Chapter 14

Probabilistic neural networks: a brief overview of theory, implementation, and application

Behshad Mohebali[1], Amirhessam Tahmassebi[1], Anke Meyer-Baese[1], Amir H. Gandomi[2,3]

[1]*Department of Scientific Computing, Florida State University, Tallahassee, FL, United States;* [2]*Faculty of Engineering and Information Technology, University of Technology Sydney, Ultimo, Australia;* [3]*School of Business, Stevens Institute of Technology, Hoboken, NJ, United States*

1. Introduction

Probabilistic neural networks (PNNs) are a group of artificial neural network built using Parzen's approach to devise a family of probability density function estimators (Parzen, 1962) that would asymptotically approach Bayes optimal by minimizing the "expected risk," known as "Bayes strategies" (Mood, 1950). In a PNN, there is no need for massive back-propagation training computations. Instead, each data pattern is represented with a unit that measures the similarity of the input patterns to the data pattern. PNNs have shown great potential for tackling complex scientific and engineering problems. Major categories of issues that researchers have attempted to address using PNN are as follows:

- Classification of labeled stationary data patterns
- Classification of data patterns where the data have a time-varying probabilistic density function
- Corresponding author email address: a.h.gandomi@stevens.edu (Amir H. Gandomi)
- Signal processing applications, dealing with waveforms as data patterns
- Unsupervised algorithms that work with unlabeled data sets

Handbook of Probabilistic Models. https://doi.org/10.1016/B978-0-12-816514-0.00014-X

The first category might be the simplest type among the mentioned four. Here, the assumption is that the probability density function of the data does not have significant and meaningful variations through the life span of the network. In Bankert (1994), the researchers used a PNN based algorithm to classify 16 × 16 pixel pictures of clouds into 10 categories. A set of 95 labeled pictures were used to form the PNN. In Wu et al. (2007), PNN is used to classify 1800 pictures of leaves into 32 categories of plants based on their geometrical properties. Although the feature extraction process in the latter is more sophisticated and application dependent, the underlying principles are the same as what was introduced in Specht (1990). In Nishanth and Ravi (2016), the authors used PNN to impute the missing categorical features in an incomplete data set. This application relies on PNNs' ability to produce satisfactory results even with small data sets.[1]

The assumption of time invariance may not hold in most of the practical use cases and the probability density function of the data shows nontrivial changes over time. In that regard, Rutkowski (2004) has introduced an adaptive PNN that can track the changes in the PDF of the data and adjust its inner parameters to take those changes into account. This approach has been used in Hazrati and Erfanian (2010) to recognize EEG signals coming from experiment subjects when they "think" about closing their hands to grab a virtual ball in virtual reality.

Another important category of applications for PNNs is signal processing. This can be recognizing the occurrence of an event (Tripathy et al., 2010), the prediction of severity of an event (Adeli and Panakkat, 2009; Asencio-Cortés et al., 2017), using the time-domain waveform of a parameter of interest, or classifying a set of events after preprocessing is done on the waveforms and the features are extracted as in Mishra et al. (2008) and Wang et al. (2013). In Tripathy et al. (2010), PNN with optimized smoothing factor is used to distinguish between two events: magnetizing inrush and internal fault, each of which would warrant a different course of action. In signal processing applications, raw data are usually a time-domain waveform or a wavelet. This means each data pattern might consist of hundreds of data points. In such scenarios, the role of preprocessing, feature extraction, and feature reduction becomes more prominent as these procedures can decrease the computational burden while improving the overall performance of the network at the same time. For example, in Wang et al. (2013), the data patterns are windows of 200 data points around the R peaks on an ECG waveforms obtained from multiple subjects with the sampling rate of 360 Hz. The window is chosen in a way so that the significant features of ECG

1. For example, if the data rows that actually have a value for a certain categorical feature is a small percentage of all the data rows.

beat that would come before and after the R peak are included. Then, the 200 elements of the data vector are normalized using z-score. Principal component analysis (PCA) and linear discriminant analysis are then applied to the normalized data to reduce the number of features that would be fed to the network while maximizing their discriminatory potential. The data are then used to categorize the waveforms into eight types of ECG beats.

Mishra et al. (2008) uses PNN to distinguish 11 types of disturbances in power quality[2] using waveforms of voltage magnitude, frequency, and the phase of the power supply. First, the S-transform of the waveforms is calculated. Then, the covariance matrix of the S-transform matrix (called S-matrix) is calculated. The extracted features are based on standard deviation and energy of the transformed signals as follows:

- Standard deviation of the vector that is formed by selecting the maximum value on each column of the S-matrix
- The energy of the vector that is formed by selecting the maximum value on each column of the S-matrix
- Standard deviation of the vector that is formed by selecting the maximum value on each row of the S-matrix
- Standard deviation of the phase contour

Mishra et al. (2008) shows the importance of feature selection by demonstrating that a PNN can work properly even with small number of features as long as the features contain enough discriminatory information about the model that the data represent.

Although PNNs need labeled data to operate, unlabeled data can be used in special cases after being labeled by unsupervised techniques. In Song et al. (2007), the unlabeled, unstructured set of MRI images are labeled by a self-organizing map (SOM). The SOM algorithm "softly" labels the images in the sense that each image can contribute to multiple classes. However, the degree of contribution to each class might be different.

The rest of this chapter serves as an application-oriented introduction to PNNs and is structured as follows. First, the fundamental statistical concepts that were introduced by Parzen (1962) and used by Specht (1990) to develop PNNs are briefly reviewed in Section 2. Then, Section 3 shows how PNNs use those mentioned concepts for classification. Section 4 deals with some of the practical challenges of implementing PNNs. A simple example of a PNN classifier written in Python is included in Section 5. This chapter is concluded in Section 6.

2. Power quality is a measure of how steady the power supply is. It shows how closely the nominal values of the voltage magnitude and frequency and the sinusoidal waveform is followed by the output of the power supply.

2. Preliminary concepts: nonparametric estimation methods[3]

For classification tasks, we need to estimate class-related PDFs as they determine the classifier's structure. Thus, each PDF is characterized by a certain parameter set. For Gaussian distributions, the covariance and the mean value are needed, and they are estimated from the sample data. Let us also assume that we have a set of training sample representative of the type of features and underlying classes, with each labeled as to its correct class. This yields a learning problem. When the form of the densities is known, we are faced with a parameter estimation problem.

In nonparametric estimation, there is no information available about class-related PDFs, and so they have to be estimated directly from the data set. There are many types of nonparametric techniques for pattern recognition. One procedure is based on estimating the density functions $p(\mathrm{x}|\omega_i)$ from sample patterns. If the achieved results are good, they can be included in the optimal classifier. Another approach estimates directly the posteriori probabilities $P(\omega_i|\mathrm{x})$, and is closely related to nonparametric decision procedures. They bypass probability estimation and go directly to decision functions.

The following nonparametric estimation techniques related in its concepts to PNN will be reviewed:

- Parzen windows
- k nearest neighbor
- Potential function

2.1 Parzen windows

One of the most important nonparametric methods for PDF estimation is "Parzen windows" (Meisel, 1972; Poggio and Girosi, 1990). For a better understanding, we will take the simple one-dimensional case. The goal is to estimate the PDF $p(x)$ at the point x. This requires to determine the number of the samples N_h within the interval $[x - h, x + h]$ and then to divide by the total number of all feature vectors M and by the interval length $2h$. Based on the described procedure, we will obtain an estimate for the PDF at x

$$\widehat{p}(x) = \frac{N_h(x)}{2hM} \tag{14.1}$$

As a support function k_h, we will choose

$$K_h = \begin{cases} 0,5 & : \quad |m| \leq |1| \\ 0 & : \quad |m| > |1| \end{cases} \tag{14.2}$$

3. This section is taken from Meyer-Baese and Meyer-Baese (2004).

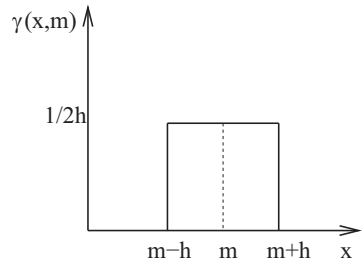

FIGURE 14.1 Clustering process of a two-dimensional vector table. *From Anke Meyer-Bäse, Statistical and syntactic pattern recognition. In Pattern recognition for medical imaging, 147–255, 2004.*

From Eq. (14.1) we get

$$\widehat{p}(x) = \frac{1}{hM} \sum_{i=1}^{M} K\left(\frac{x - m_i}{h}\right) \tag{14.3}$$

with the ith component of the sum being equal to zero if m_i falls outside the interval $[x - h, x + h]$. This leads to

$$\gamma(x, m) = \frac{1}{h} K\left(\frac{x - m}{h}\right) \tag{14.4}$$

as it can be seen from Fig. 14.1.

If $\widehat{p}(x)$ is considered to be a function corresponding to the number of samples, we obtain thus

$$\widehat{p}(x) = \widehat{p}(x, M) \tag{14.5}$$

Parzen showed that the estimate $\widehat{p}$ with $M \to \infty$ is bias free, if $h = h(M)$ and

$$\lim_{x \to \infty} h(M) = 0 \tag{14.6}$$

In practice, where only a finite number of samples are possible, a right compromise between M and h has to be made. The choice of h is crucial, and it is recommended to start with an initial estimate of h and then modify it iteratively to minimize the misclassification error. Theoretically, a large M is necessary for acceptable performance. But in practice, a large number of data points increase the computational complexity unnecessarily.

Typical choices for the function $K(m)$ are

$$K(m) = (2\pi)^{\frac{-1}{2}} e^{-\frac{m^2}{2}} \tag{14.7}$$

$$K(m) = \frac{1}{\pi(1 + m^2)} \tag{14.8}$$

or

$$K(m) = \begin{cases} 1 - |m| & : \quad |m| \leq |1| \\ 0 & : \quad |m| > |1| \end{cases} \tag{14.9}$$

2.2 k nearest neighbor density estimation

In the Parzen windows estimation, the length of the interval is fixed, while the number of samples falling inside an interval varies from point to point. For the k nearest neighbor density estimation, exactly the reverse holds: the number of samples k falling inside an interval is fixed, while the interval length around x will be varied each time, to include the same number of samples k. We can generalize for the n-dimensional case: in low density areas the hypervolume $V(\mathrm{x})$ is large, while in high density areas it is small.

The estimation rule can be given now as

$$\widehat{p}(\mathrm{x}) = \frac{k}{NV(\mathrm{x})} \tag{14.10}$$

and reflects the dependence of the volume $V(\mathrm{x})$. N represents the total number of samples, while k describes the number of points falling inside the volume $V(\mathrm{x})$.

This procedure can be very easily elucidated based on a two-class classification task: an unknown feature vector x should be assigned to one of the two classes ω_1 or ω_2. The decision is made by computing its Euclidean distance d from all the trainings vectors belonging to various classes. With r_1, we denote the radius of the hypersphere centered at x that contains k points from class ω_1, while r_2 is the corresponding radius of the hypersphere belonging to class ω_2.V_1 and V_2 are the two hypersphere volumes.

The k nearest neighbor classification rule in case of two classes ω_1 and, respectively, ω_2 can now be stated

$$\text{Assign x to class } \omega_1(\omega_2) \text{ if } \frac{V_2}{V_1} > (<) \frac{N_1}{N_2} \frac{P(\omega_2)}{P(\omega_1)} \tag{14.11}$$

If we adopt the Mahalanobis distance instead of the Euclidean distance, then we will have hyperellipsoids instead of hyperspheres.

2.3 Potential functions

Potential functions represent a useful method for estimating an unknown PDF $\widehat{p}(x)$ from the available feature vectors (Andrews, 1972 and Meisel, 1972). The estimated PDF is given by a superposition of potential functions $\gamma(\mathbf{x},\mathbf{m})$

$$\widehat{p}(x) = \frac{1}{M} \sum_{j=1}^{M} \gamma(\mathbf{x}, \mathbf{m}_j) \tag{14.12}$$

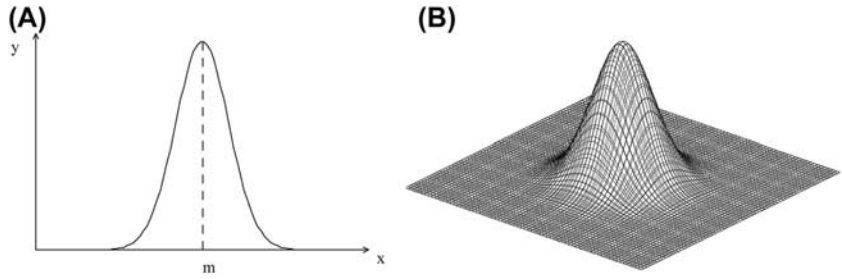

FIGURE 14.2 Potential functions for the (A) one-dimensional and (B) two-dimensional case. *From Anke Meyer-Bäse, Statistical and syntactic pattern recognition. In Pattern recognition for medical imaging, 147–255, 2004.*

Fig. 14.2A and B illustrate an example of a potential function for the one-dimensional and two-dimensional cases.

A possible potential function is

$$\gamma(\mathbf{x},\mathbf{m}) = \frac{1}{(2\pi)^{\frac{n}{2}}\sigma^n} e^{\left(\frac{-\|\mathbf{x}-\mathbf{m}\|^2}{2\sigma^2}\right)} \tag{14.13}$$

where $\|\mathbf{x}\|$ is a norm in the n-dimensional space. The potential function defines a distance measure (Mahalanobis distance) between two feature vectors $\mathbf{x}$ and $\mathbf{m}$.

Eq. (14.12) describes the complete algorithm for a prespecified potential function. $\widehat{p}(x)$ can be estimated for every x directly from Eq. (14.12).

The choice of the potential function is not so trivial because the width of the potential function plays herein an important role. The smaller its width, the more it peaks and the higher it is. This means it considers only feature vectors in its immediate neighborhood. Larger widths produce a potential function of a smoother shape, as it can be seen from Fig. 14.3.

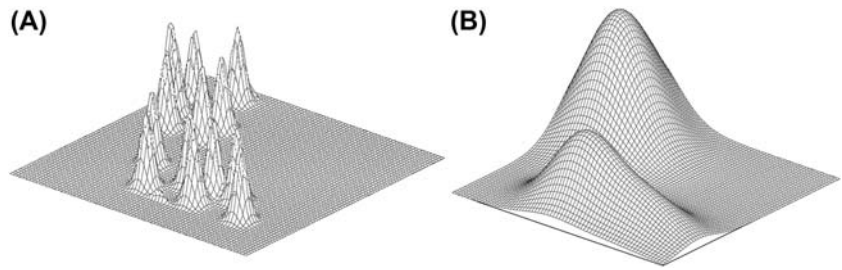

FIGURE 14.3 Importance of the width of the potential function (A) sharper surfaces and (B) smoother surfaces. *From Anke Meyer-Bäse, Statistical and syntactic pattern recognition. In Pattern recognition for medical imaging, 147–255, 2004.*

The choice of a certain variance has a considerable importance on the overlapping degree of neighboring potential functions. This is especially critical when potential functions describing feature vectors belonging to different classes overlap.

There are several possible ways of determining σ Batchelor (1974):

1. Let us assume that within the distance σ from a specific feature vector, there are L other feature vectors. The average value describes the distance $D_L(\mathbf{m})$ from the feature vector to the Lth feature vector:

$$\sigma = \frac{1}{M}\sum_{i=1}^{M} D_L(\mathbf{m_i}) \tag{14.14}$$

To determine L, we have to take into account the distribution of the feature vectors. In many practical problems, $L = 10$ is a good choice.

2. σ can also be chosen as a multiple of the minimal distance between two feature vectors. This is necessary to achieve a certain overlapping degree. In Batchelor (1974), it is recommended to set the multiple equal to 4.

The potential function has the following general properties (Meisel, 1972):

1. $\gamma(\mathbf{x},\mathbf{m})$ has its maximum at $\mathbf{x} = \mathbf{m}$.
2. $\gamma(\mathbf{x},\mathbf{m})$ goes asymptotically toward zero if the distance between the two feature vectors is very large. This is of special importance for multimodal distributions.[4]
3. $\gamma(\mathbf{x},\mathbf{m})$ is a continuous function decreasing monotonically on both sides from the maximum.
4. If $\gamma(\mathbf{x}_1,\mathbf{m}_1) = \gamma(\mathbf{x}_2,\mathbf{m}_1)$, then the feature vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ have the same similarity degree with respect to $\mathbf{m}_1$.

There are several types of known potential functions: besides the above-mentioned unimodal or multimodal normal distributions, potential functions built from orthonormal functions are also of interest (Meisel, 1972). They have the following form

$$\gamma(x,m) = \sum_{i=1}^{R} \lambda_i^2 \Phi_i(x)\Phi_i(\mathbf{m}) \tag{14.15}$$

where Φ_i(x) is an orthonormal function and λ is a constant. Orthonormal functions fulfill

$$\int \Phi_i(x)\Phi_j(x)dx = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases} \tag{14.16}$$

with $\lambda_i = 1$.

4. There are more than one clusters for each class.

It is easy to see that they are potential functions because

$$\sum_{i=1}^{\infty} \Phi_i(\mathrm{x})\Phi_i(\mathbf{m}) = \delta(\mathrm{x} - \mathbf{m}) \tag{14.17}$$

where $\delta(\mathbf{x})$ is a Dirac function and $\{\Phi_i\}$ describes a set of orthonormal functions.

The resulting discriminant function can be determined from Eq. (14.16) and is given by

$$\widehat{p}(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M} \gamma(\mathbf{x}, \mathbf{m}_i) = \frac{1}{M}\sum_{i=1}^{M}\sum_{k=1}^{R} \Phi_k(\mathbf{x})\Phi_k(\mathbf{m}_i) \tag{14.18}$$

where $\widehat{p}(\mathbf{x})$ is the estimated PDF, that is,

$$\widehat{p}(\mathbf{x}) = \sum_{k=1}^{R} c_k \Phi_k(\mathbf{x}) \tag{14.19}$$

The estimated coefficients c_k are

$$c_k = \frac{1}{M}\sum_{i=1}^{M} \Phi_k(\mathbf{m}_i) \tag{14.20}$$

It's important to note that the potential function estimator is both unbiased and asymptotically consistent. We also can easily see that the potential function method is related to Parzen windows. In fact, the smooth function used for estimation is known as either *kernels* or potential functions or Parzen windows.

3. Structure of probabilistic neural networks

To better understand the inner mechanism of the PNNs, one has to look back to Parzen (1962), in which Parzen showed that the probability density function of a set of random variables $X_1, X_2, \ldots, X_n$ with unknown PDF $f(X)$ is estimated with a family of estimators in the form of

$$f_n(x) = \frac{1}{nh(n)}\sum_{j=1}^{n} K\left(\frac{x - X_j}{h(n)}\right) \tag{14.21}$$

where $h(n)$ is a sequence of numbers that satisfies:

$$\lim_{n\to\infty} h(n) = 0 \tag{14.22}$$

Fig. 14.1 shows a simple example of a function $K(y)$ that can be used in Eq. (14.21). However, $K(y)$ is generally a Borel function that satisfies these conditions:

$$1.\ \sup_{-\infty<y<\infty} K(y) < \infty \tag{14.23}$$

$$2.\ \int_{-\infty}^{\infty} |K(y)|dy < \infty \tag{14.24}$$

$$3.\ \lim_{y\to\infty} |yK(y)|dy = 0 \tag{14.25}$$

$$4.\ \int_{-\infty}^{\infty} K(y)dy = 1 \tag{14.26}$$

then

$$\lim_{n\to\infty} E[f_n(x) - f(x)]^2 = 0 \tag{14.27}$$

Eq. (14.27) shows that the Parzen's family of PDFs can estimate the unknown PDF of variable X as $n \to \infty$. The reader is encouraged to study Parzen (1962) for its fundamental importance for PNNs. Specht used this concept in Specht (1990) to formulate an approach to classify patterns of data with unknown class based on initial set of patterns, the real class of which is known.

To explain how a PNN will address that task, consider a sequence of independent identically distributed pairs of random variables as $\{X_i,Y_i\}, i = 1,2, \ldots, n$, known as the training sequence, where $Y_i \in \{1,2, \ldots, M\}$ is the class associated with the pattern $X_i \in A \subset R^p$. The random variable X_i has the unknown probability density function $f_m(x)$ based on its class (fm(x) and is called "class conditional density" in Rutkowski (2004)).

Define the discriminant function of class j as follows:

$$d_j(x) = p_i f_j(x) \tag{14.28}$$

where p_j is the prior probability of class j and is given by:

$$p_j = \frac{n_j}{n} \tag{14.29}$$

where n is the number of patterns in the training sequence and n_j is the number of patterns belonging to class j. $f_j(x)$ is the probability density function of variable x. The class of a given pattern x with unknown class is determined as m if:

$$\begin{aligned} &d_m(x) > d_j(x) \\ &\forall j \in \{1,2,\ldots,M\}, j \neq m \end{aligned} \tag{14.30}$$

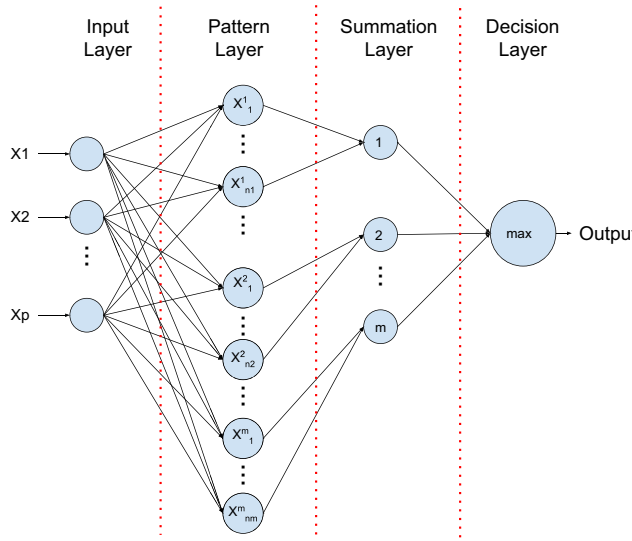

FIGURE 14.4 The basic structure of a probabilistic neural network according to Specht (1990). The network has four basic layers: the input layer that grabs and distributes the input vector, the pattern layer that applies the kernel to the input, the summation layer that gets the average of the output of the pattern units for each class, and the decision layer that declares the class assigned to input vector based on the unit with the maximum output from the summation layer.

Since $f_j(x)$ is usually unknown, its estimator is used in its stead. Using Eq. (14.21), we can estimate $f_j(x)$ as follows:

$$\widehat{f}_{j,n}(x) = \frac{1}{h_{nj}^{p}} \sum_{i=1}^{n_j} K\left(x, X_i^{(j)}\right) \tag{14.31}$$

where $X^{(j)}{}_i$ is the ith pattern in the training sequence that belongs to class j. p is the dimension of the input vector and the patterns. Fig. 14.4 shows a basic illustration of what has been formulated so far. The network that Specht (1990) introduced has four layers:

1. Input layer
2. Pattern layer
3. Summation layer
4. Decision layer

The input layer is just a set of p junctions for getting the input vector and distributing the input to the next layer. The pattern layer is where the bulk of the calculations takes place. Each pattern in the training sequence has a dedicated pattern unit that applies a two-step process to the input vector before passing its result to the next layer. In the first step, the pattern unit compares its dedicated

pattern to the input vector. This is the distance measure that is explained for the potential functions. Define $D\left(x, X_i^{(j)}\right)$ as the distance measure of input vector x and the ith pattern in the training sequence belonging to class j:

- Dot product, defined as

$$D\left(x, X_i^{(j)}\right) = \sum_{k=1}^{p} x_k \cdot X_{i,k}^{(j)} \tag{14.32}$$

where x_k is the kth element of input vector, and $X_{i,k}^{(j)}$ is the kth element of the ith pattern in the training sequence that belongs to class j. Both vectors are normalized to unit length before the dot product (Specht, 1990).

- Euclidean distance, defined as

$$D\left(x, X_i^{(j)}\right) = \sqrt{\sum_{i=1}^{p} \left(x_i - X_i^{(j)}\right)^2} \tag{14.33}$$

- Manhattan (or city block) distance:

$$D\left(x, X_i^{(j)}\right) = \sum_{i=1}^{p} \left|x_i - X_i^{(j)}\right| \tag{14.34}$$

The second step is applying a nonlinear function (called kernel) to the distance between input and the training pattern. Define the kernel as $K(x,u) = K(D(x,u))$. The kernel function needs to satisfy the four conditions that were put for potential function:

1. The kernel has to have its maximum at $x = u$

2.

$$\lim_{D(x,u) \to \infty} K(x, u) = 0 \tag{14.35}$$

3. K (x,u) is continuous on $-\infty < x < \infty$

4. If K(x_1,u) = K(x_2,u), meaning D(x_1,u) = D(x_2,u), then x_1 and x_2 vectors have the same degree of similarity with respect to u.

The implication from the first and second conditions is that the appropriate kernel has to be chosen with respect to the choice of the distance measure. The implication of the forth condition is that the vectors with the same distance from the data pattern u will have the same result after the kernel is applied to them. These are some examples of acceptable kernels for the mentioned distance measures:

- For dot product:

$$K(x, u) = \exp\left[\frac{x.u - 1}{2\sigma^2}\right] \tag{14.36}$$

Note that both x and u are normalized before the dot product. Therefore, the maximum value of the dot product can be 1 that occurs at $x = u$.

- For Euclidean distance:

$$K(x,u) = \exp\left[\frac{-D(x,u)}{2\sigma^2}\right] \tag{14.37}$$

where $D(x,u)$ is the Euclidean distance between x and u. The Euclidean distance is always positive which means the maximum value for the kernel happens when $xx = u$ and $D(x,u) = 0$. This is also true for the Manhattan distance measure.

- For the Manhattan distance:

$$K(x,u) = \exp\left[\frac{-D(x,u)^2}{2\sigma^2}\right] \tag{14.38}$$

The pattern units will pass the result of their calculations to the summation units. Each class $j \in \{1,2, \ldots, M\}$ has a dedicated unit in the summation layer. The summing unit of class j calculates the average of the values coming from the pattern units which had a pattern that was associated with class j, as Fig. 14.4 shows.

Another feature that can be incorporated in the summation layer is determining the significance of a false decision for any given class j. Define l_j as the loss coefficient associated with the decision that a given input x belongs to class j while its actual class is different. This coefficient can be added to the discriminant function as follows:

$$\widehat{d}_j(x) = p_j l_j \widehat{f}_j(x) \tag{14.39}$$

The value of l_j cannot be deducted from the data and is subjectively set depending on the application and significance of the false positive decision for class j. In an application where there is no difference between the classes in that regard, l_j can be set to 1 for all $j \in \{1,2, \ldots, M\}$.

The role of the decision layer is to pick the largest $\widehat{d}_j(x)$ and declare j as the class of input vector x.

4. Improving memory performance

The PNNs have the advantage of not needing extensive training computation time that is associated with the networks that work with back-propagation training method. However, this advantage comes at the cost of requiring massive memory for operation. Each row of the data set needs an independent unit on the pattern layer to compare the similarity between the input vector and

its corresponding data set row. When the PNNs were formulated by Specht (1990), the size of the data sets was significantly smaller than today. Still, the memory requirement was taxing for the technology of the day. This fact still holds today despite the advancement of the hardware technology since early 1990s. We explore two families of approaches to reduce the size of the PNN problem without significant loss of performance.

4.1 Feature reduction using principal component analysis

The conventional PCA can be used to reduce the size of the data patterns and the inputs without losing much of the information embedded in the data set. In that regard, PCA projects a high dimensional data vector $X \in R^n$ into a lower dimension subspace $R^q \subset R^n$, knowing $n > q$. Here, a brief explanation about applying PCA to data patterns is included. The details of the method and why it works can be found in Jolliffe (2011).

Suppose $X \in R^{d \times n}$ is a matrix of data patterns where each row represents a pattern and each column represents an individual feature. The goal is to find a mapping $M \in R^{n \times q}: R^n \rightarrow R^q$ that can map the rows of matrix X to new rows of matrix $\widehat{X}$ where $\widehat{X} \in R^{d \times q}$ is the lower dimension approximation of matrix X. The matrix W is obtained in this fashion:

1. Center the data matrix by subtracting each element by the mean value of its corresponding column:

$$X_{ij\_new} = X_{ij\_old} - \mu_j \tag{14.40}$$

μ_j is given by:

$$\mu_j = \frac{1}{d} \sum_{i=1}^{d} X_{ij} \tag{14.41}$$

where d is the number of rows in matrix X, which is the number of data patterns.

2. Form the covariance matrix as $V = X^T X$
3. Find the eigenvalues of matrix V and their corresponding eigenvectors. Suppose $\lambda_1, \lambda_2, \ldots \lambda_n$ are eigenvalues of the covariance matrix V in descending order and $\upsilon_1, \upsilon_2, \ldots \upsilon_n$ are their corresponding eigenvectors (υ_i corresponding with λ_i and $\lambda_1 > \lambda_2 >, \ldots, > \lambda_n$).
4. Form the mapping matrix M, the columns of which are the first q eigenvectors of matrix V:

$$M = [\upsilon_1 | \upsilon_2 | \ldots | \upsilon_q]$$

Choosing q is a tradeoff between the level of dimension reduction and the preservation of the information. As q increases toward n, the information loss will reduce. However, smaller values of q will reduce the size of the data more.

5. Calculate the mapped data using matrix M:

$$\widehat{X} = XM \tag{14.42}$$

This approach can reduce the computation time by reducing the number of input units and the weight matrix. This reduction can reduce the calculation time that is spent in the input layer.

PCA has been used in Wu et al. (2007) to reduce the dimension of the input vector from 12 to 5 before using the input to classify leaf images into 32 categories of plants. Also Othman and Basri (2011) used PCA to extract features from a data set of medical images. Because the original inputs are vectors made from individual images of $M \times N$ pixels, each input vector has $M \times N$ elements. This is dramatically more than the 12 input features in Wu et al. (2007). Then PCA is used to reduce the dimension of the input vectors from $M \times N$ to $d \ll M \times N$. In Wang et al. (2013), the input vector is a waveform obtained by applying a window of 200 data points to ECG signals. The mentioned windows are centered on R peaks (100 points on each side of the R peak). This means the original input vector has 200 elements. The vector elements are then normalized using Z-score method to decrease superficial differences between the sample waveforms using this formula:

$$x_{z\_score} = \frac{x_{original} - \mu}{\sigma} \tag{14.43}$$

where μ is the mean value of $x_{original}$ and σ is the standard deviation. Then, PCA is used to reduce the features before the data were used to form the pattern layer.

4.2 Pattern layer size reduction using clustering

In a normal PNN, each data pattern is assigned to a pattern unit in the pattern layer that individually measures the similarity between the input vector and its assigned data pattern. One can imagine that in this architecture the pattern layer can get extremely (and possibly impractically) large as the size of the data gets bigger. We know from Parzen (1962) that larger data sets can increase the accuracy of the estimation of the probability density function of the data. So a larger data set should be welcome and not a concern for practicality. One prominent approach to reduce the number of pattern units without losing too much information is creating clusters from data patterns using k-means algorithm. This comes from the idea that not all the data patterns contain original, independent, and discriminating information.[5]

5. This is independent from PCA, which can be used in conjunction with k-means to reduce the computation burden of the PNN even more.

With this approach, each pattern unit has a group of one or more data patterns assigned to it. Each pattern group (known as cluster) is represented by its centroid. Here is a brief explanation of the k-means clustering algorithm for PNN:

1. Determine the number of the clusters, which will be the same as the number of the pattern units.
2. Assign a random centroid to each cluster. The centroid of the cluster is the mean value of all the data patterns in the cluster. Initially, it has a random value that will be adjusted by the data patterns that will be added.
3. Add a new data pattern to the cluster that can minimize this formula:

$$\frac{(A^j)^2}{(A^j+1)^2}\left\|x^k - x^j\right\|^2 \tag{14.44}$$

where A^j is the number of patterns already in the cluster, x^j is the coordinates of the cluster centroid, and x^k is the new data pattern vector that is being added to the cluster.

4. Update the cluster centroid using the coordinates of the newly added pattern that minimized Eq. (14.44):

$$x^j_{new} = \frac{A^j_{old}x^j_{old} + x^k}{A^j_{old}+1} \tag{14.45}$$

$$A^j_{new} = A^j_{old} + 1 \tag{14.46}$$

5. After all the data patterns are added to their clusters, use the cluster centroids instead of the data patterns to form the input weights in the pattern layer.

Note that the output of each pattern unit needs to be multiplied by the number of data patterns in its associated cluster (A^j) before it is added to the summation unit.

5. Simple probabilistic neural network example in Python

In this example, we have included three clusters (in red, yellow, and green) in two-dimensional coordinates (feature 1 and feature 2). Table 14.1 presents the data points along with the class label for each point. In addition to this, Fig. 14.5 presents the two-dimensional feature space for the simple PNN example. The data points are clustered into red, yellow, and green. The desired point to be clustered is shown as black star. In this example, we tried to reflect

TABLE 14.1 Simple probabilistic neural network example: Data.

Feature 1	Feature 2	Class label
0.1	0.9	1 (red)
0.5	0.9	1 (red)
0.2	0.7	1 (red)
0.6	0.6	2 (yellow)
0.8	0.8	2 (yellow)
0.4	0.5	2 (yellow)
0.8	0.5	3 (green)
0.6	0.3	3 (green)
0.3	0.2	3 (green)

the architecture of the PNN as simple as we can. The code is written in Python (Tahmassebi, 2018). It includes three different functions: (1) a function to create a simple dummy data in two-dimensional, (2) a function to calculate all the PNN stages including input layer, pattern layer, kernel, and summation, and (3) a function to visualize the results and data points. All these functions are called in the main function. In this example, we simply employed a two-dimensional Gaussian distribution as the kernel. However, any other kernels can be used. As shown, the desired point (shown in black star) is obviously in the green cluster and the PNN model as seen did correctly cluster this point as green.

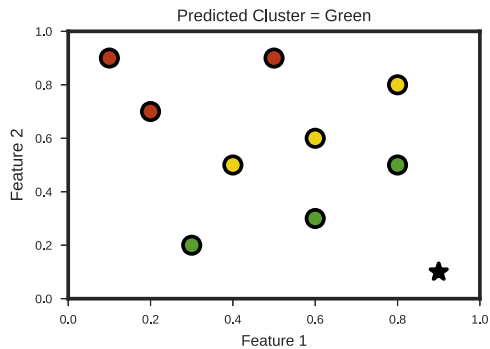

FIGURE 14.5 The two-dimensional feature space for probabilistic neural network. The data points are clustered into red, yellow, and green. The desired point to be clustered is shown as black star.

```
# Loading Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
sns.set_style("ticks")
mpl.rcParams['axes.linewidth'] = 3
mpl.rcParams['lines.linewidth'] =7

# Function to create the data
def Create_DataFrame():
    # defining the features and class labels as a dictionary
    data = {
            "Feature_1" : [0.1,0.5,0.2,0.6,0.8,0.4,0.8,0.6,0.3],
            "Feature_2" : [0.9,0.9,0.7,0.6,0.8,0.5,0.5,0.3,0.2],
            "Class_Label" : [1,1,1,2,2,2,3,3,3]
           }

    # converting the dictionary into a dataframe
    df = pd.DataFrame(data = data)

    return df

# Function to calculate PNN
def PNN(df, DesiredPoint):
    # defining a group for each class labels
    Clusters = df.groupby("Class_Label")
    # defining the number of classes as clusters
    NumClusters = len(Clusters)
    # an empty dictionary for calculating the sum of Gaussian for
    each class
    GaussianSums = dict()
    # defining the number of features
    NumFeatures = df.shape[1] - 1
    # defining the standard deviation for Gaussian distribution
    Sigma = 1.0
    # creating features array
    Features = df.drop(["Class_Label"], axis = 1).values

    # INPUT LAYER OF PNN
    # defining a row variable for moving over the data row by row
    _row = 0
    # loop over the number of clusters
    for i in range(1, NumClusters + 1):
        # initialize the GaussianSum for each class
        GaussianSums[i] = 0.0
        # defining the number of points per cluster
        PointsPerCluster = len(Clusters.get_group(i))

        # PATTERN LAYER OF PNN
        # defininning temporary sum for holding the sum of X and Y
    elements
        TempSum = 0.0
        # loop over points of each cluster and GaussianSum
    calculation
```

```
        for j in range(1, PointsPerCluster + 1):
            # calculating the X element of Gaussian
            TempX = ( DesiredPoint[0] - Features[_row][0]  )**2
            # calculating the y element of Gaussian
            TempY = ( DesiredPoint[1] - Features[_row][1]  )**2
            # calculating the Gaussian
            TempCoeff = -(TempX + TempY)/(2.0 * Sigma**2)
            # adding the calculated Gaussian for all the points per
     cluster
            TempSum +=  TempCoeff
            # incrementing the row to cover all points per cluster
            _row += 1
        # storing the GaussianSum per cluster in a dictionary
        GaussianSums[i] = TempSum

    # returning the key of the maximum GaussianSum per cluster
    CalculatedClass = max(GaussianSums, key = GaussianSums.get)

    # Visualization
    Visualization(df, Features, DesiredPoint, CalculatedClass)
    print("Calculated Class = " + str(CalculatedClass))

# Function to visualize the data
def Visualization(df, Features, DesiredPoint, CalculatedClass):

    color_dict = {1 : "Red", 2 : "Yellow", 3 : "Green"}
    plt.figure(figsize=(6,4))
    plt.scatter(Features[:,0],
                Features[:,1],
                s = 200.,
                c = df["Class_Label"],
                cmap=plt.cm.prism,
                marker = "o",
                lw = 3,
                edgecolor='k')
    plt.scatter(DesiredPoint[0],
                DesiredPoint[1],
                s = 200.,
                c = "k",
                marker = "*",
                lw = 3,
                edgecolor='k')
    plt.xlabel("Feature 1", fontsize = 20)
    plt.ylabel("Feature 2", fontsize = 20)
    plt.title("Predicted Cluster = " + color_dict[CalculatedClass],
     fontsize = 20)
    plt.xlim([0,1])
    plt.ylim([0,1])
    plt.show()

# Main Function
def main():
    # desired point for clustering
    DesiredPoint = [0.4, 0.7]
    df = Create_DataFrame()
    PNN(df, DesiredPoint)
```

6. Conclusions

This chapter has been an introduction to PNNs and their numerous applications in science and engineering from a practical point of view. These applications range from simple pattern recognition to complex waveform classification. The PNNs are most effective when used alongside methods of feature extraction and feature reduction, the latter of which can reduce the volume of required calculation and memory as well. Because the pattern units of the same class operate independently from the ones of another class in the pattern layer, the PNN can be considered a great use case for parallel computing. By parallel computing gaining traction during the last decade, the PNNs can emerge again as an attractive alternative to the feedforward backpropagation networks in applications that have massive amounts of data available for training.

References

Adeli, H., Panakkat, A., 2009. A probabilistic neural network for earthquake magnitude prediction. Neural Networks 22, 1018−1024.

Andrews, H., 1972. Mathematical Techniques in Pattern Recognition. J. Wiley Verlag.

Asencio-Cortés, G., Martínez-Álvarez, F., Troncoso, A., Morales-Esteban, A., 2017. Medium-large earthquake magnitude prediction in Tokyo with artificial neural networks. Neural Computing and Applications 28, 1043−1055.

Bankert, R.L., 1994. Cloud classification of avhrr imagery in maritime regions using a probabilistic neural network. Journal of Applied Meteorology 33, 909−918.

Batchelor, B., 1974. Practical Approach to Pattern Classification. Plenum Press Verlag.

Hazrati, M.K., Erfanian, A., 2010. An online eeg-based brain-computer interface for controlling hand grasp using an adaptive probabilistic neural network. Medical Engineering and Physics 32, 730−739.

Jolliffe, I., 2011. Principal component analysis. In: International Encyclopedia, of Statistical Science. Springer, pp. 1094−1096.

Meisel, W., 1972. Computer-Oriented Approaches to Pattern Recognition. Academic Press.

Meyer-Baese, A., Meyer-Baese, A., 2004. Pattern Recognition for Medical Imaging. Academic Press.

Mishra, S., Bhende, C., Panigrahi, B., 2008. Detection and classification of power quality disturbances using s-transform and probabilistic neural network. IEEE Transactions on Power Delivery 23, 280−287.

Mood, A.M., 1950. Introduction to the Theory of Statistics.

Nishanth, K.J., Ravi, V., 2016. Probabilistic neural network based categorical data imputation. Neurocomputing 218, 17−25.

Othman, M.F., Basri, M.A.M., 2011. Probabilistic neural network for brain tumor classification. In: Intelligent Systems, Modelling and Simulation (ISMS), 2011 Second International Conference on. IEEE, pp. 136−138.

Parzen, E., 1962. On estimation of a probability density function and mode. The Annals of Mathematical Statistics 33, 1065−1076.

Poggio, T., Girosi, F., 1990. Networks and the best approximation property. Biological Cybernetics 63, 169−176.

Rutkowski, L., 2004. Adaptive probabilistic neural networks for pattern classification in time-varying environment. IEEE Transactions on Neural Networks 15, 811−827.

Song, T., Jamshidi, M.M., Lee, R.R., Huang, M., 2007. A modified probabilistic neural network for partial volume segmentation in brain mr image. IEEE Transactions on Neural Networks 18, 1424−1432.

Specht, D.F., 1990. Probabilistic neural networks. Neural Networks 3, 109−118.

Tahmassebi, A., 2018. ideeple: deep learning in a flash. In: Disruptive Technologies in Information Sciences, 10652. International Society for Optics and Photonics, p. 106520S.

Tripathy, M., Maheshwari, R.P., Verma, H., 2010. Power transformer differential protection based on optimal probabilistic neural network. IEEE Transactions on Power Delivery 25, 102−112.

Wang, J.-S., Chiang, W.-C., Hsu, Y.-L., Yang, Y.-T.C., 2013. Ecg arrhythmia classification using a probabilistic neural network with a feature reduction method. Neurocomputing 116, 38−45.

Wu, S.G., Bao, F.S., Xu, E.Y., Wang, Y.-X., Chang, Y.-F., Xiang, Q.-L., 2007. A leaf recognition algorithm for plant classification using probabilistic neural network. In: Signal Processing and Information Technology, 2007 IEEE International Symposium on. IEEE, pp. 11−16.