Visualization of Learning in Multi-layer Perceptron Networks using PCA

Marcus Gallagher, Tom Downs

Abstract—**This article is concerned with the use of scientific visualization methods for the analysis of feedforward neural networks. Inevitably, the kinds of data associated with the design and implementation of neural networks are of very high dimensionality, presenting a major challenge for visualization. A method is described using the well-known statistical technique of Principal Component Analysis. This is found to be an effective and useful method of visualizing the learning trajectories of many learning algorithms such as backpropagation, and can also be used to provide insight into the learning process and the nature of the error surface.**

Keywords—**Visualization, Multi-layer Perceptron, Error Surface, Artificial Neural Network, Principal Component Analysis.**

I. Introduction

The multi-layer perceptron (MLP) is the most widely applied and researched Artificial Neural Network (NN) model. MLP networks implement mappings from input space to output space and are normally applied to supervised learning tasks. Learning or training is achieved using iterative methods such as backpropagation to adjust the connection weights within a network. This is commonly formulated as a multivariate optimization problem over a high-dimensional space of possible weight configurations. The MLP training process is often described as a search of an *error surface* for a weight vector which gives the smallest possible error value. Iterative training produces data in the form of a trajectory in the search space. The high dimensionality of the weight space, typically ranging from tens to thousands, makes visualization of the training process very difficult.

In this article, a new approach to the visualization of learning in MLPs is described, using Principal Component Analysis (PCA). Section II provides a review of previous work in the application of visualization to neural networks. After briefly reviewing PCA in Section III, the method for weight space learning trajectory visualization is described and experimental results are presented in Section IV. Section V shows how the evolution of the principal components during learning provides additional information about the MLP training process, and Section VI gives some conclusions.

II. Neural Network Training and Visualization Methods

Research in the domain of neural networks and scientific visualization is usually concerned with one of two general goals. The focus of the first goal is on analysis of network results, i.e., understanding or interpreting the mapping produced by a trained network. This has been a long-standing interest in neural network research, because of the fundamental "black-box" property of neural networks. For examples of visualization techniques developed to assist with such understanding, see [25], [27]. This work is also invariably related to the area of rule extraction from neural networks (see, e.g. [1] for an introduction and references).

The second goal (which is the focus of this article) is to attempt to gain a more fundamental insight into the network itself using visualization and exploratory techniques. This work focuses primarily on the dynamics of learning algorithms, the structure of the error surface and its relation to the training set and also to the fixed properties of the network configuration. There are a number of other quantities that can be examined, such as the input, hidden and output unit activation spaces. By adopting this perspective we hope to gain some useful practical insights into the general mapping properties of NNs, and to be able to more fully understand and compare the behaviour of different training algorithms and procedures.

There are a number of straightforward ways of displaying multivariate data. Scatterplot techniques or Draftsman's Displays [20] have found wide application in the fields of machine learning and pattern recognition (see, e.g. [26]). Alternatively, glyphs can be used to represent data points (either in conjunction with two or three-dimensional coordinate axes, or representing all dimensions of the data through features of a composite, familiar object) [6]. Perhaps the most well-known general method of visualization using glyphs is the Chernoff Faces technique, which has previously been applied to the visualization of NNs as well as animating the backpropagation training process by updating the face display during learning (e.g. after each iteration) [29]. The well-known Hinton diagram [9] is also a visualization using glyphs (in this case, boxes representing network weight values). Hinton-style diagrams have been used to examine the mappings produced by trained networks, and further to experimentally adjust the number of hidden units to optimize performance [24] (see also [17]). The Bond diagram [30], [31] attempts to improve on the Hinton diagram by making the network topology more prominent, representing nodes (bias values) and weights in a network graphically. Some extensions and a combination of the Hinton and Bond diagrams are considered in [28].

While visualization using glyphs transforms each data point into a pictorial or geometrical symbol whose properties reflect the values of the variables of each data point, one might also consider a functional representation that maps a multivariate data set onto $\mathbb{R}^1$. Andrews' curves are the most well-known technique of this kind [12], [20]. The method is well-suited to data sets of up to about 20 dimensions, for a limited number of data points (perhaps 50). For an example application of Andrews' curves to neural network training data, see [7].

The most common attempt to visualize the error surface of a NN in the literature involves plotting ranges of two different weight values against the error function, thereby producing a surface that is actually a two dimensional "slice" of the true N-dimensional error surface. Such an approach is intuitively appealing, and its application has been described extensively in the literature (see, e.g. [10], [23]). Hush and Horne in particular use this method of visualization to describe and illustrate the general properties of error surfaces [10]. Lawrence [17] suggests a draftsman-like method of plotting many 2-D slices of the error surface, choosing two weights at random from the network for each plot. Androulakis et al. [2] adopt a slightly more sophisticated approach to examining the structure of the error surface using 2-D projections of the maximum and minimum values of the Hessian, and colouring points in this space according to convergence behaviour of the algorithm.

Wejchert and Tesauro consider a trajectory diagram (Fig. 1), which emphasizes the visualization of the learning process itself [30], [31]. They suggest representing the multidimensional coordinate system in the 2-D plane by projecting the vector components radially, component-by-component via a simple transformation. This "star-like" projection makes the axes non-orthogonal so that the points plotted are non-unique. This kind of visualization can be viewed as the network is trained, leading to a trajectory of points through weight space, followed by the learning algorithm. Although useful, the technique has limited applicability to high-dimensional data. We propose a visualization method using PCA below, which can be viewed as an adaptation of the trajectory diagram.

A number of researchers have examined different data spaces in NNs other than weight space. A commonly used concept is that of hyperplanes in the input space [22]. Visualization of hyperplane-constructed decision regions has been used in the literature to examine how a network solves a given problem [8]. An extension can be made by

This research was supported by a University of Queensland Postgraduate Research Scholarship, and a scholarship from the School of Information Technology and Electrical Engineering, University of Queensland.

M. Gallagher is a Lecturer in the School of Information Technology and Electrical Engineering, University of Queensland 4072, Australia. E-mail: marcusg@itee.uq.edu.au .

T. Downs is a Professor in the School of Information Technology and Electrical Engineering, University of Queensland 4072, Australia. E-mail: td@itee.uq.edu.au .

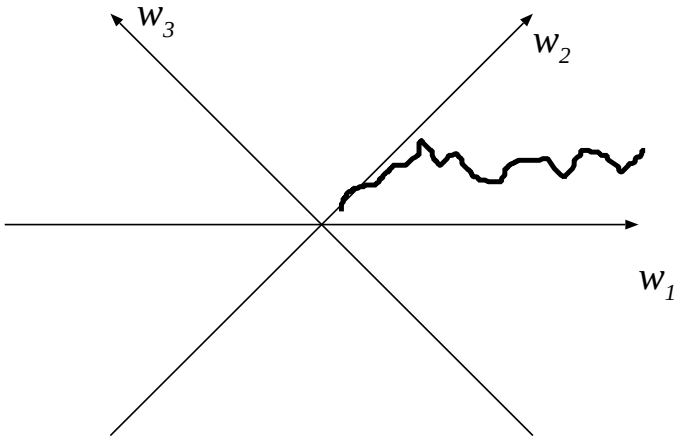

Fig. 1. Illustration of a trajectory diagram. This visualizes the path taken through weight space by a learning algorithm via a non-orthogonal projection.

viewing a sigmoidal activation function as a smoothed approximation to a threshold function, and plotting the input space shaded with varying levels of grey to represent output values between 0 (black) and 1 (white). Such displays are sometimes called response function plots [5]. The challenging "Two-Spirals" learning task [16] is a problem to which this visualization technique is often applied. It is also possible to use hyperplanes and response function plots to visualize the "decision regions" of the space of hidden layer outputs [21]. In this way it can be seen how a hidden layer can transform a given problem into a linearly separable one, which is then solved by the output layer of weights. This method has been used in the analysis of *N-2-N* encoder networks [19], [21]. The utility of visualizing response function plots for both output and hidden layer units is clearly demonstrated in [13], where gaps in the response of the network are shown to be directly related to generalization performance.

It is also worth noting that a number of neural network software packages and simulators are at least partly concerned with visualization, for example the Neurograph [32] and Neuralis [14] packages. Liao and Moody [18] describe a recent software toolkit in which the focus is on the visual interpretation of a previously trained network and the corresponding training data set. The tools may be used to assess the importance of inputs to the task at hand interactively, to select the important features for training and to prune hidden units to improve training performance.

While a number of useful approaches have been developed, the visualization of neural networks is an area in which a relatively small amount of work has been done. Significant advancements in computer technology (especially graphical capabilities, processing speed and storage capacity) have created an environment to support the development of new methods of visualizing the high-dimensional data associated with NNs. NN weight data is typically of very high dimensionality, and most current methods proposed in this area scale awkwardly to networks with more than a few weights. In the next section, a method of visualization is developed using principal component analysis, which is more robust in this respect and remains flexible enough to be used in a general MLP training environment.

III. Principal Component Analysis

Principal Component Analysis (PCA) (or the Karhunen-Loève expansion) is a technique widely used in the statistical community, primarily for descriptive but also for inferential purposes [11], [12]. In this section we briefly review PCA, and by doing so introduce our notation.

The technique is often used to create a projection of multivariate data onto a space of lower dimensionality, for visualization purposes, whilst attempting to preserve as much of the structural nature of the data as possible [12]. If the data can be well represented in a two or three-dimensional subspace (which is captured by the first two or three principal components), it is convenient to directly plot this projection on a computer screen. Further, since many existing techniques for multivariate data visualization have proven effective for data of dimensionality up to about 10 [15], if the data can be effectively reduced to this kind of dimensionality, these techniques could be used following PCA.

Let $\mathbf{w} = (w_1, \ldots, w_N)$ be an N-dimensional weight vector containing all the weights in a network, $\mathbf{w} \in \mathbb{R}^N$. The weights are collected sequentially into this vector, from an input unit to each hidden unit in turn and from a hidden unit to each output (assuming a bias unit after the last normal node in the input and hidden layers). The training process produces a trajectory (dataset) of points $\mathbf{w}^j, j = 1 \ldots s$ in weight space, on which PCA can be performed. This can be achieved firstly by calculating the $N \times N$ covariance matrix $\mathbf{\Sigma}$ of this trajectory

$$\mathbf{\Sigma} = \frac{1}{s} \sum_{j=1}^{s} (\mathbf{w}^s - \overline{\mathbf{w}})(\mathbf{w}^s - \overline{\mathbf{w}})^T \tag{1}$$

where $\overline{\mathbf{w}}$ is the mean $\overline{\mathbf{w}} = \frac{1}{s}\sum_{j=1}^{s} \mathbf{w}^j$. The eigensystem of $\mathbf{\Sigma}$ is then found, providing a new set of basis vectors and coordinate system (ie, a rotation of weight space). To reduce the dimensionality of the weight space to some value $d < N$, the d largest eigenvalues and corresponding eigenvectors are chosen and the remaining $N - d$ values discarded. Performing this dimensionality reduction results in the smallest possible loss of (variance) information, for any other similarly defined choice of basis vectors [12]. The direction of greatest variance in the data is given by the first principal component (eigenvector) (PC) $\mathbf{v}_i$, corresponding to the largest principal value (eigenvalue) (PV) λ_i. Note that PCA can be conducted using more computationally efficient methods, such as the Singular Value Decomposition (SVD) [12].

In practice, PCA is quite often performed on the correlation matrix rather than the covariance matrix [12]. The main reason for this is that if the variables in question are measured in different units, it may be that the variances of some variables dominate the first few PCs, simply because of the units they are measured in (e.g. consider one variable measured in meters together with some other variables measured in millimeters). In the case of neural network weight data, this problem does not arise - the covariance matrix is therefore used for PCA in the following sections.

IV. Weight Space Learning Trajectory Visualization

This section presents a number of results of using PCA for weight space learning trajectory visualization. In many problem instances, PCA is conducted on samples where each data point is independently drawn from some unknown distribution. Indeed, many of the algebraic properties of the technique rely on this assumption (together with the assumption that the sampling distribution is multivariate Normal). However for the visualization of weight space data the assumption clearly does not hold - the data can be considered as a time series of observations of the weight vector evolving in discrete time steps (iterations of the learning algorithm). PCA remains a useful exploratory method for such data [12], as the formal properties are primarily directed towards the use of PCA for inference.

Before we examine the benefits of using PCA for visualization of learning trajectories, we will first establish the viability of the approach by demonstrating the dimensionality reductions that can be achieved. We do this by investigating three variations on an artificial learning task and two more practical learning situations.

1. **The student-teacher learning model.**

This is an artificial learning scenario, in which two networks are created. The weights of one network, the teacher, are initialized in some

Network (Dataset)	PC1	PC2	PC3
2-2-1 (student-teacher)	93.57 (6.16)	5.08 (5.28)	1.01 (0.94)
4-4-1 (student-teacher)	89.46 (7.18)	7.21 (5.48)	1.68 (1.32)
8-8-1 (student-teacher)	80.59 (7.62)	8.47 (3.46)	3.05 (1.37)
51-15-2 (credit card)	96.57 (0.78)	2.34 (0.49)	0.56 (0.20)
9-5-2 (cancer)	89.74 (2.17)	7.28 (1.23)	2.31 (1.24)

TABLE I

PERCENTAGE OF VARIANCE CAPTURED BY THE FIRST THREE PCS.

way, and this represents the "oracle" for the learning task. The training set is produced by generating random input vectors from which the teacher network produces the target output vectors. The other network, the student, is then trained using these input/output pairs.

Three network configurations were chosen for the student-teacher scenario. These were 2-2-1, 4-4-1 and 8-8-1, which correspond to N = 9, 25 and 81 weights respectively (including bias weights). For each configuration, 10 randomly generated problems were used. In addition, 10 separate training runs were conducted for each problem, commencing from a different (randomly generated) weight initialization (range [-0.25,0.25]). The random problems were produced by creating a teacher network of identical topology to the learner, with weights generated from an $\mathcal{N}(0,1)$ distribution. Thus a total of 300 training sessions were conducted. Standard backpropagation was used [3], with learning rate $\eta = 0.1$. A training set of 10^4 input patterns was randomly generated for each of the 10 problems (each element of each input pattern being drawn independently from a $\mathcal{N}(0,1)$ distribution), and the target outputs for each pattern were produced using the teacher network. Weights were updated in stochastic mode (i.e. after each pattern presentation), using examples chosen at random from the training set. Note that this "student-teacher" model of learning guarantees the presence of several global minima (ie, points which are functionally equivalent to the teacher weight configuration) and which therefore give zero error[1]. The model is especially useful in an experimental setting because several parameters of the learning problem (i.e. the number of nodes in each network layer (as done here) and the number of training patterns) can be systematically varied with ease. Each network was trained for a total of 10^4 iterations, and the values of the weights were recorded after every tenth iteration.

2. **Two more practical learning problems.**

These two datasets are benchmark problems held in the UCI machine learning repository [4]. The first learning task is credit card approval on the basis of 51 real inputs. The data set assumes two boolean outputs. A 51-15-2 network was used for this problem (a total of 812 weights, including bias weights) and the credit card approval dataset contained 690 patterns.

The second dataset is the Wisconsin breast cancer database which contains 699 patterns, each consisting of 9 real inputs and two boolean output variables. The outputs represent a classification of benign or malignant on the basis of the inputs. For this problem a 9-5-2 network was chosen (62 weights).

For both problems, the entire dataset was used to train the network, and each network was trained 10 times from a random initialization, using standard backpropagation with learning rate $\eta = 0.25$. Backpropagation with stochastic weight updating was used. Each network was trained for 10^5 pattern presentations, and the weights were recorded after every 20 pattern presentations.

It is clear that the effectiveness of PCA as a visualization technique will be maximized when the majority of the variance of the data in question can be captured by the first two or three principal components. Table I shows the network topology, dataset, and the average percentage of variance captured by each of the first three PCs of the points in weight space along the learning trajectory. Each PC value was averaged over the 10 training runs. The standard deviation for each percentage is shown in brackets. Clearly, the majority of the variance in the weight data is captured by the first PC. This result suggests that whilst training is a search across an N-dimensional error surface, the *intrinsic dimensionality* of the path taken by back-propagation is in fact much smaller than N. We can therefore observe the learning path along just the first few PCs (in fact, usually only the first one or two), without any significant loss of information. Note that for the student-teacher problems, the average amount of variance captured by the first PC tends to decrease as the number of weights increases while in contrast, the credit card problem trajectories show the highest level of variance captured by PC1, despite also having the largest network size.

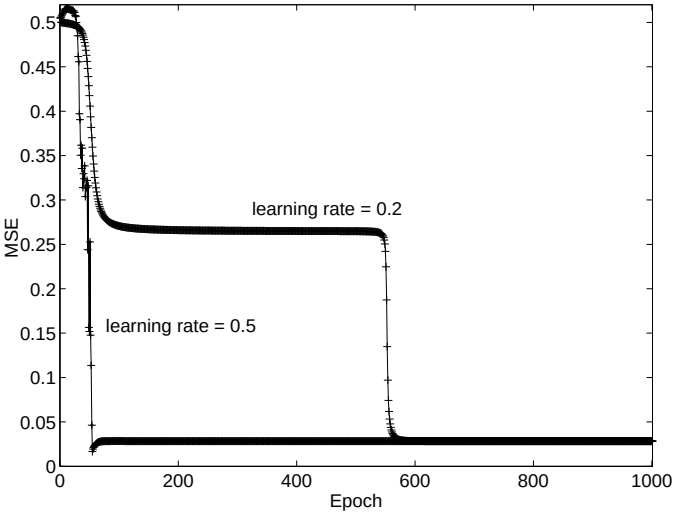

Fig. 2. Training curves for the XOR problem, with two different learning rates.

To demonstrate the visualization method we use the XOR training problem which, although relatively trivial, often creates difficulties for backpropagation and in such cases displays interesting learning behaviour. The network we employ is 2-2-1 so there are 9 weights, offering ample opportunity for dimensionality reduction. The learning behaviour exhibited by XOR can also be observed in other datasets (e.g, the five datasets above) depending on the tuning of parameters, initial weight values, etc.

Fig. 2 shows the familiar graph of MSE as a function of training epochs for two networks that are started from the same initial point in weight space. The only difference between the training instances is the value of the learning rate parameter, η (0.2 and 0.5) used. Although the curves show a decrease in error over time, it is tempting to interpret features of the curve in terms of the progress of training through weight space. Fig. 3 plots MSE against the first principal component of the same training instances in Fig. 2. In the $\eta = 0.2$ curve, PC1 captures 71.4% of the variance in the data, and in the $\eta = 0.5$ curve, PC1 captures 94.2% of the variance in the data. Fig. 3 provides a much more accurate representation of the progression in weight space than does Fig. 2. Specifically, distances between data points in Fig. 3 represent the actual distances between the points, along that direction in weight space. In contrast, points in Fig. 3 are made equidistant along the x-axis. Fig. 2 is therefore a scaled[2] version of Fig. 3. Plotting point values over a curve also gives some idea of the "velocity" of the trajectory (in terms of the time spent on that region of the error surface). The effect of an increased learning rate is observable from Fig. 2 (and one can "zoom in" around the point where error drops rapidly)

[1] Symmetry guarantees the presence of $2^{N_h} N_h!$ equivalent points on the error surface (i.e. weight vectors) each corresponding to a global minimum [3].

[2] Note also that the data in Fig. 2 must strictly increase along the x-axis, unlike Fig. 3 where training might move in either direction along PC1, as seen in this figure for the $\eta = 0.5$ curve.

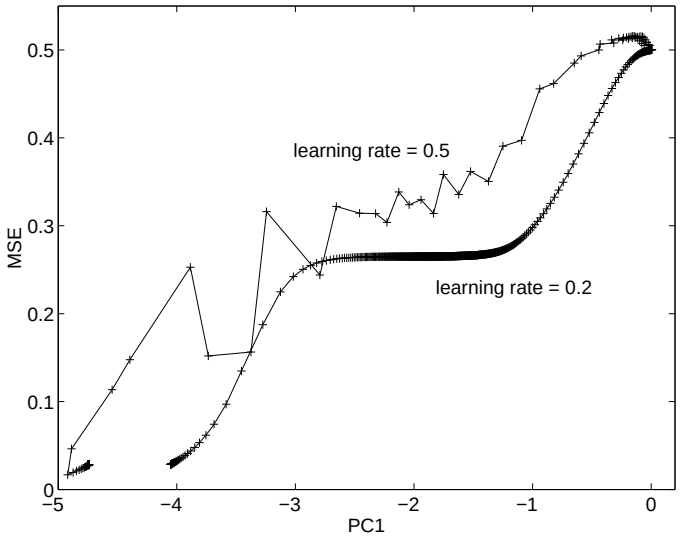

Fig. 3. Learning trajectories for the XOR problem against the first principal component with respect to the data, for two different learning rates.

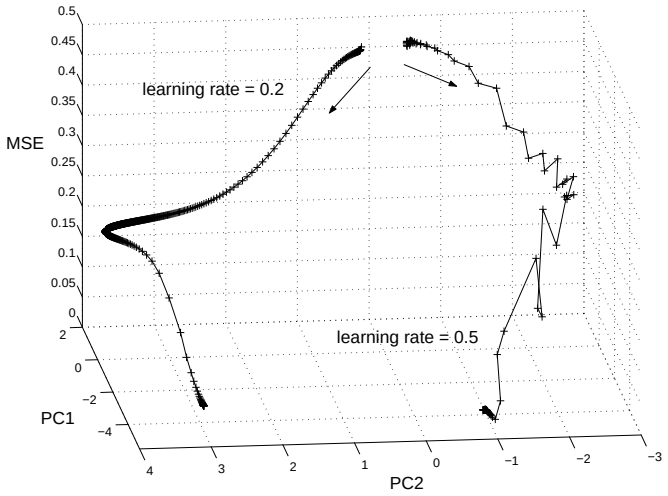

Fig. 4. Learning trajectories for the XOR problem against the first principal component with respect to the data, for two different learning rates.

but Fig. 3 provides a better visualization of this, again because of the above-mentioned scaling effect.

Fig. 4 extends Fig. 3 by plotting MSE against the first two principal components for the same two learning trajectories. For the $\eta = 0.2$ curve, PC2 accounts for 28.1% of the data variance, while for the $\eta = 0.5$ curve, PC2 accounts for 4.5% of data variance. By observing the trajectory in three dimensions the user is able to appreciate some of the geometry of the multidimensional weight space, subject to the limitation of PCA. For both trajectories, Fig. 4 reveals a slow movement in one direction along PC2 (increasing for $\eta = 0.2$, decreasing for $\eta = 0.5$), and then in the opposite direction. It is important to note that although we show the trajectories in Fig. 3 and Fig. 4 on the same axes, the principal components for each trajectory are not the same, since each set of PCs is calculated from its respective dataset. This means that the trajectories should only be compared *qualitatively*; we do not observe the trajectories moving in the same projection of weight space.

Training with stochastic backpropagation tends to introduce further "noise" into the learning trajectory, where successive upward movements on the error surface become possible. Fig. 5 shows a stochastic

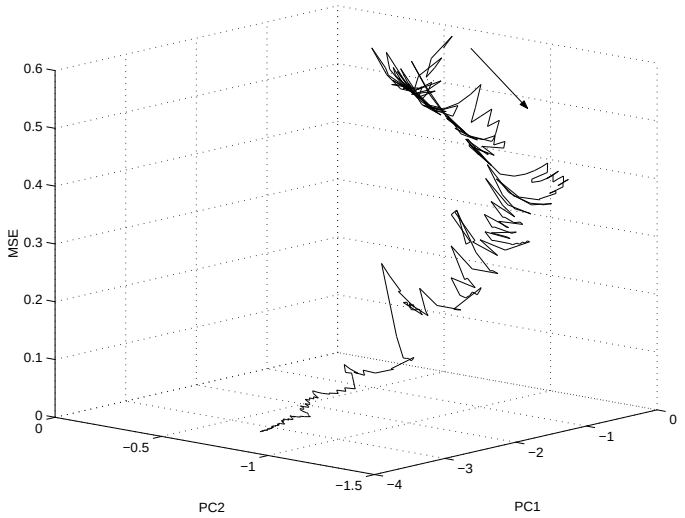

Fig. 5. Learning trajectories for the XOR problem against the first principal components for stochastic backpropagation.

backpropagation learning trajectory with $\eta = 0.1$, starting from the same initial point as the trajectories in Figs. 3 and 4. In this case, the amounts of variance represented by PC1 and PC2 are 96.5% and 2.2% respectively. The trajectory is seen to move with a significant amount of oscillation or "sloshing" across a rain-gutter like ravine, while making gradual progress towards a region of low error. Visualization of this trajectory using principal components is seen to provide an effective representation of its complex behaviour on the error surface.

V. Temporal Principal Component Behaviour

Since our data is a learning algorithm trajectory, it is of interest to study the evolution of the PCs and PVs as learning proceeds. The first question that might be asked is, how does the amount of variance captured by the first, second, ..., PCs change as learning progresses? This question can be answered by looking at the evolution of the PVs as learning proceeds.

In this section the trajectories produced by training on the cancer classification problem are used to illustrate the temporal PC visualization process (results produced for the credit card problem were similar). Fig. 6 is a plot of the percentage of variance captured by a PC as a function of pattern presentations. At any stage of training (along the x-axis), principal components must be recalculated. These curves are therefore produced by sampling the weight data at k equally spaced points (here, k=10 is chosen) in the data, and PCA is performed on that portion of the trajectory, $\mathbf{R}^* = \{\mathbf{w}^1, \ldots, \mathbf{w}^j\},\ j = \frac{s}{k}, \frac{2s}{k}, \ldots, s$. Curves are shown for different PVs corresponding to the PCs indicated. The plot reveals how much variance the first three PC's capture as training proceeds. As can be seen, the amount of variance captured by each PC does not change to a large extent, irrespective of how much training has advanced. When the amount of variance captured by the first PC decreases, this amount is largely regained in the second PC. The curves are averages of all training runs conducted on the cancer dataset, and error bars indicate the standard deviations involved with each point on the curves. The final points on the ends of the curves therefore show the average success of PCA in capturing the variance over all the training processes. As mentioned above, performing PCA on a subset of the complete trajectory leads to a different set of PCs, because a different set of weight data is being used.

Although Fig. 6 indicates that the PVs (and hence the percentage of variance captured by the corresponding PCs) are similar throughout training, it is of further interest to investigate the evolution of the PCs themselves. For a given PC, the coefficients $b_1, \ldots, b_n$ indicate how

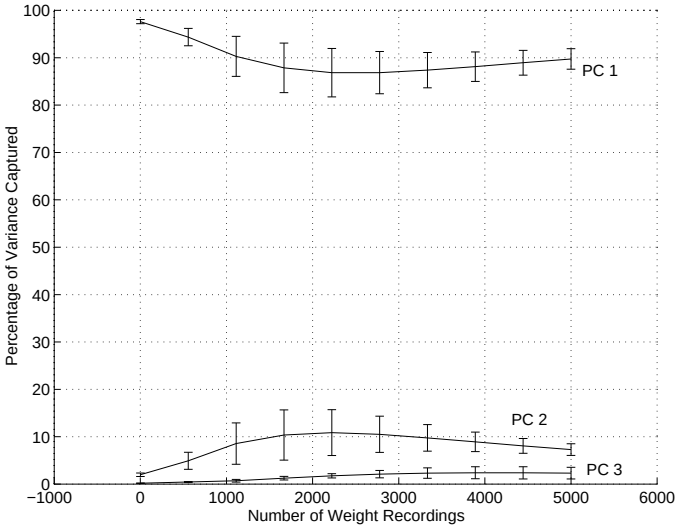

Fig. 6. Change in the amount of variance captured by the PCs over the learning process.

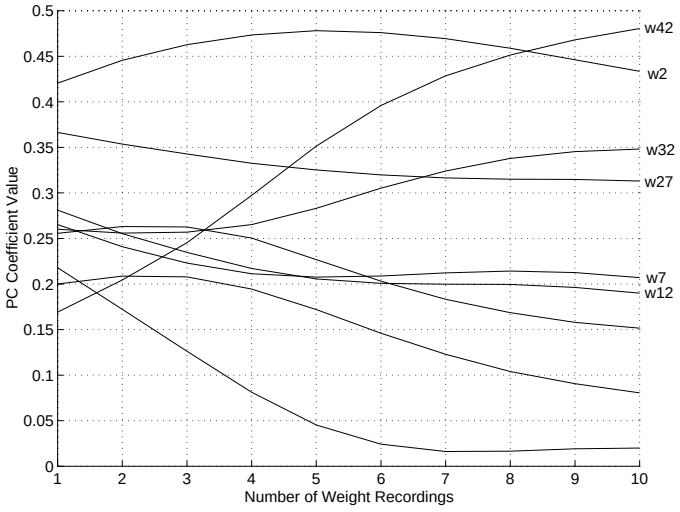

Fig. 7. Values of the coefficients of the first PC over the learning process.

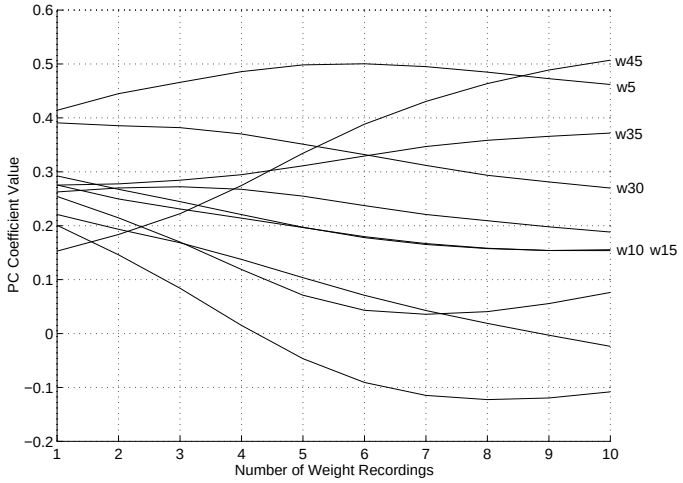

Fig. 8. Values of the coefficients of the first PC over a different learning process.

"close" the direction of that PC in weight space is to each weight axis. The vectors of coefficients are the eigenvectors of the covariance matrix $\mathbf{\Sigma}$, hence $\|(b_1, \ldots, b_n)\| = 1$. If a PC v_i captures most of the variance in the trajectory, a corresponding b_i of large magnitude indicates that the direction followed by the trajectory is close, in weight space, to weight axis w_i, and it follows that the weight has made a large contribution to the direction of the trajectory. The sign of the coefficients within a PC provides an obvious indication of the direction of variance of the coefficients in relation to each other (e.g. if b_i = 0.7 and b_j = -0.65, then the trajectory moves in a direction close to increasing w_i and decreasing w_j). Between PCs, the sign of the coefficients is unimportant, as reversing the sign on all coefficients of a given PC leaves the variance captured by that PC unchanged, as well as its orthogonality with the other PCs.

Fig. 7 shows the evolution of the coefficients $(b_1, \ldots, b_{62})$ of the first PC against pattern presentations for a single training run conducted on the cancer dataset. Of the 62 coefficients for this trajectory, only those coefficients whose absolute value exceeded 0.2 at one or more of the sample points are shown (9 in this case). This shows how the direction of the first PC changes as learning progresses, in terms of the weights which produced the greatest variance. The fact that many of the coefficients remain small allows us to determine which of the weights contributed most to the trajectory. In examining the contributions of weights to the 10 different trajectories produced on the cancer dataset, it was found that the six weights labeled in Fig. 7 ($w_2, w_7, w_{12}, w_{27}, w_{32}$ and w_{42}) were also among the most significant for a second trajectory. Another group of six weights ($w_5, w_{10}, w_{15}, w_{30}, w_{35}$ and w_{45}) were prominent in three of the other trajectories produced (one of which is shown in Fig. 8). If we examine these groups in relation to the structure of the network, we find that, in each case, the six weights are *from the same inputs* to a single hidden unit in the network. Comparing Fig. 7 and Fig. 8 reveals a similar pattern of evolution for the six weights, suggesting that the trajectories are symmetrically related. In each case the values of the weights connected to different hidden units have evolved similarly. This can be related to the known symmetry which exists in weight space. On an intuitive level, particular hidden units have associated themselves with particular inputs, and this happens often for different trajectories started from different initialization points. Further analysis may reveal other similar, less-obvious relationships that may provide insight into the learning process.

VI. Conclusion

This paper demonstrates how the technique of PCA can be used for visualization of the learning process in MLPs. This technique allows visualization of the trajectory followed during learning in networks of practical size, where the dimensionality of the search space is often very high. The results suggest that learning trajectories such as those produced by backpropagation can be represented in a low dimensional subspace to a high degree of accuracy. It is also suggested that this visualization method would be useful to qualitatively compare training algorithms, including the settings of algorithm parameters), or in application where such parameters are commonly tweaked by trail and error to optimize results. The technique seems to provide an informative and useful representation of the behaviour of learning trajectories, particularly compared to simply examining error curves as a function of training iterations.

The temporal evolution of the PCs and PVs has also been explored. This reveals information about the dynamics of the path taken through weight space by the given training algorithm. It is also possible to observe the evolution of the mapping produced by the network, and to interpret different aspects of this process (e.g. the role which is played by a particular hidden unit).

PCA has a number of advantages as a visualization technique for MLPs. Its application is independent of the learning algorithm used, the training set, and the various adjustable parameters of the learning process, as it processes the data that results from the learning process. PCA can also be performed in a computationally efficient manner and

hence it scales up with the size of the problem under consideration. Data is also inexpensive to collect, being generated automatically as part of training without requiring additional information (i.e. data are just points in the weight space). Avenues for future work include combining the use of PCA with other multivariate visualization techniques (e.g, glyphs) and constructing an animated display of learning.

REFERENCES

[1] R. Andrews, J. Diederich, and A. B. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995.

[2] G. S. Androulakis, G. D. Magoulas, and M. N. Vrahatis. Geometry of learning: Visualizing the performance of neural network supervised training methods. *Nonlinear Analysis, Theory, Methods & Applications*, 30(7):4359–4544, 1997.

[3] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, 1995.

[4] C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases. Retrieved from http://www.ics.uci.edu/∼mlearn/MLRepository.html (16/11/01), 1998.

[5] M. W. Craven and J. W. Shavlik. Visualizing learning and computation in artificial neural networks. Technical Report 91-5, University of Wisconsin Computer Sciences Department, 1991.

[6] B. S. Everitt. *Graphical Techniques for Multivariate Data*. Heinemann Educational Books, London, 1978.

[7] M. Gallagher. *Multi-layer perceptron error surfaces: visualization, structure and modelling*. PhD thesis, Dept. Computer Science and Electrical Engineering, University of Queensland, 2000.

[8] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA, 1991.

[9] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Distributed representations. In *Parallel Distributed Processing*, volume 1, chapter 3, pages 77–109. MIT Press, Cambridge, MA, 1986.

[10] D. R. Hush, B. Horne, and J. M. Salas. Error surfaces for multi-layer perceptrons. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):1152–1161, 1992.

[11] E. J. Jackson. *A User's Guide to Principal Components*. Wiley, New York, 1991.

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

[13] M. M. Kantardzic, A. A. Aly, and A. S. Elmaghraby. Visualization of neural-network gaps based on error analysis. *IEEE Transactions on Neural Networks*, 10(2):419–426, 1999.

[14] S. Khuri and J. Williams. Neuralis: an artificial neural network package. *SIGCSE-Bulletin*, 28:25–27, 1996.

[15] W. J. Krzanowski. *Principles of Multivariate Analysis: A User's Perspective*. Clarendon Press, Oxford, 1988.

[16] K. J. Lang and M. J. Witbrock. Learning to tell two spirals apart. In T. Sejnowski D. Touretzky, G. Hinton, editor, *Proceedings of the 1988 Connectionist Models Summer School*, pages 52–59, San Mateo, CA, 1988. Morgan Kaufmann.

[17] S. R. Lawrence. *Neural Networks For Real World Tasks: Limitations and Solutions*. PhD thesis, University of Queensland, 1997.

[18] Y. Liao and J. Moody. A neural network visualization and sensitivity analysis toolkit. In S. Amari et. al., editor, *International Conference on Neural Information Processing (ICONIP'96)*, pages 1069–1074, Berlin, New York, 1996. Springer.

[19] R. Lister. Visualizing weight dynamics in the N-2-N encoder. In *IEEE International Conference on Neural Networks*, volume 2, pages 684–689, Piscataway, NJ, 1993. IEEE.

[20] B. F. J. Manly. *Multivariate Statistical Methods: A Primer*. Chapman and Hall, London, 1994.

[21] P. W. Munro. Visualizations of 2-D hidden unit space. In *International Joint Conference on Neural Networks*, volume III, pages 468–473, New York, 1992. IEEE.

[22] N. J. Nilsson. *Learning Machines*. McGraw-Hill, New York, 1965.

[23] S. Phillips. The effect of representation on error surface. In P. Leong and M. Jabri, editors, *Fourth Australian Conference on Neural Networks (ACNN'93)*, pages 86–89, Australia, 1993. University of Sydney.

[24] A. J. Pinz and H. Bischof. Constructing a neural network for the interpretation of the species of trees in aerial photographs. In *10th International Conference on Pattern Recognition*, volume 1, pages 755–757, Los Alamitos, CA, 1990. IEEE Comp. Soc. Press.

[25] T. Plate, J. Bert, J. Grace, and P. Band. Visualizing the function computed by a feedforward neural network. *Neural Computation*, 12(6):1337–1354, 2000.

[26] B. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, UK, 1996.

[27] R. Rojas. Visualizing the learning process for neural networks. In M. Verleysen, editor, *European Symposium on Artificial Neural Networks (ESANN'94)*, pages 211–216, Brussels, Belgium, 1994. D Facto.

[28] M. J. Streeter, M. O. Ward, and S. A. Alvarez. N^2VIS: An interactive visualization tool for neural networks. In *Visual Data Exploration and Analysis VIII*, volume 4302 of *Proc. SPIE*, San Jose, CA, 2001.

[29] E. van der Poel and I. Cloete. Animating neural network training. *South African Computer Journal*, (7):44–52, 1992.

[30] J. Wejchert and G. Tesauro. Neural network visualization. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 465–472, San Mateo, CA, 1990. Morgan Kauffman.

[31] J. Wejchert and G. Tesauro. Visualizing processes in neural networks. *IBM Journal of Research and Development*, 35(1/2):244–253, 1991.

[32] P. Wilke. Visualization of neural networks using NEUROGRAPH. *University Education Uses of Visualization in Scientific Computing*, A-48:105–117, 1994.