Annals of Emerging Technologies in Computing (AETiC)
Vol. 3, No. 4, 2019

Research Article

Numerical Discrimination of the Generalisation Model from Learnt Weights in Neural Networks

Richard N M Rudd-Ortner[1,2,*] and Lyudmilla Milhaylova[1]

[1]Department of Automatic Control and Systems Engineering, University of Sheffield, UK
RNMRudd-Orthner1@sheffield.ac.uk; L.S.Milhaylova@sheffield.ac.uk
[2]MASS KSA (a Cohort plc company), Riyadh, Kingdom of Saudi Arabia,
rruddorthner@mass.co.uk
*Correspondence: ruddorthner@gmail.com

Received: 1st September 2019; Accepted: 17th September 2019; Published: 1st October 2019

Abstract: This research demonstrates a method of discriminating the numerical relationships of neural network layer inputs to the layer outputs established from the learnt weights and biases of a neural network's generalisation model. It is demonstrated with a mathematical form of a neural network rather than an image, speech or textual translation application as this provides clarity in the understanding gained from the generalisation model. It is also reliant on the input format but that format is not unlike an image pixel input format and as such the research is applicable to other applications too. The research results have shown that weight and biases can be used to discriminate the mathematical relationships between inputs and make discriminations of what mathematical operators are used between them in the learnt generalisation model. This may be a step towards gaining definitions and understanding for intractable problems that a Neural Network has generalised in a solution. For validating them, or as a mechanism for creating a model used as an alternative to traditional approaches, but derived from a neural network approach as a development tool for solving those problems. The demonstrated method was optimised using learning rate and the number of nodes and in this example achieves a low loss at 7.6e-6, a low Mean Absolute Error at 1e-3 with a high accuracy score of 1.0. But during the experiments a sensitivity to the number of epochs and the use of the random shuffle was discovered, and a comparison with an alternative shuffle using a non-random reordering demonstrated a lower but comparable performance, and is a subject for further research but demonstrated in this "decomposition" class architecture.

Keywords: ***Weight capture; Information Assurance; Safety-Critical AI; Decomposition Rule-Extraction***

1. Introduction

In terms of mission and safety critical applications, using neural networks is a challenge in confidence [1]. Part of those challenges may be establishing if the neural network learning is complete

Richard N M Rudd-Ortner and Lyudmilla Milhaylova, "Numerical Discrimination of the Generalisation Model from Learnt Weights in Neural Networks", *Annals of Emerging Technologies in Computing (AETiC)*, Print ISSN: 2516-0281, Online ISSN: 2516-029X, pp. 1-14, Vol. 3, No. 4, 1st October 2019, Published by International Association of Educators and Researchers (IAER), DOI: 10.33166/AETiC.2019.04.001, Available: http://aetic.theiaer.org/archive/v3/v3n4/p1.html.

and has suitable regularisation [2] to cover new or unexpected inputs and optimising [3]. Should unexpected inputs occur then there may be a commitment for a safety or mission critical outcome to be made with verification and validation [4]. A decomposition approach to this is to understand the weights and biases and their semantic meaning with respect to the inputs. Such that the generalisation model can be understood and perhaps fail states or indeterminism can be identified and have tailored safe outcomes. Commonly neural network's learnt models are considered to be complex and not interpretable [5], but this paper demonstrates a method of taking understanding from the weights. There has also been work in this area and papers reviewing approaches of algorithms are cited [6, 7, 8] and some of these approaches use Recurrent Neural Network (RNN) and Multi Layer Perceptron (MLP) types. However, this paper will demonstrate a clear case for analysis approach, for insights in further work.

The paper outlines an approach for a clear case in an analysis and basic architecture in section two. In section three a numerical representation is presented. In section four the dataset using that numerical representation from section three is described. In section five the input numerical representation is used with the datasets and performs some optimisation to increase the accuracy in the prediction. In section six the biases and weights are examined with the initial single parameter experiment. In section seven the discrimination action is shown using two parameters with a divide operator. Section eight then includes a further parameter that is an addition operator and re-demonstrates the discrimination action. Section nine presents findings, Section eleven optimises the model with learning rate and number of nodes experiments. Section ten makes an experiment with an alternative shuffle algorithm. The final conclusions are drawn in section twelve.

2. The Approach and Architecture

The aim is to understand what the generalisation model has learnt and use an input numerical representation that will allow discrimination of the relationships from inputs within the weights and biases. The approach is to gain the clarity of understanding of those relationships as a clear case for analysis using a deliberately simplistic architecture that will use minimal complexities. The model application is numerical and mathematical operators such that the abstraction of the application area is simplistic and generic rather than an abstraction from imagery, audio or grammatical context specifically. This will also have the advantage of the understanding of the representation and learnt model contributions not being complicated by additional application effects like recording noise or quantisation of which those effects can be added or specifically understood within those application areas.

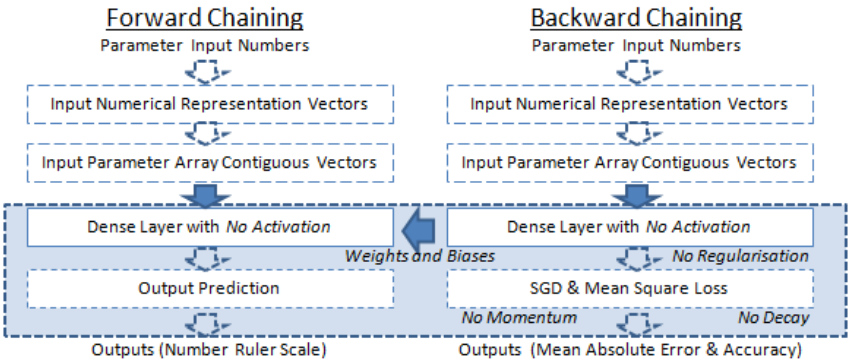

Figure 1. Simplistic Architecture for Clear Case for Analysis with a Single Layer

Although this method is anticipated to be applied to a number of layers in succession as a decomposition approach, for simplicity and clarity of understanding in this paper a single layer Neural Network will be used with the number of neurons equal to the number of input vector array length *(vectorLength)* at the outset of the analysis, and there is also no activation function used. The weights are initialised to the value 1.0 and the biases to 0.0. There is no regularisation function as we want to over fit a model from pure data samples to get the best estimate of the weights in the final learnt weights used for the prediction values to demonstrate the method clearly. The loss is a mean

square loss and the optimiser is Stochastic Gradient Descent with a learning rate set to 1.0, but will be experimented with later, also there is no momentum or decay. The metrics are mean absolute error and accuracy. The architecture is shown in Fig 1 diagrammatically.

3. The Input Representation

An input format is required to represent numerical values and the form that this method has used is a number line array where each address in that array represents a linier number increase like a ruler. The numerical resolution value of each array address has a fixed uniform step, but to capture values with sub-resolution values between the array position addresses, then a proportional value scaling of the two position addressee's values is used and the combined value between them is one. That is to say the values in the array are ratios of subsample positions between two addresses. Actually this is not unlike a flattened image of pixels, were the pixel position represents a scalar value instead of a relative positional placement, and the pixel value of the pixel represents a ratio of proportional value in a number line instead of an intensity. This format also is reversible back to a numerical value using a centre of gravity process.

3.1. Coding a Value into the Representation

To set a value in this form as an input array then the following representation is used. This representation is a function that will take these inputs:

- The value to be represented *(TheValue)*,
- The number range's maximum and minimum possible values *(RangeMax* and *RangeMin)*,
- The vector array to be used *(VectorArray)*,
- The length of the vector array in addresses *(VectorLength)*.

The output from this function is the vector array but populated with a value (or values) in the addresses that represent the scalar value. Fig 2 is the more formal definition, but also see Appendix A for a pseudo code version of this definition:

$$res = \left(\frac{(\boldsymbol{Range_{max}} - \boldsymbol{Range_{min}})}{(\boldsymbol{VectorLength} - 1)}\right)$$

$$offest\_res = \left(\frac{\boldsymbol{TheValue} - \boldsymbol{Range_{min}}}{res}\right)$$

$$pos1 = \lfloor offest\_res \rfloor \qquad pos2 = pos1 + 1$$

$$value2 = (offest_{res}) - pos1 \qquad value1 = 1.0 - value2$$

$$\boldsymbol{VectorArray}[Pos1] = value1 \quad \boldsymbol{VectorArray}[Pos2] = value2$$

Figure 2. Encoding Format Formal Definition

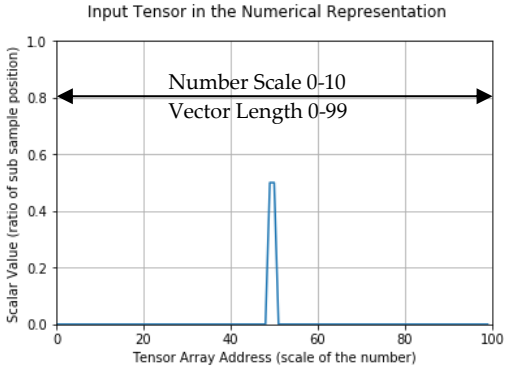

Figure 3. Example Input Tensor for the Value 5 in a scale 0-10

Fig 3 is an example of an encoded input tensor that is a representation of the value 5 in a scale 0-10. The vector array length is 100 addresses (0-99) and each address has a sample resolution of 0.1010 recurring. Therefore, the array positions 49 and 50 have been set to the scalar value of 0.5 and

that expresses the subsample position ratio between those two samples that it is an equal distance between them.

3.2. Decoding a Value from the Representation

To recover the scalar value from the vector array is a centre of gravity process. As a function, this function returns the scalar value *(TheValue)* and from the following inputs:

- Input vector array *(VectorArray)*,
- The length of the vector array in addresses *(VectorLength)*:
- The number range maximum and minimum possible values *(RangeMax* and *RangeMin)*

In Fig 4 is the more formal definition, but also see Appendix A for a pseudo code version of this definition:

$$res = \left(\frac{(\boldsymbol{Range_{max}} - \boldsymbol{Range_{min}})}{(\boldsymbol{VectorLength} - 1)}\right)$$

$$number\_line = (\{0,1 \ldots \boldsymbol{VectorLength}\} * res + \boldsymbol{Range_{min}})$$

$$weighted_{line} = (number\_line * \boldsymbol{VecrorArray})$$

$$\boldsymbol{TheValue} = \left(\frac{\sum weighted\_line}{\sum number\_line}\right)$$

Figure 4. Decoding Format Formal Definition

These two functions therefore provide encoding and decoding methods that allow numerical scalar values to be represented into a neural network. It is also in a form that is consistent with imagery processing applications of neural networks. However, because the input tensor and output prediction are understandable it also provides a basis for clear case for analysis of how and what the weight and biases have represented.

4. The Dataset

In this example the numerical representation is used for each value in a vector length of 100 addresses, and there are 100,000 vectors as a matrix. The number representation vector resolution is 0.1010 reoccurring and the dataset size is 100,000 numbers between that range of 0-10 which means that there is 100,000 number representation vector values between 0-9.9999 in steps of 0.0001. Fig 5 shows the matrix dataset representation as an image.

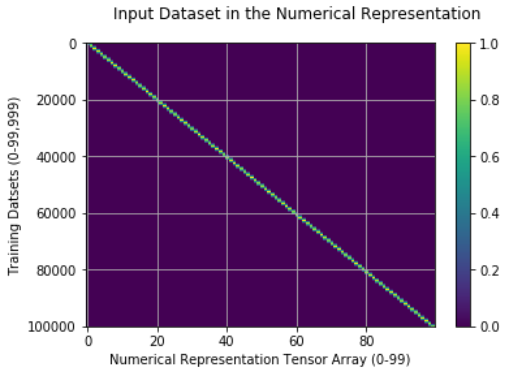

Figure 5. Dataset Matrix Expressed as a Pixel Image

5. The Initial Experimental Use

As an example, using a simple function: $y = sin(x) * x$ where y is the desired output and x is the input, then in compatibility with Keras a set of input tensors can be constructed for the different input values using the input numerical representation, and form a matrix of values were the input from each learning dataset data item is sequenced. The output y training category is the list of expected scalar values. At the outset the values are in numerical order, as shown in Fig 6.

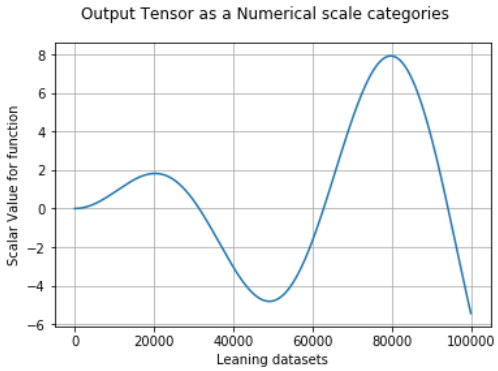

Figure 6. Output Expected y Value Categories

However, when the model is fitted it may appear that the learning order may be important and perhaps has an implication for the learning rate and the order that nodes are updated. Fig 7 graphs the input tensors in blue, in the numerical representation for each learning value in the dataset. In red is the expected output from the example *sin(x) * x* function, and in green dashed is the output model prediction after model learning.

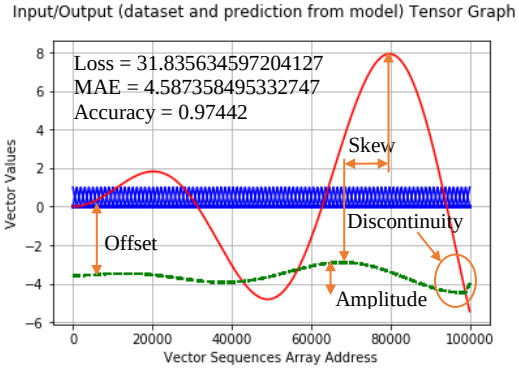

Figure 7. Input, Output and Model Prediction without Shuffle

Fig 7 shows that there is a trend in the prediction (green dashed) towards the expected output (red). However, the prediction is offset in value centring, has a relative skew and has an amplitude loss. It maybe that the skew is a filter effect implied by ordering in the dataset that creates a period of update constructively and destructively during the learning process as the strong mapping of inputs to outputs sweeps the values in a direction. To test this an experiment with the shuffle enabled is conducted as this will affect the updating direction for individual nodes in a less deterministic order. Fig 8 is the same model with the shuffle enabled, note the edge discontinuity is largely unaffected.

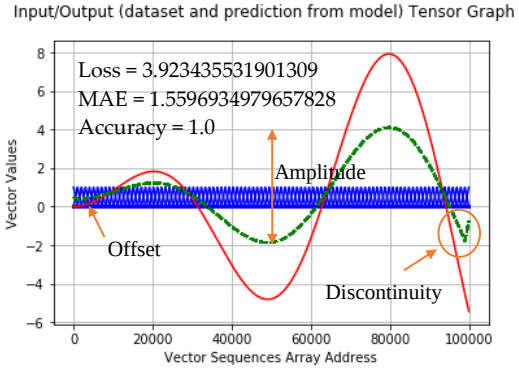

Figure 8. Input, Output and Model Prediction with Shuffle

In Fig 8 there is a stronger trend, minimal skew and the offset bias as reduced, so it may be that the skew is being caused by the numerical sweep direction of the updates. However, the amplitude

is still reduced and the learning rate or the number of updates may need to be increased. To test this, 10 epochs are used to cause the nodes to be updated more and results in better fitting, and also reduces the discontinuity (as shown in Fig 9).

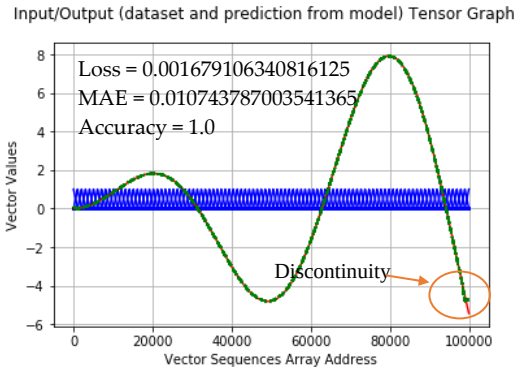

Figure 9. Input, Output, Model Prediction with Shuffle 10 epochs LR=1

In order that the learning rate and epochs can be discriminated that same experiment is re-run with a single epoch but with the learning rate set to 10.0 instead of 1.0, and the results are shown in Fig 10.

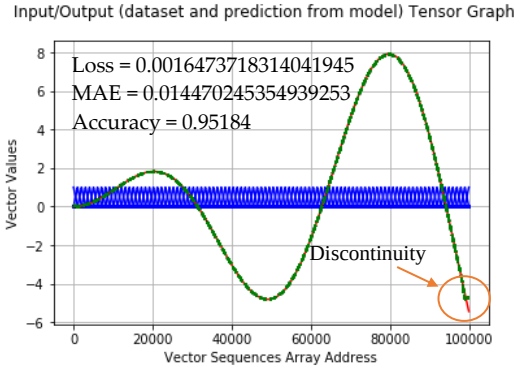

Figure 10. Input, Output, Model Prediction with Shuffle 1 epoch LR=10

Fig 10 is visually identical to Fig 9 and this indicates that the learning rate can be increased to reduce epochs and vice versa. However, the discontinuity is unaffected, but the 10 successive shuffles did have a ~4% discernible extra benefit to accuracy, which implies that it is not learning rate (LR) or epochs alone but the number of updates times their influence combined. However, the extra epochs reorder shuffles also did have an extra benefit beyond the visually discernible, and perhaps further minimises the skew.

6. Weight and Bias Representation

Now that the prediction from the model is accurate within a model that has understandable input to output determinism. The internal weight and bias representations can be examined, and Fig 11 is using the highest scoring settings of 10 epochs and the learning rate (LR) equalling 1.

In the weight tensor in Fig 11 it can be seen that the output prediction mapping in Fig 9 is almost identical to the prediction including the minor edge discontinuity, although the weights in Fig 11 has an offset in value. Note that the maximum value in the prediction was 8 and is over that in the weights and is captured in the biases in Fig 12 as a -0.24461249 offset which is representative of the weight offset observed in Fig 11.

Now that the weights and biases are correlating with the prediction and expected results, except for a minor discontinuity in the weights, also as the prediction is accurate and the problem solution is deterministically understandable, more input parameters can be introduced.

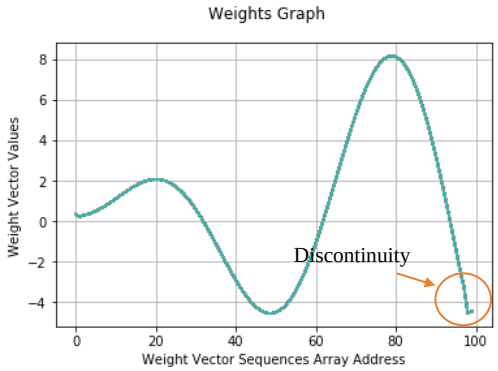

Figure 11. Weight Tensor after Fitting

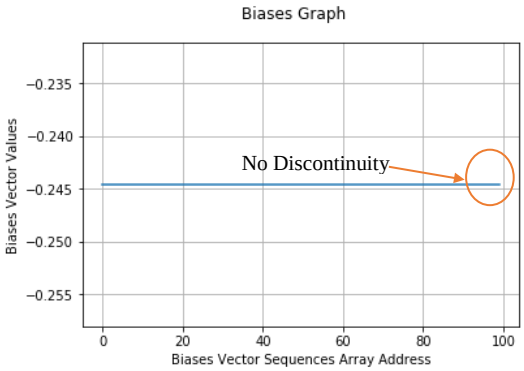

Figure 12. Bias Tensor after Fitting

7. Discrimination Action

As demonstrated a mathematical function can be represented in a neural network where that function can clearly be understood and observed in the weights so a second input parameter is added. The input parameter is introduced as a second vector input array in the same input numerical representation encoding, but added to the end of the first parameters tensor like a second colour in an image. Although using contiguous array values rather than interleaved values, as might be in colour images. This is so the weights and biases may be readily understood as two contiguous array vectors combined rather than having a interleaving relationship when plotted. Although conceptually that should make little difference to the processing just to the representation in the weights and biases. However, it is that representation that will be used to deuce the mathematical discrimination of mathematical relationships. The function is updated to take inputs x, z and output y, and the function is $y = sin(x)*z$ where $z=x/2$. Then an example of an input tensor is shown in Fig 13:

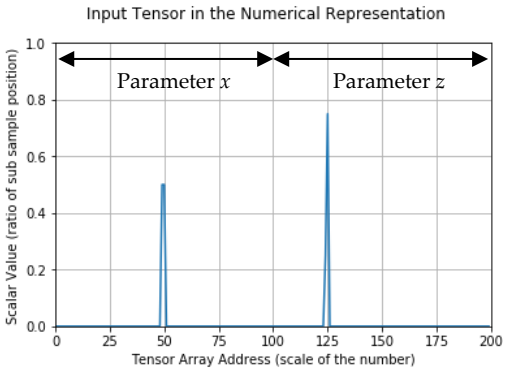

Figure 13. Input Tensor Example with Two Parameters

Fig 13 is a single input tensor example, and the tensor is 200 addresses long twice the original single parameter tensor length. The first parameter (x) is again the value 5.0 and remains in the scale

0-10. The second parameter's value is 2.5 as per the function of z. Both parameters are encoded in the numerical representation and the second parameter is half the distance from the 100 address then the first parameter is from the 0 address. This positioning is important as it will be represented in the weights and biases. The complete dataset as an image is in Fig 14 with both parameters. Note they don't overlap for clarity.

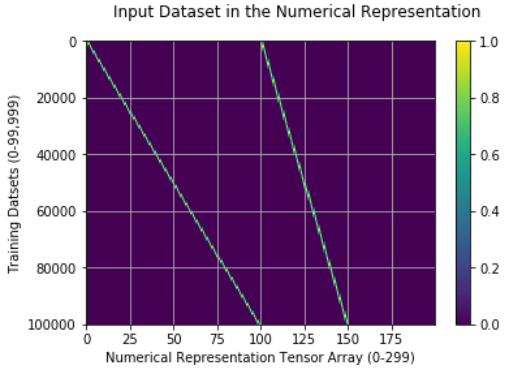

Figure 14. The Dataset as an Image with Two Parameters

The training dataset graph of the outputs numerical categories is in Fig 15 and is similar to the original with one parameter and is of the same length, as the number of output categories is unchanged regardless of the addition of an extra input parameter. This is because the extra parameter was added as if it was an extra image colour channel to the contiguous input vector array.

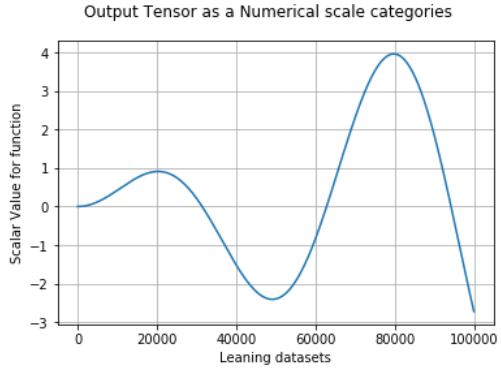

Figure 15. Expected Output with the Two Parameter Function

In Fig 16 is the model prediction (green dashed), inputs (Blue) and the expected correct output (red) and again the prediction from the model is accurate and the red line is almost eclipsed by the dashed green line prediction, apart from a small minor discontinuity at the edge, but appears to have reduced with an extra parameter that is causing more neuron updates.

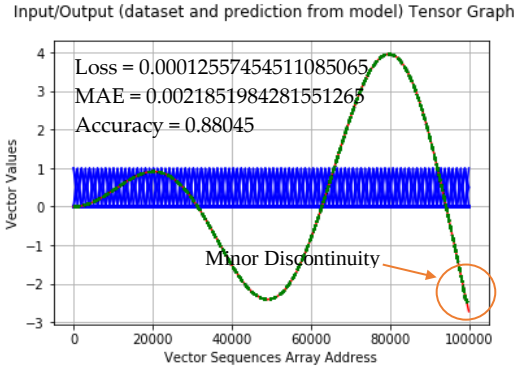

Figure 16. Input, Output and Model Prediction with Two Parameters

The weights tensor using the two parameters representation dataset shows the relationship of the inputs: x and z to y individually and are shown in Fig 17 as the weight vector tensor.

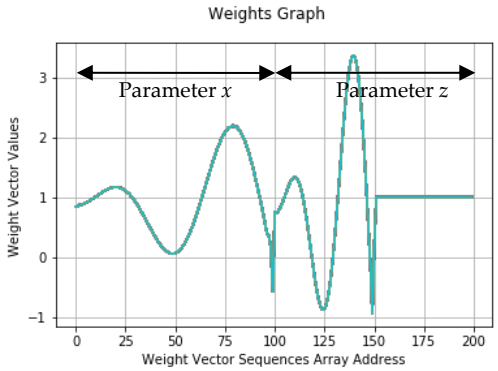

Figure 17. Weight Tensor from Two Parameters

Visually in Fig 17 it can be seen that the second parameter's weight vector length used reflects the first parameter x, but uses only half the number range in the vector as $z=x/2$. That means it used half the vector length number range (and compressed the representation relative to x) as per the numerical representation used in the input. Therefore the relationship between x and z is a scale of 2 where z is half the scale of x and thus half the vector length was used in the input representation. The same concept would be true for multiply operators although it would be a longer z parameter number range (and be expanded) to capture the scale of z, and a greater vector length number range for x would be used, if the resolution between parameters is preserved respectively.

Although in this example the min and max number range in the numerical representation would require to be larger. Given that the input number range is being reflected in the weights and biases of the generalisation model and relates to: y to x and y to z then a subtraction or addition operator coded from the inputted numerical representation might offset the position in the weights rather than compress or expand the number scale range used, and that offset would be also be captured in the weights an biases too. This might allow discrimination of single operators and their relationships between inputs, or to put it in the context of colour imagery between the colours in the image.

8. Discrimination of Add Operations

Adding a third parameter v, which will be $v=z+4$ and as such is related to z with an addition offset of 4 but also related to x as $v=x/2+4$, then we can see in the weights in Fig 18 that the weight vector is now 300 long, i.e. 100 for each parameter: weight vector tensor 0-99 addresses are parameter 1 (or x), weight values addresses 100 to 199 are parameter 2 (or z) and addresses 200-299 are parameter 3 (or v).

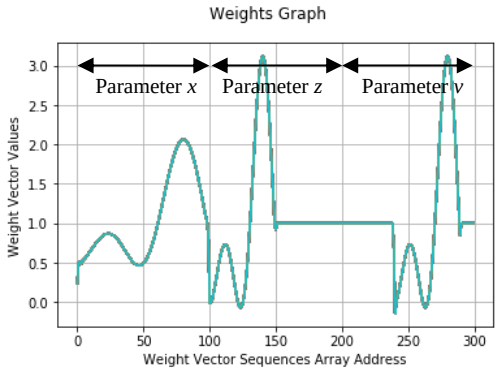

Figure 18. Weight Tensor from Three Parameters

Therefore an offset between parameter 2 (or z) and parameter 3 (or v) can be observed and that offset is proportionally the value 4 in the input number range used in the input tensor numerical representation (0-10). We can also note that Parameters 2 (or z) and 3 (or v) are similar showing that the relationship between z and v, but is offset and as that offset is to the right (higher values) the operator is an addition rather than a subtraction. Also the parameters z and v are mimics of parameter

1 (or x) but are expressed in a smaller number vector range (i.e. are compressed) so both are related to parameter 1 (or x) but as the value range used is half that of x as it is a division relationship. Meaning that $z = x/2$ and either $v = z+4$ or $v=x/2+4$.

9. Experiment Findings

This analysis of Fig 18 has shown that all parameters are related to the function of x, but z is a scaling of x and v is and offset of z. Also it should be noted that the amplitude of the parameters 2 (z) and 3 (v) is also about one and a half that of the first parameter 1 (x) suggesting that a different density of neuron's weights have been updated, and the learning rate influence from those updates is uniform between them but the input number range used is not. This is because the parameter z and v number scales used a narrower number range in the input then that of parameter x and had unused number ranges in the input numerical representation. So a smaller number of weights are updating more, and as the dataset size is equal and also the learning rate influence is common. That might suggest that learning rate conceptually could be set more dynamically rather than set as a fixed layer global hyper parameter and that may be a requirement should there be unexercised number ranges. Interestingly, a 2018 paper by Baydin, et al. [9] looks at combining learning rate with gradient descent optimisation, although in that case the Nesterov momentum is used and in this case momentum and decay are disabled. The 2018 Baydin et al. paper rediscovers and modernises a concept that was originally proposed by Almeida et al. in 1998 [10] and supports that theory of the hyper-parameter learning rate being more integrated to gradient descent. Also, in prediction mode the biases was effected by the initialisation weights used. With experimentation of different weight initialisation schemes there was a numerical residue of the initialisation vector used in the unused input's numerical representation areas. Imagery this may not be thought to occur because the image formats may have no unused pixels. However, it could be a concern in a convolutional network with strides and padding in the padded areas. As such in this analysis the initialisation weight biases are set to a constant value to improve accuracy in the unused range as that initialisation influence is constant. Also noting that the discontinuity was shown in the biases with two parameters as shown in Fig 19, but removed from the biases when three parameters were used as the number of updates increased and the unused vector at the edge was less with the parameter 3 (or v) which had an addition offset in the number range used.

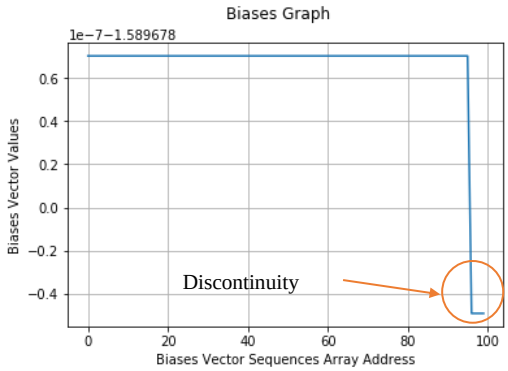

Figure 19. Bias Tensor from Two Parameters

10. Optimising Learning Rate

When optimising Learning Rate (*LR*), the current value is 1.0 and the number of neurons (*NumN*) equals the numerical representation's vector length of 100 (0-99). It may be expected that when the number of neurons is changed then the learning rate may also change as the receptive field of the outputs will be different as a different number of neurons update in the network are forming the generalisation model.

In Table 1, the first three rows show the Learning Rate (*LR*) and Number of Neurons (*NumN*) are being reduced by the Factors (*F*) 1, 2 and 4 and a comparable performance is sustained shown in

red. Also in the last two rows a difference factor between the Learning Rate and Number of Neurons is used and improves the performance and has allowed Learning Rate to be reduced to a value of 0.5 and the Number of Neurons to 5 (shown in the last row in green). As the learning rate is reduced it is worth investigating the shuffle and its' effect as the reorder had an effect and one of those effects is to disrupt the linier numerical order in the dataset. It may not be necessary to reorder randomly but disrupt the order to a more least adjacent order (or least neighbour order).

Table 1. Learning Rate Adaptations

F	*LR*	*NumN*	*Results*
1	*1*	*100*	*Loss = 2.8103146424641545e-05* *MAE = 0.001414917127403678* *Accuracy = 0.92052*
2	*0.5*	*50*	*Loss = 2.810314579707485e-05* *MAE = 0.0014149171285957709* *Accuracy = 0.92052*
4	*0.25*	*25*	*Loss = 2.8103146134955545e-05* *MAE = 0 0.0014149171285957709* *Accuracy = 0.92052*
4/20	*0.25*	*5*	*Loss = 8.954253649751535e-06* *MAE = 0.001068463125228882* *Accuracy = 1.0*
2/20	*0.5*	*5*	*Loss = 7.575523433283706e-06* *MAE = 0.0010504415178298951* *Accuracy = 1.0*

11. Shuffles in Least Adjacent Order

Random reshuffles may be effective, but perhaps other schemes could be experimented with. Because the dataset in the "number representation" is in a numerical order, and as found earlier it maybe that the dataset order could be important a numerical disruption can be applied, as a stride placement of the original dataset at an offset of two and then in filling the missing gaps with the remaining data in reverse order as in Table 2, See Appendix A for a pseudo code version of this shuffle.

Table 2. Example Single Least Adjacent Shuffle Reordering

Data Sets Sequence Order									
0	1	2	3	4	5	6	7	8	9
0	9	1	8	2	7	3	6	4	5

Algorithmically this process is repeatable and will re-arrange back to the original order after a *dataset length-1* number of iterations as Table 3 illustrates with an illustrative example dataset length of 10:

Table 3. Example of 10 Iterations Least Adjacent Shuffle Reordering

Shuffle Iterations	Data Sets Sequence Order									
	0	1	2	3	4	5	6	7	8	9
	0	9	1	8	2	7	3	6	4	5
	0	5	9	4	1	6	8	3	2	7
	0	7	5	2	9	3	4	8	1	6
	0	6	7	1	5	8	2	4	9	3
	0	3	6	9	7	4	1	2	5	8
	0	8	3	5	6	2	9	1	7	4
	0	4	8	7	3	1	5	9	6	2
	0	2	4	6	8	9	7	5	3	1
	0	1	2	3	4	5	6	7	8	9

Applying this shuffle concept to the dataset with the three parameter example, with only 10 reordering iterations in-place of the random shuffle provides these results:

- The loss = 7.621810897380783e-06
- Mean Absolute Error = 0.0010328395795822143
- Accuracy = 1.0

The random shuffle is still a slight improvement on the alternative shuffle but it is very comparable in performance. The image matrix depiction of the input dataset with 10 iterations of the Least Adjacent shuffle is shown in Fig 20.

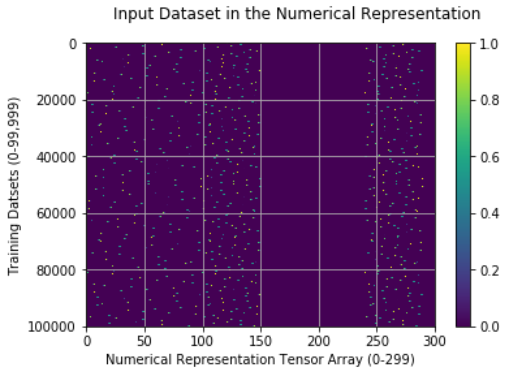

Figure 20. Dataset Matrix Image with Alternative Shuffle with 10 Iterations

The Least Adjacent shuffle may not have added benefit but it's comparisons are very small scales of difference and it shows that an alternative shuffle can be as almost effective as the random shuffle. More work on alternative shuffles is needed, but it does however demonstrate that this Neural Network configuration provides a mechanism for analysis.

Finally in Fig 21 (left), the input in Blue is plotted with the prediction in green dashed and uses the alternative shuffle and the thick Red line expected answer is completely overlaid by the Green dashed line prediction, even where the edge discontinuity was present, showing that this mechanism for analysis can also achieve high degrees of accuracy and determinism. Also if the operators that were discriminated from the inputs x, z and v are reversed (Corrected) then the *sin* function is remaining, as shown in Fig 21 (right), as the sine function was not part of the input parameters and is a residual function in the output and was not within the inputs from the previous layer. This demonstrates that the layer inputs can be discriminated and revealed the function of the current layer.

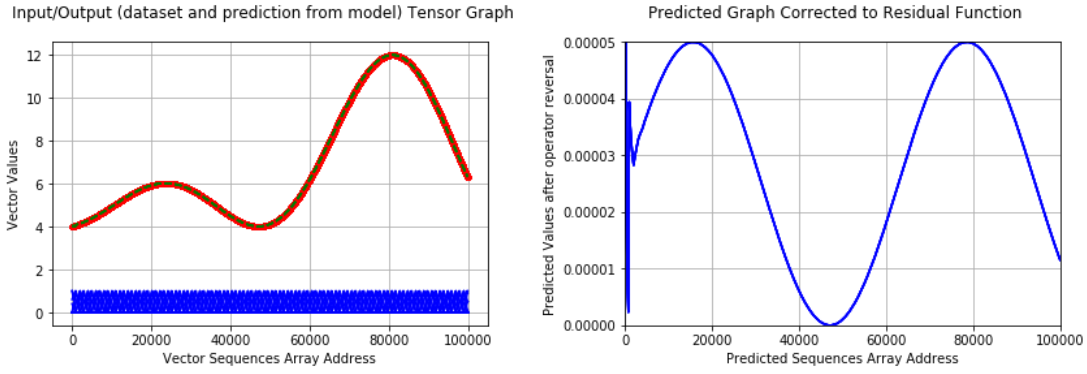

Figure 21. Input, Output and Model Prediction with Alternative Shuffle

The alternative shuffle appears to have had a comparable performance with the random scheme but not quite surpassing it. Although, more work is required in this area, on other alternative schemes and also for setting the number of iterations in this scheme.

12. Conclusions

The paper has provided a configuration architecture with an encoding numerical representation that allows the basis for insights into the captured generalisation model within the weights and biases, and also has demonstrated the discrimination of the input parameter's mathematical relationships derived from a learnt model in an approach that may be complementary to a decomposition approach to rule extraction as an algebraic symbolic formula extraction. The numerical representation also has close applications to imagery formats. Further work is also needed in the initialisation tensors, and by experimentation the paper used a constant value initialisation scheme, and experiments with other non constant initialisation values schemes appeared to leave a residue of that initialisation tensor in the weights in places where the neuron receptive field was not exercised because the input number range was unused. There may be an opportunity to place influences into the initialisation scheme that can be placed into those unused vector ranges for assigned safety critical outcomes covering indeterminism. That same architecture may also be used for the analysis of hyper-parameters in further work. The paper's results should not be miss-interpreted as a case for a new shuffle algorithm or for a change in concept of learning rate or for defining a basis for calculating the number of nodes for optimisations, although further work using this architecture baseline could follow in those areas. The paper did demonstrate that the architecture provides a clear case for analysis mechanism for that work, but also for its' original intention to gaining insights into the captured generalisation model within the weights and biases, and as such could form a foundation for a decomposition classification approach for formula extraction as part of a rule extraction approach. The paper did experiment with hyper-parameters to enhanced the overall performance to demonstrate the effect of these parameters to form a weight model with close understood correlation of both predictions and the generalisation model to increase the accuracy in the understanding of the validity of the capture in the weights.

References

[1] Kurd, Z., Kelly, T. and Austin, J. (2006). Developing artificial neural networks for safety critical systems. Neural Computing and Applications, 16(1), pp.11-19.

[2] Zhang, G. (2000). Neural networks for classification: a survey. IEEE Transactions on Systems, Man, and Cybernetics, 30(4), pp.451 - 462.

[3] Tan, S. and Mayrovouniotis, M. (1995). Reducing data dimensionality through optimizing neural network inputs. AIChE Journal, 41(6), pp.1471-1480.

[4] Hull, J., Ward, D. and Zakrzewski, R. (2002). Verification and validation of neural networks for safety-critical applications. Proceedings of the 2002 American Control Conference, IEEE Cat. No.CH37301.

[5] De Wilde, P. (1997). Neural network models. London: Springer, pp.16, 36.

[6] Hailesilassie, T. (2019). Rule extraction algorithm for deep neural networks: A review. Available: https://arxiv.org/ftp/arxiv/papers/1610/1610.05267.pdf [Accessed 21 Apr. 2019].

[7] GopiKrishna, T. (2014). Evaluation of rule extraction algorithms. Available: http://aircconline.com/ijdkp/V4N3/4314ijdkp02.pdf [Accessed 23 Apr. 2019].

[8] Bologna, G. (2019). A Simple Convolutional Neural Network with Rule Extraction. Applied Sciences, 9(12), p.2411. Available: https://www.mdpi.com/2076-3417/9/12/2411/htm [Accessed 29 Jun. 2019].

[9] Baydin, A., Cornish, R., Rubio, D., Schmidt, M., and Wood, F. (2018). Online Learning Rate Adaptation with Hypergradient Descent. In Sixth International Conference on Learning Representations (ICLR). Available: https://openreview.net/forum?id=BkrsAzWAb [Accessed 24 May 2019].

[10] Almeida, L. B., Langlois, T., Amaral, J. D., and Plakhov, A. Parameter adaptation in stochastic optimization. In Saad, D. (ed.), On-Line Learning in Neural Networks. Cambridge University Press, 1998.

Appendix A

The Numerical Representation Encoding method in pseudo code:

```
Function setvalue (vectorArray, val, minRange, maxRange)
  diff            = maxRange - minRange
  vecLength       = get_length_v(vectorArray)
  res             = diff / (vecLength - 1)
  offset          = val - minRange
  offsetRes       = offset / res
  p1              = int_truncate_s(floor_s(offsetRes))
  p2              = int_truncate_s(p1 + 1)
  v2              = (offsetRes) - p1
  v1              = 1 - v2
  vectorArray[p1] = vectorArray[p1] + v1
  vectorArray[p2] = vectorArray[p2] + v2
  return vectorArray
End
```

The Numerical Representation Decoding method in pseudo code:

```
Function getvalue (vectorArray, minRange, maxRange)
  diff            = maxRange - minRange
  vecLength       = get_length_v(vectorArray)
  res             = diff / (veclength - 1)
  adrs            = ramp(0, vecLength)
  posVal          = add_vs(multiply_vs(adrs, res), minRange)
  weightVec       = multiply_vv(posVal, vectorArray)
  sumVec          = sum_v(vectorArray)
  sumWeight       = sum_v(weightVec)
  val             = sumWeight / sumVec
  return val
End;
```

Least Adjacent Order Alternative Shuffle experiment:

Least adjacent re-ordering pseudo code function for the input tensors:

```
Function shuffle_xdata(xDataset, vectorLength, noOfParams)
     arrayLength = vectorLength*noOfParams-1
     xDatasetshuff = zeros((noOfDatasets,vectorLength*noOfParams))
     for n = 0 to int_truncate(noOfDatasets/2)-1
        xDatasetshuff[n*2  ][0:arrayLength] = xDataset[n                ][0:arrayLength]
        xDatasetshuff[n*2+1][0:arrayLength] = xDataset[noOfDatasets -1-n][0:arrayLength]
     End loop
     return xDatasetshuff
End
```

Least adjacent re-ordering in pseudo code function for the training dataset categories tensor:

```
def shuffle_ydata(yDataset):
     yDatasetshuff = zeros(noOfDatasets)
     for n = 0 to int_truncate(noOfDatasets/2)-1
        yDatsetshuff[n*2  ] = yDataset[n               ]
        yDatsetshuff[n*2+1] = yDataset[noOfDatasets-1-n]
     End loop
     return yDatasetshuff
End
```