Chapter 11

Radial Basis Functions

11.1 Polynomial Regression

In the interpolation problem of Chapter 1, we saw that a polynomial of degree $n-1$ will connect n data points. The regression problem is different in that we are given n data points, and we are asked to fit a polynomial of degree k to it. There now need be no connection between the degree and the number of data points, and typically, we have many more points than degrees.

1. **Remark:** We begin by assuming that the function

   $$f : X \subset \mathbb{R} \to Y \subset \mathbb{R}$$

   is of degree k, and let the number of data points be p. Form the partial Vandermonde matrix (size will be $p \times (k+1)$), so that we are solving the matrix equation:

   $$VC = Y$$

   where the i^{th} row of V is:

   $$(x_i^k \; x_i^{k-1} \; \ldots x_1 \; 1) \tag{11.1}$$

2. **Exercise:** Find the least squares solution to the 1-d polynomial regression problem by using the SVD and the generalized inverse.

3. **Exercise:** Rewrite Matlab's `vander` command so that `Y=vander1(X,k)` where X is $m \times 1$ (m is the number of points), $k-1$ is the degree of the polynomial and Y is the $m \times k$ matrix whose rows look like those in Equation 11.1.

4. **Remark:** Polynomials seldom do a good job at regression, and the following exercises show this.

5. **Exercise:** Load the data in the file `poly_regres.mat`. Here you will find data sets P and T, where P are points uniformly distributed between -1 and 1, and T are the desired targets (thus, the desired graph are points (p_i, t_i)). Note that P and T are given as rows, but will need to be columns in our suite of commands. Look at the following sequence of commands, and reproduce them. Be sure you understand what each one does.

```
Y=-1:0.01:1;
TT=T+0.1*randn(size(T));
A=vander1(P',18);
G=A\TT';
Z=polyval(G,Y');
plot(P,T,P,TT,'*',Y,Z);
```

   In this example, we used a polynomial of degree 17. Now try the same thing with:

   (a) Use the original targets, with degrees 10, 50, 100.

   (b) Use the noisy targets with degrees 10, 50, 100.

   Which polynomials work "best"?

6. **Remark:** A second order polynomial of 1 variable has the following basis terms: $\{1, x, x^2\}$. A second order polynomial of 2 variables has: $\{1, x, y, xy, x^2, y^2\}$. Going to dimension 3: $\{1, x, y, z, xy, xz, yz, x^2, y^2, z^2\}$. Recall that for each basis term, there is an associated unknown constant. This unknown is called a parameter, and the number of parameters will be the number of columns in the regression matrix V.

7. **Exercise:** Show that for a second order polynomial in dimension n, there are:
$$\frac{n^2 + 3n + 2}{2} = \frac{(n+2)!}{2!n!}$$
unknown constants, or parameters.

8. **Remark:** In general, if we use a polynomial of degree d in dimension n, there are
$$\frac{(n+d)!}{d!\, n!} \approx n^d$$
parameters. How many parameters do we have if we wish to use a polynomial of degree 10 in dimension 5?

9. **Exercise:** Write a Matlab subroutine for which we input data in $\mathbb{R}^4$ and outputs the regression matrix for a second order approximation. Solve the regression problem for the data in the file "`polydat.mat`".

10. **Remark:** This explosion in the number of parameters needed for regression is known as **the curse of dimensionality**.

11. **Exercise:** Suppose that we have a unit box in n dimensions (corners lie at the points $(\pm 1, \pm 1, \ldots, \pm 1)$), and we wish to subdivide each edge by cutting it into k equal pieces. Verify that there are k^n sub-boxes. Consider what this number will be if k is any positive integer, and n is conservatively large, say $n = 1000$! This rules out any simple, general, "look-up table" type of solution to the regression problem.

12. **Definition:** Let $\{f_i\}_{i \in \mathcal{I}}$ be a set of real-valued functions whose domain is $[0,1]^n$. This set forms a *basis* for the set of continuous functions on $[0,1]^n$ if the functions are linearly independent and the set is *dense*. If that is the case, we say that it is possible to formulate any continuous function g in terms of the basis:
$$g(x) = \sum_{i \in \mathcal{I}} \alpha_i f_i(x)$$

13. **Example:** The Stone-Weierstrasse Theorem says that the set $\{x^n\}_{i=1}^{\infty}$ is dense in $C[0,1]$. This is the polynomial approximation that we've been doing. Notice that the set of basis functions did not depend on the data (although, of course, the coefficients did).

14. **Example:** The Fourier Basis is dense for 2π periodic functions, or on $C[0, 2\pi]$. The Fourier Basis is given by: $\{\sin(nx), cos(mx)\}_{m,n=1}^{\infty}$

15. **Definition:** A set of basis functions for $C[a,b]^n$ is said to be *fixed* if the basis does not depend on the data.

16. **Remark:** Recent results [2, 3] have shown that *any* fixed basis will suffer from the curse of dimensionality. This leads us to consider basis functions that are *semi-analytic* in that the basis functions we choose will be data-dependent. Ideally, the number of parameters needed for regression should depend not on the dimension of the data, but on the complexity of the function.

11.2 Distance Matrices

We will begin considering a type of regression model known as "Radial Basis Functions" shortly. Before we do that, let us consider a special type of matrix that we will be using extensively.

1. **Definition: The Euclidean Distance Matrix (EDM)** Let $\{\boldsymbol{x}^{(i)}\}_{i=1}^{P}$ be a set of points in $\mathbb{R}^n$. Let the matrix $\boldsymbol{A}$ be the P by P matrix so that the $(i,j)^{\text{th}}$ entry of $\boldsymbol{A}$ is given by:
$$A_{ij} = \|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2$$
Then $\boldsymbol{A}$ is the Euclidean Distance Matrix (EDM) for the data set $\{\boldsymbol{x}^{(i)}\}_{i=1}^{P}$.

2. **Exercise:** By hand, construct the EDM for the data (given as $n \times p$, where n is the dimension, and p is the number of points):

(a) $X = (1, 2, 3, 4)$

(b) $X = \begin{pmatrix} -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

(c) $X = \begin{pmatrix} -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & 1 \end{pmatrix}$

3. **Exercise:** Print and analyze the Matlab function EDM. Be sure you understand what it (and the functions it calls) are doing.

4. **Remark:** The EDM is a symmetric matrix with 0's along its diagonal.

5. **Exercise:** What can we say about the eigenvalues and eigenvectors of the EDM?

6. **Matlab Comment:** One can find the minimum element in a vector: `min(x)` or in a matrix: `min(min(X))`

7. **Theorem (Schoenberg, 1937)**. Let $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p\}$ be p distinct points in $\mathbb{R}^n$. Then the EDM is invertible.

8. **Remark:** This theorem implies that, if $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p\}$ are p distinct points in $\mathbb{R}^n$, and $\{y_1, \ldots, y_p\}$ are points in $\mathbb{R}$, then there exists a function $f : \mathbb{R}^n \to \mathbb{R}$ such that:

$$f(\boldsymbol{x}) = \alpha_1 \|\boldsymbol{x} - \boldsymbol{x}_1\| + \ldots + \alpha_p \|\boldsymbol{x} - \boldsymbol{x}_p\|$$

and $f(\boldsymbol{x}_i) = y_i$. Therefore, this function f solves the interpolation problem.

9. **Exercise:** Verify that the α_i defined above are solved:

$$A\boldsymbol{\alpha} = Y$$

where $A_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ and Y is a column vector.

10. **Exercise:** This approximation is a generalization of a piecewise linear interpolation. To see this, try plotting

$$f(x) = -2|x + 1| + |x - 2| + \frac{1}{2}|x - 3|$$

(Hint: In Matlab, $|x|$ =`abs(x)`).

11. **Remark:** As we saw in Chapter 2, the invertibility of a matrix depends on its smallest eigenvalue. A recent theorem states "how invertible" the EDM is:

12. **Theorem [?]:** Let $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p\}$ be p distinct points in $\mathbb{R}^n$. Let ϵ be the smallest element in the EDM, so that

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \geq \epsilon, \; i \neq j$$

Then we have that all eigenvalues λ_i are all bounded away from the origin- there is a constant c so that:

$$\lambda_i \geq \frac{c\epsilon}{\sqrt{n}} \tag{11.2}$$

13. **Exercise:** Let's examine what this is saying by looking at random data in $\mathbb{R}^4$. Write the following script and plot the resulting x and y. For comparisons, plot x versus $x/2$. This will take a few minutes, so we print the "i" to see where the loop is. Your output should be similar to Figure 11.1.

```
for i=1:100
i
X=rand(50,4); %Randomly choose 50 points
A=edm(X);
V=eig(A);
x(i)=min(min(A+eye(50)));  %This is epsilon.
y(i)=min(abs(V));          %Smallest eigenvalue.
end
plot(x,y);
```

11.3 Radial Basis Functions

1. **Definition: The Transfer Function and Matrix** Let $\phi : \mathbb{R}^+ \to \mathbb{R}$ be chosen from the list below:

$$\begin{array}{lcll}
\phi(r,\sigma) & = & \exp\left(\frac{-r^2}{\sigma^2}\right) & \text{Gaussian} \\
\phi(r) & = & r^3 & \text{Cubic} \\
\phi(r) & = & r^2 \log(r) & \text{Thin Plate Spline} \\
\phi(r) & = & \frac{1}{r+1} & \text{Cauchy} \\
\phi(r,\beta) & = & \sqrt{r^2+\beta} & \text{Multiquadric} \\
\phi(r,\beta) & = & \frac{1}{\sqrt{r^2+\beta}} & \text{Inverse Multiquadric}
\end{array}$$

Then ϕ is a *transfer function.* If we apply ϕ to a matrix elementwise so that:

$$\boldsymbol{\Phi} = (\phi(\boldsymbol{A}))_{ij} = (\phi(A_{ij}))$$

Then $\boldsymbol{\Phi}$ is the transfer matrix. Note that when A is a matrix, Matlab will automatically perform some functions elementwise- for example, `sin(A)`.

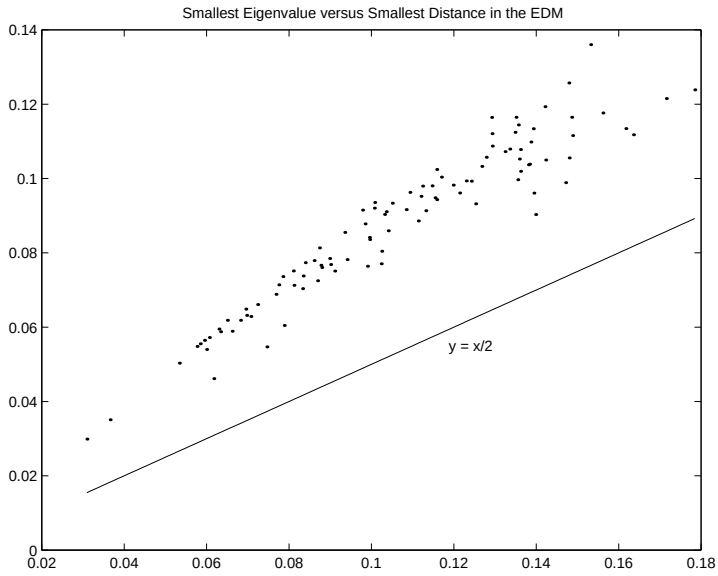

Figure 11.1: This plot illustrates the statement in Equation (11.2), which tells us how invertible the Euclidean Distance Matrix is.

2. **Example:** If $\phi(r) = \sqrt{r^2 + 1}$, then ϕ applied to the matrix A will be done via:

   ```
   (A.^2+1).^0.5
   ```

3. **Remark:** There are other transfer functions one can choose. For a broader definition of transfer functions, see Micchelli [26]. We will examine the effects of the transfer function on the radial approximation shortly.

4. **Remark:** Matlab only uses a Gaussian transfer function. The next few exercises show some of the implications.

5. **Exercise:** Run Matlab's demos: `demorb1` and `demorb3` to see how σ effects the functional approximation. (You can just type "demo", then go to the Neural Network Toolbox demos, then choose one of the RBF demos- "underlapping" neurons, and "overlapping" neurons).

6. **Exercise:** We need to consider how different values of σ effects the width of the Gaussian. Use Maple to animate the Gaussian function:

   ```
   with(plots):
   animate(exp(-x^2/t^2),x=-5..5,t=0.5..2);
   ```

7. **Definition:** We'll make a working definition of the *width* of the Gaussian: It is the value a so that k percentage of the area is between $-a$ and a (so k is between 0 and 1). The actual value of k will be problem-dependent.

8. **Exercise:** We will approximate:

$$\left(\int_{-b}^{b} \mathrm{e}^{-x^2}\, dx\right)^2 = \int_{-b}^{b}\int_{-b}^{b} \mathrm{e}^{-(x^2+y^2)}\, dx\, dy \approx \int\int_{B} \mathrm{e}^{-(x^2+y^2)}\, dB$$

where B is the disk of radius b. Show that this last integral is:

$$\pi\left(1-\mathrm{e}^{-b^2}\right)$$

9. **Exercise:** Using the previous exercise, conclude also that:

$$\int_{-\infty}^{\infty} \mathrm{e}^{\frac{-x^2}{\sigma^2}}\, dx = \sigma\sqrt{\pi}$$

10. **Exercise:** Use the previous two exercises to show that our working definition of the "width" a, means that, given a we would like to find σ so that:

$$\int_{-a}^{a} \mathrm{e}^{\frac{-x^2}{\sigma^2}}\, dx \approx k \int_{-\infty}^{\infty} \mathrm{e}^{\frac{-x^2}{\sigma^2}}\, dx$$

11. **Exercise:** Show that the last exercise implies that, if we are given k and a, then we should take σ to be:

$$\sigma = \frac{a}{\sqrt{-\ln(1-k^2)}} \tag{11.3}$$

12. **Remark:** Matlab uses the following approximation for σ (See, for example, `designrb`, which is in the file `newrb.m`):

$$\sigma = \frac{a}{\sqrt{-\ln(0.5)}}$$

11.4 Interpolation via Radial Basis Functions

1. **Notation:** Let $\{\boldsymbol{x}^{(i)}\}_{i=1}^{P} \subset \mathbb{R}^N$. Suppose we also have a data set $\{\boldsymbol{y}^{(i)}\}_{i=1}^{P} \subset \mathbb{R}^M$. Let $\boldsymbol{Y}$ be the P by M matrix constructed so that its i^{th} row is $(\boldsymbol{y}^{(i)})^{\mathrm{T}}$.

2. **Definition:** The interpolation problem is to determine a function $\boldsymbol{F}$ such that

$$\boldsymbol{y}^i = \boldsymbol{F}(\boldsymbol{x}^{(i)}),\ i = 1 \ldots P \tag{11.4}$$

3. **Remark:** So far, $\boldsymbol{F}$ is not well defined. Think about how many functions (continuous, not continuous, differentiable, not differentiable) will go through those points.

4. **Remark:** We will assume a certain form for $\boldsymbol{F}$ so that Equation (11.4) has the least squares solution.

5. **Remark:** Assume $\boldsymbol{F}$ is defined so that the j^{th} element of $\boldsymbol{F}(\boldsymbol{x}^{(i)})$ is given by:
$$\boldsymbol{F}_j(\boldsymbol{x}^{(i)}) = \sum_{k=1}^{P} \omega_{kj}\ \phi\left(\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(k)}\|\right) + b_j$$
where the ω_{ij} are parameters to be determined, where i runs from 1 to P, and j runs from 1 to m. Then the interpolation problem translates to the following matrix equation, for which we need to solve for the constants $\omega_{ij} \in \Omega$, which is a P by M matrix.
$$\boldsymbol{\Phi}\Omega + \boldsymbol{b} = \boldsymbol{Y} \tag{11.5}$$
By the properties listed above, if $\boldsymbol{b} = \boldsymbol{0}$, then this equation has a unique solution
$$\Omega = \boldsymbol{\Phi}^{-1}\boldsymbol{Y}$$
since $\boldsymbol{\Phi}$ is nonsingular, and ϕ has been chosen from the classes allowed.

6. **Exercise:** Show that Equation (11.5) can be written as a single matrix equation:
$$\hat{\boldsymbol{\Phi}}\hat{\Omega} = \boldsymbol{Y}$$
so that we can also solve for the vector $\boldsymbol{b}$. Hint: Think about the sizes of the matrices.

7. **Definition:** A *training set* is a set of data that is used for computing the weights and biases of a model. The *validation set* is a different set of data that is used to test the performance of the model found using the training set. We'll see why these are important in the next section.

11.5 Generalization

We saw earlier with polynomials that many times we do not desire an interpolative solution. We will be willing to trade some accuracy for a smoother, less complex function approximation.

1. **Ill-Defined Problems:** Not all problems have solutions. We may know this ahead of time, but wish to find a solution that is "close". For example, we know that there is a fundamental difference between $\mathbb{R}^n$ and $\mathbb{R}^m$, and so there may be problems in finding functions that map one to another.

2. **Example:** Consider the following example. The object in Figure 11.2 is not one dimensional, and so there does not exist a continuous function from $\mathbb{R}^1$ to the object presented. However, as soon as we discretize the object to data points, then we can interpolate the data. One solution is

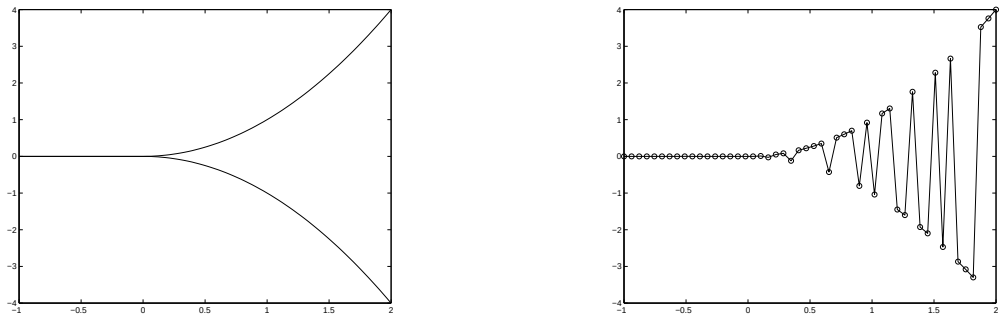

Figure 11.2: The image to the left is the original range set. The image to the right depicts the RBF solution to the discretized range. This shows that, while the RBF can obtain an exact solution to the interpolation problem, that solution is often not the optimal solution to generalize the range.

presented in Figure 11.2. Note that the best (in terms of least squares error) one dimensional representation of this object should be the zero function.

3. **Remark:** From [4], we obtain an extension of the RBF as stated earlier. Rather than computing the full EDM, we replace the model of the function $\boldsymbol{F}$:

$$\boldsymbol{F}_j(\boldsymbol{x}^{(i)}) = \sum_{k=1}^{K} \omega_{kj}\ \phi\left(\|\boldsymbol{x}^{(i)} - \boldsymbol{c}_k\|\right) + b_j$$

where we will refer to the set $\{\boldsymbol{c}_k\}_{k=1}^{K}$ as the **set of centers**.

4. **Remark:** Given a set of $K << P$ centers, we redefine the transfer matrix so that

$$(\boldsymbol{\Phi}_c)_{ij} = \phi\left(\|\boldsymbol{x}^{(i)} - \boldsymbol{c}_j\|\right)$$

and $\boldsymbol{\Phi}_c$ is now a P by K matrix.

5. **Remark:** Once a set of centers has been chosen, then we have changed the interpolation problem to a problem of finding the least squares solution to:

$$\hat{\boldsymbol{\Phi}}_c \hat{\Omega} = \boldsymbol{Y}$$

whose solution is:

$$\hat{\Omega} = \hat{\boldsymbol{\Phi}}_c^{+} \boldsymbol{Y} \tag{11.6}$$

where $\hat{\boldsymbol{\Phi}}_c^{+}$ is the Moore-Penrose generalized inverse of $\hat{\boldsymbol{\Phi}}_c$.

6. **Remark:** We will be using only this new form of the transfer matrix, and so will suppress the "c" subscripts for the remainder of this chapter.

7. **Summary thus far:** We have introduced the RBF equations, and the role of the EDM. We have now reduced the size of the EDM via the use of centers. Once the set of centers has been chosen, we can solve the RBF equation using the pseudo-inverse. That is, solve Equation (11.6). In Matlab, we can use either the SVD, the backslash command, or the "pinv" command.

   Therefore, in using an RBF we must answer 2 questions:

   (a) What should the transfer function be? If it has an extra parameter (like the Gaussian), what should it be? A rule of thumb for the Gaussian: The width should be wider than the distance between data points, but smaller than the diameter of the set.

   (b) How many centers should I use, and where should I put them?

   Once these questions are answered, we can solve the problem

8. **Remark:** For the first question, there is no general answer, and we've examined some issues related to which transfer function to use. In theory, it doesn't matter, as long as the data is scaled appropriately. In practice, the choice of transfer function may force you to rescale your data, and may impact on the number of centers needed.

9. **Remark:** On the number of centers, we need to remember the tradeoff between generalization and memorization. In general, we want the fewest number of centers necessary to attain our error goal on the *Training Set* while still getting a good error on the *Validation Set*. In most problems, these goals will be contradictory.

10. **Remark:** If we use only 1 center, we'll get some really bad errors. If we use the entire data set for the centers, the error on the training set will be zero. Somewhere in between we must be getting a good tradeoff, as Figure 11.3 depicts.

11. **Remark:** On the placement of centers, we have again some options. They are listed below, then we will consider each seperately:

    (a) Choose the centers at random.

    (b) Choose the centers using Linear Algebra (Orthogonal Least Squares).

    (c) Choose the centers using Nonlinear Optimization.

12. **Exercise:** We examine the question: If we fix the number of centers to be k, will one random choice of centers be as good as any other?

    Construct a training set of 100 points using $y = \sin(2\pi x) + \epsilon$, where ϵ is normally distributed noise with std 0.1. Fix the number of centers at 30.

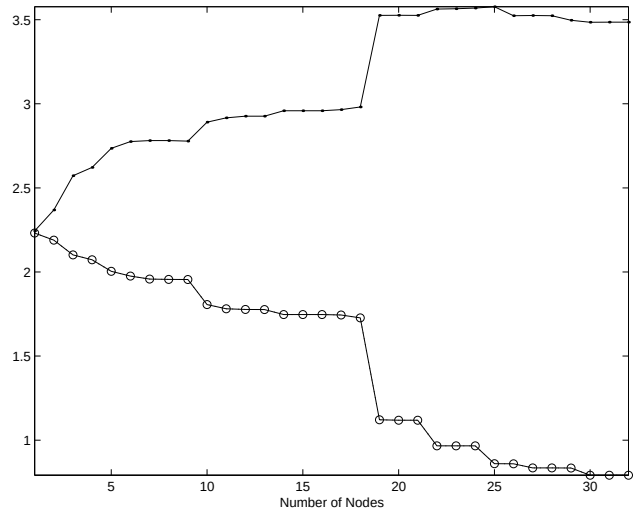

Figure 11.3: This is a plot of the error over the training set (plotted using circles) and the error over the validation set (plotted using dots). Using this picture, the best error possible is the point at which these curves intersect, which is using 1 center.

We're going to want to choose centers at random from the domain set. How should we code it? Remember- we do not want to choose 2 indices to be the same! See the sample code `randr.m`

Now, construct the RBF approximations using the cubic transfer function. We want to consider the error of the approximation. Use the function `rbfDesig.m`.

Construct a new data set using 1000 points. Compare the "true output" versus the RBF output on the data. Compute the mean squared error.

Repeat 25 times for different choices of the 30 centers and plot the 25 errors. What is your conclusion?

13. **Remark:** In using nonlinear optimization, the weights ω_{ij} and biases b_j are updated slowly so that the overall mean squared error gets smaller and smaller. If E is the mean squared error, then the update rule is:

$$\omega_{ij} = \omega_{ij} + h\frac{\partial E}{\partial \omega_{ij}}$$

and

$$b_j = b_j + h\frac{\partial E}{\partial b_j}$$

and we can even update the center locations:

$$\boldsymbol{c}_k = \boldsymbol{c}_k + h\frac{\partial E}{\partial \boldsymbol{c}_k}$$

14. **Remark:** Although these methods are much more complex in implementation than OLS (below), they are used extensively when the training data are not known all at once.

11.6 Orthogonal Least Squares

The following is a summary of the work in the reference [5]. We present the multidimensional extension that is the basis of the method used in Matlab's subfunction: `designrb`, which can be found in the file `newrb`. (Hint: To find the file, in Matlab type `which newrb`). We go through this algorithm in detail so that we can modify it to suit our needs. Unfortunately, Matlab's equations are transposed from our setup, so we will include the Matlab versions in regular type.

1. **Basic Idea:**

   (a) Notation Setup: Let $\boldsymbol{\Phi}$ be the full interpolation matrix, where we index the p columns:

   $$\boldsymbol{\Phi} = [\Phi_1, \ldots, \Phi_p]$$

   Let the domain be in $\mathbb{R}^n$ and the range be data in $\mathbb{R}^m$. Form the modified matrix:

   $$\hat{\boldsymbol{\Phi}} = [\Phi_1, \ldots, \Phi_p, \texttt{ones(p,1)}]$$

   so that the corresponding coefficient matrix Ω will now contain the *weights* w_{ij} (a $p \times m$ matrix) and *biases* $\boldsymbol{b}$ (an m-vector):

   $$\hat{\Omega} = \begin{bmatrix} \Omega \\ \boldsymbol{b}^T \end{bmatrix}$$

   (b) Suppose that we begin by assuming that all data points are candidate centers. We look at the RBF equation:

   $$\hat{\boldsymbol{\Phi}}\hat{\Omega} = \boldsymbol{Y}$$

   where $\boldsymbol{Y}$ is a $p \times m$ matrix.

   (c) Think of the columns of $\boldsymbol{\Phi}$ as spanning one dimensional subspaces of $\mathbb{R}^p$. Then the columns of $\boldsymbol{Y}$ also form subspaces of $\mathbb{R}^p$.

   (d) We will choose that subspace of $\boldsymbol{\Phi}$ that most closely spans the column space of $\boldsymbol{Y}$ (as vectors in $\mathbb{R}^p$).

2. Algorithm: The basic algorithm proceeds as follows:
   (a) Look for the column of $\mathbf{\Phi}$ that most closely points in the same direction as a column of $\boldsymbol{Y}$.
   (b) Take that column out of $\mathbf{\Phi}$ (which makes $\mathbf{\Phi}$ smaller), then deflate the remaining columns. Another way to say this is that we remove the component of the columns of $\mathbf{\Phi}$ that point in the direction of our "winner".
   (c) Repeat.
3. **Remark:** Before continuing, let's review the linear algebra that we'll need to perform the operations listed above.
4. **Exercise:** Show that, if we have a set of vectors $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k]$ and a vector $\boldsymbol{y}$, then the vector in X that most closely points in the direction of $\boldsymbol{y}$ (or $-\boldsymbol{y}$) is found by computing the maximum of:
$$\left(\frac{\boldsymbol{x}_i^T \boldsymbol{y}}{\|\boldsymbol{x}_i\| \|\boldsymbol{y}\|} \right)^2 \tag{11.7}$$
5. **Exercise:** First, recall that:
$$\frac{\boldsymbol{v}\boldsymbol{v}^T}{\|\boldsymbol{v}\|^2} \tag{11.8}$$
is an orthogonal projector to the one dimensional subspace spanned by $\boldsymbol{v}$. Using this, show that the following set of Matlab commands will return a matrix whose columns are in the orthogonal complement of the subspace spanned by $\boldsymbol{v}$. That is, $\boldsymbol{v}$ is orthogonal to the columns of the final matrix in the computation:

```
a=v'*P/(v'*v);
P=P-v*a;
```

6. **Remark:** There are some other interesting programming elements in the Matlab suite of radial basis functions. Below, we examine some Matlab "tricks" that will come in handy.
7. **Exercise:** Let X and Y be matrices of the same size, and x and y be vectors. Then explain what each line of commands will do:
   (a) `X(:,3)=[];`
   (b) `x(2)=[];`
   (c) `X=[X Y(:,3)];`
   (d) `x=[x y(2)];`
   (e) `I=1:2:5;  X=Y(:,I);`
   (f) `[m,n]=size(X); X=[X;ones(1,N)];`

(g) `A=X./Y;`

8. **Exercise:** Let p be a $n \times k$ matrix whos *columns* are data points $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$. Let a be the desired spread of the Gaussian (See Equation 11.3). Verify that the (i, j) component of the following Matlab commands is

$$\mathrm{e}^{\frac{-\|\boldsymbol{x}^{(i)}-\boldsymbol{x}^{(j)}\|^2}{\sigma^2}}$$

```
b=sqrt(-log(0.5))/a;
P=radbas(dist(p',p)*b);
```

9. **Exercise:** Show that, if $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k]$ are column vectors, then

$$\texttt{sum(X.*X)=}\left(\|\boldsymbol{x}_1\|^2, \ldots, \|\boldsymbol{x}_k\|^2\right)$$

10. **Exercise:** Write a Matlab function, `mnormal` that will input a matrix, and output the matrix with normalized columns.

11. **Exercise:** Let $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m]$ and $Y = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k]$. Show that the (i, j) component of `X'*Y` is given by:

$$\boldsymbol{x}_i^T \boldsymbol{y}_j$$

12. **Exercise:** Verify that the following Matlab commands solves the (least squares) equation: $wp + b = t$ for w and b:

```
[pr,pc]=size(p);
x=t/[p;ones(1,pc)];
w=x(:,1:pr);
b=x(:,pr+1);
```

13. **Exercise:** What does the following set of Matlab commands do? (Assume M is a matrix)

```
replace=find(isnan(M));
M(replace)=zeros(size(replace));
```

14. **Remark:** We are now ready to examine the Matlab function `newrb` and its main subfunction, `designrb`. These programs use the OLS algorithm [5] described previously. This algorithm will tell us the columns of $\boldsymbol{\Phi}$ that are most critical to the approximation, and therefore will tell us which data points to use as centers.

15. **Matlab Comment:** The function `newrb` is basically a wrapper for the function (included in newrb) `designrb`. The introductory programming lines in `newrb` sets up the RBF neural network, then calls `designrb` to train it.

16. **Remark:** As a programming note, we will be selecting columns from $\mathbf{\Phi}$ and holding them in a matrix W. At this point, you may want to print the function `newrb.m` for reference. The algorithm below is an explanation of it.

Algorithm 11.6.1 *Orthogonal Least Squares for RBFs*

The function design is to input the domain and range (p, t)[1], the error goal and gaussian spread (`eg`, `sp`), and output the centers `w1`, σ's (actually, $\frac{1}{\sigma}$) are stored in `b1`, the coefficients $\omega_{ij} \in$`w2` and the biases $b_j \in$`b2`. The other output arguments are optional.

```
function [w1,b1,w2,b2,k,tr] = designrb(p,t,eg,sp)

[r,q] = size(p);
[s2,q] = size(t);
b = sqrt(-log(.5))/sp;

% RADIAL BASIS LAYER OUTPUTS
P = radbas(dist(p',p)*b);
PP = sum(P.*P)';
d = t';
dd = sum(d.*d)';
```

17. **Exercise:** For each matrix above, write down the size (in terms of "number of points" and "dimension").

So far, we've constructed the interpolation matrix `P`=$\mathbf{\Phi}^T$, and initialized some of our variables. We're now ready to initialize the algorithm:

```
% CALCULATE "ERRORS" ASSOCIATED WITH VECTORS
e = ((P' * d)' .^ 2) ./ (dd * PP');
```

18. **Exercise:** What is the size of `e`? Show that the (j, k) component of `e` is given by (in terms of our previous notation):

$$\frac{\boldsymbol{\phi}_k^T \boldsymbol{y}_j}{\boldsymbol{y}_j^T \boldsymbol{y}_j \boldsymbol{\phi}_k^T \boldsymbol{\phi}_k}$$

In view of Equation 11.7, what does `e` represent?

In the following lines, there is some intriguing code. The integer `pick` is used to store our column choice. The vector `used` will be used to store the columns of $\mathbf{\Phi}$ that have been chosen, and the vector `left` are the columns remaining.

[1]Data is entered as "dimension" by "number of points" in Matlab, which is the transpose of our notation

```
% PICK VECTOR WITH MOST "ERROR"
pick = findLargeColumn(e);
used = [];
left = 1:q;
W = P(:,pick);
P(:,pick) = []; PP(pick,:) = [];
e(:,pick) = [];
```

The matrix `W` is initialized as the best column of $\mathbf{\Phi}$. The matrices `P` and `e` has their corresponding column removed, `PP` has one element removed. Now update `used` and `left`.

```
used = [used left(pick)];
left(pick) = [];

% CALCULATE ACTUAL ERROR
w1 = p(:,used)';
a1 = radbas(dist(w1,p)*b);
[w2,b2] = solvelin2(a1,t);
a2 = w2*a1 + b2*ones(1,q);
sse = sumsqr(t-a2);
```

19. **Exercise:** Explain each of the previous lines. What is the difference between `W` and `w1`?

    We now begin the MAIN LOOP:

```
for k = 1:(q-1)

  % CHECK ERROR
  if (sse < eg), break, end

  % CALCULATE "ERRORS" ASSOCIATED WITH VECTORS
  wj = W(:,k);
  a = wj' * P / (wj'*wj);
  P = P - wj * a;
  PP = sum(P.*P)';
  e = ((P' * d)' .^ 2) ./ (dd * PP');
```

20. **Exercise:** Explain the previous lines of code. To assist you, you may want to refer back to Equation 11.8.

```
  % PICK VECTOR WITH MOST "ERROR"
  pick = findLargeColumn(e);
  W = [W, P(:,pick)];
  P(:,pick) = []; PP(pick,:) = [];
```

```
  e(:,pick) = [];
  used = [used left(pick)];
  left(pick) = [];

  % CALCULATE ACTUAL ERROR
  w1 = p(:,used)';
  a1 = radbas(dist(w1,p)*b);
  [w2,b2] = solvelin2(a1,t);
  a2 = w2*a1 + b2*ones(1,q);
  sse = sumsqr(t-a2);
end

[S1,R] = size(w1);
b1 = ones(S1,1)*b;
```

21. **Exercise:** If you wanted to be able to output the training errors, how would you modify `designrb`?

22. **Exercise:** The following two subfunctions are also included in `newrb.m`. Go through and explain each line.

```
%=========================================================
function i = findLargeColumn(m)

replace = find(isnan(m));
m(replace) = zeros(size(replace));

m = sum(m .^ 2,1);
i = find(m == max(m));
i = i(1);

%=========================================================

function [w,b] = solvelin2(p,t)

if nargout <= 1
  w= t/p;
else
  [pr,pc] = size(p);
  x = t/[p; ones(1,pc)];
  w = x(:,1:pr);
  b = x(:,pr+1);
end
```

23. **Exercise:** Try using RBFs for interpolating a picture. Load the image `clown2.mat`. Think of the image as the graph of a function, then use RBFs to interpolate.

11.7 Homework: Iris Classification

1. Read over and understand the script file attached.
2. Type in the script file.
3. Make appropriate changes so that once the domain set X is loaded, perform a KL reduction to the best three dimensional space. This will be the new domain for the RBF.
4. Run both the original script and the new script, and compare your answers.

11.8 Sample Script File

```
%Script for rbf example on IRIS data

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%The following are handy to name as variables, rather
%than hard-coding them.   NUMITERS is the number of
%steps through the Gaussian width (so we train the
%network that many times).  NUMTRAIN is the number of
%training points we'll use from the data.
%
%Further down, load the data IRIS.NET, and load
%the random indices for the data.  We could call
%randr, but to keep the same random set for more
%than one sessions, the indices were saved.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
numiters=20;
numtrain=120;

X=load('IRIS.NET');
Y=X(:,5:7);
X(:,5:7)=[];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  eg = error goal for the RBF
%  sp = Gaussian spread for the RBF
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

eg=0.1;
sp=linspace(0.05,1.6,numiters);
```

```
load rand001
ii=RandInd;
P=X(ii(1:numtrain),:)';
T=Y(ii(1:numtrain),:)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  This suppress those annoying "Your matrix is
%  singular" warnings, which we expect to get here...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
warning off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  MAIN LOOP:
%  Train the network on the subsets P and T
%  Simulate the network on the full data set X
%  Get the error in the classification (e)
%  Get the number of weights (c)
%  Get the maximum weight size (W)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


for i=1:numiters
i
net=newrb(P,T,eg,sp(i));
z=sim(net,X');
e(i)=0;

[c(i),b]=size(net.IW{1});

for k=1:150
[x,j]=max(z(:,k));
if Y(k,j)~=1
e(i)=e(i)+1;
end
end
W(i)=max(max(abs(net.LW{2,1})));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Plotting routines
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

subplot(2,2,1)
```

```
plot3(X(1:50,1),X(1:50,2),X(1:50,3),'r*')
hold on
plot3(X(51:100,1),X(51:100,2),X(51:100,3),'b^')
plot3(X(101:150,1),X(101:150,2),X(101:150,3),'go')
hold off
subplot(2,2,2)
plot(sp,e)
title('Error versus Gaussian Width')
subplot(2,2,3)
plot(sp,c)
title('Number of centers from OLS versus Gaussian Width');
subplot(2,2,4)
semilogy(sp,W)
title('Maximum Weight versus Gaussian Width');
```