Abstract—The back propagation training algorithm, used to train non-linear feed forward multi-layer artificial neural networks, is capable of estimating the error present in the data presented to a network. While of no use during the training of a network, such information can be useful after training to permit the input data to be itself adjusted to better fit the internal model of a trained neural network. After this has been done, the difference between the modified and original data can be useful. This paper discusses how such data adjusting may be done, demonstrates the results for two simple data sets and suggests some uses that may be made of such differences.

I. INTRODUCTION

In the 50 or so years since artificial neural networks (ANN) were first developed they have proved very successful at building models of input/output relationships from a series of examples of inputs and corresponding outputs. One common form of an ANN is the multi-layer non-linear feed forward network, commonly called either a “back propagation” or a “multi-layer perceptron” (MLP) network [1]. A more extensive description of these networks will be found, for example, in [2]. In these networks a training algorithm [3,4] is used that adjusts the weights in the network by a sequence of corrections that should be made to each weight.

During this process the error at the inputs can also be estimated, and reflects a measure of the mismatch between the input values currently presented and those the current internal model would have expected for the given output(s). During network training input errors are not normally calculated as they are cannot be used in any way. However, once a network has been trained, we propose that calculating the input errors may be beneficial when new (test) data is being presented to a network. While of no use during the training of a network, such information can be useful after training to permit the input data to be itself adjusted to better fit the internal model of a trained neural network. After this has been done, the difference between the modified and original data can be useful. This paper discusses how such data adjusting may be done, demonstrates the results for two simple data sets and suggests some uses that may be made of such differences.

II. THE NETWORK AND DATA TRAINING ALGORITHMS

Consider a non-linear, multi layer feed forward ANN, commonly called either a back propagation network or MLP. The following terminology will be used in the formulae that describe the recall and training of this network.

A superscript L will be used to identify a network layer, where L=0 is the input layer and layers are numbered in ascending order towards the output. The layer L+1 is the layer above L and L-1 is the layer below L.

Within a layer the neurons are sequentially numbered from L1 (the leftmost neuron in layer L) to LmaxN, where maxN is the number of neurons in that layer. Thus the inputs to a network would be numbered sequentially from 01 to 0maxN. Let X_L^i be the output of the ith neuron in layer L and I_L^i be the internal activation of the ith neuron in layer L.

We use two subscripts to define a weight: the first to identify the neuron (or input) providing input to this weight, the other the neuron that this weight supplies information to. Thus W_L^i is the weight joining the ith neuron in layer L to the jth neuron in layer L+1. W_L is the weight joining the fixed bias input to the ith neuron in the Lth layer. This fixed bias input is normally set to one, although any values (other than zero) may, in principle, be used.

Inputs (Layer 0)

Layer 1

Layer 2

Fig. 1. Part of a three layer ANN, with neurons shown as circles with the neuron number inside and such weights as are shown being shown as heavy lines.

This connectivity can be seen in figure 1. Let f be the output transform (e.g. tanh or sigmoid). Three discrete steps can be used to describe the traditional training and use of this
network. A fourth is introduced here to permit the adjustment of the input data to a trained network.

**Step 1. The forward pass to calculate the network outputs**

The output $X^L_{i}$ from the $i^{th}$ neuron in the $L^{th}$ layer is given by:

$$X^L_{i} = f(t^L_{i}) = f\left(\sum_{j=1}^{(L-1)\max N} W_{ij}^{L-1} X_{j}^{L-1} + W_{i0}^L\right) \quad (1)$$

To obtain the overall network output repeatedly apply Equation 1, starting with neuron 1 and then proceeding in order to neuron 1 max N. After the outputs from all neurons in layer one are calculated, then the outputs from all the neurons in layer two are calculated, and so on until the outputs from all the neurons in the top layer (the output layer) are calculated.

**Step 2. Assigning the responsibility for the output error amongst all the neurons**

Let us assume that there is some global error function $E$ that is everywhere a differentiable function of the weights in the network. A suitable definition of the total error at the outputs of the network is

$$E = 0.5 \sum_{j=1}^{(L-1)\max N} (d^L_{j} - o^L_{j})^2$$

where $d^L_{j}$ is the desired output from the $j^{th}$ output neuron and $o^L_{j}$ is the actual output from that neuron and the factor of 0.5 is only introduced to simplify the expression for the error at an individual output neuron. The 'local' error at a particular neuron $k$ is thus:

$$e^L_{k} = -\delta E/\delta X^L_{k} = -\delta E/\delta o^L_{k} \times \delta o^L_{k}/\delta t^L_{k} = (d^L_{k} - o^L_{k}) \times f'(t^L_{k}) \quad (2)$$

Then the error in the output of the $i^{th}$ neuron in the $L^{th}$ layer is:

$$e^L_{i} = \delta E/\delta t^L_{i} \quad (3)$$

Similarly the error in the output of the $i^{th}$ neuron in the $L+1^{th}$ layer is:

$$e^{L+1}_{i} = -\delta E/\delta t^{L+1}_{i} \quad (4)$$

Equation 1 can provide the relationship between $t^L_{i}$ and $t^{L+1}_{i}$. This relationship, together with Equations 3 and 4, gives a relationship between $e^L_{i}$ and $e^{L+1}_{i}$.

$$e^L_{i} = f'(t^L_{i}) \times \sum_{k=1}^{(L+1)\max N} (e^{L+1}_{k} \times W_{ik}^{L+1}) \quad (5)$$

Equation 5 expresses the error in a neuron in layer $L$ in terms of the errors at all the neurons in layer $L+1$ and the weights connecting each neuron in layer $L+1$ to the neurons in layer $L$. Thus, provided we know the error at the outputs, repeated application of Equation 4 will allow us to calculate the errors at all the neurons in the network other than the output neurons.

**Step 3 Adjusting weights to reduce the network output error**

Once the error at each neuron has been estimated the values of the weights connecting the neurons are adjusted so as to decrease the contribution to the network output error by some learning coefficient $lcoeff$. The value of $lcoeff$ used is often layer dependent, being larger the closer the layer is to the inputs so as to compensate for the attenuation in the error signal as it flows back from the outputs towards the inputs. The change made to each weight is given by

$$\Delta W^L_{ij} = -lcoeff \times \frac{\delta E}{\delta W^L_{ij}} \quad (6)$$

As $\frac{\delta E}{\delta W^L_{ij}} = \frac{\delta E}{\delta t^L_{i}} \times \frac{\delta t^L_{i}}{\delta W^L_{ij}} = -e^L_{i} \times X^{L-1}_{j}$ this reduces to:

$$\Delta W^L_{ij} = lcoeff \times e^L_{i} \times X^{L-1}_{j} \quad (6)$$

Equation 6 is applied repeatedly to calculate the change in each weight in the network.

The normal training of a network from each example involves setting the inputs and repeatedly applying Equation 1 until the network output is known. Then the output error at each network output is determined using Equations 2 and 4 and applied repeatedly to assign the error estimates throughout the network. Finally Equation 6 is used on each weight to calculate the changes that should be made to each weight value.

**Step 4. Adjusting the test data to better suit the fully trained network’s internal model**

The repeated application of Equation 5 during Step 2 can propagate an estimate of the error right back to the network inputs, although this is not normally done as such information is not normally useful during network training. However, once a network has been trained, Steps 1 and 2 can be used to estimate the input errors for an example. Here the input error should be interpreted as a measure of how closely the expected output values fit those expected by the internal model that the network learned during training in order to produce the expected output. The use of such an initial error estimate is limited but can be improved by keeping the network unchanged but now adjusting the example to better suit the internal model: that is to reduce the input error estimate to be below some user chosen threshold. Once the example has been adjusted to suit the internal model a comparison between it and the original example can be used as a better estimate of the errors existing in the original example (as far as the network’s internal model is concerned). The example is adjusted to suit the model by applying a fraction of the error estimate obtained from Step 2 to the data as shown in Equation 7. The output is then recalculated (Step 1) and the input errors re-estimated (Step 2) and this cycle is repeated until all input error estimates are below some user specified threshold.

$$X^0_{i} = X^0_{i} + lcoeff^0 \times e^0_{i} \quad (7)$$

The most meaningful estimate of the error in an input is the difference between the original and final values of the input $X^0_{i}$. The value used for $lcoeff^0$ in Equation 7 is not necessarily the same as any of the values used in Equation 5.
The intent of Equation 7 is to perform a gradient descent down the input data error surface to find the closest point that meets the desired output and has no error value greater than the threshold. If \( \text{lcoeff} \) in Equation 7 is too high it is quite possible that the initial corrections made may be so large as to focus training towards a more distant point that also meets the desired input error requirement.

While artificial neural networks can interpolate, they are generally poor at extrapolation. Unconditional application of Equation 7 can result in values for \( X^j \) that are outside the range of values on which the network was trained. Under such a condition the resulting adjusted data values may be of little worth as the data may have been adjusted to a meaningless position outside the networks range of expertise. To prevent this, the possible values that may be obtained from Equation 7 should be clipped to a range of values on which the network was trained. Since all inputs are normally scaled to suit the range of the neurone transfer function in use, and corrections are made to these scaled data values, this simply corresponds to bounding each input to the scaling range in use. In the results that follow this bounding action has been applied unless explicitly stated otherwise and all input and output values have been un-scaled from the internal values used for ease of interpretation.

### III. EXPERIMENTAL RESULTS

As a first example consider the well-known 3-bit parity data set, a compact but non-trivial data set. A maximally connected 3-3-1 network was trained on the ideal data for the 3-bit relationship shown in the first four columns of Table 1 (ignoring the italic figures in brackets). After training for 200 passes through the data the network showed no input error above 0.01 for any input of any training example. However, when a data set that consisted of the sum of the ideal data set and some noise (shown in italics in the brackets in Table 1) was used as a test set and input to the network, significant input errors were observed. Using the technique described above (and without changing the weight values in the network in any way) the data was adapted to better suit the network’s internal model. The difference between the initial and final values for each input of each example was recorded. Columns five to seven show the difference between the initial data and the final adjusted data, which is the estimate of the noise in this deliberately modified input data. The final column shows the modest number of times the cycle described in Step 4 was needed to reduce all input errors below 0.1 with \( \text{lcoeff} = 0.15 \). Clearly the estimates of the introduced error are quite accurate and show that this approach works even for a sparse data set.

Consider now a real life classification data set. The Iris data [5] has been extensively used and is known to be hard to totally classify. A regular architecture network trained on the first 75 data examples is extremely unlikely to be able to correctly classify all the second 75 data examples. A 4-3-3 network was trained on the first 75 examples and after 200 passes through the data was able to correctly classify each training example; although the confidence associated with the classification (the correct output value minus the largest of the other output values) was not always high. Then the internal model developed during training was used to estimate the error in the input values of the examples in the training data set (which ideally should have been zero) and in the second set of 75 data examples (the test set), three of which the network misclassified. Error here means the mismatch between the example being tested and the expectation of the internal model as built from the training examples.

Sixteen of the 75 training and 14 of the 75 test examples required some data adjustment before all input errors were less than 0.1. The average correction made to the training set inputs was 0.04, 0.03, 0.04 and 0.06 respectively. The equivalent figures for the test set were 0.05, 0.03, 0.05 and 0.06 respectively. The similarity between the two sets of values shows that the internal model produced by training using the training set is generally a reasonable description of both the training and test sets. However, some examples were exceptions to this general statement. A selection of these examples is shown in Table 2.

Consider example 34. While the network did correctly identify this as being class 2, the confidence of the decision (the difference between the correct and second place outputs) is not high (0.45). In order to better fit the internal model and be clearly identified as category 2 the input data has to be significantly adjusted (as shown by the final error estimates). Compare this with example 19 which being a more typical class 2 example (as far as the internal model in the network is

| Table 1. The error estimates (columns 5 to 7) for the 3-bit parity data when tested on the training data modified by the values shown in italics in the brackets in columns 1 to 3. |
|---|---|---|---|---|---|
| Input (noise added in italics) | Correct Output | Final Error Estimate | Cycles |
| Input | Output | 1 | 2 | 3 | | |
| 1 | 0 | 0.07 | 0.07 | 0.10 | 9 |
| 0 | 0 | -0.10 | 0.00 | 4 |
| 0 | 1 | 0.00 | 0.00 | 0.27 | 6 |
| 0 | 1 | 0.00 | 0.00 | 0.00 | 0 |
| 1 | 0 | 0.00 | 0.00 | 0.00 | 0 |
| 1 | 0 | 0.10 | 0.00 | 0.00 | 8 |
| 0 | 1 | 0.00 | -0.25 | 0.00 | 18 |
| 0 | 1 | 0.00 | 0.00 | -0.08 | 7 |
Determining the reason why bounding (limiting the possible values any input can be adjusted to) is necessary. In Table 2, bounding is not enforced for the entry labeled 125w and the final error estimates, if applied, would give input values far outside the training range.

Neural networks are better at interpolation than extrapolation, and while undoubtedly these error corrections would match the data to some point that the internal model believes corresponds to a class 3 example, points this far from the defined training problem space region are less likely to be meaningful. Entry 125 shows a different point that is within the training range and which still corresponds to a class 3 in the internal model. Compare this with example 110, a correctly identified class 3 example. The confidence of the decision is modest (0.674) and the reason, as far as the internal model is concerned, is that both inputs 3 and 4 are larger than the model would have ideally expected. Note that the ‘ideal’ inputs, as far as the internal model is concerned, are not the same as for example 125. It may be reasonable to assume that the internal model has a complex structure with many different regions associated with each class.

Finally consider example 106, which is more typical of the majority of examples in that only minor changes are suggested. It is possible that in such cases the information obtained by adjusting the data is most useful. Of course, any usefulness has as a precondition the requirement that the quality of the internal model developed during the original training of the network is high.

IV. DISCUSSION

Estimates of the noise present in a particular example’s input are relative to the internal model that the network built up from training data during a conventional training phase. Obviously the quality of this model has a significant bearing on the reliability that can be placed on the noise estimate. Further, modifying the input data to fit the network’s internal model will produce one, but not necessarily the only, revised set of input values that better fit the model. This is particularly significant if the adjustment that needs to be made to the data is large. The correction suggested for totally misclassified data, for example, may be open to suspicion.

The successful adjustment to data relies on there being a gradient in the error surface towards a point of better fit. Experiments were conducted with a data set consisting of two uniform regions corresponding to different classifications. Attempts to adjust a single point that was given the wrong classification into the region that did correspond with this point’s true classification were not in general successful as there was insufficient gradient to give guidance to the adjustments.

However, provided the networks internal model does not consist of large regions of uniform output and is a reasonable approximation of the training and test data, and the range of possible input values is bounded, the difference between the original data and the adjusted data may give a useful insight not available from just a consideration of the difference between the actual output of the network and the ideal output.

Consideration of the average noise estimates of particular inputs across the whole of the training data may give an insight into which of the data collection mechanisms that provide each input could most profitably be adjusted in order to improve the performance of the network. Comparisons of the noise estimate figures between the examples in a data set can reveal examples that are marginally fitted to the model. This again may lead to insights into areas in which the data collection process can be improved - for example, if all noisy examples come from one extreme of the range of data collected, or all noisy data samples come from one particular source.

The sample data sets described above are classification tasks for which nothing other than binary outputs makes sense. However, the same process can be used for analogue output values. For example, suppose a network has been trained to predict credit worthiness in such a way that a value of 0.5
represents a marginal credit risk. An application that generated a score of, say, 0.6, while clearly somewhat better than marginal, might be able to be better assessed by adjusting the data to produce an output of 0.5. This would show how far the combination of input values were from the nearest corresponding internal expectation values for marginal credit risk. Adjusting any one input value to see the effect on the output is obviously trivial, but in real life applications the output is unlikely to have orthogonal dependency on the input values and testing combination of changes to analogue inputs would obviously be impractical. Since credit worthiness may be assumed to be a complex combination of interacting input factors, input ‘error’ estimates could provide more information about the application on which to base the final decision.

V. CONCLUSION

This paper has proposed using a well established training algorithm in a way not previously suggested so that a trained network may be used as a data analysis tool. The results on two data sets show that the proposed technique is practicable. While this may be considered as establishing a ‘proof of concept’, more needs to be done to fully establish the usefulness of the approach, in particular to establish how the magnitude of the sequence of adjustments to the input data affects the final result.

ACKNOWLEDGMENT

The author would like to acknowledge all members of the Complex Intelligent Systems Laboratory for valuable discussion.

REFERENCES
