Introduction to NETLAB

NETLAB is a Matlab toolbox for experimenting with neural networks (and many other things)

Available from:
www1.aston.ac.uk/eas/research/groups/ncrg/resources/netlab/

Installation: follow instructions from the site:

1. get three files: netlab.zip, nethelp.zip, foptions.m,
2. unzip them,
3. move foptions.m to the netlab directory,
4. install Matlab path to the netlab directory

Key features

- Gaussian mixture model with EM training algorithm
- Linear and logistic regression
- Multi-layer perceptron with linear, logistic and softmax outputs and appropriate error functions
- Radial basis function (RBF) networks with both Gaussian and non-local basis functions
- Optimisers, including quasi-Newton methods, conjugate gradients and scaled conjugate gradients
- K-nearest neighbour classifier
- K-means clustering
- Numerous demos (demnlab.m, dem*.m)
- …..

Example1: Building Mixture Models

```
%initialize the model
ncentres = 2;
input_dim = 1;
mix = gmm(input_dim, ncentres, 'spherical');
mix.centres=[50; 60]; %manual initialization

% Print out the initial model
disp('    Priors        Centres          Variances')
disp([mix.priors' mix.centres mix.covars'])

% Set up vector of options for EM trainer
options = zeros(1, 18);
options(1)  = 1;        % Prints out error values.
options(14) = 10;       % Max. Number of iterations.
```


Training Mixture Models

```
%Loading data (only x)
load mix2 x

%Training:
[mix, options, errlog] = gmmem(mix, x, options);

% Printing
disp('    Priors        Centres          Sigmas')
disp([mix.priors' mix.centres mix.covars.^0.5'])

%Other important functions (use ‘help fname’ to learn more):
% gmm
% gmminit
% gmmem
% gmmactiv, gmmprob
```


How to learn Netlab?

1. Run demos: >> demnlab
2. Look at the list of demos: >> help netlab (or doc netlab)
3. Study demo scripts, e.g.: >> edit demgmm2
4. Run them manually, step-by-step, from the editor
5. Use ‘help’ (‘doc’) to get more info about unknown functions
6. Modify the scripts as you wish (changing their names!)

Key demos to study:

demmgmm2: mixture model
demglm1: generalised linear model
demmlp2: multi-layer perceptron

Functions for Gaussian Mixture Models

mix = gmm(nin, ncentres, covtype)
create a nin-dimensional mixture model with ncentres components; covtype may be ‘spherical’, ‘diagonal’, or ‘full’

mix = gmminit(mix, x, options)
initialise the mixture model by clustering data x with help of the k-means clustering algorithm.

[mix, options, errlog] = gmmem(mix, x, options)
apply the EM algorithm to find mixture parameters

a = gmmactiv(mix, x)
apply the mixture model to data x (calculate activations, one per component)

Functions for single layer networks (GLM)

net = glm(nin, nout, func)
create a generalised linear model; func may be
'linear' (linear regression),
'logistic' (logistic regression), or
'softmax' (modeling probabilities)

y = glmfwd(net, x)
apply the model to data x

error = glmerr(net, x, t)
calculate the error

net = glmtrain(net, options, x, t)
train the network using the iterative reweighted least squares (IRLS) algorithm

Functions for Multilayer Perceptrons

net=mlp(nin, nhidden, nout, act_function)
creates a structure "net" , where

- nin=number of inputs
- nhidden=number of hidden units
- nout=number of outputs
- act_function= name of activation function for the output layer (a string):
'linear', 'logistic', or 'softmax'

Hidden units always use hyperbolic tangent ("logistic sigmoid scaled to (-1, 1)")

Limitation: only one hidden layer is implemented

Functions for Multilayer Perceptrons

>> net=mlp(2,3,1,'logistic')

creates a structure "net" with the following fields:

type: 'mlp'
nin: 2
nhidden: 3
nout: 1
nwts: 13
outfn: 'logistic'
w1: [2x3 double]
b1: [-0.2661 -0.0117 -0.0266]
w2: [3x1 double]
b2: 0.3873

Functions for Multilayer Perceptrons

We may access and modify all the fields using the "." operator.

E.g.:
>> a=net.w1

a =
0.5107 0.7603 0.8469
1.4655 0.8327 -0.6392

>>net.w1=rand(2,3);

Functions for Multilayer Perceptrons

a=mlpfwd(net, x)
applies the network net to input data x (input patterns are rows of x; x has to have net.nin columns)

error=mlperr(net, x, t)
calculates error of network net, applied to input data x, and desired outputs (targets) t

type 'help mlpfwd' and 'help mlperr' for more options!

Training MLP

[net, options] = netopt(net, options, x, t, alg)
trains the network net on trainig data x (inputs) and t (targets), using some options (a vector of 18 numbers) and a learning algorithm alg

Most important training algorithms:
'graddesc': gradient descent (backpropagation)
'quasinew': quasi-Newton optimisation
'scg': scaled conjugate gradients (very fast)

[net, error] = mlptrain(net, x, t, iterations)
An 'easy' training function using scg optimisation procedure

Options for 'graddesc'

The "options" argument is a vector of 18 numbers:

- options(1) is set to 1 to display error values;
- options(2) is the absolute precision required for the solution (stop criterion)
- options(3) is a measure of the precision required of the objective function (another stop criterion)
- options(7) determines search strategy.
  - If it is set to 1 then a line minimiser is used.
  - If it is 0 (the default), then each parameter update is a fixed multiple (the learning rate) of the negative gradient added to a fixed multiple (the momentum) of the previous parameter update (backpropagation)
- options(14) is the maximum number of iterations; default 100.
- options(17) is the momentum (alpha); default 0.5.
- options(18) is the learning rate (eta); default 0.01.

Additional links

softmax:

the outputs of the net should sum-up to 1,
so they can be interpreted as probabilities

http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-12.html

IRLS: Iteratively Reweighted Least Squares

http://en.wikipedia.org/wiki/IRLS

Line search and other optimisation methods:
Chapter 10 from "Numerical Recipes“,

http://www.nrbook.com/c/