Lab assignment 4: Training multi-layer networks

- Task 1: Error back-propagation, feedforward layered networks
- Task 2: Autoencoder

As usual, describe everything in a report. If you don't know how to structure it, you may follow the guidelines.

Task 1: Error back-propagation, feedforward layered networks

Write a function implementing the error back-propagation algorithm for training a feedforward layered network (multilayer perceptron) with one hidden layer.

The function should receive three input arguments: a training set, a corresponding target, and a number of units in the hidden layers, nh.

The number of inputs is fixed: it is the dimension of input vectors (patterns in the training set), plus one. The number of outputs is also fixed: it is the dimension of output vectors.

As outputs, the function should return the two weight matrices, whi and woh.

Therefore in Matlab the function should be defined as:

```
function [whi woh] = backprop(x,t,nh)
```

You may use either the sigmoid or the hyperbolic tangent as activation function; the advantage of the latter is that it is already available in Matlab (function tanh).

We want to train the network with the "training-by-epoch" strategy: at each pattern we compute a weight update (for all weights in both layers), but we sum it into two cumulative updating vector (e.g., dwhi and dwoh), then we apply the weight updates whi=whi+dwhi and woh=woh+dwoh only at the end of a training epoch, i.e., after we have processed all patterns in the training set.

The cost function, as seen for the Adaline and as seen for the error back-propagation algorithm, will be the Mean Squared Error (mse) defined as:

```
sum{l=1:npatterns}sum{k=1:no}(output_kl-target_kl)^2
```

where output_kl is the value of so(k) computed starting from pattern x(l,:), and target_kl is the corresponding target.

Skeleton pseudo-code:

```
%init
assign parameters nepochs, ni, no, npatterns, eta,
initialize weights whi and woh (random in-1,+1),
initialize activations sh and so (zeros)
add column of ones to training set and to hidden units sh
%main loop
for t=1:nepochs
  initialize mse (zero)
  for l=1:npatterns
    % feedforward step
    compute sh and so
    % back-propagation step
    compute deltao and dwoh
    compute deltah and dwhi
    update mse=mse+mean((so-t(l,:)).^2) (mean of all outputs)
  end
  apply weight update woh=woh+dwoh, whi=whi+dwhi
  compute final mse = mse/npatterns
end
%finalizations, if necessary
```

You may provide printouts to check the progress of training. For instance, it may be interesting to see how the mse is proceeding.

Make also a stripped-down version of the function that is used to test a trained network: delete everything but the feedforward step and the computation of the mse; read as input parameters a data set, whi and woh (already trained), and return as output the network outputs (vector so) for each pattern. You may call this function mlptest (multi-layer perceptron test).

Test the algorithm with simple data first (for debugging you may use a toy problem), then try it with the Semeion data set.

Task 2 (optional): Autoencoder

The simplest **autoencoder network** has one input layer, one hidden layer (nh<ni), and one output layer (no=ni). It is trained using **the same pattern as both the input and the target**. So for instance input pattern `x(l,:)` has target `x(l,:)` (the same as the input).

Note that, although we have used neural networks only for classification so far, in this case we don't have any classes to learn. This is a special case of **unsupervised training**. In fact, it is sometimes called "self-supervised", since the target we use is the input pattern itself.

Use your backpropagation function to train a multilayer perceptron as an autoencoder for the Semeion data. Training an autoencoder only amounts to using a multi-layer perceptron and error back-propagation for data prepared in a special way (target = input); it is not a different neural network algorithm.

An autoencoder learns an **internal, compressed representation** for the data. The interesting output, therefore, is the value of its hidden layer. What we hope is that for similar patterns we will have similar representations; for instance, we hope that images representing a "1" will correspond to very similar representations, and quite similar to "7" but different from "0" or "8". In other words, that the network will **learn the**

classes.

To check this, make a new version of mlptest (say, "autoencode") that outputs the hidden layer (sh) instead of the output. This even-more-stripped-down version does not even need the second layer; just whi is enough to compute sh. So autoencode is obtained from mlptest mostly by deleting even more code.

Experiment with different numbers of hidden units and inspect the values of the hidden layer to see whether any regularity appears.

UPLOAD BELOW YOUR CODE, DATA, AND REPORT

Submission status

Submission status	Submitted for grading
Grading status	Not graded
Due date	Sunday, 24 November 2013, 11:55 PM
Time remaining	Assignment was submitted 42 mins 25 secs late
Last modified	Monday, 25 November 2013, 12:37 AM
File submissions	Ahmed_LAB4.rar