2. PRINCIPLES OF OPERATION

The Ni1000 Recognition Accelerator is optimized for use in systems that require fast classification capabilities. Classification is the process of associating input data with categories or *classes.* In an optical character recognition application, for example, classification would consist of identifying the characters represented in data scanned off an image sensor. The output could be ASCII character codes corresponding to the images perceived by the sensor.

A recognition system needs a memory that will allow it to distinguish among the classes. Generally, an algorithm produces such memory. The system studies a sample of inputs typical to the problem, called the *training set*, and learns the differences between classes based on their characteristics. The training set is a collection of input patterns and responses (i.e., expected class identification) typical to the problem. Learning is usually a phase in product development; classification is the operational mode. The Ni1000 Recognition Accelerator also facilitates on-chip learning for systems that require additional adaptation in the field.

2.1 Pattern Recognition

To illustrate the recognition process, consider a hypothetical industrial inspection task of separating nuts from bolts out of a stream of hardware moving on a conveyor past a set of sensors. The task of the recognition system developer is to devise a system that takes data from the sensors and assigns it a *class*, e.g., nut or bolt. A system that can accomplish the task appears in Figure 2-1. Before entering the classifier, data output by the sensors may need to pass through a *pre-processor* that extracts *features,* such as the weight, size, shape, or aspect ratio of each piece of hardware. Each property is then a component of the *feature vector*. For example, the size is one feature in the feature vector; shape is another. The classification engine processes the vector and completes the recognition by making a sorting decision.

Components of feature vectors tend to have random individual values. All bolts do not have the same weight, but their weight does have both upper and lower limits. Weighing a representative sample of bolts and charting the results to show the weight variation among bolts produces a Probability Density Function (PDF).

In our example, nuts are generally lighter than bolts. The two hypothetical PDF graphs appear on the same axes in Figure 2-2. Using Bayes Rule, the system sorts the parts into the class with the higher PDF value. In Figure 2-2, pieces with weights below point C are likely to be nuts. Those heavier than C are probably bolts. Since virtually no bolts weigh less than B and no nuts weigh more than C, on the [A, B] and [C, D] intervals the decision is fairly simple.

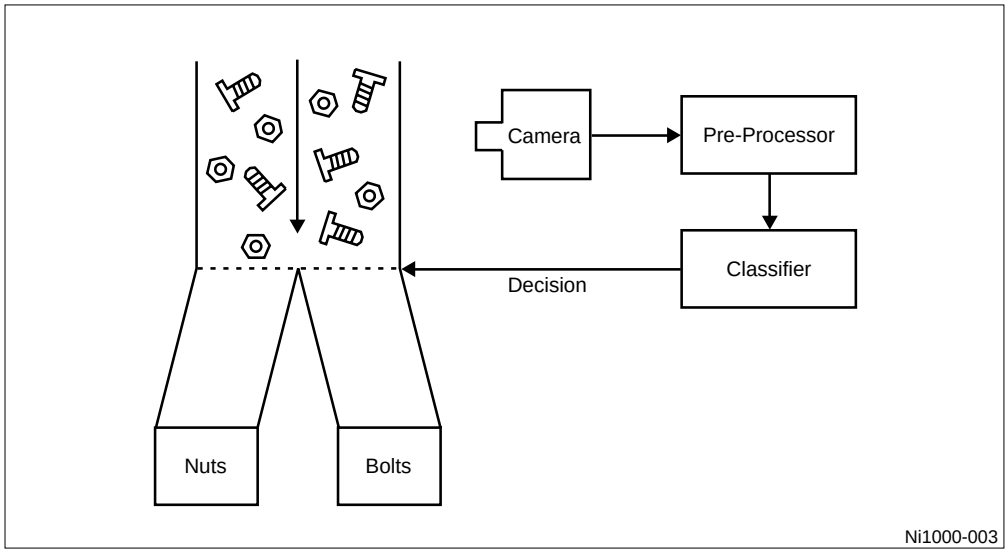

Figure 2-1. Hypothetical Pattern Recognition System

However, on [B, C] both choices are possible, and the system picks the one with the higher PDF. If the input is outside the interval [A, D], it is something unexpected, and the system classifies it as neither a bolt nor a nut. The system thus exhibits novelty detection capability.

Additional features provide additional information to the classifier. The Ni1000 Recognition Accelerator can handle input vectors with up to 256 dimensions.

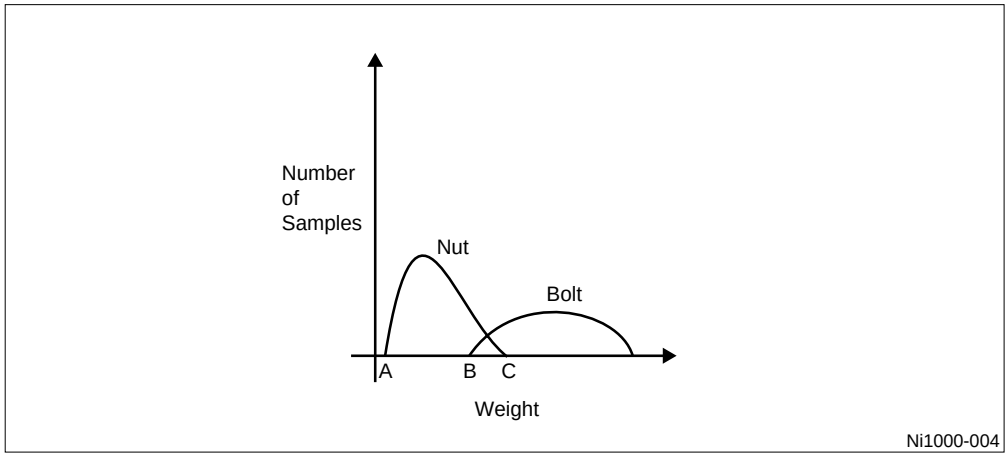

Figure 2-2. PDFs For Nut and Bolt Weights

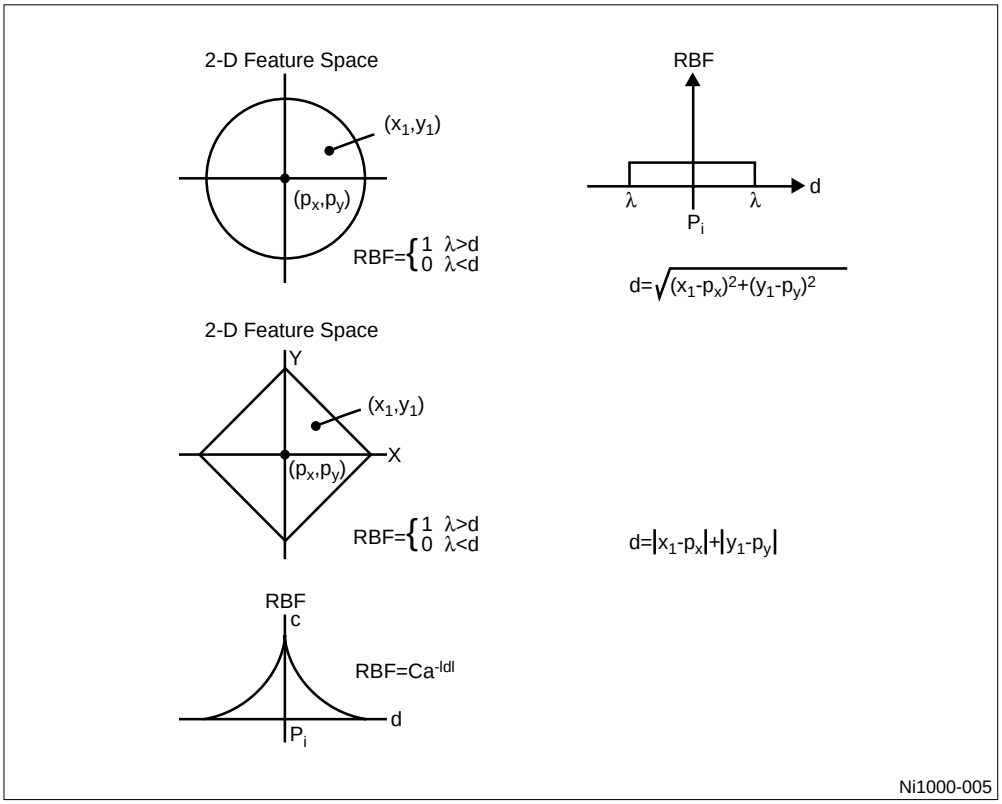

Figure 2-3. RBF Examples

Input vectors describe points in a multidimensional *feature space*, which is the complete range of possible patterns of input data. In the example of the nut and bolt sorter, the axes of the feature space are the inputs from the sensors; they may include size, color, weight, and shape of each piece of hardware. Features belonging to each class tend to cluster into regions just as nuts in the example have similar weights. The learning process approximates the locations of the regions in feature space using Radial Basis Functions (RBF). Three examples of RBFs appear in Figure 2-3. An input close to the center of an RBF elicits a large response. Inputs far from the center produce insignificant responses. Figure 2-4 illustrates a two dimensional class region approximated using circular *fields of influence* centered on stored examples.

The classification process maps input vectors onto feature space. The classifier then outputs the class of the region into which the input fell. If multiple classes fire, as may occur when an input falls into overlapping regions, probabilistic information can help resolve ambiguities.

Examples of radial fields of influence are common in biological systems. Neuroscientists have found that a ganglion cell in the retina responds only to light detected by a small, circular area called the cell's receptive field. The cell produces no response, or it is actually inhibited from producing a response if it detects light outside the area. Receptive fields of adjacent cells tile the retina in overlapping RBF receptive areas to cover regions in feature space.

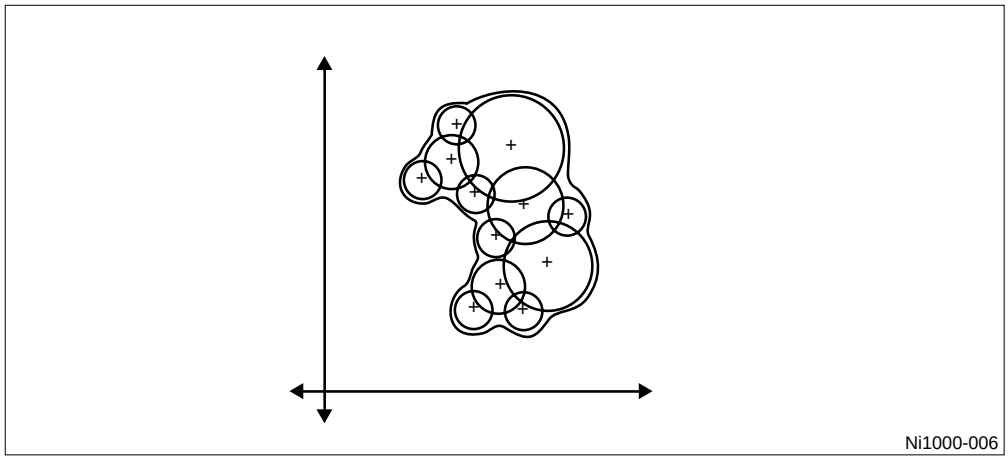

Figure 2-4. Approximating Feature Space with RBFs

2.2 Learning and Classification Algorithms

During the learning process, the recognition system develops a memory of class-region approximations and probability-density-function estimates for each point in feature space. The system studies a *training set* of input examples, along with their class labels, and learns to distinguish among the classes under the control of a learning algorithm.

The Ni1000 Recognition Accelerator's on-chip microcontroller is intended to execute the learning algorithm code. It facilitates incremental learning outside the factory to adapt and customize the chip's memory to special circumstances that arise in the field. The Accelerator supports algorithms like Restricted Coulomb Energy (RCE) and Probabilistic RCE (PRCE) as well as other radial basis function algorithms like Probabilistic Neural Networks (PNN) and custom algorithms.

Alternatively, the prototypes and their parameters can be loaded into the chip from outside. They may be the result of learning performed off chip, or they may simply consist of data that can take advantage of the calculations performed by the classification pipeline.

2.2.1 Restricted Coulomb Energy (RCE)

In RCE and other algorithms compatible with the Ni1000 Recognition Accelerator, learning is a process of approximating class regions in feature space with radial basis functions. The algorithm selectively stores a set of prototypical inputs, called *prototypes*, that are obtained from the training data, and it assigns to the field of influence of each prototype a radius called *lambda (λ)* or *threshold distance*.

Before learning begins, the designer specifies minimum and maximum values for lambda. During learning, the Accelerator computes the distance between each input vector and any existing prototypes. Distances are defined as the sum of the differences between each component (dimension) of an input vector, *u*, and the corresponding component (dimension) of

a stored prototype vector, *p*. This metric, *d,* is called the *city-block distance* or *Manhattan distance* and is computed as:

$$d = \sum_{0 \le i \le 256} |u_i - p_i| \qquad (1)$$

The algorithm then compares the distances with each prototype's lambda (λ) to determine whether or not the training input vector (which has an associated class) is within that prototype's field of influence. If the input vector does not fall within the field of influence of any prototype, the Accelerator stores it as a new prototype along with its class label and sets its lambda to λ_{max}. If, however, the input is within the field of influence of a prototype in a class different than the one with which the input is tagged, the input is stored and both its lambda and the prototype's lambda are set to the distance between them. The algorithm does not store the input as a new prototype if it only falls within the field of influence of one or more prototype(s) in the same class. Instead, it increments the count for each firing prototype. Iterations through the data set continue until prototype storage and lambda adjustments stop.

The pseudo-code for a typical RCE or PRCE training procedure is shown below. In the procedure, the italicized steps are for PRCE only. All others apply to both RCE and PRCE. The on-chip microcode supplied with the development system performs both RCE and PRCE specifications during learning.

```
{  // Learn RCE/PRCE
Set λmin and λmax
do
     {  // begin epoch
     Reset Ck's
     do
          {  // learn vector
          Input next vector and its associated class (classi).
          Compute the input vector's distance to each of the stored prototypes.
          Compare distances to corresponding prototype's lambdas and determine firings.
          Compute Dmin using the Dmin calculation procedure shown below.
          If ((no prototypes of classi exist) or (no prototypes fire))
               store input vector with λ = Dmin
          else for k = 1 to khighest stored
               if (Pk firing and (class of Pk = classi)) then Ck = Ck + 1
               if (Pk firing and (class of Pk != classi) and (λ of Pk != λmin )) then
                    {
                    if (distance to Pk > λmin ) then λ of Pk = distance to Pk
                    else λ of Pk = λmin
                    }
               }
          } while more input vectors are available // learn vector
     } while ((new prototypes were stored) or (any λ changed)   // end of epoch
}  // Learn RCE/PRCE
```

In the above pseudo-code, D_{min} is calculated as:

$D_{min} = \lambda_{max}$
For k = 1 ...

During classification, the algorithm computes the distances from the input vector to each of the prototype vectors stored during learning. If the distance to a prototype is less than the prototype's lambda, the input receives the prototype's class label. The result of the RCE classification is a union of all firing classes. Probabilistic methods like PRCE described below can resolve ambiguities in case of multiple firings.

2.2.2 Probabilistic RCE (PRCE)

PRCE outputs the probability that an input belongs to a given class. While learning, the classifier stores prototypes and computes lambdas (λs) using the RCE algorithm. In classification, it computes probability density estimates throughout feature space for each class. The Ni1000 Accelerator performs the probabilistic calculations in parallel with the class firing calculations used in RCE.

The PRCE algorithm finds its roots in Bayes Decision Theory. A Bayesian classifier computes an input's probability of belonging to each class by using corresponding class probability density functions (PDF's). In a simple two-category problem in which class A and class B are the possible categories, an input vector *u* is a member of:

Class A when $C_A * f_A(u) > C_B * f_B(u)$ (2a)

or

Class B when $C_A * f_A(u) < C_B * f_B(u)$ (2b)

where C_A and C_B are the *a priori* probabilities of pattern occurrence from category A and B, respectively. The *a priori* probability C_A is the ratio of the number of training patterns belonging to class A, to the total number of training patterns, and $C_B = 1 - C_A$. Functions f_A and f_B are the probability density functions for class A and class B, respectively.

The construction of decision boundaries requires knowledge of the underlying PDF's, which must be determined through training. The PDF for each class may be constructed from a linear combination of a family of radially symmetric distribution functions, each centered on a prototype stored during training. Decaying exponentials shown in Figure 2-5 approximate each prototype's contribution to the PDF of its class:

$$PDF_{Class} = \sum_{P_j \in Class} C_j \cdot 2^{-k_j \cdot d_j} \quad (3)$$

where *d* is the Manhattan distance between the input and the prototype, *k* is a decay constant specified by the host program before initiating PRCE classification, and C_j is the *a priori* rate obtained during training. C_j is the number of training patterns that belong to a class that fell within the distance λ_j of the prototype p_j. An example of the resultant PDFs appears Figure 2-6.

The PRCE classification is a mapping of the input vector onto feature space, resulting in a set of PDF values (one value for each class). The host can then select the largest PDF value for each vector to determine its class.

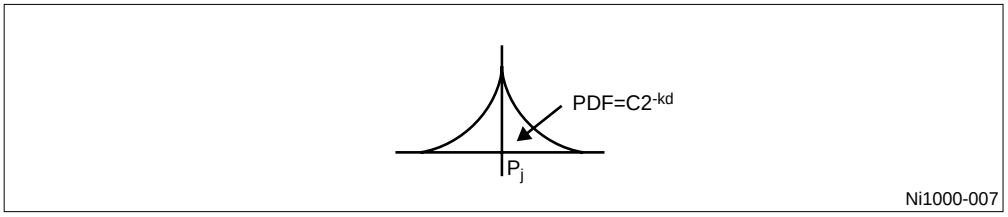

Figure 2-5. PDF Contribution of a Single Prototype

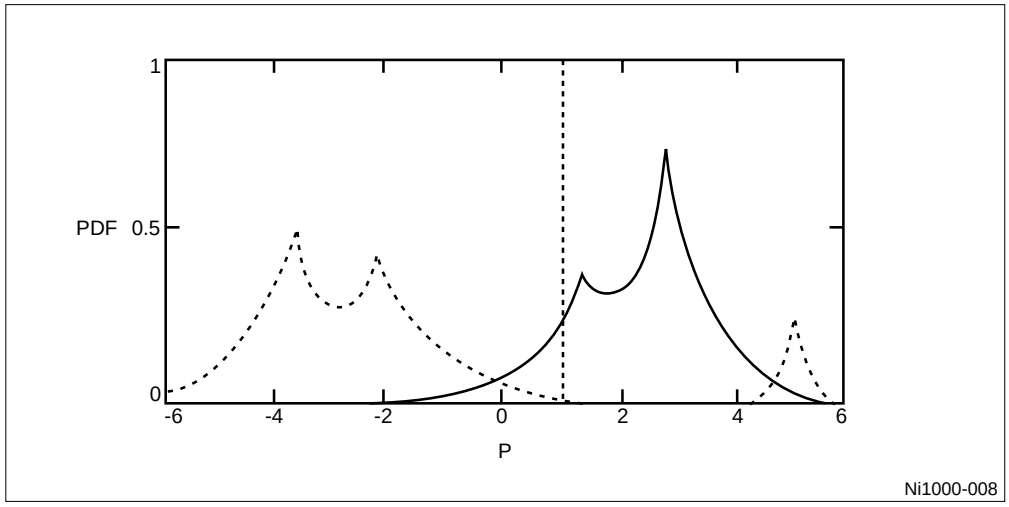

Figure 2-6. Hypothetical PDFs for a Two-Class Problem