Programming task 5

Radial Basis Function Networks
And
Genetic Algorithms
For
System identification and Filter design

<u>Radial Basis Function Networks</u>

In the first part of this task we wish to use a modular neural network, the Radial Basis Function (RBF) network as an alternative for feedforward backpropagation neural networks. These networks consists of a Kohonen network in the first layer, with Gaussian (Radial basis) functions at the outputs, and an output layer of linear neurons. The total network works in a supervised manner, the training algorithm is the ordinary MIMO-LMS, since the output layer is linear. The RBF is a universal mapping machine, which means that it can approximate any multivariate function with arbitrary accuracy (provided we have enough neurons in the Kohonen network).

a) We will use the Radial Basis Function network in order to approximate the same ”sofa” shaped function as in the first task. The function is defined as:

$$f(x_1,x_2)=\sin(\tfrac{x_1^2}{4}+\tfrac{x_2^2}{2}) \quad ,x_1\in[-1,3], x_2\in[-3,3]$$

- Create the training sequence by sampling the region where the inputs are specified and use the specified function to create the target sequence.

OBS: Reuse the training sequence from task 1.

From the training sequence we can determine in which region the inputs lie, i.e. the specification domain of the function: $x_1\in[-1,3], x_2\in[-3,3]$. For the Kohonen network each neuron handles a part of the input specification domain, and since we know that this domain is uniformly distributed we can distribute the neuron weights (the centers of attraction) uniformly in the input space.

- Make three different Kohonen networks, and choose the number of neurons in these networks as: 25, 100, 400. Use the Gaussian function, defined in the lectures, at each output.
- Choose the weights for the Kohonen network uniformly distributed inside the input space domain.
- In this task we will keep the weights in the first layer the same all the time, i.e. no training of the Kohonen network is needed.

OBS: If the input vectors are non-uniformly distributed we need to train the Kohonen network with the Self-Organizing Feature Map (SOFM) algorithm. This will cause the neuron weights to be non-uniformly distributed to match the inputs, which in turn will lead to increased accuracy at regions where the inputs are dense.

- Add a linear layer of neurons (in our case it will be one neuron with 26, 101, 401 weights, respectively) to each network. Use biases for the neurons.
- Train the RBF networks with the training sequence and plot the error during training. Some averaging of the error can be appropriate for the plot. Use an appropriate stopping criterion.
- What impact does the number of neurons in the first layer have on the convergence rate?
- Is the convergence rate higher with RBF:s than with the backpropagation algorithm?
- Make a plot of the function using the RBF networks after convergence. Use inputs uniformly distributed in the input space.
- What kind of impact does the number of neurons in the first layer have on the final approximation performance?
- Since we have fixed the neurons weights in the Kohonen network we can analytically find the least square weights at the linear output layer, from the normal equations. Create a new training sequence by letting the original input vectors pass the Kohonen network and the non-linear functions and keep the same target sequence (this only has to be done for the network with 400 neurons). Use this new training sequence to find the optimal least square weights for the output layer, by solving the normal equations.
- Make a plot of the function using this optimal RBF network. Is the approximation performance better than for the one found by training?

Genetic Algorithms

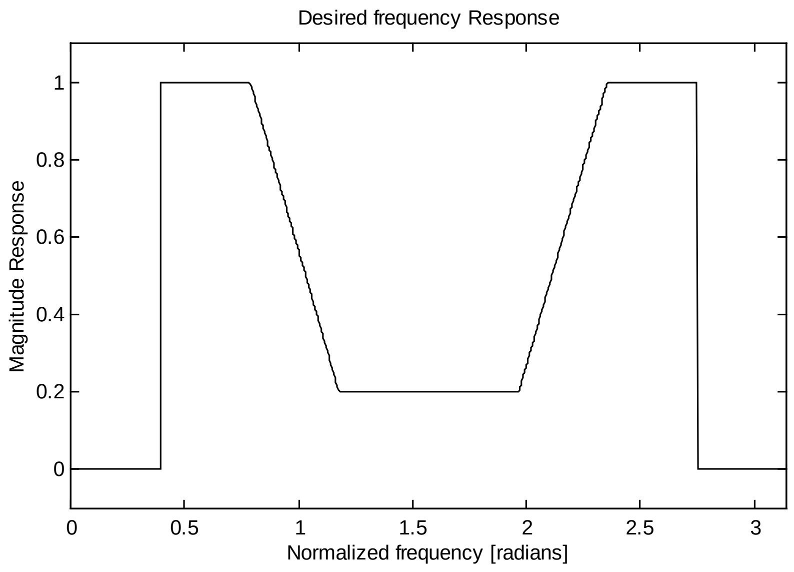
Genetic Algorithms are optimization tools. As such we can use them in any supervised neural network (we can also use them in some of the unsupervised neural networks, as long as we have an error criteria). In this part we wish to create an FIR filter with an arbitrary transfer function by the use of Genetic Algorithms (GA:s). The FIR filter should be a linear phase filter of odd length. This will give symmetry in the filter taps.
OBS: We do not have to restrict the filter in any way when using GA:s, but in this task we do it for simplicity.

b) We will use a Genetic Algorithm where each parent gives birth to 10 children as a new population. We use the exponential decrease function to weight the suitability of becoming a new parent. When initiating the first parent use a pure delay of half the FIR filter length, i.e. an impulse at position (L-1)/2 of the FIR filter taps and the rest of the taps at zero.

- Create a function where a desired frequency response and the length of the FIR filter are taken as input parameters. The function should then find the least square or the minimax approximation to the desired transfer function (use an extra input parameter which decides the approximation criteria) with a Genetic Algorithm. When finding the least square- or the minimax-filter, the only thing that has to be modified is the definition of the error function.
- Run the Genetic Algorithm with the desired frequency response chosen according to the figure below, when using both the Least Square and the Minimax approximation.

OBS: When using the Minimax error criteria one should allow for transition regions where the desired frequency response has abrupt changes. Omitting some points around these regions when sampling the frequency does this. Otherwise the maximum error will always appear at these regions.

Use the FIR filter lengths: 33, 65, 129. The stepsize and the probability of change, p_i, should be decreased during learning. The constant in the exponentially weighting can be set to 0.1. Sample the frequency region at 1024 points in order to calculate the error function. Use an appropriate stopping criterion.



OBS: The desired response is chosen as:
R=[zeros(1,128) ones(1,128) linspace(1,0.2,128) 0.2*ones(1,256) linspace(0.2,1,128) ones(1,128) zeros(1,128)];

- Make a plot of the error during learning, some averaging can be appropriate. What do you think about the convergence rate? Is it acceptable for filter design?
- Make a linear plot of the frequency response from each filter and the desired response in the same plot, after convergence.
- Choose any desired ”odd shaped” filter response and use the GA to find an approximation.
- Make a plot of the FIR filter response and the desired response.
- Adjust the stepsize and the probability factor and make a comment of the sensitivity of these choices.