATLAB Command Window type:
2. nntool

This opens Neural Network Tool GUI.

1. **Import Data**:
   * Click *Import* → select x and t from workspace, or *Load* a .mat file.
   * Ensure inputs are in matrix form: inputs (features × samples), targets (targets × samples).
2. **Create a new network**:
   * Click *New* → choose *Fitting Network* (function fitting) or *Pattern Recognition* if classification.
   * Set hidden layers: e.g., [10] for single hidden layer with 10 neurons.
   * Click *Create*.
3. **Train**:
   * Click *Train* and wait. The GUI shows training progress and performance.
   * After training click *Test* to see regression plots and errors.
4. **Export**:
   * Click *Export* → give variable name net\_gui (or default). This exports the trained net to base workspace.
5. **Save Session** (optional):
   * Click *File* → *Save Session*.

**Observation to record**

* Training time, final MSE, validation performance.
* Paste screenshots of GUI windows into your report.

**Viva**

1. How to export a trained network from GUI to workspace?
2. Which training functions does NN GUI offer?

**Experiment 8 — Various Network Structures (Feedforward, RBF, Recurrent)**

**Aim**

Implement and compare different NN structures: feedforward, radial basis function (RBF), and simple recurrent networks.

**Theory (short)**

* **Feedforward** (fitnet/feedforwardnet) — general regression/classification.
* **RBF (newrb)** — fast for interpolation-like tasks; good for local approximation.
* **Recurrent (layrecnet, narx/net)** — handles sequence/time-series due to feedback loops.

**Step-by-step (student hand-on)**

**A. Feedforward (review)**  
Use code from Experiment 6.

**B. Radial Basis Function (RBF)**

1. Create a script lab8\_rbf.m:
2. clear; close all; clc
3. [x,t] = simplefit\_dataset;
4. goal = 0.0; spread = 1.0; MN = 25; % tune these parameters
5. net\_rbf = newrb(x,t,goal,spread,MN);
6. y\_rbf = net\_rbf(x);
7. perf\_rbf = perform(net\_rbf,t,y\_rbf);
8. fprintf('RBF perf (MSE): %.6f\\n', perf\_rbf);
9. figure; plotregression(t,y\_rbf);
10. Run and note perf\_rbf.

**C. Recurrent network (layrecnet)**

1. Make a time-series example script lab8\_rec.m:
2. clear; close all; clc
3. tvec = 0:0.1:6.2;
4. P = sin(tvec);
5. T = sin(tvec + 0.1); % shifted target
6. % prepare sequences (cell arrays)
7. Pcell = con2seq(P);
8. Tcell = con2seq(T);
9. net\_rec = layrecnet(1:2,10); % delays 1:2, 10 neurons
10. [net\_rec,tr] = train(net\_rec,Pcell,Tcell);
11. % simulate
12. y\_rec = net\_rec(Pcell);
13. y\_rec\_num = cell2mat(y\_rec);
14. figure; plot(tvec,T,'-o',tvec,y\_rec\_num,'-\*'); legend('Target','Network Output');
15. perf\_rec = perform(net\_rec,Tcell,y\_rec);
16. fprintf('Recurrent perf: %.6f\\n', perf\_rec);
17. Run and compare recurrent output vs feedforward.

**Student observations (write results)**

Create table:

| **Architecture** | **Parameters (neurons/delays)** | **Train Time** | **Train MSE** | **Test MSE** | **Comments** |
| --- | --- | --- | --- | --- | --- |
| Feedforward |  |  |  |  |  |
| RBF |  |  |  |  |  |
| Recurrent |  |  |  |  |  |

**Viva**

1. When prefer RBF vs feedforward?
2. What are feedback delays in recurrent nets?

**Experiment 9 — Training Algorithms (trainlm, traingd, trainbr comparison)**

**Aim**

Compare training algorithms: Levenberg–Marquardt (trainlm), Gradient Descent (traingd), and Bayesian Regularization (trainbr).

**Theory (brief)**

* trainlm — fast for moderate-sized networks (requires more memory).
* traingd — basic gradient descent (slower, may need many epochs).
* trainbr — Bayesian Regularization reduces overfitting by modifying performance function.

**Step-by-step (student run)**

1. Create script lab9\_trainalgos.m:
2. clear; close all; clc
3. [x,t] = simplefit\_dataset;
4. algos = {'trainlm','traingd','trainbr'};
5. results = struct();
6. for i = 1:length(algos)
7. net = fitnet(10);
8. net.trainFcn = algos{i};
9. tic
10. [net,tr] = train(net,x,t);
11. elapsed = toc;
12. y = net(x);
13. perf = perform(net,t,y);
14. results(i).name = algos{i};
15. results(i).perf = perf;
16. results(i).time = elapsed;
17. fprintf('%s -> perf: %.6f, time: %.2fs\\n', algos{i}, perf, elapsed);
18. % Save plots if required
19. figure; plotperform(tr); title(['Performance - ' algos{i}]);
20. end
21. Run the script. Record performance & training times for each algorithm.

**Observation Table**

| **Algorithm** | **HiddenSize** | **Train Time (s)** | **Train MSE** | **Val MSE** | **Test MSE** | **Overfitting (Y/N)** |
| --- | --- | --- | --- | --- | --- | --- |
| trainlm | 10 |  |  |  |  |  |
| traingd | 10 |  |  |  |  |  |
| trainbr | 10 |  |  |  |  |  |

**Student Tasks**

* Repeat with noisy data (add small Gaussian noise to t) and test which algorithm generalizes best.
* Change hidden layer size and comment when trainlm memory becomes heavy.

**Viva**

1. Why might trainbr be preferable on noisy datasets?
2. When would you not use trainlm?

**Experiment 10 — Application of NN to Control System (Simulink + MATLAB)**

**Aim**

Design an ANN controller for a simple plant and compare closed-loop responses with a standard PID controller.

**Theory (brief)**

A network can be trained to act as an inverse model or controller using I/O data from the plant. When embedded in Simulink, the network can provide control signals based on current/desired outputs.

**Assumptions & Plant**

For lab: use plant G(s)=1s+1G(s) = \frac{1}{s+1} (first-order) or the instructor’s supplied plant. We will use sample time Ts = 0.05.

**Procedure (step-by-step)**

**A. Generate training data in MATLAB**

1. Create script exp10\_data\_gen.m:
2. clear; close all; clc
3. Ts = 0.05;
4. t = 0:Ts:20;
5. % persistently exciting input (sum of sinusoids)
6. u = sin(0.5\*t) + 0.5\*sin(1.5\*t);
7. G = tf(1,[1 1]); % plant
8. y = lsim(G,u,t); % plant response
9. % Prepare dataset (1 x N)
10. P = u; % inputs (controller output -> plant input)
11. T = y; % desired plant output
12. save('exp10\_plantdata.mat','t','u','y','P','T','Ts');
13. Run the script. Verify P and T.

**B. Train ANN to map error/reference to control signal**  
(choose an approach: inverse model or direct controller. Here we train net to map desired output r(t) to u(t) for inverse model.)

1. Create exp10\_train.m:
2. clear; close all; clc
3. load('exp10\_plantdata.mat','P','T');
4. % We'll train net to model inverse: map T -> P (approx inverse model)
5. net = feedforwardnet(20);
6. net.divideParam.trainRatio = 0.7;
7. net.divideParam.valRatio = 0.15;
8. net.divideParam.testRatio = 0.15;
9. % scale inputs and targets (recommended)
10. [Pn, settingsP] = mapminmax(P);
11. [Tn, settingsT] = mapminmax(T);
12. % Train: Input = target (Tn), Target = control (Pn)
13. [net,tr] = train(net,Tn,Pn);
14. % Save net and scaling settings
15. save('exp10\_controller.mat','net','settingsP','settingsT')
16. Run exp10\_train.m.

**C. Build Simulink closed-loop**

1. Open Simulink, create new model and save as exp10\_control.slx.
2. Add blocks:
   * **Step** (or Signal Builder) — for reference r
   * **Sum** block — to compute e = r - y (if training as direct controller may use error)
   * **Neural Network** block — set *Network* to net (from exp10\_controller.mat loaded into workspace)
   * **Transfer Fcn** block — 1/(s+1) as the plant
   * **Scope**, **To Workspace**
3. If scaling used:
   * Implement scaling/unscaling in Simulink using MATLAB Function blocks or Gain + Bias (use mapminmax parameters saved earlier — you can create a small MATLAB function to scale/unsclae or use the Neural Network block’s built-in preprocessing if available).
   * Alternatively, train on unscaled data if Simulink scaling is complicated (but be careful with large values).
4. Connect blocks in closed loop: Reference r -> Neural Network -> Plant -> Output y -> Feedback to Sum.
5. Run simulation and observe plant output and control signal.
6. For comparison, add a **PID Controller** block (from Simulink Library Simulink -> Continuous -> PID Controller) in a parallel model and compare rise time, overshoot and settling time.

**Measurements to record**

* Rise time (0–100% or 10–90%), overshoot (%), settling time (±2% band), steady-state error.
* Tabulate results for NN controller vs PID.

**Table — Control performance**

| **Controller** | **Rise time (s)** | **Overshoot (%)** | **Settling time (s)** | **Steady-state error** | **Remarks** |
| --- | --- | --- | --- | --- | --- |
| NN |  |  |  |  |  |
| PID |  |  |  |  |  |

**Handy MATLAB snippets students will use often**

**Convert matrix to timeseries**

% x is 1xN

Ts = 0.1;

tvec = (0:length(x)-1)\*Ts;

P\_ts = timeseries(x', tvec'); % for From Workspace block

**Check variable in base workspace**

whos

**Load saved network into base workspace**

load('exp6\_net.mat'); % brings 'net' and 'tr' into workspace

**Scale data with mapminmax**

[Pn, settingsP] = mapminmax(P);

[Tn, settingsT] = mapminmax(T);

% To reverse: P = mapminmax('reverse',Pn,settingsP);