Neural Comput & Applic (1998)7:37-51

Implementing probabilistic neural networks

Fabio Ancona*, Anna Maria Colla**, Stefano Rovetta*, and Rodolfo Zunino*

* Department of Biophysical and Electronics Engineering, University of Genova – Via all'Opera Pia 11a 16145 Genova, Italy
*Elsag Bailey S.p.A.

Correspondence and offprint requests to: Stefano Rovetta, Department of Biophysical and Electronics Engineering, University of Genova – Via all'Opera Pia 11a 16145 Genova, Italy,

A modified PNN training algorithm is proposed. The standard PNN, though requiring a very short training time, when implemented in hardware exhibits the drawbacks of being costly in terms of classification time and of requiring an unlimited number of units. The proposed modification overcomes the latter drawback by introducing an elimination criterion to avoid the storage of unnecessary patterns. The distortion in the density estimation introduced by this criterion is compensated for by a cross-validation procedure to adapt the network parameters. The present paper deals with a specific real-world application, i.e., handwritten character classification. The proposed algorithm makes it possible to realize the PNN in hardware, and, at the same time, compensates for some inadequacies arising from the theoretical basis of the PNN, which does not perform well with small training sets.

Keywords: Digital neural processor; Generalization; Hardware implementation; Probabilistic Neural Networks; Random optimization

1 Introduction

The Probabilistic Neural Network (PNN) model, described by D.F. Specht in [1], is a neural implementation of the Parzen windows [2] [3] probability density approximation method, mainly (but not exclusively) oriented toward classification problems. It was originally devised to provide a neural tool capable of very fast training on real-world problems; as compared with the backpropagation, for a given level of performance, the speedup reported was about 200 000:1. The simplicity of the structure and its theoretical foundations are further advantages of the PNN model.

However, when designing a hardware implementation, the user is faced with severe drawbacks, mainly related to the constructive criterion used for training. For instance, in practical applications, a property needed for the training procedure is often re-trainability. In other words, if the performance of the trained network is not satisfactory on new patterns, the procedure should be able to learn these patterns without need for restarting from scratch. The basic algorithm, though theoretically very well-suited to this purpose, in practice poses a limit to this ability, in that the network is allowed to grow indefinitely.

The problem of implementing a PNN has been approached in different ways. Among others, in [4] the author of the model described an alternative architecture combining the Adaline [5] with the PNN. In [6] P. Burrascano proposed the application of the Learning Vector Quantization [7] algorithm to the PNN, by fixing the number of pattern units, which play the role of prototypes for vector quantization.

In the present paper, we describe a modification to the standard algorithm. It has been devised for the implementation of PNN on a commercially available digital neural chip, the Nestor/Intel Ni1000 Recognition Accelerator, designed for the realization of pattern recognition methods based on localized functions (PNN, RCE, PRCE) [8]. The implementation is oriented toward a classification application requiring a large number of training patterns, namely, recognition of handwritten digits. Some modifications to the basic procedure have been designed and tested. The new formulation of the training algorithm is not limited to the case under examination, but can be extended to the general case of an upper bound to the availability of resources.

2 The PNN model

2.1 Parzen's estimate and the Bayesian decision criterion

The *Parzen windows* method is a nonparametric identification procedure that synthesizes an estimate of a probability density function by superposition of a number of windows, replicas of a function $g()$ called the *kernel* (often a unimodal symmetric function):

$$f(x) \cong f_n(x) = \frac{1}{n\lambda} \sum_{i=1}^{n} g\left(\frac{x - x(l)}{\lambda}\right) \qquad (1)$$

where x is the dummy argument for a point in the sample space, the patterns $x(l)$ form the training set, and λ is a function of n such that

$$\lim_{n\to\infty} \lambda = 0 \quad \text{and} \quad \lim_{n\to\infty} n\lambda = \infty$$

E. Parzen proved that this estimate is consistent in quadratic mean:

$$\lim_{n\to\infty} E|f(x) - f_n(x)|^2 = 0 \qquad (2)$$

where the density $f(x)$ is continuous. Consistency, as defined here, implies that this simple approximation method is also "well-behaved," in the sense that, when the number of training examples grows, the estimate tends smoothly to the correct density. Finally, there are results that enable one to apply this technique to multivariate densities [9] when unimodal kernel functions are used.

The Bayesian decision criterion is the optimal classification procedure if the probability density functions of the classes to be discriminated (prior densities) are known. Therefore, by using Parzen's method to approximate separately the conditional class probability densities, the Bayesian procedure can be applied:

$$c = \text{argmax}_k \left\{\gamma^{(k)} f^{(k)}(\mathbf{x})\right\} \qquad (3)$$

where c is the class label output by the classifier, $\mathbf{x}$ is a vector in the input space, $f^{(k)}(\mathbf{x})$ is the density of the k-th class, and the coefficient $\gamma^{(k)}$ contains the prior probability and the risk coefficients, if any, associated with that class.

Parzen's method is an attractive estimation procedure, as it is a fast and straightforward way of learning probabilities from a training set. The trainability by examples and the nonparametric structure lead naturally to a neural implementation of the method.

2.2 The neural implementation

The structure of a PNN directly reflects the Bayes criterion applied to the Parzen estimation method. The described structure is sketched in Figure 1. The first layer (we do not count the input fan-out units) is that of the *pattern units*. There is one unit for each pattern, implementing a window, and the window size is a free parameter (the "smoothing parameter"). The l-th pattern unit computes its output in the following way:

$$Z_l = \mathbf{x}(l) \cdot \mathbf{w} \tag{4}$$

$$y_l = g(Z_l) \tag{5}$$

where Z_l is the "net input" of the l-th pattern unit, y_l is its output, $\mathbf{x}(l)$ is the l-th input pattern vector, $\mathbf{w}$ is the vector of the input weights of the unit l, and the activation function $g()$ is the kernel function that is used to implement the window centred on the l-th training pattern. Hence the first layer implements the single windows.

The subsequent layer (of *summation units*) computes the sum of the windows for each class to obtain the required density estimate. Finally, the output units perform a comparison between the estimated probabilities at the point given as the input pattern; the unit corresponding to the maximum class probability provides the classification result. Therefore the output layer implements the decision criterion by means of suitable coefficients γ that weight its input connection. As in the classical formulation, these coefficients can include risk weights, if they are available.

2.3 The training algorithm

The PNN is trained by a very fast procedure that requires only localized parameter adaptations. This is in contrast with other classical training algorithms, like backpropagation, in which all parameters have to be modified at each learning step. The procedure consists in adding a pattern unit for each new training pattern, adapting its input weights so that they coincide with the pattern itself. The subsequent activation function is designed to implement the desired kernel function, centred on the selected pattern.

The summation layer has no adaptive parameters. The weights γ in the output layer are given by the relative frequencies of patterns in the training set, so that their values need not be computed from the patterns but only from their number and proportions. Therefore, the training is not iterative but one-shot, and only a single pass through the training set is required.

3 Implementation issues

3.1 Some disadvantages of the PNN training algorithm

It is apparent that, in the presence of a large number of training examples, the number of pattern units may become prohibitive. In the case of retraining, the situation becomes still worse: the number of units needed is unbounded. This is an obvious problem to be faced when designing a hardware implementation of the algorithm. It is not easy to dribble the problem by reducing the number of training patterns. In practical applications, a large number of examples is necessary either to evenly cover the sample space, whose dimensionality is usually large, or to reduce the effect of the non-deterministic components of the pattern generation process (noise). This latter phenomenon is well-known in the fields of speech analysis (e.g., classification of phonemes) and OCR, since a limited number of patterns such as basic phonemes or alphanumeric characters may appear in a very large number of variants.

Moreover, the Bayesian approach itself requires a good approximation to the probability density functions, since it is based on the principle of approximating *probabilities*, unlike other methods that focus on *decision regions* (see for instance [2]). These methods only require that the corresponding region borders be correctly approximated, whereas PNN computes the regions in an implicit way, that is, by comparing probability densities. Therefore the densities should be globally correct. This means that, as a classifier, the PNN may not be very robust with respect to errors on the representation of the $f^{(k)}(\mathbf{x})$, hence leaving out some patterns may degrade the performance more than for other classification criteria.

Finally, there is also an issue regarding the classification cost. A large number of learned patterns can lead to an opposite situation, as compared with the usual neural algorithms. The normal case is that of a system that needs a computationally intensive training, but that, in the classification phase, is very fast. A PNN instead is trained in time linear with the number of patterns and in one pass, but its classification phase requires combining all pattern to yield the Parzen estimate, hence the classification is very slow if compared to other models.

These problems can be summarized in the need for adapting a limited number of resources to an unbounded number of training cases. The training procedure should be modified accordingly, since it has to cope with a number of adaptable parameters that is smaller than in the theoretical model. On the other hand, there are also reasons that theoretically justify a reduced number of windows.

The Parzen estimate is adequate in the presence of continuous densities, as stated by the consistency property (equation 2). This property is referred to the asymptotic case, i.e., when an infinite number of patterns are available. However, all real problems involve limited samples. The densities to be approximated are therefore expected to be not only continuous but also smooth. In practice, smoothness in probability usually means that non-deterministic components are present in the input generation process not only on the class (as the classical "signal + noise" model assumes), but also on the positions of the input examples in the pattern space.

Since this is a necessary requirement, the question arises as to what level of smoothness is appropriate. The answer is not easy; however, as a qualitative consideration, adding too much detail to the representation of the empirically determined distributions may lead to overfitting. In such a situation, adding more pattern units than needed is obviously an error. The rationale underlying these arguments, which are referred in general to the statistical estimation of a parameter from a limited sample, can be found in works such as [10] and [11]. In [12] a broad overview of these and related problems is presented. But see also the original view presented in [13] and other related works.

3.2 Implementing PNN training with limited resources

The proposed approach relies on an empirical procedure, in that it is based on the cross-validation principle. Its main aim is the optimization of the parameters in the case of a limited number of available pattern units.

The Parzen estimate requires two choices. The selection of a suitable kernel function poses no particular problem, provided that some requirements are met. The user can thus adopt the function that better meets the needs of a specific implementation. As a reference, the Gaussian kernel is often used.

The effect of the smoothing parameter has been studied by Specht in [14]. For $\lambda \to 0$, the Parzen estimate reduces to an uninformative distribution, i.e., the uniform distribution. For $\lambda \to \infty$, it reduces to what in statistics is usually called the "empirical distribution," i.e., the one that attributes a finite probability $1/N$ to each of the N patterns in the training set, and a null probability to all other points of the pattern space. However, the results of the work cited above indicate that, within a sufficiently wide range of variations of λ, the classification performance does not change very much. Still, if there are fewer pattern units than training examples, this parameter may be of greater importance. Our proposed learning is based on the choice of a proper value for λ.

Two modifications to the basic structure are proposed. The first is the design of a criterion for selecting some patterns to be stored and rejecting others that are considered unnecessary. Since our aim is to reduce the number of needed units, a natural choice is to decide whether a pattern may offer a significant contribution to the classification. In case it does not, the probability estimate should be updated in an alternative way.

Consequently, the second modification implements this alternative updating. This consists in assigning new roles to the parameters of the model. We allow the value of the window size λ to vary from unit to unit. We also introduce additional parameters. The algorithm is not a one-shot procedure, but an iterative one; however, it remains computationally light.

Finally, the new algorithm ensures that the number n of implemented pattern units will not exceed the available number $n_{\max}$.

The pattern rejection mechanism introduces a distortion into the estimate of the class-conditional probabilities because the pattern frequencies do not represent anymore estimates of their probabilities. Therefore, we need to add normalization coefficients on the output of the pattern units. These take the form of adaptable weights $\mathbf{w}'$ on the path from the pattern units to the summation units, which now implement weighted sums. For the same reasons, also the class coefficients γ, which are fixed in the original PNN scheme, are now adaptable during training. The modified structure is sketched in Figure 2.

The modified algorithm is structured into two blocks: the first is the creation of the windows; the second is the optimization of the window parameters.

The first block is similar to the PNN training but features the rejection mechanism to avoid taking into account the patterns that have already been satisfactorily classified. Rejection is decided when the ratio of the winning class probability to the second maximum class probability is higher than a preset threshold value r. If the two estimated probabilities do not differ in a sufficiently large amount, the pattern is added. This block requires a random selection of the training patterns. The function $\mathrm{randominteger}(m, n)$ returns a random integer value v in the interval $m \leq v \leq n$.

Even if we use kernels with an infinite support (as Gaussians are), a division by zero is possible when implementing the algorithm with limited precision hardware. In the implementation, a test is required that is not shown here for simplicity.

The second block is the optimization of the parameters, and is performed only when there is no pattern rejection. The parameters are represented by a vector $\mathbf{p}$ containing the window sizes and the normalization coefficients. The optimization procedure is based on a cross-validation test. The error function to be minimized, denoted by $\mathrm{cost}(\mathbf{p})$, is computed as the mean classification error on the test set (T patterns):

$$\mathrm{cost}(\mathbf{p}) = \frac{1}{T} \sum_{l=1}^{T} \mathrm{error}(\mathrm{class}(\mathbf{x}(l)), c(\mathbf{p}, \mathbf{x}(l)))$$

where $c(\mathbf{p}, \mathbf{x}(l))$ is the class label computed on the pattern

$\mathbf{x}(l)$ using the parameters $\mathbf{p}$ and the function $\mathrm{error}(a, b)$ returns 0 if the class labels a and b are equal, and returns 1 otherwise. The function $\mathrm{class}(\mathbf{x})$ indicates the real class label associated with the pattern $\mathbf{x}$.

The optimization of the parameters is obtained by a random search algorithm, presented in the following. We indicate it by the function $\mathrm{optimize}(\mathbf{p})$ which, given a vector $\mathbf{p}$, returns a new value for it in the space of parameters. The optimization is iterated $I_{\max}$ times, the limit being set by the user. The selection mechanism ensures that only steps that lower the cost will be accepted.

The algorithm is outlined in the following pseudo-code procedure.

```
ALGORITHM: PNNtraining
begin procedure
mark all patterns as unused
while training set contains unused
    patterns and cost(p) < C_th
     repeat
          set l = randominteger(1, N) and
    set k = class(x(l))
     until x(l) used
     mark x(l) as used
     compute the class probability
    values
     choose class c with max
    probability
     choose class c' with the second
    max probability
     if f^(c)(x(l))/f^(c')(x(l)) < r and n < n_max
    then
          add a new pattern unit
    centred on x(l)
          while number of iterations
    ≤ I_max
               select new p = optimize(p)
          end while
     end if
end while
end procedure
```

The random search algorithm adopted was described in [15]. Here we apply the modifications presented in [16]. The selection criterion is random with a cost-dependent bias, that is adapted according to the costs in the past training steps. This seems to act as a heuristic criterion to mimic the gradient descent procedure, in a random way, in the cases where the true gradient descent criterion is not applicable. A convergence theorem for this algorithm was presented in [17].

The parameters of the algorithm are: ξ, the update step (a vector in the space of parameters); β, a bias vector that keeps track of "good" directions. The function $\mathrm{randomrealvector}(m, n)$ returns a random real vector ξ with components in the interval $m \leq \xi_i \leq n$. The maximum allowed value for the components is denoted by $\xi_{\max}$. This does not guarantee that the resulting point will be bounded within a given volume. Again, in the practical implementation, there is a supplementary test on the value of $\mathbf{p}$ not included in the presented algorithm.

```
ALGORITHM: optimize(p)
begin procedure
set ξ = randomrealvector(0, ξ_max)
if cost(p + ξ + β) < cost(p) then
     p = p + ξ + β and β = 0.4ξ + 0.2β
else
     if cost(p − ξ + β) < cost(p) then
          p = p − ξ + β and β = β − 0.4ξ
     else
          β = 0.5β
     end if
end if
return value = p
end procedure
```

The selection of a random search optimization procedure is due mainly to the lack of an analytical expression for the cost function. The simple form of the method and its relative speed yield good results with a small effort, while avoiding the need for reformulating a gradient-based algorithm.

In the proposed algorithm, the parameter λ has not the meaning of a smoothness parameter in the strict sense of Parzen's theory. This is because, as previously remarked, the pattern units layer does not take into account any more the relative frequencies of patterns. Hence we allow the pattern units to implement windows of different sizes and normalization factors. The key point is that now these parameters are adapted on the basis of the *generalization performance*, estimated on a test set. The rejection threshold allows one to decide if the number of learned patterns should be small (smooth probabilities) or large.

4 The application: Optical Character Recognition

4.1 The problem

We shall now present the application problem examined. The specific task is the recognition of handwritten digits, as

a module within systems for automated mail delivery. Typically, the reading of handwritten numerical information is related to the recognition of the ZIP code.

The data base [**?**] contains 2000 character images (200 examples for each decimal digit) of 448 binary pixels (28 by 16 images), as illustrated in the examples of Figure 3(a). The images are normalized in position and size. As an additional pre-processing, a local averaging is performed on 4 by 4 square areas. This reduces the input space dimension, so that the actual patterns are 7 by 4 grayscale images, or 28-dimensional vectors. Some examples of the resulting patterns are shown in Figure 3(b). Practical experience suggests that, in this kind of application, having fewer inputs with (approximately) continuous values is often better than having many binary inputs. This is because the information is "more concentrated" and can be exploited in a more efficient way. The pixel depth is 4 bits, which agrees with the 5-bit representation of the selected neural processor, described in the next subsection.

The recognition process starts with the described low level pre-processing. Then a classifier is used to identify, within acceptable error margins, the single characters that make up a numerical sequence. The PNN plays the role of such a classifier. It is also possible to adopt ensemble techniques for a more robust classification by building a team of different classifiers. The subsequent step can be the application of Hidden Markov Model analysis to identify the digits in their context.

For the training of the single digit classifier, the database is split into three sets: a training set (1000 patterns) for the training of the pattern units, a test set (800 patterns) for the cross-validation during training, and a final validation set (200 patterns).

4.2 The hardware

As previously remarked, the described algorithm has a general validity. However, to give a set of working conditions for the application described in the present study, we will describe the actual hardware platform on which the algorithm has been developed.

The Nestor/Intel Ni1000 Recognition Accelerator is a digital pattern recognition processor, designed to implement prototype-based classifiers. It can store 1024 reference patterns, of dimension 256 and of 5-bit precision in a Flash memory. These patterns can be used to apply several algorithms. The PNN, the Restricted Coulomb Energy (RCE) and the Probabilistic RCE (PRCE) are pre-programmed in microcode and executed by a 16-bit general-purpose microcontroller. The actual operations required by the algorithms are executed with the aid of a floating-point classifier engine that computes the L_1 metric, or "Manhattan" distance:

$$D\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}\right) = \sum_i \left| x_i^{(1)} - x_i^{(2)} \right| \qquad (6)$$

and the probability density functions. This structure, along with the availability of a RAM memory, allows the user to program custom algorithms.

The processor is completed by a software development environment, working on a PC under Windows interface, that provides the functions of data management, training and testing, as well as programming. A software emulator is also available and has been used to develop the application in the design phase.

4.3 Experimental results

It is not unusual to have huge quantities of data in OCR problems. In our case, the data sets are only a subset of the available data base; however, they are sufficient to saturate the capacity of the processor. The Ni1000 can store at most 1024 pattern units. But it should be stressed that the effective training set is composed of all the training patterns *plus* the cross-validation test patterns. This is due to the fact that the test patterns are actually used to drive the training process (this is true even in the case of a "passive" use of the cross-validation technique, such as *stopped training*). Therefore, the corresponding standard PNN should be made up of 1800 pattern units; this configuration would not be implementable on the selected hardware device.

For the same reasons, a direct comparison between the standard PNN and the modified PNN is not feasible. The presented results were obtained by a standard PNN trained on the training set alone, and by a modified PNN trained on the training set and on the cross-validation test set. Of course, since, for the standard PNN, the error on the training set is always null, in all cases the results are specified in terms of errors on the final validation test set.

The smoothing parameter here represents the variance of a Gaussian kernel:

$$g(\mathbf{x}) = \frac{1}{\lambda\sqrt{2\pi}} 2^{(\mathbf{x}-\mathbf{w})/\lambda} \qquad (7)$$

where the exponential in base 2 is used because it is available in hardware, and is computed with the following estimate:

$$\lambda = \frac{\alpha}{N} \sum_{l=0}^{N} \left| \mathbf{x}(l) - \mu^{(\mathrm{class}(\mathbf{x}(l)))} \right|^2 \qquad (8)$$

where $\mu^{(k)}$ = average pattern of the class k (barycentre of its distribution). The parameter α is used to correct empirically the Parzen estimate based on experimental evidence, and is set initially to 1 (no correction).

The overall performance can be assessed by the average test error rate, which is 10.7%, although the rate on the single classes is variable (as is well known, some handwritten characters are difficult to discriminate, e.g., a slanted "1" and a "7"). Instead of comparing error percentages, we verify how many pattern units are required to obtain a given performance.

The standard PNN requires 1000 units (one for each training pattern) to reach a 10.7% error rate. The modified PNN was trained with $C_{\text{th}} = 10\%$, an initial value $r = .2$ and $I_{\max} = 100$, and of course $n_{\max} = 1024$. The desired cost was obtained by 196 neurons, much fewer than the 1000 neurons required by the standard algorithm.

Table 1 summarizes the average results for different numbers of units. These results refer to a set of experiments aimed at the selection of the empirical parameters α and r. Although, in these experiments, the results were variable (due to the random evolution of different parameters), the number of neurons sufficient to reach an error percentage of about 30% was about 50, whereas the desired performance was reached by around 200 neurons. These are average values, since the random selection of patterns does not allow for a deterministic descent in the *generalization* performance, although it would if the training performance were measured. The overall error rate, being compensated for by the possible context reconstruction, is satisfactory from an application point of view, especially if the small size of the training set is taken into account.

5 Conclusions and future work

Experimental verifications have demonstrated that, if compared with the standard version, the modified algorithm requires a much smaller number of units to obtain a comparable performance level. This makes it not only suitable for a hardware implementation but also more flexible as a learning tool for classification. The PNN standard algorithm does not allow for the application of a cross-validation procedure or other methods for controlling its generalization performance.

The PNN is an interesting model, thanks to the properties presented in the introductive sections. However, the Parzen method (the basic inductive tool adopted by the network), was developed in the sixties when the *small sample* statistical approach was not available, as pointed out in [12]. Therefore its validity is somehow limited by the asymptotic approach.

The present research has adopted a different methodological point of view that is ultimately consistent with the small-sample principles. As a matter of fact, it is not aimed at the minimization of the empirical risk (classification error on the training set), but adopts, in the limits of practical applicability, an estimate of the generalization ability as its performance criterion (classification error on the cross-validation test set). To this end, the storage capacity of the network is controlled by monitoring the test set performance. Moreover, the structure of the network is modified to implement another principle of the small sample theory, in that it is able to add a percentage of *regularization* to the estimation of the probabilities. At the same time, the method takes into account the limited availability of resources for the practical realization of a PNN device.

The theoretical framework for the proposed method is a ground for future research, which will aim at a quantitative assessment of the properties of the model, as well as at the development of systematic criteria for assigning values to the training parameters. The parameter optimization procedure can also be studied in more detail to provide faster alternatives, although a properly tuned random search mechanism can avoid local minima, unlike other minimization algorithms.

Acknowledgements

The authors acknowledge the cooperation of D. Benigno and D. Badalacco, who implemented and tested the algorithm, and A. C. Carassino's valuable help for the installation and testing of the NI1000.

References

[1] Donald F. Specht. Probabilistic neural networks. *Neural Networks*, 3:109–118, 1990.

[2] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.

[3] E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[4] Donald F. Specht. Probabilistic neural networks and polynomial adaline as a complementary technique for classification. *IEEE Transactions on Neural Networks*, 1:111–121, May 1990.

[5] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pages 96–104, New York, 1960. IRE.

[6] Pietro Burrascano. Learning vector quantization for the probabilistic neural network. *IEEE Transactions on Neural Networks*, 2(4):458–461, July 1991.

[7] Teuvo Kohonen. *Self Organization and Associative Memories*. Springer Series in Information Sciences. Springer, 1982.

[8] Intel Corporation. *Ni1000 Beta Release 2.3 documentation*, 1994.

[9] T. Cacoullos. Estimation of a multivariate density. *Annals of the Institute of Statistical Mathematics (Tokyo)*, 18(2):179–189, 1966.

[10] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[11] John E. Moody. The *effective* number of parameters: an analysis of generalization and regularization in nonlinear learning systems. In John E. Moody, Steven J. Hanson, and Richard P. Lippman, editors, *Advances in Neural Information Processing Systems IV*, volume 4, pages 847–854, 1991.

[12] Vladimir Naumovich Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[13] David H. Wolpert. Off-training set error and a priori differences between learning algorithms. Technical Report SFI-TR-95-01-003, The Santa Fe Institute, 1399 Hyde Park Road – Santa Fe, NM, 87501, 1995.

[14] Donald F. Specht. Generation of polynomial discriminant functions for pattern recognition. *IEEE Transactions on Electronic Computers*, 16:308–319, 1967.

[15] J. Matyas. Random optimization. *Automation and remote control*, 26:246–253, 1965.

[16] F. J. Solis and J. B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6:19–30, 1981.

[17] N. Baba, T. Shoman, and Y. Sawaragi. A modified convergence theorem for a random optimization method. *Information sciences*, 13:159–166, 1989.

Table 1: Test error percentage versus number of units.

Number of units	Error percentage
10	38%
20	35%
30	32%
50	30%
100	25%
150	16%
200	11%

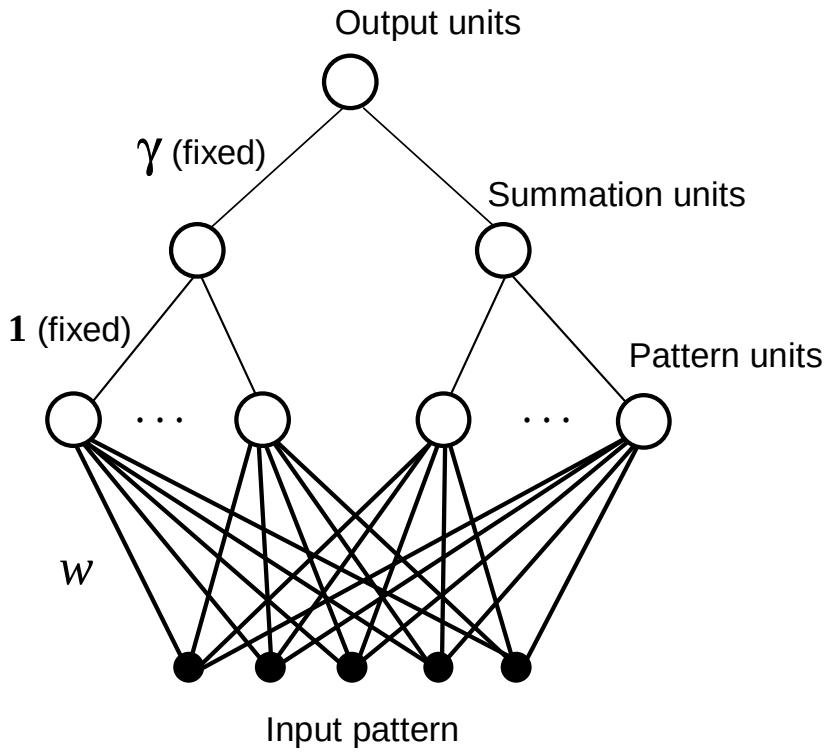

Figure 1: The PNN scheme. Heavy lines indicate adaptable weights.

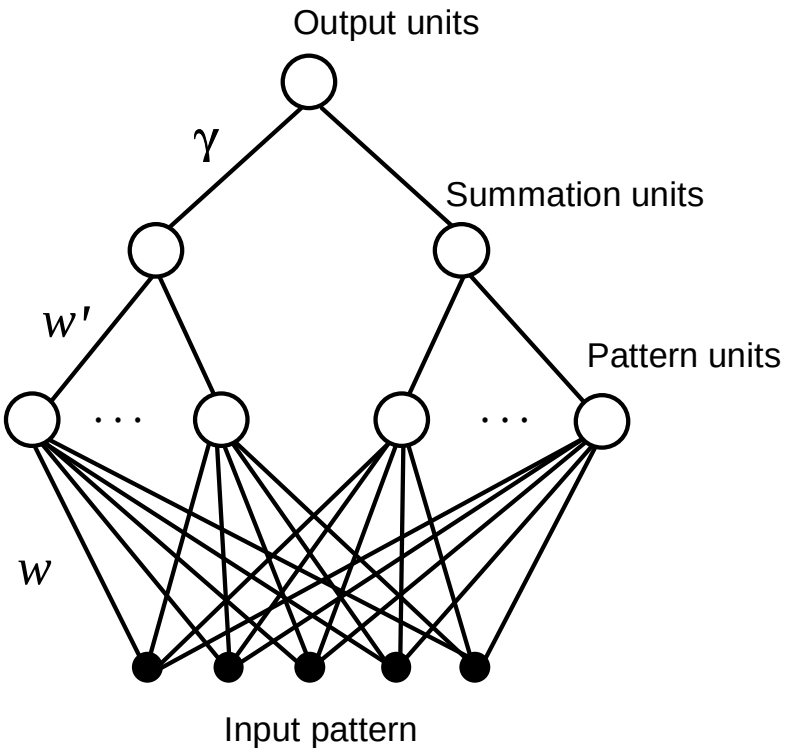

Figure 2: The modified PNN scheme. Heavy lines indicate adaptable weights.

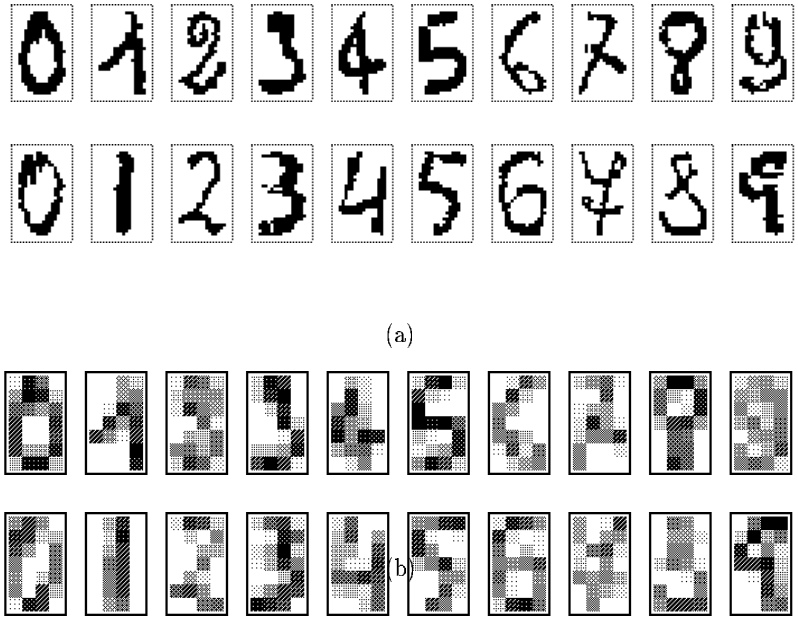

Figure 3: Some examples from the OCR database: (a) original, 1-bit per pixel images; (b) the corresponding pre-processed (greyscale) patterns[1]