ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

Mixed Order Associative Networks for Function Approximation, Optimisation and Sampling

Kevin Swingler, Leslie S. Smith

University of Stirling - Computing Science and Mathematics
Stirling, FK9 4LA. Scotland

Abstract. A mixed order associative neural network with n neurons and a modified Hebbian learning rule can learn any function $f : \{-1, 1\}^n \rightarrow \mathbb{R}$ and reproduce its output as the network's energy function. The network weights are equal to Walsh coefficients, the fixed point attractors are local maxima in the function, and partial sums across the weights of the network calculate averages for hyperplanes through the function. If the network is trained on data sampled from a distribution, then marginal and conditional probability calculations may be made and samples from the distribution generated from the network. These qualities make the network ideal for optimisation fitness function modelling and make the relationships amongst variables explicit in a way that architectures such as the MLP do not.

1 Introduction

Function approximation lies at the heart of much of computational intelligence research. Many neural networks can be viewed as function approximators. It can be desirable to build a neural function approximator (NFA) that reproduces the output of any function and allows a transparent analysis of the weights. Where the function is a probability mass function (PMF), it is desirable to have a NFA that allows the calculation of marginal, joint, disjoint, and conditional probabilities as well as providing a method for sampling from the underlying joint distribution. Where the NFA is to be used as an aid to search and optimisation, it is useful if it is capable of producing output values that are outside the range of its experience and of modelling local optima as fields of attraction. Let $\mathbf{B} = \{-1, 1\}^n$ and $f(\mathbf{B}) \rightarrow \mathbb{R}$ be any function over $\mathbf{B}$. This paper demonstrates how a mixed order associative network (MOAN) displays all of these qualities for real valued functions of binary input vectors: $f(\mathbf{B})$.

Hopfield networks [1] are able to store patterns as point attractors in n dimensional binary space. Patterns are learned using a Hebbian weight update rule and recovered by settling the network to a stable state. Rather than learning known patterns like a normal Hopfield network, Swingler, [2] demonstrated a Hopfield network which learns samples $\mathbf{c} \in \mathbf{B}$ at a variable learning rate determined by $f(\mathbf{c})$. This yields a simple modification to the Hebbian rule: $w_{ij} = w_{ij} + f(\mathbf{c})u_i u_j$ where w_{ij} is the weight between neurons i and j, u_i is the output of neuron i and we set $u_i = c_i \; \forall i$.

A MOAN is a Hopfield like network with weights between neuron subsets of multiple orders. We use variable rate Hebbian learning to train MOANs on samples from $(\mathbf{B}, f(\mathbf{B}))$ and analyse their use. Section 2 describes variable learning rate MOANs, section 3 analyses their use as function approximators and section 4 describes their use in making probability calculations from a PMF.

ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

2 Mixed Order Associative Networks

In a standard Hopfield network, connections are between pairs of neurons. Mixed order associative networks (MOANs) have weighted hyper-synapses (we will just call them weights) between subsets of zero or more neurons. A fully connected n-neuron MOAN has weights that connect all subsets of all sizes from 0 to n. The weights have no direction so they have a natural symmetry. First order weights are not self-connections, but are treated as if they were linked to bias units - they have a constant positive activation that is not affected by network updates. The single size zero weight calculates the average of $-f(\mathbf{c})$. Sparse MOANs with far fewer connections may also be built. We now formally define a fully connected MOAN. Let n be the number of neurons in the network. The vector of neuron outputs is defined as $\mathbf{u} = u_1, \ldots, u_n$, $\mathbf{u} \in \mathbf{B}$.

Neurons are organised into unique subsets of size $0 \ldots n$. There are $\binom{n}{k}$ subsets of each size k. Each subset, Q_i has a single associated weight, denoted by W_i. The set of subsets and the set of weights both have $i = 0 \ldots 2^n - 1$. We index both by taking the integer value of a binary pattern that indicates the presence of a neuron in a subset as 1 and its absence as 0. The least significant bit refers to unit 1 and the most significant bit refers to unit n. For example, 1010 represents the subset where neurons 2 and 4 are connected, which makes it W_{10}. Values are set in the MOAN using:

$$u_i = c_i \quad \forall i \tag{1}$$

and patterns can be learned in the same way as a Hopfield network using the weight update rule $W_i = W_i + \prod_{u \in Q_i} u$. A MOAN can also learn a function $f(\mathbf{c})$ using a modified Hebbian rule. We define the update to the weight connecting the neuron subset Q_i caused by pattern $\mathbf{c}$ and function output $f(\mathbf{c})$ as

$$W_i = W_i + \prod_{u \in Q_i} u f(\mathbf{c}) \tag{2}$$

In the special case of W_0, we use $W_0 = W_0 - f(\mathbf{c})$. When training is complete, we divide every weight by the number of training patterns learned, m:

$$W_i = \frac{W_i}{m} \quad i = 0 \ldots 2^n - 1 \tag{3}$$

To re-calculate the activation of a neuron, we pick every subset to which the neuron belongs, that is, every Q_k where the binary representation of k has a 1 at the bit position relating to the chosen neuron. For each Q_k in that set, we calculate the product of the activations of the neurons, excluding the target neuron. This value is then multiplied by the subset weight W_k and these products are all summed. To calculate the activation, a_i of unit i, let L be the set of indices of subsets that contain u_i, and use:

$$a_i = \sum_{k \in L} \left(W_k \prod_{j \in Q_k \setminus i} u_j \right) \tag{4}$$

where k enumerates each subset index in L, W_k is the weight associated with neuron subset Q_k and $j \in Q_k \setminus i$ indicates the index of every member of Q_k, except neuron i itself. A neuron's output is then calculated using the threshold function:

ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

$$u_i = \begin{cases} 1, & \text{if } a_i > 0 \\ -1, & \text{otherwise} \end{cases} \tag{5}$$

A trained MOAN settles to an attractor point by repeated application of the activation rules 4 and 5, choosing neurons in random order. Any given state of the MOAN has an associated consistency measure, C, which is the negation of the network energy level:

$$C = \sum_{i=0}^{2^n-1} W_i \prod_{u \in Q_i} u \tag{6}$$

3 Analysis

In this section we take a MOAN trained on every pattern $\mathbf{c} \in \mathbf{B}$ and its function output, $f(\mathbf{c})$. This exhaustive training means that the network is not an approximation to $f(\mathbf{c})$, but a replication of it. A network trained on a less than exhaustive sample would produce an approximate model. We do so to demonstrate the exact properties of the network and leave for future work an analysis of approximate models.

3.1 Analysis of the Network Weights

One way to represent functions such as $f(\mathbf{B})$ that has been studied extensively makes use of Walsh functions [3], [4], which form a basis for functions of binary vectors. Any function $f : \{-1, 1\}^n \rightarrow \mathbb{R}$ can be represented as a weighted linear sum of Walsh functions.

A Walsh function carries out a bitwise comparison between two binary strings $\mathbf{c}$ and $\mathbf{j}$, 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). The Walsh literature generally flips the output so that false becomes 1 and true becomes -1, but we will not take that step and without loss of functionality, we define a Walsh function in logical notation as $\psi_j(\mathbf{c}) = \oplus_{i=1}^{l}(c_i \wedge j_i)$ where $\oplus$ is a modified XOR operator, which returns 1 if the argument list contains an even number of 1s and -1 otherwise. The Walsh transform of an n-bit function, $f(\mathbf{c})$, produces 2^n Walsh coefficients, ω_c, indexed by the 2^n combinations across $f(\mathbf{c})$. Each Walsh coefficient, ω_c is calculated by

$$\omega_c = \frac{1}{2^n} \sum_{j=0}^{2^n-1} f(j)\psi_j(\mathbf{c}) \tag{7}$$

and any function $f(\mathbf{c})$ can now be restated as a Walsh sum

$$f(\mathbf{c}) = \sum_{j=0}^{2^n} \omega_j \psi_j(\mathbf{c}) \tag{8}$$

Note that equations 7 and 8 both hold true whether we calculate the Walsh coefficients with the recoding favoured by the literature or leave the result of XOR as true=1

ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

and false=-1. Using the latter coding, if we index the MOAN weights by the neuron subset index as described above, we find that the weight values are equal to the Walsh coefficients of the same index. That is,

$$W_i = \omega_i \quad \forall i \tag{9}$$

Any weights that have a value of zero may be removed from the network as they have no impact on the energy function or the attractors. This often results in a sparse network. Clearly, the highest order interaction in a function is defined by the highest order non-zero weight in the network. Any particular non-zero weight tells us that there is an interaction between the bit values that are set to 1 in the binary representation of the weight's index.

3.2 Calculating Function Output

Setting the outputs of the MOAN to the values of a pattern, $\mathbf{c}$ and using the consistency calculation of equation 6 we can use the network to calculate the value of $f(\mathbf{c})$ directly, that is:

$$C(\mathbf{c}) = f(\mathbf{c}) \quad \forall \mathbf{c} \tag{10}$$

We can also use the weights of the network to calculate schemata average. Let a schema be $\mathbf{h} = h_i$, $i = 1 \ldots n$, $h_i \in \{-1, 1, *\}$, where the $*$ is a wildcard, define a hyperplane. We can take a partial sum of the network weights as determined by the schema to calculate its average. To determine which weights are included in a partial sum and whether they are added or subtracted, we use two functions that operate on $\mathbf{h}$. They are:

$$\alpha(h_i) = \begin{cases} 0 & \text{if } h_i = * \\ 1 & \text{if } h_i \in \{-1, 1\} \end{cases} \tag{11}$$

$$\beta(h_i) = \begin{cases} -1 & \text{if } h_i \in \{-1, *\} \\ 1 & \text{if } h_i = 1 \end{cases} \tag{12}$$

We write $\alpha(\mathbf{h})$ and $\beta(\mathbf{h})$ to apply a function to every bit in $\mathbf{h}$. We then use $\alpha(\mathbf{h})$ to determine which weights to include and $\psi(\beta(\mathbf{h}))$ to determine addition or subtraction. We use $\alpha(\mathbf{h})$ to generate the set of strings $\mathbf{c} \in \Upsilon$ where $c_i = 0$ if $\alpha(h_i) = 0$ and $c_i \in \{0, 1\}$ if $\alpha(h_i) = 1$. These strings are then converted to their integer equivalents to determine the indices of the weights to include. Let υ be this set of indices and $s = |\upsilon|$ be its size, then we take the s bits from $\psi(\beta(\mathbf{h}))$ indexed by each member of υ to determine addition or subtraction in the partial sum. The sum is defined as

$$f(\mathbf{h}) = \sum_{i \in \upsilon} W_i \psi_i(\beta(\mathbf{h})) \tag{13}$$

Note that $\psi_i(\beta(\mathbf{h})) \in \{-1, 1\}$ so each term in the sum is either W_i or $-W_i$. Equation 13 allows us to extract the schema average directly from the network weights without having to calculate the activations of any neurons.

ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

3.3 Local Maxima and Stored Memories

Input patterns that correspond to local maxima in $f(\mathbf{B})$ are point attractors in the MOAN and can be found by a local search by applying equations 4 and 5 to update neurons in random order until no changes to neuron output are produced. The resulting pattern across the neuron outputs represents a vector, $\mathbf{c}$, which is a local maximum in the output of $f(\mathbf{B})$ and a stored memory in the associative network. Each local maximum represents a stored pattern in the associative network.

4 Probability Mass Functions

The MOAN may be used as a probability mass function (PMF) for calculating marginal, joint, disjoint and conditional probabilities. To produce a network with the correct weights for calculating probabilities, we let $P(\mathbf{B})$ be the PMF of patterns in $\mathbf{B}$ and $p(\mathbf{c})$ be the probability of a particular $\mathbf{c}$. The MOAN is trained on samples from $\mathbf{B}$ with a constant learning rate of 1. When a sufficient number of samples has been learned, we adjust the weights so that:

$$W_i = \frac{W_i}{sp} \quad \forall i \tag{14}$$

where s is the number of samples learned and p is the number of distinct patterns in that sample set. To calculate a full joint probability, $p(\mathbf{c})$ from the trained model, we find that $p(\mathbf{c}) = C(\mathbf{c})$ where $C(\mathbf{c})$ is the consistency function defined in 6. To calculate a joint probability where only a subset of bits are set, we define $\mathbf{h}$, υ, s and β as we did in section 3.2 and calculate $p(\mathbf{h})$:

$$p(\mathbf{h}) = \frac{1}{2^d} \sum_{i=0}^{s-1} \upsilon_i W_{\upsilon_i}(\beta(\mathbf{h})) \tag{15}$$

We can treat a schema as a set membership definition where a value from {1,-1} indicates membership and a * indicates non-membership. A union $\mathbf{g} \cup \mathbf{h}$ where $\mathbf{g} \cap \mathbf{h} = \emptyset$ is made by keeping a * in bits where both $\mathbf{g}$ and $\mathbf{h}$ have a * and using the non-* value from $\mathbf{g}$ or $\mathbf{h}$ otherwise. For example 1** ∪ **1 = 1*1. From this definition, we can see that a joint probability is calculated as

$$p(\mathbf{g} \wedge \mathbf{h}) = p(\mathbf{h} \cup \mathbf{g}) \tag{16}$$

for example, $p(1 * * \wedge * * 1) = p(1 * * \cup * * 1) = p(1 * 1)$. The disjoint probability across two schemata such as $p(1 * * \vee * * 1)$ is calculated as

$$p(\mathbf{g} \vee \mathbf{h}) = p(\mathbf{g}) + p(\mathbf{h}) - p(\mathbf{h} \cup \mathbf{g}) \tag{17}$$

It should be clear how the two-clause examples above extend to m clauses such as $p(1 * * \wedge * * 1 \wedge * 1 *)$. Conditional probabilities such as $p(1 * 1 | * * 1)$ can be calculated as

$$p(\mathbf{g}|\mathbf{h}) = \frac{p(\mathbf{h} \cup \mathbf{g})}{p(\mathbf{h})} \tag{18}$$

ESANN 2013 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, i6doc.com publ., ISBN 978-2-87419-081-0.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100131010.

4.1 Sampling

Sampling from the learned distribution in the network is quite straightforward. We build a sample vector one bit at a time, picking bits at random from a reducing list of unset bits. The sample vector, $\mathbf{c}$ starts off with all bits set to *, i.e. $\mathbf{c} = *^n$. We then pick a random bit, c_i, set it to 1 in $\mathbf{c}$ and calculate $p(\mathbf{c})$ using equation 15. c_i is left set to 1 with probability $p(\mathbf{c})$ and flipped to -1 otherwise. Subsequent bits are set based the marginal $p(\mathbf{c} \wedge c_i = 1|\mathbf{c})$ where $\mathbf{c} \wedge c_i = 1$ indicates a vector made by setting bit i of $\mathbf{c}$ to 1. The marginal probability calculation is repeated for each bit, chosen at random from the list of unset bits, until all bits are set.

5 Experimental Results

We have carried out a large number of experiments on a range of functions with known mixed order interactions to verify these results. In all cases, we have found that an exhaustively trained MOAN's weights are equal to the Walsh coefficients of the function used and that equation 6 reproduces the function output. We have verified the probability calculations described in section 4 and found that samples from the network using the method described above exhibit the same distribution as the original PMF. Analysis of the network weights confirmed what we knew about the interaction order of each function learned.

6 Conclusions

Mixed order associative networks can model any $f(\mathbf{B})$. The resulting model makes linkage analysis easy, which is useful if $f(\mathbf{B})$ is an objective function to be used by an optimsation method such as a genetic algorithm. Also of use to optimisation algorithms is a method for sampling from the underlying distribution, which is presented here. We have not addressed the issue of the exponential growth in network size with the number of neurons, which is a severe limitation of the work in its current form. We note that the resulting networks are often sparse but that discovering the correct subset of weights for any function is an NP problem. This work shows the results of an exhaustive sampling of noise free data from the function to be learned. Further work will address the task of learning sparse networks from noisy samples.

References

[1] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, 79(8):2554–2558, April 1982.

[2] Kevin Swingler. On the capacity of Hopfield neural networks as EDAs for solving combinatorial optimisation problems. In *Proc. IJCCI*, pages 152–157. SciTePress, 2012.

[3] J.L. Walsh. A closed set of normal orthogonal functions. *Amer. J. Math*, 45:5–24, 1923.

[4] K.G. Beauchamp. *Applications of Walsh and Related Functions*. Academic Press, London, 1984.