Orthogonal Transformation of Output Principal Components For Improved Tolerance To Error

T.P. Mann, C. Eggen[†], W. Fox[†], D. Krout, G. Anderson[†], M. A. El Sharkawi, R.J. Marks II

Department of Electrical Engineering
Box 352500
University of Washington
Seattle WA 98115

[†]Applied Physics Laboratory
1013 NE 40th St
University of Washington
Seattle WA 98112

Abstract – Preprocessing of data to be learned by a neural network is typically done to improve neural network performance. Output preprocessing is especially important since it directly affects the influence of error in the hidden layers on the error in the neural network output. Principal component analysis is a commonly used preprocessing method that can improve network performance by reducing the output dimensionality and reducing the number of parameters in a neural network model. Transforming the principal components of the outputs with an orthonormal matrix prior to scaling can further improve network performance.

I. INTRODUCTION

Preprocessing the outputs to be learned by a neural network is an important step prior to network training. If the outputs have magnitudes that are larger than the range of the activation function of the output nodes, then the outputs need to be scaled appropriately for training, and the neural network outputs must also be unscaled when the neural network is used. Additionally, principal component analysis [1] is often used to reduce the dimensionality of the outputs. This is beneficial since it reduces the number of parameters in a neural network model, and also eliminates redundancy in the dataset.

In order to distinguish between output data before and after preprocessing, unpreprocessed output vectors are referred to as data, and preprocessed output vectors are referred to as targets. The preprocessing methods are affine transformations that map data to targets, and the networks are then trained on the targets. Data error is used to refer to the difference between the neural network output and the data vectors after the preprocessing has been undone, and target error is used to refer to the difference between raw network output and target vectors.

The neural networks considered in this paper are multilayer perceptrons [2] with a linear output layer. The weights of the linear output layer are algebraically chosen, and then the network weights that are not in the linear output layer are chosen by training the neural network using the RPROP [3] algorithm. The default preprocessing method for vector valued output data in this paper is mean centering, followed by principal component analysis and then range scaling. The range scaling is necessary to match the dynamic range of the targets and the dynamic range of the nonlinearities of the sigmoidal activation functions. The principal component analysis and the range scaling in output preprocessing can each be expressed as matrix operations, and can be composed into a single matrix. The entries of this matrix will be used as weights that determine the node activations in the linear output layer. An alternative treatment is to further preprocess the data by taking linear combinations of the principal components before the range scaling step. This alternative treatment can improve the network performance in some cases.

A. Assessing Error in Outputs

When a network is used to compute outputs whose correct values are known, then the output preprocessing must be undone to evaluate the data error. There are several reasons to prefer evaluation of data error to target error. One reason is that the outputs often have units; for instance, in the sonar application discussed below, entries in the output data vectors to be learned have units of decibels, and it is appropriate to interpret network performance using the units associated with the data. Another reason is that if a neural network is to be used as a component in a larger system, then other components may have requirements for the output vectors that are incompatible with the raw network output. If this is the case, then the appropriate error to be analyzed is the error in vectors that must be used by other components in the system. Finally, evaluation of the data error makes it simpler to distinguish between error incurred by the principal component analysis (due to not spanning all of the variance with the principal components) and error incurred by faulty network predictions. The first kind of error can only be corrected by adjusting the preprocessing steps, but the second kind of error can be corrected by adjusting weights before the last hidden layer, e.g., by network training. Examining only the target error can not yield information about variance in the data not spanned by the chosen principal components.

B. Multilayer Perceptrons

The notation used in this paper to express the function computed by a multilayer perceptron is as follows. Let x_j be a vector whose entries are the activations of the nodes in the *jth* hidden layer of a multilayer perceptron. The vector of input node activations to a multilayer perceptron is represented by i, and the vector of output node activations is represented by o. The sigmoidal activation function is written with the symbol 'σ'. The weights of the connections used to determine the activations of nodes in layer j are denoted by a matrix M_{j-1}, that represents connections between node in layer j and nodes in layer $j-1$, and a vector b_{j-1}, that represents the biases for nodes in layer j. The output of a multilayer perceptron with a linear output layer and L hidden layers is defined with the following recursive equations:

$$o = M_L x_L + b_L, \tag{1}$$

$$x_j = \sigma\left(M_{j-1} x_{j-1} + b_{j-1}\right), j > 0, \tag{2}$$

$$x_0 = i. \tag{3}$$

The entries of the matrix M_L and the vector b_L are explicitly chosen by the output preprocessing steps, and the other weights are chosen by neural network training on the vectors after preprocessing.

II. PREPROCESSING AND ERROR

Output preprocessing has an intrinsic effect on the tolerance to error of the multilayer perceptron. This is due to the interaction of the scaling of the outputs and the nonlinearity of the sigmoidal activation function. The sigmoidal activation function can be used to increase the tolerance of a multilayer perceptron to error. If a scalar target is in the nonlinear region of the sigmoid, then the error will be less for a given error δ in a prediction before the sigmoid is applied than if the target is in the linear region of the sigmoid. For vectors, the error is generally less if the vector has a large norm than a small norm for this reason.

The default output preprocessing method is principal component analysis followed by range scaling of the outputs. The alternative method is to compute an orthogonal transformation of the principal components before range scaling. The matrix M_L is then used to undo the preprocessing and transform the last layer of hidden node activations into the final prediction of the output vector. For both the default preprocessing method and the alternative method, the matrix M_L will have orthogonal columns, and is the product of an orthonormal matrix O and a scaling matrix S: For the default preprocessing method, O will consist of the principal components of the data, and for the alternative method, O will be a matrix of linear combinations of the principal components. For M_L to undo the preprocessing, it must first undo the range scaling and then the orthogonal transformation of the preprocessing:

$$M_L = O^T S^{-1}. \tag{4}$$

The values of the vector of node activations in the last hidden layer, x_L^*, which yield the optimal approximation to a data vector d is

$$x_L^* = SO\left(d - b_L\right). \tag{5}$$

In general, there is a prediction error that causes the actual node activations in the hidden layer before the last hidden layer to deviate from the optimal node activations by an error vector ε:

$$x_L = x_L^* + \varepsilon. \tag{6}$$

The entries of ε are constrained, since the entries of x_L must lie in the range of the sigmoidal activation function, and any ε which causes the entries of x_L to exceed that range can not be the result of network error. However, for each admissible ε, there is a corresponding vector δ, which represents error in the layer *L-1:*

$$x_L^* + \varepsilon = \sigma\left(\delta + \sigma^{-1}\left(x_L^*\right)\right). \tag{7}$$

The vector δ represents error after transformation of the vector x_{L-1} by the matrix M_{L-1} and vector b_{L-1}, and not error in the vector x_{L-1} of node activations. Since the error vector δ is applied before the sigmoidal activation function is used to determine the activations of the nodes in layer L, all vectors are admissible as error vectors if applied before the sigmoid activation function.

A. Tolerance To Error

The tolerance to error, $T(M_L,D)$ [4], can be estimated as a function of output matrix M_L, and output dataset D. Suppose that D has n data vectors, each vector written as d_i, and for each data vector d_i there is an error vector δ_i. For notational convenience, the data set D is assumed to have vectors with zero mean, and the bias vector b_L is consequently the zero vector. Then the tolerance to error is computed:

$$T(M_L, D) = \frac{1}{n}\sum_{i=1}^{n}\left\| M_L \sigma\left(\delta_i + \sigma^{-1}\left(M_L^+ d_i\right)\right) - d_i \right\|. \tag{8}$$

Here M_L^+ is the pseudoinverse of M_L [5]. This quantity varies for different choices of M_L, for the same dataset, due to the nonlinearity of the sigmoidal activation function. Additionally, the geometrical properties of the output dataset will play a role and determine choices of weights for the matrix M_L that lead to small values of $T(M_L,D)$.

For empirical study of the role of the matrix M_L in network error, error vectors δ_i are treated as random variables that are drawn from a multivariate Gaussian distribution with zero mean and uncorrelated entries with the same standard deviation. The probability density function for each of the error vectors δ is

$$p(\delta) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}\delta^T \Sigma^{-1}\delta}. \quad (9)$$

In a multilayer perceptron, the error vectors for a particular data matrix are fixed once the weights are chosen, and are not random variables. However, treating the errors as random variables allows the error properties of the weight matrices *S* and *O* in the last layer of weights to be empirically studied without training many independent neural networks for a particular set of output weights.

III. WHEN TO ROTATE PRINCIPAL COMPONENTS

The tolerance to error of the default preprocessing method for a dataset *D* is $T(P^T S_P^{-1}, D)$, where *P* is the matrix of principal components and S_p is a scaling matrix. Some datasets will be sensitive to a set of orthonormal matrices such that for an orthonormal matrix *O* in that set, the value of $T(P^T O^T S_{OP}^{-1}, D)$ is less than $T(P^T S_P^{-1}, D)$. These datasets are characterized by a distortion in their spatial distribution when the range scaling is applied after rotation of principal components. Some datasets are relatively insensitive to choice of *O*, and the tolerance to error will not change significantly as *O* varies. For instance, rotation of a disk or ellipse does not change the tolerance to error. Rotation of a multivariate Gaussian has an effect on the tolerance to error, primarily due to the outliers that will always be part of a large sample of vectors drawn from a Gaussian distribution. Fig. 1 shows a dataset that has been selected to illustrate conditions where the choice of *O* is important.

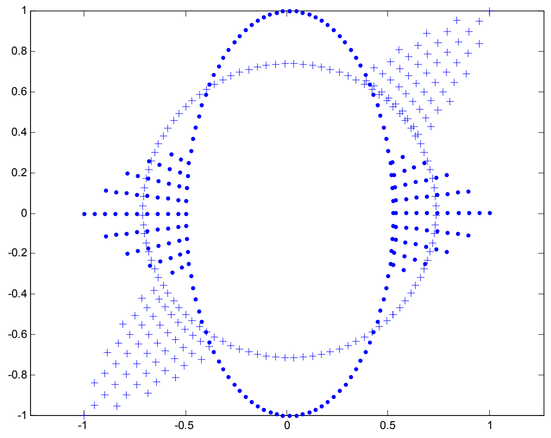

Fig. 1. A range normalized dataset (plotted with dots) chosen for its sensitivity to rotation, and that dataset rotated 45 degrees (plotted with 'x' symbols) before range normalization

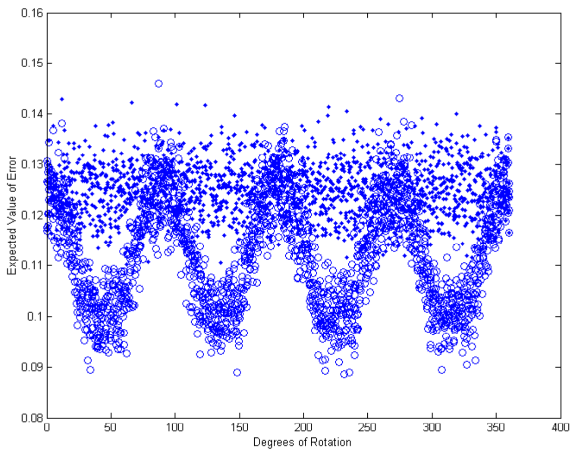

Fig. 2. Comparison of tolerance to error of rotated sensitive dataset (plotted with circles) and unrotated sensitive dataset (plotted with '.' symbols). The points were generated in pairs, with the same noise matrix applied to the rotated and unrotated dataset for each pair.

Fig. 1 shows the dataset after it has been range normalized after two treatments. The first treatment is principal component analysis, and the second treatment is principal component analysis followed by rotation by 45 degrees.

Fig. 2 shows a comparison of $T(P^T S_P^{-1}, D)$ and $T(P^T O^T S_{OP}^{-1}, D)$ for a set of 2 by 2 orthogonal rotation matrices. The points in fig. 2 were generated in pairs as follows. For each rotation angle θ sampled between 0 and 360 degrees, a noise matrix was generated whose columns were distributed as in (9). A 2 by 2 rotation matrix *O* was generated which rotated the vectors in *D* in the plane by θ degrees. Then $T(P^T S_P^{-1}, D)$ and $T(P^T O^T S_{OP}^{-1}, D)$ were evaluated and plotted for the noise matrix generated. This procedure was repeated several times for each angle, to show the variability in tolerance to error for different random noise matrices. Notice the periodic symmetry in fig. 2. In fig. 3, a different dataset is displayed

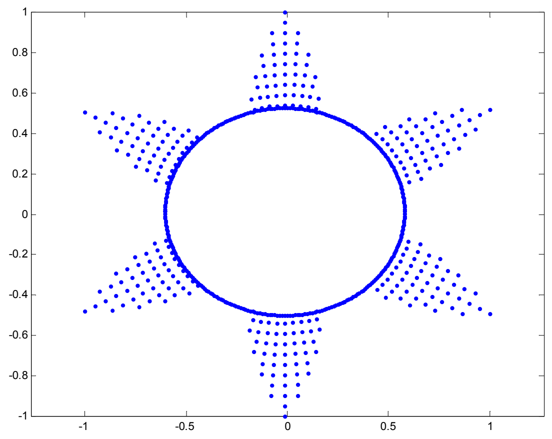

Fig. 3. A dataset chosen for its insensitivity to rotation.

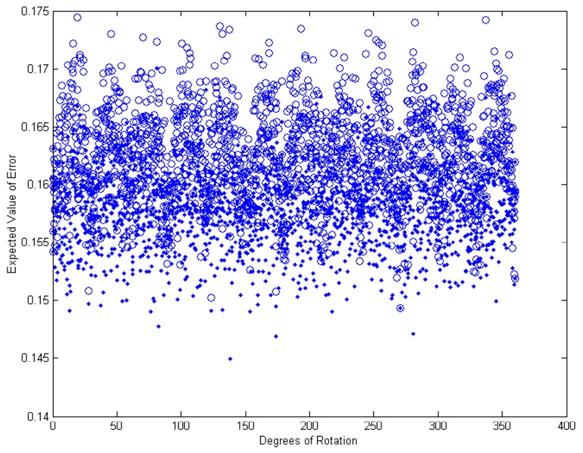

Fig. 4. Tolerance to error for dataset that is insensitive to rotation.

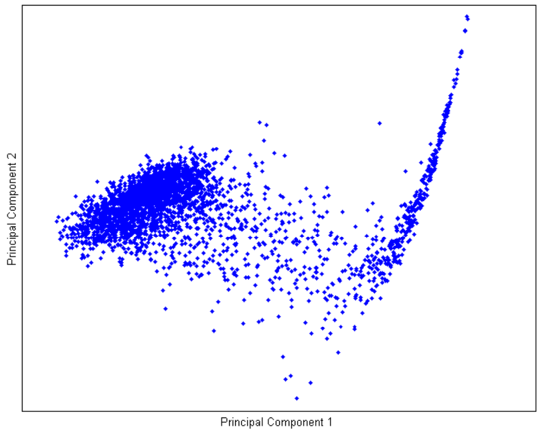

Fig. 5. First two principal components of output data.

that has been chosen to have little sensitivity to rotation. As shown in fig. 4, rotating this dataset does not increase tolerance to error, and the geometrical properties of this dataset after range scaling are not strongly influenced by rotation angle. The dataset used for experimental results below is sensitive to rotation, and rotation of principal components can be expected to improve network performance.

IV. EXPERIMENTAL RESULTS

A. Problem Domain: Predicting Sonar Reverberation

The objective for the problem domain is to train a multilayer perceptron so that it can predict the reverberation of a sonar signal in a marine environment. The reverberation time series is a measure of the intensity of sound that is reflected back to the sound source via many paths as a function of time, and is caused by an inhomogeneous marine environment. The inputs to the network are the sonar transducer parameters and a description of the salient features of the marine environment, such as the sound speed profile as a function of the depth and composition of the sea floor. The output units are decibels as a function of time, and the data has a large dynamic range. The data used for training was simulated by a physics based sonar simulation software package. Fig. 5 is a plot of the first two principal components of the data. This dataset has strong assymetry, and the tolerance to error is sensitive to rotation.

B. Comparison of Performance of Output Preprocessing Methods

In order to evaluate this method, networks were trained according to different normalization schemes and then their performance was evaluated on 10000 points of testing data. First, an explicit optimization was undertaken using genetic algorithms, and an orthonormal matrix $O_{Optimized}$ was found which had a high tolerance to error when error was simulated as in (8). Secondly, a random orthonormal matrix O_{Random} was determined by computing the singular value decomposition of a matrix whose entries were i.i.d. normal random variables with zero mean and unit variance. Principal component analysis was used for all three networks to reduce the dimensionality of the outputs from 600 elements to 50 elements; inspection of the singular values revealed that 50 principal components were sufficient to span the data with very little error. For each method, seven networks were trained starting with independent random initializations of weights. The networks were trained on 20000 datapoints for 10000 epochs. The networks all had the same topology of 30 inputs, 40 nodes in the first hidden layer, and 50 nodes in the last hidden layer. One group of networks was trained on the range scaled principal components, one group of networks was trained on the range scaled principal components after rotation by the randomly generated orthonormal matrix, and one group of networks was trained on the range scaled principal components after rotation by the optimized orthonormal matrix. Table 1 shows the 95 percent confidence interval for the mean RMS error for each group of networks for the testing set. As can be seen from Table 1, the randomly chosen matrix did the best when the neural networks were actually trained. Although it had inferior performance when the networks were trained, the optimized matrix had superior performance when simulated with (8).

Table 1. 95 percent confidence interval for mean RMS testing error of networks after training

M_L	Mean RMS Testing Error
$P^T S_P^{-1}$	(6.57,7.01)
$P^T O^T_{Random} S_{OP}^{-1}$	(5.44,5.57)
$P^T O^T_{Optimized} S_{OP}^{-1}$	(6.00,6.45)

V. CONCLUSIONS

The output data can play a crucial role in a proper choice of output matrices that transform the vector of node activations in the last hidden layer into the networks prediction of output data. Although rotation improved network performance, there are several possible reasons why the optimized matrix yielded inferior performance to the randomly chosen network. One possibility is that the genetic algorithms got stuck in one of the many local minima in the optimization surface. Another possibility is that the structure of network error must be taken into account when choosing the output matrix, and that treating error as a random variable led to an inappropriate optimization function. Nevertheless, both rotations yielded superior performance to the networks trained on the principal components, and this technique should prove useful on many datasets.

ACKNOWLEDGEMENTS

This work was supported by the Office of Naval Research. T.P.M. wishes to thank Charla Lambert for helpful comments on the manuscript.

REFERENCES

[1] I. T. Jolliffe, *Principal Component Analysis*. New York: Springer-Verlag, 1986.

[2] S. Haykin, *Neural Networks: A Comprehensive Foundation*. New York: Macmillan College Publishing Company, 1994.

[3] M. Riedmiller, and H. Braun, “A direct adaptive method for faster backpropagation learning: The RPROP algorithm,” *Proceedings of the IEEE International Conference on Neural Networks*, 1993.

[4] T. Mann, “Affine transformations and fan-out neural networks,” M.S. thesis, University of Washington, Seattle, WA, USA, 2002.

[5] G. H. Golub, C. F. Van Loan, *Matrix Computations*. Baltimore: Johns Hopkins University Press, 1989.