Neurocomputing 141 (2014) 65–75

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Training and making calculations with mixed order hyper-networks

Kevin Swingler *, Leslie S. Smith

Division of Computing Science, University of Stirling, Stirling FK9 4LA, Scotland, United Kingdom

ARTICLE INFO

Article history:
Received 14 June 2013
Received in revised form
7 October 2013
Accepted 27 November 2013
Available online 8 April 2014

Keywords:
High order neural networks
Optimisation
Walsh functions

ABSTRACT

A neural network with mixed order weights, 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.

© 2014 Elsevier B.V. All rights reserved.

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, composite, 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 basins of attraction.

Many different neural network architectures exist and many possess one or more of the qualities listed above. Traditionally, artificial neural networks contain weights that connect neurons in pairs (i.e. second order weights). Perceptrons and multilayer perceptrons (MLPs) are commonly used as function approximators, but they do not allow for output values that are greater than those seen during training and local optima are not easy to analyse as attractors in energy space. Hopfield networks [8] and Boltzmann machines [1] encode attractors to local minima in energy space but are not generally used as function approximators. With the exception that some networks have a bias weight which could be considered to be of first order as it only connects to one live neuron, these networks contain only second order weights. There has been some interest in higher order networks — those with weights that connect more than two neurons. For example, [10] investigated high order MLPs, [4] investigated high order Hopfield like networks for optimisation and [13] presented a higher order associative memory.

This paper presents a neural network with high order weights, the Mixed Order Hyper-Network (MOHN) with an analysis of its capabilities. Additionally, a new learning rule is presented which allows the MOHN to act as either a content addressable memory, much like a high order Hopfield network, or a function approximator for functions $f:\{-1,1\}^n \rightarrow \mathbb{R}$. Local maxima in the function output are attractor states in the network and can produce output values that are higher than any seen during training. An analysis of the weights of the network is presented, which inspires a new method for calculating average values across hyper-planes in the input space. The network can also be trained to represent probability mass functions (PMFs), where the hyper-plane method can be used to calculate marginal, conditional and composite probabilities. These probability calculations allow a method of sampling from the underlying joint distribution.

This paper is organised as follows. Section 2 introduces the notation to be used and the concept of a schema as a definition of partially defined vector. Section 3 describes the structure of a MOHN, introduces the learning rule and describes the calculation of function outputs and various probabilities. Section 4 describes an algorithm for sampling from a learned distribution and Section 5 describes the process of climbing to attractor states (or sampling from local optima). An analysis of the weights of the resulting network is given in Section 6. The body of the paper concerns

* Corresponding author.
E-mail addresses: kms@cs.stir.ac.uk (K. Swingler), lss@cs.stir.ac.uk (L.S. Smith).

http://dx.doi.org/10.1016/j.neucom.2013.11.041
0925-2312/© 2014 Elsevier B.V. All rights reserved.

networks that are fully connected or trained on a full sample of the function to be learned. Section 6.4 introduces partial networks and is followed by some conclusions.

2. Functions, vectors, variables and schemata

The MOHN under consideration here is designed to learn functions that map a binary vector to a real valued output. That is,

$$f : \{-1, 1\}^n \rightarrow \mathbb{R} \tag{1}$$

Input vectors are written as $\mathbf{x}$ and output values as y.

2.1. Schemata

It is also necessary to define sub-vectors which indicate that some of the variables in $\mathbf{x}$ have been set and some have not (or have unknown values, or may take any value). These sub-vectors are called schemata and are defined over $\{-1, 1, *\}^n$ where $*$ denotes an unknown value or wildcard. Schemata define a set of possible instantiations of all the variables in $\mathbf{x}$. For example, $(1, *) = \{(1, -1), (1, 1)\}$.

2.1.1. Intersections of schemata

The intersection (logical AND, denoted $\cap$) of a number of schemata produces a new schema that is the intersection of the possible instantiations of their respective sets. For example, $1* \cap *1 = \{(1, -1), (1, 1)\} \cap \{(-1, 1), (1, 1)\} = (1, 1)$. If $\mathbf{H}$ is a set of non-contradictory schemata, then the intersection of the members of $\mathbf{H}$ is a single schema built by setting known values from any schema in $\mathbf{H}$ and leaving the rest unknown. For example, $(1, *, *, *) \cap (1, *, *, -1) \cap (*, *, 1, *) = (1, *, 1, -1)$. The function table for $\cap$ across two variables is given in Table 1. Note that $\varnothing$ indicates the empty set.

The intersection of a set of schemata is defined formally as follows. Let $\mathbf{H}$ be a set where $\mathbf{h}_j, j = 1 \ldots m$, are individual schemata and $h_{j,i}, i = 1 \ldots n$, represents the ith variable in $\mathbf{h}_j$. The intersection of the ith variable across all $\mathbf{H}$ is $g_i = \bigcap_{j=1}^{m} h_{j,i}$. This can be used to create a single schema, $\mathbf{g}$, which represents the intersection of all the schemata in $\mathbf{H}$. Each g_i is calculated separately using the following equation:

$$\mathbf{g}_i = \begin{cases} \varnothing & \text{if } \exists j \exists k \; h_{j,i} \neq h_{k,i} \wedge h_{j,i} \neq * \wedge h_{k,i} \neq * \\ 1 & \text{if } \forall j \; h_{j,i} \in \{1, *\} \wedge \exists j \; h_{j,i} = 1 \\ -1 & \text{if } \forall j \; h_{j,i} \in \{-1, *\} \wedge \exists j \; h_{j,i} = -1 \\ * & \text{if } \forall j \; h_{j,i} = * \end{cases} \tag{2}$$

or, more legibly,

$$\mathbf{g}_i = \begin{cases} \varnothing & \text{if any two values at position } i \text{ are known and different} \\ h_i \in \{-1, 1\} & \text{if there are known values and they are all the same} \\ * & \text{if none of the values are known} \end{cases} \tag{3}$$

Table 1
Function table for mixing schema variables with AND ($\cap$). Each variable in a schema may be processed independently to build the result.

g_i	h_i	$g_i \cap h_i$
−1	1	∅
1	−1	∅
−1	−1	−1
1	1	1
−1	*	−1
*	−1	−1
1	*	1
*	1	1
*	*	*

Two schemata are contradictory if they disagree on any known value. The intersection of contradictory schemata is an empty set, for example $(*, 1) \cap (*, -1) = \varnothing$ and if any $g_i = \varnothing$ then $\mathbf{g} \leftarrow \varnothing$.

2.1.2. Unions of schemata

The union (logical OR, denoted $\cup$) of a set of schemata is the union of all the possible instantiations of each schema in the set. For example, $(1, *) \cup (*, 1) = \{(1, 1), (1, -1)\} \cup \{(1, 1), (-1, 1)\} = \{(-1, 1), (1, -1), (1, 1)\}$. There is no way to represent a union of schemata in a single schema as there is for an intersection. Section 3.3.3 describes a method for computing the composite probability of a union of schemata without always needing to enumerate each member of the set.

3. MOHN structure, neurons, weights and naming

A MOHN has a fixed number of neurons, n. Each neuron can be in one of three states: $u_i \in \{-1, 1, 0\}$ where a value of 0 indicates a wild card or unknown value (the $*$ in the schemata described in Section 2.1). The state of the network is defined by a vector, $\mathbf{u}$ of n binary variables representing the neurons' outputs:

$$\mathbf{u} = u_{n-1} \ldots u_0, \quad u_i \in \{-1, 1, 0\} \tag{4}$$

The neurons are indexed in reverse order so that the right most neuron is the least significant bit in a binary word. This produces the weight and neuron subset indexing method described below. Take the neurons as a set, N so that the power set of N, $Q = \mathcal{P}(N)$ defines every possible subset of neurons. A single neuron subset, Q_k, $k \in \{0 \ldots 2^n - 1\}$ has its contents defined by the index, k, which is calculated by constructing a binary string containing a 1 at every position i where $u_i \in Q_k$ and a 0 elsewhere and converting that string to an integer using standard position coding. That is,

$$k = \sum_{i : u_i \in Q_k} 2^i \tag{5}$$

The structure of a MOHN is defined by a set, $\mathbf{W}$ of real valued weights. The index of a weight, $0 \leq k \leq 2^{n-1}$, is an integer value determined by the indices of the neurons to which the weight is connected in the same way as a neuron subset index is calculated. That is, weight W_k connects the neurons in subset Q_k. The weights of any given network are a subset of all 2^n possible weights for a network of size n. In this way the weights define a hyper-graph connecting the elements of $\mathbf{u}$:

$$\mathbf{W} \subseteq \{W_j : j = 0 \ldots 2^{n-1}\}, \quad W_j \in \mathbb{R} \tag{6}$$

As noted, there is a mapping between the subsets in $\mathbf{Q}$ and the weights in $\mathbf{W}$. There are times in what follows when it is necessary to refer not to the weights or the neuron subsets, but to the indices that define them. For this the function $X(\mathbf{S})$ is used, which produces a set containing the indices of the given set. For example, $X(\{W_0, W_2, W_5\}) = \{0, 2, 5\}$. This provides a mapping between neuron subsets and weights, allowing $\mathbf{Q_i} : i \in X(\mathbf{W})$ to mean the subset of neuron subsets that correspond to the weights in the set $\mathbf{W}$. A fully connected network has a single weight for every possible neuron subset, so $X(\mathbf{W}) = X(\mathbf{Q})$. A partially connected network has weights between a subset of the neuron subsets, that is $X(\mathbf{W}) \subset X(\mathbf{Q})$.

The weights each have an associated order, defined by the number of neurons they connect. There is a single zero-order weight, which connects no neurons, but has a weight all the same. There are n first order weights and n corresponding first order neuron subsets, which are, of course, the n neurons of the network. The first order weights are the equivalent of bias inputs in a standard neural network. In general, there are $\binom{n}{k}$ weights of order k in a fully connected network of size n.

Download English Version:

https://daneshyari.com/en/article/406611

Download Persian Version:

https://daneshyari.com/article/406611

Daneshyari.com