A theoretical framework for multiple neural network systems

Mike W. Shields, Matthew C. Casey*

Department of Computing, School of Electronics and Physical Sciences, University of Surrey, Guildford, Surrey, GU2 7XH, UK

Abstract

Multiple neural network systems have become popular techniques for tackling complex tasks, often giving improved performance compared to single network systems. For example, modular systems can provide improvements in generalisation through task decomposition, whereas multiple classifier and regressor systems typically improve generalisation through the ensemble combination of redundant networks. Whilst there has been significant focus on understanding the theoretical properties of some of these multi-net systems, particularly ensemble systems, there has been little theoretical work on understanding the properties of the generic combination of networks, important in developing more complex systems, perhaps even those a step closer to their biological counterparts. In this paper we provide a formal framework in which the generic combination of neural networks can be described, and in which the properties of the system can be rigorously analyzed. We achieve this by describing multi-net systems in terms of partially ordered sets and state transition systems. By way of example, we explore an abstract version of learning applied to a generic multi-net system that can combine an arbitrary number of networks in sequence and in parallel. By using the framework we show with a constructive proof that, under specific conditions, if it is possible to train the generic system, then training can be achieved by the abstract technique described.

Keywords: Multi-net systems; ensembles; multiple classifier systems; partially ordered sets; state transition systems

1. Introduction

Neural network research has reached the stage where we have at least a good understanding of the most popular architectures and algorithms (cf. [4]), and yet there remains a significant gap between the capabilities of such networks compared to the capability, say, of the human brain. Whilst improved learning algorithms allow us to achieve good levels of performance on tasks such as regression or classification, neural networks fall short of the equivalent complexity of biological systems. One way in which the performance and capability of artificial neural systems has been improved is through the combination of multiple networks [26,31]. These *multi-net systems* [43] are simply ways in which networks can be combined into a cohesive architecture, often with a specific learning algorithm (for example [6,18,27]), and whilst they still fall short of biological complexity, they have been used to demonstrate improved performance and capability in a number of areas, including regression [39], classification [31] and, perhaps more relevantly, computational modelling [23,26].

Whilst these combined systems have proven popular, our formal understanding of their properties and capabilities is not complete. For ensembles, our understanding depends upon the task; we have a relatively good understanding of regression, but not for classification [8]. Other architectures require a more specific configuration of the components and choice of learning algorithm [29]. In general this means that the choice of an optimum topology and parameterization is a matter for trial-and-error, often without knowing if convergence to a stable solution is possible, or whether a simpler solution is better. To overcome this, a formal understanding of these systems is therefore desirable to remove such uncertainties. Whilst statistical analysis is proving useful for specific architectures [7,19,22,44,47], there has been little work on expanding this to a generic framework in which all such multi-net systems may be described.

In this paper we provide an abstract, general theory of multi-net systems using a framework in which the generic combination of networks can be described, and in which the properties of the system can be rigorously analyzed. To achieve this we use partially ordered sets to describe the topology of multi-net systems, abstract sets to represent weights and other parameters, abstract functions to represent the activation function and combinations, and transition systems to model the dynamics of activation and learning. The use of transition systems builds upon their wide use in concurrency theory to provide interleaving operation semantics (cf. [30,37]). This novel application of set theory provides a framework in which we can prove properties of combined systems. In particular, we provide a constructive characterisation of a supervised learning algorithm that can be synthesised by an abstract scheme, the latter motivated by the ideas of gradient descent learning, which can be used to train a generic multi-net system, such

* Corresponding author. Tel./fax: +44 (0) 1483 689635.
E-mail addresses: m.casey@surrey.ac.uk (Matthew Casey), myramike@gmail.com (Mike Shields)

that the parameters of the system converge to meet a specific criterion function. Importantly, this is achieved without resorting to specific details of the components involved, and especially without relying upon computing gradients and the constraints this brings. It should be stressed, however, that the algorithm in question is completely abstract, involving no numerical data, and hence without immediate application.

In section 2 of this paper we provide a background to multi-net systems and existing work on formalism. In section 3 we provide the framework to formally define the topology, parameterization and learning algorithm of a multi-net system and in section 4 we consider how an abstract formulation of supervised learning using a backward pass can be used to train a generic multi-net system. We spend considerable effort in these two sections to ensure that the formal foundation of the framework is rigorous to allow it to be applied successfully to multi-net systems in general. In section 5 we conclude with a discussion on the limitations of the proposed framework and consider future work.

2. Multi-net systems

The idea of combining components together to form a cohesive system which is more capable than its individual parts is not new, even for neural networks. For example, Hebb's discussion on learning includes the concept of 'superordinate' systems built from integrating *cell assemblies* [25]. This idea of a combination of neural components has been used successfully for computational models of cognitive abilities (for example, [23,41]), and appears to be a natural extension of the neural modelling paradigm, building from neurons, to layers, to networks [26], with some architectures even linked to functional specialism in the brain [20].

Particular multi-net architectures, or in general combinations of machine learning techniques, have demonstrated tangible performance improvement on tasks such as regression [39] and classification [31]. Broad types of combination include ensemble, modular and hybrid systems [43], but more detailed taxonomy do not recognize such clear divisions, with the behaviour of parallel, sequential and hybrid types of architecture dependent upon the process of learning [38]. Here, the choice of learning algorithm can make such systems co-operative (typically ensembles), competitive (modular), static (pre-configured prior to combination) or dynamic (configured in-situ), for the same or similar configurations [42].

Exemplars of modular systems are the mixture-of-experts (ME) and the hierarchical mixture-of-experts (HME) architectures [27,28], in which a task is automatically decomposed into sub-tasks solved by individual expert networks using a competitive algorithm, essentially allocating input patterns to each expert. With a particular configuration and algorithm, convergence properties are known [29,36], yet such a specific result does not apply to the more general use of ME, which has proven of benefit for different cognitive simulations [14,15,26].

Ensemble systems have also proven popular with the development of algorithms such as AdaBoost [18] and negative correlation (NC) learning [34]. Whilst properties of ensembles for regression problems are mostly well understood [7,39] (especially with the recent linkage of the Ambiguity [32] and the bias-variance-covariance [45] decompositions to learning in NC, which results in being able to understand how to construct an accurate regressor, given the use of a quadratic error function [9]) there is only limited theory [19,22,44] for ensembles of classifiers (*multiple classifier systems*) with no complete understanding of how accurate ensemble classifiers can be constructed, but which is focused on the notion of *diversity* [33].

Whilst there is a good theoretical understanding of ME and HME, and a developing understanding of ensembles, there is no such theory for other types of multi-net system. Examples include sequential systems [13,35], adaptive parallel combinations of networks [10,46] or ad-hoc systems, popular in computational models of brain function and behaviour, such as aspects of human vision [11], numerical abilities [14,17] and emotion [3], to name but some. Yet despite a lack of understanding of the properties of the combined system, we know significantly more about the individual components, which it would be useful to apply to the combined system.

Attempts have been made to define a theoretical framework that can be used to describe a more general class of multi-net system. Parallel (cf. ensembles) and sequential combinations of networks were formalized by Bottou and Gallinari [5], in which they explored ways in which combinations of learning algorithms could be defined. Amari [2] defined a stochastic model of neural networks that was used to describe both single network and multi-net systems (the ME architecture), and which could be extended to other types of combination. In a similar way, the formal definition of the HME architecture and algorithm [28] could also be extended to more general types of multi-net system. Kittler, Hatef, Duin and Matas [31] also recognized the need for a theoretical framework for describing combinations of classifiers, but their work was restricted to a particular configuration, namely the parallel combination of classifiers (or networks), as used in an ensemble. Similarly, Giacinto and Roli [22] used a statistical framework to demonstrate how an optimal Bayes classifier can be constructed theoretically from a dynamic classifier selection (DCS) system [21]. Whilst each of these have abstracted some of the properties of the components, for example the types of component [31], none consider how properties of the whole system might be rigorously analyzed. If we are to understand such properties of a more general class of multi-net system, important perhaps for the construction of improved models of the brain and other more complex systems, then we must develop a generic formal understanding of these systems

and use knowledge of the components to infer properties of the whole – something that can only be achieved in a rigorous formal framework.

In this paper, we present an abstract, set-based model of multi-net systems. The starting point for the work reported here is Casey [12]. He used directed trees to define a generic architecture of multi-net systems, together with a framework for the description of the associated learning algorithm. However, like previous work, this lacked sufficient rigor to explore the properties of the systems. In this paper we build upon this work by considering combined systems generically by developing an abstract model of multi-nets. The model is abstract in that it represents multi-net topologies in terms of partially ordered sets, with weights considered non-numerically in combination with abstract functions. We also formalize the notion of a feedforward state of the system (the production of an output given a set of inputs), and the concept of learning as a strategy to change the parameterization of the system to one that satisfies a predicate function, modelling convergence to some defined state. We provide this formal description so that the abstract properties of the generic class of multi-net systems can be analyzed.

Whilst providing a formal framework is useful to consistently define multi-net architectures, it is only really useful when used to discover properties of the combined system that were not previously known. Consequently, in this paper we present results from the application of the formalism to understand how supervised learning can be achieved in a generic system. The system is generic in the sense that we do not specify topological constraints on the components, such as the type of network or neurons. In particular, we define an abstract form of supervised learning for this generic system, motivated by the backward training pass of algorithms such as backpropagation [40] and other gradient descent learners. We then prove that, if we have any type of feedforward multi-net system used for a supervised learning task, if we can precisely define the convergence predicate, then there exists a learning algorithm, based upon the abstract form described, that can be used to train the system. Whilst this is perhaps intuitive, the significance is in the generality of the result: we do not require that the system consists of neurons, perceptrons or indeed, any other type of network, rather, our constraint lies in being able to describe the system using the formalism only. The limitation is that, whilst this is a demonstration of the usefulness of abstracting combinations of networks, in order to be practical, the detail of the strategy and associated predicate must be provided, something that we do not deal with in this paper.

3. From multi-nets to state transition systems

In this section we present the basic mathematical concepts underlying the work presented here. The first of these is that of a partially ordered set, which we use instead of directed trees to describe the topology of a multi-net system (and visually via a Hasse diagram) [16]. The second is that of a transition system, which provides us with the means to model dynamical aspects of these systems (both feedforward and feedback).

3.1. Partial orders

Casey [12] pictures multi-nets as directed trees, but we shall find it convenient to treat these from the point of view of partially ordered sets. A partially ordered set consists of a set V and a relation on V, which we shall denote by $\leq$. If $u,v \in V$, then we shall interpret $u \leq v$ to mean that u is either equal to or lies below v, where the root of the tree lies above all other nodes. In general, the relation $\leq$ satisfies three conditions; if $u,v,w \in V$, then:

1) $u \leq u$ ($\leq$ is *reflexive*);
2) If $u \leq v$ and $v \leq u$, then $u = v$ ($\leq$ is *antisymmetric*);
3) If $u \leq v$ and $v \leq w$, then $u \leq w$ ($\leq$ is *transitive*).

We write $u < v$ if $u \leq v$ and $u \neq v$, and we write $u \not\leq v$ if $u \leq v$ is false. We add an additional constraint to this formulation to avoid two separate branches of the tree from being connected[1]. For $u,v,w \in V$:

If $u \leq v,w$, then either $v \leq w$ or $w \leq v$ (1)

Finite partial orders (that is to say, partial orders where V is a finite set) may be represented pictorially by a Hasse diagram, in which elements of the set are represented by nodes, and the existence of a sequence of arrows from, say, a to b indicates $a > b$. Fig. 1b) shows a Hasse diagram for an ensemble $V_{ens} = \{t,u,v,w\}$ where three networks are combined in parallel. In this diagram $u < t$, $v < t$, $w < t$, but, $v \not\leq w$, etc. In general, we say that $a_0 \in V$ is a root node if $a \leq a_0$ for all $a \in V$, so that t is a root node of V_{ens}. Observe that if $b_0 \in V$ is also a root node, then $a_0 \leq b_0$ and $b_0 \leq a_0$, but then $a_0 = b_0$ by antisymmetry, so root nodes, if they exist, are unique. We write $root(V)$ for the root of V.

Definition 1. A *framework* is a finite, non-empty partially ordered set with a root node, satisfying (1). We also define a relation $\lhd$ on V by:

$a \lhd b$ iff $a < b$, $a < c < b$, for no $c \in V$ (2)

$a \lhd b$ means that there is an arrow in the Hasse diagram from b to a.

Fig. 1d) shows a framework V_{hme}. Here, for example, we have $u \lhd t$ and $x \lhd v$, but not $y \lhd t$, since $y < v < t$. Define $^{\bullet}a = \{b \in V : b \lhd a\}$. We call elements of $^{\bullet}a$ the children of a. For example, in V_{hme}, $^{\bullet}t = \{u,v,w\}$ and

[1] This is not used in any of the proofs given in this paper, but serves to identify these partial orders with a tree-like Hasse diagram, such as that shown in Fig. 1c).

${}^{\bullet}z = \varnothing$. Note that z is a leaf of V_{hme} and in general, we define:

$$leaf(V) = \left\{a \in V : {}^{\bullet}a = \varnothing\right\} \quad (3)$$

So far we have provided a formal notation and informal depictions (via a Hasse diagram) of how we can describe a multi-net system. Indeed, this description is quite generic and does not yet constrain the characteristics of the nodes themselves. We next look at how the dynamics of a multi-net system can be described.

3.2. Transition systems

In our discussion of the dynamics of feedforward multi-net systems, we shall use the language of transition systems (cf. [1]).

A transition system T is composed of a non-empty set Q of *states*, a non-empty set A of *actions* and a relation $\rightarrow \subseteq Q \times A \times Q$, the *transition relation* of T. The elements of $\rightarrow$ are *ordered triples* (q,a,q'), where $q,q' \in Q$ and $a \in A$. We shall write $q \rightarrow^{a} q'$ to indicate that $(q,a,q') \in \rightarrow$. Intuitively, $q \rightarrow^{a} q'$ means that if the system is in state q, then the action a may take place, after which the system is in state q'. For finite (Q finite) transition systems, there is a graphical representation in which elements of Q are nodes and $q \rightarrow^{a} q'$ is indicated by an arrow from q to q' labelled a.

If we consider a feedforward multi-net system, such as an ensemble, then *states* describe static conditions of the multi-net, and *actions* transform one static situation into another. Such an action occurs when an individual node takes the values on its inputs, evaluates the output, which will also depend on its current local parameterization, and transmits it, either to a parent node or to the output of the entire multi-net. In this way the networks within the system take input, change state and pass on their output.

At the beginning of the feedforward sweep, the inputs to the nodes and the local parameterizations have specific values (for example, the weights). Changes to this state as a result of applying the input depend upon these parameters and the node functions. A state, therefore, is embodied by a function associating nodes with values and parameterizations. An action involves an activation of a node, altering the value at its output. These dynamics may be described by *execution sequences*.

If x is a sequence of actions and $q,q' \in Q$, then we define $q \rightarrow^{x} q'$, providing that:

1) If $x = \Lambda$ (the empty sequence), then $q \rightarrow^{x} q'$ iff $q = q'$;
2) If $x = a_1 \cdots a_n$, where $a_1,\cdots,a_n \in A$, then $q \rightarrow^{x} q'$ iff there exists $q_1,\cdots,q_n \in Q$ such that $q \rightarrow^{a_1} q_1 \rightarrow \cdots \rightarrow^{a_n} q_n = q'$.

We shall say that an execution $q \rightarrow^{x} q'$ is *maximal from* q if and only if $q' \rightarrow^{a} q''$ for no $a \in A$ and $q'' \in Q$, such that q' is the maximal state from q. We shall prove (Theorem 1) that from any state $\underline{\sigma}_0$ in a given multi-net with a given parameterization:

1) There exists a maximal execution $\underline{\sigma}_0 \rightarrow^{x} \underline{\sigma}$; (A)
2) If $\underline{\sigma}_0 \rightarrow^{x} \underline{\sigma}$ and $\underline{\sigma}_0 \rightarrow^{y} \underline{\sigma}'$ are maximal executions, then $\underline{\sigma} = \underline{\sigma}'$. (B)

The practical consequence of this is that there is a function $G_{\underline{\theta}} : I \rightarrow O$, where I is the set of all input values that can be applied to the leaves of the multi-net; O is the set of all values that can appear as an output at the root of the multi-net, such that if $\underline{\sigma}_0$ is a state in which $\underline{\theta}$ is the parameterization and $\underline{x} \in I$ gives the values on the leaves of the multi-net in state $\underline{\sigma}_0$, then $G_{\underline{\theta}}(\underline{x})$ is the value of the output at the root in state $\underline{\sigma}$ where $\underline{\sigma}_0 \rightarrow^{x} \underline{\sigma}$ is any maximal execution. If $\underline{\theta}$ is the parameterization of the multi-net after training, then $G_{\underline{\theta}}$ describes its functionality. The consequence of this is that we can describe an arbitrary complex neural model (or any other combination of functions) precisely in terms of its inputs, operations and outputs, parameters and functions and know that such a description guarantees the system can produce an output (A) deterministically (B). We shall refer to (A) and (B) again later.

3.3. Multi-net systems: feedforward state

We now use the idea of a framework and of a transition system to define a feedforward multi-net system.

One of the points of this paper is that many aspects of the combination of networks can be described and analyzed with no reference to numerical issues. For example, Theorem 2 (section 4.3) shows that any reasonable training scheme can be achieved by an abstract backward pass of parameter (weight) modification instructions. The proof would be much more complicated if we focussed on the numerical properties and such systems (for example, computing gradients). However, once the existence of the proof is established, it is necessary to make it more concrete to be of use.

Our abstract view of a multi-net system explores the topology of the network, represented by a framework (Definition 1) of nodes. We abstract the weights, biases, etc. by associating each node $v \in V$ with an abstract set Θ_v of parameters. Likewise, we do not refer to an

activation function, but that each node $v \in V$ is associated with a function ϕ_v that takes the values on the node's inputs and calculates an output, depending on the current value of its local parameters. We shall also assume that each node $v \in V$ is associated with a non-empty set O_v of output values.

To be specific, suppose that $v \in V$. If v is not a leaf node, then there are distinct nodes $v_1, \cdots, v_n$ such that $v_i \lhd v$, for each so that $\{v_1, \cdots, v_n\} = {}^\bullet v$. We can therefore consider inputs to v to be vectors with coordinates $v_1, \cdots, v_n$. Technically, an input vector to v is a function that maps the children of v to the union of the outputs of each child:

$$\underline{x} : {}^\bullet v \to \bigcup_{w \in {}^\bullet v} O_w \tag{4}$$

with the property that $\underline{x}(w) \in O_w$ for each $w \in {}^\bullet v$. Note that this is equivalent to, say, describing the connections a neuron has from a previous network layer; each connection provides part of the input to the neuron.

We shall write $\underline{x}_w$ for $\underline{x}(w)$. The set of all such vectors $\underline{x}$ constitutes the *input space* of v in V and will be denoted by I_v. For uniformity, in the case of nodes $v \in \mathit{leaf}(V)$, we shall define ${}^\bullet v = \{\bot_v\}$, so that the inputs to v will belong to a set $O_{\bot_v}$, which is the set of all valid inputs to leaf v from the environment. The input space of $v \in \mathit{leaf}(V)$ may now be defined as for non-root nodes.

We are now in a position to give a formal definition of a multi-net.

Definition 2. A *multi-net* is defined to be a triple $\mathrm{N} = (F, \Theta, \Phi)$, where:

1) $F = (V, \leq)$ is a multi-net framework (Definition 1);
2) Θ is an indexed family of non-empty sets Θ_v, $v \in V$, the parameterization sets of $\mathbf{N}$;
3) Φ is an indexed family of node functions ϕ_v, $v \in V$, where:

$$\phi_v : \Theta_v \times I_v \to O_v \tag{5}$$

We shall now describe the feedforward dynamics of the system. If we consider the parameterization of the multi-net to be fixed (learning will be dealt with later), then what changes during the feedforward sweep are the values on the outputs of the nodes. We assume for the moment that the external inputs to the multi-net remain unchanged. We may therefore define a *feedforward state* to be a function that maps the framework to the union of the outputs of each node feeding the system output:

$$\underline{\sigma} : V^+ \to \bigcup_{v \in V^+} O_v \tag{6}$$

where $V^+ = V \cup \{\bot_v : v \in \mathit{leaf}(V)\}$ and $\underline{\sigma}(v) \in O_v$, for each $v \in V^+$. $\underline{\sigma}(v)$ gives the value residing on the output of the node v input to its parent, whilst $\underline{\sigma}(\bot_v)$ gives the value on the input to the leaf v, the notional parent of $\bot_v$. V^+ is therefore the set of all nodes that provide output used by the system (including the output of the input layer).

Let Σ_{N} denote the set of all feedforward states of the multi-net N. What will change a state is the activation of some node, as it computes its output from the current input. We may therefore consider V as the set of actions. The transition relation for these states will depend on the parameterization of the multi-net. We may represent a parameterization by a function that maps the framework to the union of all the node parameterizations:

$$\underline{\theta} : V \to \bigcup_{v \in V} \Theta_v \tag{7}$$

satisfying $\underline{\theta}(v) \in \Theta_v$, for all $v \in V$. We denote the set of all parameterisations of $\mathbf{N}$ by $\underline{\Theta}_{\mathbf{N}}$.

We are now interested in describing relations $\to_{\underline{\theta}} \subseteq \Sigma_{\mathrm{N}} \times V \times \Sigma_{\mathrm{N}}$, which for a given parameterization $\underline{\theta}$ describes the possible set of state transitions for the system.

In any state $\underline{\sigma} \in \Sigma_{\mathrm{N}}$, the inputs to $v \in V$, that is the values lying on the outputs to the elements $w \in {}^\bullet v$, will have values $\underline{\sigma}(w)$ and so we can define a vector $\underline{z}_{\underline{\sigma},v} \in I_v$ by

$$\underline{z}_{\underline{\sigma},v}(w) = \underline{\sigma}(w), \quad w \in {}^\bullet v \tag{8}$$

$\underline{z}_{\underline{\sigma},v}$ is the vector of inputs to node v in state $\underline{\sigma}$.

What we are interested in is the output of the system for a given state, defined as the feedforward state of each component node – we give inputs to the system and propagate these through to the output in a single pass. To achieve this, we define the notion of *stability*. Let us say that a node $v \in V$ is *stable* in state $\underline{\sigma} \in \Sigma_{\mathrm{N}}$ with respect to $\underline{\theta} \in \underline{\Theta}_{\mathrm{N}}$ if $\phi_v(\underline{\theta}(v), \underline{z}_{\underline{\sigma},v}) = \underline{\sigma}(v)$, otherwise, it is unstable. A node is stable in a given state with respect to a given parameterization if its output equals the value computed by the node from its inputs in that state and its local parameters.

We may now define $\underline{\sigma} \to_{\underline{\theta}}^{v} \underline{\sigma}'$ iff v is not stable at $\underline{\sigma}$ with respect to $\underline{\theta}$

$$\underline{\sigma}' = \underline{\sigma}[v / \phi_v(\underline{\theta}_v, \underline{z}_{\underline{\sigma},v})] \tag{9}$$

which is the operation of the action $\to_{\underline{\theta}}^{v}$ that moves the

node v from state $\underline{\sigma}$ to state $\underline{\sigma'}$, where

$$\underline{\sigma}[v/a](w) = \begin{cases} \underline{\sigma}(w) \text{ if } w \neq v \\ a \text{ otherwise} \end{cases} \quad (10)$$

So $\underline{\sigma} \rightarrow_{\underline{\theta}}^{v} \underline{\sigma'}$ when the output to v at $\underline{\sigma}$ with respect to $\underline{\theta}$ does not match the inputs. The effect of the transition is to update this output according to the node function ϕ_v.

We say that $\underline{\sigma}$ is *stable* w.r.t. to $\underline{\theta}$ if every $v \in V$ is stable at $\underline{\sigma}$ with respect to $\underline{\theta}$. By definition, if $x \in V^*$ and $\underline{\sigma'}$ is stable, then for every $\underline{\sigma} \in \Sigma_{\mathbf{N}}$, $\underline{\sigma} \rightarrow_{\underline{\theta}}^{x} \underline{\sigma'}$ is a maximal execution (cf. section 3.2) and $\underline{\sigma'}(root(V))$ is the final stable output. Write $Stbl_{\underline{\theta}}(\Sigma_{\mathrm{N}})$ for the set of all states $\underline{\sigma} \in \Sigma_{\mathrm{N}}$ stable with respect to $\underline{\theta}$.

Definition 3. If N is a multi-net then define the input space of N to be the set of all functions

$$\underline{x} : leaf(V) \rightarrow \bigcup_{v \in leaf(V)} O_{\perp_v} \quad (11)$$

such that $\underline{x}(v) \in O_{\perp_v}$ for each $v \in leaf(V)$. Define the output space of N to be $O_{\mathrm{N}} = O_{root(V)}$. If $\underline{\sigma} \in Stbl_{\underline{\theta}}(\Sigma_{\mathrm{N}})$, then define $\underline{x}_{\underline{\sigma}} \in I_{\mathrm{N}}$ by $\underline{x}_{\underline{\sigma}}(\perp_v) = \underline{\sigma}(\perp_v)$, for all $v \in leaf(V)$, and $y_{\underline{\sigma}} \in O_{\mathrm{N}}$ by $y_{\underline{\sigma}} = \underline{\sigma}(rootV)$.

We have explained in section 3.2, in general terms, how we may use transition systems to describe the dynamics of systems as sequences of state-transforming actions. We have now defined 'state' and 'transition' for multi-net systems. The following result establishes (A) – that we can describe the output of the system given an input and parameterization.

Proposition 1. For every $\underline{\sigma} \in \Sigma_{\mathrm{N}}$ and $\underline{\theta} \in \underline{\Theta}_{\mathrm{N}}$, there exists a maximal execution $\underline{\sigma} \rightarrow_{\underline{\theta}}^{x} \underline{\sigma'}$.

Proof. Define

$$L_{\underline{\sigma},\underline{\theta}} = \{w \in V : v \leq w \Rightarrow v \text{ is stable w.r.t. } \underline{\theta}\} \quad (12)$$

which is the set of nodes w, such that all are stable. We may readily verify:

1) $\underline{\sigma} \in Stbl_{\underline{\theta}}(\Sigma_{\mathrm{N}})$ iff $L_{\underline{\sigma},\underline{\theta}} = V$;
2) If w is minimal in the set $V \setminus L_{\underline{\sigma},\underline{\theta}}$, then there exists $\underline{\sigma'}$ such that $\underline{\sigma} \rightarrow_{\underline{\theta}}^{w} \underline{\sigma'}$ and $L_{\underline{\sigma},\underline{\theta}} \cup \{w\} \subseteq L_{\underline{\sigma'},\underline{\theta}}$.

Here, if $X \subseteq V$, then v is minimal in X if $v \in X$ and $w \notin X$, for $w < v$. If A and B are sets then:

$$A \setminus B = \{a \in A : a \notin B\} \quad (13)$$

So to say that w is minimal in $V \setminus L_{\underline{\sigma},\underline{\theta}}$ is to say that w is not stable w.r.t. θ, but all its children are.

We may now argue by induction on $|V \setminus L_{\underline{\sigma},\underline{\theta}}|$ (the cardinality of $V \setminus L_{\underline{\sigma},\underline{\theta}}$). If $|V \setminus L_{\underline{\sigma},\underline{\theta}}| = 0$, then $L_{\underline{\sigma},\underline{\theta}} = V$, so $\underline{\sigma}$ is stable, from 1), and $\underline{\sigma} \rightarrow_{\underline{\theta}}^{\Lambda} \underline{\sigma'}$ is a maximal execution. If $|V \setminus L_{\underline{\sigma},\underline{\theta}}| > 0$, then let w be minimal in the set $V \setminus L_{\underline{\sigma},\underline{\theta}}$. w cannot be stable, since if it were, then as $v \in L_{\underline{\sigma},\underline{\theta}}$ for all $v \leq w$ it would follow that $w \in L_{\underline{\sigma},\underline{\theta}}$, a contradiction. So $\underline{\sigma} \rightarrow_{\underline{\theta}}^{w} \underline{\sigma''}$ for some $\underline{\sigma''} \in \Sigma_{\mathrm{N}}$. But by 2), $L_{\underline{\sigma},\underline{\theta}} \cup \{w\} \subseteq L_{\underline{\sigma''},\underline{\theta}}$, so $|V \setminus L_{\underline{\sigma''},\underline{\theta}}| < |V \setminus L_{\underline{\sigma},\underline{\theta}}|$.

By induction, there exists a maximal execution $\underline{\sigma''} \rightarrow_{\underline{\theta}}^{x} \underline{\sigma'}$, but now $\underline{\sigma} \rightarrow_{\underline{\theta}}^{w.x} \underline{\sigma'}$ is a maximal execution. □

We also need to know that the system is deterministic, that is any given input and set of parameters, there is a unique output. The next result establishes (B).

Proposition 2. Suppose that $\underline{\sigma}_1, \underline{\sigma}_2 \in Stbl_{\underline{\theta}}(\Sigma_{\mathbf{N}})$ such that $\underline{x}_{\underline{\sigma}_1} = \underline{x}_{\underline{\sigma}_2}$, then $\underline{\sigma}_1 = \underline{\sigma}_2$.

Proof. Define

$$T_{\underline{\sigma}_1,\underline{\sigma}_2,\underline{\theta}} = \{w \in V^+ : v \leq w \Rightarrow \underline{\sigma}_1(v) = \underline{\sigma}_2(v)\} \quad (14)$$

which is the set of nodes w, such that all the outputs of the nodes from the two states are equal. We observe that $T_{\underline{\sigma}_1,\underline{\sigma}_2,\underline{\theta}} = V^+ \Leftrightarrow \underline{\sigma}_1 = \underline{\sigma}_2$. It is also the case that for all $v \in leaf(V)$, $\perp_v \in T_{\underline{\sigma}_1,\underline{\sigma}_2,\underline{\theta}}$, by hypothesis, so $T_{\underline{\sigma}_1,\underline{\sigma}_2,\underline{\theta}} \neq \varnothing$. We shall assume that $T_{\underline{\sigma}_1,\underline{\sigma}_2,\underline{\theta}} \neq V^+$ and obtain a contradiction. Suppose that w is minimal in $V^+ \setminus T_{\underline{\sigma}_1,\underline{\sigma}_2,\underline{\theta}}$, then $v \in T_{\underline{\sigma}_1,\underline{\sigma}_2,\underline{\theta}}$ for all $v < w$ and in particular, $^{\bullet}w \subseteq T_{\underline{\sigma}_1,\underline{\sigma}_2,\underline{\theta}}$. From the stability definition

$$\underline{\sigma}_1(w) = \phi_w(\underline{\theta}, \underline{z}_{\underline{\sigma}_1,\underline{\theta}}) = \phi_w(\underline{\theta}, \underline{z}_{\underline{\sigma}_2,\underline{\theta}}) = \sigma_2(w) \quad (15)$$

But $v \in T_{\underline{\sigma}_1,\underline{\sigma}_2,\underline{\theta}}$ for all $v < w$, so $w \in T_{\underline{\sigma}_1,\underline{\sigma}_2,\underline{\theta}}$, the required contradiction. □

Given the two propositions and the definition of input and output, we may now describe the function computed by a multi-net in a given configuration.

Theorem 1. Suppose that N is a multi-net and that $\underline{\theta} \in \underline{\Theta}_{\mathrm{N}}$, then there is a total function

$$G_{\mathrm{N},\underline{\theta}} : I_{\mathrm{N}} \rightarrow O_{\mathrm{N}} \quad (16)$$

such that if $\underline{\sigma} \in Stbl_{\underline{\theta}}(\Sigma_{\mathrm{N}})$, then

$$G_{\mathrm{N},\underline{\theta}}(\underline{x}_{\underline{\sigma}}) = y_{\underline{\sigma}} \tag{17}$$

Proof. The function $G_{\mathrm{N},\underline{\theta}}$ is well defined because if $\underline{\sigma}_1, \underline{\sigma}_2 \in Stbl_{\underline{\theta}}(\Sigma_{\mathrm{N}})$ and $\underline{x}_{\underline{\sigma}_1} = \underline{x}_{\underline{\sigma}_2}$, then by Proposition 2, $\underline{\sigma}_1 = \underline{\sigma}_2$ and in particular, $y_{\underline{\sigma}_1} = y_{\underline{\sigma}_2}$.

The function is total because if $\underline{x} \in I_{\mathrm{N}}$ and $\underline{\sigma} \in \Sigma_{\mathrm{N}}$ is any state such that $\underline{x}_{\underline{\sigma}} = \underline{x}$, then by Proposition 1, there exists a maximal execution, $\underline{\sigma} \rightarrow_{\underline{\theta}}^{x} \underline{\sigma'}$ and clearly $\underline{x}_{\underline{\sigma'}} = \underline{x}_{\underline{\sigma}} = \underline{x}$, since no transition alters the values on the inputs to the multi-net. Hence, $G_{\mathrm{N},\underline{\theta}}(\underline{x})$ is defined and equal to $y_{\underline{\sigma'}}$. □

If all the sets I_{v_i} are equal to some set I, then we say that N is *uniform* and in such cases, we have a function $\hat{G}_{\mathrm{N},\underline{\theta}} : I \rightarrow O_{\mathrm{N}}$ defined by

$$\hat{G}_{\mathrm{N},\underline{\theta}}(x) = G_{\mathrm{N},\underline{\theta}}(\underline{x}) \tag{18}$$

where $\underline{x}(\perp_v) = x$, for each $v \in leaf(V)$.

3.4. Multi-net systems: strategies for learning

We now turn to the training of multi-nets, and specifically to systems governed by some criterion function that can be used to measure the convergence of the system to the criterion. The criterion is liberal in the sense that it can specify any required stopping condition. For example, we might treat such criterion functions as error functions used in supervised learning systems, where the error is some measure of the difference between the desired and actual output of the system. However, this does not exclude other criteria or types of learning, such as unsupervised learning systems, in which our criterion may be measured in a way not necessarily related to a target response, for example through a simple number of learning cycles.

Taking our abstract view, we consider training to be the application of an operator S to parameterizations. Thus, for a given (uniform) multi-net N, a parameterization $\underline{\theta} \in \Theta$, an input $\underline{x} \in I_{\mathrm{N}}$ and an output $y \in I_{\mathrm{N}}$, the operator will generate a new parameterization $\underline{\theta'} \in \Theta$.

We may therefore identify a strategy with a function:

$$T : I_{\mathrm{N}} \times O_{\mathrm{N}} \times \underline{\Theta}_{\mathrm{N}} \rightarrow \underline{\Theta}_{\mathrm{N}} \tag{19}$$

so that with the notation of the previous paragraph, $\underline{\theta'} = T(\underline{x}, y, \underline{\theta})$. In fact, as the strategy will only be applied when the multi-net has attained a stable state following a feedforward pass, we may consider a strategy to be defined by a function

$$S : I_{\mathbf{N}} \times \underline{\Theta}_{\mathbf{N}} \rightarrow \underline{\Theta}_{\mathbf{N}} \tag{20}$$

where $S(\underline{x}, \underline{\theta}) = T(\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x}), \underline{\theta})$.

Let us consider this in connection with the whole training process. We fix $\underline{x} \in I_{\mathrm{N}}$ and suppose that N has an initial parameterization $\underline{\theta}_0 \in \underline{\Theta}_{\mathbf{N}}$. Repeated application of a feedforward pass, together with the application of the strategy as embodied by S gives us a sequence of parameterizations $\underline{\theta}_0, \ldots, \underline{\theta}_r, \ldots$, where for each $r \geq 0$, $\underline{\theta}_{r+1} = S(\underline{x}, \underline{\theta}_r)$. We note that if $\underline{\theta}_{r+1} = \underline{\theta}_r$, then

$$\underline{\theta}_{r+2} = S(\underline{x}, \underline{\theta}_{r+1}) = S(\underline{x}, \underline{\theta}_r) = \underline{\theta}_{r+1} \tag{21}$$

and by induction this gives:

Lemma 1. If $\underline{\theta}_{r+1} = \underline{\theta}_r$, then for all $i \geq r$, $\underline{\theta}_i = \underline{\theta}_r$. □

If $\underline{\theta}_{r+1} = \underline{\theta}_r$, for all $i \geq r$, then we say that S converges for $\underline{x}$ from $\underline{\theta}_0$.

Training will have some form of goal, of course, and we shall suppose that this is expressed by a set; strictly speaking, the extension of a predicate:

$$P \subseteq I_{\mathrm{N}} \times O_{\mathrm{N}} \tag{22}$$

P expresses a goal in the sense that training has succeeded if $(\underline{x}, G_{\mathrm{N},\underline{\theta}}(\underline{x})) \in P$; P may be considered as the extension of a predicate defining successful training.

We shall say that S is a strategy for N with respect to criterion P if and only if

$$S(\underline{x}, \underline{\theta}) = \underline{\theta} \Leftrightarrow (\underline{x}, G_{\mathrm{N},\underline{\theta}}(\underline{x})) \in P \tag{23}$$

Lemma 1 tells us that $S(\underline{x}, \underline{\theta}) = \underline{\theta}$ precisely when S converges to $\underline{\theta}$ from some initial parameterization, so S is a strategy with respect to P precisely when it converges to a parameterization satisfying P.

In the previous section we defined the feedforward operation of a multi-net system using partially ordered sets and state transitions. Along the way we have had to provide formal proof of certain properties of these systems to ensure completeness and rigor (such as the stable and deterministic output of the system). Whilst this is perhaps lengthy, this formal approach is necessary in order to provide a solid foundation for the exploration of multi-net system properties. Lastly, we have provided a way of describing the learning process within the system. However, as yet we have provided no implementation detail using the framework, or exploited the mathematical properties of the definitions. In the next section we start to do this by exploring a supervised learning strategy for the generic class of multi-net systems.

4. Supervised learning for multi-net systems

So far we have provided a formal definition of a multi-net system and its associated learning algorithm. The most important aspect of this approach is that it does not constrain the type of networks that can be combined. All that is required is a suitable topology, set of feedforward functions, parameters and learning strategy functions. However, whilst this definition is perhaps interesting, to be useful we need to consider example systems in which we can start to use this formal foundation to infer properties of

the combined system.

In this section we first motivate our discussion on learning by considering an arbitrary multi-layer perceptron (MLP) trained using a supervised learning technique, such as the host of gradient descent algorithms. However, we do this only to abstract the notion of a backward training pass, without constraining ourselves with unnecessary details, such as requiring the computation of gradients. By treating the MLP as a series of individual layers that are themselves separate networks, we generalize the notion of an arbitrary feedforward network to a generic sequential and parallel combination of networks that can be trained using an abstract algorithm. Such a definition can already encompass a wide range of techniques, including partially connected feedforward networks, in-situ trained ensembles and ME.

4.1. System definition

We start by considering the simple MLP as shown in Fig. 2, which, without loss of generality, we have considered as a two-layer network with an arbitrary number of inputs, hidden layer neurons (n) and outputs (m). Each unit operates using a combination of inputs, together with an activation function.

We define $\mathrm{N}_{mlp} = (F_{mlp}, \Theta_{mlp}, \Phi_{mlp})$ as a multi-net, by

1) $F_{mlp} = (V_{mlp}, \leq)$, with $V_{mlp} = \{t, u\}$ and $u \lhd t$;
2) $\Theta_{mlp} = \{\Theta_t, \Theta_u\}$, $\Theta_u = \mathrm{R}^{\mathrm{n}}$ and $\Theta_t = \mathrm{R}^{\mathrm{m}}$ where R denotes the set of real numbers;
3) $\Phi_{mlp} = \{\phi_t, \phi_u\}$.

4.2. Learning as a backward pass

We now consider how this simple system is trained using a backward pass of adjustments to parameters, before extending this to the concept of a multi-net system. In such backward propagating techniques, each node receives some form of instruction from its parent in order to update its weights. Depending upon this instruction, and its current parameterization and output, it will modify its parameters and propagate an instruction down to its children. A simple example of this is the basic backpropagation algorithm [40] in which the notional error of each hidden neuron is calculated using the error on the output. This suggests that at each node v we have a non-empty set A_v of instructions together with an *adjustment function*

$$j_v : A_v \times O_v \times \Theta_v \to \Theta_v, \tag{24}$$

and a family of *instruction propagation functions*

$$a_{v,u} : A_v \times O_v \times \Theta_v \to A_u, \; u \in {}^{\bullet}v \tag{25}$$

An application of j_v adjusts the parameters at node v according to an instruction a received, the current value y on v and the current parameter θ of v, giving a new parameter $j_v(a, y, \theta)$. Each child $u \in {}^{\bullet}v$ of v will then receive one instruction $a_{v,u}(a, y, \theta)$.

We assume that each set A_v contains an instruction to do nothing, which we denote by 0_v, so that we have:

$$j_v(0_v, y, \theta) = \theta, \; a_{v,u}(0_v, y, \theta) = 0_u \tag{26}$$

for all $y \in O_v$, $\theta \in \Theta_v$. We shall require further that only a zero instruction can keep the parameter at the root node fixed, that is:

$$j_{root(V)}(a, y, \theta) = \theta \Rightarrow a = 0_{root(V)}. \tag{27}$$

We shall refer to this as the *strictness condition*.

At this point we note that none of these definitions constrains the topology of the system, except that it can be described using a partially ordered set with a maximal element. Furthermore, our description of the adjustment functions are only motivated by algorithms such as backpropagation, without having to define what adjustments are made, or indeed whether they are consequent from supervised learning or otherwise. As such, we have therefore abstracted the notion of an MLP to the generic class of multi-net systems being trained using an abstract learning algorithm.

As with the feedforward sweep, we adopt a state-based approach to the description of the backward pass. That is to say, we conceive of the multi-net going through a series of changes as the adjustments sweep down from the root.

A state is an instantaneous snapshot of the system. Given the purpose of the j_v and $a_{v,u}$ functions, we see that what the state must record are the instructions, outputs and parameters currently at each node. Consequently, we define a *feedback state* to be a function that maps the nodes to the union of instructions, outputs and parameterizations for each node:

$$\underline{\rho} : V \to \bigcup_{v \in V} A_v \times \bigcup_{v \in V} O_v \times \bigcup_{v \in V} \Theta_v \tag{28}$$

such that for each $v \in V$, $\underline{\rho}(v) \in A_v \times O_v \times \Theta_v$. If $\underline{\rho}(v) = (a, y, \theta)$, then we define $a_{\underline{\rho}(v)} = a$, $y_{\underline{\rho}(v)} = y$ and $\theta_{\underline{\rho}(v)} = \theta$. Thus $a_{\underline{\rho}(v)}$ gives the instruction considered by v in state $\underline{\rho}$ and similarly for the current output and current parameter. Let Ψ_{N} denote the set of all feedback states.

We also have a notion of state transition, the expression $\underline{\rho} \to^{v} \underline{\rho}'$ says that if the algorithm is applied locally at node v in state $\underline{\rho}$, then the state will be transformed to $\underline{\rho}'$. $\underline{\rho} \to^{v} \underline{\rho}'$ holds precisely when for all $w \in V$:

$$a_{\underline{\rho}'(w)} = \begin{cases} a_{\underline{\rho}(w)} \text{ if } w \not\trianglelefteq v \\ a_{v,w}(a_{\underline{\rho}(v)}, y_{\underline{\rho}(v)}, \theta_{\underline{\rho}(v)}) \text{ otherwise} \end{cases} \quad (29)$$

$$y_{\underline{\rho}'(w)} = y_{\underline{\rho}(w)} \quad (30)$$

$$\theta_{\underline{\rho}'(w)} = \begin{cases} \theta_{\underline{\rho}(w)} \text{ if } w \neq v \\ j_v(a_{\underline{\rho}(v)}, y_{\underline{\rho}(v)}, \theta_{\underline{\rho}(v)}) \text{ otherwise} \end{cases} \quad (31)$$

So the application of the algorithm at node v propagates an instruction down to each of the children of v according the functions $a_{v,u}$, leaving everything else unchanged. (Outputs at nodes only change in the feedforward pass.) Finally, the application of the algorithm at v will change its own local parameters, leaving all others unchanged.

Putting together the node modifications together in the case of a backward pass is not quite as simple as in the case of the feedforward formalism, since there is no concept corresponding to a stable state, which guarantees that each node is only modified once during a sweep. We have to impose this explicitly here. Essentially, the idea is that no node v should be modified until every node $w \geq v$ has been modified. We should therefore consider sequences of the form

$$\underline{\rho} \rightarrow^{v_1} \underline{\rho}_1 \cdots \rightarrow^{v_n} \underline{\rho}_n \quad (32)$$

such that $v_i > v_j \Rightarrow i < j$. Note that this means $v_1 = root(V)$. Any such sequence $v_1 \cdots v_n$ is called a *topological sort of* V and we denote the set of all topological sorts of V by $TS(V)$. We now have a function:

$$R : TS(V) \times \Psi_{\mathrm{N}} \rightarrow \Psi_{\mathrm{N}} \quad (33)$$

given by

$$R(v_1 \cdots v_n, \underline{\rho}) = \underline{\rho}' \Leftrightarrow \underline{\rho} \rightarrow^{v_1} \underline{\rho}_1 \cdots \rightarrow^{v_n} \underline{\rho}_n = \underline{\rho}' \quad (34)$$

On the face of it, it would seem that the application of the algorithm depends on the order in which it is applied to the nodes. Fortunately, this is not the case, as the following result shows.

Proposition 3. With the above notation, if $\alpha, \alpha' \in TS(V)$, then $R(\alpha, \underline{\rho}) = R(\alpha', \underline{\rho})$, for all $\underline{\rho} \in \Psi_{\mathbf{N}}$.

Proof. Let $v, v' \in V$ and define $v \iota v' \Leftrightarrow v \not\trianglelefteq v' \wedge v' \not\trianglelefteq v$. If $\alpha, \alpha' \in V^*$, then define $\alpha \equiv^{(1)} \alpha'$ if and only if there exists $\beta, \beta'' \in V^*$ and $v, v' \in V$ such that $\alpha = \beta v v' \beta''$, $\alpha' = \beta v' v \beta''$ and $v \iota v'$. Now define $\alpha \equiv \alpha'$ if and only if either $\alpha = \alpha'$ or there exists $\alpha_1, ..., \alpha_n \in TS(V)$, such that $\alpha = \alpha_1 \equiv^{(1)} ... \equiv^{(1)} = \alpha_n = \alpha$. First, we show:

1) If $\alpha \in TS(V)$ and $\alpha \equiv \alpha'$, then $\alpha' \in TS(V)$ and $R(\alpha, \underline{\rho}) = R(\alpha', \underline{\rho})$;

In view of the definition of $\equiv$, in order to prove 1) it suffices to prove that

2) If $\alpha \in TS(V)$ and $\alpha \equiv^{(1)} \alpha'$, then $\alpha' \in TS(V)$ and $R(\alpha, \underline{\rho}) = R(\alpha', \underline{\rho})$;

Next we prove

3) If $\alpha, \alpha' \in TS(V)$, then $\alpha \equiv \alpha'$.

The proof of the proposition now proceeds as follows. If $\alpha, \alpha' \in TS(V)$, then $\alpha \equiv \alpha'$ by 3), and so $R(\alpha, \underline{\rho}) = R(\alpha', \underline{\rho})$ by 1). We prove (2) and (3).

For 2), suppose that $\alpha = v_1 \cdots v_r v v' v_{r+1} \cdots v_n$ and $\alpha' = v_1 \cdots v_r v' v v_{r+1} \cdots v_n$ with $v \iota v'$, so that $\alpha \equiv^{(1)} \alpha'$. If $\alpha' \notin TS(V)$, then we would have to have $v \leq v'$. But, this would contradict $v \iota v'$. Hence, $\alpha' \in TS(V)$. We also note that if $\underline{\rho} \rightarrow^{v} \underline{\rho}' \rightarrow^{v'} \hat{\underline{\rho}}$, $\underline{\rho} \rightarrow^{v'} \underline{\rho}'' \rightarrow^{v'} \hat{\underline{\rho}}'$ and $v \iota v'$, then a simple calculation shows that $\hat{\underline{\rho}} = \hat{\underline{\rho}}'$. Hence, $R(\alpha, \underline{\rho}) = R(\alpha', \underline{\rho})$. We have argued that if $\alpha \in TS(V)$ and $\alpha \equiv^{(1)} \alpha'$, then $\alpha' \in TS(V)$ and $R(\alpha, \underline{\rho}) = R(\alpha', \underline{\rho})$. Now 1) follows, as we have already explained.

For 3), suppose that $\alpha, \alpha' \in TS(V)$. We argue by induction on $n(\alpha') = |V| - \ell(\alpha \wedge \alpha')$, where $\alpha \wedge \alpha'$ denotes the longest common prefix of α and α', and in general $\ell(\beta)$ denotes the length of β. For example, if $V = \{a,b,c,d\}$, $\alpha = abcd$, $\alpha' = abdc$, then $|V| = 4$, $\alpha \wedge \alpha' = ab$, and so $|V| - \ell(\alpha \wedge \alpha') = 4 - \ell(ab) = 2$.

In the base case of the induction, $n(\alpha') = 0$, so that $|V| = \ell(\alpha \wedge \alpha')$. But $|V| = \ell(\alpha) = \ell(\alpha')$, so in this case $\alpha = \alpha'$ and so $\alpha \equiv \alpha'$ by definition of $\equiv$. For the induction step, we show that there exists $\alpha'' \in TS(V)$ such that $\alpha' = \alpha''$ and $n(\alpha'') < n(\alpha')$. By induction, $\alpha = \alpha''$, but $\equiv$ is an equivalence relation, so $\alpha \equiv \alpha'$, as required.

Suppose that $|V| - \ell(\alpha \wedge \alpha') > 0$ and suppose that $\alpha = \beta v \gamma$, $\alpha' = \beta v_1 \cdots v_r v \gamma'$, $v_1, \cdots, v_r, v \in V$, $\beta, \gamma, \gamma' \in V^*$ with $v \neq v$, so that $\beta = \alpha \wedge \alpha'$. As $\beta v_1 \cdots v_r v \gamma' \in TS(V)$, $v \not\trianglelefteq v_i$, $1 \leq i \leq r$, and as all the v_i occur in γ and $\beta v \gamma \in TS(V)$, $v_i \not\trianglelefteq v$, $1 \leq i \leq r$, hence, $v_i \iota v$, $1 \leq i \leq r$, and so

$$\begin{aligned} \alpha' = \beta v_1 \cdots v_r v \gamma' &\equiv^{(1)} \beta v_1 \cdots \\ &\cdots \equiv v_{r-1} v v_r \gamma' \equiv^{(1)} \cdots \equiv^{(1)} \beta v v_1 \cdots v_r \gamma' = \alpha'' \end{aligned}, \quad (35)$$

Hence $\alpha' \equiv \alpha''$. But now $\alpha'' \in TS(V)$, by the first part of

the proof, and $\beta v \le \alpha, \alpha''$, so that $\ell(\alpha \wedge \alpha'') > \ell(\alpha \wedge \alpha')$ and hence $n(\alpha'') < n(\alpha')$ By induction $\alpha \equiv \alpha''$, and as $\alpha' \equiv \alpha''$, $\alpha \equiv \alpha'$, by transitivity. □

As a result, we have a function $B : \Psi_{\mathrm{N}} \to \Psi_{\mathrm{N}}$ given by $B(\underline{\rho}) = R(\alpha, \underline{\rho})$ for any $\alpha \in TS(V)$. $B(\underline{\rho})$ is the new feedback state following one sweep of the algorithm.

There is one further matter to consider and that is how the backward pass is initiated. We propose that this depends on a function

$$Q : I_{\mathrm{N}} \times O_{\mathrm{N}} \to A_{root(V)} \tag{36}$$

which compares the input with the output and issues an instruction to the root node. Note that, in line with our aim of maximal generality, we have not specified *how* the input and output are compared here.

4.3. Abstract training strategy

Let us now see how our abstract scheme determines a strategy – does it lead to convergence? Let $\underline{x} \in I_{\mathrm{N}}$ and $\underline{\theta} \in \underline{\Theta}_{\mathrm{N}}$. We know from Propositions 1 and 2 that there exists a unique $\underline{\sigma} \in Stbl_{\underline{\theta}}(\Sigma_{\mathbf{N}})$ such that $\underline{\sigma} = \underline{\sigma}_{\underline{x}}$. We define $\underline{\rho}(\underline{x}, \underline{\theta}) \in \Psi_{\mathbf{N}}$ as follows, for each $v \in V$

$$a_{\underline{\rho}(\underline{x},\underline{\theta})(v)} = \begin{cases} Q(\underline{x}, G_{\mathrm{N},\underline{\theta}}(\underline{x})) \text{ if } v = root(V) \\ 0_v \text{ otherwise} \end{cases} \tag{37}$$

$$y_{\underline{\rho}(\underline{x},\underline{\theta})(v)} = y_v \tag{38}$$

$$\theta_{\underline{\rho}(\underline{x},\underline{\theta})(v)} = \theta_v \tag{39}$$

In words, $\underline{\rho}(\underline{x}, \underline{\theta})$ represents the situation in which a feedforward pass has just completed with parameterization $\underline{\theta}$, no node except the root is being instructed to change and the root is to receive the instruction computed by Q on the basis of the current input and output of the multi-net.

Definition 4 An abstract backward pass learning scheme for a multi-net N is a triple $\mathrm{B} = (Q, \mathrm{A}, \mathrm{J})$, where

1) $Q : I_{\mathrm{N}} \times O_{\mathrm{N}} \to A_{root(V)}$;
2) A is an indexed family of instruction propagation functions $a_{v,u}$, $v \in V$, $u \in {}^{\bullet}v$;
3) J is an indexed family of adjustment functions j_v, $v \in V$.

Let us now see how our scheme determines a strategy. Given $\underline{x} \in I_{\mathrm{N}}$ and $\underline{\theta} \in \underline{\Theta}_{\mathrm{N}}$, we construct $\underline{\rho}(\underline{x}, \underline{\theta})$. We now appeal to Proposition 3, which asserts the existence of a feedback state $B(\underline{\rho}(\underline{x}, \underline{\theta})) \in \Psi_{\mathrm{N}}$. To obtain $S(\underline{x}, \underline{\theta})$, we simply extract the parameters from $B(\underline{\rho}(\underline{x}, \underline{\theta}))$, that is, for each $v \in V$, define

$$S_B(\underline{x}, \underline{\theta})_v = \underline{\theta}_{B(\underline{\rho}(\underline{x},\underline{\theta}))(v)} \tag{40}$$

Proposition 4. Suppose that $\mathrm{B} = (Q, \mathrm{A}, \mathrm{J})$ is an abstract scheme, then S_B is a training strategy with respect to the predicate P_B given by

$$(\underline{x}, y) \in P_B \Leftrightarrow Q(\underline{x}, y) = 0_{root(V)}. \tag{41}$$

Proof. If $Q(\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x}, \underline{\theta})) = 0_{root(V)}$, then $a_{\underline{\rho}(\underline{x},\underline{\theta})(v)} = 0_v$, for all $v \in V$, by (37) and hence $S_B(\underline{x}, \underline{\theta}) = \underline{\theta}$. Conversely, if $S_B(\underline{x}, \underline{\theta}) = \underline{\theta}$, then in particular,

$$\begin{aligned} \underline{\theta}_{root(V)} &= S_B(\underline{x}, \underline{\theta})_{root(V)} \\ &= j_{root(V)}(Q(\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x})), G_{\mathbf{N},\underline{\theta}}(\underline{x}), \underline{\theta}_{root(V)}) \end{aligned} \tag{42}$$

so that $Q(\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x}, \underline{\theta})) = 0_{root(V)}$, by (27). Now apply (23). □

We know that every abstract scheme determines a strategy. Not all strategies may so be determined; Proposition 5 will show that such strategies must be regular, in the sense defined by Definition 5. Broadly speaking, regularity reflects the manner in which abstract schemes generate strategies. For example, a strategy S_B will make the same alterations at the root node given the same pair $(\underline{x}, y)$ and parameterization. This is one aspect of 1) of Definition 5. Indeed, if $\underline{\theta}, \underline{\theta}' \in \Theta_{\mathbf{N}}$ agree on all nodes above a node $v \in V$, then S_B will have the same effect on their parameters. We capture these ideas using the relations $\equiv_{\underline{x},v}$, defined below.

First, if $v \in V$, then define $\uparrow v = \{w \in V : w \ge v\}$. Suppose $(\underline{x}, \underline{\theta}) \in I_{\mathbf{N}} \times \underline{\Theta}_{\mathbf{N}}$, then by Propositions 1 and 2, there exists a unique $\underline{\sigma} \in Stbl(\Sigma_{\mathbf{N}})$ such that, $\underline{\sigma} = \underline{\sigma}_{\underline{x}}$. We shall name this state $\underline{\sigma}(\underline{x}, \underline{\theta})$. Note that $G_{\mathbf{N},\underline{\theta}}(\underline{x}) = \underline{\sigma}(\underline{x}, \underline{\theta})_{root(V)}$. Now, for each $v \in V$ and $\underline{x}, \in I_{\mathbf{N}}$, we define a relation $\equiv_{\underline{x},v}$ over $\underline{\Theta}_{\mathbf{N}}$ by

$$\underline{\theta} \equiv_{\underline{x},v} \underline{\theta}' \Leftrightarrow \forall w \in \uparrow v . \underline{\theta}_w = \underline{\theta}'_w \wedge \underline{\sigma}(\underline{x}, \underline{\theta})_w = \underline{\sigma}(\underline{x}, \underline{\theta}')_w \tag{43}$$

Of course, $\equiv_{\underline{x},v}$ is an equivalence relation.

The following definition lists those properties of strategies S_B which, as we shall see later, in Theorem 2, precisely characterise them.

Definition 5. We shall define a strategy S to be *regular*, if and only if for each $v \in V$, $\underline{\theta}, \underline{\theta}' \in \Theta_{\mathbf{N}}$ and $\underline{x} \in I_{\mathbf{N}}$,

1) $\underline{\theta} \equiv_{\underline{x},v} \underline{\theta}' \Rightarrow \forall u \ge v . S(\underline{x}, \underline{\theta})_u = S(\underline{x}, \underline{\theta}')_u$;
2) $S(\underline{x}, \underline{\theta}) = \underline{\theta} \wedge G_{\mathbf{N},\underline{\theta}}(\underline{x}) = G_{\mathbf{N},\underline{\theta}'}(\underline{x}) \Rightarrow S(\underline{x}, \underline{\theta}') = \underline{\theta}'$;
3) $S(\underline{x}, \underline{\theta})_{root(V)} = \underline{\theta}_{root(V)} \Rightarrow S(\underline{x}, \underline{\theta}) = \underline{\theta}$.

Proposition 5. Suppose that B is an abstract scheme, then S_B is regular.

Proof. Let $v \in V$, $\underline{\theta}, \underline{\theta}' \in \Theta_{\mathbf{N}}$ and $\underline{x} \in I_{\mathbf{N}}$. We establish the conditions of definition 5.

1) Suppose that $\underline{\theta} \equiv_{\underline{x},v} \underline{\theta}'$ and let suppose that $v_1 \cdots v_n \in TS(V)$. Let $\underline{\rho}_0 = \underline{\rho}(\underline{x}, \underline{\theta}) \rightarrow^{v_1} \underline{\rho}_1 \cdots \rightarrow^{v_n} \underline{\rho}_n$ and $\underline{\rho}'_0 = \underline{\rho}(\underline{x}, \underline{\theta}') \rightarrow^{v_1} \underline{\rho}'_1 \cdots \rightarrow^{v_n} \underline{\rho}'_n$. We observe that

$$\underline{\theta}_{\underline{\rho}_i(v_j)} = \begin{cases} \underline{\theta}_{v_j} \text{ if } i < j \\ S_B(\underline{x}, \underline{\theta})_{v_j} \text{ otherwise} \end{cases} \tag{44}$$

$$\underline{\theta}_{\underline{\rho}'_i(v_j)} = \begin{cases} \underline{\theta}'_{v_j} \text{ if } i < j \\ S_B(\underline{x}, \underline{\theta}')_{v_j} \text{ otherwise} \end{cases} \tag{45}$$

Now, $v = v_r$, for some $1 \le r \le n$ and by the definition of topological sort, if $u \in \uparrow v$, then $u = v_i$ for some $i \le r$. Hence, for all $i \le r$, $\underline{\theta}_{v_i} = \underline{\theta}'_{v_i}$ and $\underline{\sigma}(\underline{x}, \underline{\theta})_{v_i} = \underline{\sigma}(\underline{x}, \underline{\theta}')_{v_i}$.

By (37) and the fact that $G_{\mathbf{N},\underline{\theta}}(\underline{x}) = \underline{\sigma}(\underline{x}, \underline{\theta}')_{v_1}$,

$$\begin{aligned} a_{\underline{\rho}_1(v_1)} &= Q(\underline{x}, \underline{\sigma}(\underline{x}, \underline{\theta})_{v_1}) = Q(\underline{x}, \underline{\sigma}(\underline{x}, \underline{\theta}')_{v_1}) = a_{\underline{\rho}'_1(v_1)} \\ \theta_{\underline{\rho}_1(v_1)} &= j_{v_1}(Q(\underline{x}, \underline{\sigma}(\underline{x}, \underline{\theta})_{v_1}), \underline{\sigma}(\underline{x}, \underline{\theta})_{v_1}, \underline{\theta}_{v_1}) \\ &= j_{v_1}(Q(\underline{x}, \underline{\sigma}(\underline{x}, \underline{\theta}')_{v_1}), \underline{\sigma}(\underline{x}, \underline{\theta}')_{v_1}, \underline{\theta}'_{v_1}) = \theta_{\underline{\rho}'_1(v_1)} \end{aligned} \tag{46}$$

We now assume that if $i < r$, then for all $j \le i$, then

$$\theta_{\underline{\rho}_i(v_j)} = \theta_{\underline{\rho}'_i(v_j)} \wedge a_{\underline{\rho}_i(v_j)} = a_{\underline{\rho}'_i(v_j)} \tag{47}$$

And prove that this holds for $i+1$. The proof of 1) now follows by induction. We have already established the base case $i = 1$.

Let $j \le i+1$. If $j \le i$, then

$$a_{\underline{\rho}_{i+1}(v_j)} = a_{\underline{\rho}_i(v_j)} = a_{\underline{\rho}'_i(v_j)} = a_{\underline{\rho}'_{i+1}(v_j)} \tag{48}$$

whereas if $j = i+1$, then by the property of topological sorts, $v_j \in {}^{\bullet}v_k$ for $k \le i$, and so

$$\begin{aligned} a_{\underline{\rho}_{i+1}(v_j)} &= a_{v_k, v_j}(a_{\underline{\rho}_k(v_k), \underline{\sigma}(\underline{x}, \underline{\theta})_{v_k}}, \underline{\theta}_{v_k}) \\ &= a_{v_k, v_j}(a_{\underline{\rho}'_k(v_k), \underline{\sigma}(\underline{x}, \underline{\theta}')_{v_k}}, \underline{\theta}'_{v_k}) = a_{\underline{\rho}'_{i+1}(v_j)} \end{aligned} \tag{49}$$

Similarly, if $j \le i$, then

$$\theta_{\underline{\rho}_{i+1}(v_j)} = \theta_{\underline{\rho}_i(v_j)} = \theta_{\underline{\rho}'_i(v_j)} = \theta_{\underline{\rho}'_{i+1}(v_j)} \tag{50}$$

where if $j = i+1$ and $v_j \in {}^{\bullet}v_k$ for $k \le i$, then

$$\begin{aligned} \theta_{\underline{\rho}_{i+1}(v_j)} &= j_{v_{i+1}}(a_{\underline{\rho}_{i+1}(v_{i+1}), \underline{\sigma}(\underline{x}, \underline{\theta}))_{v_{i+1}}}, \underline{\theta}_{v_{i+1}}) \\ &= j_{v_{i+1}}(a_{\underline{\rho}'_{i+1}(v_{i+1}), \underline{\sigma}(\underline{x}, \underline{\theta}'))_{v_{i+1}}}, \underline{\theta}'_{v_{i+1}}) = \theta_{\underline{\rho}'_{i+1}(v_j)} \end{aligned} \tag{51}$$

This completes the induction step. Next, we prove 2). We have, using (23)

$$\begin{aligned} &S_B(\underline{x}, \underline{\theta}) = \underline{\theta} \wedge G_{\mathbf{N},\underline{\theta}}(\underline{x}) = G_{\mathbf{N},\underline{\theta}'}(\underline{x}) \\ &\quad \Rightarrow Q(\underline{x}, G_{\mathbf{N},\underline{\theta}'}(\underline{x})) = Q(\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x})) = 0_{root(V)} \\ &\quad \Rightarrow S(\underline{x}, \underline{\theta}') = \underline{\theta}' \end{aligned} \tag{52}$$

Finally, we show 3). Using (27) and (31), we have

$$\begin{aligned} &S(\underline{x}, \underline{\theta})_{root(V)} = \underline{\theta}_{root(V)} \\ &\Rightarrow j_{root(V)}(Q(\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x})), G_{\mathbf{N},\underline{\theta}}(\underline{x}), \underline{\theta}_{root(V)}) = \underline{\theta}_{root(V)} \\ &\Rightarrow Q(\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x})) = 0_{root(V)} \Rightarrow S(\underline{x}, \underline{\theta}) = \underline{\theta} \end{aligned} \tag{53}$$

□

As we have pointed out, regularity characterises precisely those strategies which are determined by abstract schemes. The proof of this is by construction, so that we need to define the various sets and functions that make up an abstract scheme.

Unlike a strategy, an abstract scheme does not 'know' in advance the parameterizations and outputs at the nodes. However, as it percolates through the multi-net, it acquires knowledge of the values at the nodes it has already 'met'. The point of regularity is that this is all the information it needs to replicate the local effect of a strategy. The 'instruction' values will be records of such information, together with that of the current input. This information is encapsulated in the triples $(\alpha, \beta, \underline{x})$.

If $v \in V$, then we define $\underline{\uparrow} v = \uparrow v \setminus \{v\}$ and we define U_v to be the set of all functions

$$\alpha : \underline{\uparrow} v \rightarrow \bigcup_{w \in \underline{\uparrow} v} \Theta_w \tag{54}$$

such that $\alpha(w) \in \Theta_w$ for all $w \in \underline{\uparrow} v$. We also define V_v to be the set of all functions

$$\beta : \underline{\uparrow} v \rightarrow \bigcup_{w \in \underline{\uparrow} v} O_w \tag{55}$$

such that $\beta(w) \in \Theta_w$ for all $w \in \underline{\uparrow} v$. If $v \in V \setminus \{root(V)\}$, then we define

$$E_v = U_v \times V_v \times I_{\mathbf{N}} \text{ and } A_v = E_v \cup \{0_v\} \tag{56}$$

$$A_{root(V)} = (I_{\mathbf{N}} \times O_{\mathbf{N}}) \cup \{0_{root(V)}\}, \tag{57}$$

where we assume that 0_v is some element not appearing in E_v and that $0_{root(V)}$ is some element not appearing in $I_{\mathbf{N}} \times O_{\mathbf{N}}$.

As the 'instruction' values are records of the values already encountered, a new instruction should be generated from an old one by an augmentation by an extra pair of values. This is the point of the s function below.

If $v \in V \setminus \{root(V)\}$ and $(\alpha, \beta, \underline{x}) \in A_v$ and $(\hat{\theta}, \hat{y}) \in \Theta_v \times O_v$, then we define $s((\alpha, \beta, \underline{x}), \hat{\theta}, \hat{y}) = (\alpha', \beta', \underline{x})$, where

$$\alpha'(u) = \begin{cases} \alpha(u) \text{ if } u \in \underline{\uparrow} v \\ \hat{\theta} \text{ if } u = v \end{cases} \tag{58}$$

$$\beta'(u) = \begin{cases} \beta(u) \text{ if } u \in \uparrow v \\ \hat{y} \text{ if } u = v \end{cases} \tag{59}$$

We can easily check that if $\underline{a} \in A_v$ and $(\hat{\theta}, \hat{y}) \in \Theta_v \times O_v$, then $s(\underline{a}, \hat{\theta}, \hat{y}) \in A_u$ for every $u \in {}^\bullet v$. We use this fact without further comment in the definition of the $a_{v,u}$ functions.

We also define $s((\underline{x}, y), \hat{\theta}, \hat{y}) = (\alpha', \beta', \underline{x})$, where $\alpha'(root(V)) = \hat{\theta}$ and $\beta'(root(V)) = \hat{y}$. Of course, $s((\underline{x}, y), \hat{\theta}, \hat{y}) \in A_{root(V)}$.

Given $(\underline{x}, \underline{\theta}) \in I_{\mathbf{N}} \times \Theta_{\mathbf{N}}$, we have values $\underline{\theta}_u$ and $s(\underline{x}, \underline{\theta})_u$. The function $\underline{a}_v$ below selects those values sitting on nodes above v, which are precisely those encountered by the algorithm before reaching that node.

For each $v \in V \setminus \{root(V)\}$, define a function $\underline{a}_v : \Theta_{\mathbf{N}} \to A_v$ by

$$\begin{aligned} &\underline{a}_v(\underline{\theta}) = (\alpha, \beta, \underline{x}) \Leftrightarrow \\ &\forall w \in \uparrow v . \alpha(w) = \underline{\theta}_w \wedge \beta(w) = \underline{\sigma}(\underline{x}, \underline{\theta})_w \end{aligned} \tag{60}$$

It is an immediate consequence of the definition that for all $v \in V \setminus \{root(V)\}$ and $u \in {}^\bullet v$

$$\underline{a}_u(\underline{\theta}) = s(\underline{a}_v(\underline{\theta}), \underline{\sigma}(\underline{x}, \underline{\theta})_v, \underline{\theta}_v) \tag{61}$$

The idea behind this construction is that $\underline{a} \in A_v$ records those parts of $\underline{\theta}$ and $\underline{\sigma}(\underline{x}, \underline{\theta})$ that the algorithm has encountered on the way to node v and that regularity ensures that this information is enough to determine how it should modify the parameter at v, as the next lemma makes clear.

Lemma 2: Suppose that S is a regular strategy $\underline{x} \in I_{\mathrm{N}}$ and $\underline{\theta}, \underline{\theta}' \in \Theta_{\mathrm{N}}$, then

1) If $\underline{\theta}_{root(V)} = \underline{\theta}'_{root(V)}$ and $G_{N,\underline{\theta}}(\underline{x}) = G_{N,\underline{\theta}'}(\underline{x})$, then $S(\underline{x}, \underline{\theta})_{root(V)} = S(\underline{x}, \underline{\theta}')_{root(V)}$;
2) If $\underline{a}_v(\underline{\theta}) = \underline{a}_v(\underline{\theta}')$, $\underline{\theta}_v = \underline{\theta}'_v$ and $\underline{\sigma}(\underline{x}, \underline{\theta})_v = \underline{\sigma}(\underline{x}, \underline{\theta}')_v$, then $S(\underline{x}, \underline{\theta})_v = S(\underline{x}, \underline{\theta}')_v$.

Proof.

1) The hypothesis translates as $\underline{\theta} \equiv_{\underline{x}, root(V)} \underline{\theta}'$ and the conclusion follows from 1) of Definition 5.
2) Let $w \in \uparrow v$ so by (61), $\underline{a}_w(\underline{\theta}) = \underline{a}_w(\underline{\theta}') = (\alpha, \beta, \underline{x})$, say. So, $\underline{\theta}_w = \alpha(w) = \underline{\theta}'_w$ and $\underline{\sigma}(\underline{x}, \underline{\theta})_w = \beta(w) = \underline{\sigma}(\underline{x}, \underline{\theta}')_w$. As $\underline{\theta}_v = \underline{\theta}'_v$ and $\underline{\sigma}(\underline{x}, \underline{\theta})_v = \underline{\sigma}(\underline{x}, \underline{\theta}')_v$, it follows that $\underline{\theta} \equiv_{\underline{x}, v} \underline{\theta}'$. By regularity, $S(\underline{x}, \underline{\theta})_v = S(\underline{x}, \underline{\theta}')_v$. □

We next observe that if $|\Theta_{root(V)}| = 1$, then we will always have $S(\underline{x}, \underline{\theta})_{root(V)} = \underline{\theta}_{root(V)}$, whence $S(\underline{x}, \underline{\theta}) = \underline{\theta}$, by 3) of Definition 5. If $S(\underline{x}, \underline{\theta}) = \underline{\theta}$, then $S = S_B$, where $Q^S(\underline{x}, y) = 0_{root(V)}$, for all $(\underline{x}, y) \in I_{\mathbf{N}} \times O_{\mathbf{N}}$. We shall therefore assume that $|\Theta_{root(V)}| > 1$, so that by the axiom of choice there exists a bijective function

$$\tau : \Theta_{root(V)} \to \Theta_{root(V)} \tag{62}$$

such that $\tau(\theta) \neq \theta$ for each $\theta \in \Theta_{root(V)}$. Fix such a function for the purpose of the following construction.

We now pass to the construction; we define $B^S = (Q^S, \mathbf{A}^S, \mathbf{J}^S)$, where

$$Q^S(\underline{x}, y) = \begin{cases} 0_{root(V)} \text{ if } \exists \underline{\theta} \in \Theta_{\mathbf{N}} . S(\underline{x}, \underline{\theta}) = \underline{\theta} \\ \qquad \wedge G_{\mathbf{N}, \underline{\theta}}(\underline{x}) = y \\ (\underline{x}, y) \text{ otherwise} \end{cases} \tag{63}$$

To repeat an earlier observation, an abstract scheme knows only $\underline{x}$ and y. However, if we know that $S(\underline{x}, \hat{\underline{\theta}}) = \hat{\underline{\theta}}$ for any parameterization $\hat{\underline{\theta}}$, then $(\underline{x}, y) \in P$, where S is a strategy for P, so that in such a case we must have $Q(\underline{x}, y) = 0_{root(V)}$. Otherwise, we just pass the information $(\underline{x}, y)$ down to the children of the root node.

$\mathbf{A}^S$ is an indexed family of functions $a^S_{v,u}$, $v \in V$, $u \in {}^\bullet v$, given by

$$a^S_{root(V),u}(0_{root(V)}, \hat{y}, \hat{\theta}) = 0_u \tag{64}$$

$$a^S_{root(V),u}((\underline{x}, y), \hat{y}, \hat{\theta}) = \underline{a}_u(\underline{\theta}) \tag{65}$$

$$a^S_{v,u}(0_v, \hat{y}, \hat{\theta}) = 0_u \tag{66}$$

$$a^S_{v,u}(\underline{a}, \hat{y}, \hat{\theta}) = s(\underline{a}, \hat{\theta}, \hat{y}) \tag{67}$$

where $\underline{\theta}_{root(V)} = \hat{\theta} \wedge \underline{\sigma}(x, \underline{\theta})_{root(V)} = \hat{y}$.

$\mathbf{J}^S$ is an indexed family of functions j^S_v, $v \in V$, given by

$$j^S_{root(V)}(0_{root(V)}, \hat{y}, \hat{\theta}) = \hat{\theta} \tag{68}$$

$$j^S_{root(V)}((\underline{x}, y), \hat{y}, \hat{\theta}) = \begin{cases} S(\underline{x}, \underline{\theta})_{root(V)} \text{ if } \hat{y} = y \\ \qquad \wedge \underline{\theta}_{root(V)} = \hat{\theta} \\ \qquad \wedge S(\underline{x}, \underline{\theta})_{root(V)} \neq \hat{\theta} \\ \tau(\hat{\theta}) \text{ otherwise} \end{cases} \tag{69}$$

$$j^S_v(0_v, \hat{y}, \hat{\theta}) = \hat{\theta} \tag{70}$$

$$j^S_v(\underline{a}, \hat{y}, \hat{\theta}) = \begin{cases} S(\underline{x}, \underline{\theta})_v \text{ if } s(\underline{a}, \hat{y}, \hat{\theta}) \leftrightarrow \underline{\theta} \\ \hat{\theta} \text{ otherwise} \end{cases} \tag{71}$$

where, if $\underline{a} \in A_v$, then

$$\underline{a} \leftrightarrow \underline{\theta} \Leftrightarrow \underline{a}_v(\underline{\theta}) = \underline{a} \tag{72}$$

Proposition 6: B^S is an abstract scheme.

Proof. The functions j_v^S are well defined by Lemma 2. It remains to establish (26) and the strictness condition (27).

$j_v^S(0_v, \hat{y}, \hat{\theta}) = \hat{\theta}$ and $a_{v,u}^S(0_v, \hat{y}, \hat{\theta}) = 0_u$ holds by (64), (66), (68) and (70). It remains to be shown that $j_v^S((\underline{x}, y), \hat{y}, \hat{\theta}) \neq \hat{\theta}$, which gives (27).

Suppose than that $j_{root(V)}^S((\underline{x}, y), \hat{y}, \hat{\theta}) = \hat{\theta}$, then $j_{root(V)}^S((\alpha, \beta, \underline{x}), \hat{y}, \hat{\theta}) \neq \tau_{root(V)}(\hat{\theta})$, so that $\hat{\theta} = j_{root(V)}^S((\underline{x}, y), \hat{y}, \hat{\theta}) = S(\underline{x}, \underline{\theta})_{root(V)} \neq \hat{\theta}$, by (69), a contradiction. □

Theorem 2. Suppose that S is a strategy, then S is regular if and only if there exists an abstract scheme B such that $S = S_B$. Furthermore, if S is a strategy with respect to a criterion P, then $P_{B^S} = P$.

Proof. If $S = S_B$ for some abstract scheme B, then S is regular , by Proposition 5. Conversely, suppose that S is regular. B^S is an abstract scheme by Proposition 6. We shall prove that $S = S_{B^S}$.

Let $(\underline{x}, \underline{\theta}) \in I_{\mathbf{N}} \times \underline{\Theta}_{\mathbf{N}}$. Suppose first that S is a strategy with respect to a criterion P, then by (23),

$$P = \{((\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x})) : \underline{\theta} \in \Theta_{\mathbf{N}} \wedge S(\underline{x}, \underline{\theta}) = \underline{\theta}\}. \tag{73}$$

By Proposition 4, S_{B^S} is a strategy with respect to

$$P_{B^S} = \{((\underline{x}, y) \in I_{\mathbf{N}} \times O_{\mathbf{N}} : Q^S(\underline{x}, y) = 0_{root(V)}\}. \tag{74}$$

But, by (63) and (73),

$$\begin{aligned} Q^S(\underline{x}, y) = 0_{root(V)} &\Leftrightarrow \exists \underline{\theta} \in \Theta_{\mathbf{N}}. S(\underline{x}, \underline{\theta}) = \underline{\theta} \\ &\Leftrightarrow G_{\mathbf{N},\underline{\theta}}(\underline{x}) = y \Leftrightarrow (\underline{x}, y) \in P \end{aligned}, \tag{75}$$

so that $P_{B^S} = P$.

From this, we deduce that

$$\begin{aligned} S(\underline{x}, \underline{\theta}) = \underline{\theta} &\Rightarrow (\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x})) \in P \\ &\Rightarrow (\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x})) \in P_{B^S} \\ &\Rightarrow Q^S(\underline{x}, y) = 0_{root(V)} \end{aligned}. \tag{76}$$

Hence, by (37), $a_{\underline{\rho}(\underline{x},\underline{\theta})(v)} = 0_v$ for all $v \in V$, and so $S_{B^S}(\underline{x}, \underline{\theta}) = \underline{\theta} = S(\underline{x}, \underline{\theta})$.

It remains to be shown that $S_{B^S}(\underline{x}, \underline{\theta}) = S(\underline{x}, \underline{\theta})$ when $S(\underline{x}, \underline{\theta}) \neq \underline{\theta}$.

We first show that $a_{\underline{\rho}(\underline{x},\underline{\theta})(root(V))} = (\underline{x}, G_{\mathbf{N},\underline{\theta}}(x))$. Indeed, if, $a_{\underline{\rho}(\underline{x},\underline{\theta})(root(V))} \neq (\underline{x}, G_{\mathbf{N},\underline{\theta}}(x))$, then $Q^S(\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x})) = 0_{root(V)}$ and hence there exists $\hat{\underline{\theta}} \in \underline{\Theta}_{\mathbf{N}}$ such that $S(\underline{x}, \hat{\underline{\theta}}) = \hat{\theta}$ and $G_{\mathbf{N},\hat{\underline{\theta}}}(\underline{x}) = y = G_{\mathbf{N},\underline{\theta}}(\underline{x})$. By 2) of Definition 5, $S(\underline{x}, \underline{\theta}) = \underline{\theta}$, a contradiction.

Let $\underline{\rho}_0 := \underline{\rho}(\underline{x}, \underline{\theta})$ and $v_1 \cdots v_n \in TS(V)$ and suppose that $\underline{\rho}_0 \rightarrow^{v_1} \underline{\rho}_1 \cdots \rightarrow^{v_n} \underline{\rho}_n$. We shall argue by induction that for all $1 \leq i < n$

$$\theta_{\underline{\rho}_i(v_j)} = S(\underline{x}, \underline{\theta})_{v_j},\ 1 \leq j \leq i \tag{77}$$

$$a_{\underline{\rho}_i(v_j)} = \underline{a}_{v_j}(\underline{\theta}),\ 1 < j \leq i+1 \tag{78}$$

For the base case,

$$\begin{aligned} \theta_{\underline{\rho}_1(v_1)} &= j_{v_1}(a_{\underline{\rho}_1(v_1)}, \underline{\sigma}(\underline{x}, \underline{\theta})_{v_1}, \underline{\theta}_{v_1}) \\ &= j_{v_1}((\underline{x}, G_{\mathbf{N},\underline{\theta}}(x)), G_{\mathbf{N},\underline{\theta}}(x), \underline{\theta}_{v_1}) = S(\underline{x}, \underline{\theta})_{v_1} \end{aligned} \tag{79}$$

since $S(\underline{x}, \underline{\theta}) \neq \underline{\theta}$ and hence $S(\underline{x}, \underline{\theta})_{v_1} \neq \underline{\theta}_{v_1}$, by 1) of Definition 5, while by (29), (65) and (61)

$$\begin{aligned} a_{\underline{\rho}_1(v_2)} &= a_{v_1,v_2}((\underline{x}, G_{\mathbf{N},\underline{\theta}}(\underline{x})), G_{\mathbf{N},\underline{\theta}}(\underline{x}), \underline{\theta}_{v_1}) \\ &= \underline{a}_{v_2}(\underline{\theta}'), \text{ where } \underline{\theta}'_{v_1} = \underline{\theta}_{v_1} \wedge G_{\mathbf{N},\underline{\theta}'}(\underline{x}) = G_{\mathbf{N},\underline{\theta}}(\underline{x}) \\ &= \underline{a}_{v_2}(\underline{\theta}) \end{aligned} \tag{80}$$

Finally, suppose that the induction hypothesis holds for $1 \leq r < n$; we show that it holds for $r+1$. We have, by (71) and induction

$$\theta_{\underline{\rho}_{r+1}(v_j)} = \theta_{\underline{\rho}(v_j)} = S(\underline{x}, \underline{\theta})_{v_j} \text{ if } j \leq r \tag{81}$$

$$\begin{aligned} \theta_{\underline{\rho}_{r+1}(v_{r+1})} &= j_{v_{r+1}}(a_{\underline{\rho}_{r+1}(v_{r+1})}, \underline{\sigma}(\underline{x}, \underline{\theta})_{v_j}, \underline{\theta}_{v_j}) \\ &= j_{v_{r+1}}(\underline{a}_{v_{r+1}}(\underline{\theta}), \underline{\sigma}(\underline{x}, \underline{\theta})_{v_j}, \underline{\theta}_{v_j}) \\ &= S(\underline{x}, \underline{\theta})_{v_j} \end{aligned} \tag{82}$$

as $s(\underline{a}_{v_{r+1}}(\underline{\theta}), \underline{\sigma}(\underline{x}, \underline{\theta})_{v_j}, \underline{\theta}_{v_j}) \leftrightarrow \underline{\theta}$, by (61). Similarly

$$a_{\underline{\rho}_{r+1}(v_j)} = a_{\underline{\rho}(v_j)} = \underline{a}_{v_j}(\underline{\theta}) \text{ if } j \leq r+1 \tag{83}$$

and by the property of topological sorts $v_{r+2} \in^{\bullet} v_k$ for some $k \leq r+1$ and so, using induction (61) and (66)

$$\begin{aligned} a_{\underline{\rho}_{r+1}(v_{r+2})} &= a_{v_k,v_{r+2}}(a_{\underline{\rho}_k(v_{r+2})}, \underline{\sigma}(\underline{x}, \underline{\theta})_{v_k}, \underline{\theta}_{v_j}) \\ &= a_{v_k,v_{r+2}}(\underline{a}_{v_k}(\underline{\theta}), \underline{\sigma}(\underline{x}, \underline{\theta})_{v_k}, \underline{\theta}_{v_k}) \\ &= \underline{a}_{v_{r+2}}(\underline{\theta}) \end{aligned} \tag{84}$$

completing the proof. □

4.4. Discussion

The significance of this result is as follows. If it is possible to express *precisely* the aims of training a multi-net, that is, as an extension to a predicate, then if training is possible at all, it may be achieved somehow by the abstract scheme we describe.

Here then, we have described a generic feedforward multi-net system in order to consider how a backward

training pass of adjustments can be used to modify the parameters of the system corresponding to some convergence predicate. We have considered, without loss of generality, a system in which there are two components (the hidden layer and the output layer), with the leaf nodes feeding sequentially the root node. Because of the generality of this situation, we could consider a similar system in which there are two or more leaf nodes feeding the root. This corresponds to the class of MLPs that consist of partial connections between the hidden and output layers. Furthermore, since we have not constrained the operation of the root node in terms of how it combines the outputs of the leaves, this may be achieved simply via a weighted combination without activation, or indeed weight adjustment during learning. Such a combination also describes a simple ensemble. Further suitable definitions of the functions associated with each node can then describe more complex combination techniques, such as the majority voting scheme, or the competitive combination in ME. Indeed, because we have abstracted the parameters and functions, we are not constrained in how the networks are combined, provided we can describe them using the framework.

A similar exercise of comparing theoretically combined systems was considered by Brown [7] when he proposed the linkage between NC learning [34], Dyn-Co [24] and ME [27]. Whilst this only considered a related set of systems, the benefit of such an approach is apparent: we can consider a larger class of systems abstractly and explore their properties collectively. In the first instance we have provided one such analysis by defining an abstract supervised learning algorithm for the generic class of multi-net systems. By providing the algorithm, we have exploited the mathematical framework to give a constructive proof of the existence of such a convergent scheme.

Whilst this can help to unify theoretical notions of neural systems, for example our demonstration that there exists a supervised training scheme for this general class of feedforward multi-net systems, we have not made this concrete in any way. The proof of existence is powerful, but consequently limited. To be of use, making such an algorithm concrete is essential, but obviously relies upon the ability of defining appropriate functions and parameters, something which is not trivial in itself.

5. Conclusion

In this paper we have provided a formal framework in which the general class of multi-net systems can be described and rigorously analyzed. Furthermore, we have proven that, given an appropriately constructed partially ordered set, that there exists a learning algorithm that can be used to train the system to a given criterion, although we do not know what this algorithm might be.

We feel that a key contribution of this paper is that it takes a formal, abstract view of the area; abstract in that no reference is made in the model to numbers. By doing this we have made a start at unifying the different types of multi-net system in a way that can be used to infer properties of the whole from the component properties, together with other, more specific theories. Of course, in practical applications we need to make this concrete, but we believe that a considerable amount of neural network theory can be elucidated in its absence. Essentially, we are offering a different, and we believe novel, perspective on the problem area. This poses an interesting *mathematical* problem. Given an abstract multi-net system and criterion, is it possible to encode the various parameters and functions so that the latter are computable, that is recursive, and that the resulting system is isomorphic (does exactly the same thing as the abstract system)? A positive result would enhance the consequences of Theorem 2.

The next stages of this research are to consider what implications this has on existing multi-net architectures, and in particular whether this helps us to understand better multiple classifier systems, as well as exploring what properties of individual system components can be used to inform us about properties of the system as a whole. In the first instance this means looking at the properties of the combined system in order to systematically break them down to their component parts (sub-multi-nets), so that these can then be put back together to infer properties of the whole once again. This will allow us to consider the impact of each component on the overall system, something that is important in understanding and quantifying the notion of diversity in ensemble classifiers, but which requires relating the abstract system to the concrete. For example, by being able to relate component performance to system performance, it may be possible to determine the maximum capability of a given system and compare that with other configurations.

Acknowledgments

We would like to thank David Pitt for suggesting that we should work together, marrying theoretical computer science to multi-net systems – not as far apart as we first thought. We would also like to thank Gavin Brown for providing comments on an early draft of the paper, and the two anonymous reviewers for their comments and helpful suggestions.

References

[1] S.Abramsky, D.M.Gabbay and T.S.E.Maibaum, Handbook of Logic in Computer Science, Volume 1. Background: Mathematical Structures (Clarendon Press, Oxford, UK, 1992).

[2] S.-I.Amari, Information Geometry of the EM and em Algorithms for Neural Networks, Neural Networks 8 (9) (1995) 1379-1408.

[3] J.L.Armony, D.Servan-Schreiber, J.D.Cohen and J.E.LeDoux, Computational Modeling of Emotion: Explorations Through the Anatomy and Physiology of Fear Conditioning, Trends in Cognitive Sciences 1 (1) (1997) 28-34.

[4] C.M.Bishop, Neural Networks for Pattern Recognition (Clarendon Press, Oxford, UK, 1995).

[5] L.Bottou and P.Gallinari, A Framework for the Cooperation of Learning Algorithms, in: R.P.Lippmann, J.E.Moody, and D.S.Touretzky, ed., Advances in Neural Information Processing Systems (1991) 781-788.
[6] L.Breiman, Bagging Predictors, Machine Learning 24 (2) (1996) 123-140.
[7] G.Brown, Diversity in Neural Network Ensembles, Unpublished doctoral thesis, University of Birmingham, Birmingham, UK, 2004.
[8] G.Brown, J.L.Wyatt, R.Harris and X.Yao, Diversity Creation Methods: A Survey and Categorisation, Information Fusion 6 (1) (2005) 5-20.
[9] G.Brown, J.L.Wyatt and P.Tino, Managing Diversity in Regression Ensembles, Journal of Machine Learning Research 6 (2005) 1621-1650.
[10] J.L.Buessler, J.P.Urban and J.Gresser, Additive Composition of Supervised Self-organizing Maps, Neural Processing Letters 15 (1) (2002) 9-20.
[11] S.Cameron, S.Grossberg and F.H.Guenther, A Self-organizing Neural Network Architecture for Navigation Using Optic Flow, Neural Computation 10 (2) (1998) 313-352.
[12] M.C.Casey, Integrated Learning in Multi-net Systems, Unpublished doctoral thesis, University of Surrey, Guildford, UK, 2004.
[13] M.C.Casey and K.Ahmad, In-situ Learning in Multi-net Systems, in: Z.R.Yang, R.Everson, and H.Yin, ed., Proceedings of the 5th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL 2004), Lecture Notes in Computer Science 3177 (Springer-Verlag, Heidelberg, 2004) 752-757.
[14] M.C.Casey and K.Ahmad, A Competitive Neural Model of Small Number Detection, Neural Networks 19 (10) (2006) 1475-1489.
[15] M.N.Dailey and G.W.Cottrell, Organization of Face and Object Recognition in Modular Neural Network Models, Neural Networks 12 (7-8) (1999) 1053-1073.
[16] B.A.Davey and H.A.Priestley, Introduction to Lattices and Order (Cambridge University Press, Cambridge, UK, 1990).
[17] S.Dehaene and J.P.Changeux, Development of Elementary Numerical Abilities: A Neuronal Model, Journal of Cognitive Neuroscience 5 (4) (1993) 390-407.
[18] Y.Freund and R.E.Schapire, Experiments with a New Boosting Algorithm, in: Machine Learning: Proceedings of the 13th International Conference (Morgan Kaufmann, 1996) 148-156.
[19] G.Fumera and F.Roli, A Theoretical and Experimental Analysis of Linear Combiners for Multiple Classifier Systems, IEEE Transations on Pattern Analysis and Machine Intelligence 27 (6) (2005) 942-956.
[20] M.S.Gazzaniga, Organization of the Human Brain, Science 245 (1989) 947-952.
[21] G.Giacinto and F.Roli, Dynamic Classifier Selection Based on Multiple Classifier Behaviour, Pattern Recognition 34 (9) (2001) 1879-1881.
[22] G.Giacinto and F.Roli, A Theoretical Framework for Dynamic Classifier Selection, in: Proceedings of the 15th International Conference on Pattern Recognition (2000) 8-11.
[23] S.Grossberg and D.V.Repin, A Neural Model of How the Brain Represents and Compares Multi-digit Numbers: Spatial and Categorical Processes, Neural Networks 16 (8) (2003) 1107-1140.
[24] J.V.Hansen, Combining Predictors: Meta Machine Learning Methods and Bias/Variance & Ambiguity Decompositions, University of Aarhus, Aarhus, Denmark, 2000.
[25] D.O.Hebb, The Organization of Behavior: A Neuropsychological Theory (John Wiley & Sons, New York, 1949).
[26] R.A.Jacobs, M.I.Jordan and A.G.Barto, Task Decomposition through Competition in a Modular Connectionist Architecture: The What and Where Vision Tasks, Cognitive Science 15 (1991) 219-250.
[27] R.A.Jacobs, M.I.Jordan, S.J.Nowlan and G.E.Hinton, Adaptive Mixtures of Local Experts, Neural Computation 3 (1) (1991) 79-87.
[28] M.I.Jordan and R.A.Jacobs, Hierarchical Mixtures of Experts and the EM Algorithm, Neural Computation 6 (2) (1994) 181-214.
[29] M.I.Jordan and L.Xu, Convergence Results for the EM Approach to Mixtures of Experts Architectures, Neural Networks 8 (1995) 1409-1431.
[30] R.M.Keller, Formal Verification of Parallel Programs, Communications of the ACM 19 (7) (1976) 371-384.
[31] J.Kittler, M.Hatef, R.P.W.Duin and J.Matas, On Combining Classifiers, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (3) (1998) 226-239.
[32] A.Krogh and J.Vedelsby, Neural Network Ensembles, Cross Validation, and Active Learning, in: G.Tesauro, D.S.Touretzky, and T.K.Leen, ed., Advances in Neural Information Processing Systems (1995) 231-238.
[33] L.I.Kuncheva and C.J.Whitaker, Measures of Diversity in Classifier Ensembles, Machine Learning 51 (2) (2003) 181-207.
[34] Y.Liu and X.Yao, Ensemble Learning via Negative Correlation, Neural Networks 12 (10) (1999) 1399-1404.
[35] B.Lu and M.Ito, Task Decomposition and Module Combination Based on Class Relations: A Modular Neural Network for Pattern Classification, IEEE Transactions on Neural Networks 10 (5) (1999) 1244-1256.
[36] J.Ma, L.Xu and M.I.Jordan, Asymptotic Convergence Rate of the EM Algorithm for Gaussian Mixtures, Neural Computation 12 (12) (2000) 2881-2908.
[37] R.Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Science 92 (Springer-Verlag, Heidelberg, 1980).
[38] D.Partridge and N.Griffith, Multiple Classifier Systems: Software Engineered, Automatically Modular Leading to a Taxonomic Overview, Pattern Analysis and Applications 5 (2) (2002) 180-188.
[39] M.P.Perrone, Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization, Unpublished doctoral thesis, Brown University, Providence, RI, 1993.
[40] D.E.Rumelhart, G.E.Hinton and R.J.Williams, Learning Internal Representations by Error Propagation, in: D.E.Rumelhart and J.L.McClelland, ed., Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations (MIT Press, Cambridge, MA., 1986) 318-362.
[41] D.E.Rumelhart and D.Zipser, Feature Discovery by Competitive Learning, in: D.E.Rumelhart and J.L.McClelland, ed., Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations (MIT Press, Cambridge, MA., 1986) 151-193.
[42] A.J.C.Sharkey, Types of Multinet System, in: F.Roli and J.Kittler, ed., Proceedings of the Third International Workshop on Multiple Classifier Systems (MCS 2002) (Springer-Verlag, Berlin, Heidelberg, New York, 2002) 108-117.
[43] A.J.C.Sharkey, Multi-Net Systems, in: A.J.C.Sharkey, ed., Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems (Springer-Verlag, London, 1999) 1-30.
[44] K.Tumer and J.Ghosh, Analysis of Decision Boundaries in Linearly Combined Neural Classifiers, Pattern Recognition 29 (2) (1996) 341-348.
[45] N.Ueda and R.Nakano, Generalization Error of Ensemble Estimators, in: Proceedings of the IEEE International Conference on Neural Networks (1996) 90-95.
[46] N.M.Wanas, L.Hodge and M.S.Kamel, Adaptive Training Algorithm for an Ensemble of Networks, in: Proceedings of the 2001 International Joint Conference on Neural Networks (IJCNN'01) (IEEE Computer Society Press, Los Alamitos, CA., 2001) 2590-2595.
[47] L.Xu and M.I.Jordan, On Convergence Properties of the EM Algorithm for Gaussian Mixtures, Neural Computation 8 (1) (1996) 129-151.

Figure Captions

Fig. 1. a) an ensemble system where the output of three components (*u*, *v*, and *w*) are combined, say, using a weighted average; b) the Hasse diagram for the equivalent system (V_{ens}), with the addition of a root node (*t*) in place of the weighted summation; c) a HME system consisting of two levels of experts (*u*, *x*, *y*), combined by two gating networks (*w*, *z*); d) the Hasse diagram for the equivalent system (V_{hme}), with the addition of two nodes (*t*, *v*) in place of the combinations.

Fig. 2. a) an arbitrary two layer MLP; b) the same MLP depicted with each layer as a node; c) the Hasse diagram for the equivalent system (V_{mlp}).

Figure

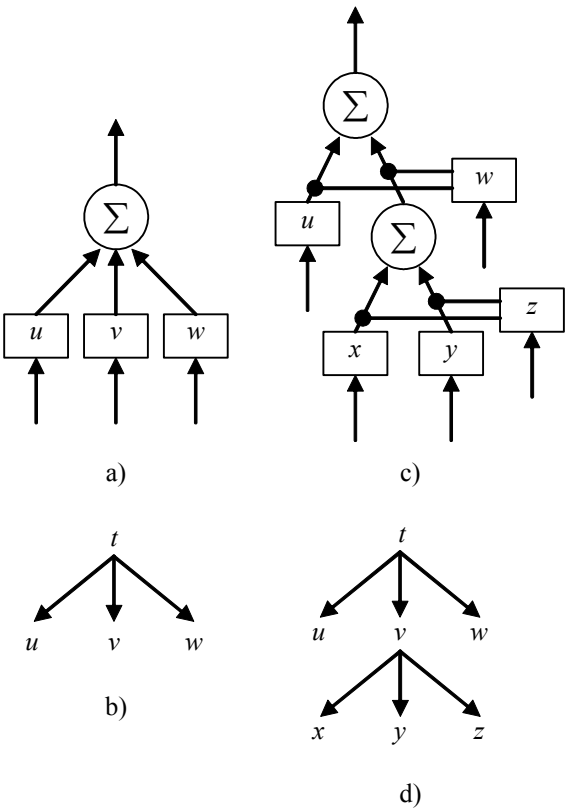

Fig. 1.

Figure

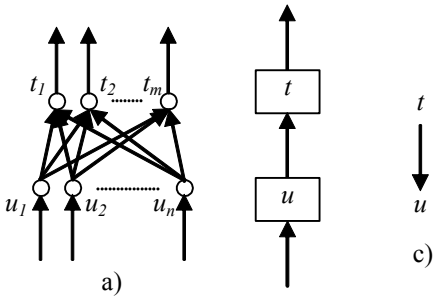

Fig. 2.

Biosketches

Mike Shields received his BA in 1971 from Warwick in pure mathematics and his PhD from Leeds in 1975 in computer science. From 1990 to 2005 he worked as a lecturer, later reader in the Department of Computing at the University of Surrey, becoming a visiting reader in 2006. His research interests are in theoretical computer science, particularly concurrency and formal semantics.

Matthew Casey received his BSc in Mathematics and Computer Science from the University of Kent in 1992. He then spent 10 years in industry, working for Data Sciences, IBM and Anite Telecoms before studying part-time for his PhD from the University of Surrey, awarded in 2004. In 2002 he was appointed a lecturer in the Department of Computing. His research interests are in the theory and application of multiple neural networks, especially for computational neuroscience in modelling vision and multisensory processing.

*** Photo of the author(s)**

Click here to download Photo of the author(s): 09-07-02 Shields Casey Photographs.doc

Photographs

Mike Shields

Matthew Casey