Physical Interpretation of Backpropagated Error in Neural Networks

M.A. van Wyk

27 April 2023

Abstract

In this note, we shed light on the physical meaning for the backpropagated error used by the backpropagation training algorithm. Essentially, for a given scalar output of the neural network, its backpropagated error is a linear apportionment of the error at it, in proportion to the linear gain between the outputs of neurons and the output according to a linearised-systems-view of the network. For multiple outputs, superposition provides the total/nett backpropagated error at the outputs of neurons.

Subsequently, we present some elementary statistical analysis for backpropagated errors in the network.

1. Introduction

A given input $\boldsymbol{x}$ from the input-output training set applied to a feedforward neural network (FNN) with some preassigned weights $\boldsymbol{W}$, result in an network output error $e(\boldsymbol{x}, \boldsymbol{W})$. Then, the immediate challenge faced is, how much (sign and magnitude of the increment) each (scalar) weight needs to be adjusted in order to reduce the output error and optimise the accuracy of the network. For a linear feedforward network with a fixed (i.e., time invariant) training set, the appropriate weights can be determined in a single step. Unfortunately, for a non-linear feedforward network, this is not achievable and the best we can do is to adjust the weights incrementally towards a possibly local optimal setting. Moreover, due to the non-linear, layered structure of such a network, it is apparent that the weights of the outermost function of the function composition in (1) (below) describing the network must be updated first, followed by the next function in the composition, etc. This is precisely the modus operandi of the backpropagation algorithm which is based on the backpropagated error.

For a particular weight, to determine quantitatively an increment that will improve the network's performance, its influence on the output (and hence the output error) must first be quantified. Determining the precise amount by which a weight influences the output in a non-linear network is impossible either due to non-invertibility of certain activation functions or the tedium it entails particularly for a large network. However, linear approximation allows one to estimate a weight's contribution to the output error by summing the linearised gains of all the shortest paths in the network that contain it. Appropriate normalisation then reveals that such a linear approximation essentially turns out to be linear apportionment of the output error among some selection of neurons over which the normalisation is performed. Furthermore, for sufficient redundancy built into a network, one expect that the output error can be made arbitrarily small, and by implication the actual errors at each neuron's output, resulting in the backpropagated errors to becoming accurate estimates.

Section 1 provides the context and terminology needed for the rest of the note, while Section 2 derives the expression for the backpropagated error, followed by its interpretation in Section 3. There, different normalisations provide different apportionments and hence interpretations. Section 4 uses the results of earlier sections to characterise the backpropagated error in terms correlations among apportioned errors for certain collections of neurons.

2. Context and Terminology

Here, we consider *feedforward neural networks* (FNNs) consisting of $L \geq 2$ layers, including the input layer of nodes.[1] To reduce the number of indexed quantities in the presentation, we will assume all neurons in the network to use the same activation function, namely $f(\cdot)$. This limitation is easily overcome by the introduction of indices, one index if uniqueness per layer or per track is required or two if uniqueness per neuron is required. Conventionally, layers are organised horizontally and numbered from left to right, starting with the input layer with network input $\boldsymbol{x}$ as 'layer 0' containing source/input nodes instead of neurons. Within each layer, neurons are organised vertically and then numbered from top to bottom, starting with the topmost neuron labelled as neuron (with coordinates) $(\ell, 1)$ followed by $(\ell, 2)$, etc., for the ℓ^{th} layer. Initially, before training commences, each layer is assumed to have exactly N neurons (bias node included), while in the fully trained network, redundant neurons would have been culled by the learning process, resulting in each layer containing at most N neurons. The assumption of all layers having N neurons, enables us to collectively represent all weights of the neural network in an orderly manner, using a single matrix $\boldsymbol{W}$.

Each weight in the network has three indices (i.e., subscripts) which are assigned according to the following convention: the first index of a weight specifies the layer, say layer ℓ, that it feeds *into*, the second index specifies which particular neuron, say m, in that layer it feeds into, with the third index specifying which neuron in the previous layer (implicitly, layer $\ell - 1$) it feeds *from*, say n. For example, $w_{10,27,3}$ identifies the weight that scales the output of (source) neuron 3 of (source) layer 9 and feeds this result into (destination) neuron 27 of (destination) layer 10. This notation makes the implicit assumption that the source layer is always located directly to the left of the destination layer. However, if this is not the case, then a fourth index would be required to uniquely identify weights.

Inside neuron (ℓ, j) there is linear combiner with output $\tilde{u}_{\ell,j} \coloneqq \sum_j w_{\ell,j} u_{\ell-1,j}$ (also referred to as the neuron's accumulated linear output) to which the neuron's activation function is applied to yield its output, $u_{\ell,j} \coloneqq f(\tilde{u}_{\ell,j})$. For simplicity of presentation and without loss of generality, we will focus on scalar-output FNNs (i.e., single-output FNNs). This, together with the naming convention adopted here, means that the last layer, layer L, contains a single neuron. The expression describing the network output in terms of the network input and network weights takes the form

$$u_{L,1}(\boldsymbol{x}, \boldsymbol{W}) = w_{L+1,1} f\left(\sum_{j_L=1}^{N} w_{L,1,j_L} f\left(\sum_{j_{L-1}=1}^{N} w_{L-1,j_L,j_{L-1}} f\left(\sum_{j_{L-2}=1}^{N} w_{L-2,j_{L-1},j_{L-2}} f\left(\cdots f\left(\sum_{j_1=1}^{N} w_{1,j_2,j_1} x_{j_1}\right)\right)\right)\right)\right) \quad \ldots(1)$$

where, to simplify the expression, we introduced redundancies.[2,3] Some thought reveals that this composition must be trained from the outside to the inside, i.e., backward from the output to the input.[4]

3. Derivation of the Backpropagated Error

For our discussion here, we consider the 'stochastic' quadratic cost function,

$$J_{\boldsymbol{x}} \equiv J(\boldsymbol{x}) = \frac{1}{2}\Big(u_{L,1}(\boldsymbol{x}) - d(\boldsymbol{x})\Big)^2,$$

[1] A general convention in the literature is to use the term 'neuron' for either a neuron or a node.

[2] For each layer, we introduced fictitious weights that 'feed *into*' its bias node, fictitiously; furthermore, this bias node is then also written as $f(\cdot)$.

[3] The network output scaling $w_{L+1,1}$ is required to account for training data not appropriately normalised. However, in this note, we will set $w_{L+1,1} = 1$.

[4] To simplify notation, for the single-output NN, the network output will be denoted $u(\boldsymbol{x})$, instead of $u_{L,1}(\boldsymbol{x})$. The same simplification will be used for the output error too.

where, $d(\boldsymbol{x})$ is the *desired* network output as a function of the network input $\boldsymbol{x}$, thus resulting in the error at the network output given by $e_{L,1}(\boldsymbol{x},\boldsymbol{W}) \coloneqq u_{L,1}(\boldsymbol{x}) - d(\boldsymbol{x})$.[5] For this particular choice of cost function, the output error can be expressed as the derivative of $J_{\boldsymbol{x}}$ with respect to the output of the output neuron, namely

$$e_{L,1}(\boldsymbol{x}) = \frac{\partial J_{\boldsymbol{x}}}{\partial u_{L,1}}.$$

When calculating this partial derivative, both $\boldsymbol{x}$ and $\boldsymbol{W}$ are implicitly held constant.

Generalisation: The latter expression enables us to define a kind of 'error' at the output of an arbitrary neuron in the network by simply differentiating the cost function $J_{\boldsymbol{x}}$ with respect to the output of the neuron in question. As such, this 'error' at the output of neuron (ℓ,k) is given by

$$e_{\ell,k}(\boldsymbol{x}) = \frac{\partial J_{\boldsymbol{x}}}{\partial u_{\ell,k}}.$$

We will term this error $e_{\ell,k}(\boldsymbol{x})$ the *backpropagated error* at the output of neuron (ℓ,k). Using the chain rule, this expression becomes

$$e_{\ell,k}(\boldsymbol{x}) = \frac{\partial J_{\boldsymbol{x}}}{\partial u_{\ell,k}} = \underbrace{\frac{\partial J_{\boldsymbol{x}}}{\partial u_{L,1}}}_{e_{L,1}(\boldsymbol{x})} \underbrace{\frac{\partial u_{L,1}}{\partial u_{\ell,k}}}_{G_{(\ell,k)\to(L,1)}} \qquad \ldots(2)$$

where $G_{(\ell,k)\to(L,1)}$ is the linearised gain of the portion of the (linearised) FNN that connects neuron (ℓ,k) to the network output. This amounts to simply identifying and linearising all paths connecting neuron (ℓ,k) to the output neuron and then summing these linearised path gains to obtain $G_{(\ell,k)\to(L,1)}$. Equation (2) can be interpreted as the linear system with input $e_{L,1}(\boldsymbol{x})$, output $e_{\ell,k}(\boldsymbol{x})$ and input-dependent and weight-dependent linear gain $G_{(\ell,k)\to(L,1)} \equiv G_{(\ell,k)\to(L,1)}(\boldsymbol{x},\boldsymbol{W})$.[6] which means that the output is related to the input and gain according to

$$e_{\ell,k}(\boldsymbol{x}) = G_{(\ell,k)\to(L,1)}\, e_{L,1}(\boldsymbol{x}) = G_{(\ell,k)\to(L,1)}\, e(\boldsymbol{x}). \qquad \ldots(3)$$

This shows that the backpropagated error at the output of any neuron (or even a node) is proportional to the error at the output of the FNN. From linear systems theory, it follows immediately that

$$e_{\ell,k}(\boldsymbol{x}) = \sum_{j=1}^{N} G_{(\ell,k)\to(\ell+1,j)}\, e_{\ell+1,j}(\boldsymbol{x})$$

where $G_{(\ell,k)\to(\ell+1,j)} \equiv G_{(\ell,k)\to(\ell+1,j)}(\boldsymbol{x},\boldsymbol{W})$ which, with the aid of the chain rule, gives

$$G_{(\ell,k)\to(\ell+1,j)} = \frac{\partial u_{\ell+1,j}}{\partial u_{\ell,k}} = \frac{\partial u_{\ell+1,j}}{\partial \tilde{u}_{\ell+1,j}} \frac{\partial \tilde{u}_{\ell+1,j}}{\partial u_{\ell,k}} = f'\big(\tilde{u}_{\ell+1,j}\big) w_{\ell+1,j,k} =: \breve{w}_{\ell+1,j,k},$$

where $\breve{w}_{\ell+1,j,k}$ simply absorbs the activation function's derivative and consequently,

$$e_{\ell,k}(\boldsymbol{x}) = \sum_{j=1}^{N} f'\big(\tilde{u}_{\ell+1,j}\big) w_{\ell+1,j,k}\, e_{\ell+1,j}(\boldsymbol{x}) \equiv \sum_{j=1}^{N} \breve{w}_{\ell+1,j,k}\, e_{\ell+1,j}(\boldsymbol{x}), \qquad \ldots(4)$$

which expresses the backpropagated error $e_{\ell,k}(\boldsymbol{x})$ as a linear combination of the backpropagated errors $\{e_{\ell+1,j}(\boldsymbol{x})\}_{j=1}^{N}$, one layer to the right. Equation (4) describes how the backpropagated error propagates

[5] Strictly speaking, both the backpropagated errors and neuron outputs depend on *both* the applied inputs and the prevailing weight settings of the NN, expressed as $e_{\ell,k}(\boldsymbol{x},\boldsymbol{W})$ and $u_{\ell,k}(\boldsymbol{x},\boldsymbol{W})$. However, to reduce clutter we usually only indicate dependence on the applied inputs.

[6] Here, we resist the temptation to simplify our notation to $G_{(\ell,k)}$ to signify the gain from neuron (ℓ,k) all the way to the network output.

from one layer to the next, moving from the output layer towards the input layer. From (4), we now distil the following error backpropagation *rule*:

> ***The backpropagated error at the output of a(n originating) neuron/node of some layer, is the backpropagated error from a(n affected) neuron, one layer to the right (so called affected layer), multiplied by the activation function's derivative evaluated at the affected neuron's accumulated linear output, multiplied by the weight feeding from the originating neuron to the affected neuron, then summing this for all affected neurons in the affected layer.***

Example: To demonstrate the above essentials, consider the three-layer FNN shown below in Figure 1.[7] Starting from the left, we will label the input layer as layer 0, the hidden layer as layer 1 and the output layer as layer 2. Within each layer, starting from the top and proceeding downward, we label the first neuron as 1, the next below it as 2. The input to the network, is denoted $\boldsymbol{x} \coloneqq [x_1, x_2]^T$.

For the input $\boldsymbol{x}$ and all weights considered fixed, the network output error $e(\boldsymbol{x}) \equiv e_{2,1}(\boldsymbol{x})$ follows as

$$\frac{\partial J_{\boldsymbol{x}}}{\partial u_{2,1}} = u_{2,1}(\boldsymbol{x}) - d(\boldsymbol{x}) = e(\boldsymbol{x}) \, .$$

We now move to the hidden layer, i.e., the layer feeding the output later, namely layer 1. The backpropagated error at $u_{1,1}$, the output of neuron 1 of layer 1 for fixed network input $\boldsymbol{x}$, is given by

$$e_{1,1}(\boldsymbol{x}) = \frac{\partial J_{\boldsymbol{x}}}{\partial u_{1,1}} = \underbrace{\frac{\partial J_{\boldsymbol{x}}}{\partial u_{2,1}}}_{e_{2,1}(\boldsymbol{x})} \underbrace{\frac{\partial u_{2,1}}{\partial \tilde{u}_{2,1}}}_{f'(\tilde{u}_{2,1})} \underbrace{\frac{\partial \tilde{u}_{2,1}}{\partial u_{1,1}}}_{w_{2,1,1}} = e(\boldsymbol{x}) f'(\tilde{u}_{2,1}) w_{2,1,1} \, ,$$

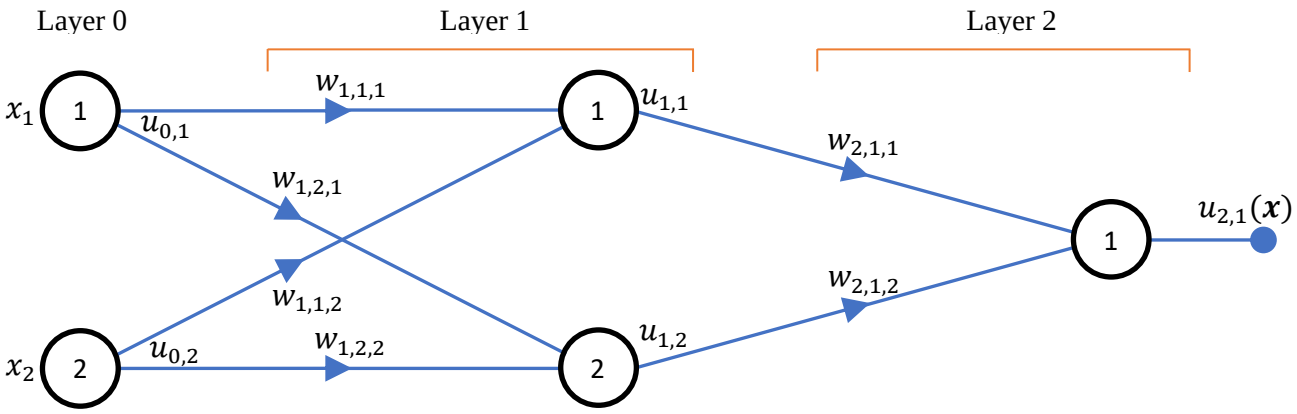

Figure 1.

since by the conventions chosen here, $u \equiv u_{2,1} \coloneqq f(\tilde{u}_{2,1})$ and $\tilde{u}_{2,1} \coloneqq \sum_i w_{2,1,i} u_{1,i}$. Similarly, the back propagated error $e_{1,2}(\boldsymbol{x})$ at $u_{1,2}(\boldsymbol{x})$, the output of neuron 2 of layer 1, for fixed input $\boldsymbol{x}$, is given by

$$e_{1,2}(\boldsymbol{x}) = \frac{\partial J_{\boldsymbol{x}}}{\partial u_{1,2}} = \underbrace{\frac{\partial J_{\boldsymbol{x}}}{\partial u_{2,1}}}_{e_{2,1}(\boldsymbol{x})} \underbrace{\frac{\partial u_{2,1}}{\partial \tilde{u}_{2,1}}}_{f'(\tilde{u}_{2,1})} \underbrace{\frac{\partial \tilde{u}_{2,1}}{\partial u_{1,2}}}_{w_{2,1,2}} = e(\boldsymbol{x}) f'(\tilde{u}_{2,1}) w_{2,1,2} \, .$$

To demonstrate that the backpropagated error can also be written for the input layer, we now advance to the layer 0. The backpropagated error $e_{0,1}(\boldsymbol{x})$ at $u_{0,1}(\boldsymbol{x})$, the output of neuron 1 of layer 0 for fixed input $\boldsymbol{x}$, is given by

[7] To reduce visual complexity, we omitted bias nodes for the output and hidden layers.

$$
\begin{aligned}
e_{0,1}(\boldsymbol{x}) &= \frac{\partial J_{\boldsymbol{x}}}{\partial u_{0,1}} = \frac{\partial J_{\boldsymbol{x}}}{\partial u_{1,1}} f'(\tilde{u}_{1,1}) w_{1,1,1} + \frac{\partial J_{\boldsymbol{x}}}{\partial u_{1,2}} f'(\tilde{u}_{1,2}) w_{1,2,1} \\
&= e_{1,1}(\boldsymbol{x}) f'(\tilde{u}_{1,1}) w_{1,1,1} + e_{1,2}(\boldsymbol{x}) f'(\tilde{u}_{1,2}) w_{1,2,1} \\
&= \{e_{2,1}(\boldsymbol{x}) f'(\tilde{u}_{2,1}) w_{2,1,1}\} f'(\tilde{u}_{1,1}) w_{1,1,1} + \{e_{2,1}(\boldsymbol{x}) f'(\tilde{u}_{2,1}) w_{2,1,2}\} f'(\tilde{u}_{1,2}) w_{1,2,1} \\
&= \{f'(\tilde{u}_{1,1}) f'(\tilde{u}_{2,1}) w_{1,1,1} w_{2,1,1} + f'(\tilde{u}_{1,2}) f'(\tilde{u}_{2,1}) w_{1,2,1} w_{2,1,2}\} e(\boldsymbol{x}) .
\end{aligned}
$$

Similarly, the backpropagated error $e_{0,2}(\boldsymbol{x})$ at output $u_{0,2}(\boldsymbol{x})$, for fixed input $\boldsymbol{x}$, is given by

$$
\begin{aligned}
e_{0,2}(\boldsymbol{x}) &= \frac{\partial J_{\boldsymbol{x}}}{\partial u_{0,2}} = \frac{\partial J_{\boldsymbol{x}}}{\partial u_{1,1}} f'(\tilde{u}_{1,1}) w_{1,1,2} + \frac{\partial J_{\boldsymbol{x}}}{\partial u_{1,2}} f'(\tilde{u}_{1,2}) w_{1,2,2} \\
&= e_{1,1}(\boldsymbol{x}) f'(\tilde{u}_{1,1}) w_{1,1,2} + e_{1,2}(\boldsymbol{x}) f'(\tilde{u}_{1,2}) w_{1,2,2} \\
&= \{e_{2,1}(\boldsymbol{x}) f'(\tilde{u}_{2,1}) w_{2,1,1}\} f'(\tilde{u}_{1,1}) w_{1,1,2} + \{e_{2,1}(\boldsymbol{x}) f'(\tilde{u}_{2,1}) w_{2,1,2}\} f'(\tilde{u}_{1,2}) w_{1,2,2} \\
&= \{f'(\tilde{u}_{1,1}) f'(\tilde{u}_{2,1}) w_{1,1,2} w_{2,1,1} + f'(\tilde{u}_{1,2}) f'(\tilde{u}_{2,1}) w_{1,2,2} w_{2,1,2}\} e(\boldsymbol{x}) .
\end{aligned}
$$

Note that the last two backpropagation errors were presented merely to demonstrate the process of back propagating the linearise error; these would not come into play during the use of gradient descent to adapt the network weights. Finally, note that these last two expressions could have been written down by inspection, using the above distilled backpropagation error rule. This concludes the example.

4. Interpretation of the Backpropagated Error

For the network inputs and weight settings kept fixed, the output error is fixed and to now interpret the backpropagated error, requires us to compare backpropagated errors of different neurons in the network. This reveals that the greater the linearised gain of the portion of the network between the neuron of interest and the output neuron, the greater the value of its backpropagated error. Thus, the backpropagated error represents the linearised contribution of the particular neuron to the error at the output. Consequently, if the linear gain between the neuron and the output is *zero*, then this neuron does *not* contribute to the output error for the prevailing situation (i.e., current input and network weight values). This naturally leads to the conclusion that the backpropagated error is simply just a *linear*[8] apportionment of the resulting network output error, among neurons. However, this apportionment is generally not normalised and hence summing over all backpropagated errors will generally be proportional but not be equal to the error at the output. Normalisation can be applied in several ways, three of which are discussed below. As we will see, however, even though these processes of normalisation are helpful with providing physically tangible interpretations of the backpropagated error, with some exceptions, often they are superfluous.

We conclude this discussion with the observation that the backpropagated error is merely an apparent linear measure of the error contributed by each neuron in the network towards the network output error. With sufficient redundancy in each layer of the network and sufficiently large training data set, as the learning process converges, the true error at the output of each neuron can be made sufficiently small for the backpropagated error there to approximate the true error there sufficiently good.

In-Layer Normalisation

Equation (3) can be interpreted as an 'unnormalised' (linear) apportionment of the output error among the neurons of a given (internal) layer, say layer $\ell \geq 1$, in accordance with the (linear) gains connecting its neurons to the output neuron. Normalising *within* a layer, yields the in-layer or 'vertically' normalised version of (3), namely

$$
\overline{e}_{\ell,k}(\boldsymbol{x}) = \overline{G}_{(\ell,k)\to(L,1)}\, e_{L,1}(\boldsymbol{x}), \qquad \ldots(5)
$$

[8] Unfortunately, it is usually either impossible or computationally too expensive to non-linearly apportion the contributions of internal neuron (ℓ, k)'s true output error $\varepsilon_{\ell,k}(\boldsymbol{x})$ towards the network output error, $\varepsilon_{L,1}(\boldsymbol{x}) \equiv \boldsymbol{e}_{L,1}(x)$.

where

$$\overline{e}_{\ell,k}(\boldsymbol{x}) \coloneqq \frac{e_{\ell,k}(\boldsymbol{x})}{\Gamma_\ell}, \quad \overline{G}_{(\ell,k)\to(L,1)} \coloneqq \frac{G_{(\ell,k)\to(L,1)}}{\Gamma_\ell} \quad \text{and} \quad \Gamma_\ell(\boldsymbol{x},\boldsymbol{W}) \coloneqq \sum_{k=1}^{N} G_{(\ell,k)\to(L,1)}(\boldsymbol{x},\boldsymbol{W}),$$

as long as $\Gamma_\ell \neq 0$ which is quite possible since these path gains are signed real values. Now,

$$e(\boldsymbol{x}) = \sum_{k=1}^{N} \overline{e}_{\ell,k}(\boldsymbol{x}),$$

and now the output error is fully accounted for among the neurons within the ℓ^{th} layer and hence we can now summarise all normalised backpropagated errors in the FNN within an $N \times L$ matrix,[9] namely

$$\overline{\boldsymbol{E}}(\boldsymbol{x}) \coloneqq \begin{bmatrix} \overline{e}_{1,1}(\boldsymbol{x}) & \cdots & \overline{e}_{L,1}(\boldsymbol{x}) \\ \vdots & \ddots & \vdots \\ \overline{e}_{1,N}(\boldsymbol{x}) & \cdots & \overline{e}_{L,N}(\boldsymbol{x}) \end{bmatrix},$$

with each column summing to $e(\boldsymbol{x})$ and all elements summed together yielding $N \cdot e(\boldsymbol{x})$.

Network-Wide Normalisation

Equation (3) can also be interpreted as an 'unnormalised' (linear) apportionment of the output error among *all* the internal neurons in the FNN. As before, the greater the (linear) gain is between a given internal neuron and the output neuron, the greater this internal neuron's (linear) contribution is to the output error. The normalised version of (3) now becomes

$$\bar{\bar{e}}_{\ell,k}(\boldsymbol{x}) = \bar{\bar{G}}_{(\ell,k)\to(L,1)}\, e(\boldsymbol{x}), \qquad \ldots(6)$$

where

$$\bar{\bar{e}}_{\ell,k}(\boldsymbol{x}) \coloneqq \frac{e_{\ell,k}(\boldsymbol{x})}{\Gamma}, \quad \bar{\bar{G}}_{(\ell,k)\to(L,1)} \coloneqq \frac{G_{(\ell,k)\to(L,1)}}{\Gamma} \quad \text{and} \quad \Gamma(\boldsymbol{x},\boldsymbol{W}) \coloneqq \sum_{\ell=1}^{L} \Gamma_\ell(\boldsymbol{x},\boldsymbol{W}),$$

as long as $\Gamma \neq 0$. An alternative expanded expression for Γ is

$$\Gamma(\boldsymbol{x},\boldsymbol{W}) \coloneqq \sum_{\ell=1}^{L}\sum_{j=1}^{N} G_{(\ell,k)\to(L,1)}(\boldsymbol{x},\boldsymbol{W}).$$

We could now summarise the normalised backpropagated errors in an $N \times L$ matrix, namely

$$\bar{\bar{\boldsymbol{E}}}(\boldsymbol{x}) \coloneqq \begin{bmatrix} \bar{\bar{e}}_{1,1}(\boldsymbol{x}) & \cdots & \bar{\bar{e}}_{L,1}(\boldsymbol{x}) \\ \vdots & \ddots & \vdots \\ \bar{\bar{e}}_{1,N}(\boldsymbol{x}) & \cdots & \bar{\bar{e}}_{L,N}(\boldsymbol{x}) \end{bmatrix},$$

with the property that the elements of this matrix sum to $e(\boldsymbol{x})$. This type of normalisation depicts weighted voting where mutual influences among neurons are neglected just as is done with mutual influences among human voters during political elections.

In-Track Normalisation

Here, normalisation is performed horizontally across layers a long a *track*, i.e., where the layer varies but the neuron position is held constant. Strictly speaking, the term 'track' is a misnomer but merely used for its geometric utility. For obvious reasons, both the input and output layers are omitted. For reasons of economy, we will omit the mathematical details here.

[9] Even though the backpropagated errors for input nodes can be calculated, these do not contribute to the backpropagation-based training and hence we exclude these from the normalisation processes discussed here.

5. Statistical Aspects of Backpropagated Error

In signal processing, measurable characteristics of errors are of cardinal importance for analysis and design. One such characteristic is the correlation between errors determined at different parts of a system, here an FNN.

Now, from (3) we have

$$e_{\ell_1,k_1}(\boldsymbol{x})e_{\ell_2,k_2}(\boldsymbol{x}) = G_{(\ell_1,k_1)\to(L,1)}\, G_{(\ell_2,k_2)\to(L,1)}\, |e(\boldsymbol{x})|^2.$$

Averaging over the training set, we obtain the correlation relation,

$$R_{(\ell_1,k_1),(\ell_2,k_2)} \coloneqq \langle e_{\ell_1,k_1}(\boldsymbol{x})e_{\ell_2,k_2}(\boldsymbol{x})\rangle = G_{(\ell_1,k_1)\to(L,1)}\, G_{(\ell_2,k_2)\to(L,1)} R_e\,, \qquad \ldots(7)$$

where $R_e \coloneqq \langle |e(\boldsymbol{x})|^2\rangle_{\boldsymbol{x}}$ and $\langle\cdot\rangle_{\boldsymbol{x}}$ denotes the empirical average over the training set. Note that the correlation of these two backpropagated errors is only zero when either the mean square output error R_e or at least one of the two path gains involved is zero. Next, we consider the correlation between two backpropagated errors at two neurons in adjacent layers in the network. Using (4), we obtain

$$\begin{aligned} R_{(\ell,k),(\ell+1,m)} &= \langle e_{\ell,k}(\boldsymbol{x})e_{\ell+1,m}(\boldsymbol{x})\rangle \\ &= \sum_{j=1}^{N} \breve{w}_{\ell+1,j,k}\, \langle e_{\ell+1,j}(\boldsymbol{x})e_{\ell+1,m}(\boldsymbol{x})\rangle \\ &= \sum_{j=1}^{N} \breve{w}_{\ell+1,j,k}\, R_{(\ell+1,k),(\ell+1,m)}\,. \end{aligned} \qquad \ldots(8)$$

From the correlation appearing on the right side of the last result, we observe that we also need

$$\begin{aligned} R_{(\ell,k),(\ell,m)} &= \langle e_{\ell,k}(\boldsymbol{x})e_{\ell,m}(\boldsymbol{x})\rangle \\ &= \langle \sum_{i=1}^{N} \breve{w}_{\ell+1,i,k}\, e_{\ell+1,i}(\boldsymbol{x}) \sum_{j=1}^{N} \breve{w}_{\ell+1,j,m}\, e_{\ell+1,j}(\boldsymbol{x})\rangle \\ &= \sum_{i,j=1}^{N} \breve{w}_{\ell+1,i,k}\, \breve{w}_{\ell+1,j,m}\, \langle e_{\ell+1,i}(\boldsymbol{x})e_{\ell+1,j}(\boldsymbol{x})\rangle \\ &= \sum_{i,j=1}^{N} \breve{w}_{\ell+1,i,k}\, \breve{w}_{\ell+1,j,m}\, R_{(\ell+1,i),(\ell+1,j)}\,. \end{aligned} \qquad \ldots(9)$$

Since the backpropagated errors associated with any two neurons in the network are correlated, according to (7), we compelled to move towards jointly considering the backpropagation errors of groups of neurons. In order to have sufficient structure to perform meaningful analyses, we consider structured vectors of errors, i.e., not vectors with arbitrary chosen errors. To this end, we define two types of backpropagated error vectors, namely

$$\boldsymbol{e}_{\ell,\cdot} \coloneqq \begin{bmatrix} e_{\ell,1}(\boldsymbol{x}) \\ \vdots \\ e_{\ell,N}(\boldsymbol{x}) \end{bmatrix} \in \mathbb{R}^{N\times 1} \quad \text{and} \quad \boldsymbol{e}_{\cdot,t}^T \coloneqq [e_{1,t}(\boldsymbol{x}) \quad \cdots \quad e_{L-1,t}(\boldsymbol{x})] \in \mathbb{R}^{1\times(L-1)}$$

where $\boldsymbol{e}_{\ell,:}$ is called the ℓ^{th} *layer error vector* and $\boldsymbol{e}_{:,t}$ is called the t^{th} *track error vector*. Notice that, here we have deliberately neglected normalisation since it cancels on either side of the equations below.

Since (4) applied layer-wise can be expressed as,

$$\boldsymbol{e}_{\ell,\cdot} \coloneqq \boldsymbol{V}_{\ell+1}^T \boldsymbol{e}_{\ell+1,\cdot}\,,$$

we can write the following inner and outer products,

$$\boldsymbol{e}_{\ell,\cdot}^T \boldsymbol{e}_{\ell+1,\cdot} \coloneqq \boldsymbol{e}_{\ell+1,\cdot}^T \boldsymbol{V}_{\ell+1}^T \boldsymbol{e}_{\ell+1,\cdot}\,,$$

and

$$\boldsymbol{e}_{\ell,\cdot}\boldsymbol{e}_{\ell+1,\cdot}^T \coloneqq \boldsymbol{V}_{\ell+1}^T\boldsymbol{e}_{\ell+1,\cdot}\boldsymbol{e}_{\ell+1,\cdot}^T \quad \text{and} \quad \boldsymbol{e}_{\ell,\cdot}\boldsymbol{e}_{\ell,\cdot}^T \coloneqq \boldsymbol{V}_{\ell+1}^T\boldsymbol{e}_{\ell+1,\cdot}\boldsymbol{e}_{\ell+1,\cdot}^T\boldsymbol{V}_{\ell+1}\,,$$

for $\ell = L-2, L-1, \ldots, 2$, with these equations' initial conditions given by

$$\boldsymbol{e}_{L-1,\cdot} = \overleftarrow{\boldsymbol{W}}_L \odot e_{L,1}(\boldsymbol{x}) \quad \text{where} \quad \overleftarrow{\boldsymbol{W}}_L \coloneqq \begin{bmatrix} \overleftarrow{w}_{L,1,1} & \overleftarrow{w}_{L,1,2} & \cdots & \overleftarrow{w}_{L,1,1} \end{bmatrix}^T,$$

and $\odot$ represents the component-wise product of two vectors of the same dimension. Averaging over the training set, then yields

$$\langle \boldsymbol{e}_{\ell,\cdot}^T\boldsymbol{e}_{\ell+1,\cdot}\rangle_{\boldsymbol{x}} \coloneqq \langle \boldsymbol{e}_{\ell+1,\cdot}^T\boldsymbol{V}_{\ell+1}^T\boldsymbol{e}_{\ell+1,\cdot}\rangle_{\boldsymbol{x}}\,, \tag{10}$$

$$\boldsymbol{R}_{\ell,\ell+1} \coloneqq \langle \boldsymbol{e}_{\ell,\cdot}\boldsymbol{e}_{\ell+1,\cdot}^T\rangle_{\boldsymbol{x}} \coloneqq \boldsymbol{V}_{\ell+1}^T\langle \boldsymbol{e}_{\ell+1,\cdot}\boldsymbol{e}_{\ell+1,\cdot}^T\rangle_{\boldsymbol{x}} = \boldsymbol{V}_{\ell+1}^T\boldsymbol{R}_{\ell+1,\ell+1}\,, \tag{11}$$

and

$$\boldsymbol{R}_{\ell,\ell} \coloneqq \langle \boldsymbol{e}_{\ell,\cdot}\boldsymbol{e}_{\ell,\cdot}^T\rangle_{\boldsymbol{x}} \coloneqq \boldsymbol{V}_{\ell+1}^T\langle \boldsymbol{e}_{\ell+1,\cdot}\boldsymbol{e}_{\ell+1,\cdot}^T\rangle_{\boldsymbol{x}} = \boldsymbol{V}_{\ell+1}^T\boldsymbol{R}_{\ell+1,\ell+1}V_{\ell+1}\,. \tag{12}$$

The latter two equations also follow from (8) and (9), respectively. Here $\boldsymbol{R}_{\ell,\ell+1}$ represents the cross-correlation matrix describing the cross-correlation between the ℓ^{th} and $(\ell+1)^{\text{th}}$ layer (backpropagated) error vectors and $\boldsymbol{R}_{\ell,\ell}$ is the correlation matrix for $\boldsymbol{e}_{\ell,\cdot}$, the ℓ^{th} layer (backpropagated) error vector. In general, $\boldsymbol{R}_{\ell,\ell}$ is symmetric but not Toeplitz since it is a correlation matrix and not an autocorrelation matrix.

Taking a closer look at these equations from the perspective of the properties of the backpropagated error embodied in (7), and assuming that neuron/node outputs and corresponding backpropagated errors are uncorrelated, then (10) assumes the form,

$$\langle \boldsymbol{e}_{\ell,\cdot}^T\boldsymbol{e}_{\ell+1,\cdot}\rangle_{\boldsymbol{x}} \equiv \langle \mathcal{W}_\ell\rangle_{\boldsymbol{x}}R_e\,, \qquad R_e \coloneqq \langle |e(\boldsymbol{x})|^2\rangle_{\boldsymbol{x}}$$

which is just the output error power scaled by $\mathcal{W}_\ell$ a scalar function of only the network inputs $\boldsymbol{x}$ and weights $\boldsymbol{W}_\ell$ located between the ℓ^{th} layer and the network output, i.e., $\mathcal{W}_\ell \equiv \mathcal{W}_\ell(\boldsymbol{x}, \boldsymbol{W}_\ell)$. Since, due to averaging over $\boldsymbol{x}$, $\langle \mathcal{W}_\ell\rangle_{\boldsymbol{x}}$ only depends on the weight settings $\boldsymbol{W}_\ell$, then so does the above inner product. It immediately follows that the backpropagated error vectors $\boldsymbol{e}_\ell$ and $\boldsymbol{e}_{\ell+1,\cdot}$ generally are orthogonal only if $\langle \mathcal{W}_\ell\rangle_{\boldsymbol{x}} = 0$ for the specific weight settings, $\boldsymbol{W}_\ell$.

If all neuron/node outputs and corresponding backpropagated errors are uncorrelated, then (12) takes the form,

$$\boldsymbol{R}_{\ell,\ell} \coloneqq \langle \boldsymbol{e}_{\ell,\cdot}\boldsymbol{e}_{\ell+1,\cdot}^T\rangle_{\boldsymbol{x}} = \boldsymbol{V}_{\ell+1}^T\langle \boldsymbol{e}_{\ell+1,\cdot}^T\rangle_{\boldsymbol{x}} = \langle \boldsymbol{\mathcal{V}}_{\ell+1}\rangle_{\boldsymbol{x}}R_e\,,$$

where $\boldsymbol{\mathcal{V}}_{\ell+1} \equiv \boldsymbol{\mathcal{V}}_{\ell+1}(\boldsymbol{x}, \boldsymbol{W}_\ell)$ is a real symmetric matrix depending only on inputs $\boldsymbol{x}$ in the training set and the weight settings, $\boldsymbol{W}_\ell$. The symmetric matrix $\langle \boldsymbol{\mathcal{V}}_{\ell+1}\rangle_{\boldsymbol{x}}$ depends only on the weight settings $\boldsymbol{W}_\ell$. Therefore, $\boldsymbol{R}_{\ell,\ell}$ turns out to merely be a matrix scaling of the output error power, R_e.

[*This section is still under construction.*]
