In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, (ICANNGA'97)*, pp. 514-517, 1997.

Improved Center Point Selection for Probabilistic Neural Networks

D. Randall Wilson, Tony R. Martinez
E-mail: *randy@axon.cs.byu.edu*, *martinez@cs.byu.edu*
Neural Network and Machine Learning Laboratory, WWW: http://axon.cs.byu.edu
Computer Science Department, Brigham Young University, Provo, UT 84602, U.S.A.

Abstract. Probabilistic Neural Networks (PNN) typically learn more quickly than many neural network models and have had success on a variety of applications. However, in their basic form, they tend to have a large number of hidden nodes. One common solution to this problem is to keep only a randomly-selected subset of the original training data in building the network. This paper presents an algorithm called the Reduced Probabilistic Neural Network (RPNN) that seeks to choose a better-than-random subset of the available instances to use as center points of nodes in the network. The algorithm tends to retain non-noisy border points while removing nodes with instances in regions of the input space that are highly homogeneous. In experiments on 22 datasets, the RPNN had better average generalization accuracy than two other PNN models, while requiring an average of less than one-third the number of nodes.

1 Introduction

Probabilistic Neural Networks (PNN) [1] often learn more quickly than many neural network models such as backpropagation networks [2], and have had success on a variety of applications. PNN's are a special form of radial basis function (RBF) network [3] used for classification.

The network learns from a *training set T*, which is a collection of examples called *instances*. Each instance i has an input vector $\boldsymbol{y}_i$, and an output class, denoted as $class_i$. During execution, the network receives additional input vectors, denoted as $\boldsymbol{x}$, and outputs the class that $\boldsymbol{x}$ seems most likely to belong to.

The probabilistic neural network used in this paper is shown in Figure 1. The first (leftmost) layer contains one input node for each input attribute in an application. All connections in the network have a weight of 1, which means that the input vector is passed directly to each hidden node.

There is one hidden node for each training instance i in the training set. Each hidden node h_i has a center point $\boldsymbol{y}_i$ associated with it, which is the input vector of instance i. A hidden node also has a *spread factor*, σ_i, which determines the size of its *receptive field*. There are a variety of ways to set this parameter. In this paper, we set σ_i equal to a fraction f of the distance to the nearest neighbor of each instance i. The value of f begins at 0.5 and a binary search is performed to fine-tune this value. At each of five steps the value of f that results in the highest average confidence of classification is chosen.

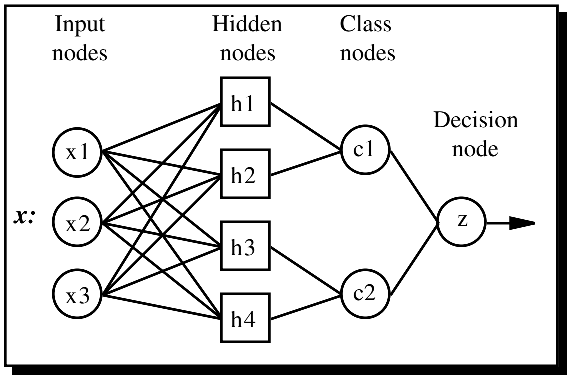

Figure 1. Probabilistic Neural Network.

A hidden node receives an input vector $\boldsymbol{x}$ and outputs an activation given by the Gaussian function g, which returns a value of 1 if $\boldsymbol{x}$ and $\boldsymbol{y}_i$ are equal, and drops to an insignificant value as the distance grows:

$$g(\boldsymbol{x}, \boldsymbol{y}_i, \sigma_i) = \exp[-D^2(\boldsymbol{x}, \boldsymbol{y}_i) / 2\sigma_i^2] \qquad (1)$$

The distance function D determines how far apart the two vectors are. By far the most common distance function used in PNN's is Euclidean distance. However, in order to appropriately handle applications that have both linear and nominal attributes, we use a heterogeneous distance function $HVDM$ [4][5] that uses normalized Euclidean distance for linear attributes and the Value Difference Metric (VDM) [6] for nominal attributes. It is defined as follows:

$$HVDM(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)} \qquad (2)$$

where m is the number of attributes. The function

$d_a(x,y)$ returns a distance between the two values x and y for attribute a and is defined as:

$$d_a(x,y) = \begin{cases} 1, & \text{if } x \text{ or } y \text{ is unknown} \\ vdm_a(x,y), & \text{if } a \text{ is nominal} \\ diff_a(x,y), & \text{if } a \text{ is linear} \end{cases} \tag{3}$$

The function $d_a(x,y)$ uses the following function, based on the Value Difference Metric (VDM) [6] for nominal (discrete, unordered) attributes:

$$vdm_a(x,y) = \sqrt{\sum_{c=1}^{C} \left| \frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right|^2} \tag{4}$$

where $N_{a,x}$ is the number of times attribute a had value x; $N_{a,x,c}$ is the number of times attribute a had value x and the output class was c; and C is the number of output classes.

For linear attributes the following function is used:

$$diff_a(x,y) = \frac{|x-y|}{4s_a} \tag{5}$$

where s_a is the sample standard deviation of the values occurring for attribute a in the training set.

Each hidden node h_i in the network is connected to a single class node. If the output class of instance i is j, then h_i is connected to class node c_j. Each *class node* c_j computes the sum of the activations of the hidden nodes that are connected to it (i.e., all the hidden nodes for a particular class) and passes this sum to a decision node. The *decision node* outputs the class with the highest summed activation.

One of the greatest advantages of this network is that it does not require any iterative training, and thus can learn quite quickly. However, one of the main disadvantages of this network is that it has one hidden node for each training instance and thus requires more computational resources (storage and time) during execution than many other models. When simulated on a serial machine, O(n) time is required to classify a single input vector. On a parallel system, only O(log n) time is required, but n nodes and nm connections are still required (where n is the number of instances in the training set, and m is the number of input attributes).

The most direct way to reduce storage requirements and speed up execution is to reduce the number of nodes in the network. One common solution to this problem is to keep only a randomly-selected subset of the original training data in building the network. However, arbitrarily removing instances can reduce generalization accuracy. In addition, it is difficult to know how many nodes can be safely removed without a reasonable stopping criterion.

Other subset selection algorithms exist in linear regression theory [7], including *forward selection*, in which the network starts with no nodes and nodes are added one at a time to the network. Another method that has been used [8] is k-means clustering [9].

This paper presents an algorithm called the Reduced Probabilistic Neural Network (RPNN) that begins with all of the available training instances as node centers and selectively removes them one at a time until classification accuracy suffers. The algorithm tends to retain only non-noisy border points while removing nodes with instances in regions of the input space that are highly homogeneous. The next section gives details of this algorithm.

2 Reduction Algorithm

The Reduced Probabilistic Neural Network (RPNN) begins with one node per training instance, as does the original PNN, and then uses the following basic rule to determine which nodes are removed from the network:

> *Remove a node if it does not cause more instances in the original training set to be classified incorrectly by the nodes remaining in the network.*

In other words, if the removal of a node does not hurt classification, remove it. When applying this rule to the network, the order of removal is important. In particular, it may be desirable to remove instances far from decision boundaries first, since they have the least effect on decisions. RPNN does this by finding the distance of every instance from its nearest *enemy*, which is the nearest neighbor of a different class, and then sorting the instances by that distance. The above rule is then applied beginning with the node furthest from its nearest enemy and proceeding to that which is closest to its nearest enemy.

In order to decide if the removal of a node degrades classification accuracy, each instance in the original training set is queried to see if its classification would be altered by the removal of the instance in question.

Specifically, in our serial implementation each instance maintains a vector of activations with one activation level for each class. The removal of a particular node would subtract some amount of activation (dependent on the distance) from its own class if removed. In addition, if the removed instance *I* is the nearest neighbor of some other instance *A*, then *A* must find a new nearest neighbor, and update its σ accordingly, which in turn changes what effect *A* has on all other instances.

The change in activation due to both the removal of *I* and the possible change in σ of other nodes may be enough to cause the classification of some instances to change. The change can cause an instance that used to be correctly classified to be misclassified, or cause an instance that was misclassified to be correctly classified. Such changes are counted, and if the number of newly misclassified instances is less than or equal to the number of new correctly classified instances, then the removal is performed, and the changes in activation values and σ parameters are made permanent. Otherwise they are restored to their previous values.

In order to reduce the effect of noisy instances on the network, the instance corresponding to the node that is being considered for removal is not included in the tabulation. This means a node can be removed even if its instance is itself no longer classified correctly, as long as other instances are not hurt.

To further reduce the effect of noise, a noise-reduction pass through the network is done first, beginning with the instance closest to its nearest enemy, since noisy instances are often close to instances of another class. During the noise reduction step, the criteria for removal is more strict. In order to be removed, an instance must not hurt classification, as explained above, and it must also strictly increase the average *confidence* of classification. The confidence for each node is defined as the activation of the correct class divided by the sum of activations for all of the output classes.

Noisy instances are often located near instances of another class but far from instances of their own class, so their removal will increase confidence of nearby instance's classification while having a much smaller effect on instances of their own class. Other instances, however, will typically lower confidence of nearby neighbors, which are largely of the same class, while having a smaller effect on instances of different classes. Therefore, the test of confidence is appropriate during the noise-reduction pass, but would prevent almost any pruning from taking place if used in the remainder of the algorithm.

3 Empirical Results

The Reduced Probabilistic Neural Network (RPNN) algorithm was implemented and tested on 22 applications from the Machine Learning Database Repository at the University of California, Irvine [10].

Each test consisted of ten trials. Each trial consisted of learning from 90% of the training instances, and then seeing how many of the remaining 10% of the instances were classified correctly.

The RPNN was compared to EPNN, a standard Probabilistic Neural Network (PNN) that retains 100% of the instances in the training set and uses a normalized Euclidean distance metric with σ set to the distance of a node's instance to its nearest neighbor. The RPNN was also compared to HPNN, a PNN that uses the same heterogeneous distance function HVDM as RPNN, but retains 100% of the instances. HPNN and RPNN both used a dynamically-adjusted spreading factor, as explained in Section 1, and RPNN used only a subset of the available instances for generalization.

Table 1 summarizes the empirical results. For each database the table shows the average accuracy for the EPNN and HPNN using all of the instances, and for the RPNN, using the percentage of instances shown.

The last line of Table 1 shows that RPNN had the highest average accuracy over all 22 datasets of the three algorithms, while using less than one-third of the instances (on average) for generalization. RPNN's average accuracy was slightly higher than HPNN, and both of these were substantially higher than EPNN, due in part to the use of the HVDM distance function.

Using a dynamically-adjusted spreading factor had very little effect on the accuracy of HPNN (less than 1% on average), but resulted in a large improvement on RPNN (75.5% accuracy instead of 71.5%) as well as improved size reduction.

The success of RPNN varies depending upon the application. For example, on the *Vowel* dataset, it retained almost two-thirds of the instances while suffering a large drop in accuracy compared to the other two models. However, in the *Echocardiogram* dataset, the RPNN used only 9% of the data while improving generalization accuracy by over 12%. Future research will focus on identifying characteristics of applications that help determine whether the RPNN model is appropriate.

It should be noted that these datasets are not especially large (only a few hundred instances in most cases), and that the reduction in size can be even more dramatic when there are more instances available. This is especially true when the number of instances is large compared to the complexity of the decision surface.

Dataset	EPNN	HPNN	RPNN	*(size)*
Anneal	76.2	76.2	94.9	*38.3*
Audiology	36.0	57.0	54.0	*38.9*
Australian	80.1	83.8	79.7	*27.5*
Breast Cancer (WI)	97.0	94.7	92.9	*20.0*
Bridges	52.4	55.2	57.3	*21.4*
Crx	75.4	84.4	82.3	*27.2*
Echocardiogram	78.0	76.8	90.9	*9.0*
Flag	45.7	53.6	47.0	*35.7*
Heart (Hungarian)	64.0	66.7	80.3	*25.1*
Heart (More)	46.0	68.8	71.8	*21.3*
Heart	80.7	81.5	73.3	*24.8*
Heart (Swiss)	38.9	93.5	78.3	*1.4*
Hepatitis	79.3	80.6	77.3	*15.3*
Horse-Colic	67.1	67.1	68.7	*17.8*
Iris	94.0	91.3	94.7	*44.2*
Liver-Bupa	62.5	66.6	57.6	*29.2*
Pima-Indians-Diabetes	76.3	74.1	67.2	*30.3*
Promoters	54.3	84.5	88.7	*52.5*
Soybean-Large	13.0	13.0	49.9	*51.5*
Vowel	92.0	92.4	84.7	*65.2*
Wine	94.4	97.2	92.2	*40.7*
Zoo	78.9	71.1	82.2	*26.1*
Average	**67.4**	**74.1**	**75.7**	***30.2***

Table 1. Generalization Accuracy of PNN and RPNN.

4 Conclusion

The Reduced Probabilistic Neural Network (RPNN) reduces the size and execution time of a PNN by removing nodes from the network that are estimated to be least needed for proper generalization. It tends to retain non-noisy border points in the input space while removing nodes that are either noisy or have centers that are far from the decision boundaries. By so doing, it can fairly quickly find a reasonable subset of nodes to include in the PNN, thus reducing network complexity and execution time, as well as reducing sensitivity to noise.

The RPNN requires O(n^2) time for learning on a serial machine, but only O($n\log n$) time in a parallel network, and in our experiments on 22 datasets reduced storage by over two-thirds on average.

It is possible that the RPNN could achieve even higher size reduction as well as more robust accuracy by employing search techniques such as genetic algorithms after initial pruning. Such search techniques could find additional nodes to remove, fine tune the spreading factor of individual nodes, and even adjust the nodes' center points. Future research will address this question, and continue to seek improved size reduction techniques. The results of this study are encouraging and show the potential for substantial reduction without sacrificing generalization ability.

References

[1] Specht, Donald F., "Enhancements to Probabilistic Neural Networks," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '92)*, 1, pp. 761-768, 1992.

[2] Rumelhart, D. E., and J. L. McClelland, *Parallel Distributed Processing*, MIT Press, 1986.

[3] Wasserman, Philip D., *Advanced Methods in Neural Computing*, New York, NY: Van Nostrand Reinhold, pp. 147-176, 1993.

[4] Wilson, D. Randall, and Tony R. Martinez, "Heterogeneous Radial Basis Functions," *Proceedings of the International Conference on Neural Networks (ICNN'96)*, 2, pp. 1263-1267, 1996.

[5] Wilson, D. Randall, and Tony R. Martinez, "Improved Heterogeneous Distance Functions," *Journal of Artificial Intelligence Research (JAIR)*, 6, 1, pp. 1-34, 1997.

[6] Stanfill, C., and D. Waltz, "Toward memory-based reasoning," *Communications of the ACM*, 29, 1986.

[7] Rawlings, J. O., *Applied Regression Analysis*. Wadsworth & Brooks/Cole, Pacific Grove, CA, 1988.

[8] MacQueen, J., "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, Berkeley, CA, pp. 281-297, 1967.

[9] Leonard, J. A., M. A. Kramer, and L. H. Ungar, "Using Radial Basis Functions to Approximate a Function and Its Error Bounds," *IEEE Transactions on Neural Networks*, 3, 4, pp. 624-627, 1992

[10] Merz, C. J., and P. M. Murphy, *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California Irvine, Department of Information and Computer Science, 1996. Internet: http://www.ics.uci.edu/~mlearn/MLRepository.html.