Datamining and Neural Networks: Exercise Session 1

In this session basic principles of multilayer perceptrons for nonlinear function estimation are illustrated using the MATLAB Neural Network Toolbox. The code in this document was written for MATLAB version R2010b or later.

Feel free to contact us if you still have questions after consulting the documentation and/or your colleagues. Our email addresses are lynn.houthuys@esat.kuleuven.be and joachim.schreurs@esat.kuleuven.be.

1 Function Approximation (noiseless case)

Run the command `nnd11gn` which opens a window in which you can investigate the relationship between the number of hidden neurons and the complexity of the function to approximate. Try several possible combinations of difficulty index and number of hidden units. In which cases one has underfitting and overfitting (and why) ?

2 The role of the hidden layer and output layer

In linear regression the relation between an n-dimensional input and a univariate output pattern is modelled by a linear relation of the form

$$y_p = w_1 x_{1,p} + w_2 x_{2,p} + \ldots + w_n x_{n,p} + \beta.$$

The parameters $\{w_i\}_1^n$ and β are found by minimizing the sum of squared errors

$$\sum_{p=1}^{P} \left(y_p - (w_1 x_{1,p} + w_2 x_{2,p} + \ldots + w_n x_{n,p} + \beta)\right)^2.$$

1. Given a set of input and output patterns $\{(x_p, y_p)\}_1^P$, how would you solve the linear regression problem using a neural network? Outline the architecture of the network. Specify and motivate the choices you make for

   (a) the number of layers required to learn this function,

   (b) the number of neurons in each layer,

   (c) and the transfer functions used in each layer.

2. Create a vector if 21 input patterns, equally spaced on the closed interval $[0, 1]$.

3. Use the input patterns from (2), to create output patterns given by the following relationship:

   $$y_p = -cos(0.8\pi x_p).$$

   Plot the value of the output patterns against the input patterns. Will a linear model adequately capture the relationship between x_p and y_p? How will a network of the architecture chosen in (1) perform on this data set?

4. Train a neural network with one hidden layer containing two neurons on this data set. Use the following code to train your neural network (in which `x` and `y` contain the input and ouput patterns respectively)

   ```
   net = fitnet(2);
   net = configure(net,x,y);
   net.inputs{1}.processFcns = {};
   net.outputs{2}.processFcns = {};
   [net, tr] = train(net,x,y);
   ```

   Lines two, three and four prevent the network from rescaling the inputs and outputs.

5. Obtain the activations x_p^1 and x_p^2 of the first and second hidden neuron respectively when presented with input pattern x_p. Do this for every input pattern.

   Hint: You may use the functions `hidden_layer_weights` to obtain the biases and weights of the hidden neurons and `hidden_layer_transfer_function` to obtain the activation function of the hidden layer. These functions are available from Toledo and should be placed in your current directory. For example, to obtain the biases and weights of the hidden neurons use:

   ```
   [biases, weights] = hidden_layer_weights(net);
   % biases(i) and weights(i) contain the bias and
   % weight for the i-th hidden neuron
   ```

6. Plot the values of y_p, x_p^1 and x_p^2 against p. How would you model the relationship between y_p and (x_p^1, x_p^2)?

7. The learnt weights, bias and transfer function of the output neuron define a function together. Use this function to define $output_p$, the output of the network for each pattern p. plot $output_p$ against the input patterns and compare this to the plot in (3).

   Hint: You may use the functions `output_layer_weights` to obtain the bias and weights of the output neuron and `output_layer_transfer_function` to obtain the activation function of the output layer.

3 Function Approximation (noisy case)

Generate a training set of a cosine wave (data scaled between -1 and $+1$ on each axis) with Gaussian distributed noise. Investigate the obtained results in terms of the following choices:

- Size of the training data set (e.g. 30, 150, 1200 data points).
- Amount of noise (`randn`) on the training data (standard deviation 0.2, 0.6, 1.2).
- Number of hidden units in the case of multilayer perceptron (MLP) with one hidden layer.
- Training algorithms: backpropagation, conjugate gradient, quasi-Newton, Levenberg-Marquardt.
- Training until a local minimum is reached versus early stopping on a validation set.
- Cost function with or without regularization. Vary the regularization constant.
- Choice of initial weights.

The following examples will help you in this investigation:

Example 1: MLP with 5 hidden units trained by scaled conjugate gradient and early stopping on a validation set. Both training and validation sets are contaminated by normally distributed noise with a standard deviation of 0.5.

```
%% Create a training and validation set
train_x = linspace(-1,1,100);
train_y = cos(2*pi*train_x) + 0.5*randn(size(train_x));
val_x = linspace(-0.9,0.9,100);
val_y = cos(2*pi*val_x) + 0.5*randn(size(val_x));
x = [train_x val_x];
y = [train_y val_y];
%% Create a network with 5 hidden neurons
net = fitnet(5, 'trainscg');
net.divideFcn = 'divideind';
net.divideParam = struct('trainInd', 1:100, ...
                         'valInd', 101:200, ...
                         'testInd', []); % no test set
[net, tr] = train(net, x, y);
%% Get approximated function on training set
train_yhat = net(train_x);

plot(train_x, train_y, 'r*');
hold on;
plot(train_x, train_yhat, '-');
plot(train_x, cos(2*pi*train_x), 'g-');
hold off;
legend('Training Set', 'Approximated Function', 'True Function');
```

Example 2: Cost function with regularization. `net.performParam.regularization` can take on any value between 0 and 1 inclusive. The default value of 0 corresponds to no regularization, 1 will yield a network with all weights and biases set to zero.

```
%% Set the regularization parameter
net = fitnet(40, 'trainscg');
net.performParam.regularization = 0.1;
net.divideFcn = 'divideind';
net.divideParam = struct('trainInd', 1:100, ...
                         'valInd', 101:200, ...
                         'testInd', []); % no test set
[net, tr] = train(net, x, y);
```

4 Curse of dimensionality

Consider the function $f : \mathbb{R}^m \mapsto \mathbb{R}$ given by

$$f(x) = \text{sinc}\left(\sqrt{\sum_{i=1}^{m} x_i^2}\right)$$

where

$$\text{sinc}(t) = \begin{cases} \frac{\sin(\pi t)}{\pi t} & t \neq 0 \\ 1 & t = 0 \end{cases}$$

Figure 1 shows the bivariate version of f.

Try to find a good approximation of the nonlinear mapping from $x \in \mathbb{R}^m$ to $y \in \mathbb{R}$. Proceed hereby as follows:

1. Consider the cases $m = 1$ (one-dimensional sinc function), $m = 2$ (mexican hat function), $m = 5$.
2. Generate a suitable and meaningful training set, which has to be chosen in such a way that you obtain a good approximation (over a certain compact interval). Also decide about a meaningful test set.
3. For each of the cases with different number of inputs m, visually check the neural net approximation by plotting the approxmiated function against $\text{sinc}(r)$.
4. Which training methods are most suitable in each of the cases? What is the model complexity (number of hidden units) that you need in the different cases?
5. How does the increase in dimensionality effect the number of neurons?

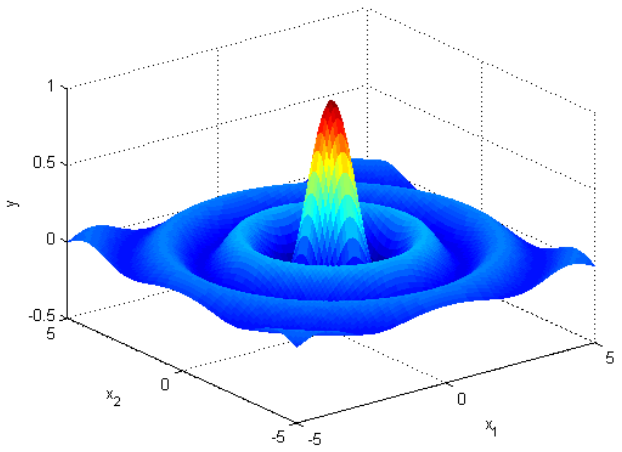

Figure 1: Plot of $\text{sinc}(\sqrt{x_1^2 + x_2^2})$ for $-5 \leq x_1, x_2 \leq 5$.