ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/225525743

# Automatic scaling using gamma learning for feedforward neural networks

**Chapter** · January 2006

DOI: 10.1007/3-540-59497-3_198

CITATIONS
4

READS
11

**4 authors**, including:

Andries Engelbrecht
University of Pretoria
**300** PUBLICATIONS **13,472** CITATIONS
SEE PROFILE

Jaco Geldenhuys
Stellenbosch University
**47** PUBLICATIONS **540** CITATIONS
SEE PROFILE

Jacek M. Zurada
University of Louisville
**351** PUBLICATIONS **5,678** CITATIONS
SEE PROFILE

**Some of the authors of this publication are also working on these related projects:**

Project Nonnegative Matrix Factorization View project

Project Fitness Landscape Analysis of Neural Networks View project

All content following this page was uploaded by Jaco Geldenhuys on 04 January 2017.

The user has requested enhancement of the downloaded file.

# AUTOMATIC SCALING USING GAMMA LEARNING FOR FEEDFORWARD NEURAL NETWORKS

AP Engelbrecht I Cloete J Geldenhuys JM Zurada*
ian@cs.sun.ac.za

*Computer Science Department, University of Stellenbosch, Stellenbosch 7600, South Africa*

**Abstract**

Standard error back-propagation requires output data that is scaled to lie within the active area of the activation function. We show that normalizing data to conform to this requirement is not only a time-consuming process, but can also introduce inaccuracies in modelling of the data. In this paper we propose the gamma learning rule for feedforward neural networks which eliminates the need to scale output data before training. We show that the utilization of "self-scaling" units results in faster convergence and more accurate results compared to the rescaled results of standard back-propagation.

## 1 Introduction

Many artificial neural networks trained with the popular error back propagating training algorithm, also called the *delta rule*, contain units having the well-known sigmoid activation function where $\lambda$ is a positive constant [Zurada, 1992a]:

$$f(\lambda, y) = \frac{1}{1 + e^{-\lambda y}} \tag{1}$$

A problem with this squashing function is that its output is always in the range $[0, 1]$, thus requiring scaling of the desired output before training to fit into this range. In addition to scaling of the output data, the input data is normally scaled to lie within the active area of the sigmoid activation function (e.g. to the range $[-\sqrt{3}, \sqrt{3}]$). In this paper we investigate the effects that scaling of the output data has on the learning process.

In practice, the data set presented to a neural network often contains values which lie outside the active range of the sigmoid activation function. If the delta learning rule with sigmoid activation functions is used to learn the data set, the data must be pre-processed before training. During pre-processing, the data is compressed to fit into the active range of the sigmoid function. This scaled data set is then used for training purposes. To interpret the results obtained from the neural network, the outputs must be rescaled to the original range. From the user's viewpoint the accuracy obtained by the neural network refers to this rescaled data set. We show that the scaling of outputs into a smaller range than the original unscaled range leads to longer training times to reach a specified accuracy on the rescaled data.

In this paper we extend the delta rule to the so-called *gamma learning rule* which adjusts the output range of the sigmoid activation function during learning. Thus, the gamma rule is effectively performing automatic scaling – a property applicable to almost all applications. Recently, Zurada proposed the *lambda learning*

*University of Louisville, Louisville, Kentucky 40292, USA *rule* where the constant $\lambda$ in (1) is treated as a variable and also adapted during training [Zurada, 1992b]. We base the derivation of the gamma rule on the lambda rule and denote the combination as the *lambda-gamma learning rule*, which is more general than the delta rule.

The gamma rule is reminiscent of biological neurons which are able to adjust to signals of various natures through transmitter depletion and contrast enhancement. For instance, cells in the auditory system exhibit "stimulus selectivity" [Morgan, 1991], becoming attuned to a characteristic frequency.

In the next section we investigate the effects of scaling of the output data, and show the advantages of self-scaling output units. The lambda-gamma rule is derived for a single neuron in section 3, and extended to single layer learning in section 4. Section 5 generalizes the rule for hidden layer learning. A complete general learning algorithm is presented in section 6, and experimental results are reported in section 7.

## 2 Effects of scaling

For the purpose of this exposition assume an output layer which consists of one neuron[1]. Without loss of generality, assume that the desired output data is scaled into the range $[0, 1]$ using linear scaling:

$$d_s = c_1 d_u + c_2 \tag{2}$$

where $d_u$ is the original unscaled desired output data (i.e. raw data), and $d_s$ is the corresponding scaled desired output data to be used for training. To scale data to the range $[0, 1]$ the scaling factors $c_1$ and $c_2$ are the following:

$$c_1 = \frac{1}{\max_{p=1,\ldots,P}\{d_u^{(p)}\} - \min_{p=1,\ldots,P}\{d_u^{(p)}\}} \qquad c_2 = \frac{-\min_{p=1,\ldots,P}\{d_u^{(p)}\}}{\max_{p=1,\ldots,P}\{d_u^{(p)}\} - \min_{p=1,\ldots,P}\{d_u^{(p)}\}} \tag{3}$$

with $P$ the total number of patterns. Then, from (2) the rescaled desired output $d_r$ is

$$d_r = \frac{1}{c_1} d_s - \frac{c_2}{c_1} \tag{4}$$

Let $o_s$ denote the actual output of output neuron $o$. Then, similarly to (4), $o_r$ is the actual output rescaled to the original output range. Assume it is possible to learn original unscaled data, and let $o_u$ denote the actual unscaled output of neuron $o$. Since the values of desired output data are not changed during training, it is clear that $d_u = d_r$, and under ideal conditions we will also have that $o_u = o_r$. However, this will require perfect learning with zero error which is in practice not realizable. Let $MSE_s$ and $MSE_r$ respectively denote the mean square error for the scaled and rescaled data over the entire training set. Then,

$$MSE_s = \sum_{p=1}^{P} (d_s^{(p)} - o_s^{(p)})^2 / P \tag{5}$$

$$MSE_r = \sum_{p=1}^{P} (d_r^{(p)} - o_r^{(p)})^2 / P \tag{6}$$

By substitution of equation (4) in (6) we obtain

$$MSE_r = \sum_{p=1}^{P} [(\frac{1}{c_1} d_s^{(p)} - \frac{c_2}{c_1}) - (\frac{1}{c_1} o_s^{(p)} - \frac{c_2}{c_1})]^2 / P = (\frac{1}{c_1})^2 MSE_s \tag{7}$$

Equation (7) illustrates a clear relation between the scaled and rescaled error. If

$$|c_1| < 1 \qquad (c_1 \neq 0) \tag{8}$$

[1]The derivations in this section can easily be extrapolated to an output layer with more than one neuron.

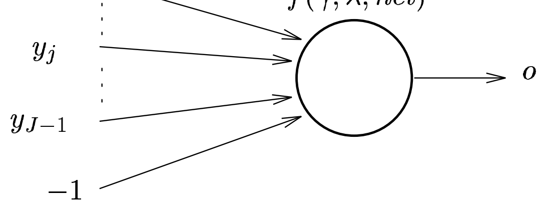

Figure 1: Lambda-gamma learning for a single neuron

then (7) indicates that the rescaled error is a factor of $(\frac{1}{c_1})^2$ larger than the scaled error, where condition (8) corresponds to the compression of data into a smaller range than the original range.

For the following, assume it is possible to learn the original unscaled data. Let $MSE_u$ denote the mean square error for the unscaled data. The relationship illustrated above indicates that in order to obtain a rescaled accuracy $MSE_r$ which is equal to $MSE_u$, the network must be trained longer until

$$MSE_s = (c_1)^2 MSE_r \tag{9}$$

On the other hand, if

$$|c_1| > 1 \tag{10}$$

we have from (7) that the rescaled error is a factor of $(\frac{1}{c_1})^2$ smaller than the scaled error. This corresponds to our claim that training on data which is expanded over a wider range will lead to faster convergence, since a scaling factor $c_1$ which conforms to condition (10) represents the scaling of data to a larger range than the original unscaled range.

From this investigation into the effects of scaling we conclude that it is preferable to use "self-scaling" output units and to learn original unscaled data when the range is greater than $[0, 1]$. This will significantly decrease the number of training cycles compared to learning scaled data, especially when $|c_1|$ is very small. Currently only linear self-scaling output units are available. In the next sections we propose the use of sigmoid self-scaling output units where the output range of the sigmoid activation function is dynamically adapted to span the original output range. The online adjustment of the output range enables the learning of unscaled data.

## 3 Lambda-gamma single neuron learning

The customary sigmoid function (1) for a single neuron is modified to include a *range coefficient* $\gamma$ and a *steepness coefficient* $\lambda$, to give:

$$f(\gamma, \lambda, net) = \frac{\gamma}{1 + e^{-\lambda net}} \tag{11}$$

where the activation value is $net(\vec{w}, \vec{y}) = \vec{w}^t \vec{y}$, the augmented input vector is $\vec{y} = [y_1 \; y_2 \; \ldots \; y_{n-1} \; -1]^t$ and the weight vector is $\vec{w} = [w_1 \; w_2 \; \ldots \; w_n]^t$. In the classical delta rule the neuron therefore learns in $(n-1)$-dimensional non-augmented weight space in which $n$ weights are adjustable. The lambda and gamma learning rules expand the learning space to $(n+1)$ dimensions, while the lambda-gamma learning rule expands it to $(n+2)$ dimensions. In addition to weight learning, both the steepness $\lambda$ and the range $\gamma$ undergo adjustments in the negative gradient direction.

Referring to Figure 1 and using the customary expression for error between the desired value $d$ and the actual output of the neuron $o$,

$$E(\gamma, \lambda, \vec{w}) = \frac{1}{2}[d - o(\gamma, \lambda, \vec{w})]^2$$

where

$$o(\gamma, \lambda, \vec{w}) = f(\gamma, \lambda, net(\vec{w}, \vec{y}))$$

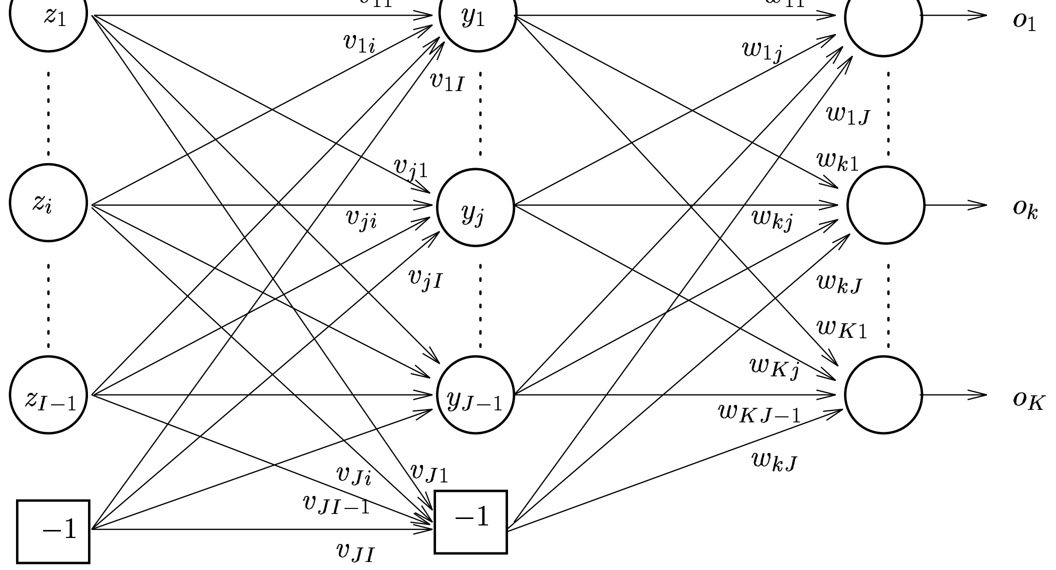

Figure 2: Layered feedforward network

we obtain the following weight adjustments for single neuron learning:

$$\Delta w_j = -\eta_1 \frac{\partial E}{\partial w_j} = -\eta 1 \frac{\partial E}{\partial o} \frac{\partial o}{\partial net} \frac{\partial net}{\partial w_j} = \eta_1 (d-o) \frac{\lambda}{\gamma} o(\gamma - o) y_j \tag{12}$$

$$\Delta \lambda = -\eta_2 \frac{\partial E}{\partial \lambda} = -\eta_2 \frac{\partial E}{\partial o} \frac{\partial o}{\partial \lambda} = \eta_2 (d-o) \frac{1}{\gamma} o(\gamma - o) net \tag{13}$$

$$\Delta \gamma = -\eta_3 \frac{\partial E}{\partial \gamma} = -\eta_3 \frac{\partial E}{\partial o} \frac{\partial o}{\partial \gamma} = \eta_3 (d-o) \frac{1}{\gamma} o \tag{14}$$

where $\eta_1, \eta_2$ and $\eta_3$ are positive learning constants usually selected as arbitrarily small values. Inspection of expression (12) coincides with the delta learning rule [Zurada, 1992a] when $\lambda = \gamma = 1$, and the lambda learning rule [Zurada, 1992b] when $\gamma = 1$. Expression (13) similarly reduces to the lambda rule when $\gamma = 1$. The extension to the lambda-gamma learning rule where a neuron's activation value can be "self-scaling" to the desired range is repesented by expression (14).

The next section illustrates single layer learning for the lambda-gamma learning rule.

## 4 Single layer learning

Assume that the neurons in the output layer $\vec{o}$ undergo training. In addition to a net input and activation value, each neuron $o_k$ $(k = 1, \ldots, K)$ has a range coefficient $\gamma_{o_k}$ and a steepness coefficient $\lambda_{o_k}$. The range and steepness coefficients are trained along with the weights $w_{kj}$ for all hidden neurons $y_j$ $(j = 1, \ldots, J)$ shown as the rightmost two layers in Figure 2. The usual definitions for error and error signal terms are used:

$$E = \frac{1}{2} \sum_{k=1}^{K} [d_k - o_k(\gamma_{o_k}, \lambda_{o_k}, net_{o_k})]^2 \tag{15}$$

$$\delta_{o_k} = \frac{\partial E}{\partial net_{o_k}} = -\frac{\lambda_{o_k}}{\gamma_{o_k}} (d_k - o_k) o_k (\gamma_{o_k} - o_k) \tag{16}$$

where $\vec{d}$ and $\vec{o}$ are respectively the desired output and the actual output vectors, and $\vec{\delta}_o$ and $\vec{\delta}_y$ are respectively the error signal term vectors for the output and hidden layers. The adjustments to the learning variables are given below:

$$\Delta w_{kj} = -\eta_1 \frac{\partial E}{\partial w_{kj}} = -\eta_1 \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_{o_k}} \frac{\partial net_{o_k}}{\partial w_{kj}} = -\eta_1 \delta_{o_k} y_j \tag{17}$$

$$\Delta\lambda_{o_k} = -\eta_2 \frac{}{\partial\lambda_{o_k}} = -\eta_2 \frac{}{\partial o_k}\frac{}{\partial\lambda_{o_k}} = -\eta_2 \delta_{o_k} \frac{}{\lambda_{o_k}} \tag{18}$$

$$\Delta\gamma_{o_k} = -\eta_3 \frac{\partial E}{\partial\gamma_{o_k}} = -\eta_3 \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial\gamma_{o_k}} = \eta_3 (d_k - o_k)\frac{1}{\gamma_{o_k}} o_k \tag{19}$$

where

$$o_k = f(\gamma_{o_k}, \lambda_{o_k}, net_{o_k}(\vec{w}_k, \vec{y})) \quad \text{and} \quad net_{o_k}(\vec{w}_k, \vec{y}) = \sum_{j=1}^{J} w_{kj} y_j$$

Hidden layer learning for the lambda-gamma learning rule is described in the next section.

## 5 Hidden layer learning

To train the neurons $y_j$ $(j = 1, \ldots, J)$ in the hidden layer, the weights $v_{ji}$ $(i = 1, \ldots, I)$, the steepness coefficient $\lambda_{y_j}$ and the range coefficient $\gamma_{y_j}$ must be adjusted during each iteration of the learning algorithm, using respectively

$$\Delta v_{ji} = -\eta_1 \frac{\partial E}{\partial v_{ji}} \tag{20}$$

$$\Delta\lambda_{y_j} = -\eta_2 \frac{\partial E}{\partial\lambda_{y_j}} \tag{21}$$

$$\Delta\gamma_{y_j} = -\eta_3 \frac{\partial E}{\partial\gamma_{y_j}} \tag{22}$$

Using the error (15) and error signal term

$$\delta_{y_j} = \frac{\partial E}{\partial net_{y_j}} = \frac{\lambda_{y_j}}{\gamma_{y_j}}(\gamma_{y_j} - y_j) y_j \sum_{k=1}^{K} \delta_{o_k} w_{kj} \tag{23}$$

we obtain

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial net_{y_j}}\frac{\partial net_{y_j}}{\partial v_{ji}} = \delta_{y_j} z_i \tag{24}$$

$$\frac{\partial E}{\partial\lambda_{y_j}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial\lambda_{y_j}} = \delta_{y_j}\frac{net_{y_j}}{\lambda_{y_j}} \tag{25}$$

$$\frac{\partial E}{\partial\gamma_{y_j}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial\gamma_{y_j}} = \frac{1}{\gamma_{y_j}} f(\gamma_{y_j}, \lambda_{y_j}, net_{y_j}) \sum_{k=1}^{K} \delta_{o_k} w_{kj} \tag{26}$$

where we have from (15)

$$\frac{\partial E}{\partial y_j} = \sum_{k=1}^{K} \delta_{o_k} w_{kj} \tag{27}$$

Substitution of equations (24), (25) and (26) into equations (20), (21) and (22) respectively yields the following adjustments for the hidden layer neurons:

$$\begin{aligned}
\Delta v_{ji} &= -\eta_1 \delta_{y_j} z_i \\
\Delta\lambda_{y_j} &= -\eta_2 \delta_{y_j}\frac{net_{y_j}}{\lambda_{y_j}} \\
\Delta\gamma_{y_j} &= -\eta_3 \frac{1}{\gamma_{y_j}} f(\gamma_{y_j}, \lambda_{y_j}, net_{y_j}) \sum_{k=1}^{K} \delta_{o_k} w_{kj}
\end{aligned}$$

where

$$y_j = f(\gamma_{y_j}, \lambda_{y_j}, net_{y_j}(\vec{v}_j, \vec{z})) \quad \text{and} \quad net_{y_j}(\vec{v}_j, \vec{z}) = \sum_{i=1}^{I} v_{ji} z_i$$

with $\delta_{o_k}$ and $\delta_{y_j}$ respectively the error signal term of the $k$-th output neuron and the $j$-th hidden neuron by equations (16) and (23). As mentioned previously, the adjustments reduce to that of the delta rule when $\lambda$ and $\gamma$ are constants equal to 1, the lambda rule when $\gamma$ is equal to 1 and the gamma rule when $\lambda$ is equal to 1. For training, the complete back-propagation algorithm in [Zurada, 1992b] is updated to reflect the changes given above. The updated algorithm is presented in the next section.

## 6 Complete lambda-gamma learning algorithm

The algorithm presented in [Zurada, 1992b] is modified below to reflect the lambda-gamma learning rule which is more general than the delta and lambda learning rules. Changes correspond to the adjustments to weights, steepness and range coefficients.

**Begin:** Given $P$ training pairs of vectors of inputs and desired outputs $\{(\vec{z}_1, \vec{d}_1), (\vec{z}_2, \vec{d}_2), \ldots, (\vec{z}_p, \vec{d}_p)\}$ where $\vec{z}_i$ is $(I \times 1)$, $\vec{d}_i$ is $(K \times 1)$ and $i = 1, \ldots, P$; $\vec{y}$ is $(J \times 1)$ and $\vec{o}$ is $(K \times 1)$.

**Step 1:** Choose the values of the learning rates $\eta_1, \eta_2$ and $\eta_3$ according to the learning rule:

| | |
|---|---|
| Delta learning rule | $\eta_1 > 0,\ \eta_2 = 0,\ \eta_3 = 0$ |
| Lambda learning rule | $\eta_1 > 0,\ \eta_2 > 0,\ \eta_3 = 0$ |
| Gamma learning rule | $\eta_1 > 0,\ \eta_2 = 0,\ \eta_3 > 0$ |
| Lambda-gamma learning rule | $\eta_1 > 0,\ \eta_2 > 0,\ \eta_3 > 0$ |

Choose an acceptable training error $E_{max}$. Weights $W$ $(K \times J)$ and $V$ $(J \times I)$ are initialized to small random values. Initialize the number of cycles $q$ and the training pairs counter $p$ to $q = 1,\ p = 1$. Let $E = 0$ and initialize the steepness and range coefficients

$$\lambda_{y_j} = \gamma_{y_j} = 1 \quad \forall\, j = 1, \ldots, J \qquad \text{and} \qquad \lambda_{o_k} = \gamma_{o_k} = 1 \quad \forall\, k = 1, \ldots, K$$

**Step 2:** Start training. Input is presented and the layers' outputs are computed using $f(\gamma, \lambda, net)$ as in equation (11):

$$\vec{z} = \vec{z}_p, \quad \vec{d} = \vec{d}_p \qquad \text{and} \qquad y_j = f(\gamma_{y_j}, \lambda_{y_j}, \vec{v}_j^{\,t} \vec{z}) \quad \forall\, j = 1, \ldots, J$$

where $\vec{v}_j$, a column vector, is the $j$-th row of $V$ and

$$o_k = f(\gamma_{o_k}, \lambda_{o_k}, \vec{w}_k^{\,t} \vec{y}) \quad \forall\, k = 1, \ldots, K$$

where $\vec{w}_k$, a column vector, is the $k$-th row of $W$.

**Step 3:** The error value is computed:

$$E = E + \frac{1}{2}(d_k - o_k)^2 \quad \forall\, k = 1, \ldots, K$$

**Step 4:** The error signal vectors $\vec{\delta}_o$ $(K \times 1)$ and $\vec{\delta}_y$ $(J \times 1)$ of both the output and hidden layers are computed

$$\delta_{o_k} = -\frac{\lambda_{o_k}}{\gamma_{o_k}}(d_k - o_k) o_k (\gamma_{o_k} - o_k) \quad \forall\, k = 1, \ldots, K$$

$$\delta_{y_j} = \frac{\lambda_{y_j}}{\gamma_{y_j}} y_j (\gamma_{y_j} - y_j) \sum_{k=1}^{K} \delta_{o_k} w_{kj} \quad \forall\, j = 1, \ldots . J$$

**Step 5:** Output layer weights and gains are adjusted:

$$w_{kj} = w_{kj} + \eta_1 \delta_{o_k} y_j \qquad \lambda_{o_k} = \lambda_{o_k} + \eta_2 \delta_{o_k} \frac{net_{o_k}}{\lambda_{o_k}} \qquad \gamma_{o_k} = \gamma_{o_k} + \eta_3 (d_k - o_k) \frac{1}{\gamma_{o_k}} o_k$$

for all $k = 1, \ldots, K$ and $j = 1, \ldots, J$.

$$v_{ji} = v_{ji} + \eta_1 \delta_{y_j} z_i \quad \lambda_{y_j} = \lambda_{y_j} + \eta_2 \frac{1}{\lambda_{y_j}} \delta_{y_j} net_{y_j} \quad \gamma_{y_j} = \gamma_{y_j} + \eta_3 \frac{1}{\gamma_{y_j}} f(\gamma_{y_j}, \lambda_{y_j}, net_{y_j}) \sum_{k=1}^{K} \delta_{o_k} w_{kj}$$

for all $j = 1, \ldots, J$ and $i = 1, \ldots, I$.

**Step 7:** If $p < P$ then let $p = p + 1$ and go to Step 2; otherwise go to Step 8.

**Step 8:** One training cycle is completed. If $E < E_{max}$ then terminate the training session. Output the cycle counter $q$ and error $E$; otherwise let $E = 0,\ p = 1,\ q = q + 1$ and initiate a new training cycle by going to Step 2.

## 7 Experimental results

We have used a simple function approximation experiment to substantiate our claims to the effects of scaling. A 1-10-1 network architecture was used to approximate the function $f(z) = |z|$. For the experiments described below, we have used the lambda-gamma learning algorithm to train on original unscaled data, and the delta learning algorithm to train on the scaled data. For illustration purposes, Figure 3 also shows the learning profile on the rescaled output data. The mean square error $MSE_r$ on the rescaled output data is calculated form equations (4) and (6) after each epoch. Both experiments use the same initial weights, which are initialized as random values in the range $[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}]$.

- **Experiment 1:** For this experiment we have $z \in [0.4, 0.6]$, and $d \in [0.4, 0.6]$. The desired outputs $d$ are linearly scaled to $[0, 1]$ using (2). From (3) we have $|c_1| = 5$, which illustrates the effect when output data is scaled to a larger range than the original. Figure 3(a) shows that the mean square error for the rescaled data is smaller than the mean square error of the scaled data for each epoch when $|c_1| > 1$. For example, from Figure 3(a) we see that a required error of 0.0005 on the scaled data has already been reached at epoch 55 on the rescaled data compared to epoch 150 on the scaled data.

- **Experiment 2:** For this experiment we have $z \in [-5, 5]$ and $d \in [0, 5]$. The desired outputs $d$ are linearly scaled to $[0, 1]$ using (2). Then, from (3) we have $|c_1| = \frac{1}{5}$ which corresponds to the compression of data. From Figure 3(b) we observe that an error of 0.02, which is reached at epoch 20 using lambda-gamma learning on unscaled data, is reached at epoch 140 on the rescaled data. Longer training is therefore required on scaled data to obtain a specified error equivalent on the original data.

Figure 3 also shows the learning profile for the lambda-gamma rule on unscaled data. Table 1 shows that condition (7) holds for arbitrarily selected epochs: for any given epoch, the error $MSE_r$ on the rescaled data is a factor of $(\frac{1}{c_1})^2$ larger than the error $MSE_s$ on the scaled data when $|c_1| < 1$, and a factor $(\frac{1}{c_1})^2$ smaller when $|c_1| > 1$.

The results presented in this section confirm our conclusion that training on output data that is scaled into a smaller range causes longer training times to reach a required accuracy on the rescaled output data (training accuracy is normally specified in terms of the rescaled data). The problem escalates as $|c_1|$ becomes very small. With the lambda-gamma learning rule, the same accuracy is obtained in less training cycles.

## 8 Conclusions

We have derived a relationship between the mean square errors for scaled and rescaled data when output data is linearly scaled. This relationship has indicated that the compression of data causes longer training

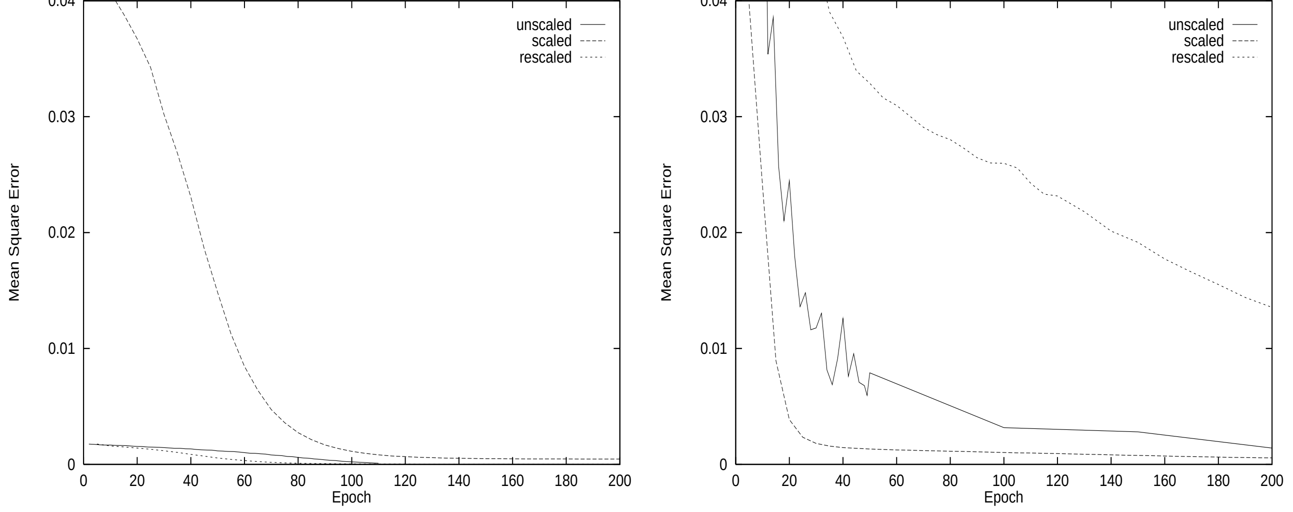

Figure 3: Learning profiles for unscaled, scaled and rescaled data.

| | Experiment 1 | | | Experiment 2 | | |
|---|---|---|---|---|---|---|
| Epoch | $MSE_s$ | $MSE_r$ | $(\frac{1}{c_1})^2 MSE_s$ | $MSE_s$ | $MSE_r$ | $(\frac{1}{c_1})^2 MSE_s$ |
| 5 | 0.043914 | 0.001757 | 0.0017566 | 0.037712 | 0.94279 | 0.9428 |
| 50 | 0.013873 | 0.000555 | 0.0005549 | 0.001315 | 0.03288 | 0.032875 |
| 100 | 0.001086 | 0.00043 | 0.00004344 | 0.001039 | 0.025983 | 0.025975 |
| 150 | 0.000489 | 0.00002 | 0.00001956 | 0.000766 | 0.01915 | 0.01915 |
| 200 | 0.000455 | 0.000018 | 0.0000182 | 0.000542 | 0.013551 | 0.01355 |

Table 1: Comparison of $MSE_s$ with $MSE_r$.

times compared to training on the unscaled data. We have presented the lambda-gamma learning algorithm which utilizes self-scaling sigmoid output units.

In order to perform scaling, the maximum and minimum ranges of the input and output must be known. For incremental learning systems this is difficult to obtain, since all training pairs are not available before training. Upper and lower bounds need to be determined beforehand. Gamma learning eliminates this problem since the output range of the sigmoid activation function is dynamically adjusted during training. The lambda-gamma learning rule further seems to eliminate the need for internal rescaling within the units as reported by Rigler, Irvine and Vogl [Rigler, 1991].

## References

[Morgan, 1991] DP Morgan and CL Scofield, *Neural Networks and Speech Processing*, Kluwer Academic Publishers, 1991.

[Rigler, 1991] AK Rigler, JM Irvine and TP Vogl, *Rescaling of Variables in Back Propagation Learning*, Neural Networks, 4, pp 225–229, 1991.

[Zurada, 1992a] JM Zurada, *Introduction to Artificial Neural Systems*, West Publishing Company, 1992.

[Zurada, 1992b] JM Zurada, *Lambda Learning Rule for Feedforward Neural Networks*, Proceedings of the IEEE International Conference on Neural Networks, March 28–31, 1992, San Fransisco, California.