A Modified Bottleneck Neural Network for Dimensionality Reduction

Eduardo Filemón Vázquez Santacruz, Debrup Chakraborty

Centro de Investigaciones y Estudios Avanzados del IPN
Av. IPN 2508, Col: San Pedro Zacatenco
Mexico D.F, Mexico
email:evazquez@computacion.cs.cinvestav.mx, debrup@cs.cinvestav.mx

(Paper received on July 10, 2007, accepted on September 1, 2007)

Abstract. We present a modification of the bottleneck neural network for dimensionality reduction. We call our scheme a modified bottleneck network (MBNN). Unlike a traditional bottleneck network for dimensionality reduction, an MBNN uses the class information and thus the transformed data can be suitably used for classification problem. We also propose a new technique to create ensembles of neural networks using multiple projections of the same data obtained from different MBNNs. We justify the suitability of the proposed method by some experiments on some classification problems.

1 Introduction

Dimensionality reduction is probably one of the most important task in any pattern recognition problem. This problem has been addressed from different viewpoints, and by various kinds of tools. Naively, dimensionality reduction can be defined as follows. Given the input data $X = \{x : x \in \Re^p\}$, we want to find a transformation $\Phi : X \rightarrow X'$, where $X' = \{x' : x' \in \Re^q\}$ and $q < p$. Typically the set X' optimizes certain criteria J. The criteria J usually depends on the problem that is being solved, like for a classification problem J may be a measure of class separability or the misclassification rate for a fixed classifier. Two important types of dimensionality reduction are feature selection, where only a subset of the p features present in the data points in X are selected and an other type is feature extraction where the data is projected into some low dimensional space. In the second type of transformation, the original features present in the data may loose their individual identity and the final features obtained in X' may be a combination of all features present in the data points of X. Note that the term feature extraction is a generic term which signifies extracting features from a given set of features and includes such transforms which increases the dimensionality of a given data set.

Dimensionality reduction enables obtaining a data whose representation is less complex. This data thus can be learned by a learning machine which is less complex. This gives rise to a computational saving in construction and use of the learning machine. Also dimensionality reduction can give rise to improved prediction accuracies in scenarios where the training sample size is small (which is true in most real life scenarios)[12].

Special Issue in Neural Networks and Associative Memories
Research in Computing Science 28, 2007, pp. 127-136

The problem of dimensionality reduction has been well addressed in literature and it has been tried out in various paradigms. Previous studies on dimensionality reduction focused mainly around statistical approaches like Principal Components Analysis (PCA)[14], linear discriminant analysis (LDA) [10] etc. These methods attempt to reduce the dimensionality of the feature space by creating new features which are combination of the original ones. Hence, PCA and related methods are feature extraction techniques which extract a new set of features from the available set of features, and the dimensionality of the extracted feature space is less than that of the original one. There are numerous variants of the PCA which tries to solve the projection problem using various modifications of the main PCA technique [24]. The main drawback of these methods is that the new features lose their original identity. Leaving aside the feature extraction methods, there have been other works on feature selection using statistical techniques [13, 18].

Blum and Langley [4] have given an excellent survey for selection of relevant features in machine learning. These approaches are different in evaluation of the feature subsets. One can broadly classify the approaches as filter approaches and wrapper approaches. In filter approach, the feature evaluation index is independent of the main classification/function approximation algorithm, whereas in wrapper approaches the features are evaluated by the main algorithm itself. Wrapper approaches are considered better as the relevance of a feature is generally dependent on the task being performed and also on the tool being used to do the task [16].

There are many feature selection algorithms that use soft computing or computational intelligence tools. Methods described in [6, 19] use genetic algorithms to select the relevant feature subsets. Methods described in [20, 22, 23, 25] and a variety of others use neural networks for feature selection. Feature selection has also been attempted using fuzzy and neuro-fuzzy techniques [8, 21].

Our contribution: In this paper we discuss a technique to do dimensionality reduction using a neural network. Our method is a feature extraction technique. Our technique projects a given data into a low dimensional space while preserving the class separability of the data. Our technique depends on an special neural architecture called the bottleneck neural network. The bottleneck neural network has been previously used for data compression. The traditional bottleneck network do not use class information present in a data thus the compression achieved is sort of unsupervised, which may not be suited for classification problems. We discuss a simple variant of the bottleneck network which uses the class information for classification. We discuss several ways to use the transformed data for classification. We also propose a new technique to create neural network ensembles using the projected data obtained from a modified bottleneck network. We test the method on some real life classification problems.

2 The Bottleneck Neural Network

The bottleneck neural network is a nice strategy which has been used for data compression. Given a data set $\Xi = \{\xi_1, \xi_2, \ldots \xi_m\} \subset \Re^p$, a multilayered perceptron

(MLP) with p nodes in the input and output layer and q ($q < p$) nodes in a hidden layer is trained. Further we shall refer to such an architecture with the notation $p : q : p$. For training this network input-output pairs of the form (ξ_i, ξ_i) $(i = 1, 2, \ldots n)$ are used. So the network is trained to learn an identity map, and for a properly trained network, the output for ξ_i would be ξ_i itself. Now, the output of the hidden layer for each ξ_i would be a q dimensional representation of ξ_i. Thus a reduction of dimensionality is obtained. The bottleneck neural network has been previously used in many applications [3, 17].

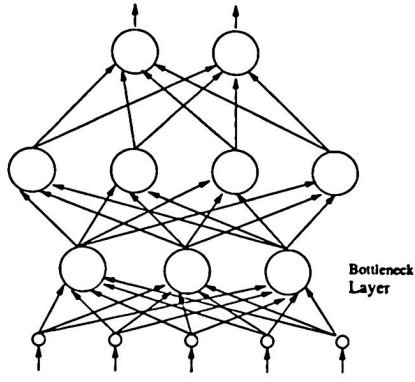

Fig. 1. The Modified Bottleneck Network

2.1 Modified Bottleneck Network

Here we propose a variant of the bottleneck neural network which we call the modified bottleneck network (MBNN). Let $X = \{(x_i, y_i) : i = 1, 2, \ldots n\}$ be a training data where $x_i \in \Re^p$ and $y_i \in \Re^s$. We train an MLP with this data. We have no restriction on the architecture of the MLP except that at least one of the hidden layers (say, the r^{th} layer) contains q (where $q < p$) nodes, we call this layer as the bottleneck layer. Note, unlike the traditional bottleneck network in MBNN we are not training the network to learn an identity map but our network tries to learn the real input-output mapping present in the data.

Let $\mathcal{B}$ denote a MBNN. $\mathcal{B}$ is not different from an ordinary MLP in terms of structure or the learning algorithm except the minimal restriction on its architecture as stated earlier. The output of $\mathcal{B}$ is not the output of its output layer but is the output of the bottleneck layer. For example, let us consider a data set $S = \{(x, y) : x \in \Re^5, y \in \Re^2)\}$. Figure 1 shows a MBNN trained with S. As evident from Fig. 1, the MBNN has an architecture of $5 : 3 : 4 : 2$ with the first hidden layer (the hidden

layer containing 3 nodes) as the bottleneck layer. One could have also chosen the second hidden layer as the bottleneck layer. The network is trained with the data set S using any convenient learning algorithm like the backpropagation or some of its variants. For using this network, the output is tapped from the first hidden layer. So for any input vector $\xi \in \Re^5$, the output of the network would be $\alpha \in \Re^3$.

For the general case, let $\mathcal{B}_X$ be a MBNN trained with $X = \{(x_i, y_i) : x \in \Re^p, y \in \Re^s)\}$. Let the r^{th} layer of $\mathcal{B}_X$ be the bottleneck layer with q nodes. Let $\alpha_i \in \Re^q$ denote the output of the r^{th} layer of $\mathcal{B}_X$ for a data point x. From a properly trained neural network we collect the output of the r^{th} node and this output serves as a projection of the original p dimensional data in a q dimensional space. We call this data as the *reduced data*. It is likely that the reduced data retains the properties required for learning the underlying function that the original data represents. This is because the reduced data represents an internal configuration of the data within the network, and truly this configuration helps the neural network in learning the proper mapping present in the data.

The reduced data set obtained from a trained network can further be used to train a new network possibly with a smaller architecture. Thus this will lead to lesser training time and possibly better generalization. Next we discuss some of the ways in which the reduced data obtained from an MBNN can be used.

3 How to Use the Reduced Data

An MBNN transforms a data set into a low dimensional space using a nonlinear transform. Also, an important feature of this transform is that it uses the output information (eg. the class labels for a classification data) for performing the transform. The reduced data represents an internal configuration of a trained network which further gets transformed into the output. This intermediate configuration is thus suitable to be used in any other neural network. Next we discuss some scenarios where a reduced data set may be useful.

3.1 As a Transform Before Training an MLP for Prediction

It is common knowledge that as dimensionality of a data goes up we need a bigger network to learn the data. A bigger network means more adjustable parameters, and as the number of parameters increases, the possibility of a network to overfit becomes more. An overfitted network though performs good with the training data can miserably fail to give acceptable results on test data sets. This problem becomes more acute if the number of training samples are small. To avoid this a natural and popular practice is to apply a dimensionality reduction on the data before training a network with the data. Thus by reducing the dimension of the original data, the data can be learned by a smaller network and thus possibility of overfitting becomes less. The most used technique is to use Principal Components Analysis (PCA) on the data. The PCA technique is a generic technique for dimensionality reduction and it projects the data in a new space where it has maximum variance. This may

not be the best objective for projection when we know what we need to do with the data. For example, when the desired task is classification, class separability may be a better objective than maximizing the variance. Also, if the targeted tool is a multilayered perceptron, which functions in a highly nonlinear manner and produces highly nonlinear class boundaries, use of the PCA (or other related projection techniques like LDA) may not be always useful. The reduced data set obtained by a MBNN will have better representation than the other known data projection methods in terms of classifiability by a neural network. The reason for this being that the reduced data obtained by a MBNN is an internal representation of a neural network meant for classifying the data.

A critique to this method may be that we need to train a network to get the reduced data itself. Thus for data sets with huge dimensions the MBNN may itself overfit the data if we need a network which performs good on the training data. This is of course true. But as the MBNN is not used for prediction purposes, an overfitted MBNN may not do us much harm. And we extract the reduced data of low dimension from an MBNN and train a new network (possibly much smaller than the MBNN itself), thus the network we use for prediction will have better generalization capabilities.

3.2 In Constructing Neural Network Ensembles

Ensemble methods like bagging [5] and boosting [9] can enhance the prediction ability of any classifier to a great extent. Neural network ensembles has also been reported to perform better than single networks. Bagging is a method very suited to neural networks, as bagging can decrease the variance of predictions in classifiers, and neural networks which are known to be "unstable" have large variance in prediction [5]. For neural networks, bagging involves creation of multiple bootstrap samples from a given data set and training multiple neural networks with those bootstrap samples. The final result is obtained by suitably aggregating the outputs of the candidate networks of the ensemble. Thus creating a bagging ensemble of neural networks is quite expensive in terms of computation, as neural network training is expensive. Using an bottleneck neural network $\mathcal{BN}$ one can project the data to a low dimensional space, and use bootstrap samples of the projected data to train individual candidates of the ensemble. This can give considerable savings in training times.

3.3 In Methods Which Uses Explicit or Implicit Density Estimation

There is a family of works which observes that one of the reasons of the phenomenon of overfitting in MLPs is due to the fact that MLPs are trained with a finite training sample [7, 11, 15]. In a conventional training algorithm for training MLPs, the network gets trained with a fixed sample of training points in each epoch. In [11] it was proposed that overfitting can be reduced if the training set can be indefinitely expanded, and the network be trained with a new set of data in each epoch. To

achieve this, the authors in [11], use a method to estimate the probability density of the data, and generate a new data set following the same probability density in each epoch, thus the MLP faces new points in each epoch. In [15], the method reported in [11] was modified by improving the procedure for density estimation. In [7] another method to expand the training set was reported and it do not depend on explicit density estimation but uses a k-NN heuristic to generate new points in each epoch.

All these methods reported in [7, 11, 15] can reduce the overfitting in MLP networks to a great extent. But as in [11, 15] a sort of a kernel density estimate is considered using gaussian kernels, the correctness of the density estimate decreases with the increase of dimension of the data. In [7] a K-nn technique is used which also fails for high dimensional data. An easy fix of this problem may be to use the reduced data obtained from an MBNN for training and density estimation not the original data set.

3.4 Bagging with Multiple Projections of the Same Data

As discussed earlier bagging involves aggregating outputs of k classifiers trained with k different bootstrap samples drawn from a given data set X. The idea behind bagging is to create independent classifiers from the same data set. Each bootstrap samples thus acts as a different data set for each classifier. Here we propose to use different projections of the same data to train each candidate of the ensemble. We propose to use the MBNN for projecting the data.

Given a data set $X = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) : i = 1, 2, \ldots n, \boldsymbol{x} \in \Re^p, \boldsymbol{y} \in \Re^s\}$ we train k MBNNs $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$. Each $\mathcal{B}_i$ can have different architectures with different hidden layers and different hidden nodes. The only restriction in each $\mathcal{B}_i$ being that the input layer should contain p nodes, the output layer should contain s nodes, and for any hidden layer should contain q(where $q < p$) nodes. From each of the networks we can collect a different projection of the original data X, possibly in different dimensions. Thus using each $\mathcal{B}_i$ we obtain the following transform $\mathcal{B}_i : X \rightarrow X_i'$. Thus we train k new neural networks using the data sets X_i' for $i = 1, \ldots, k$. We create an ensemble of these k networks, and aggregate the final outputs using a suitable aggregation function.

4 Experimental Results

We study the suitability of this method by applying it to some real life classification problems. The data sets used are Lung Cancer, Sonar, Iono, DNA and Protein. The summary of the data sets are presented in Table 1.

In all the simulations performed we used the neural network tool-box of MATLAB. We used "traingdx" as the learning algorithm, which is a variant of backpropagation with an ability to automatically tune the learning rate when necessary. Tables 2,3,4,5 and 6 summarizes the results. Each row in the tables stands for a different method and the columns report the average performance (with standard deviation) in percentage on the test data. Each reported result is of a 10 fold cross

Table 1. Summary of the data sets

Data	Source	Number of points	No of features	No of Classes
Lung Cancer	[2]	32	56	3
Sonar	[2]	208	60	2
Iono	[2]	351	34	2
DNA	[2]	2000	180	3
Protein	[1]	698	125	4

validation repeated 20 times. We explain each row of each table in more details as follows.

Table 2. Results on Lung Cancer

	Methods	Performance
1	MBNN Single	80.16667 ± 4.1578
2	MBNN Bagging	80.6667 ± 5.23
3	MBNN MP Bagging	83.7778 ± 3.674
4	PCA Single	78.1667 ± 3.6388
5	PCA Bagging	75.8333 ± 4.2492
6	Full Data Single NN	77.1667 ± 5.6501
7	Full Data Bagging	82.44 ± 2.194

1. **MBNN Single:**We trained a MBNN with 10 nodes in the hidden layer to get a 10 dimensional reduced data. We trained 20 different networks with the same reduced data each having the same architecture with 10 nodes in a single hidden layer. Row 1 of the tables shows the average performance of those 20 networks.
2. **MBNN Bagging** Next using the same reduced data of 10 dimension as obtained in row one we perform a bagging with 10 candidates in the ensemble. We use the majority vote to aggregate the results of the 10 different candidate networks. We created 20 different ensembles using the same reduced data. Row 2 shows the average performance of those 20 ensembles.
3. **MBNN MP Bagging:** MP bagging signifies bagging with multiple projections. In this experiment we trained 10 different MBNN each with 10 nodes in the bottleneck layer and thus obtained 10 different projections of the data in 10 dimensions. We trained 10 different networks with 10 nodes in a single hidden layer and aggregated their results using majority voting. This experiment was also performed 20 times. The results of this experiment are summarized in row 3 of the tables.
4. **PCA Single:**We ran a Principal Components Analysis on the original data with all features and took the 10 most significant components thus reducing the

Table 3. Results on Sonar

	Methods	Performance
1	MBNN Single	82.1143 ± 3.048
2	MBNN Bagging	82.1929 ± 3.2514
3	MBNN MP Bagging	85.7929 ± 0.6167
4	PCA Single	77.6786 ± 1.4881
5	PCA Bagging	78.7857 ± 1.1756
6	Full Data Single NN	79.6214 ± 2.8686
7	Full Data Bagging	84.5429 ± 0.8891

Table 4. Results on Iono

	Methods	Performance
1	MBNN Single	89.9786 ± 0.9993
2	MBNN Bagging	90.2937 ± 1.1636
3	MBNN MP Bagging	90.8341 ± 0.2622
4	PCA Single	86.7246 ± 0.7528
5	PCA Bagging	86.1579 ± 0.6861
6	Full Data Single NN	87.8389 ± 1.7492
7	Full Data Bagging	91.0611 ± 0.9597

Table 5. Results on DNA

	Methods	Performance
1	MBNN Single	92.425 ± 0.675
2	MBNN Bagging	92.49 ± 0.5611
3	MBNN MP Bagging	94.585 ± 0.094428
4	PCA Single	90.72 ± 0.3048
5	PCA Bagging	90.6 ± 0.2856
6	Full Data Single NN	89.3 ± 0.1768
7	Full Data Bagging	94.8 ± 0.2619

Table 6. Results on Protein

	Methods	Performance
1	MBNN Single	76.3377 ± 1.4511
2	MBNN Bagging	76.4 ± 1.7995
3	MBNN MP Bagging	79.5584 ± 0.459
4	PCA Single	63.7143 ± 1.2786
5	PCA Bagging	64.4 ± 1.3853
6	Full Data Single NN	69.7633 ± 4.1204
7	Full Data Bagging	78 ± 0.5898

original data to a 10 dimensional data. Row 4 of the tables shows the average performance and the best performance among 20 MLPs with 10 hidden nodes in a single hidden layer trained with the data of 10 features obtained by using PCA.

5. **PCA Bagging:** The 10 dimensional data obtained by PCA is also used to train an ensemble of 10 MLPs (each 10 nodes in a single hidden layer) using bagging. Row 5 of the tables gives the average and best performance of 20 such ensembles created with the 10 dimensional data obtained by PCA.
6. **Full Data Single NN** Row 6 gives the result on the data with all features using a ordinary MLP.
7. **Full Data Bagging** Row 7 gives the results on the data sets for bagging using the full dimensional data.

The results clearly show that the reduced data retains the class separability for all data sets experimented with. For all the data sets the MBNN single gives a better result than a single neural network trained with the whole data. The results obtained by using a PCA for dimensionality reduction are significantly poorer than the results obtained using a MBNN. MBNN MP bagging gives very encouraging results in all cases. In fact the results obtained by MBNN MP bagging outperform all other methods and is quite comparable with the results obtained by bagging on the full data set.

5 Conclusion

We presented a simple modification of the traditional bottleneck network for dimensionality reduction. We also showed some specific ways to use the reduced data obtained from an MBNN. The method of multiple projection bagging seems to work quite well as a method to create neural network ensembles. The methods are tested on some real life classification problems and the results obtained are encouraging.

References

1. http://www.nersc.gov/~cding/protein
2. A. Asuncion and D.J. Newman, UCI Machine Learning Repository [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science
3. Y. Araki, T. Ohki, D. Citterio, M. Hagiwara, K. Suzuki, "A new method for inverting feedforward neural networks," IEEE International Conference on Systems, Man and Cybernetics, 2003. Volume 2, pp.1612 - 1617, 2003
4. A.L. Blum, P. Langley, "Selection of relevant features and examples in machine learning", *Artificial Intelligence*, vol 97, no 1, pp. 245-271, 1997.
5. Leo Breiman, "Bagging predictors", *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.
6. F.Z. Brill, D.E. Brown and W.N. Martin, "Fast genetic selection of features for neural network classifiers", *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp. 324-328, 1992.

7. D. Chakraborty and N. R. Pal, "Expanding the training set for better generalization in MLP," Proceedings of *International Conference on Communication, Devices and Intelligent Systems*, CODIS-2004, pp. 454-457, 2004.
8. R. De, N.R. Pal and S.K. Pal, " Feature analysis: neural network and fuzzy set theoretic approaches", *Pattern Recognition* vol 30, no 10,pp. 1579-1590, 1997.
9. Y. Freund and R.E. Schapire, "A short introduction to boosting", *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771-780, 1999.
10. K. Fukunaga, *Statistical Pattern Recognition*, Academic Press, San Diego, CA, USA, 1991.
11. L. Holmstrom and P. Koistinen, "Using additive noise in backpropagation training," *IEEE Trans. Neural Networks*, vol. 3, pp. 24-38, 1992.
12. A. K. Jain, B. Chandrasekaran, "Dimensionality and sample size considerations in pattern recognition practice," in P.R. Krishnaiah, N.L. Kanal (eds.), *Handbook of Statistics*, vol 2, North-Holland, Amsterdam, pp. 835-855, 1982.
13. A. K. Jain and D. Zongker, "Feature selection: evaluation, application and small sample performance," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 2, pp. 153-148, 1997
14. I. T. Jollife, *Principal Component Analysis*, Springer Verlag, New York, 1986.
15. G. N. Karystinos and D. A. Pados, "On overfitting, generalization, and randomly expanded training sets," *IEEE Trans Neural Networks* vol 11, no. 5, pp. 1050-1057, 2000.
16. F. Kohavi and G. John, "Wrappers for feature subset selection", *Artificial Intelligence*, vol 97, no 1, pp. 273-342, 1997.
17. M. Marseguerra ans A. Zoia, "The autoassosiative neural network in signal analysis: I. Data dimensionality reduction and its geometric interpretation", *Annals of Neuclear Energy*, vol. 32, pp. 1191-1296, 2005.
18. J. Novovicova, P. Pudil and J. Kittler, "Divergence based feature selection for multimodal class densities", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol 18, no 2, 1996.
19. M.L. Raymer, W.F. Punch, E.D. Goodman, L.A. Kuhn and A.K. Jain, "Dimensionality reduction using genetic algorithms", *IEEE Trans. on Evolutionary Computing*, vol 4, no 2, pp. 164-171, 2000.
20. D.W. Ruck, S.K. Rogers, M. Kabrisky, "Feature selection using a multilayered perceptron", *Journal of Neural Network Computing*, pp. 40-48, 1990.
21. M. R. Rezaee, B. Goedhart, B. P. F. Lelieveldt and J.H.C. Reiber, "Fuzzy feature selection," *Pattern Recognition*, vol. 32, pp. 2011-2019, 1999.
22. R. Setino, "Neural network feature selector", *IEEE Trans. Neural Networks*, vol 8, pp.654-662, 1997.
23. J.M. Steppe Jr, "Integrated feature and architecture selection", *IEEE Trans. Neural Networks*, vol 7, pp. 1007-1014, 1996.
24. M. Ture, I. Kurt, Z. Akturk,"Comparison of dimension reduction methods using patient satisfaction data", *Expert Systems with Applications*, Vol 32, pp. 422-426, 2007
25. J.M. Zurada, A. Malinowski and S. Usui, "Perturbation method for detecting redundant inputs of perceptron networks", *Neurocomputing*, vol 14, pp. 177-193, 1997.

Real World Applications