Opening the Black Box: Analysing MLP Functionality using Walsh Functions

Kevin Swingler
Computing Science and Maths, University of Stirling, Stirling, FK9 4LA. Scotland

Keywords: Multilayer Perceptrons, Walsh Functions, Network Function Analysis

Abstract: The Multilayer Perceptron (MLP) is a neural network architecture that is widely used for regression, classification and time series forecasting. On often cited disadvantage of the MLP, however, is the difficulty associated with human understanding of a particular MLP's function. This so called black box limitation is due to the fact that the weights of the network reveal little about structure of the function they implement. This paper proposes a method for understanding the structure of the function learned by MLPs that model functions of the class $f : \{-1,1\}^n \to \mathbb{R}^m$. This includes regression and classification models. A Walsh decomposition of the function implemented by a trained MLP is performed and the coefficients analysed. The advantage of a Walsh decomposition is that it explicitly separates the contribution to the function made by each subset of input neurons. It also allows networks to be compared in terms of their structure and complexity. The method is demonstrated on some small toy functions and on the larger problem of the MNIST handwritten digit classification data set.

1 INTRODUCTION

The multilayer perceptron (MLP) (Rumelhart et al., 1986) is a widely used neural network architecture. It has been applied to regression, classification and novelty detection problems and has been extended in various ways to process time varying data, e.g. (Elman, 1990). In the field of data mining MLPs are a common choice amongst other candidates such as classification trees, support vector machines and multiple regression. Due to the wide variety of tasks for which they are suited, and their ability as universal approximators, MLPs have become very popular. However, there is one aspect of the MLP that restricts and complicates its application, and that is the role of the hidden neurons. A common criticism of the MLP is that its knowledge is not represented in a human readable form. The comparison that is often made is with classification or regression trees, which represent partitions in the input space explicitly in their structure. This makes human understanding of the underlying function and the reasons behind any given output quite easy. Given a picture of a classification tree, a human may apply it to an input pattern without even needing a computer to run the algorithm. This is far from simple with an MLP.

The hidden units in an MLP act as feature detectors, combining inputs from below into higher order features that are, in turn, combined by higher layers still. The common learning algorithms such as back propagation of error (Rumelhart et al., 1986) have no explicit means of ensuring that the features are optimally arranged. Different neurons can share the same feature, or have overlapping representations. In networks where each layer is fully connected to the one above, every hidden neuron in a layer shares the same receptive field, so their roles often overlap. This makes analysis even more difficult as hidden neurons do not have independent roles. The inclusion of additional layers of hidden neurons compounds the problem further.

Some work has been carried out on the analysis of hidden neurons in MLPs. For example, (Kamimura, 1993) used an entropy based analysis to identify important hidden units (known as principal hidden units) in a network for the purpose of pruning an oversize hidden layer. (Sanger, 1989) proposed a method of contribution analysis based on the products of hidden unit activations and weights and (Gorman and Sejnowski, 1988) presented a specific analysis of the hidden units of a network trained to classify sonar targets.

The question of how to extract rules from multilayer perceptrons has received more attention and is still a very active field of research. (Kulluk et al., 2013) propose a fuzzy rule extraction method for neural networks, which they call Fuzzy DIFACONN. (Hruschka and Ebecken, 2006) propose a clustering based approach to MLP rule extraction that uses genetic algorithm based clustering to identify clusters of hidden unit activations which are then used to generate classification rules. (Saad and Wunsch II, 2007) use an inversion method to generate rules in the form of hyperplanes. Inverting an MLP (i.e. finding the inputs that lead to a desired output) is done by gradient descent and using an evolutionary algorithm. Both (Augasta and Kathirvalavakumar, 2012) and (Jivani et al., 2014) present recent comparative studies of neural network rule extraction, distinguishing between methods that are decompositional, pedagogical and eclectic. A decompositional approach extracts rules from the weights and activations of the neural network itself. The pedagogical approach, which is taken in this paper, treats the neural network as a black box and generates rules based on the outputs generated by the network in response to a set of input patterns. Eclectic rule extraction combines both of the aforementioned approaches.

More work has concentrated on choosing the right number of hidden units for a specific data set. (Baum and Haussler, 1989) bound the number of weights by the target error size, (Uphadyaya and Eryurek, 1992) bounded the number of hidden units by the number of patterns to be learned, (Widrow and Lehr, 1990) chose a bound based on the number of output units in the network, and (Weigend et al., 1992) pointed out that the amount of noise in the training data has an impact on the number of units used. Some have taken a dynamic approach to network structure discovery, for example (Bartlett, 1994) used an information theoretic approach to add or remove hidden neurons during training. The problem with this approach to training an MLP is that the existing weights are found in an attempt to minimise error for that number of hidden neurons. Adding a new one may mean the existing weights are starting in a configuration that is unsuitable for a network with more neurons. Other search methods have also been applied to finding the right structure in an MLP. (Castillo et al., 2000) and (Yao, 1999) used genetic algorithms to search the space of network structures, for example.

When using MLPs (and other machine learning techniques), it is common practice to produce several models to be used in an ensemble (Krogh and Vedelsby, 1994). Due to the random start point of the weight values, and the differences in architecture across the networks in an ensemble, it is not easy to know whether or not different networks are functionally different. It is possible to train a number of different MLPs that all implement the same function (perhaps with differing quality of fit across the weights) with very different configurations of weight values. For example, one could re-order the hidden units of any trained network (along with their weights) and produce many different looking networks, all with identical functionality. One way to compare MLPs is to compare their outputs, but a structural comparison might also be desirable, and that is what we present here.

Note the distinction between the structure of an MLP, which is defined by the neurons and connecting weights, and the structure of the function it implements, which can be viewed in a number of other ways. This paper views the underlying function implemented by an MLP in terms of the contribution of subsets of input variables. The number of variables in a subset is called its order, and there are $\binom{n}{k}$ subsets of order k in a network of n inputs. The first order subsets are the single input variables alone. The second order subsets are each of the possible pairs of variables, and so on. There is a single order n set, which is the entire set of inputs. Any function can be represented as a weighted sum of the values in each of these subsets. The weights (known as coefficients in the chosen analysis) are independent (unlike the weights in an MLP, whose values are determined to an extent by other weights in the network) and specific to their variable subset. The first order coefficients describe the effect of each variable in isolation, the second order coefficients describe the contribution of variable pairs, and so on. The method for decomposing a neural network function into separate components described in this paper is the Walsh transform. When the phrase "network functionality" is used in this paper, it means the form the function takes in terms of how the interactions between different subsets of input variables affect each output variable.

Section 2 describes the Walsh transform in some detail. This is followed by a description of the method for producing Walsh coefficients from a neural network in section 3. Section 4 introduces some functions that will be used in experiments described in following sections. Section 5 demonstrates how the method can be used to track the complexity of MLPs during training and section 6 demonstrates how a partial transform on a small sample from a larger network can provide useful insights. The Walsh method is compared to other methods of understanding network structure in section 7. Finally, sections 8 and 9 offer some conclusions and ideas for further work.

2 WALSH FUNCTIONS

Walsh functions (Walsh, 1923), (Beauchamp, 1984) form a basis for real valued functions of binary

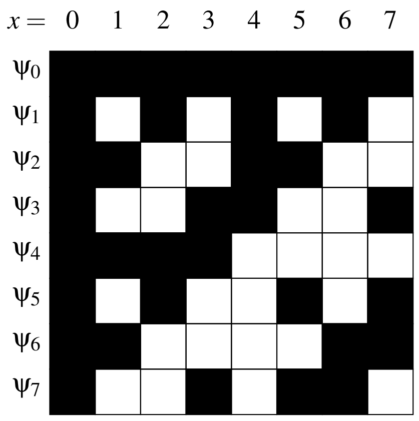

Figure 1: A pictorial representation of an order 3 Walsh matrix with black squares representing 1 and white squares -1. A Walsh sum is calculated by summing the product of the Walsh coefficient associated with each row by the values in the column indexed by the function input.

vectors. Any function $f : \{-1,1\}^n \to \mathbb{R}$ can be represented as a weighted linear sum of Walsh functions. The Walsh functions take the form of a sequence of bit strings over $\{-1,1\}^{2^n}$ where n is the number of variables in the function input. n is known as the Walsh function order. There are 2^n Walsh functions of order n, each 2^n bits long. Figure 1 shows a representation of the order 3 Walsh functions. Each Walsh function has an index from 1 to 2^n, with the j^{th} function being ψ_j and bit number x of the j^{th} Walsh function is $\psi_j(x)$. As figure 1 shows, the Walsh functions can be viewed as a matrix of values from $\{-1,1\}$ with rows representing each Walsh function and columns representing each bit.

A Walsh representation of a function $f(\mathbf{x})$ is defined by a vector of parameters, the Walsh coefficients, $\omega = \omega_0 \ldots \omega_{2^n-1}$. Each ω_j is associated with the Walsh function ψ_j, that is a row in the Walsh matrix. Each possible input, $\mathbf{x}$ is given an index, x, which is calculated by replacing any -1 in $\mathbf{x}$ with 0 and converting the result to base 10. For example if $\mathbf{x} = (1,-1,1)$, then $x = 5$. Each column of the Walsh matrix corresponds to a value of x.

The Walsh representation of $f(\mathbf{x})$ is constructed as a sum over all ω_j. In the sum, each ω_j is either added to or subtracted from the total, depending on the value of the bit corresponding to x (i.e. column x in the Walsh matrix), which gives the function for the Walsh sum:

$$f(\mathbf{x}) = \sum_{j=0}^{2^n} \omega_j \psi_j(x) \tag{1}$$

2.0.1 Constructing the Walsh Functions

The value of a single cell in the Walsh matrix, $\psi_j(x)$ is calculated from the binary representation of the coordinates (j,x), of $\mathbf{j}$ and $\mathbf{x}$, and returns +1 or -1 depending on the parity of the number of 1 bits in shared positions. Using logical notation, a Walsh function is derived from the result of an XOR (parity count) of an AND (agreement of bits with a value of 1):

$$\psi_j(x) = \oplus_{i=1}^{n}(x_i \wedge j_i) \tag{2}$$

where $\oplus$ is a parity operator, which returns 1 if the argument list contains an even number of 1s and -1 otherwise.

2.0.2 Calculating the Coefficients - the Walsh Transform

The Walsh transform of an n-bit function, $f(\mathbf{x})$, produces 2^n Walsh coefficients, ω_x, indexed by the 2^n combinations across $f(\mathbf{x})$. Each Walsh coefficient, ω_x is calculated by

$$\omega_x = \frac{1}{2^n} \sum_{j=0}^{2^n-1} f(j)\psi_j(x) \tag{3}$$

Each of the resulting Walsh coefficients has an index, which defines the set of input variables over which it operates. Converting the index to a binary representation over n bits produces a representation of the variables associated with the coefficient where a 1 in position i indicates that x_i contributes to the effect of that coefficient. For example, over 4 bits, the coefficient ω_9 produces a binary word 1001, which tells us that x_1 and x_4 contribute to the effect of ω_9. The order of a coefficient is defined as the number of bits it contains that are set to 1. For example, ω_2 and ω_8 are first order as they have one bit set to 1, and ω_9 is second order. The magnitude of a coefficient indicates its importance in contributing to the output of the function on average across all possible input patterns.

A function of n inputs produces 2^n Walsh coefficients, so it is not always possible to consider the value of each coefficient individually. In this work we look at individual coefficients and also define some simple aggregate measures for summarising the results of a Walsh transform. They are the number of non-zero coefficients, which is taken as a crude measure of overall complexity, and the average magnitude of coefficients at each order, which produces a set of values that measure the contribution to the function's output made on average by interactions of each possible order.

3 METHOD

In this context, the Walsh transform is not used to understand the training data, but to understand a neural network that was trained on that data. The analysis is in terms of the inputs to and the outputs from the network, not its weights or activations, making this a pedagogical approach. The black box of the neural function is assessed in terms of its Walsh decomposition. Walsh functions map a vector of binary valued inputs onto a real valued output, so any function with this structure is amenable to the analysis. As shown below, multiple output neurons and classification networks may also be analysed with this approach, so the outputs can be nominal, discrete or continuous.

As neural networks can generalise and produce an output for any given input pattern, we can generate an exhaustive or randomly sampled data set from which to perform the Walsh transform. A full Walsh decomposition, as defined in equation 3 requires an exhaustive sample of the input space. In all but the smallest of networks, this is unfeasible in an acceptable time period, so the coefficients must be calculated from a sample. In either case, the sample used to calculate the coefficients is generated from the whole input space, not just the training data. The significant coefficients (those that are significantly far from zero) can be very informative about the underlying structure of the function (in this case, the MLP). The procedure is similar to that of pedagogical rule discovery in that it treats the MLP as a black box and performs an analysis on the output values that the network produces in response to input patterns. The method proceeds as follows:

1. Build a single MLP using your chosen method of design and weight learning;
2. Generate input patterns (either exhaustively or at random) and allow the MLP to generate its associated output, thus producing $(\mathbf{x}, f(\mathbf{x}))$ pairs;
3. Use the resulting $(\mathbf{x}, f(\mathbf{x}))$ pairs to perform a Walsh transform using equation 3;
4. Analyse the significant coefficient values, ω_x.

The method can also be used for MLPs designed for classification rather than regression. In such cases, there is normally a single output neuron for each class, with a target output value of one when the input belongs to the neuron's designated class and zero otherwise. Properly trained, each neuron represents the probability of a new pattern belonging to its designated class. Such a network is effectively a number of related functions (one for each class) with a continuous output between zero and one. Each output neuron can be analysed in turn using the same procedure.

Step 4, the analysis of the ω_x values can take many forms. This paper discriminates between analysis during training (section 5) where the goal is to gain an insight into the level of complexity a network achieves as learning progresses, and post training analysis, designed to provide insights into the function of the trained network. The example of such an analysis in section 6.1 shows how the generalisation ability of a network may be investigated from the results of the Walsh analysis. The goal of the analysis is not to generate rules, so this is not another rule extraction method, rather it is designed to give human insights into the hidden life of the MLP.

4 EXPERIMENTS

A set of functions of increasing complexity[1] were chosen to generate data to test this analysis. They are:

OneMax, which simply counts the number of values set to one across the inputs. This is a first order function as each variable contributes to the output independently of any others. The OneMax function is calculated as

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i \tag{4}$$

Vertical symmetry, which arranges the bits in the input pattern in a square and measures symmetry across the vertical centre line. This is a second order function and is calculated as

$$f(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} \delta_{ij} s_{ij} \tag{5}$$

where δ_{ij} is the Kronecker delta between x_i and x_j, and s_{ij} is 1 when i and j are in symmetrical positions and 0 otherwise.

K-bit trap functions are defined by the number of inputs with a value of one. The output is highest when all the inputs are set to one, but when at least one input has a value of zero, the output is equal to one less than the number of inputs with a value of zero. For example, in three bits, $f(111) = 3$ is the function's maximum, $f(000) = 2$ produces the next highest output, and $f(011) = 0$ is a global minimum. A k-bit trap function over n inputs, where k is a factor of n is defined by concatenating subsets of k inputs n/k times. Let $\mathbf{b} \in \mathbf{x}$ be one such subset and $c_0(\mathbf{b})$ be the number of bits in $\mathbf{b}$ set to zero.

[1]Complexity has a specific meaning in this context. It describes the number and order of the interactions between inputs that produce a function's output.

$$f(\mathbf{x}) = \sum_{\mathbf{b} \in \mathbf{x}} f(\mathbf{b}) \tag{6}$$

where

$$f(\mathbf{b}) = \begin{cases} c_0(\mathbf{b}) - 1, & \text{if } c_0(\mathbf{b}) > 0 \\ k, & \text{if } c_0(\mathbf{b}) = 0 \end{cases} \tag{7}$$

The first case in equation 7, which applies to all but 1 in 2^k patterns, could be modelled with a first order network (a linear perceptron, for example), which is a local minimum in the error space. The 'trap' part of the function is caused by the second case in equation 7, which requires the output to be high when all of the inputs have a value of one. This requires a higher order function, including components at orders from 1 to k, but only a small proportion of the data (1 in 2^k of them) contains any clue to this.

5 ANALYSIS DURING TRAINING

Experiments were conducted to investigate the structure of the function represented by an MLP as it learns. The MLP used in these experiments had a single hidden layer and one linear output neuron. The functions described above were used to generate training data, which was used to train a standard MLP using the error back propagation algorithm. At the end of each epoch (a single full pass through the training data), a Walsh transform was performed on the predictions made by the network in its current state.

Summary statistics designed to reflect the complexity of the function the network has implemented and the level of contribution from each order of interaction were calculated from the Walsh coefficients. The complexity of the function was calculated as the number of significant non-zero Walsh coefficients. The size of the contribution from an order of interaction, o was calculated as the average of the absolute value of the coefficients of order o. Experiment 1 trained networks on the simple OneMax function (equation 4). Figure 2 shows the training error and network complexity of an MLP trained on the OneMax function. During learning, the network initially becomes over complex and then, as the error drops, the network complexity also drops to the correct level.

In experiment 2, an MLP was trained on the symmetry function of equation 5, which contains only second order features. Figure 3 shows the results of analysing the Walsh coefficients of the network function during learning. Three lines are shown. The solid line shows the network prediction error over time and the broken lines show the contribution of the first and second order coefficients in the Walsh analysis of the network function. Note the point in the error plot where the error falls quickly corresponds to the point in the Walsh analysis where the second order coefficients grow past those of first order. Compare this chart to that in figure 4, where the same problem is given to another MLP with the same structure, but which becomes trapped at a local error minimum, which is a first order dominated approximation to the function. The plot suggests that the higher order components cannot increase their contribution and that this network is unlikely to improve.

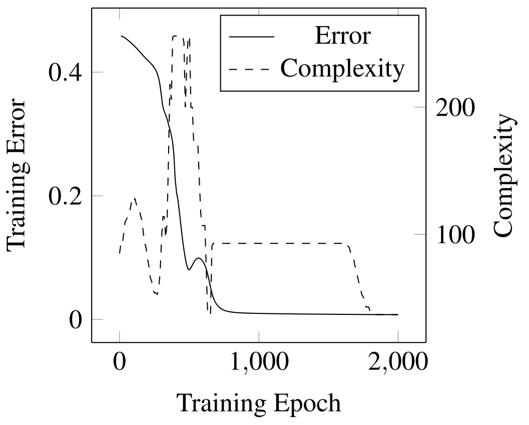

Figure 2: Comparing training error with network complexity during learning of the OneMax function with an MLP with one hidden unit. Note that complexity falls almost 1000 epochs after the training error has settled at its minimum.

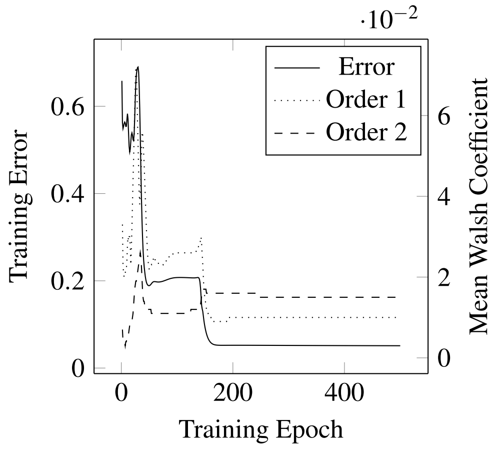

Figure 3: Network error and contribution of first and second order Walsh coefficients during training of an MLP on a second order function. Note the fall in the error rate when the second order coefficients overtake the first.

Figure 5 shows the error of an MLP decrease as it learns the 4 bit trap problem described by equation 6. The contribution of the first, second, third and fourth

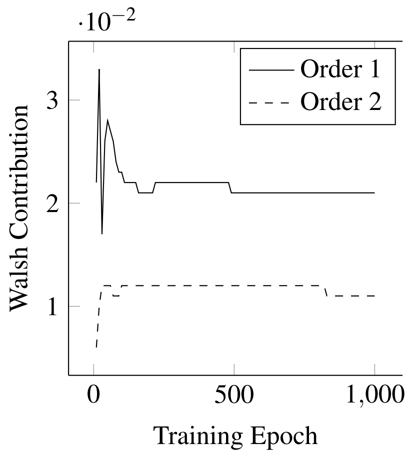

Figure 4: Contribution of first and second order Walsh coefficients during training of an MLP on a second order function, stuck in a local minimum.

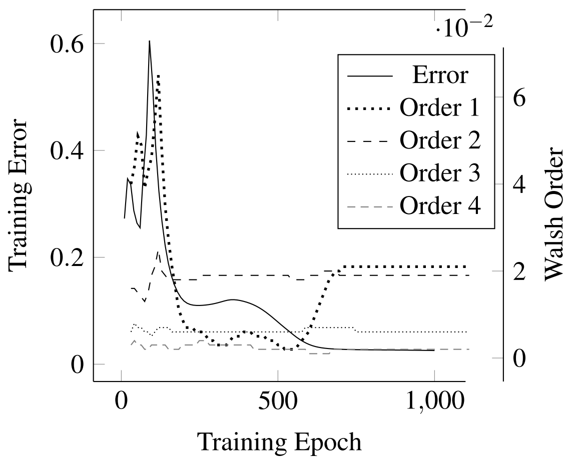

Figure 5: The contribution of first, second, third and fourth order Walsh coefficients during training of an MLP on a the 4-bit trap function plotted with the average error per pass through the data set (solid line).

order Walsh coefficients are each summed and plotted separately. The final, correct configuration can be seen in the right hand part of the plot, with the first order coefficients having the strongest contribution, but with the second, third and fourth also required to escape the 'trap' of the order below. The plot shows the first order coefficients growing first (as they did in figure 3), causing the average error to rise due to the higher order trap part of the function. The first order components are suppressed by the high error they cause, but the error doesn't settle to its lowest point until the first order coefficients recover the correct level of contribution.

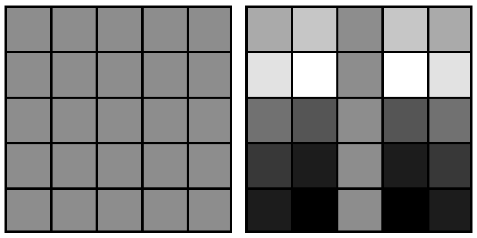
(a) First order (b) Second order

Figure 6: First and second order coefficients of a symmetry counting function. In (a), the coefficients are all zero. In (b), the shade of gray indicates a non zero second order coefficient across the two pixels with shared gray level.

5.1 A Second Order Function

In the following experiment, a second order function is investigated. The function is a measure of pattern symmetry, as defined in equation 5. Figure 6a shows the first order coefficients of a network trained to measure the symmetry of an image. Unsurprisingly, it shows no first order coefficients of importance. Mid grey indicates values close to zero, which suggests either that the variable that corresponds to the coefficient is unimportant or that variables are involved at higher orders. The higher order coefficient values tell us which of these possibilities is true.

Figure 6b shows the second order coefficients of a Walsh transform of the symmetry predicting MLP. The plot is produced by finding pairs of inputs that share a non-zero second order coefficient and setting them both to the same, unique shade of gray. Note that the centre column inputs share no second order relationships and are shaded mid-gray. The others are shaded so that their gray level matches that of the inputs with which they share a non-zero second order coefficient. The depth of shade does not indicate the size of the parameter, just that a connection exists. The shading is to discriminate between input pairs.

It is clear from figure 6b that each input is important to the calculation of the function output, so the interpretation of the zero valued first order coefficients is that the inputs' contributions are important, but only at orders above one.

The next experiment described in this paper makes use of partial samples from both the coefficients and the input space to gain an insight into the structure of an MLP trained on a pattern recognition task. It also illustrates the way error rates can be compared to determine how much of the network's functionality has been explained by the computed coefficients.

6 PARTIAL WALSH ANALYSIS

For even moderately large numbers of inputs, calculating every Walsh coefficient can take an impractically long amount of time. In such cases, a partial Walsh analysis can still be useful. A partial analysis calculates the values of only a small subset of the Walsh coefficients. An obvious choice for the subset of coefficients to calculate are those of the lower orders. ω_0 is the average output of the function (in this case, the MLP) across the sampled data. The first order coefficients, $\omega_1, \omega_2, \omega_4, \ldots$ represent the average contribution of each input in isolation. In general, order k coefficients represent the additional contribution of each subset of inputs of size k to the function output. The number of coefficients of order k from a set of inputs of size n is $\binom{n}{k}$, a figure which rises exponentially with k up to $k = n/2$ and then falls exponentially after that, to the point where there is just one order n coefficient. In general, one might expect a function to have significant interactions at the lower orders rather than the higher ones, so the number of coefficients of interest can be said to rise exponentially with their order.

It is also possible to estimate the Walsh coefficients from a sample of random input patterns and their associated predicted outputs from the network, rather than analysing every input pattern exhaustively. As with the calculation of any statistic from a sample, the values gained are estimates, but they can still provide useful insights into the functioning of a neural network. The number of samples required to estimate coefficients accurately grows exponentially with their order, so the low order coefficients can be estimated with smaller samples than the higher order coefficients require.

The values that are found as a result of sampling a small proportion of the Walsh coefficients can be used to reconstruct an estimate of the function implemented by the MLP that produced them. This reconstructed function can be used to generate predictions on the test data. The accuracy of this model will almost always be worse than that of the MLP but by comparing the respective error rates, the proportion of MLP'a ability that is captured by the Walsh coefficients can be measured.

6.1 Measuring Generalisation

Generalisation is the ability of an MLP to produce correct outputs for patterns that were not in its training data. As the weights of the network are difficult to analyse, the performance of the learned function in areas of input space that are outside those covered by the training data can be difficult to assess. Test and validation sets perform this task to a degree, but this paper proposes a new method based on a Walsh analysis where the network is analysed with random input patterns. The use of random inputs (i.e. patterns where each input takes an independent, uniformly distributed random value) allows a trained network to be tested on potentially massive test sets. Of course, these random patterns do not have associated target outputs, but as the Walsh analysis makes use of only the predicted output from the network, the test patterns do not need a target (or desired) output. This allows the analysis to explore a far greater variety of the input/output space of a trained neural network.

The Walsh coefficients of an MLP function are generated by randomly sampling from the whole input space, not just the part of it covered by the training or test data. The coefficients give a picture of the general shape of the function, not just its behaviour on the training ot test data. The experiments described in this section demonstrate the use of a Walsh decomposition of an MLP trained on the MNIST (Lecun and Cortes,) handwritten digit data set. The goal is not to produce a better classification rate than those already reported in the literature. The goal is to train some different networks and use a Walsh analysis to gain an insight into their structure.

6.1.1 Learning the MNIST Data

The MNIST images are made up of 28×28 pixels, making 784 inputs, each with a value from 0 to 255, indicating a grey level in an anti-aliased image. In this work, input values were passed through a threshold to create binary patterns rather than the grey level images of the raw MNIST data. A neural network with 784 inputs, 20 hidden units and 10 outputs (1 for each of the digits from 0 to 9) was trained on the standard MNIST training set, where the images are centred on their centre of mass. The resulting network implements 10 different functions, each mapping the input pattern to a continuous output variable that reflects how well the input pattern matches digits that correspond to its class (i.e. the identity of the digit). These functions are not independent as they share the weights between the input and hidden layer. Across a well trained network, the outputs should sum to one. The network achieved a correct classification rate of 89%, which is poor compared to any serious attempt, but useful for illustration purposes here.

An individual Walsh decomposition for each output neuron based on 50,000 random input samples and their associated network output was performed after training had completed. This produced ten Walsh decompositions. Initially, only the first order coefficients were calculated. The first order coefficients were plotted on a grid where the pixel locations from the inputs correspond to the first order coefficients of the Walsh decomposition, as shown in figure 7, in which it is clear that the network does not even have a very general first order model of the patterns that make each digit. Take the coefficients for the digit "1", for example. Very few of the pixels are used—four or five central pixels have large positive coefficients (making a positive contribution the output neuron value for class "1") and there are a small number of negative (shown in white) pixels to the top left and bottom right which cause pixels in their respective locations to diminish the output for the class "1" output neuron. This shows the network to have a reliance (one might argue and over-reliance) on specific inputs for making a classification. This would manifest itself as a poor ability to cope with noise in any test data where the specific pixels were altered.

When used to reconstruct a first order approximation to the network's function, the first order coefficients alone for this model achieved a root mean squared error (RMSE) of 0.17, indicating that the first order coefficients have captured a large proportion of the network's functionality. The RMSE of a decomposition with respect to the MLP it was derived from is calculated by using equation 1 to produce an output for a number of random samples from the input space, which is compared with the output from the MLP given the same input. A very low RMSE for a partial decomposition indicates that the remaining coefficients (those excluded from the partial decomposition) make very little additional contribution to the function output.

Note that all of the images in this paper are produced by normalising the coefficients being plotted to a range that causes the colours to vary from white to black. This leads to some distortion when the range is very small, as there may be a small distance between the highest and lowest coefficient. If all coefficients are close to zero, the highest will still appear black and the lowest white, though in reality they are all similarly small. This means that you cannot compare one plot with another in terms of absolute values.

6.1.2 Adding Training Noise

A common method of improving generalisation is to add noise to the input values in the training data. By randomly flipping 10% of the input bits each time a pattern was learned, the correct classification rate on the test data increased to 93%. The impact on the first order coefficients can be seen in figure 8 where it is clear that some (but not all) of the digits are now quite clearly identified across more of the input variables. The coefficients reveal the degree to which some classes have a clearly defined shape in the network and others do not. Returning to the example of the digit "1", figure 8 shows how a larger number of central pixels have a positive effect on the output neuron for class "1". The white inhibitory pixels are also more clear and widespread in these figures.

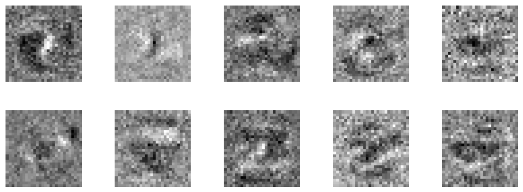
Figure 7: First order Walsh coefficients from a network trained on the MNIST data with no added noise. Grey squares indicate no contribution to classification from a first order component. Greater depth of black or white indicates stronger contribution (positive or negative).

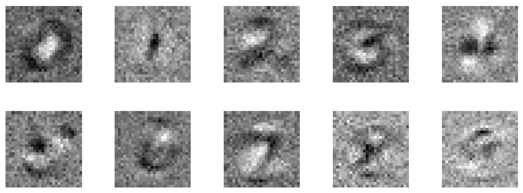
Figure 8: First order Walsh coefficients from a network trained on the MNIST data with 10 percent added noise. The noise ensures that no individual input can be relied upon to produce a correct classification, and so produces a model that covers more of the input space, and so is better at generalisation.

When used to reconstruct a first order approximation to the network's function, the first order coefficients alone for this model achieved a root mean squared error of 0.32, showing the first order coefficients to be responsible for less of the MLP's functionality, even though more of them are used. The remaining network functionality is of a higher order, leading to the conclusion that this network, which has better generalisation ability than the first, is more complex in the sense that it relies on more higher order interactions between the input variables to make its classifications. Discovering the higher order coefficients of interest is not trivial as significant coefficients are sparse among all possible coefficients.

6.1.3 Adding Training Jitter

Another method for attempting to improve the generalisation ability of a handwritten digit classifier is to jitter the training data, which means to shift each training image a small random number of pixels in a random direction before each is presented to the network for learning. Each training pattern is learned several times, each time with a new random shift to its location. This has the effect of blurring the first order coefficients across the input space, making them less useful for classification. Imagine an extreme case where the digit for "1" is moved to any location in the input field. No single pixel would be set to 1 (i.e black) more often than any other when the input pattern represents a 1, so there would be no useful first order information in the data. Second order contributions between pixel pairs would be required for a good classification model. Pairs of pixels that are both set to 1 or both set to 0 when the input pattern represents "1" would have a positive coefficient and those that differed would have a negative coefficient. The strongest effect we would expect in the example of the digit "1" would be positive weights between pixels that were above and below each other in the image field.

The hypothesis is that an MLP trained on jittered inputs would produce weaker first order and stronger second order coefficients. To illustrate this point, take a simpler example than the MNIST data using hand designed digits over 25 pixels arranged in a 5×5 grid. Concentrating on the example for digit "1" and allowing that digit to be represented by any pattern of three or more black pixels above each other on an otherwise white background. Figure 9 shows some example "1" digit images. The other digits (0 and 2 to 9) were also hand designed and a data set was created containing equal numbers of examples of each. The other digits were fixed in their location, but the "1" digits were placed at random in the input field and given random lengths of 3,4, or 5 pixels. After training a neural network to distinguish between the examples of "1" digits and designs for the other digits from 0 to 9, a full first and second order Walsh decomposition was performed by sampling random input patterns and the associated output from the neuron corresponding the class "1".

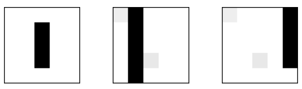
Figure 9: Three examples of small training patterns for the digit "1", varied by location and length, but maintaining the vertical quality.

Figure 10 shows the first order coefficients for the output for "1". Note that there is very little variation in the values as the value of any individual pixel makes no consistent contribution to the output. Second order coefficients are not as straight forward to plot and view as those of first order, as there is one coefficient for every pair of input variables. To visually represent some of the second order coefficients, those with the highest absolute values were chosen and plotted as pairs joined with a line, as shown in figure 10, in which black dots indicate a positive coefficient between the two inputs and white dots indicate a negative coefficient. Each figure shows a small number of coefficients (three positive and four negative). It is clear that pairs of pixels that are in the same vertical line share a positive coefficient and pairs that are in different columns share a negative coefficient. Not every second order coefficient has a significant value. As with the first order coefficients, a small number of them are sufficient to allow correct classifications to be made, so there is no pressure during training for further weight changes to produce a function where every coefficient has the expected value.

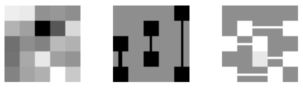
Figure 10: First order coefficients (left) of a network trained on the randomly placed "1" digits of figure 9, and some examples of large positive (middle) and negative (right) second order coefficients from the same network.

On a toy example, this is easy to see. On the larger MNIST data, the process is not as straight forward. This is partly because there are many more second order coefficients to sample and partly because not all of them need to take a value. As pointed out above, and as seen in the first order examples before noise is added, a sparse subset of coefficients are actually needed to reproduce the functionality of the network and once the error is sufficiently low, there is no pressure to change the weights further. However, the process was repeated for the MNIST data. An MLP was trained on the MNIST data and during training, each input pattern was moved by up to 4 pixels in one of the eight possible directions.

Figure 11 shows the first order coefficients, with the result of the shifted input patterns clearer to see in some classes than others. The root mean squared error between the first order Walsh decomposition of this MLP and the output of the MLP was 0.10, indicating that, contrary to expectations, the network in which the patterns were shifted around is better able to rely on the pixel values in isolation, rather than needing higher order coefficients.

6.1.4 Choosing Higher Order Coefficients

The number of coefficients that might be calculated grows exponentially with the number of variables and the order of the coefficients so it is crucial to choose the coefficients of to sample carefully. This section discusses some possible heuristics that might be employed when choosing which higher order coefficients

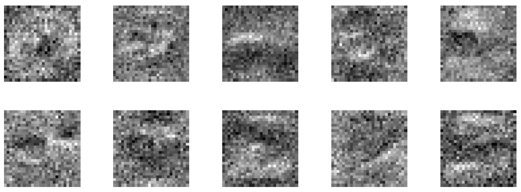

Figure 11: First order coefficients calculated from the output neurons corresponding to each class of hand written digit from training data subjected to random jitter.

to sample. The coefficients can be used to approximate the function that the neural network has implemented, using equation 1 and assuming a zero value for all of the un-calculated coefficients. The mean difference between the network output and that predicted by the Walsh decomposition indicates the mean size of the missing coefficients. When that is sufficiently small, there can be few remaining coefficients to discover.

The best heuristic in most circumstances is to start with the lowest order coefficients first. In the case of the MNIST data, the first order coefficients accounted for most of the network's ability, and so gave a reasonable picture of its functional shape. The identity of the significant first order coefficients may be used to drive the choice of second order coefficients. The choice is between an assumption that variables that are useful at the first order level will also be useful in combination and the opposite assumption that variables that were found to have no first order contribution might play their role in higher orders. With human knowledge of the training process, one might be able to make informed choices as to which pixels might interact.

Another option with image data such as the MNIST set is to calculate coefficients that join neighbouring pixels. For example, each of the 784 pixels might share a second order coefficient with any of the remaining 783, but each pixel has only eight immediate neighbours. By hypothesising that written digits are made by a continuous stroke and so neighbouring pixels are more likely to interact to influence classification output, a small field around each pixel can be chosen, greatly reducing the number of calculations to be made. Second order Walsh coefficients in a small neighbourhood act as edge detectors, giving negative coefficients at points where neighbouring pixels that disagree contribute positively to the function output. This approach shares a great deal with the use of Markov Random Fields for image processing such as segmentation and edge detection, (see (Li, 1995) for example). The key difference in this context is that we are not using the local field to process an image, but to analyse an MLP that was trained to classify a set of images. If the MLP has not captured certain features, then the analysis cannot reveal them, so a failure to find features that might have been expected is not a reflection of the method, but on the particular MLP's underlying functional shape.

7 COMPARISON WITH OTHER METHODS

Recent published work in this field, such as the papers mentioned in the introduction, has concentrated on rule discovery, though what constitutes a rule is quite flexible. (Jian-guo et al., 2008), for example build a binary truth table to represent the function of the MLP. The Walsh method is a pedagogical approach, according to the definitions in (Jivani et al., 2014) as it treats the MLP as a black box. One of the advantages of the pedagogical approach is that the rules that are produced are easy to interpret.

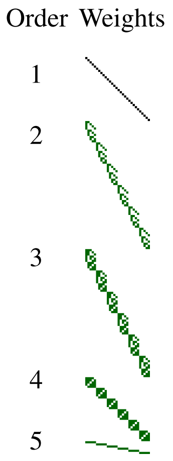

Figure 12: 5 Bit Trap Walsh Coefficients.

The Walsh decomposition approach certainly aids interpretability, but it cannot be considered a rule extraction algorithm as it does not generate rules. Instead, it provides insight into the complexity of an MLP, highlighting both the level of complexity, and the variables involved. For example, in the *k*-bit trap function, it is clear from an examination of the coefficients that inputs are organised into subsets which interact within the traps, but that they are independent across traps.

One advantage of the Walsh method is that the coefficients may be easily visualised. Figure 12 shows the coefficients generated from an MLP that has learned a 5-bit trap function over 30 inputs. The figure is generated by discarding non-significant coefficients and then sorting the remaining coefficients into combinatorial sequence so that low order coefficients are at the top of the figure. Each row of the figure represents a single coefficient as the binary equivalent of its index. For example ω_5 is a second order coefficient with binary representation 101, meaning that the coefficient measures the interaction between inputs 1 and 3. Dark pixels represent connected inputs in the figure.

Another advantage of the pedagogical rule extraction approach is that it is portable across network architectures as it treats the network as a black box. The Walsh method shares this advantage. A common feature of rule extraction methods is that they accept a reduction in accuracy in return for a simpler set of rules. The rule set can be evaluated on the same test data as the MLP that generated the rules and the trade-off between accuracy and size of the rule set needs to be managed. To reproduce the functionality of the network perfectly with a rule set can require a great many rules and a large number of exceptions (or rules that apply to a very small area of input space). The Walsh method shares this limitation, but for different reasons. As the Walsh functions are a basis set, there is no function that they cannot represent, so there is no network whose behaviour cannot be perfectly reproduced. Any network with binary inputs can have its behaviour perfectly reproduced by a Walsh decomposition, but only by a full decomposition from an exhaustive sample of input,output pairs. This is possible for small networks, but infeasible for networks with large numbers of inputs. A sample of coefficients must then be calculated from a sample of data points, which will lead to an approximate representation of the MLP function.

Classification rules are generally local in that they partition a data set into subspaces that share the same output. This works well when the inputs are numeric as the conditional part of the rule can specify a range. When the inputs are discrete, as in the binary case studied here, the rules cannot partition the input space across a range. In such cases, a rule set may not be the best way to understand a function. Take the character recognition task for example, we can learn more by visualising the coefficients (even just those of low order) than by studying a long list of rules. Walsh coefficients are global as they describe the contribution of an input or group of inputs across the entire input space. This means that it is not possible to partition the input space and so derive simple rules. Every coefficient plays a part in calculating the output from every input pattern. General statements can still be made, however, but they are of the form “When variable $x = 1$, the output increases” or “When variables a and b are equal, the output decreases”. These statements can be generated directly from the coefficients.

8 CONCLUSIONS

An MLP trained on binary input data with either numeric or categorical output neurons can be analysed using Walsh functions. Such an analysis can reveal the relative complexity of different networks, give an insight into the way the function represented by an MLP evolves during learning and shed light on which areas of input space a network has utilised in learning that function. This understanding can help in understanding how well a network will generalise to new data and where its likely points of failure may be. An exhaustive Walsh decomposition is only possible for small networks, but a partial decomposition based on a random sample from the network’s input space can still be used to gain valuable insights into the specific function learned by an MLP.

9 FURTHER WORK

This work has used Walsh functions as its method of complexity analysis, but other basis functions–particularly those suitable for real valued inputs–are also worthy of investigation. As the analysis is not designed to reconstruct the function, merely to shed light on its structure in a human readable form, it should be possible to use an information theoretic measure of interaction such as mutual entropy.

The method provides a useful measure of network complexity that is not based on the number of weights in the network. Training methods that favour simple models over more complex ones often use parameter counts (in the case of MLP, the weights) as a measure of complexity. For example, minimum description length (MDL) methods are often based on parameter counts, but might usefully be adapted to account for other types of complexity such as that described here. The Walsh analysis reveals that two networks of equal size do not necessarily share an equal complexity. The relationship between network complexity and network size is an interesting field of study in its own right. Of course, this analysis is not restricted to use with MLPs. Any regression function may be used, but it is well applied to MLPs as they are difficult to analyse in terms of the structure of their weights alone.

The number of Walsh coefficients to consider grows exponentially with the number of inputs to the network, so it is not possible to exhaustively calculate every possible one in a large network. For networks that contain key interactions at a number of different higher orders, the task of finding the significant coefficients becomes a great problem. Work on heuristics for finding the significant high order coefficients in a sparse coefficient space is ongoing. One approach is to build a probabilistic model of the importance of different neurons and connection orders and sample coefficients from that model. As more coefficients are found, the quality of the model improves and allows the faster discovery of others.

REFERENCES

Augasta, M. and Kathirvalavakumar, T. (2012). Rule extraction from neural networks - a comparative study. pages 404–408. cited By (since 1996)0.

Bartlett, E. B. (1994). Dynamic node architecture learning: An information theoretic approach. *Neural Networks*, 7(1):129–140.

Baum, E. B. and Haussler, D. (1989). What size net gives valid generalization? *Neural Comput.*, 1(1):151–160.

Beauchamp, K. (1984). *Applications of Walsh and Related Functions*. Academic Press, London.

Castillo, P. A., Carpio, J., Merelo, J., Prieto, A., Rivas, V., and Romero, G. (2000). Evolving multilayer perceptrons.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

Gorman, R. P. and Sejnowski, T. J. (1988). Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1(1):75–89.

Hruschka, E. R. and Ebecken, N. F. (2006). Extracting rules from multilayer perceptrons in classification problems: A clustering-based approach. *Neurocomputing*, 70(13):384 – 397. Neural Networks Selected Papers from the 7th Brazilian Symposium on Neural Networks (SBRN '04) 7th Brazilian Symposium on Neural Networks.

Jian-guo, W., Jian-hong, Y., Wen-xing, Z., and Jin-wu, X. (2008). Rule extraction from artificial neural network with optimized activation functions. In *Intelligent System and Knowledge Engineering, 2008. ISKE 2008. 3rd International Conference on*, volume 1, pages 873–879. IEEE.

Jivani, K., Ambasana, J., and Kanani, S. (2014). A survey on rule extraction approaches based techniques for data classification using neural network. *International Journal of Futuristic Trends in Engineering and Technology*, 1(1).

Kamimura, R. (1993). Principal hidden unit analysis with minimum entropy method. In Gielen, S. and Kappen, B., editors, *ICANN 1993*, pages 760–763. Springer London.

Krogh, A. and Vedelsby, J. (1994). Neural network ensembles, cross validation, and active learning. In *NIPS*, pages 231–238.

Kulluk, S., Özbakir, L., and Baykasoğlu, A. (2013). Fuzzy difaconn-miner: A novel approach for fuzzy rule extraction from neural networks. *Expert Systems with Applications*, 40(3):938 – 946. FUZZYSS11: 2nd International Fuzzy Systems Symposium 17-18 November 2011, Ankara, Turkey.

Lecun, Y. and Cortes, C. The MNIST database of handwritten digits.

Li, S. Z. (1995). *Markov random field modeling in computer vision*. Springer-Verlag New York, Inc.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA.

Saad, E. and Wunsch II, D. (2007). Neural network explanation using inversion. *Neural Networks*, 20(1):78–93. cited By (since 1996)22.

Sanger, D. (1989). Contribution analysis: A technique for assigning responsibilities to hidden units in connectionist networks. *Connection Science*, 1(2):115–138.

Uphadyaya, B. and Eryurek, E. (1992). Application of neural networks for sensor validation and plant monitoring. *Neural Technology*, (97):170–176.

Walsh, J. (1923). A closed set of normal orthogonal functions. *Amer. J. Math*, 45:5–24.

Weigend, A. S., Huberman, B. A., and Rumelhart, D. E. (1992). Predicting Sunspots and Exchange Rates with Connectionist Networks. In Casdagli, M. and Eubank, S., editors, *Nonlinear modeling and forecasting*, pages 395–432. Addison-Wesley.

Widrow, B. and Lehr, M. (1990). 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442.

Yao, X. (1999). Evolving artificial neural networks. In *Proceedings of the IEEE*, volume 87, pages 1423–1447.