# Performing Basic Mathematics with Neurons/Nets: Part I

R. S. Neville
Department of Computation
UMIST, PO Box 88, Manchester, M60 1QD
Email: r.neville@co.umist.ac.uk

**Abstract - In this article we introduce a novel way of performing basic mathematics or symbolic algebra with neurons. The method is novel because we enable knowledge encapsulated in trained neurons to be utilised to prescribe the weights vectors of other neurons. The knowledge encapsulated in the trained neurons in this case is numerical values.**

## I. INTRODUCTION

The approach this article takes is to present a methodology which enables algebraic calculations to be performed with neurons. Algebra is the manipulation of symbols in order to portray relationships and properties of numbers in symbolic form in mathematics. Basic mathematics assigns numerical values to a set of symbols and then performs basic arithmetic operations. To enable a set of units to perform mathematic calculations, each neuron is assigned a symbolic tag. The tagged neurons (e.g. two units $A$ and $B$) are then trained to represent numeric values. The two trained neurons' weight vectors (i.e. $W^A$ and $W^B$) are then used to "prescribe" another neuron's weight vector $W^C$ for unit $C$. The newly "prescribed" weights allow neuron $C$ to output a numeric value which is a function of $A$ and $B$. Hopfield set the weights of a neural network using a "prescription" method. He developed one of the first associative memories in 1982, the so-called Hopfield Net. However, Hopfield does not adapt the weight as has been done in other training regimes. He set them up and in his original paper [1] he presented a method or what he calls a "prescription" for making a weight set, given a set of patterns which have to be stored in the net. His approach to information storage in a neural network was to load the patterns onto the net, and this is the storage prescription for Hopfield Nets. Hence, the weights are loaded onto the net by taking the sum of the two components across all the patterns for a given node pair and connected node weight.

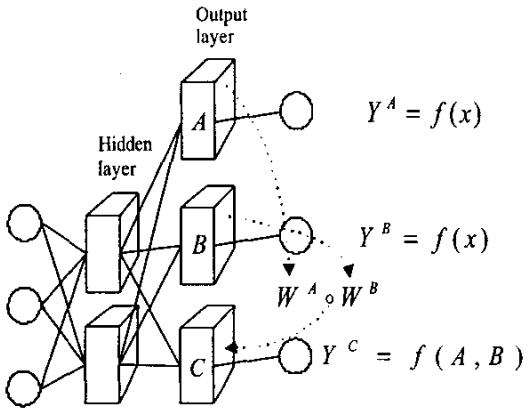



**FIGURE 1. TWO NEURONS $A$ AND $B$ ARE USED TO PRESCRIBE THE WEIGHTS OF A THIRD NEURON $C$**

This article shows that we incorporate prior knowledge into a neural network by using the knowledge in a trained net to prescribe the weights for a new output unit. This research has associations with work carried out by Sun [2] into Hybrid systems, the method we use to prescribe the weights could be viewed as a hybrid learning technique. Sun et al [3-6] has previously used hybrid models for learning sequential decision making. Our method of prescribing weights may also provide a possible answer to an architecture-related issue in Sun's paper 'Hybrid connectionist-symbolic Modules' [2]; namely, "how do we incorporate prior knowledge into hybrid architectures". This area requires further investigation. (Hybrid systems focus on learning and architectures that feature hybrid representations and support hybrid learning. Hybrid representations may combine (sub-conceptual) procedural knowledge and (conceptual) declarative knowledge [5, 6]. Hybrid learning may involve several types of learning, for example reinforcement learning and rule learning [5].)

### *A. Structure of Paper and Sister Paper*

This paper should be read in association with its sister paper [7]. The current paper gives a brief history of using neural networks to perform mathematics while part II gives an expanded history.

Following the brief introduction to sigma-pi units the Sigma-pi Neuron Model is then introduced. This is followed by an introduction to performing mathematics with neurons/nets. A description of the algebraic manipulation algorithmics is then given. The technique used to convert from 'weight space' to 'latent space' and vise versa is then presented. The mathematic functions performed on neurons and neural networks are then highlighted. The research is then supported by an experimental work section. Then follows a section which discusses the results, plus a sub-section which considers the significance of current research results; this section is expanded in [7]. Finally some concluding remarks are made.

The sister paper [7] to this article includes the following support sections for this article: motivations for this research; and a discussion on use of 'latent space'.

### *B. Brief History of Using Neural Networks to Perform Mathematics*

Neural networks have been used to solve mathematics problems since before the 1980s [8]. The majority of researchers have used neural networks to solve mathematical

0-7803-7278-6/02/$10.00 ©2002 IEEE

problems such as: matrix algebra [9, 10], matrix inversion [8, 11, 12], differential equations [13], and ordinary and partial differential equations [14]. Recently, radial basic function networks have been used to solve differential equations [15]. Most of the above researchers utilised recurrent networks. Other workers in the field used feedforward neural networks [9]. The basic premise of the majority of the research carried out previously was to train the network in order to solve a particular type of mathematical equation. This was normally done by physically mapping parts of the equation to different layers of the net. Our research does not solve particular mathematical equations. Our goal is to perform mathematics with neurons. The neurons were trained to represent numeric value, in this case, but they could be trained to represent any function [7].

### C. *Introduction to the Sigma-pi Neuron Model*

The neural model utilised for this work is termed a sigma-pi neuron model [16-20]. The sigma-pi neurons' functionality viewed as a matrix equation [20] is:

$$Y = \sigma(W_\mu \cdot P_\mu) \tag{1}$$

where, for a single unit, $W$ is a row matrix of the weights $W_\mu = [w_0,...,w_n]$, and $P$ is a column matrix of the probabilities of the weights being addressed $P_\mu = [P_0 \quad ... \quad P_n]^T$ where $\mu \in \{0,...,n\}$ is the index and $\sigma$ is a sigmoid function, $Y = \sigma(U) = 1/1 + e^{-U/\rho}$, given $\rho$ defines the shape of the sigmoid. They are termed sigma-pi units as their functionality has previously been defined by Gurney [16]. The real-valued activation can be defined as:

$$a_{(t)} = \frac{1}{w_m 2^n} \sum_\mu w_\mu \prod_{i=1}^{i=n} (1 + \bar{\mu}_j z_i) \tag{2}$$

where $z_i$, the input probability distribution, defines the probability of the input $x_i$.

The sigma-pi unit's activation is formulated by summing the weights times, $\prod_{i=1}^{i=n}$, the probability of the weight being addressed, given a specific input, $x_i$. The probability is calculated as a product, which aggregates all the units' inputs, $x_1,...,x_n$. Sigma-pi units may be viewed as probabilistic neural networks (PNN) [16-18, 21-24]. Recently, a sub-set of sigma-pi neurons, multi-cube units (MCU), has been used to study information processing in dendrites [25, 26].

### D. *Introduction to Performing Mathematics with Neurons/Nets*

To enable neurons to perform mathematics, two neurons were first labeled with symbolic tags $A$ and $B$. The neurons were then both trained to represent real-valued numeric values. Neurons' $A$ and $B$ weights were then used to prescribe the weights for a new neural unit symbolically tagged $C$, Figure 1. A methodology for performing basic mathematics with neurons or neural networks is now presented.

**TABLE 1. TABULATION OF MATHEMATICAL OPERATIONS CARRIED OUT ON NEURONS AND NEURAL NETWORKS.**

| Function | Mathematical representation | Functions performed in 'latent space' |
|---|---|---|
| Addition | C=A+B | $(\forall w_\mu^C \mid w_\mu^C = w_\mu^A + w_\mu^B)$ |
| Subtraction | C=A-B | $(\forall w_\mu^C \mid w_\mu^C = w_\mu^A - w_\mu^B)$ |
| Multiplication | C=A×B | $(\forall w_\mu^C \mid w_\mu^C = w_\mu^A \times w_\mu^B)$ |
| Division | C=A/B | $(\forall w_\mu^C \mid w_\mu^C = w_\mu^A \div w_\mu^B)$ |
| Squaring | C=(B)^2 | $(\forall w_\mu^C \mid w_\mu^C = (w_\mu^B)^2)$ |
| Square root | C=(B)^1/2 | $(\forall w_\mu^C \mid w_\mu^C = \sqrt{w_\mu^B})$ |

*1) Overview of Algebraic Manipulation Algorithmics:* The basic premise is that knowledge (e.g. information) encapsulated in trained neural networks may be reused. The information is reused in order to perform algebraic manipulations of the functions learnt by the previously trained net. In order to prescribe a new set of weights, a set of steps needs to be followed in order to calculate the new set of weights.

2) *Algebraic Manipulation Algorithmics*: The steps necessary to carry out mathematical operations using sigma-pi neurons are listed below:

**Step 1)** One neuron is trained to represent a value and then allocated a symbol or letter (i.e. $A$).

**Step 2)** Another neuron is trained to represent another value and then allocated a symbolic character (i.e. $B$).

**Step 3)** The weights of the two neurons $A$ and $B$ are transformed from 'weight space' to 'latent space'.

**Step 4)** The two sets of weights in 'latent space' are subjected to a mathematical function, re. Table 1.

**Step 5)** The resultant weights from the function manipulation are then transformed back to 'weight space'.

**Step 6)** The weights are then assigned to another neuron and assigned a symbol (i.e. $C$).

Note: operations 1) and 2) may be carried out in parallel if $A$ and $B$ are output units of a multi-layer neural network. Steps 3 and 4 require that the weights be transformed. Firstly, it is necessary to convert from 'weight space' to 'latent space', then back to 'weight space' after the mathematical operations have been performed on the weights in 'latent space'. The following paragraphs state how this conversion was achieved.

3) *Converting from 'Weight Space' to 'Latent Space'*: To convert from 'weight space' to 'latent space' the following transformation was performed:

$$w_{\mu}^{'} = 1/2\,(w_{\mu}/w_{m} + 1) \qquad (3)$$

$w_{\mu}$ are the weight values in 'weight space', $w_{m}$ is the maximum weight possible, and $w_{\mu}^{'}$ are the weight values in 'latent space'. In the case of the linear output neuron model the following matrix transformations are performed:

$$\{w_0, \ldots, w_n\} \Rightarrow \{w_0^{'}, \ldots, w_n^{'}\} \qquad (4)$$

4) *Converting from 'Latent Space' to 'Weight Space'*: Once the basic mathematical operation had been carried out in 'latent space', the weights were then transformed back from 'latent space' to 'weight space' using:

$$w_{\mu} = w_{m}(2w_{\mu}^{'} - 1) \qquad (5)$$

In the case of the linear output neuron model, the following matrix transformations are performed:

$$\{w_0^{'}, \ldots, w_n^{'}\} \Rightarrow \{w_0, \ldots, w_n\} \qquad (6)$$

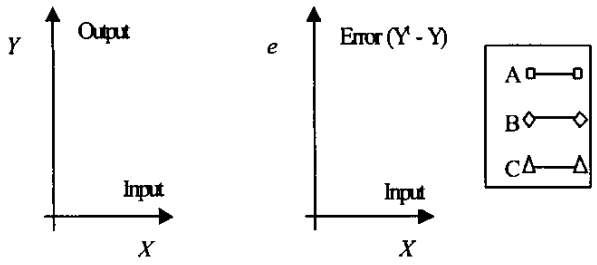



**FIGURE 2. AXIS INFO RMATION FOR GRAPHS DISPLAYED IN THE FOLLOWING EXPERIMENTAL WORK.**

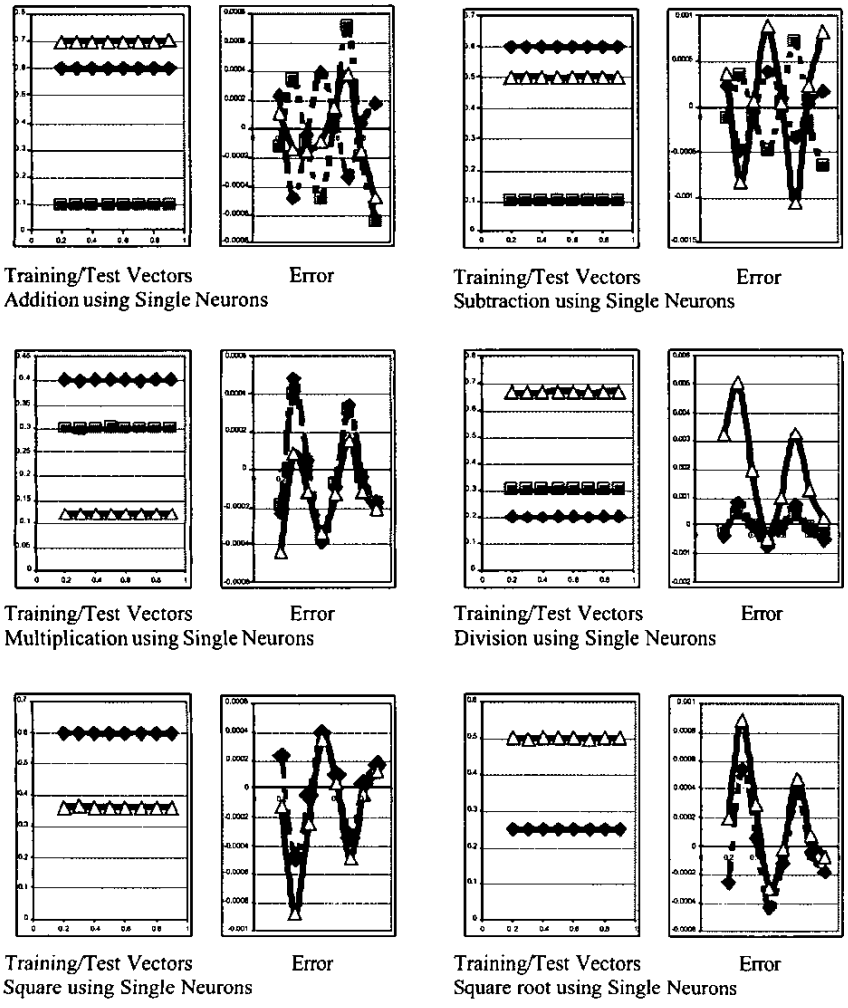



**FIGURE 3. SINGLE NEURONS MATHEMATICAL COMPUTATIONS.**

*5) Mathematic Functions Performed on Neurons and Neural Networks*:

The mathematical functions which the two sets of weights in 'latent space' perform are presented in Table 1.

The left hand column of Table 1 defines the mathematical function, the central column defines the symbolic mathematical calculation, and the right hand column states the functions performed in 'latent space' in order to perform the required mathematic operation.

## II. EXPERIMENTAL WORK

In the following experimental work, two symbolic tags are associated to two neurons. The two neurons are then trained to represent two different numeric values. Once trained, the two neurons are used to prescribe the weight vectors of a third neuron. The calculation is an algebraic calculation which is a function of the other two neurons. This can be viewed as mapping information in $A$ and $B$ to $C$, and hence:

$$A, B \mapsto C = f(A, B) \qquad (7)$$

### *A. Experimental Delimitations*

In order to test the conjecture that basic mathematics can be

performed with neurons, a set of mathematic calculations depicted in Table 1 are carried out. Two different network structures are used. The first experiment uses a set of single neurons. Each of the neurons has 8-inputs and maps $(x|x \in \{x_1,...,x_n\}) \rightarrow y$, where $x$ is a set of inputs which map to a single output. The second is a multi-layer 1-8-3 feedforward neural network. The two-layer net has a single input which is connected to eight 1-input hidden units which are then connected to three 8-input visible units, one for each output $A$, $B$, and $C$. Note that $A$ and $B$ are trained, and $C$'s weights are prescribed. Again, the two-layer net maps $(x|x \in \{x_1,...,x_n\}) \rightarrow y$. Note that when transformations are carried out on networks, only the output unit weight matrices of $A$ and $B$ are used to prescribe $C$'s weight matrix.

The single neurons were trained for 1000 epochs with the following backpropagation rule learning rate and momentum settings: $\alpha = 50.0$, $\lambda = 0.4$; its outputs were linear activation-output functions. The multi-layer neural network was also trained for 1000 epochs with the following backpropagation learning rate and momentum settings: $\alpha = 2.0$, $\lambda = 0.9$. The hidden units used sigmoidal activation-output functions with a $\rho = 0.001$ and the output unit/s were linear activation-output functions. Normally, the range of the weights is set to $-w_m \leq w_\mu \leq +w_m$, where $w_m = \max(w_\mu)$ is the maximum weight value, $w_m = 1$. This enables the unit to output real-valued outputs $Y \in [0,1]$.

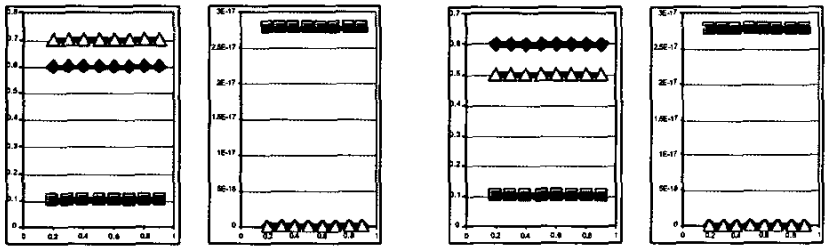
To compare the errors of each of the trained units and the unit with the prescribed weights, an average error modulus (EM) was calculated:

$$\overline{|e|} = 1/n \sum_{p=1}^{p=n} \sqrt{(y_i^t - y_i)^2} \qquad (8)$$

where $n$ was the number of training or test vectors, $y_i^t$ was the target output and $y_i$ was the actual output.

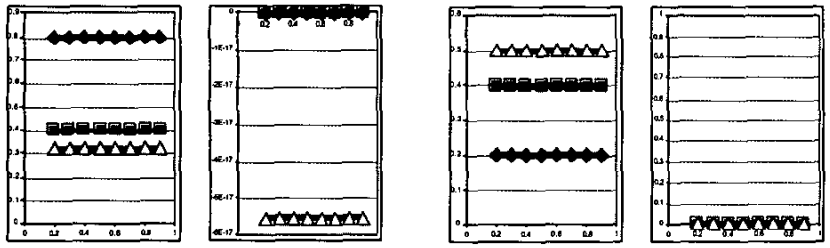
Each neuron/net was trained with a set of input/output pattern pairs, $p$. The two single neurons were trained with eight input-output pattern pairs $\{0.2,0.3,\ 0.4,0.5,\ 0.6,0.7,\ 0.8,0.9\}^{INPUT} \Rightarrow \{Y\}^{OUTPUT}$ which were equally distributed across the input. The training patterns were presented sequentially to the network, and each pattern was selected at random from the set of training vectors. The networks were trained with eight input-output pattern pairs $\{0.2,0.3,\ 0.4,0.5,0.6,\ 0.7,0.8,\ 0.9\}^{INPUT} \Rightarrow \{Y\}^{OUTPUT}$ which were equally distributed across the input. In all cases the different input patterns map to a single numerical output value $y$.

The results are presented as pairs of graphs. The graph on the left hand side plots the nodes output, $Y$, on the vertical axis against the nodes input on the horizontal axis. The right hand graph plots the nodes error, $y^t$-$y$, on the vertical axis against the nodes input, $x$, on the horizontal axis. Each graph displays three sets of results, each result is coded with a graphical symbol. The symbols for neurons $A$, $B$, and $C$ were: a square, a diamond, and a triangle. A diagram depicting a typical pair of graphs with associated graphical symbols



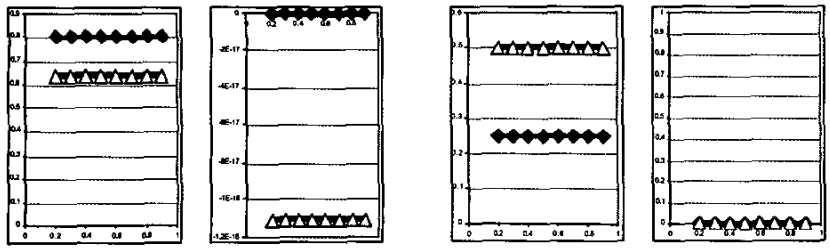
Training/Test Vectors Error
Addition using Neural Networks

Training/Test Vectors Error
Subtraction using Neural Networks



Training/Test Vectors Error
Multiplication using Neural Networks

Training/Test Vectors Error
Division using Neural Networks



Training/Test Vectors Error
Square using Neural Networks

Training/Test Vectors Error
Square root using Neural Networks

**FIGURE 4 MULTI-LAYER NEURAL NETWORKS MATHEMATICAL COMPUTATIONS.**

for $A$, $B$, and $C$ is shown in Figure 2.

*B. Experimental Work - Using Neurons and Networks*

The experiments which were carried out utilised two different neural network architectures. The two sections below present the results for the two different topologies: a set of single neurons and a multi-layer feedforward neuron network. Table 1 depicts the calculations carried out in 'latent space'.

*1) Single Neurons:* To test the conjecture that basic mathematics can be carried out using single neurons, three neurons tagged $A$, $B$ and $C$ were used. Each of the mathematical calculations defined in Table 1 were tested using the algorithmic approach stated above. For example, neuron $A$ was trained to output 0.1, and neuron $B$ was trained to output 0.6. After following algorithmic steps 3 to 6, C's output should be 0.7 if neuron $C$ was used to calculate $C=A+B$ then $C=0.7=0.1+0.6$. Figure 3 depicts the results of performing all the mathematical calculations tabulated in Table 2.

*2) Neural Networks:* In this set of experiments, the conjecture that basic mathematics can be carried out using a set of output units of a multi-layer feedforward neural network was tested. The three output units were tagged $A$, $B$, and $C$. Each of the mathematical calculations defined in Table 1 was tested using the algorithmic approach stated above. For example, output unit $A$ was trained to output 0.3, and output unit $B$ was trained to output 0.1. After following algorithmic steps 3 to 6, $C$'s output should be 0.2 if neuron $C$ was used to calculate $C=A-B$ then $C=0.2=0.3-0.1$. Figure 4 depicts the results of performing all the mathematical calculations tabulated in Table 3.

## III. DISCUSSION OF RESULTS

The results from the experimental work have been presented in a graphical form in Figures 3 and 4 and in a tabular form in Tables 2 and 3. The graphical presentation of the results highlights the fact that the single neuron units do not achieve as low an error modulus, $|\overline{e}|$, as the networks.

The results also illustrate that unit $C$'s error is a function of units $A$ and $B$'s errors. If the error plot, Figure 3 – Addition, is viewed, it shows that $e^C$'s error is approximately the sum of the error of $e^A$ plus the error of $e^B$.

This means that $e^C$'s error is approximately the addition of $e^A$'s and $e^B$'s error, which correlates with the addition of $A$ and $B$. The error plot shows that $e^A$'s and $e^B$'s error plots are approximately the inverse of each other, hence it is not too surprising that $e^C$'s error is less than both $e^A$'s and $e^B$'s. This conclusion can be drawn from the results of addition, subtraction and multiplication in Figure 3. However, the division function does not reduce the error, the square function does not square the error and the square root function does not reduce the error. It should be noted that mathematical functions performed by neural networks have extremely low errors, so neural networks, rather than single neural units, should be utilised when performing mathematical computations.

**TABLE 2**
**SINGLE NEURON RESULTS, DEPICTS THE UNITS' ERROR MODULUS, MEAN ERROR MODULUS AND COMPARES C's ERROR MODULUS WITH THE MEAN ERROR MODULUS.**

| Function | Mathematical Representation | Error Modulus $\overline{\lvert e^A \rvert}$ | Error Modulus $\overline{\lvert e^B \rvert}$ | Error Modulus $\overline{\lvert e^C \rvert}$ | Mean Error Modulus $\overline{\lvert e \rvert}$ |
|---|---|---|---|---|---|
| Addition | $C=A+B$ | 0.000299 | 0.000213 | 0.000195 | 0.000236 |
| Subtraction | $C=B-A$ | 0.000299 | 0.000213 | 0.000489 | 0.000334 |
| Multiplication | $C=A*B$ | 0.000213 | 0.000186 | 0.000202 | 0.0002 |
| Division | $C=B\div A=B/A$ | 0.000186 | 0.000413 | 0.002341 | 0.00098 |
| Squaring | $C=B^2$ | | 0.000213 | 0.000314 | 0.000264 |
| Square root | $C=\sqrt{B}=B^{1/2}$ | | 0.000238 | 0.000322 | 0.00028 |
| Mean error modulus of all functions | | 0.000249 | 0.000246 | 0.000644 | 0.000382 |

**TABLE 3**
**MULTI-LAYER NETWORK RESULTS, DEPICTS THE UNITS' ERROR MODULUS, MEAN ERROR MODULUS AND COMPARES C's ERROR MODULUS WITH THE MEAN ERROR MODULUS.**

| Function | Mathematical Representation | Error Modulus $\overline{\lvert e^A \rvert}$ | Error Modulus $\overline{\lvert e^B \rvert}$ | Error Modulus $\overline{\lvert e^C \rvert}$ | Mean Error Modulus $\overline{\lvert e \rvert}$ |
|---|---|---|---|---|---|
| Addition | $C=A+B$ | 0 | 2.78E-17 | 0 | 9.27E-18 |
| Subtraction | $C=B-A$ | 2.47E-17 | 0 | 0 | 8.23E-18 |
| Multiplication | $C=A*B$ | 0 | 0 | 4.93E-17 | 1.64E-17 |
| Division | $C=B\div A=B/A$ | 0 | 0 | 0 | 0 |
| Squaring | $C=B^2$ | | 0 | 1.11E-16 | 5.55E-17 |
| Square root | $C=\sqrt{B}=B^{1/2}$ | | 0 | 0 | 0 |
| Mean error modulus of all functions | | 8.23E-18 | 4.63E-18 | 2.67E-17 | 1.49E-17 |

*A. Significance of Current Research Results*

This article has augmented our original research [27] which showed how to reflect, invert and scale neural networks' output by reflecting, inverting, and scaling the neural units' or nets' weights. It has laid down the basic theory

which enables trained networks to be used to prescribe weights for other networks. The new weights are prescribed by the aggregation of two or more trained units' weights. The calculations are carried out utilising the weights of trained units/nets to prescribe a new set of weights. In this paper we have shown that this can be done with units/nets which map single-valued outputs. Although these research results may not seem revolutionary, they enable us to begin research into new paradigms for artificial neural networks. The new paradigm reuses information encapsulated in trained artificial neural units/networks to prescribe the weights of new units/networks.

## IV. CONCLUSION

In this article we have laid down the theory to enable a type of mathematical algebra using neural units and networks as symbolic entities to be performed. The basic mathematical manipulation algorithm is detailed above. To perform basic mathematics with neurons or neural networks, the weights must first be cast into a secondary space (c.f. the output units' weights in the case of a multi-layer artificial neural network), before any algebraic calculations can be performed. The secondary space is a re-scaled space we term 'latent space' where the transformed data variables are sometimes called "hidden variables" [28]. These hidden variables can be transformed utilising a set of mathematical operators, tabulated in Table 1. The use of 'latent space' is analogous to many other types of mathematical transformations, such as Fourier transformations.

The important fact which this paper presents is that knowledge encapsulated in the two trained neurons or output units of a multi-layer artificial neural network can be manipulated to enable another unit or output unit of a network to perform other related tasks, in this case functions of the other two.

This article has developed methodologies which show how to perform mathematical calculations such as addition, subtraction, multiplication, division, square and square root with symbols which were instantiated as neurons or output units in a multi-layer feedforward neural network.

### A. References


[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Science of the USA*, vol. 79, pp. 2554-2558, 1982.

[2] R. Sun, "Hybrid connectionist modules," in *AI Magazine*, vol. 17, 1996, pp. 99-103.

[3] R. Sun, E. Merrill, and T. Peterson, "Skill learning using a bottom-up hybrid model," presented at The First International Conference on Cognitive Science, Seoul, Korea, 1997.

[4] R. Sun and T. Peterson, "A hybrid model for learning sequential navigation," presented at Procedding of IEEE International Symposium on Computational Intelligence in Robotics and Automation, Monterey, CA, 1997.

[5] R. Sun and T. Peterson, "Some experiments with a hybrid model for learning sequential decision making," *Information Sciences*, vol. 111, pp. 83-107, 1998.

[6] R. Sun, T. Peterson, and E. Merril, "A hybrid architecture for situated learning of reactive sequential decision making," *Applied Intelligence*, vol. 11, pp. 109-127, 1999.

[7] R. Neville, "Performing mathematics with neurons or neural networks trained to represent non-linear functions: Part II," presented at WCCI-2002 World Congress on Computational Intelligence, Hawaii, 2002.

[8] J. L. S. Jang, S.-Y. Lee, and S. Shin, "An optimization network for matrix inversion," presented at Neural Information Processing Systems, 1988.

[9] L. Wang and J. M. Mendel, "Structured trainable networks for matrix algebra," *IEEE International Joint Conference on Neural Networks*, vol. 42, pp. 125-128, 1990.

[10] L. Wang and J. M. Mendel, "Parallel structured networks for solving a wide variety of matrix algebra problems," *Journal of Parallel and Distributed Computation*, vol. 14, pp. 236-247, 1992.

[11] F. L. Luo and B. Zheng, "Neural network approach to computing matrix inversion," *Applied Mathematics and Computation*, vol. 47, pp. 109-120, 1992.

[12] J. Wang, "A recurrent neural network for real-time matrix inversion," *Applied Mathematics and Computation*, vol. 5, pp. 89-100, 1993.

[13] A. Cichocki and R. Unbehauen, "Neural networks for solving systems of linear equations and related problems," *IEEE Transactions on Circuits and Systems*, vol. 39, pp. 124-138, 1992.

[14] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, pp. 987-1000, 1998.

[15] N. Mai-Duy and T. Tran-Cong, "Numerical solutions of differential equations using multiquadric radial basis function networks," *Neural Networks*, vol. 14, pp. 185-199, 2001.

[16] K. Gurney, "Learning in nets of structured hypercubes," in *Department of Electrical Engineering*. Uxbridge: Brunel, 1989.

[17] K. N. Gurney, "Training nets of hardware realisable sigma-pi units," *Neural Networks*, vol. 5, pp. 289-303, 1992.

[18] K. Gurney, "Weighted nodes and RAM-nets: A unified approach," *Journal of Intelligent Systems*, vol. 2, pp. 155-185, 1992.

[19] R. S. Neville and T. J. Stonham, "Adaptive critic for sigma-pi networks," *Neural Networks*, vol. 9, pp. 603-625, 1996.

[20] R. S. Neville, T. J. Stonham, and R. J. Glover, "Partially pre-calculated weights for the backpropagation learning regime and high accuracy function mapping using continuous input RAM-based sigma-pi nets," *Neural Networks*, vol. 13, pp. 91-110, 2000.

[21] J. Austin, "A review of RAM based neural networks," presented at Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, Turin, Italy, 1994.

[22] T. Clarkson, D. Gorse, J Taylor, and C. Ng, "Learning probabilistic RAM nets using VLSI structures," *In IEEE Transactions on Computers*, pp. 1552-1561, 1992.

[23] D. K. Milligan, "Annealing in RAM-based learning networks," Brunel University, Uxbridge, West London, Middlesex, U.K., Technical Memorandum CN/R/142, 1988.

[24] A. Ferguson, L. C. Dixon, and H. Bolouri, "Learning algorithms for RAM-based neural networks," *Annals of Mathematics & Artificial Intelligence*, 1996.

[25] K. Gurney, "Information processing in dendrites: I. Input pattern generalisation," *Neural Networks*, vol. 14, pp. 991-1004, 2001.

[26] K. Gurney, "Information processing in dendrites: II. Information theoretic complexity," *Neural Networks*, vol. 14, pp. 1005-1022, 2001.

[27] R. S. Neville, "Toward second-order generalisation," presented at IEEE World Congress on Computational Intelligence, Anchorage, Alaska, 1998.

[28] B. S. Everitt, *An Introduction to Latent Variable Models*: London: Chapman and Hall, 1984.