Swingler *Big Data Analytics* (2016) 1:8
DOI 10.1186/s41044-016-0009-x

Big Data Analytics

RESEARCH Open Access

Structure discovery in mixed order hyper networks

Kevin Swingler

Correspondence: kms@cs.stir.ac.uk
Computing and Mathematics, University of Stirling, FK9 4LA Stirling, UK

Abstract

Background: Mixed Order Hyper Networks (MOHNs) are a type of neural network in which the interactions between inputs are modelled explicitly by weights that can connect any number of neurons. Such networks have a human readability that networks with hidden units lack. They can be used for regression, classification or as content addressable memories and have been shown to be useful as fitness function models in constraint satisfaction tasks. They are fast to train and, when their structure is fixed, do not suffer from local minima in the cost function during training. However, their main drawback is that the correct structure (which neurons to connect with weights) must be discovered from data and an exhaustive search is not possible for networks of over around 30 inputs.

Results: This paper presents an algorithm designed to discover a set of weights that satisfy the joint constraints of low training error and a parsimonious model. The combined structure discovery and weight learning process was found to be faster, more accurate and have less variance than training an MLP.

Conclusions: There are a number of advantages to using higher order weights rather than hidden units in a neural network but discovering the correct structure for those weights can be challenging. With the method proposed in this paper, the use of high order networks becomes tractable.

Keywords: High order neural networks, Structure discovery, Linkage learning

Background

Mixed order hyper-networks (MOHNs) [1] are neural networks in which weights can connect any number of neurons, rather than the usual two. They can be used as regression models or classifiers like MLPs, as content addressable memories like Hopfield networks [2], and as probability density estimators and fitness function models for use in optimisation [3]. MOHNs can form a basis for functions in $f : \{-1, 1\}^n \rightarrow \mathbb{R}$, making them universal function models in that space. They contain no hidden units, using higher order weights instead, which has advantages including improved human readability, faster and more stable weight learning, the ability to compare multiple networks like for like, and finer control over the complexity of the function (and so improved regularisation).

Higher order weights also have a disadvantage: the correct weights to include in a network must be identified. A network of n inputs can contain up to 2^n different weights, so small networks may be fully connected and then pruned by removing insignificant weights. In larger networks, many of the possible weights cannot even be considered as

the computational time required to do is prohibitive. In these cases, weights must be chosen using heuristics that improve the chance of a weight contributing to the function being selected.

This paper addresses the question of how to choose which weights to include in a MOHN when it is impossible to test more than a small fraction of the possible weights. The inclusion or exclusion of different weights has an effect on the variance/bias trade-off of a model. Adding weights improves the training error but can, after a point, increase the test error. Adding weights will also increase the complexity of calculations made by the network, slowing both the learning process and inference.

The remainder of the paper is organised as follows. This section concludes with a short look at existing approaches to discovering structure in graphical models and is followed by a summary of mixed order hyper networks. "Statement of the problem" section describes the problem addressed by this paper and is followed by "Methods" section, which describes the proposed algorithm in detail. The "Results and discussion" section describes some experimental results and is followed by conclusions and suggestions for some future directions for this work.

Existing work

Inspiration for a method of discovering the structure of a MOHN can be sought from other fields where computational networks are built from data. This section considers methods for learning the structure of multi layer perceptrons (MLPs), Bayesian belief networks (BNNs) and Markov random fields (MRFs) and considers methods of feature selection.

Multi layer perceptrons

The standard structure of an MLP contains an input layer, one or more hidden layers and an output layer. Each layer is fully connected to the neurons in the layer above. This structure can be changed dynamically during learning by adding or removing weights or neurons. Approaches to dynamically changing the structure of an MLP during training involve adding or removing weights or neurons and their associated set of weights. Bartlett [4], for example proposed an algorithm that added hidden units each time the training error flattened, and removed units based on an information theoretic measure. He also pointed out that the network weights were often optimised to the structure, and adding new ones didn't allow the network to escape the local optimum it was in. LeCun et al. [5] proposed the Optimal Brain Damage algorithm, which removes weights with low saliency, which is defined based on the second derivative of the cost function. Some algorithms continue to train all of the weights after each iteration of adding or removing weights. Others, such as DMP3 [6] freeze existing weights and only train the newly added ones. Some algorithms add neurons in a restricted structure, for example in the Upstart algorithm [7], the network becomes a tree structure as new neurons are added below existing parent neurons. Although not strictly a structure discovery approach, dropout [8] is a method that drops random neurons during training and then approximates the average output of all the resulting smaller networks at test time. There have also been evolutionary approaches to MLP structure discovery, for example [9] introduces a new crossover operator to allow a GA to discover MLP structure, solving the permutation

problem (being that network structure tells you little about network function). See [10] for a review of evolutionary approaches to neural network learning.

There is some meaning attached to each weight in a MOHN in a way that is not present in an MLP. Adding an additional fully connected hidden unit to a single hidden layer MLP (or removing one) involves all of the input units in a way that is not defined until the weight values are learned, and even then the unit's contribution is unclear. In a MOHN, a weight connects a subset of the inputs and its contribution to the output value is clearly defined.

Bayesian belief networks

A BBN is a directed graph of conditional probability functions in which child nodes represent the distribution across a variable conditional on the values of its immediate parents. Structure discovery in a BNN involves finding a pattern of connectivity that accurately represents the joint distribution across the variables while keeping the complexity of the model low. Complexity can be controlled by limiting the number of parents a node can take (as used in the original K2 algorithm [11]) or by minimum description length (e.g. [12]). Genetic algorithms [13] and evolutionary programming [14, 15] have both been used to discover BNN structure, as have other methods such as branch and bound [16]. In all of these cases, the goal is to reduce the number of possible connections the algorithm has to choose from when adding more.

Markov random fields

A MRF is an alternative representation for joint probability functions using an undirected graph. MRF structure is very similar to that of a MOHN, with connections possible at any order. Structure discovery in MRFs has received less attention than that in MLPs and BNNs. Ravikumar et al. [17] and Lee et al. [18] have both used LASSO in the discovery of structure in MRFs, but address graphs with only pairwise connections.

McCall et al. [19] use statistical tests of independence to discover second order connections between pairs of variables subject to a limit on the number of connections a node can have and follow this with a clique finding algorithm to infer higher order connections [20].

Feature selection

Observing that each possible subset of k variables chosen from all n has an associated candidate weight, the problem of structure discovery may be thought of as a feature selection task where each subset represents a single feature. In data mining, feature selection is generally applied to choosing a subset of variables, and can be split into two main approaches. Wrapper methods [21] combine feature selection with a chosen modelling method, and filter methods work independently of a modelling method as a pre-learning step. Many methods, such as stepwise regression [22] use a greedy approach that involves growing a feature set incrementally. Other methods employ an evolutionary approach [23, 24].

Mixed order hyper networks

A Mixed Order Hyper Network is a neural network in which weights can connect any number of neurons. A MOHN has a fixed number of n neurons and $\leq 2^n$ weights, which

may be added or removed dynamically during learning. The state of the MOHN is determined by the vector of neuron outputs, $\mathbf{u} = u_0 \ldots u_{n-1}$. This paper discusses binary MOHNs, where $u_i \in \{-1, 1\}$. The structure of a MOHN is defined by a set, $\mathbf{W}$ of real valued weights, each connecting $0 \leq k \leq n$ neurons. Each weight $W_j \in \mathbf{W}$ is defined by a tuple, $W_j = (w_j, Q_j)$ where w_j is its value and Q_j is the set of neurons it connects. The weights define a hyper graph connecting the elements of $\mathbf{u}$. Figure 1 shows an example network. The main difference between a MOHN and a normal neural network is that the weights can connect any number of neurons. For example, in Fig. 1 the weight w_7 connects the three neurons, u_0, u_1, and u_2.

When used as a regression model, a MOHN defines a function $f : \{-1, 1\}^n \rightarrow \mathbb{R}$, where the output of the function is calculated by

$$\hat{y} = \sum_{w_j \in \mathbf{W}} w_j \prod_{u \in Q_j} u \tag{1}$$

When used as a content addressable memory, neurons are updated by calculating an activation value and then applying a threshold as follows:

$$a_i = \sum_{j:u_i \in Q_j} \left(w_j \prod_{k \in Q_j \setminus i} u_k \right) \tag{2}$$

where $j : u_i \in Q_j$ makes j enumerate the index of each weight that connects to u_i and $k \in Q_j \setminus i$ indicates the index of every member of Q_j, except neuron i itself. A neuron's output is then calculated using the threshold function in Eq. 3.

$$u_i = \begin{cases} 1, & \text{if } a_i > 0 \\ -1, & \text{otherwise} \end{cases} \tag{3}$$

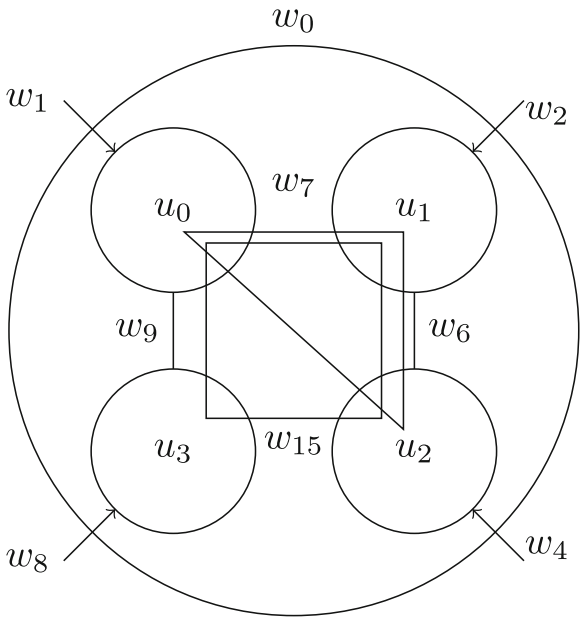

Fig. 1 A four neuron MOHN with some of the weights shown. w_7 is the triangle and w_{15} is the square

Weight values may be learned online using the delta rule in the form:

$$w_j = w_j + \alpha(f(\mathbf{x}) - \hat{f}(\mathbf{x})) \prod_{u \in Q_j} u \tag{4}$$

where $\alpha < 1$ is the learning rate. Each weight should be initially set to the difference between the mean function output when the input to the weight has odd parity and even parity, as below.

$$w_j = \frac{1}{2}\langle f(\mathbf{x}|Q_j^+)\rangle - \langle f(\mathbf{x}|Q_j^-)\rangle \tag{5}$$

where $\langle f(\mathbf{x}|Q_j^+)\rangle$ is the expected value of $f(x)$ when the parity of the values in Q_j is positive, and $\langle f(\mathbf{x}|Q_j^-)\rangle$ is the expectation when the parity across Q_j is negative.

The LASSO algorithm [25] may also be used to estimate the weight values in a MOHN. LASSO performs regression with an additional constraint on the L^1 norm of the weight vector. The learning algorithm minimises the sum:

$$\sum_{\mathbf{x} \in D}(f(\mathbf{x}) - \hat{f}(\mathbf{x})) + \lambda \sum_{w \in W} |w| \tag{6}$$

where λ controls the degree of regularisation. When $\lambda = 0$, the LASSO solution becomes the ordinary least squares solution. With $\lambda > 0$ the regularisation causes the sum of the absolute weight values to shrink such that weight values can be forced to zero. This not only allows LASSO to reject input variables that contribute little, but also to reject higher order weights that are not needed. When training a MOHN, the input to the LASSO regression algorithm is a vector containing a variable for each weight in the model. The value associated with each weight on which the regression is performed is the product of the input values connected to that weight. For a comparison of the MOHN learning rules, see [26].

Swingler [1] showed how a fully connected MOHN, trained on an exhaustive sample of input, output pairs from any function is able to represent that function perfectly. Such a MOHN forms a basis for such functions that is equivalent to the Walsh basis [27]. Any weight that is not required to model the function will go to zero, and may be removed, leaving a sparse structure that still fully represents the function. Normally, an exhaustive sample of data is not available and the network cannot be fully connected due to the number of connections required. In these circumstances, weights must be added and removed in an iterative process designed to discover the non-zero weights. No weight value will go completely to zero based on only a sample of data, so a significance test is required when considering whether to keep or remove a weight. The goal of structure discovery in a MOHN is to add and learn coefficients for those weights that would not go to zero in the full model and to exclude those that would. In principle, if the right weights can be found, any function may be represented to arbitrary accuracy.

Statement of the problem

By including the right weights with the right values, a MOHN can represent any function in $f : \{-1, 1\}^n \rightarrow \mathbb{R}$. The set of weights present in a given MOHN define its structure, while the values on those weights further define the function's shape. Each subset of weights (i.e structure) is capable of representing a subset of the possible function shapes (i.e. input $\rightarrow$ output mappings). For example, the subset containing all of the first order

weights and no others is capable of representing any linear function of the inputs. There are 2^{2^n} possible structures.

Functions may be designed by hand by specifying the structure and values of the weights. This has been done with Hopfield networks to use them as optimisation fitness functions to model the travelling salesman problem [28, 29] and graph colouring problems [30]. More often, however, the structure and shape of the function must be discovered from data.

Consider two types of scenario in which it is useful to build a model of a function. In the first, data representing observations or measurements have been collected and the requirement is to model the underlying structure of the relationship between inputs and output in the data. This is the traditional approach in machine learning, data mining or statistical learning. Often the data is fixed (cannot be sampled at random) and noisy.

In the second scenario, a function that produces the output resulting from a given input already exists and can be sampled randomly. Reasons for sampling this function to generate data to train an alternative model (in this case a MOHN) are most often found when the output of the function needs to be optimised and evaluating the existing function is costly. A MOHN can be used as a fitness function model as part of an optimisation task as local optima can be quickly identified and even removed [31]. In such cases, it is usually assumed that there is no noise in the output (the fitness score) and that the function can be sampled at random.

In regression models such as these, the weight values may be calculated independently (for example, in single linear regressions) and joined if the variables are uncorrelated (orthogonal). However, if pairs of variables are correlated, then the weights must be calculated by considering all of the variables together. The correlation between variables depends to some extent on whether the training data is a fixed sample (as is usual in machine learning) or random samples from a function (as is usual in optimisation), the latter case allowing correlations to be reduced by the sampling regime. With a uniformly random sampling regime, correlations between inputs diminish as the sample size grows. A consequence of this is that the value taken by any single weight will depend in part on the presence of other weights in the network, so a weight that appears insignificant on its own may gain significance as the network grows.

The problem of MOHN structure discovery involves discovering a sufficient number of the non-zero weights to achieve an acceptably low error without the need to test every possible weight. The difference between MOHN structure discovery and feature selection lies in the fact that many candidate weights will never be tested. The choice of new weights must be guided by the existing network structure.

To discover the correct structure in a MOHN requires a sample of training data, which may be a fixed set of samples or arbitrary samples from the function to be learned. A method for learning the value to assign to a weight is also needed along with a method for testing the statistical significance of that value. Of course, a method for choosing which weights to consider (and so expend the computational effort of calculating a weight) is also required.

It is also important to impose some form of regularisation on the structure being built to avoid over fitting. Generally, regularisation is done by limiting the size of the weights, choosing a subset of variables to include in a model (feature selection) or controlling some aspect of a statistical model's complexity (the number of hidden units in an MLP, for

example). With a MOHN, one can (indeed, must) be more explicit about complexity by choosing the correct order for the weights.

Inspiration from existing work

We have already discussed various existing methods for discovering structure in different graphical function models. A common approach to MLP structure learning, Bayesian network learning and feature selection is to use evolutionary computation. An evolutionary approach has been discounted for this work for a number of reasons. Firstly, a population of networks with different structures would share parameters, leading to duplicated weight estimation calculations. This could be solved by maintaining a super set of weights so that each was only estimated once, but that super set would constitute one large MOHN in its own right, which would allow a better estimation of weight importance than that taken from a population of smaller networks.

Approaches to growing and pruning MLPs are hindered by the fact that there is no accessible meaning attached to hidden units. A MOHN, on the other hand, is transparent in the sense that the role of a weight is clearly defined in terms of a combination of input variables. The concept of training a small network until the error flattens and then adding more weights may be used, but the weights that are added may be chosen with some knowledge of their role, unlike the approach for an MLP. Much of the difficulty in estimating a MRF is due to the problem of evaluating the partition function, but the use of LASSO to regularise a network has been demonstrated, and will be considered in this work too.

Greedy approaches to feature selection often require the entire set of individual features to be evaluated at the first step. In subsequent steps, new combinations that are limited to those containing the best feature from the first step are explored. A MOHN is an unsuitable subject for this approach as there are 2^n possible features—too many to consider even once. When building a large MOHN, there will be a great many weights that can never be evaluated. New candidate weights must be picked without knowing the estimate of their value, but based on the two things that can be known about them: the number of neurons they connect (their order) and the role of those neurons in the current network.

Many methods limit the degree of connectivity allowed in a graphical structure and these are often applied on at the level of single nodes (for example, no node may have more than x links). This idea is extended in this work, firstly by controlling the order of connections (the number of nodes each weight is connected to) and secondly by allowing both an exploratory phase, where nodes that have not yet found a use are preferred over those with several connections already, and an exploitative phase where neurons that have proved useful already are more likely to be considered as part of higher order connections. Exploration has the effect of restricting the number of connections on nodes and exploitation is more akin to the results of greedy feature selection or clique based MRF structure discovery.

Methods

The structure discovery algorithm proposed here takes an on-line, stepwise approach. Weights are added and removed as the algorithm progresses. Regularisation is applied by the choice of weights to add or remove, but can also be introduced into the regression algorithm used to learn the weight values.

For networks of even moderate size, it is impossible to test every possible weight, even in isolation, so a method for choosing which weights to consider is needed. A probability distribution is maintained, from which candidate weights are sampled and added to the model. The model then undergoes a training phase after which all the weights are tested for significance. Insignificant weights are removed and as the model grows, the weight picking distribution is altered to reflect its emerging structure.

At its most abstracted level, the algorithm proceeds as follows:

Algorithm 1 Probability distribution based structure discovery algorithm.

Start with an empty network, $W = \emptyset$
Initialise a distribution over possible weights, $P(w)$
repeat
  Sample some weights from the distribution $P(w)$
  Calculate the weight values for the resulting network
  Regularise by removing some weights
  Update the weights distribution, $P(w)$
  Calculate the test error
until The test error is sufficiently low

A number of decisions are required when implementing Algorithm 1 in detail. They are:

- A representation of the probability distribution over unpicked weights
- A method for updating the weight picking distribution
- A choice of learning rule for calculating the new weight values
- A choice of regularisation method for removing weights

The following sections consider these points in more detail. These sections compare a number of choices for each step and are followed by an example algorithm based on one choice from each.

Representing the probability distribution across weights

The structure discover algorithm is based on the premise that as not all possible weights can even be considered, heuristics for picking weights that have a higher chance of proving useful must be used. The solution is to maintain a probability distribution over the possible weights where the probability of a weight being selected is proportional to its chance of being useful. This requires a representation of the space of possible weights and a method for shaping a function to reflect a weight's potential usefulness.

Rather than use a single distribution that spans every possible weight, in this work two distributions are used. One covers the order of the weight and the other covers the probability of each neuron in the network being connected to that weight. Let the probability distribution over the weight orders of an n neuron MOHN be

$$Po(o) : o \in \{1 \dots n\} \tag{7}$$

and the probability of picking a neuron to be connected by the current choice of order, o be

$$Pn(i) : i \in \{0 \dots n-1\} \tag{8}$$

The order, o is sampled first, and then a subset, Q of o neurons are sampled without replacement from $Pn(i)$. Both distributions are discrete—there are n possible orders and n possible neurons to choose from—so their representation need not be from any parametrised class. The probabilities can be represented as a vector of size n with the usual constraint that each must be between 0 and 1 and they must sum to 1. How $Po(o)$ and $Pn(i)$ evolve as the algorithm progresses is addressed next.

Updating the weight picking distributions

At the first iteration of the algorithm, the distributions $Po(o)$ and $Pn(i)$ must be set up manually. This presents an opportunity to include any prior knowledge that exists about the function to be modelled and also allows some control over the complexity of the model to be imposed.

Distribution over weight orders

The choice made for the algorithm described here is to initialise $Po(o)$ with a discrete Laplace centred at order c where initially $c = 1$ and c is incremented as lower order weights are either used or discarded.

$$Po(o) = \frac{1}{2b} e^{-\frac{|c-o|}{b}} \tag{9}$$

where b controls the width of the distribution. As the distribution is only sampled at integer points in a limited range, it must be normalised so that the probabilities sum to 1. This is done by summing Eq. 9 over the range of possible orders $(1 \ldots n)$ to give a constant, Z and then calculating the probability of picking a weight from each order as $\frac{po(o)}{Z}$.

In the early iterations of the algorithm where $c = 1$, there is a high probability of picking first order weights and an exponentially decreasing probability of picking weights of higher order. In subsequent iterations, $Po(o)$ is updated in two ways. Firstly, c is increased to allow the algorithm to pick weights with higher orders and secondly the values of existing weights are used to shape the distribution to guide the algorithm towards orders that have yielded high value weights already.

The weight order probability distribution, $Po(o)$ is updated by counting the proportion of weights in the current network that are of each order. Let this vector of proportions be $\mathbf{p} = p_1 \ldots p_n$ where p_i is the number of weights at order i divided by the total number of weights in the network. These proportions are then used to update $Po()$ along with an updated version of the discrete Laplace distribution as follows:

$$Po(i) \leftarrow (1 - (\alpha + \beta))Po(i) + \alpha p_i + \beta \frac{1}{2b} e^{-\frac{|c-o|}{b}} \tag{10}$$

where α is the proportion of the weight order counts, $\mathbf{p}$ to include in the update and β is the proportion of the current order mode, c that is included such that $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$ and $0 < \alpha + \beta \leq 1$.

If $\alpha + \beta = 1$ the new distribution is a mixture of the current distribution of weight orders in the MOHN and the discrete Laplace distribution with a mode of c. If $\alpha + \beta < 1$ the distribution retains some memory of its previous shape, weighted by $1 - (\alpha + \beta)$. In the experiments reported in this paper, the values $\alpha = 0.6$, $\beta = 0.2$ were used and found to work well.

The weight order mode, c needs to be manipulated as learning progresses. In the work reported here, c was set to equal the lowest order with remaining unsampled weights. As

lower weight orders are exhausted, the mode naturally moves up. Of course, this does not rule out higher rates being sampled - the α component will bias the sampling towards higher orders if they prove useful. The smaller the value of b, the faster the weight order distribution drops towards zero as it moves away from c.

Distribution over neurons

Once the order, o of a new candidate weight has been sampled, the o neurons that it connects must be picked. These neurons are picked from a distribution, $Pn()$ that evolves as each neuron is picked. The shape of $Pn()$ is determined by a number of factors. Prior knowledge can be included by increasing the probability of variables that are known to be useful. If no prior knowledge is available, then $Pn()$ starts off as a uniform distribution. Once there are some weights in the network, $Pn()$ is determined by a mixture of the prior knowledge and the role played by each neuron in the existing network. To connect a weight of order o, there are two phases to the neuron picking procedure. The distribution from which the first neuron is picked is shaped by the contribution each neuron is already making. In exploratory mode, neurons that have not yet played a role are favoured and in exploitative mode, neurons that are already well connected are more likely to be picked. Subsequent neurons, up to o, are picked from a distribution that is reshaped by the set of neurons that are already connected to the existing set under construction at orders other than o.

The trade-off between exploration and exploitation can be managed. Exploration in this case means favouring neurons that have few or weak connections on the assumption that they do have a role to play, but it has yet to be found. Exploitation refers to picking neurons that already have connections on the assumption that those which have proved useful at some orders will also be useful at others.

The first step in picking the o neurons is to pick the first with a probability proportional to the contribution it makes to the model. Define the contribution made by neuron i as being the sum of the absolute values of the weights connected to neuron i.

$$C(i) = \sum_{w_j : x_i \in Q_j} |w_j| \tag{11}$$

where $w : x_i \in Q_j$ iterates over the weights connected to x_i. The proportion of the total contribution of all neurons made by neuron i is

$$Cp(i) = \frac{C(i)}{\sum_{j=0}^{n-1} C(j)} \tag{12}$$

Now let ρ control the level of exploration, such that $\rho = -1$ means full exploration (bias the search towards unused neurons), $\rho = 1$ means full exploitation (bias the search towards well used neurons) and $\rho = 0$ leads to a uniformly random choice among the neurons. Any other value of $-1 < \rho < 1$ balances the degree to which exploration or exploitation is made. The probability of picking neuron i is

$$Pn(i) = \begin{cases} (1-\rho)\frac{1}{n} + \rho Cp(i), & \text{if} \rho > 0 \\ (1+\rho)\frac{1}{n} - \rho(1 - Cp(i)), & \text{otherwise} \end{cases} \tag{13}$$

Equation 13 causes the degree of exploration to vary when $\rho < 0$ and causes the degree of exploitation to vary when $\rho > 0$. The closer to zero the value of ρ gets, the more uniformly random the neuron selection becomes.

The first neuron connected by the weight being built is selected by sampling the density defined in Eq. 13. Once the first neuron, x_i is picked, $Pn(i)$ is updated in two ways. Firstly, the chosen neuron has its probability set to zero, so $Pn(i) = 0$, to prevent it being picked again. Then, $Pn()$ is updated so that other neurons that are already connected to x_i at other orders have their probability of being chosen increased as follows

$$Pn(i) \leftarrow (1 - (\delta))Pn(i) + \delta \sum_{w:w \in V, x_i \in w} |w| \tag{14}$$

where V is the set of weights that are connected to any of the neurons that have been picked for the weight currently under construction. The sum is over all weights that are connected to both x_i and any of the other neurons already chosen for the new weight. The parameter $\delta \in \{0 \ldots 1\}$ controls the mix of the previous shape of $Pn()$ and the update. High values of δ cause the algorithm to favour neurons that are connected to those already in the set being built, and low values cause it to favour the contribution of each neuron in isolation. In this way sets of neurons that form cliques due to low order connections have a higher probability of being connected at higher orders. Finally, when the number of neurons picked equals $o-1$, the probability associated with all neurons already connected to those neurons at order o is set to zero to ensure an existing weight is not picked.

Weights are not added and learned one at a time, they are added in batches. In the experiments reported here, the number of weights added at each iteration of the algorithm was set to equal the number of input neurons.

Efficient weight picking

Once a weight is already in the model or has been tested and discarded, it is considered *used*. Only unused weights should be considered for addition to the model. When the ratio of available weights to used weights is high, it is efficient to simply pick a random weight using the procedure above and check that it is not already in the network or in a list of weights that have been considered but removed from the network. To avoid unuseful weights being repeatedly added and removed, a list of discarded weights is maintained. Newly sampled prospective weights are first compared to the members of this list and not added if they have been recently tried. As weights may appear unuseful as part of a poorly structured network, but later prove to be of use when the rest of the structure is in place, the discard list is periodically emptied to allow weights a second chance of inclusion.

This approach becomes inefficient when there are very few (or no) weights available at the chosen order, meaning very many choices are required before an available weight is found. To ensure that there are available weights at the chosen order, the algorithm keeps count of how many weights of each order have been used. There are $\binom{n}{o}$ possible weights at each order, o, so when the order o count reaches this figure, the probability of picking a weight at that order is forced to zero.

Another efficiency enhancement to the algorithm is the inclusion of a 'mopping up' procedure that is activated when the number of used weights at order o reaches a certain percentage of the total (a threshold of 90 % is used in this paper). When the order o count reaches the threshold, the few remaining weights at order o are automatically added to the model and assessed. This allows the probability of picking from order o to then be forced to zero, thus avoiding many fruitless picks from that order.

Learning rules for the weights

We have described two learning rules for a fixed structured MOHN: the online delta method and the offline LASSO. Each method has different attractive properties for estimating weight values during structure discovery. At each iteration of the structure discovery algorithm, a small proportion of new weights are added to a network whose existing weight values are likely to already be close to the correct value. As the delta rule is incremental, it can take advantage of this fact rather than starting a new, empty network. New candidate weights can be initially set using Eq. 5, after which the entire new network is improved using Eq. 4. Algorithm 3 describes this process.

The nature of the regularisation in LASSO means that weights that are not needed have values forced to zero, removing the need for an additional weight removal decision, but at the cost of estimating the entire network structure from scratch at each iteration. Algorithm 4 describes the LASSO network update method. A single value for λ may be chosen or, as is usual in the application of LASSO, a number of different settings for λ may be tried.

Regularisation and weight removal

Regularisation refers to the process of introducing additional constraints to a machine learning process to prevent over fitting. This often takes the form of a penalty on complexity or a bound on the norm of the learned parameters. Regularisation can also involve the use of an out of sample test set. All of these methods may be applied to a MOHN but

Algorithm 2 Algorithm for picking a new set of weights to add to an existing MOHN

```
Let U be the set of discarded weights
Let W be the current network weights
Let V = ∅ be a new weight set
Let n be the number of weights required
Let Po() be the probability distribution over the possible weight orders
Let Pn() be the probability distribution over the possible neurons
repeat
    Pick an order, o from Po()
    if U ∪ W ∪ V is more than 90 % full at order o then
        Add the rest of the unused order o weights to V
    else
        repeat
            Set an initial distribution across the neurons, Pn()
            Update Pn() according to current network structure
            Choose a new neuron u_i from Pn()
            Add u_i to the neuron set connected by the new weight
            Update Pn() based on the connectivity of u_i
        until o neurons have been selected
        Ensure u_i ∉ U ∪ W ∪ V
        Add u_i to V
    end if
until |V| >= n
```

Algorithm 3 Weight update algorithm for delta rule learning

Let W be the current network weights
Let V be a set of new weights, chosen using algorithm 2
Initialise the weights in V using parity learning, Eq. 5
Add the weights in V to W so $W = W \cup V$
Run delta rule learning, Eq. 4 on W until the training error flattens

Algorithm 4 Weight Update Algorithm for LASSO learning

Let W be the current network weights
Let V be a set of new weights, chosen with algorithm 2
Add the weights in V to W so $W = W \cup V$
Estimate the weight values using LASSO based on Eq. 6

the main means of regularising a MOHN is the removal of insignificant weights. In this section, two options for weight removal are considered. It is important to remove weights because the rules for updating the probability distributions from which new weights are chosen depend on the presence or absence of weights in the model. It is also desirable to keep the model small for reasons of parsimony, to avoid over fitting and to reduce the time required during learning and inference.

Equation 5 shows a first approximation to the correct value of a weight based on the difference between the mean function output for even and odd parity inputs to the weight. In cases where the difference between the distributions of the function output under each of the two parity conditions is not statistically significant, the weight may be excluded. A t-test is used to compare the mean function output between the odd and even parity input sets, allowing weights with a p-value above a chosen threshold to be removed. Some fine tuning of the critical p-value (*pcrit*) is required to ensure that the algorithm does not discard too many or too few weights. In this work, the critical p-value starts high ($pcrit = 0.3$) and diminishes towards a lower limit ($pcrit = 0.001$) as the network grows. The t-value is calculated as follows

$$t = \frac{w}{\sqrt{\frac{\sigma_w}{|D|}}} \tag{15}$$

where w is the weight value, σ_w is the variance of $f(\mathbf{x})$ and $|D|$ is the number of training data points.

Learning the weight values using LASSO forces some weights to zero, making the choice of which weights to remove from the network almost trivial. Removing all the weights with zero value is the simple part, but it is still necessary to choose the value of the regularisation parameter, λ. One approach is to calculate the coefficients at a number of different settings of λ and choose which weights to remove from that set.

The full algorithm

The full structure discovery algorithm is presented below, with reference to partial algorithms already described above.

Algorithm 5 Probability distribution based structure discovery algorithm.

Start with an empty network, $W = \emptyset$
Initialise an empty used weight set, $U = \emptyset$
Initialise the probability distribution $Po()$ over the possible network orders $m = 1 \ldots n$
repeat
    Use algorithm 2 to select a new set of candidate weights, V
    Train and merge V and W using either algorithm 3 or 4
    Remove insignificant weights based on either a t-test or the zero valued weights after LASSO
    Add the removed weights to U
    Recalculate $Po()$ using Eq. 10
    Calculate the test error
    Update the parameters c and $pcrit$
until The test error is sufficiently low or no longer improves

One advantage of this approach to regression is that a lot of information is available during network learning. Firstly, the maximum number of possible weights at each order is calculated as $\binom{n}{o}$ where o is the order and n is the total number of inputs. As the algorithm progresses, the number of weights of each order in the network may be reported and compared to the possible total. This gives a measure of the complexity of the network compared to possible complexity. By reporting the list of tried and discarded weights, it is also possible to monitor how much of the weight space the algorithm has sampled.

The user might choose to set an upper limit on the order of weights added to the network according to the size of the training sample.

Setting the control parameters

The discussion so far has introduced a number of control parameters for controlling the speed at which probability distributions evolve, the trade-off between exploration and exploitation, and the sensitivity of the weight removal process. It may appear that there are a lot of parameters to balance, but in reality, most of them can be fixed at default values and never changed. For example, α, β and δ all control the rate at which the weight probability distributions forget their previous shape. In the work reported here they were set at $\alpha = 0.6$, $\beta = 0.2$ and $\delta = 0.8$. The critical value for p in the t-test, $pcrit$ starts at 0.3 and reduces to 0.001, reducing by a factor of 0.7 on each iteration of the algorithm. The parameter b, which controls the rate at which the discrete Laplace distribution over the weight orders drops to zero was fixed at 1, which concentrates the sample on the mode and 1 step either side of it.

Results and discussion

Many neural network training algorithms are tested on data sets that are either from a public source, such as the UCI database or are of specific interest to the authors of the paper. This paper takes a different approach and uses a set of known test functions that generate training data. Functions with a known structure are chosen so that the results of the algorithm may be evaluated on how well a model structure matches the desired structure as well as by training error. One of the motivations behind the design of the MOHN is

that it may be used as a fitness function model for heuristic optimisation tasks [3]. Taking inspiration from this fact, the functions to be learned in this paper are fitness functions to optimisation problems. In such problems, the MOHN acts as a meta-heuristic as it knows nothing about the nature of the problem. It simply has a fitness function that it must learn to replicate. Having learned the function, the solutions will be attractors in the energy function of the network.

Graph colouring function

The first test is on the graph colouring problem, which involves searching for a way to colour the nodes of a graph so that no two connected nodes share a colour, using a limited palette of colours. The input is encoded in d groups of k bits where k is the number of colours available on the palette and d is the number of nodes in the graph. The colour is encoded by allocating each of the k bits in each block a colour and using patterns where only one bit (that corresponding to the chosen colour) is set to one. The fitness function has two components. One ensures that only one colour is chosen in each group of k and the other counts the number of edges that join same coloured nodes. The function is implemented as follows:

$$f(\mathbf{x}) = \frac{|e_d|}{|e_t|} \frac{\sum_{i=1}^{d} \frac{k-|i_1|}{k-1}}{d} \tag{16}$$

where $|e_d|$ is the number of edges with a different colour at each end, $|e_t|$ is the number of edges in the graph and $|i_1|$ is the number of inputs in block i with a value 1. The output of the function is 1 when a correct colouring for the graph is present and each block has only one bit set to one. The function has interactions within each block at orders up to k, which control the only-one-colour constraint and additional high order weights between blocks that are connected in the graph.

Figure 2 shows the error profile from 100 trials of learning the graph colouring function, comparing MLPs to MOHNs. Each trial involved a network of ten nodes with twelve edges added at random, but in such a way that would permit a four colouring. The network input consisted of 40 neurons (ten groups of four) and the target output was the fitness value of the given input pattern when evaluated using Eq. 16. Each resulting function was sampled repeatedly to produce on-line training data. The MLPs had 80 hidden units in a single hidden layer and were trained with the standard back propagation of error algorithm. The MOHN was trained using the delta learning rule. The training error at each epoch was recorded for each network and then averaged over the 100 trials. Figure 2 shows the mean and two standard deviation range of the training error for the MLP and the MOHN as training progressed. The MOHN is consistently faster to learn and more accurate than the MLP.

Comparing LASSO and delta learning

This paper has presented two possible learning rules for estimating the weight values on the network structure as it evolves: Delta learning and LASSO. The design justification for using the delta rule after the addition of new weights is that the existing weights should already be close to their desired values so intuition suggests that this will be faster than using LASSO across the whole network. This section presents some experimental results comparing the two using another well known fitness function, known as the k-bit trap.

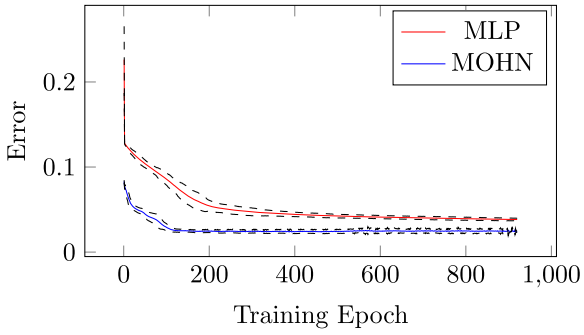

Fig. 2 Mean training error descent during learning of a graph colouring fitness function for an MLP (*top*) and a MOHN performing structure discovery (*bottom*). Dashed lines are at two standard deviations from the mean

K-bit trap functions are defined by the number of inputs with a value of zero. 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 0. 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. The function that evaluates an input vector, $\mathbf{x}$ is

$$f(\mathbf{x}) = \sum_{\mathbf{b} \in \mathbf{x}} f(\mathbf{b}) \tag{17}$$

where

$$f(\mathbf{b}) = \begin{cases} c_0(\mathbf{b}) - 1, & \text{if} c_0(\mathbf{b}) > 0 \\ k, & \text{if} c_0(\mathbf{b}) = 0 \end{cases} \tag{18}$$

The 'trap' part of the function is defined by the second case in Eq. 18, which requires the output to be high when there are no zero values set in the inputs, counter to the first case, which causes the increased presence of zeros in the inputs to increase the function output. Most input patterns suggest a first order model would suffice, with only the patterns where all k bits are set to 1 contradicting that. The trap is that learning algorithms might favour a simple model and fail to add the high order components required to avoid a large error whenever all the inputs are set to 1.

Speed and accuracy (in terms of root mean squared error) were compared over 100 runs of the structure discovery algorithm as it attempted to learn the structure and weights of a 4-bit trap repeated 5 times over 20 input variables. Weight removal with the delta rule was based on the results of a t-test and for LASSO, weights with values forced to zero were removed. Figure 3 shows training error by iteration of the structure discovery algorithm, averaged over the 100 trials. LASSO consistently achieve a lower error, but took on average over ten times as long to compute as the delta based learning. Figure 4 shows the median, inter quartile range and full range of the time taken by delta learning and LASSO to find the correct structure for the same problem.

Local minima

A well known limitation of error descent learning algorithms for MLPs is the tendency to settle into a state that represents a local minimum in the cost function. Based on experimental data, [32, 33] suggests that this is due to the fact that the MLP must combine

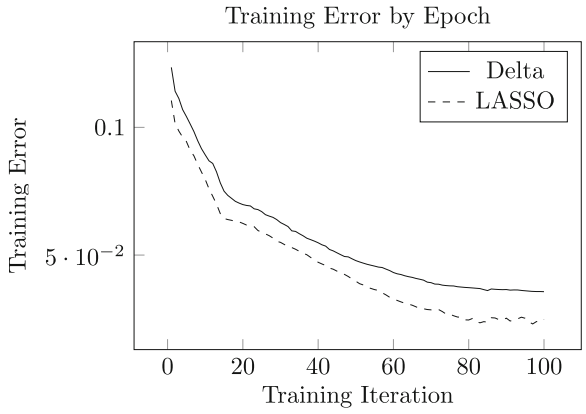

Fig. 3 The mean training error for LASSO and delta rule learning during structure discovery

learning the structure of the function and the parameters that fit that structure to the data at the same time, using the same weights. Training a fixed structure MOHN does not run the risk of settling in local minima—all of the learning rules are deterministic and will produce the same result, which represents the global error minimum for the given structure. However, the global minimum differs from structure to structure, so any fixed structure can be considered a local minimum. If a structure discovery algorithm is not able to move from any one structure to a better one, then its current state is a local minimum. The proposed algorithm can be made to avoid local minima simply by ensuring some degree of exploration. Of course, it might take an impractically long time to find the right weights in this way, but the algorithm will not become trapped.

A simple demonstration can be found in a concatenated XOR function, in which the output is the sum of the result of taking inputs in non-overlapping adjacent pairs, performing XOR on each pair, and summing the results. The function contained 20 inputs and was sampled to produce data to train an MLP and a MOHN performing structure discovery. As one might expect, the structure discovery algorithm was able to very quickly find the correct weights, as the only interactions are at order 2. With 20 inputs, there

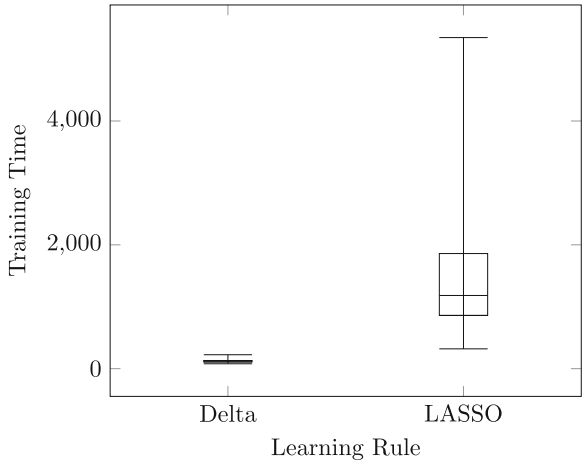

Fig. 4 Median, inter quartile range and full range of time in seconds taken to learn a 4 bit trap function over 20 bits

are 10 non-zero weights—those connecting adjacent pairs. The algorithm finds no useful weights at other orders, so the weight picking probability distribution very quickly becomes close to zero everywhere except at order 2, where it is close to 1. The algorithm then quickly includes all the second order weights and the regularisation removes all except the required 10. 40 attempts at learning the function with an MLP and with a MOHN were made. On all trials, the MOHN found the exact weights needed and the error went to zero in an average of 8 iterations of the algorithm. The MLP converged more slowly and failed to reach zero after 200 iterations, at which point most of the attempts had reached an error plateau. The MLP also showed far greater variance across its models, with some becoming trapped at higher error levels than others. Figure 5 shows the results. The set of lines to the bottom left of the plot are the error traces from the MOHNs and those that spread out towards the upper right hand side are from the MLPs. A set of local optimum at an error of around 0.001 is clearly visible in the MLP data, as are a few that settle below that point.

Visualising network structure

During the training of the MLP, very little information is available compared to that from a MOHN. As the MOHN learns, the weight profile and the weight probability distributions may be reported and analysed to understand the progress being made. This section illustrates the structure discovery process using the same k-bit trap function described above. A visual representation of network structure is used to produce an image with n columns and $|\mathbf{W}|$ rows where each column represents a neuron and each row represents a single weight. The pixel at coordinate (i, j) is plotted if w_j is connected to u_i and its colour reflects the strength of the connection. If w_j is not connected to u_i, then no pixel is plotted. The weights are sorted in combinatoric order, with first order weights at the top of the image, second order weights below them, and so on. If a weight is not present in the network, it does not appear in the image, so the height of the image depends on the number of weights in the network.

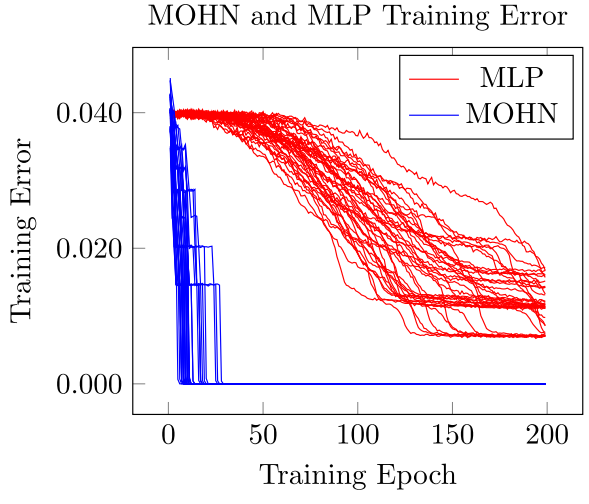

Fig. 5 Traces of training error over 40 different attempts at training a MOHN and an MLP on a concatenated XOR function over 20 inputs.The lower set of lines are from the MOHN and the upper set from the MLP

Figure 6 shows an example for a correctly fitted 5-bit trap problem over six repeated traps. The interactions among the neurons in each trap are plain to see, as is the lack of inter-trap connections. Images such as Fig. 6 provide an insight into the function that has been learned and the complexity of the representation of that function. They also allow for a human led phase of learning. If a small number of weights were missing from Fig. 6, it would be easy for the human eye to spot them and add them manually to the model for a final round of learning.

The structure of a network may be monitored during training both by full network representations such as that in Fig. 6 and by summary information about weight orders and neuron contributions. Figure 7 shows the structure of a network at selected points during the structure discovery of a 5-bit trap function. Green pixels indicate positive weights and red indicate negative. Weights at each order are easily identifiable. The behaviour of the algorithm is exposed as the network first grows (partly due to a higher critical value for the t-test) and then shrinks to the correct structure. It is also clear that the network has cleared the insignificant weights away from the lower orders before the higher order weights. Monitoring the image of a network allows the user to understand the current

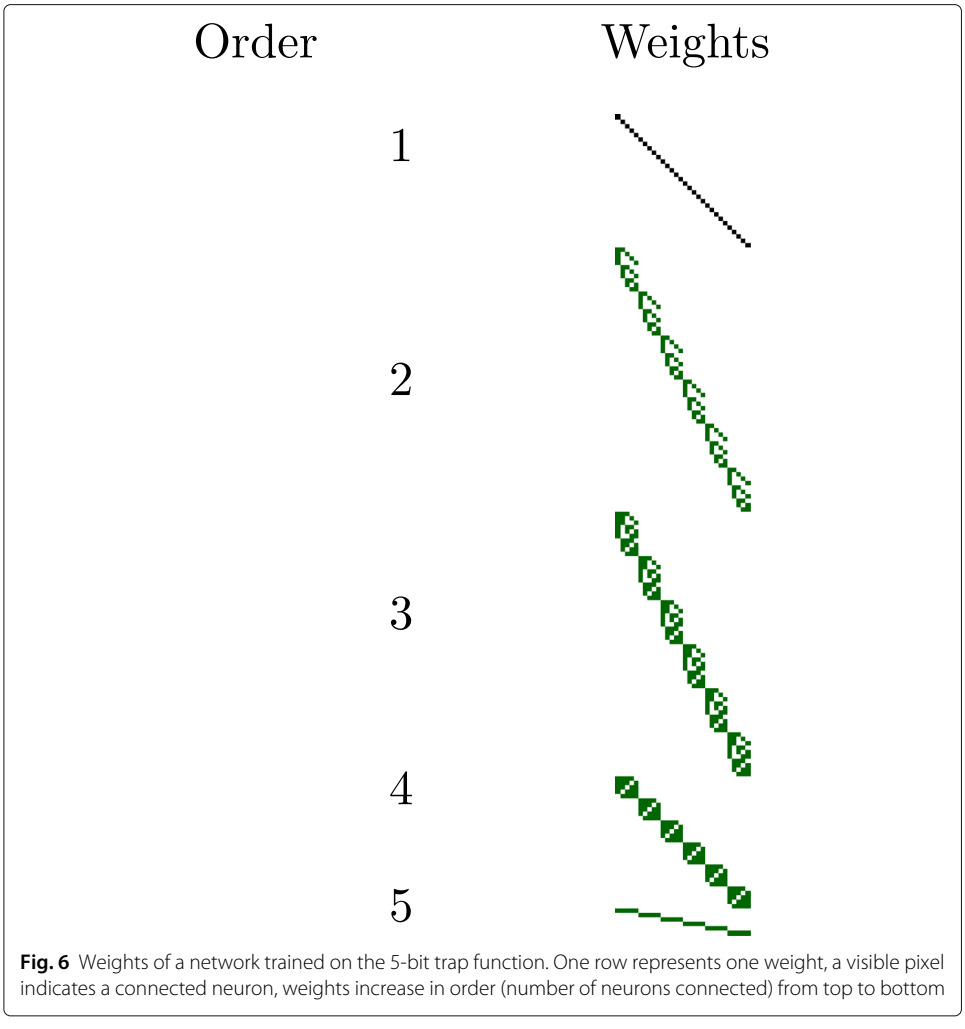

Fig. 6 Weights of a network trained on the 5-bit trap function. One row represents one weight, a visible pixel indicates a connected neuron, weights increase in order (number of neurons connected) from top to bottom

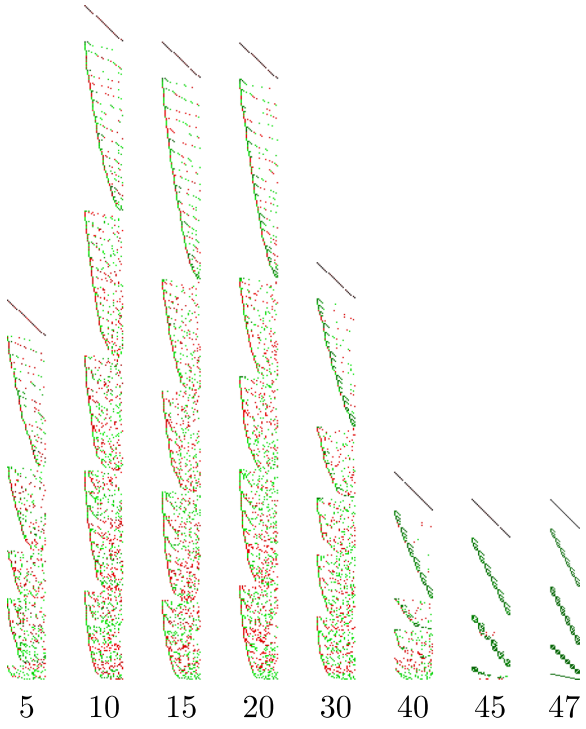

Fig. 7 The weights of a MOHN at the given number of iterations of the weight discovery algorithm when learning the 5 bit trap function over 30 variables. In each column, each row is a weight, each pixel relates to a neuron and shows that the weight is either not connected to that neuron (*white*) or is connected with a strength indicated by the colour. Red is negative and green is positive

solution's level of complexity and the rate at which it is changing, allowing decisions to be made about terminating the process or altering the structure by hand.

Figure 8 shows the weight counts for each order during training for the 5-bit trap function. By monitoring the number of weights used as training progresses, the user is able to gain an insight into the size of the remaining search space and the weight orders that remain to be explored. This helps the user make decisions about when to stop training and allows some insight into the likely quality of a model from their data. Contrast this to the process of training an MLP or a deep network, where the behaviour of the training and test errors are the only guide to the progress being made. The MOHN can provide additional metrics such as the number of weights tried since a new significant weight was

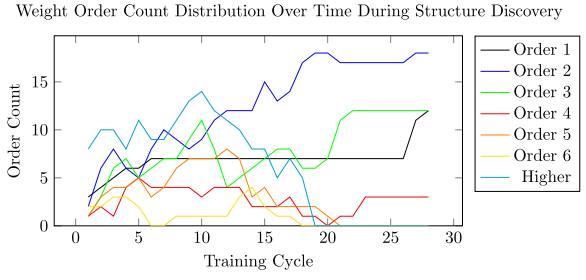

Fig. 8 The number of weights at orders from 1 to 6 and higher as the structure discovery algorithm learns a 5-bit trap function

added, which provides further insight into whether continued training is likely to yield improvements in error.

MOHN classifiers

It is also possible to train a MOHN as a classifier by assigning some neurons the role of inputs and others the role of outputs. The pattern of connectivity is restricted so that each weight connects $m > 0$ inputs and exactly one output. There are no weights between subsets of inputs alone or outputs alone. The structure discovery algorithm works in the same way as described above with the one modification that a new weight is added for each output unit in turn, but the order and the connected inputs are chosen in the same way.

A classifier MOHN was tested on the MNIST [34] hand written digit data set and compared to a standard MLP. A threshold was applied to the input data to create a binary training set but no other pre-processing was applied. The purpose of this test was to establish whether or not a MOHN could produce similar performance to an MLP on a given data set, not to achieve a new best accuracy for the MNIST data.

An MLP with 30 hidden units was trained on the thresholded MNIST training set data and acheived a correct classification rate of 95 % on training and 94 % on the test data. A MOHN was trained on the same data and acheived 94 % on training and 93 % on test. The results are comparable, but the MLP allows very little analysis of the structure of the resulting function. The MOHN, however, exposes the structure of the solution.

Figure 9, for example, shows the second order weights from each input to the output representing each class. The patterns describing each digit are clear to see. Green pixels indicate a positive connection, red are negative and brightness indicates strength of connection. Black pixels indicate a weight was not included in the model. Higher orders can also be visualised, for example Fig. 10 shows the order three connections between a single chosen input neuron, all other inputs and the output neuron representing class '1'. Positive connections between the chosen neuron and its neighbours in the '1' pattern can be seen, along with other probably spurious weights dotted about. The MOHN provides an insight into what the network has learned that is unavailable in the MLP.

Conclusion

Mixed Order Hyper Networks offer a number of attractions for model fitting. The complexity of the model is easily controlled and understood, offering more information than

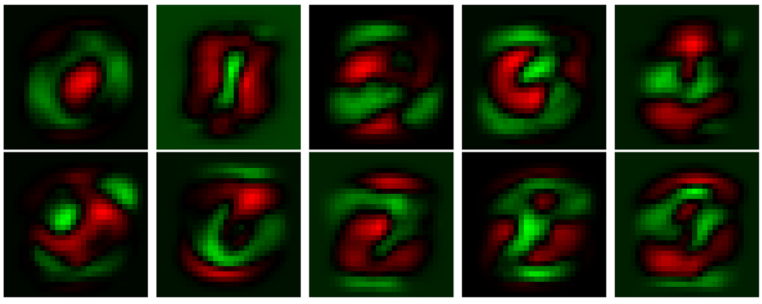

Fig. 9 The second order weights from a MOHN classifier trained on the MNIST hand written digit data set. Green squares indicate a positive weight, red indicate negative

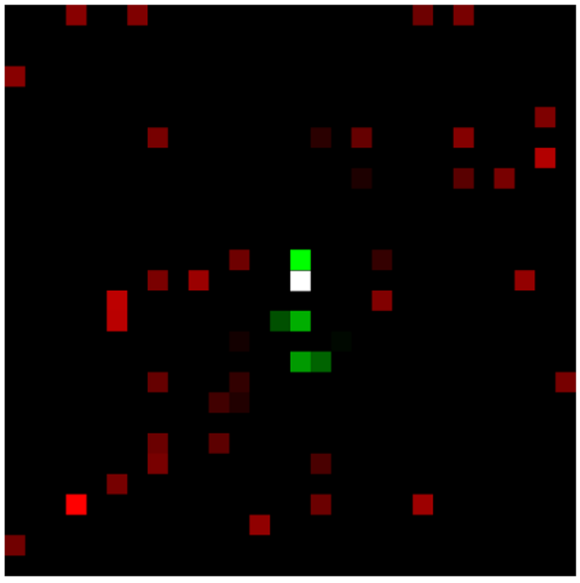

Fig. 10 The third order connections between a chosen neuron (marked in *white*) and, other inputs and the output for class '1'. Note the positive weights (*green*) to other neurons that are active for the chosen class, and negative (*red*) weights to other, further inputs. There are also some weights that seem spurious and could be pruned by hand

a simple parameter count. The structure of the model is human readable and allows prior knowledge to be included. The model can handle missing values, or impute them if required and feature selection is an integral part of the learning process. However, the challenge of discovering the correct structure for the network must be addressed explicitly as the network cannot discover high order features as part of the weight estimation process, as an MLP can. This paper proposes a method for discovering the weight structure of a MOHN in a way that allows different levels of prior knowledge to be included. In addition, as solutions are human readable, it is possible for a number of human driven iterations of the algorithm to allow the pattern discovery ability of the human eye to drive the search. More work is required in this area, in particular on methods for visualising the network structure. Such an approach would work well with a suitable human computer interface and development of an intuitive GUI is required.

Authors' information
The author lectures at the University of Stirling in the department of Computing Science and Mathematics. He is programme director of an MSc. in Big Data and also runs a spin out company providing hardware, software and consultancy for data collection and analytics. His research interests concern neural networks that replace hidden units with high order connections. This paper is the latest in a line of publications on this topic.

Competing interests
The author declares that he has no competing interests.

Received: 15 October 2015 Accepted: 14 September 2016
Published online: 01 October 2016

References

1. Swingler K, Smith LS. Training and making calculations with mixed order hyper-networks. Neurocomputing. 2014;141:65–75. doi:10.1016/j.neucom.2013.11.041.
2. Swingler K, Smith LS. An analysis of the local optima storage capacity of hopfield network based fitness function models. Trans Comput Collective Intel XVII, LNCS 8790. 2014;8790:248–71.
3. Swingler K. Local optima suppression search in mixed order hyper networks. In: Proc. UKCI 2015. Setúbal: SciTePress; 2015.

4. Bartlett EB. Dynamic node architecture learning: An information theoretic approach. Neural Networks. 1994;7(1): 129–40.
5. LeCun Y, Denker JS, Solla SA, Howard RE, Jackel LD. Optimal brain damage. In: NIPs. San Francisco: Morgan Kaufmann; 1989.
6. Andersen TL, Martinez TR. Dmp3: A dynamic multilayer perceptron construction algorithm. Int J Neural Syst. 2001;11(02):145–65.
7. Frean M. The upstart algorithm: A method for constructing and training feedforward neural networks. Neural Comput. 1990;2(2):198–209.
8. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15:1929–58.
9. García-Pedrajas N, Ortiz-Boyer D, Hervás-Martínez C. An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization. Neural Networks. 2006;19(4):514–28.
10. Yao X, Liu Y. Towards designing artificial neural networks by evolution. Appl Math Comput. 1998;91(1):83–90.
11. Cooper GF, Herskovits E. A bayesian method for the induction of probabilistic networks from data. Mach Learn. 1992;9:309–47.
12. Bouckaert RR. Probalistic network construction using the minimum description length principle. In: ECSQARU. Berlin Heidelberg: Springer; 1993. p. 41–8.
13. Larrañaga P, Kuijpers CMH, Murga RH, Yurramendi Y. Learning bayesian network structures by searching for the best ordering with genetic algorithms. IEEE Trans Syst Man Cybernet Part A. 1996;26(4):487–93.
14. Wong ML, Lee SY, Leung KS. A hybrid data mining approach to discover bayesian networks using evolutionary programming. In: GECCO. San Francisco: Morgan Kaufmann; 2002. p. 214–22.
15. Wong ML, Lam W, Leung KS. Using evolutionary programming and minimum description length principle for data mining of bayesian networks. IEEE Trans Pattern Anal Mach Intell. 1999;21(2):174–8.
16. De Campos CP, Ji Q. Efficient structure learning of bayesian networks using constraints. J Mach Learn Res. 2011;12: 663–89.
17. Ravikumar P, Wainwright MJ, Lafferty J. High-dimensional graphical model selection using l 1-regularized logistic regression. Neural Information Processing Systems. San Francisco: Morgan Kaufmann. 2006.
18. Lee SI, Ganapathi V, Koller D. Efficient structure learning of markov networks using l_1-regularization. In: Advances in Neural Information Processing Systems. San Francisco: Morgan Kaufmann; 2006. p. 817–24.
19. Brownlee AE, McCall JA, Shakya SK, Zhang Q. Structure learning and optimisation in a markov network based estimation of distribution algorithm. In: Exploitation of Linkage Learning in Evolutionary Algorithms. Berlin Heidelberg: Springer; 2010. p. 45–69.
20. Brownlee A, McCall J, Lee C. Structural coherence of problem and algorithm: An analysis for EDAs on all 2-bit and 3-bit problems. In: 2015 IEEE Congress on Evolutionary Computation (CEC). IEEE Press; 2015. p. 2066-73.
21. Kohavi R, John GH. Wrappers for feature subset selection. Artif Intell. 1997;97(1–2):273–324.
22. Hocking RR. A biometrics invited paper. the analysis and selection of variables in linear regression. Biometrics. 1976;32(1):1–49.
23. Bala J, Jong KD, Huang J, Vafaie H, Wechsler H. Using learning to facilitate the evolution of features for recognizing visual concepts. Evolutionary Computation. 1996;4:297–311.
24. Cantú-Paz E. Feature subset selection with hybrids of filters and evolutionary algorithms. In: Scalable Optimization Via Probabilistic Modeling. Berlin Heidelberg: Springer; 2006. p. 291–314.
25. Tibshirani R. Regression shrinkage and selection via the lasso. Royal Stat Soc Ser B (Methodological), J. 1996;58: 267–88.
26. Swingler K. A comparison of learning rules for mixed order hyper networks. In: Proc. IJCCI (NCTA). Setúbal: SciTePress; 2015.
27. Davidor Y. Epistasis variance: A viewpoint on GA-hardness. In: Foundations of Genetic Algorithms. San Francisco: Morgan Kaufmann; 1990. p. 23–35.
28. Hopfield JJ, Tank DW. Neural computation of decisions in optimization problems. Biol Cybernet. 1985;52:141–52.
29. Wilson GV, Pawley GS. On the stability of the travelling salesman problem algorithm of hopfield and tank. Biol Cybern. 1988;58(1):63–70. doi:10.1007/BF00363956.
30. Caparrós GJ, Ruiz MAA, Hernández FS. Hopfield neural networks for optimization: study of the different dynamics. Neurocomputing. 43(1–4):219–37.
31. Swingler K. Local optima suppression search in mixed order hyper networks. In: Computational Intelligence (UKCI), 2015 15th UK Workshop On; 2015.
32. Swingler K. A walsh analysis of multilayer perceptron function. In: Proc. IJCCI (NCTA). Setúbal: SciTePress; 2014.
33. Swingler K. Computational Intelligence In: Merelo JJ, Rosa A, Cadenas JM, Dourado A, Madani K, Filipe J, editors. Studies in Computational Intelligence. Springer; 2016. p. 303–23. doi:10.1007/978-3-319-26393-9_18. http://dx.doi.org/10.1007/978-3-319-26393-9_18.
34. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE. 1998;86(11):2278–324.