Parallelism, Localization and Chain Gradient Tuning Combinations for Fast Scalable Probabilistic Neural Networks in Data Mining Applications

Yiannis Kokkinos and Konstantinos Margaritis*

Parallel and Distributed Processing Laboratory, Department of Applied Informatics, University of Macedonia, 156 Egnatia str., P.O. Box 1591, 54006, Thessaloniki, Greece

Abstract. This work investigates the scalability of Probabilistic Neural Networks via parallelization and localization, and a chain gradient tuning. Since PNN model is inherently parallel three common parallel approaches are studied here, namely data parallel, neuron parallel and pipelining. Localization methods via clustering algorithms are utilized to reduce the hidden layer size of PNNs. A problem of localization may be present in the case of multi-class data. In this paper we propose two simple fast approximate solutions. The first is using sigma smoothing parameters obtained from the parallel PNN initial training directly to clustering. In this case a substantial reduction of neurons is achieved without significant loss of recognition accuracy. The second is an effort for an additional tuning. Via confidence outputs we employ a chain training approach to tune for the best possible PNN architecture.

Keywords: Parallel processing, Probabilistic Neural networks, data mining.

1 Introduction

Data mining tries to unlock and exploit the hidden patterns in databases [1]. Probabilistic Neural Networks (PNN) [2] are known intelligence tools for classification that derive knowledge directly from data and represent it in the form of simple well understood Bayesian models, which are most suitable for data mining applications that also need confidence levels. Bayesian classifier methods represent a powerful class of techniques to data mining, as they can in a strict mathematical sense to work under uncertainty. However the PNN hidden neuron size is usually of the order of the whole dataset size and the PNN operation is slow and demanding in memory and CPU resources. Thus for large scale systems and datasets the PNN usage is hindered. This encourages more research into the scalability of these techniques. Hence during the last five years, various works have been presented for mapping Probabilistic Neural Network in parallel processing systems, such as parallel PNN in Beowulf Clusters [3], in Grid mining with Map/Reduce [4], and in Graphic Processing Units [5]. All these works mainly focus on splitting the data-neuron matrix to speed up the slow execution

* Corresponding author.

I. Maglogiannis, V. Plagianakos, and I. Vlahavas (Eds.): SETN 2012, LNAI 7297, pp. 41–48, 2012.

times due to PNNs quadratic computational complexity, and in this way demonstrate that PNNs can be efficiently parallelized.

Yet the quadratic complexity of the original problem remains since Parallelism is only one path towards speedups. While run time can be reduced by parallelism, the computational complexity can be reduced by localization techniques which need fewer locally important neurons, to sum up for the probability distribution functions estimation. Use fewer units in PNN pattern layer but try to place them at optimal places. Like previous works clustering algorithms are also considered here to reduce the hidden layer size of PNNs.

A problem of such localization may be present in the case of multi-class data. In principle there exist no unsupervised algorithm that can sample by clustering the data points inside each class and select the best possible representative center points from every class such as a global accuracy criterion is simultaneously been satisfied. This problem is an extension of the well known k-centers problem and is NP hard [19]. In this paper we propose two simple fast approximate solutions for the above mentioned problem. The first is using sigma smoothing parameters obtained from the parallel PNN initial training directly to clustering. In this way a substantial reduction of neurons is achieved with negligible losses of classification performance. The second is an effort for an additional tuning. Via confidence outputs we employ a chain training approach to tune and test for the best possible PNN architecture. Details and experimental results from all methods are presented next.

2 PNN Architecture and Parallelization Mappings

The Probabilistic Neural Network [2-5] has four layers, namely input, pattern, summation and output. There are M classes in the output layer and each has N_m pattern neurons in the pattern layer, and a single $G_m()$, summation neuron in the summation layer. The d input layer neurons are the data features. The pattern layer is where train patterns are loaded and divided in M groups, one for each class. For an unknown sample X, the pattern neuron i of group m compute a Gaussian kernel of the form:

$$F_{m,i}(X) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\left(- \left\| X - X_{m,i} \right\|^2 / (2\sigma^2) \right) \tag{1}$$

where $X_{m,i}$ is the center of the Parzen kernel and sigma σ is the smoothing parameter (the width), that defines the range of each receptive field. The summation layer computes the conditional density functions by sum up the previous densities in $G_m(X)=(1/N_m)\Sigma F_{m,i}(X)$, and finally the output layer classifies the unknown X in class C_m that have maximum $G_m(X)\cdot h(C_m)$, where $h(C_m)$ is the class prior. The conditional probability for class C_m, can give also the confidence levels of this class and it is $Conf(C_m|X)= G_m(X)\cdot h(C_m)/(\Sigma G_m(X)\cdot h(C_m))$. A PNN with a single sigma parameter called homoscedastic, while the multi-sigma PNN called heteroscedastic. The normalization factor in the denominator of confidence (the prior of X) is the sum of all numerators for all categories C_m.

The main disadvantage of PNN network is that it has one hidden neuron for each training sample and thus requires more computational resources during execution than other models. On a serial machine, O(n) cost is required to classify a single input.

Exploitation of parallel mappings in Neural Networks can be achieved in many different levels from coarse grained to fine grained. The first, the session parallelism level (also called inter model parallelism in parallel data mining) places a different training model session to each processor. The next, the data parallelism, simultaneously learns in different training examples within the same train model session. The layer parallelism use concurrent computation for different layers. The neuron parallelism, use the same model and split the neurons to different processors. These last three levels also belong to intra model parallelism. A taxonomy review is in [6] and general guidelines on breaking any NN structure are in [7]. If the algorithm permits it, one can minimize point-to-point communication by pipelining the neuron calculations. Beyond selecting a representative training sample, PNN training phase is essentially a model selection procedure for the definition of sigma parameters and pattern neurons.

In PNN data Parallelism in Master/Worker architecture the Master node sends to all Workers the same copy of Neural Network, after that Master partition the data set and send a different partition to a different processor. Then for each epoch (a pass through all local data) each Worker independently process its local test set, exchanges its weight updates with other nodes, applies the weight updates to its copy of Neural Network, and computes local error rate and determines if local training is complete. The Search for best sigma parameters can be done by several repetitions of an epoch. This approach ensures that all parameter values required during the training phase, are locally available, decreasing the communication between the nodes and the synchronization of the parallel algorithm. The synchronization appears in the end of an epoch.

In PNN neuron parallelism the neurons are distributed in the processors. An algorithm of neuron parallelism in Master/Worker architecture is the following: 1) Master split data set into train and test (uses stratified sampling) and partition and distribute train set instances across workers to set up the local distributed pattern neurons. 2) Each Worker uses its local train set for the local pattern neurons. 3) Master broadcast the same train parameters to all workers. 4) For each test set instance Master sequentially Broadcast an instance X (or a batch B) of test set data to all workers, all workers compute local distances, kernels and send back partial sums for each class, Master reduce all partial sums for each class from all workers, and find the class of X.

In the PNN pipelined neuron parallelism, the communication time between processors is minimised to point-to-point, allowing each one node to receive-send messages with only his 2 neighbours (previous - next). Each machine keeps a partition of neurons, as in the PNN Neuron Parallel. In batches B, the evaluation points are loaded in the first node. For each batch, is calculated a list of partial sums of the class conditional probabilities for each class. The list and batch together are propagated to the next node in the pipeline that makes the same operation on them. A label of class is finally set in the ending node. This requires only point-to-point communication.

Model selection methods like leave-one-out cross-validation are often used in PNN training and evaluation for selecting the best sigma parameters. This is the one approach we use in the parallel implementations.

3 PNN Localization Problem and k-Centers Sampling

While PNN run time can be reduced by parallelism, the computational complexity can be reduced by localization techniques which need fewer locally important kernels, or neurons, to sum up for the probability density functions estimation. Localization tries to obey the parsimonious principle. The smaller and simpler is the better. All localization techniques require some additional regional information, specifically a list of k local points which are close to any given point x. Various clustering or sampling approaches are proposed to reduce neurons [8] for PNNs like LVQ in [9], k-means clustering [10], hierarchical clustering [11], the DDA algorithm in [12], Gaussian ARTMAP in [13], global k-means with Expectation–Maximization in [14] and k-medoids in [15]. After finding "representative" vectors for each class every sigma parameter depends on the clusters. The modified density equation for the PNN is:

$$F(x \mid c_m) = \frac{1}{\sum_{k=1}^{Nm} N_k^m} \sum_{k=1}^{Nm} \frac{N_k^m}{\prod_{i=1}^{d} \sigma_{ik}^m} \exp\left[- \sum_{i=1}^{d} \frac{(x_i - X_{ik}^m)^2}{2\sigma_{ik}^2} \right] \tag{2}$$

where N_k^m is the number of initial samples that covered from the kth cluster in class m, while σ_{ik}^m and X_{ik}^m are the smoothing parameter and the mean (or the medoid) in ith dimension for the cluster kth in class m, respectively. However the cluster based sigma parameters estimation tends to deteriorate the classification performance and optimization of sigma parameters, as well as of the network weights, is necessary.

In this work we compare many existing samplers like Affinity Propagation (AP) [16], k-Medoids (KM) [1], Subtractive Clustering (SC) [17] and Farthest Point clustering (FPC) [18] to sample important points inside classes.

A problem of such localization may be evident in the case of multi-class data. An efficient algorithm must select the best samples from a class and doing this with respect to the best samples from the other classes. Yet this problem is a multi-class extension of the well known k-centers problem or k median objective and is NP hard [19]. In this paper we propose two simple fast approximate solutions to work around the problem. The first we will demonstrate is using the sigma smoothing parameters obtained from the parallel PNN initial training directly to Subtractive Clustering. In this way a substantial reduction is achieved without significant loss of recognition accuracy. The second is an effort for chain training for an additional tuning.

Affinity Propagation and k-Medoids are accurate but very slow and not incremental in the sense that the next cluster point is not dependent than the previous. They are included for comparison only. We parallelise Subtractive clustering and Farthest Point clustering (FPC) as they are the only incremental, and fast and more suitable for large scaling. Extensive details will be presented elsewhere. Note that FPC needs to predefine the number of centers while Subtractive Clustering not. Nevertheless, to select sufficiently the best centers SC requires a training phase to find a suitable sigma parameter. Fortunately in this work we obtain such a sigma parameter already from the previously supervised parallel PNN training and propose here to use it.

4 Tuning by Chain Gradient Training Paradigm

Using sigmas from PNN training in the SC clustering procedure as proposed in the previous chapter may be efficient enough. Though in the localized version of PNN there is no generic method for selecting the best number of representative points from every class population in all cases. If needed, chain gradient training approach can provide additional tuning to test for the best possible tuned PNN architecture. First by using incremental clustering algorithms like subtractive clustering one can potentially sample most possible representative points from every class in an ordered fashion. Then the method gradually reduces the neuron size, equally from every class, and monitors an average 'gradient' of confidence levels from a whole inter-connected chain of network solutions. In localized PNN case this tuning is not meant for classification performance improvement, but rather for an additional reduction of neuron size without affecting its current performance.

Recall now that PNN is a data model and a data descriptor and also produce confidence levels. Let us suppose that a chain of two PNN instances namely A and B different only in the number of hidden neurons and a evaluation set ES is present.

Initially PNN(A) classifies its own A points to produce A-A confidence measure. PNN(B) classifies its own B points to produce B-B measure. Then PNN(A) classifies B points to produce A-B measure and PNN(B) classifies A points to produce B-A measure. In the third step PNN(A) classifies evaluation set ES to produce A-ES measure and PNN(B) classifies ES to produce B-ES measure. Finally, ES set classifies A and B patterns. The first two are internal measures, the next two are local and the last four are external. A schematic representation is given in fig. 1.

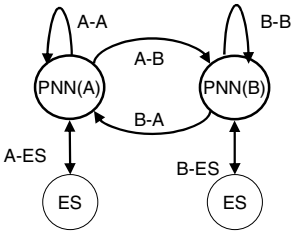

Fig. 1. A chain of two Probabilistic Neural Networks A and B different in the number of patterns, inter-connected with each other via confidence-based measures, indicated by arrows

One can have a sequence of several PNNs different in the number k of hidden neurons only. This chain training tuning scheme must be robust against oscillating local minima. The representative sampled points from every class, in an ordered fashion, are loaded to the hidden layer. If the hidden neuron size is gradually increased then we expect the chain global measure to soon reach a plateau. Confidence levels produced by the output neurons must progressively be increased for correctly classified patterns and be decreased for the falsely classified patterns. An efficient measure that can be used, is the (1+ sum of confidences for falsely classified patterns) / (1+ sum of confidences for correctly classified patterns) ratio. The next paragraphs present some experimental results.

5 Experimental Results

The experimental results for PNN data parallel, PNN neuron parallel and PNN neuron pipelined simulations are determined for 1..2..4.. 6..8 .. 10.. 12.. 14.. 16..24 machines. The speedup is S / P, where S the sequential run time in a single processor and P the time that simulates the network in parallel. All PNN Neural Network implementations are written in C using MPI library. PNN-CV train by cross-validation and PNN-SC with Subtractive Clustering are included.

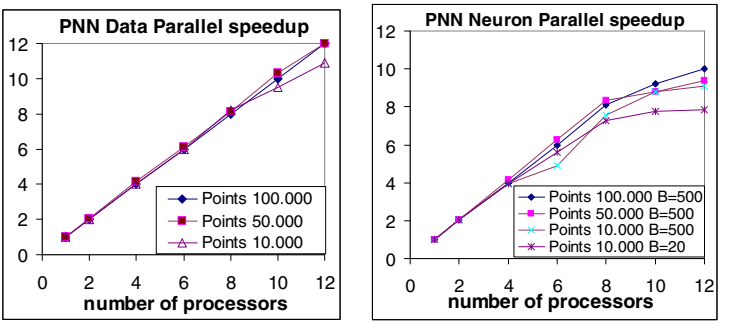

Fig. 2. (A) PNN Data Parallel, speed ups for data sizes of 10.000, 50.000 and 100.000 points, (B) PNN Neuron Parallel speedups respectively for two batch sizes B

In fig.2A for PNN data parallel with 10.000 points a linear speedup is observed up to 8 processors and when the size of problem is increased to 50.000 and 100.000 points we observe linear speedup up to 12 processors. In Fig. 2B for neuron parallelism results it appears a sub-linear speedup (9/12) up to 12 processors. This divergence is improved evidently when the size of problem is increased to 50.000 and 100.000 points and when the number of points in the batch increasing from 20 to 500.

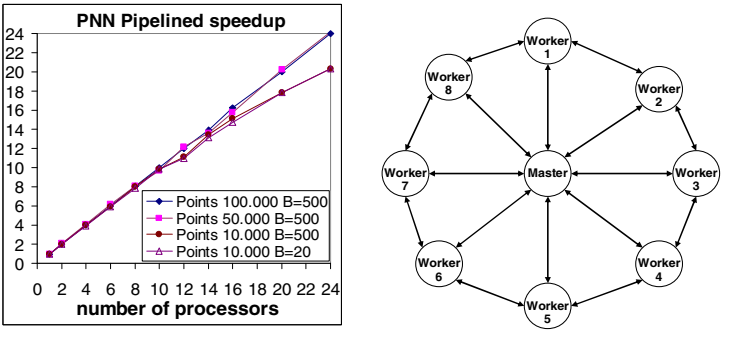

Fig. 3. (A) PNN neuron Pipelined speedups for two sets of 10.000, 50.000 and 100.000 points respectively for two batch sizes B, where one observes an impressive linear acceleration up to 24 processors. (B) The parallel programming model Master/Worker + Pipeline.

It is clearly show in fig. 3A for Pipelined calculations of 50.000 and 100.000 points an impressive linear speedup (24/24) that it is achieved up to all 24 processors we have tried. Results of PNN neuron pipelined are the best of all the previous methods.

For each network we compute error on the validation set. The network with the highest sum of confidences is then picked as one with the best generalization ability.

As standard we use several benchmark datasets downloaded from UCI ML Repository [20]. In table 1 average classification results are presented after 20 runs (10 with 50% train set and 10 with 80% train set) of each algorithm. The 1-NN is the one Nearest Neighbour rule, parallel PNN-CV is standard PNN with cross-validation, parallel PNN-FPC is localized PNN with Farthest point clustering, in where S means a single-sigma parameter used in equation 2, M means multi-sigma as a max distance from center and V means using variances and equation 2. PNN-AP is localized PNN with Affinity Propagation, PNN-KM is PNN with k-medoids. The best results are obtained with PNN-SC localized via the parallel Subtractive Clustering algorithm in which we use the same single sigma parameter founded by PNN-CV.

Table 1. Classification rates on the cross-validated test set for several benchmark datasets

Database	size	Classes	1-NN	PNN-CV	PNN-FPC			PNN-AP	PNN-KM	PNN-SC
					S	M	V	S	S	S
Iris	150	3	96	97 (σ=0.10)	93	95	92	96	96	**97**
Wine	178	3	94	95 (σ=0.22)	89	93	93	93	93	**95**
Wisconsin	683	2	97.5	98.5 (σ=0.25)	98	98	98	98	98	**98.5**
Yeast	1484	10	51	56 (σ=0.18)	45	25	25	47	49	**55**
Diabetes	768	2	70	76 (σ=0.25)	74	63	63	74	74	**75**

Sampling by AP or KM is very slow with a cost O(n^2 logn) for both and difficult to parallelise, but are included for comparison only. We parallelise efficiently PNN-SC and PNN-FPC. Only PNN-SC recovers the original performance, by using sigma parameter from previous PNN-CV in it. At most 10% points were extracted from each dataset. We observe that AP on average has similar results. Thus this localized approach produce at least 10 times faster PNN, in addition to 24 times speed up obtained from the 24 processors pipelined.

6 Conclusions

The first goal of this work is to speed up PNN. Using 24 processors the PNN training with cross validation and subtractive clustering approaches can impressively be speed up by 24 times. The next goal is to answer the question of preserving the sigma parameter founded by PNN cross-validation how many and which points can be omitted from the pattern layer without significant loss in the PNN performance. Using the sigmas found from the standard PNN directly on the Subtractive Clustering inside classes one can select most representative points and produce a localized PNN with small pattern neuron size and excellent performance that is 10 times even faster than the original version. If needed, additional tuning can be done by the chain gradient to test for the best possible PNN architecture. This approach could also be used in the training phase of other reduced nearest neighbour types of classifiers.

References

1. Dunham, M.H.: Data mining introductory and advanced topics. Prentice Hall (2004)
2. Specht, D.: Probabilistic neural networks. Neural Networks 3, 109–118 (1990)
3. Secretan, J., Georgiopoulos, M., Maidhof, I., Shibly, P., Hecker, J.: Methods for Parallelizing the Probabilistic Neural Network on a Beowulf Cluster Computer. In: International Joint Conference on Neural Networks, IJCNN 2006, Vancouver, pp. 2378–2385 (2006)
4. Cardona, K., Secretan, J., Georgiopoulos, M., Anagnostopoulos, G.: A Grid Based System for Data Mining Using MapReduce. Technical Report, TR-2007-02 (2007)
5. Bastke, S., Deml, M., Schmidt, S.: Combining statistical network data, probabilistic neural networks and the computational power of GPUs for anomaly detection in computer networks. In: 19th International Conference on Automated Planning and Scheduling, Workshop on Intelligent Security (SecArt 2009), Thessaloniki, Greece (2009)
6. Šerbedžija, N.: Simulating Artificial Neural Networks on Parallel Architectures. IEEE Computer, Special Issue on Neural Computing 29(3), 56–63 (1996)
7. Pethick, M., Liddle, M., Werstein, P., Huang, Z.: Parallelization of a backpropagation neural network on a cluster computer. In: 15th IASTED International Conference on Parallel and Distributed Computing and Systems, CA, USA, November 3-5, pp. 574–582 (2003)
8. Specht, D.F.: Enhancements to the probabilistic neural networks. In: Proc. IEEE Int. Joint Conf. Neural Networks, Baltimore, MD, pp. 761–768 (1992)
9. Burrascano, P.: Learning vector quantization for the probabilistic neural network. IEEE Transactions on Neural Networks 2, 458–461 (1991)
10. Traven, H.G.C.: A neural network approach to statistical pattern classification by "semiparametric" estimation of probability density functions. IEEE Transactions on Neural Networks 2, 366–377 (1991)
11. Babich, G.A., Camps, O.I.: Weighted Parzen windows for pattern classification. IEEE Trans. Pattern Anal. Mach. Intell. 18(5), 567–570 (1996)
12. Berthold, M., Diamond, J.: Constructive training of probabilistic neural networks. Neurocomputing 19, 167–183 (1998)
13. Zhong, M., et al.: Gap-Based Estimation: Choosing the Smoothing Parameters for Probabilistic and General Regression Neural Networks. Neural Computation 19, 2840–2864 (2007)
14. Chang, R.K.Y., Loo, C.K., Rao, M.V.C.: A Global k-means Approach for Autonomous Cluster Initialization of Probabilistic Neural Network. Informatica 32, 219–225 (2008)
15. Georgiou, V.L., Alevizos, P.D., Vrahatis, M.N.: Novel approaches to probabilistic neural networks through bagging and evolutionary estimating of prior probabilities. Neural Processing Letters 27, 153–162 (2008)
16. Frey, B.J., Dueck, D.: Clustering by passing messages between data points. Science 315(5814), 972–976 (2007)
17. Sarimveis, H., Alexandridis, A., Bafas, G.: A fast training algorithm for RBF networks based on subtractive clustering. Neurocomputing, 501–505 (2003)
18. Gonzalez, T.F.: Clustering to minimize the maximum intercluster distance. Theoretical Computer Science 38(2-3), 293–306 (1985)
19. Bern, M., Eppstein, D.: Approximation Algorithms for Geometric Problems. In: Approximation algorithms for NP-hard problems, pp. 296–345. PWS Publishing (1997)
20. Frank, A., Asuncion, A.: UCI Machine Learning Repository. University of California, Irvine (2010), `http://archive.ics.uci.edu/ml`