**Engineering and Applied Science Programs for Professionals**
**Whiting School of Engineering**
**Johns Hopkins University**
**685.621 Algorithms for Data Science**
**Neural Networks**

This document provides a rollup of neural networks. In this module the development of single layer networks are described. This is expanded for the use of classifiers as two-class classifiers, which includes the radial basis function and the probabilistic neural networks.

# Contents

# 1 Neural Networks

Machine learning for a classification task involves training over a set of samples $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]$ where $\mathbf{x} \in \Re^D$. Where the symbol $\Re$ is used to represent the $n$-dimensional space the features in $\mathbf{x}$ resides. Each sample in the training set contains one target value $\mathbf{C} = C_j = [C_1, C_2, \cdots, C_c], j = 1, 2, \cdots, c$, (known as the class labels $y_i \in \mathbf{C}, i = 1, 2, \cdots, m$) which describes the class to which the sample is a member of. The objective is to separate the data into their classes such that the degree of association is strong between the data sets of the same class and weak between members of different classes. From the class separation, an unseen sample $\mathbf{x}_0 \in \Re^D$ can then be appropriately classified. In this document two neural networks are introduced, first is the Radial Basis Function Neural Network (RBF NN) followed by the Probabilistic Neural Network (PNN).

# 2 Radial Basis Function Neural Networks (RBF NN)

The radial basis function neural network is a type of feedforward neural network that differs from the the multilayer perceptron in the activation function [4]. The RBF NN contains three very unique characteristics:

- Contains a hidden layer of branching nodes representing the input features from an observation.
- The hidden layer contains a special type of activation functions centered on the center training vector $\mathbf{x}$ of a cluster in the feature space allowing the function to have a non-negligligible response for the input vector $\mathbf{x}_0$ close to the center.
- An output layer of nodes that sum the outputs from the hidden layer. This layer uses an activation function to determine the associated class. A linear activation function is typically used for the output layer.

Viewing the Figure 1 gives a visual representation of the general RBF NN. The bias term $\mathbf{b}_C$ at each output node assures nonzero mean values of the sums

$$\hat{y}_j = \hat{w}_j^{(1)} f^{(1)}(\mathbf{x}, \mathbf{w}^{(1)}) + \cdots + \hat{w}_j^{(m)} f^{(m)}(\mathbf{x}, \mathbf{w}^{(m)}) + b_j \tag{1}$$

where $j = 1, 2, \cdots, c$
The most common RBFs are:

$$\mathbf{x} \to \mathbf{f}^{(M)}(\mathbf{x}; \mathbf{w}^{(M)}) = \exp\left[\frac{-\|\mathbf{x} - \mathbf{w}^{(M)}\|^2}{2\big(\sigma^{(M)}\big)^2}\right] \tag{2}$$

and $M = 1, \ldots, m$ with center at $\mathbf{w}^{(M)}$. Note that $\mathbf{y}^{(M)}$ is maximized when $\mathbf{x} = \mathbf{w}^{(M)}$. We usually use a hidden neurode for each exemplar input feature vector $\mathbf{x}^{(Q)}, Q = 1, \ldots, q$, so we put $M \leftarrow Q$ in this case. The center vector $\mathbf{w}^{(M)} = \big(w_1^{(M)}, \ldots, w_d^{(M)}\big)$ at the $m$th hidden neurode has $D$ components to match the input feature vector. The total number of hidden nodes is denoted by the superscriot of $(M)$, where each of the functions, $\mathbf{f}^{(M)}$, is designed to have an influence on the cluster of points that is closest to the vector $\mathbf{w}^{(M)}$. The number of clusters used for modeling is given by $M$ and can be considered as a hyper-parameter of the algorithm. The parameter $\sigma_M$ in Equation 8 is used to control the spread of the radial basis function so that its values decrease more slowly or more rapidly as $\mathbf{x}$ moves away from the center vector $\mathbf{w}^{(M)}$ - that is, as $\|\mathbf{x} - \mathbf{w}^{(M)}\|$ increases.

The activated values $\mathbf{f}^{(M)}(\cdot)$ are summed to yield a network output $\hat{\mathbf{y}}_J$ shown in Figure 1 and determined by either of the following eqations:

$$\hat{\mathbf{y}}_C = \frac{\left[\sum_{i=1}^{M} \hat{w}_j^{(m)} f^{(i)}(\mathbf{x}, \mathbf{w}^{(i)})\right]}{\left[\sum_{m=1}^{M} f^{(i)}(\mathbf{x}, \mathbf{w}^{(i)})\right]} + \mathbf{b}_C \tag{3}$$

or

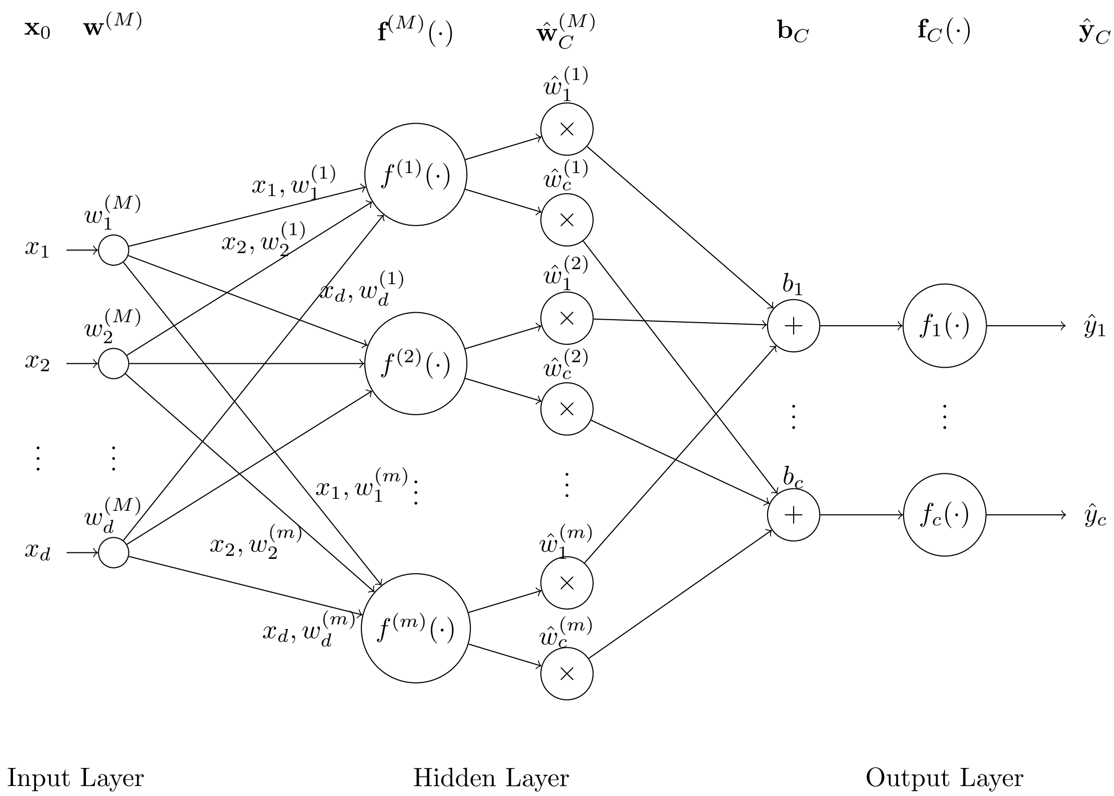


Figure 1: Simple radial basis function neural network example with $M$ nodes at the hidden layer and $J$ node at the output layer.

$$\hat{y}_j = \frac{1}{M}\left[\sum_{i=1}^{M} \hat{w}_j^{(m)} f^{(i)}(\mathbf{x}, \mathbf{w}^{(i)})\right] + b_j \tag{4}$$

allowing for the output to be normalized.

Each of the RBFs will have influence on the region of the feature space it is a part of. The important area of the feature space where each observation are clustered is covered by the $M$ RBFs that are centered in the cluster of observation feature vectors that represent the classes of interest.

Training of the RBF NN consists of at least four steps:

- Assign each weight vector in $\mathbf{w}^{(M)}$ a unique training vector from $\mathbf{x}^{(Q)}$, $\mathbf{w}^{(m)} \leftarrow \mathbf{x}^{(q)}$.
- Select the parameter of the spread, $\sigma_m$, for this class the selection is done experimentally, it should be noted, when more than one model is trained, each training model could have a different spread.
- Generate the weights $\hat{\mathbf{w}}_C^{(M)}$ using supervised training.
- The supervised training can be improved with the use of the sum square error using Equation 5

$$E = \sum_{q=1}^{Q} \sum_{j=1}^{C} (\hat{y}_j^{(q)} - y_j^{(q)})^2 \tag{5}$$

Now lets look at the two class case where we can control the calculations of the weights $\mathbf{w}_C^{(M)}$, as illustrated in Figure 2.

The activation functions $f^{(M)}(\cdot)$ receives an input test vector $\mathbf{x}_0$ and the hidden layer weights $\mathbf{w}^{(M)}$, the result from the activation function are multiplied by the trained weights $\hat{\mathbf{w}}^{(M)}$ and summed with the bias term $\mathbf{b}$. This allows for a prediction of the class label to be assigned to $\hat{y}$. This is the predictions of the class label defined by the following equation:

$$\hat{y} = \sum_{i=1}^{M} \hat{w}^i \exp\left[\frac{-\|\mathbf{x}_0 - \mathbf{w}^{(i)}\|^2}{2\big(\sigma^{(i)}\big)^2}\right] + b \tag{6}$$

This provides the predicted value as being near on of the two classes $\begin{bmatrix}-1 & 1\end{bmatrix}$. The values of the weights $\hat{\mathbf{w}}^{(M)}$ now need to be calculated. This can be done by creating a matrix $\mathbf{H}$ from the input training data $\mathbf{x}$ resulting in the weights $\hat{\mathbf{w}}$ being calculated as follows:

$$\hat{\mathbf{w}} = \mathbf{H}^{-1}\mathbf{y} \tag{7}$$

where $\mathbf{y}$ are the class labels, $\begin{bmatrix}-1 & 1\end{bmatrix}$, of the two classes of interest from the training data $\mathbf{x}$ .

Now let's consider how Charu Aggarwal in [1] describes the function $f(\cdot)$ and the spread parameter. Careful consideration must be taken when selecting and using the training data $\mathbf{x}$ in the hidden layer, if the combination of the training vectors have a small and large separation between the vectors this will increase the models complexity. If a combination of small and large separation of training vectors is used it is recommended that the training data set be large to account for this complexity. For smaller numbers of training data it is also recommended to use data that has a larger separation to avoid overfitting. This leads to the selection of selecting and calculating the denominator $2(\sigma^{(i)})2$ of the function $f(\cdot)$. For the sake of training the model we will use the value of $\sigma$ as a constant and assign a calculated value for the spread in the trained model. It should be noted that in the Matlab documentation from Mathworks R2021a documentation https://www.mathworks.com/help/deeplearning/ug/radial-basis-neural-networks.html the calculation of the spread is described as, "Each bias in the first layer is set to 0.8326/SPREAD. This gives radial basis functions that cross 0.5 at weighted inputs of ± SPREAD. This determines the width of an

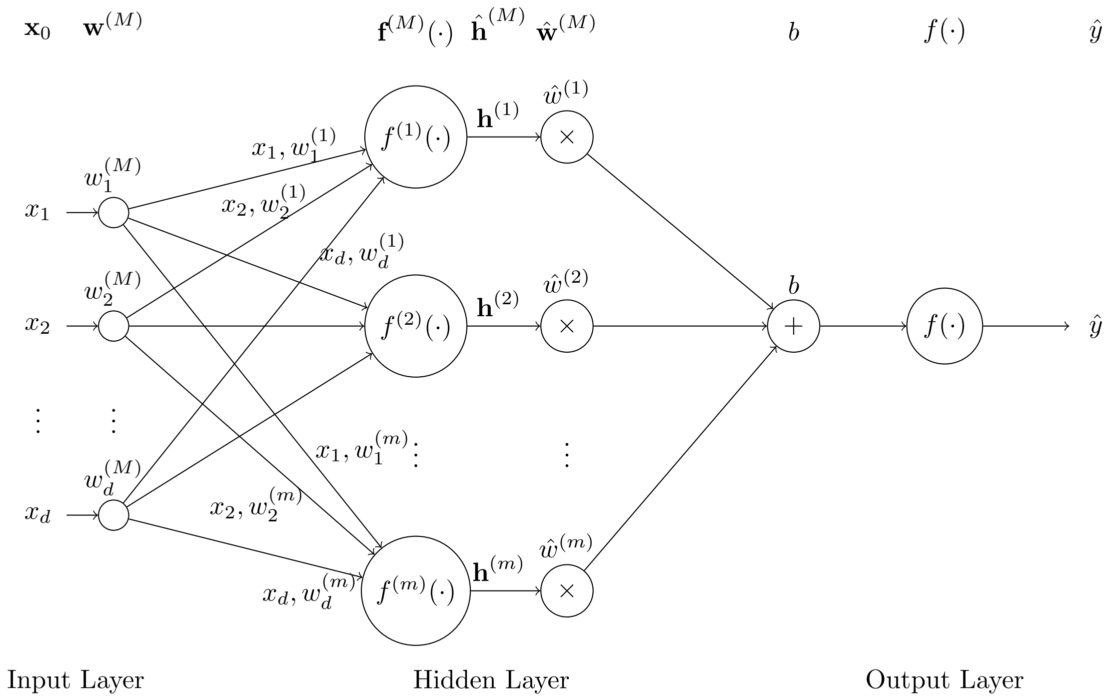


Figure 2: Simple radial basis function neural network 2 class example with $M$ nodes at the hidden layer and 1 node at the output layer.

area in the input space to which each neuron responds. If SPREAD is 4, then each radbas neuron will respond with 0.5 or more to any input vectors within a vector distance of 4 from their weight vector. SPREAD should be large enough that neurons respond strongly to overlapping regions of the input space." This will result in a class label assignment of $\begin{bmatrix}-1 & 1\end{bmatrix}$ using the following prediction function:

$$\hat{y} = \sum_{i=1}^{M} \hat{w}^i \mathbf{h}^i \ + b = \left(\hat{\mathbf{w}}\right)^T \mathbf{H} \ + b \tag{8}$$

where $\mathbf{h}^i$ is the output vector from the functions of $f^{(i)}(\cdot)$ and $i = [1, \ldots, M]$. Expanding this two class model to a set of multi-class models can be done by training the models as a one-vs-one or one-vs-all set of models. Now let's consider a set of models to represent $C$ classes, where each of the two class models are trained with the target class as $+1$ label would allow the results from Equation 8 to be returned as $-1$ or 1 allowing each two class models' outputs $\hat{y}$ to be compared with each other where an unknown observation $\mathbf{x}_0$ is the input to each model, the resulting $\hat{y}'s$ (outputs) will be processed through a decision function to determine the predicted class label. The easiest is to make a decision based on the largest value returned from the set of model outputs. In this case we let $\hat{y}_j$ represent the outputs from the $C$ models generated from the one-vs-all or the $\binom{C}{2} = \frac{C!}{2!(C-2)!}$ - combination number of models generated from the one-vs-one approach. Now the class assignment can be decided as using the following equation.

$$\hat{y} = \max\left\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_c\right\} \tag{9}$$

or

$$\hat{y} = \max\left\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{\binom{C}{2}}\right\} \tag{10}$$

The following Figure 3 Matlab code rbfClassificationExample.m show the training of a model with the set of training vectors **x** using rbfTrain.m and classifying using rbfClassify.m.

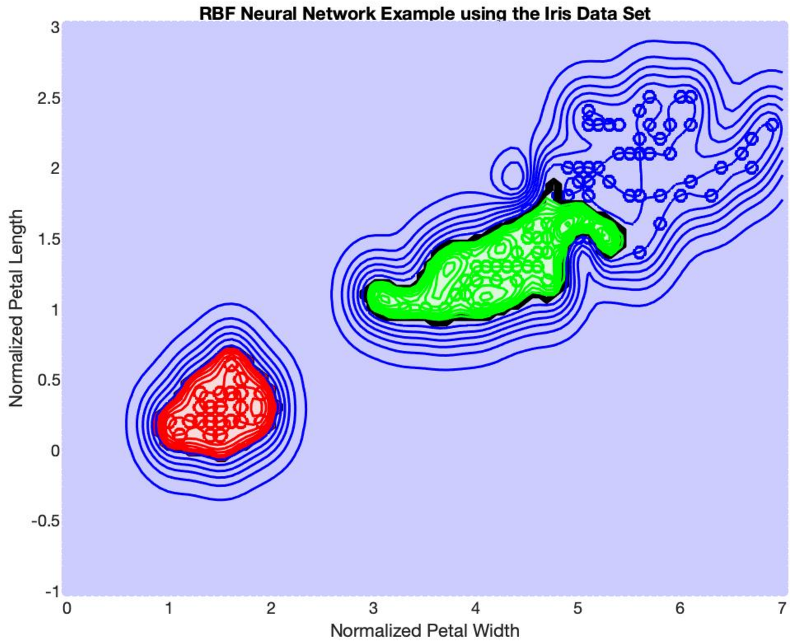


Figure 3: Radial Basis Function Neural Network Example using the Iris Data Set

```matlab
% This code is for educational and research purposes of comparisons. This
% is a RBF Neural Network with four layers on a three class
% iris data set.

clear;
clc;
close all;

irisData = readmatrix('iris.csv','Range','A2:D151');
irsData.X = irisData;

spread = 0.21;

% The following is training data to use as a simple example.
X=[0 0; 0 1.25; 1 0; 1 1.25; 1 .75; 1 2; 2 0.75; 2 2]';
y = [1 1 1 1 -1 -1 -1 -1]';

irsData.Y = [ones(1,50) ones(1,50)*(-1) ones(1,50)*(-1)]';
model_1 = rbfTrain(irsData.X, irsData.Y, spread);
irsData.Y = [ones(1,50)*(-1) ones(1,50) ones(1,50)*(-1)]';
model_2 = rbfTrain(irsData.X, irsData.Y, spread);
irsData.Y = [ones(1,50)*(-1) ones(1,50)*(-1) ones(1,50)]';
model_3 = rbfTrain(irsData.X, irsData.Y, spread);

x0 = [5.1,3.5,1.4,0.2];
```

```
[yt0_1 ypred0_1] = rbfClassify(x0, model_1);
[yt0_2 ypred0_2] = rbfClassify(x0, model_2);
[yt0_3 ypred0_3] = rbfClassify(x0, model_3);
tmp = [yt0_1;yt0_2;yt0_3];
[value y0pred] = max(tmp);

[yt1 ypred1] = rbfClassify(irsData.X, model_1);
[yt2 ypred2] = rbfClassify(irsData.X, model_2);
[yt3 ypred3] = rbfClassify(irsData.X, model_3);

tmp = [yt1;yt2;yt3];
[value ypred] = max(tmp);
irsData.Y = [ones(1,50) ones(1,50)*2 ones(1,50)*3]';
accuracy = (length(find(ypred' == irsData.Y))/150)*100;

% The following ax and ay variables test the RBF NN with the Iris data to
% determine the boundaries for the classes

irsData.Y = [ones(1,50) ones(1,50)*(-1) ones(1,50)*(-1)]';
model_1 = rbfTrain(irsData.X(:,3:4), irsData.Y, spread);
irsData.Y = [ones(1,50)*(-1) ones(1,50) ones(1,50)*(-1)]';
model_2 = rbfTrain(irsData.X(:,3:4), irsData.Y, spread);
irsData.Y = [ones(1,50)*(-1) ones(1,50)*(-1) ones(1,50)]';
model_3 = rbfTrain(irsData.X(:,3:4), irsData.Y, spread);

[Ay,Ax] = meshgrid(linspace(-1,3,101), linspace(0,7,101));
Ax = Ax(:)';
Ay = Ay(:)';
Axy = [Ax; Ay]';

[yt1 ypred1] = rbfClassify(Axy, model_1);
[yt2 ypred2] = rbfClassify(Axy, model_2);
[yt3 ypred3] = rbfClassify(Axy, model_3);

tmp = [yt1;yt2;yt3];
[value ypred] = max(tmp);

indx1 = find(ypred==1);
indx2 = find(ypred==2);
indx3 = find(ypred==3);

contour_1 = zeros(1,length(value));
contour_1(indx1) = value(indx1);
contour_2 = zeros(1,length(value));
contour_2(indx2) = value(indx2);
contour_3 = zeros(1,length(value));
contour_3(indx3) = value(indx3);

figure,plot(Axy(indx1,1),Axy(indx1,2),'.','Color',...
                    [249/255 219/255 219/255],'LineWidth',6,'MarkerSize',20)
hold on;plot(Axy(indx2,1),Axy(indx2,2),'.','Color',...
                    [219/255 249/255 219/255],'LineWidth',6,'MarkerSize',20)
hold on;plot(Axy(indx3,1),Axy(indx3,2),'.','Color',...
```

```
                          [204/255 204/255 1],'LineWidth',6,'MarkerSize',20)
hold on;plot(irsData.X(1:50,3),irsData.X(1:50,4),'ro','LineWidth',2,...
                                                         'MarkerSize',8)
hold on;plot(irsData.X(51:100,3),irsData.X(51:100,4),'go','LineWidth',2,...
                                                         'MarkerSize',7)
hold on;plot(irsData.X(101:150,3),irsData.X(101:150,4),'bo','LineWidth',...
                                                       2,'MarkerSize',7)
hold on;contour(reshape(Axy(:,1),101,101), reshape(Axy(:,2),101,101),...
                reshape(ypred,101,101),'LineColor','k','LineWidth',1.5);
% For visual representation, the following can contour plats can be
% commented out
hold on;contour(reshape(Axy(:,1),101,101), reshape(Axy(:,2),101,101),...
             reshape(contour_1,101,101),'LineColor','r','LineWidth',1.5);
hold on;contour(reshape(Axy(:,1),101,101), reshape(Axy(:,2),101,101),...
             reshape(contour_2,101,101),'LineColor','g','LineWidth',1.5);
hold on;contour(reshape(Axy(:,1),101,101), reshape(Axy(:,2),101,101),...
             reshape(contour_3,101,101),'LineColor','b','LineWidth',1.5);
title('RBF Neural Network Example using the Iris Data Set')
xlabel('Normalized Petal Width')
ylabel('Normalized Petal Length')
```

```
function model = rbfTrain(X, y, input_spread)
% This code is for educational and research purposes of comparisons. This
% is a RBF Neural Network with four layers on a three class
% iris data set.
%
% This code is modified from the old version of Matlab's RBF
%
% model = rbfMatlabTrainBRodriguez(X, y, spread)
%
% This reb training function duplicated the newrbe in Matlab.
%
% Input
%       X [n x d] training data with n observations and dimension d
%       y [n x 1] labeled targets for classification two class [-1 1]
%       spread
%
% Output
%   model - structure containing:
%     .W_hat - layer weights
%     .W - input weights
%     .bias
%     .spread
%     .input_spread
%     .error - training error [0 1]
if nargin < 3
    input_spread = 0.5;
end

[n, d] = size(X);
X = X';
H = zeros(n, n);
spread = sqrt(-log(.5))/input_spread; % This is how Matlab uses the spread
for j = 1:n
```

```
    W = X(:,j);
    D = X - W(:,ones(1,n));
        D = D.*spread;% This is how Matlab uses the spread as a bias term
    s = multiDiag(D',D);
        H(:, j) = exp(-s);
end

Htmp = [H; ones(1,size(H,1))];
Wtmp = y'/Htmp; %Wtmp*Htmp = y'
% This link provides an explanation of the right to left division
% http://www.ece.northwestern.edu/local-apps/matlabhelp/techdoc/ref/arithmeticoperators.html
% Wtmp = y'/Htmp; is equal to Wtmp = mrdivide(y',Htmp);
% Wtmp = mrdivide(y',Htmp);
% Good link for R - https://rdrr.io/rforge/pracma/src/R/mldivide.R
% Python - np.linalg.lstsq(y.T, Wtmp.T)[0].T
W_hat = Wtmp(1:size(H,1));
bias = Wtmp(end);
yt = (H * W_hat')' + bias;
ypred = ones(size(y));
ypred(find(yt<0)) = -1;
predError = 1 - length(y == ypred)/size(y,1);

model.W_hat = W_hat;
model.W = X;
model.bias = bias;
model.spread = spread;
model.input_spread = input_spread;
model.error = predError;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function returns the diagonal product of X1 and X2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function xDiag = multiDiag(X1, X2)
% Inputs
%   X1 - [d x n]
%   X2 - [n x d]
%
% Output
%   xDiag - [d x 1]

[r1,c1] = size(X1);
[r2,c2] = size(X2);

X1tmp = X1';
X1tmp = X1tmp(:);
X2tmp = X2(:);
X = zeros(c1,r1);
X(:) = X1tmp .* X2tmp;
[r1,c1] = size(X);
if r1 > 1
        xDiag = sum(X)';
else
        xDiag = X';
```

```
end
```

```
function [y ypred] = rbfClassify(X, model);
% This code is for educational and research purposes of comparisons. This
% is a RBF Neural Network with four layers on a three class
% iris data set.
%
% This code is modified from the old version of Matlab's RBF
%
% [y ypred] = rbfMatlabClassifyBRodriguez(X, y, spread)
%
% This function returns the y labels as an approximation under the gaussian
% curve while the returned value ypred returns class labels as [-1 1]
%
% Input
%   X [n x d] data to be classified with n observations and dimension d
%   model - structure containing:
%     .W_hat - layer weights
%     .W - input weights
%     .bias
%     .spread
%     .input_spread
%     .error - training error [0 1]
%
% Output
%   y [n x 1]labels as an approximation under the gaussian curve
%   ypred [n x 1] class labels [-1 1]

[n1, d1] = size(X); X = X';
[n2, d2] = size(model.W');

H = zeros(n1, n2);
for j = 1:n2
    W = model.W(:,j);
    D = X - W(:,ones(1,n1));
    D = D.*model.spread;
    s = multiDiag(D',D);
        H(:, j) = exp(-s);
end

y = (H * model.W_hat')' + model.bias;
ypred = ones(size(y));
ypred(find(y<0)) = -1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function returns the diagonal product of X1 and X2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function xDiag = multiDiag(X1, X2)
% Inputs
%   X1 - [d x n]
%   X2 - [n x d]
%
% Output
%   xDiag - [d x 1]
```

```
[r1,c1] = size(X1);
[r2,c2] = size(X2);

X1tmp = X1';
X1tmp = X1tmp(:);
X2tmp = X2(:);
X = zeros(c1,r1);
X(:) = X1tmp .* X2tmp;
[r1,c1] = size(X);
if r1 > 1
        xDiag = sum(X)';
else
        xDiag = X';
end
```

# 3 Probabilistic Neural Networks (PNN)

The classification frame work of the probabilistic neural network is shown in Figure 5 we introduced by D.F. Specht in the late 80 and published in the early 90s (Specht, 1998; 1990)[6]. These networks are similar to the feed forward neural network that have a background from Bayesian networks. The probabilistic neural network is a four layer architecture that contains input layer, pattern layer, summation layer and output layer as shown in Figure 5. It should be noted that the pattern and summation layers are part of the hidden layer.

The structure of the probabilistic neural network classifier as has four layers as shown in Figure 5, where the input layer is used to extract features from the supplied data set, pattern layer there are $n$ neurons with groups corresponding to the $C$ number of classes, summation layer has $C$ neurons where each neuron sums the values from the corresponding outputs from the pattern layer and the decision/output layer determines which class the input observation $\mathbf{x}_0$ belongs to. There are a few decisions that have to be made regarding training of this neural network. First, the number of training samples and number of classes are selected for the pattern layer; this defines the structure of the network. For example, the set of input training samples is represented as $\mathbf{x} = \mathbf{x}_N = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T \in \Re^D$ and a class label $y_i \in C = [C_1, C_2, \ldots, C_c], i = 1, 2, \ldots, c$. This will result in $c$ groups with each group in the pattern layer containing $N_C = [N_1, N_2, \ldots, N_c] \in N$ neurons per class. The difference squared sum of the input patterns vector $\mathbf{x}_0$ with the weight vectors $\mathbf{w}_N$ is performed as $(\mathbf{x}_0 - \mathbf{w}_n)^T(\mathbf{x}_0 - \mathbf{w}_n)$. Letting $z_n = (\mathbf{x}_0 - \mathbf{w}_n)^T(\mathbf{x}_0 - \mathbf{w}_n)$ allows the a non linear operator to be performed in the summation layer. Second, for the summation layer uses a non linear Equation 11 to be used with $\mathbf{z}_n$ and the smoothing parameter, $\sigma$. The nonlinear operation $f(z_n)$ assumes that the vectors $(\mathbf{x}_0 - \mathbf{w}_n)$ are normalized to unit length, range of $\begin{bmatrix}0 & 1\end{bmatrix}$. As a general guideline the value of the smoothing parameter, $\sigma$, should be chosen as a function of the dimension of the problem, $D$, and the number of training samples, $N$, (Specht, 1990).

$$f(\mathbf{z}_n) = \exp\left(\frac{\mathbf{z}_n - 1}{\sigma^2}\right) \tag{11}$$

The results from Equation 12 are sent to the summation layer which correspond to the classes from which the training weights, $\mathbf{w}_i$, were passed through. The output layer can be developed using a variety of decision functions. The easiest is to make a decision based on the largest value returned from the summation layer.

$$\sum_C = \sum_{n=1}^{N} \exp\left(\frac{\mathbf{z}_n - 1}{\sigma^2}\right) \tag{12}$$

where the $\sum_1$ for class 1 is as follows:

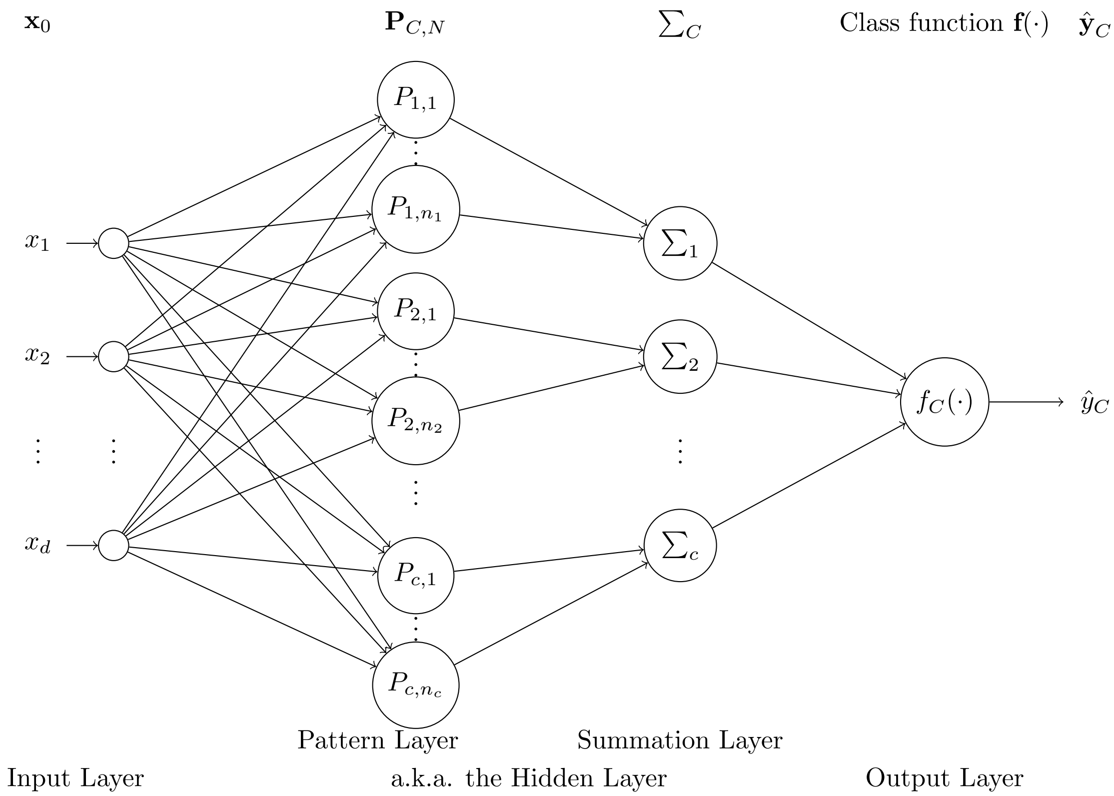


Figure 4: Simple probabilistic neural network architecture example with $C \times D$ nodes at the pattern layer and $C$ node at the summation layer.

$$\sum_{1} = \sum_{n_1=1}^{N_1} \exp\left(\frac{\mathbf{z}_{n_1} - 1}{\sigma^2}\right) \tag{13}$$

followed by the class 2 as follows:

$$\sum_{2} = \sum_{n_2=1}^{N_2} \exp\left(\frac{\mathbf{z}_{n_2} - 1}{\sigma^2}\right) \tag{14}$$

and so on until the class $c$ is reached as:

$$\sum_{c} = \sum_{n_c=1}^{N_c} \exp\left(\frac{\mathbf{z}_{n_c} - 1}{\sigma^2}\right) \tag{15}$$

Now the class assignment can be decided as using the following equation.

$$\hat{y}_C = f_C\left(\sum_{C}\right) = \max\left\{\sum_{C}\right\} = \max\left\{\sum_{1}, \sum_{2}, \ldots, \sum_{c}\right\} \tag{16}$$

The following Matlab code shows how the Iris data set is used to produce the Figure 5.

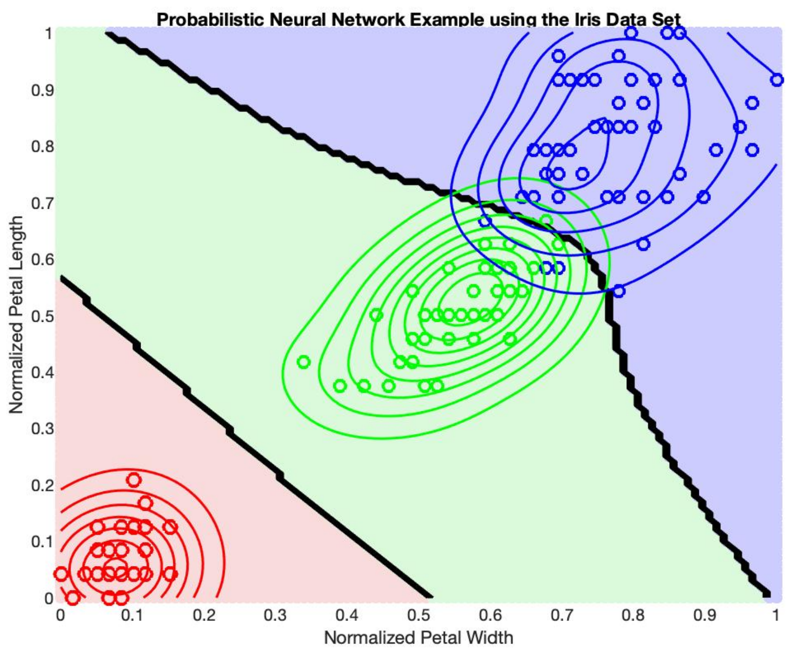


Figure 5: Probabilistic Neural Network Example using the Iris Data Set

**Matlab Code**

```matlab
% This code is for educational and research purposes of comparisons. This
% is a Probabilistic Neural Network with four layers on a three class
% iris data set.
%
% The original code was found from Abhisar Mohapatra around 2014
% Reference:
%           https://github.com/Abhisar/Probablistic-Neural--Network

clear;
clc;
close all;

irisData = readmatrix('iris.csv','Range','A2:D151');
irsData.X = irisData;
irsData.Y = [ones(1,50) ones(1,50)*2 ones(1,50)*3]';

iris_min = min(irsData.X);
iris_max = max(irsData.X);

% This normalizes the data to unit length 0 to 1. Note that if the sample
% x_0 is a single test data point the min and max values from the training
% data.
for i=1:150
    for j=1:4
        x(i,j) = (irsData.X(i,j)-iris_min(j))/(iris_max(j)-iris_min(j));
    end
end
w = x;
w1 = x(1:50,:);
w2 = x(51:100,:);
w3 = x(101:150,:);

temp = zeros(1,3);
sigma = .1;
ypred = [];

for i=1:150 % All data to determine which class the observations belong to
    sum1 = 0;
    for j=1:50 % Setosa Class
        z1 = (x(i,:)-w1(j,:))*(x(i,:)-w1(j,:))';
        sum1 = sum1 + exp(-(z1-1)/(sigma^2));
    end
    temp(1,1) = sum1/150;
    sum2 = 0;
    for j=1:50 % Versicoloe class
        z2 = (x(i,:)-w2(j,:))*(x(i,:)-w2(j,:))';
        sum2 = sum2 + exp(-(z2-1)/(sigma^2));
    end
    temp(1,2) = sum2/150;
    sum3 = 0;
    for j=1:50 % Virginica class
        z3 = (x(i,:)-w3(j,:))*(x(i,:)-w3(j,:))';
        sum3 = sum3 + exp(-(z3-1)/(sigma^2));
```

```
    end
    temp(1,3) = sum3/150;
    [value ypred(i,1)] = max(temp); % the maximum value is selected to
                                    % assign the class for the observation.
end

accuracy = (length(find(ypred == irsData.Y))/150)*100;

% The following ax and ay variables test the kernel with the Iris data to
% determine the boundaries for the classes
ax=-1:0.04:3;
ay=0:0.07:7;
[Ax,Ay] = meshgrid(linspace(-1,3,101), linspace(0,7,101));
Ax = Ax(:)';
Ay = Ay(:)';
A_xy = [Ax; Ay]';

Axy_min = min(A_xy);
Axy_max = max(A_xy);

for i=1:length(Ax)
    for j=1:2
        Axy(i,j) = (A_xy(i,j)-Axy_min(j))/(Axy_max(j)-Axy_min(j));
    end
end

ypred = [];

for i=1:length(Ax) % All Ax abd Ay data to determine which class the
                   % observations belongs to
    sum1 = 0;
    for j=1:50 % Setosa Class
        z = (Axy(i,:)-w(j,3:4))*(Axy(i,:)-w(j,3:4))';
        sum1 = sum1 + exp(-(z-1)/(sigma^2));
    end
    temp(i,1) = sum1/150;
    sum2 = 0;
    for j=51:100 % Versicoloe class
        z = (Axy(i,:)-w(j,3:4))*(Axy(i,:)-w(j,3:4))';
        sum2 = sum2 + exp(-(z-1)/(sigma^2));
    end
    temp(i,2) = sum2/150;
    sum3 = 0;
    for j=101:150 % Virginica class
        z = (Axy(i,:)-w(j,3:4))*(Axy(i,:)-w(j,3:4))';
        sum3 = sum3 + exp(-(z-1)/(sigma^2));
    end
    temp(i,3) = sum3/150;
    [value ypred(i,1)] = max(temp(i,:)); % the maximum value is selected to
                                    % assign the class for the observation.
end

indx1 = find(ypred==1);
```

```
indx2 = find(ypred==2);
indx3 = find(ypred==3);
figure,plot(Axy(indx1,1),Axy(indx1,2),'.','Color',...
                        [249/255 219/255 219/255],'LineWidth',6,'MarkerSize',20)
hold on;plot(Axy(indx2,1),Axy(indx2,2),'.','Color',...
                        [219/255 249/255 219/255],'LineWidth',6,'MarkerSize',20)
hold on;plot(Axy(indx3,1),Axy(indx3,2),'.','Color',...
                                [204/255 204/255 1],'LineWidth',6,'MarkerSize',20)
hold on;plot(w(1:50,3),w(1:50,4),'ro','LineWidth',2,'MarkerSize',8)
hold on;plot(w(51:100,3),w(51:100,4),'go','LineWidth',2,'MarkerSize',7)
hold on;plot(w(101:150,3),w(101:150,4),'bo','LineWidth',2,'MarkerSize',7)
hold on;contour(reshape(Axy(:,1),101,101), reshape(Axy(:,2),101,101),...
                        reshape(ypred,101,101),'LineColor','k','LineWidth',1.5);
hold on;contour(reshape(Axy(:,1),101,101), reshape(Axy(:,2),101,101),...
                   reshape(temp(:,1),101,101),'LineColor','r','LineWidth',1.5);
hold on;contour(reshape(Axy(:,1),101,101), reshape(Axy(:,2),101,101),...
                   reshape(temp(:,2),101,101),'LineColor','g','LineWidth',1.5);
hold on;contour(reshape(Axy(:,1),101,101), reshape(Axy(:,2),101,101),...
                   reshape(temp(:,3),101,101),'LineColor','b','LineWidth',1.5);
title('Probabilistic Neural Network Example using the Iris Data Set')
xlabel('Normalized Petal Width')
ylabel('Normalized Petal Length')
```

## 4 Summary

In this document the Radial Basis Neural Network and the Probabilistic Neural Network were introduced. The Feedforward Neural Network and the Convolutional Neural Network will be introduced in the Deep Learning module.

## References

[1] Charu C. Aggarwal, *Neural Networks and Deep Learning*, Springer, 2018

[2] Christopher M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995

[3] Martin T. Hagan, Howard B. Demuth, Mark Hudson Beale, and Orlando De Jesus, *Neural Network Design, 2nd Edition*, 2014

[4] Carl G. Looney, *Pattern Recognition using Neural Networks, Theory and Algorithms for Engineers and Scientists*, Oxford University Press, 1997

[5] Donald F. Specht, *Probabilistic Neural Networks for Classification, Mapping, or Associative Memory*, IEEE International Conference in Neural Networks, 1, pp. 525-532, 1988

[6] Donald F. Specht, *Probabilistic Neural Networks*, Neural Networks, Vol. 3, pp. 109-118, 1990

[7] Sergios Theodoridid and Konstantinos Koutroumbas, *Pattern Recognition, Third Edition*, Academic Press, 2006