Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2014) 22: 979 – 989
© TÜBİTAK
doi:10.3906/elk-1209-35

A reduced probabilistic neural network for the classification of large databases

Abdelhadi LOTFI[*], **Abdelkader BENYETTOU**
Signal Image Parole Laboratory, Department of Computer Science, University of Sciences and Technologies of Oran "Mohamed Boudiaf", Oran, Algeria

Received: 07.09.2012 • **Accepted:** 10.01.2013 • **Published Online:** 17.06.2014 • **Printed:** 16.07.2014

Abstract: The probabilistic neural network (PNN) is a special type of radial basis neural network used mainly for classification problems. Due to the size of the network after training, this type of network is usually used for problems with a small-sized training dataset. In this paper, a new training algorithm is presented for use with large training databases. Application to the handwritten digit database shows that the reduced PNN performs better than the standard PNN for all of the studied cases with a big gain in size and processing speed. This new type of neural network can be used easily for problems with large training databases like biometrics and data mining applications. An extension of the network is possible for new training samples and/or classes without retraining.

Key words: Classification, pattern recognition, reduced probabilistic neural network, handwritten digit recognition, optimization

1. Introduction

The world of pattern recognition is changing so fast that old classification techniques need to be improved regularly in order to suit the new training databases that are always growing in size. State-of-the-art techniques usually prove efficient for small-sized benchmarking problems, but when it comes to applying these methods to real-world problems, the size of the training set becomes a handicap for certain problems. In fact, in many situations, a bigger size implies a serious decrease in performance.

The probabilistic neural network (PNN) is mainly used for classification problems. It has been used extensively in recent years to solve different problems related to pattern recognition. Examples of areas where the PNN was used successfully are email security enhancement [1], intrusion detection within computer networks [2], water quality assessment [3], health and disease diagnosis [4,5], detection of resistivity for antibiotics [6], biometrics applications [7–9], and many military applications [10,11].

Although the PNN is very popular in classification, it suffers from a major drawback. The size of the network after training is equal to the size of the training dataset, which makes it impractical for use with large-scale big databases. In fact, the problem is doubled since the performance of the system usually decreases in terms of the classification accuracy and speed with a very big hidden layer.

Much work has been done to overcome certain drawbacks of the standard PNN for some specific situations. In [12], a learning vector quantization algorithm was used to train a PNN in order to make the network's size smaller. Other examples of PNN variants are the fuzzy PNN [13] and the particle swarm optimization stochastic algorithm for PNN parameter choices [14]. All of these approaches solve one part of the problem and make

[*]Correspondence: alotfi@ito.dz

another problem. In fact, by modifying the training algorithm in certain ways, the network loses some of its major advantages. For example, the rotated kernel PNN [15] has good generalization qualities, which are claimed to compare to those of support vector machines (with a reduced number of classes), but the training is very slow, even for small datasets, and the network does not support adding new classes and/or samples unless the training is restarted.

In this work, a new training algorithm is proposed for reducing the size of the PNN and increasing performance during classification as well, without modifying the general architecture of the network.

2. PNNs

2.1. Mathematical background

Like radial basis function (RBF) neural networks, PNNs use RBFs as activation functions in the second layer (hidden layer) to make a local decision function centered on a subset (a sample) of the input space [16]. After training, the global decision function is constructed by summing all of the local functions [17,18]. This is why the PNN has an advantage over the multilayer network, which tries to find a global decision function after training. The decision of a PNN is not affected by local minima if they exist.

For any classification problem, each observed vector of cluster data x (d-dimensional vector) is placed in one of the predefined labeled classes $C_i = 1, 2, \ldots, m$, where m is the number of possible classes to which x can belong. The dimension of the input vector $x$and the number of possible classes m have great impact on the performance of the network, since a high number of input parameters affect the speed and accuracy of the network. The Bayes pattern classifier implements the Bayes conditional probability rule, where the probability $P(C_i|x)$ of x belongs to class C_i. This probability is given by:

$$P(C_i|x) = \frac{P(x|C_i)P(C_i)}{\sum\limits_{j=1}^{m} P(x|C_j)P(C_j)}, \tag{1}$$

where $P(C_i|x)$ is the conditional probability density function (PDF) of x given set C_i, and $P(C_j)$ is the probability of drawing data from class C_j.

The input vector x (candidate for classification) is classified as belonging to class C_i if:

$$P(C_i|x) > P(C_j|x); \quad \forall j = 1, 2, , m; j \neq i. \tag{2}$$

The main problem in Eq. (2) is that the prior probabilities $P(x|C_i)$ (probability that sample x comes from population C_i) are most of the time unknown for the system. To solve this problem, we can estimate these probabilities from the training samples. The Parzen windowing technique for PDF estimation [17] may present a good solution. Parzen proved that a class of PDF estimators asymptotically approaches the underlying parent density, provided that it is smooth and continuous [19]. The Parzen estimator used here is:

$$f_A(x) = \frac{1}{2\pi^{d/2}\sigma^d}\frac{1}{m}\sum_{i=1}^{m}\exp\left[-\frac{(X - X_{ai})^t(X - X_{ai})}{2\sigma^2}\right], \tag{3}$$

where X_{ai} is the ith pattern from class C_A, d is the dimension of the space, and σ is a smoothing parameter.

From Eq. (3), we can see that the global decision function $f_A(x)$ is the sum of small Gaussian functions centered on the training samples. This is similar to a multidimensional interpolation. PNNs use all of the training dataset to estimate PDFs $P(x|C_j)$. These functions are then used to estimate the likelihood of the input sample as belonging to a given class [19].

2.2. Architecture

A typical architecture of a standard PNN is given in Figure 1.

A PNN has 4 layers [19], as given below.

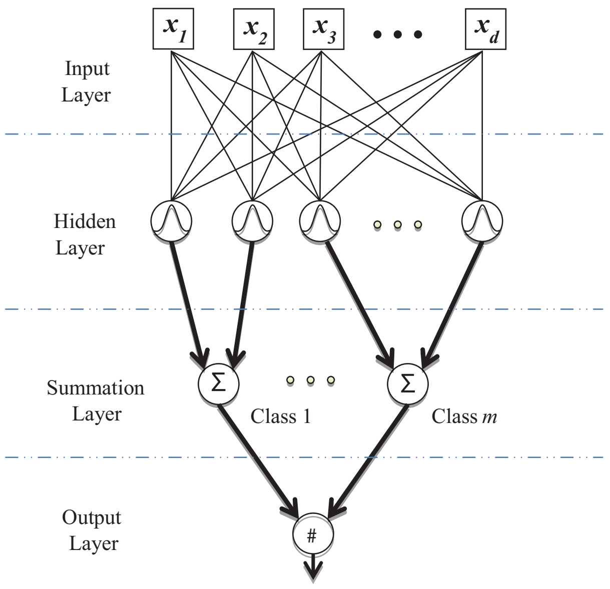

Figure 1. General architecture of a PNN.

Input layer: The number of neurons in this layer is equal to the number of parameters (variables) needed to describe the separable form input.

Pattern (hidden) layer: This layer organizes the learning set by representing each input vector by a hidden neuron that records the parameters of this vector. The activation function used in this layer is the following exponential function:

$$\exp\left(-\frac{\left(w_j^i - x\right)^t \left(w_j^i - x\right)}{2\sigma^2}\right). \tag{4}$$

The computation of the output of the hidden units is realized by the formula:

$$\emptyset_j^i(x) = \frac{1}{(2\pi)^{d/2}\sigma^d} \exp\left(-\frac{1}{2\sigma^2}\left(x - x_j^i\right)^t \left(x - x_j^i\right)\right), \tag{5}$$

where x_j^i is the jth training pattern vector from patterns in class C_i and d is the dimension of the input space.

Output layer: This is also called the summation layer. In this layer, the number of units equals the number of the existing classes. Each unit is connected to all of the neurons in the pattern layer that represent samples that belong to this class. The output of each neuron in this layer represents the probability for the input vector X to belong to class C_i. Thus:

$$P(C_i|x) = \frac{1}{(2\pi)^{d/2}\sigma^d} \frac{1}{N_i} \sum_{j=1}^{N_i} \exp\left(-\frac{\left(x - x_j^i\right)^t \left(x - x_j^i\right)}{2\sigma^2}\right). \tag{6}$$

Note that N_i is the number of training patterns from class C_i.

Sometimes another layer is added to normalize the input vectors if they are not already normalized [3]. In fact, as is the case for most multilayer networks, the input vector must be normalized to give adequate separation in the hidden layer.

The classification decision for the input vector X is given in accordance with the Bayes decision rule using:

$$\overline{C}\,(x) \;=\; \underset{i=1,\,,\,m}{\arg\max}\left\{\frac{1}{(2\pi)^{d/2}\,\sigma^{d}}\frac{1}{N_i}\sum_{j=1}^{N_i}\exp\left(-\;\frac{\left(x-x_j^i\right)^t\left(x-x_j^i\right)}{2\,\sigma^2}\right)\right\}, \tag{7}$$

where m is the number of classes presented in the training set [20].

During the training process, the radial hidden units are copied directly from the training set, one for each training sample. Each hidden node models a Gaussian function centered on a training sample. In the summation layer, there is an output unit for each class, which is connected to all of the hidden units belonging to the same class, without connections to other neurons [4]. The outputs are proportional to the density functions of different classes and normalized to form a sum equal to 1. This can be seen as a probabilistic output.

The pattern units (neurons in the hidden layer) use the following Gaussian activation function:

$$f(x) \;=\; e^{-(w_i\,-\,x_i)^t(w_i\,-\,x_i)/2\,\sigma^2}, \tag{8}$$

where w_i are the weights, x_i are the variables of the model, and σ is a smoothing parameter.

2.3. Training algorithm

The training of a PNN consists of simply copying the training samples to the hidden units in the second layer of the network. This is done in one pass, which makes a PNN very fast to train. The number of hidden neurons is equal to the number of samples. The algorithm used for training is then:

```
Begin
  Initialization
        j = 0;
        n = number of samples;
  Do
     j ← j + 1;
     Normalization: $x_{jk} \leftarrow \frac{x_{jk}}{\sqrt{\sum_i^d x_{ji}^2}}$;
     Learning process: $w_{jk} \leftarrow x_{jk}$;
     If $x \in w_i$, then $a_{ic} \leftarrow 1$;
  Until $j = n$;
End
```

3. Modified training algorithm

To get rid of the redundancies in the training sets, a new training algorithm is proposed. The algorithm should not affect the architecture of the original PNN and the resulting network has to share the same advantages with this latter one, with a reduced number of hidden units.

In order to exclude neurons that are quite similar in their response (share the same activation area), a standard training of the network is realized with the standard PNN training algorithm presented in Section 2.2. Note here that, with this algorithm, we can add or remove hidden units (even for new classes) easily by adding hidden neurons and their respective connections. After the creation of this network, another pass is necessary to delete neurons of low importance (that have the same activation area as some others) in their classification. This is done by deleting the neuron and observing the activation area when classifying the same sample that represents this unit. In other words, we delete the hidden neuron and measure its importance for classification of the same sample (each hidden neuron represents one training sample). If the new network can successfully classify this sample, the new network is retained. In the other case (wrong classification or right classification with low confidence), the network from step *(k-1)* is restored. The process is repeated for all of the samples in the training set. To make sure the network can classify all of the samples in the training set, another pass is necessary to add samples that are misclassified to the hidden layer of the PNN (see Figure 2). This last pass is done only to keep the same advantages as the standard PNN. Another advantage of this new reduced PNN (RPNN) network is that all of the boosting algorithms used for a standard PNN are applicable without major modifications. As a result, the proposed training algorithm is much slower than the standard one.

The new algorithm is given below.

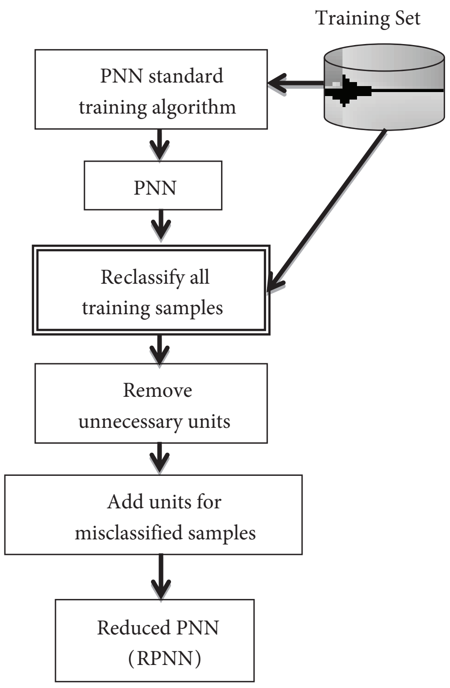
Training Set
PNN standard training algorithm
PNN
Reclassify all training samples
Remove unnecessary units
Add units for misclassified samples
Reduced PNN (RPNN)

Figure 2. The new training algorithm.

3.1. First step: construction of a standard PNN

Use the algorithm presented in Section 2.3 to construct a PNN. The hidden layer has as many neurons as there are samples. The insertion or suppression of a hidden neuron consists of adding or removing a row in the matrix of weights. Thus, it is relatively a fast operation.

3.2. Second step: reduce the network size

In this step, a smaller equivalent network is generated.

```
Begin
    Initialization
        k = 0;
        x = first training sample;
    Do
        k ← k + 1;
        Delete the $k$th node from the hidden layer
        Classify the $k$th sample in the new network (PNN*)
        If the classification generates a good probability, then adopt the new network
        Else return to the original PNN;
    Until $k = n$;
    Return PNN = PNN*;
End
```

3.3. Third step: verifying the classification

This step is necessary if we want a 100% classification rate for the training set.

```
Begin
    Initialization
        k = 0;
        x = first training sample;
    Do
        k ← k + 1;
        If classification ($k$th sample) $\neq$ target, then
        Insert sample into hidden layer;
    Until $k = n$;
    Return PNN*;
End
```

3.4. Testing algorithm

The testing algorithm for the RPNN is the same as a standard PNN testing algorithm. After a training process, the probabilistic network can be used to classify input samples. Each neuron in the hidden layer calculates the product $z_k = w_k^t x$ and will output a nonlinear function of z_k, $\varphi(z_k) = e^{\frac{z_k - 1}{\sigma^2}}$, where σ is a smoothing parameter defined by the user.

Each output neuron sums the contributions of all of the hidden neurons connected to the neuron representing a given class. The testing algorithm is given as follows:

```
Begin
    Initialization
        k = 0;
        x = test sample;
```

```
  **Do**
    k ← k + 1; $z_k \leftarrow w_k^t x$
    **If** $a_{kc} = 1$, **then** $g_c \leftarrow g_c + e^{\frac{z_k - 1}{\sigma^2}}$;
  **Until** $k = n$;
  **Return class** $\leftarrow \arg\max_i (g_i(x))$;
**End**
```

4. Database descriptions

Since the PNN is usually applied on small benchmarking datasets, handwritten digit recognition represents a challenge for a standard PNN. There are only a few applications of this type of network for real-world big databases [20]. Both the number of neurons in the input layer (dimensionality) and the number of samples (available labeled data) can dramatically affect a neural network of any type [21].

In order to test the RPNN on a real-world big database, the new classifier was tested on the Mixed National Institute of Standards and Technology (MNIST) database for handwritten digit recognition. This database is well known in pattern recognition. It contains a large number of samples (60,000 samples for training and 10,000 samples for testing). The samples are 20 $\times$ 20-pixel black and white images of handwritten digits (see Figure 3). The different Arabic digits were written by 500 different writers and size-normalized. This is a good database for people who want to try pattern recognition methods on real-world data while spending minimal effort on the preprocessing and formatting of data.

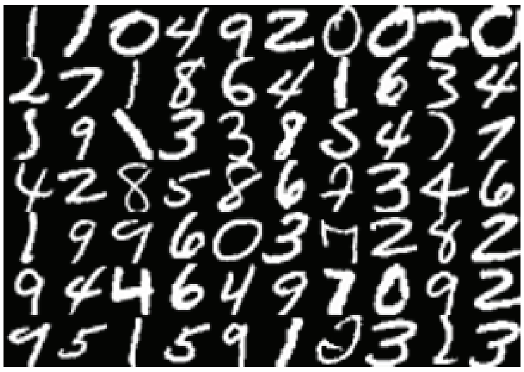

Figure 3. Some samples from the MNIST database.

Table 1. Number of samples for each class for 60,000 training samples and 10,000 testing samples.

Class	"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"
Training	5922	6739	5957	6129	5840	5419	5918	6265	5851	5948
Testing	978	1133	1031	1010	980	891	957	1027	974	1007

Table 1 shows some statistics about the training and testing database. The number of samples in each class is almost the same, which means that the database is more or less balanced. Digits 1 and 7 have more training samples than the others because they have almost the same form and can lead to misclassifications. The average is about 6000 images for training and 1000 images for testing each digit.

We have also used the 'skin segmentation dataset' to confirm our results on an even bigger database. This database is used to classify skin and nonskin images for the FERET and PAL Face databases. It contains 245,057 labeled samples, where 50,859 are face samples and 194,198 are not face samples. The training set contains 183,750 examples and the remaining examples are used for testing.

5. Experimental results

Both the RPNN and the standard PNN are tested with different subsets of the training and testing samples to observe the performance of the network as the number of training examples becomes bigger. The only parameter that is fixed is the smoothing parameter, fixed to 0.9 after many tests with a small set of training samples. Table 2 shows the experiments carried out in this paper. All of the samples in the test set are used for all of the experiments. For each experiment, the number of training samples is changed to see its impact on the performance of both the standard PNN and RPNN.

The performances of the standard PNN and RPNN are given in Tables 2 and 3, respectively. Table 4 represents the number of hidden units for the RPNN for all of the experiments.

Table 2. Performance of the standard PNN for the classification of 10,000 test examples.

Number of training samples	% Correct	% Incorrect	Time (h:min:s)
1000	85.00	15.00	00:31:50
10,000	82.00	18.00	05:30:50
20,000	80.30	19.70	11:50:13
30,000	78.70	21.30	18:48:03
40,000	93.90	6.10	25:30:16
50,000	78.65	21.35	33:17:51
60,000	80.30	19.70	40:59:40

Table 3. Performance of the RPNN for the classification of 10,000 test examples.

Number of training samples	% Correct	% Incorrect	Time (h:min:s) (h:min:s)
1000	85.69	14.30	00:08:35
10,000	94.71	5.29	00:49:52
20,000	96.19	3.81	01:03:57
30,000	96.67	3.33	01:10:49
40,000	96.89	3.11	01:27:39
50,000	97.12	2.88	01:37:09
60,000	96.94	3.06	01:52:27

Figure 4 represents the reduction ratio of the hidden units in the RPNN by the number of hidden units in a standard PNN (in %).

Results for the second experiment on the face segmentation dataset are given in Table 5.

Table 4. Number of neurons in the hidden layer.

Standard PNN	RPNN
1000	251
10,000	1644
20,000	2106
30,000	2491
40,000	2812
50,000	3184
60,000	3464

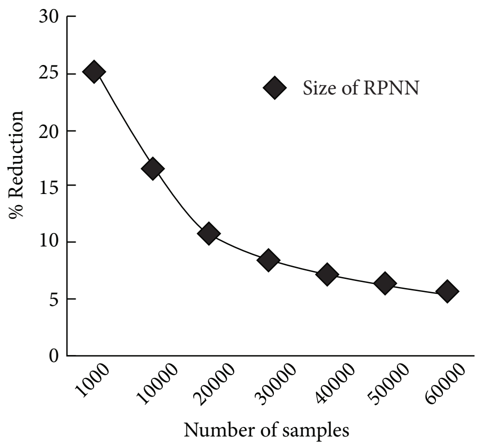

Figure 4. Size reduction ratio.

Table 5. Results for the skin segmentation database.

	Standard PNN	RPNN
No. of hidden neurons	183,750	406
% of correct classifications	100	99.9
Time to classify 1 example (s)	1200	0.7

6. Discussion

From Tables 2 and 3, we observe that the RPNN outperforms the standard PNN, both in classification accuracy and processing speed. The RPNN classifies the testing samples (unseen before) well and hence has a good generalization for the testing set. At the same time, the processing speed is greatly improved with the RPNN.

From Table 4, we notice that the number of neurons in the hidden layer is very small in the RPNN when compared to the hidden units of a standard PNN, which are equal to the number of samples. In fact, this is what justifies the improved processing speed in the RPNN over the other network.

The results in Table 5 show that the PNN and RPNN give quite similar classification performances. This is because the database contains a lot of redundant samples. However, the size of the RPNN (406 neurons) is far smaller than that of a standard PNN (183,750 neurons), which will simply copy all of the training set in the hidden layer. As a result, the time for testing is very important for the PNN. This makes the standard training algorithm totally inefficient in this case.

As the number of training samples becomes bigger, the number of required neurons in the hidden layer of an RPNN goes down. This is because the RPNN avoids over-fitting by using only a small set of the training samples to represent each class. The rest of the training set is considered as noisy redundant samples. The classification rate for the training set is 100%, which is similar to the standard PNN.

The drawback with the RPNN is that it needs more time to train since training is an evaluation of the contribution of each neuron in the hidden layer. Training a standard PNN is simply copying the training samples in the hidden layer, and this is done in one pass. This is why the standard PNN is only used for small databases.

7. Conclusion

In this paper, a new training algorithm is proposed for PNNs. The proposed algorithm reduces the size of a PNN while increasing accuracy and processing speed.

Experimental results show that the RPNN outperforms the standard PNN in terms of classification accuracy and processing speed (for the test step) for all of the cases studied here. The new PNN has the same advantages as a standard PNN. The architecture of the network is not changed and new classes and/or training samples can be added without a retraining of the network. However, the training of an RPNN is very slow compared to the training of a standard PNN.

This algorithm can be used for all cases where a standard PNN suffers from the size of the training dataset and it is preferable for real-world large databases, where the gain in processing speed becomes very important.

References

[1] T.P. Tran, T.T.S. Nguyen, P. Tsai, X. Kong, "BSPNN: boosted subspace probabilistic neural network for email security", Artificial Intelligence Review, Vol. 35, pp. 369–382, 2011.

[2] T.P. Tran, L. Cao, D. Tran, C.D. Nguyen, "Novel intrusion detection using probabilistic neural network and adaptive boosting", International Journal of Computer Science and Information Security, Vol. 6, pp. 083–091, 2009.

[3] M.R. Nikoo, R. Kerachian, S. Malakpour-Estalaki, S.N. Bashi-Azghadi, M.M. Azimi-Ghadikolaee, "A probabilistic water quality index for river water quality assessment: a case study", Environmental Monitoring and Assessment, Vol. 181, pp. 465–478, 2010.

[4] M.S. Bascil, H. Oztekin, "A study on hepatitis disease diagnosis using probabilistic neural network", Journal of Medical Systems, Vol. 36, pp. 1603–1606, 2010.

[5] M. Hariharan, M.P. Paulraj, S. Yaacob, "Time-domain features and probabilistic neural network for the detection of vocal fold pathology", Malaysian Journal of Computer Science, Vol. 23, pp. 60–67, 2010.

[6] F. Budak, E.D. Übeyli, "Detection of resistivity for antibiotics by probabilistic neural networks", Journal of Medical Systems, Vol. 35, pp. 87–91, 2009.

[7] M.I. Faraj, J. Bigun, "Synergy of lip-motion and acoustic features in biometric speech and speaker recognition", IEEE Transactions on Computers, Vol. 56, pp. 1169–1175, 2007.

[8] S. Meshoul, M. Batouche, "A novel approach for online signature verification using fisher based probabilistic neural network", IEEE Symposium on Computers and Communications, pp. 314–319, 2010.

[9] K.T. Blackwell, T.P. Vogl, H.P. Dettmar, M.A. Brown, G.S. Barbour, D.L. Alkon, "Identification of faces obscured by noise: comparison of an artificial neural network with human observers", Journal of Experimental & Theoretical Artificial Intelligence, Vol. 9, pp. 491–508, 1997.

[10] L.F. Araghi, H. Khaloozade, M.R. Arvan, "Ship identification using probabilistic neural networks (PNN)", International MultiConference of Engineers and Computer Scientists, Vol. 2, 2009.

[11] M.W. Kim, M. Arozullah, "Generalized probabilistic neural network based classifiers", International Joint Conference on Neural Networks, 2002, Vol. 3, pp. 648–653, 1992.

[12] P. Burrascano, "Learning vector quantization for the probabilistic neural network", IEEE Transactions on Neural Networks, Vol. 2, pp. 458–461, 1991.

[13] V. Georgiou, P. Alevizos, M. Vrahatis, "Fuzzy evolutionary probabilistic neural networks", Artificial Neural Networks in Pattern Recognition, Vol. 5064, pp. 113–124, 2008.

[14] I. De Falco, A. Della Cioppa, E. Tarantino, "Facing classification problems with particle swarm optimization", Applied Soft Computing, Vol. 7, pp. 652–658, 2007.

[15] I. Galleske, J. Castellanos, "A rotated kernel probabilistic neural network (RKPNN) for multi-class classification", Proceedings of the Artificial and Natural Neural Networks 7th International Conference on Computational Methods in Neural Modeling, Vol. 1, pp. 152–157, 2003.

[16] T.P. Hong, S.S. Tseng, "A probabilistic perceptron learning algorithm", Journal of Experimental & Theoretical Artificial Intelligence, Vol. 4, pp. 265–279, 1992.

[17] R.O. Duda, Pattern Classification 2nd Edition with Computer Manual 2nd Edition Set. New York, Wiley-Interscience, 2004.

[18] D.K. Kim, D.H. Kim, S.K. Chang, "Modified probabilistic neural network considering heterogeneous probabilistic density functions in the design of breakwater", Korean Society of Civil Engineers Journal of Civil Engineering, Vol. 11, pp. 65–71, 2007.

[19] D.F. Specht, "Probabilistic neural networks", Neural Networks, Vol. 3, pp. 109–118, 1990.

[20] N. Neggaz, M. Besnassi, A. Benyettou, "Application of improved AAM and probabilistic neural network to facial expression recognition", Journal of Applied Sciences, Vol. 10, pp. 1572–1579, 2010.

[21] J. Li, K. Ouazzane, H. Kazemian, Y. Jing, R. Boyd, "A neural network based solution for automatic typing errors correction", Neural Computer & Applications, Vol. 20, pp. 889–896, 2011.