# Feedforward Neural Networks as Radial Basis Function Networks

Edward Venator

EECS 484

## Introduction

In this assignment, a mapping was created from articulated arm joint angles to arm position in the x dimension. This was accomplished using a radial basis function network.

Radial basis function networks allow network responses to be constructed out of weighted sums of arbitrary functions. Although an arbitrary activation function is not biologically plausible, some activation functions can be constructed of several layers of neurons with biologically realistic sigmoid activation functions. In this assignment, Gaussian activation functions were approximated using two layers of neurons with sigmoid activation functions. The first layer, called the alpha layer, consists of perceptrons with hyperbolic tangent activation functions. The hyperbolic tangent function is a sigmoid function with a range of -1 to 1. The second layer, called the beta layer, sums the outputs of the alpha layer using weights of -1 and 1. These weights are selected so that the each beta node has a maximum at one point that slopes off in all directions. By appropriately biasing these nodes, the output closely resembles a Gaussian radial basis function. Once the beta nodes are trained, the third layer is a single node with linear activation function, just as in a normal radial basis function network. The weights of this output (gamma layer) node can be set algebraically with the pseudoinverse, or biologically by random perturbations. In this assignment, the algebraic method was used while tuning the alpha and beta layers, and then a biologically plausible algorithm was created.

## Alpha Layer Bias Selection

Professor Newman’s notes on radial basis function networks describe a method for biasing the alpha nodes of a network of this type. However, his notes make the assumption that the inputs to the system are distributed across the range -1 to 1. Since these inputs do not have that distribution, I rescaled them using Equation 1 below. This rescaling could be incorporated into the weights of the alpha nodes, but for simplicity’s sake, I just rescale the inputs at the beginning of my Matlab script.

|  |  |
| --- | --- |
|  | 1 |

Once the inputs are rescaled, the alpha node biases are generated as described in Poressor newman’s notes. The weights from the inputs are chosen as to be random numbers evenly distributed from -1 to 1. Each bias weight is then chosen to be a random number evenly distributed between where i is an index corresponding to an input (in this case i counts from 1 to 2) and w is the weight from that input into the alpha node. Figure 1 shows surface plot outputs of several alpha nodes, with scaled inputs superimposed in blue. Dark red areas correspond to areas rejected by the perceptron (-1), and blue corresponds to areas accepted by the perceptron (1).

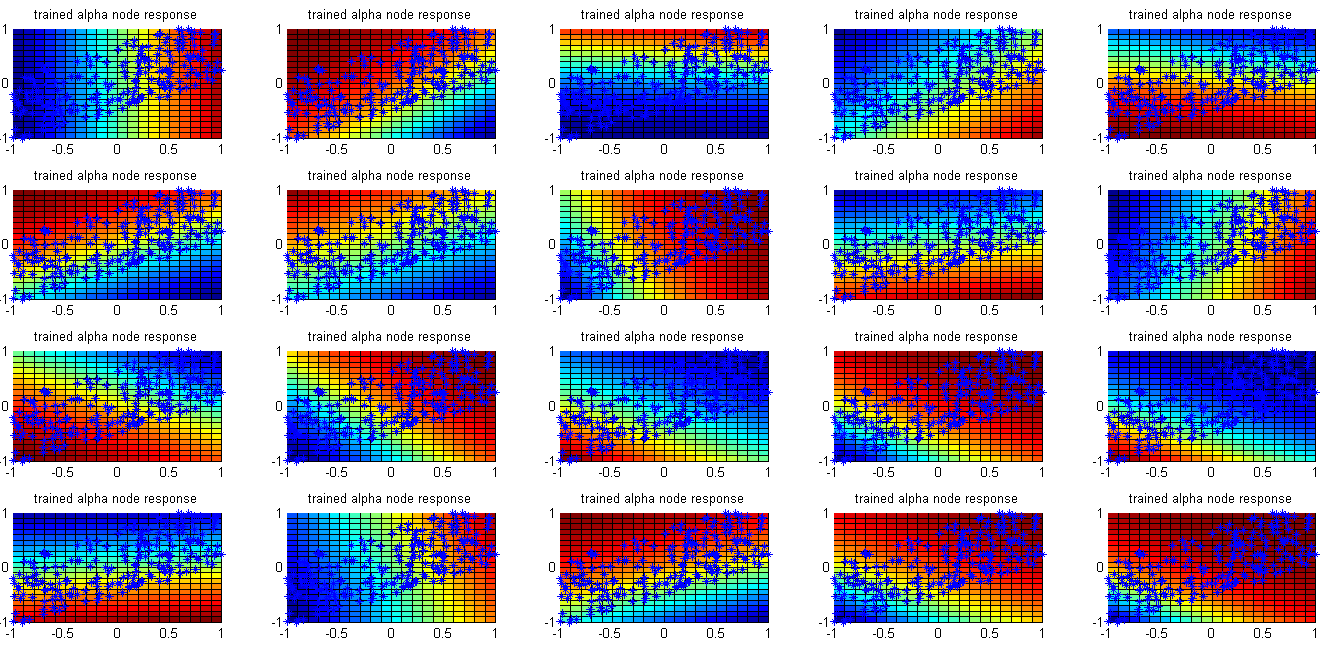


Figure 1

One important parameter to tune in this case is the number of alpha nodes. I initially used a few thousand alpha nodes, but with some tuning determined that as few as 100 alpha nodes could still produce good looking radial basis functions in a properly-tuned beta layer. Fewer alpha nodes of course makes the network faster to simulate, and would make a physical (or biological) implementation of the network more compact.

## Beta Layer Bias Selection

Biasing the beta layer proved trickier than biasing the alpha layer.

## Gamma Node Bias Selection by Random Perturbations

## Conclusions