Formal Neural Network Specification and its Implications on Standardization

Georg Dorffner, Herbert Wiklicky, Erich Prem

Austrian Research Institue for Artificial Intelligence
Schottengasse 3, A-1010 Vienna, Austria
georg@ai.univie.ac.at

Abstract

This paper introduces a formal framework for describing and specifying neural networks and discusses several important issues with implications for neural network standardization. In particular, a neural network definition and two tools for graphical description and formal specification are introduced. Issues such as the theoretical impossibility of canonical description, or the need for complete specification (including global algorithms) are discussed. Several examples, making use of the developed tools, illustrate these discussions. In sumamry, this paper aims at contributing to the important endeavour of neural network standardization both practically and theoretically.

1 Introduction

The goal of this paper is to discuss several issues concerning the formal specification and standardization of neural networks and to set a framework for it. As such it is an outcome of the project "NEUFODI - Neural Networks for Forecasting and Diagnosis Applications" which has been part of the European ESPRIT[1] research programme. The baseline behind the definition of that project was a great dissatisfaction with the lack of systematic catalogues or comparisons between the variety of neural network types introduced in literature. Especially for the practitioner it was virtually impossible to take an arbitrary network descritpion and to determine what it has in common with, or what it distinguishes from any other arbitrary network. This is furthermore reflected in the non-systematic terminology one can find throughout

[1]ESPRIT stands for "European Strategic Programme for Research in Information Technology" and is sponsored by the Commission of the European Community

literature. Among others one finds *backpropagation networks*, *Hopfield networks*, *selforganizing feature maps*, and *radial basis function networks*. The first is named after a learning rule, the second after its author, the third after what it achieves (and how), the fourth after what it contains. From reading the original publications there is no easy and straight-forward way to effectively compare these four network paradigms. This is even true for experienced practitioners, let alone for the newcomer.

In this context, the goal of a major workpackage in NEUFODI was to define a framework for systematic and unambiguous description of neural networks so as to permit an effective comparison between different models and also to identify routes for future extensions. As a second goal, unambiguous specification of neural networks, in particular of novel network types, was aimed at. Two description and specification tools have been developed as the outcome of that workpackage: NGraph – a graphical description tool –, and NSpec – a formal specification language. During the design of this framework several important issues with consequences for standardization of neural network description and terminology have been identified:

- Non-ambiguous and canonical description of all neural networks is impossible. Therefore, to permit unequivocal comparison of network models, one certain point of view or description aspect has to be chosen out of a large number of possible ones. By specifying a translation from one description to another, results from comparison can in many cases be transfered from one framework into another.

- Specification of neural network paradigms involves more than just architecture and learning rules. First of all, neural networks have to be viewed as being embedded into the environment it is performing in. For instance, categories like 'supervised learning' can only be explained when taking the type of feedback from the environment into account. Secondly, every neural network has a process associated with it specifying details such as the order in which layers are updated, or at what time teaching input is presented. Many network descriptions in current literature lack the description of these processes thus presenting the described networks in a not entirely reproducible way.

- When speaking about similar or different network *paradigms*, *models*, or *types* one needs to carefully define what the basis of distinction is (e.g. is backpropagation with momentum term in an MLP a new network type or a variation of standard backprop? What about selforganizing feature maps vs. LVQ?). We suggest a hierarchical view from the most general framework to the actual implementation.

With these issues in mind, the description and specification framework was defined from the most general view. It provides the basic elements for

standardized description of neural networks, the details of which have to be chosen in each case. For the same reason, no sufficient but only necessary definitions are given, since at the current state of research it appears premature to exclude any single formalism describable in the framework from being a neural network. In other words, the framework was designed to include most of the neural networks known today, but not to explicitly exclude any model or paradigm (such as, perhaps, cellular automata).

This paper is divided into three parts. First we introduce the definition of a neural network and the developed description framework, including the two tools NGraph and NSpec. Then we discuss the issue of unambiguous comparison of different network types and the consequences for standardization (including an example). Finally, we discuss the issue of specification of novel network types, especially with respect to issues such as patents or copyrights.

2 A Framework for Neural Networks

In this section we describe the formal framework for neural networks developed within NEUFODI. As mentioned above, the definition of a neural network was designed to be wide enough to include most existing network types, but not necessarily to explicitly exclude anything usually not called a 'neural network.'

2.1 A tripartite model

The main idea behind the general framework for describing neural networks is that they cannot be viewed in isolation, but only as being embedded in the environment they are performing in. Therefore the approach for the description of neural network paradigms covers the following three sub-systems or entities (see [Wiklicky *et al.*, 1992a]):

- the **Environment** in which the neural network operates.
- the **I/O-Component** including pre- and post-processing.
- the **Core**, that is the neural network itself.

We introduce the environment and the I/O component of a neural network as separate entities which are relevant for a complete description and classification of any connectionist system. These three basic entities are conceptually integrated by relating their features and characteristics through *heuristics* or *analytical results* concerning the overall performance of the whole system.

Among previous attempts to develop a framework for neural networks (e.g. [Fiesler and Caulfield, 1992], [Bottou and Gallinari, 1990], [Bottou and Gallinari, 1992] [Golden, 1988]) most definitions of neural networks identify the complete system with what here is called the core of a neural network. Therefore these approaches exclude the influence and characteristics of the two other entities. It is only in [Rumelhart and McClelland, 1986] where we can find a somehow similar approach. There a distinction between the neural network, i.e. the core, and the environment in which it is operating, can be found. We also decided to investigate the I/O component as a separate entity, since pre- and post-processing techniques, or in their most basic form the used coding techniques, play an important role for the overall performance of a neural network system.

There are several important examples of neural networks or their major aspects where this tripartite framework is necessary for a complete description:

- The distinction whether a neural network learning rule is *supervised* or *unsupervised* can only be made if it can be decided whether the environment supplies the target or not.
- This is even more true for neural networks for control, which can only be described in full (e.g. when and how a reinforcement signal comes in) if the environment is included. This relates to the above statement that a neural network paradigm is only completely specified if the process determining the order of updates and the points in time of pattern presentation is also described (see also section 4).
- The description and analysis of how a neural network explores similarities among the data can only be reliably given if the preprocessing and coding schemes are known.
- In many cases pre- and postprocessing critically determine the size or perhaps the architecture of a neural network.

Details about the environment and I/O component of a neural network can be found elsewhere ([Wiklicky *et al.*, 1992b], to be published). In the following sections we nevertheless concentrate on the core aspects of neral networks, unless stated otherwise.

2.2 A Conceptual Framework for the Core Component

The following should be seen as some kind of "working definition" of a neural network core (or henceforth neural network, for short) suitable for our

purposes. Compared to the models for a neural network cited above, our approach is trying to find a way in between an implementation oriented – and probably too vague –, and a theoretical – but perhaps too restricted – conceptual model. Thus, in contrast to some more biologically inspired approaches (cf. [Marcadé *et al.*, 1990]) or purely formal ones like the ones mentioned above we tried to define neural networks in an "open" way.

> *The basic structure elements of a neural network core are* **modules** *(***systems***) which interact through* **interfaces** *by exchanging informations via* **signals** *or* **events**.
>
> *These modules are attributed with* **data structures** *and* **processes** *(***algorithms***) and are organized on different* **hierarchical levels**.
>
> *The most basic elements which are not aggregates are* **units** *and* **links** *(connections). The hierarchy levels are established through the definition of* **aggregates** *by combining a set of lower level elements. The* **constituents** *of aggregates not necessarily have to be of the same type (i.e. aggregates can be homogenous or inhomogenous).*
>
> *Aggregates, besides having their own attributes, i.e. data and processes, also impose some kind of* **internal structure** *on their plain set of forming elements.*

This "working definition" of a neural network as a type of parallel computing device is purely based on structural criteria. Some important aspects of neural networks like learning, self-organization etc. are not explicitly mentioned, but we will comment on these issues below.

Our definition obviously introduces a clear static hierarchy for the computational elements. But note that such a hierarchical structure is also given in a natural way for processes operating in such a network. The same way static elements (e.g. units) are combined to form larger entities (e.g. layers) processes can be combined into more complex ones. This is done along or through signals. For example: if we have a process operating on one unit which sends a signal to another unit which in turn starts or triggers some process operating on the second unit, then these two processes together form a combined process which is "glued" together by the transmitted signal. As it is known from more general theories of (parallel) processes (e.g. [Milner, 1980] or [Hoare, 1985]) we can also introduce other forms of combination of processes, such as parallel execution. An example for this kind of process combination is the interpretation of the synchronous update of all units in a layer as *one* process step.

In addition to the above working definition, an important aspect of neural networks is that on a global level we should have at least two processes,

formed by combining lower level processes: **encoding** (or learning) and **decoding** (or evaluation). These two processes operate antagonistically resulting in a self-organization of the whole system. While the evaluation process in general is parameterized by a set of variables, usually called "weights," it changes another set of variables, usually called "activations." On one hand the encoding process changes the parameters which code the evaluation process, i.e. the weight variables; how these values are changed during learning, on the other hand is determined or parameterized by the activation values (or by some error values which for a given environment deterministcally depend on the computed activations).

2.2.1 Some more details on the "working definition"

In the following the main components of the neural network working definition are described in more detail. By doing this, we also explain why this approach presented above was chosen.

Modules: Modules as well as the interfaces between them are modeled in a way inspired heavily from techniques originating in object oriented programming [Meyer, 1988]. The elements of a neural network actually *are* objects in the sense of this programming approach. object oriented programming – perhaps in an extended version which is better suited for concurrency or parallelism, such as Actor models, [Agha, 1986] – indeed seems to be the most obvious and natural programming framework for neural networks (see [Derot *et al.*, 1989]). In this respect, we are not proposing a totally new conceptual approach but building on an already commonly used one. The locality of computation, which is explicitly required, for example, in the Hecht-Nielsen definition [Simpson, 1990], although not always purely realizable, gives a simple and natural principle for data encapsulation and information hiding. Each element knows about any other element only by the exchangable signals. The inner structure of such a neural element is kept hidden. In addition, the explicit specification of well-defined interfaces by describing the emitted and received (data) signals also supports those key principles of object oriented programming [Meyer, 1988, pp 18].

Types of Elements: Another aspect of our "working definition" related to object oriented programming is that we introduce neural element types as **abstract data types**. Every instantation of a neural element is of a certain type.

What is very typical for object oriented programming are so called **inheritance systems**. Many types or classes of objects are specified as specializations of other types or classes. This supports reusability of existing elements.

A derived class – in our case, for instance, a variation of an existing unit type – can be specified by declaring new attributes (data, functions, signals). This introduces a second type of hierarchy (besides the concrete *architectural* hierarchy through the establishment of aggregates), namely an *inheritance* hierarchy, through introducing derived element types. This allows a much better structure on the set of all neural elements for classification.

Interfaces: Establishing a connection between modules is done by the specification of the input to each module. This means that the state of one module has an influence on the transition function of another one. Thus the dynamics of a module depends on the dynamics of others. One could also say that one element of a neural network core reacts to a **signal** which it receives from another element.

Thus, an interface is always directly related to a procedural attribute of a neural element (e.g. its transition function). Its function reacts to the attribute values of another element and *changes* the values of the attributes of the neural element it is attached to.

Aggregates: Collections of more basic elements allow a modularization and hierarchical organization of a neural network architecture. On one hand this is of practical use. Aggregates generalize the well known *layer* concept which makes it easier to describe or specify in a single statement the execution of identical (parallel) processes on a whole set of elements.

On the other hand, they are also of theoretical interest as they allow a higher level information hiding for "semi-global" processes which violate the strict locality principle of Hecht-Nielsen [Simpson, 1990]. An example is the computation of the "total square sum error" or the determination of the winner unit in a Winner Take All (WTA) layer.

Besides being a neural element having "private" attributes attached to it and communicating via signals with other (high and low level) neural elements, aggregates have three specific, additional characteristics;

Set: An aggregate is a set of more basic elements. Thus for aggregates operations like union, intersection but also membership etc., are available in concrete implementations.

Size: An aggregate has a certain size, i.e. the number of elements it contains. As this size might vary either in different realizations of a certain paradigm, or dynamically within the lifetime of a concrete architecture, one has to interpret any aggregate type as a **parameterized type**.

Selection: A final aspect of an aggregate is the possibility of selection. This means that mechanisms have to exist as to how to identify a certain

element, e.g. the nearest neighbors of a unit in a paradigm like Kohonen's topological feature maps ([Kohonen, 1982]). We need this for the formulation of semi-global algorithms as well as for specifying the group-wise connection in more complicated cases.

Signals and Events: A signal in general has a source and a target. It can also carry information. Every signal is related to a certain event, i.e. some point in time when it is emitted. On could also distinguish another event, namely when the signal is received. By doing this, delayed (or timed) signals can be introduced.

In any case, signals implement the dynamical structure described above. They have three purposes. First, they allow the exchange of information or data between different elements. Secondly, the timing of signals induces a specific timing of the processes which send or receive signals (for achieving this they do not even have to carry information, therefore we also permit "void" signals, which we just call events). And thirdly, they allow the gluing together of sub-processes which are attached to different elements to create a more global sequential process as described above.

Processes: The framework includes local processes as well as global and semi-global processes built up from more elementary ones. We described such a combination or gluing method already above.

True neural networks should always have at least two global processes: encoding and decoding. We can even extend our structural definition by the following requirement: eural networks, in general, contain at least two processes which influence one another (self-organization).

There exist some prominent "non-learning" distributed systems, i.e. systems with only one of these two processes (e.g. evaluation), which we therefore can separate from neural networks, among these are cellular automata [Goles and Martinez, 1990] or [Toffoli and Margolus, 1987] or sorting networks [Knuth, 1983]. For reasons given above, however, it might not be advisable to strictly exclude them from being neural networks. Note, for instance, that also some mechanisms commonly accepted as neural networks paradigms, such as linear associative memories ([Hecht-Nielsen, 1990]) or Hopfield networks have only one dynamic process, whereas the other one is degenerated and can be realized as a one-step algorithm. Thus we introduce these additional requirements not as inextricable parts of the framework, but as often desired addendums.

Attributes and States: It is important to distinguish between processes on the one hand and procedures or functions on the other. Whereas the former are the actual dynamic entities, the latter only reflect certain algo-

rithmic recipes. The relation between processes and functional or procedural attributes is established by the fact that the execution of a procedure creates a process. When a procedure depends on a certain event or signal its execution has to be postponed. Thus signals control the execution of functions and therefore in particular the dynamical structure of processes.

Concerning the (data) types of element attributes we can make the following general specialization which again exhibits the difference between "true" neural networks and some close relatives: he elements of a neural network only have **numeric** data attributes. This is mainly in contrast to genetic algorithms [Goldberg, 1989] where in most cases the coding of the genome is of type string. In some way this separation is not too satisfying since it leads to deep representational discussions such as whether there really is a difference between a string attribute and a numeric attribute, and which one this is.

Units and Links: A design problem still left open is the following: Shall we introduce separate link objects or not? It is important to note that the *interfaces* as defined above are not necessarily identical with the links in a network (therefore the more neutral term "interface"). Interfaces connect elements possibly including links (see example below).

We could argue in favor of link objects with the help of the above distinction between an encoding and a recall mechanism. Whereas units parameterize the latter process, links would parameterize the former. In this sense weights play a dual role to activations (the one is changed during encoding, the other one during recall) which is better reflected in putting them into different elements. Thus keeping them separate in different neural elements (modeules) appears natural.

As we tried to follow a kind of object oriented design approach we tried to combine data and procedural attributes such that only local procedures, i.e. procedures attached to the same element, are allowed to change the attribute values of a certain element. In addition, we tried to keep such attribute groups as independent and small as possible and prefer a model where one element is "reading" only from a small set of neighbors. Conventional "weighted" links obviously constitute such a group of attributes (the learning function is such a locally operating transition-function).

Finally, one could argue that with symmetric connection, like in Boltzmann machines, it is impossible to decide in an obvious way with which unit some weight-link shall be associated.

Nevertheless, some designers might like to interpret weights as sub-elements of units or neurons [Marcadé *et al.*, 1990]. The general framework proposed here permits both views, at the same time providing for the possibility of mapping the two into each other. This can be done by merging the at-

tributes of a connection with the attributes of its neighboring unit module (see section 3).

2.3 Two description and specification tools

In this section two concrete description tools following the above working definition will be described.

2.3.1 NGraph

A description tool based on the definition given above – a graphical language named NGraph – was developed in the initial phase of NEUFODI. The basic idea is quite simple and can be stated like this:

Take different icons, i.e. graphical entities or objects, which represent different types or classes of neural entities. The interdependence between the neural entities is expressed by the (graphical) relation of the corresponding icons.

By this we get a "sketch" of a neural network, of concrete architectures or of general paradigms (compare section 4). This description technique is very closely related to the blueprints of architects or engineers. There is also some kind of relation to classical software design methods (cf. [Hatley and Pirbhai, 1987] etc.) On the other hand NGraph tries to give some more uniformity to the diagrams already used for describing neural networks.

An important aspect of NGraph is that it reflects the fact that different neural entities can have the same type. This "typing" has several consequences: (1) Elements of an identical type are represented through the same icon. (2) New types can be expressed by extending, varying or attributing the icons of a base type. (3) One can tell by the shape of an icon whether, for example, an element is a unit or a link.

As neural networks commonly are quite "homogenous" parallel algorithms, i.e. only a few different basic types of atomic elements (and aggregates) are used, not too many different icons have to be invented. And as various paradigms differ not in their basic element types but in their "global" structure one can think of a standardization of basic icons which are then used in the description of different paradigms or networks.

Icons for Neural Elements For representing neural elements, i.e. modules and interfaces or connections, we will use only geometrical objects which have an interior:

Neural elements, i.e. modules and interfaces, are represented by icons with an interior. These icons can have an internal segmentation, pattern or color.

The silhouette of icons representing **modules** *has to be made up from curves (splines). Straight lines can be used only as additional borderline or in the inner segmentation.*

The silhouette of icons representing **interfaces** *is made up only from straight lines. Curves (splines) are allowed only in their interior segmentation.*

When a new icon (corresponding to a new type of neural entity) is introduced, which can be done by any user of the description tool (any neural network designer), only a few rules have to obeyed. To make an icon identifiable or unique one can use several "classical" techniques, like: attaching a unique name as a label, or using different border lines (e.g. line patterns, thickness, color etc.), interior pattern or interior color.

Meta-icons and Connections Usually the relevant informations describing the type and attributes of the neural element which is represented by a certain icon should be put in its interior or very close to its borderline.

Quite often there is not enough room to put all relevant information related to some element or icon in its interior or next to it. In these cases one needs a meta-symbol indicating that such information is put somewhere else. For this purpose we use a line-icon (either a straight line or a spline) with a dot on one side.

In some cases it makes a description diagram or sketch more readable if a certain element of a neural network is represented not only by one graphical icon but several. To identify some icons as representing the same neural element or object we use the following icon: A line-icon (either a straight line or a spline) with a dot on each side. An example would be the easy depiction of full intra-layer connections by "unfolding" the layer into a feedforward network with two layers.

A third meta-icon related to the two above is used to sketch a change of topology (or of another feature) of a module. It is a line-icon (either a straight line or a spline) with a dot on one side and an arrow head on the other. The transformation it indicates is such that the situation described near the dot-end of this icon becomes the situation pictured near the arrow end.

Icons which represent neural network elements can be combined to represent a connection between them, i.e. to represent some kind of influence of the state of one object onto the state of another. There a three prototypical cases: explicate connection, connecting by concatenation and connecting by overlap.

To represent a specific type of connection one again can label connection icons by different attributes, or use special kinds of lines (pattern, thickness, color etc.) to represent a certain connection type.

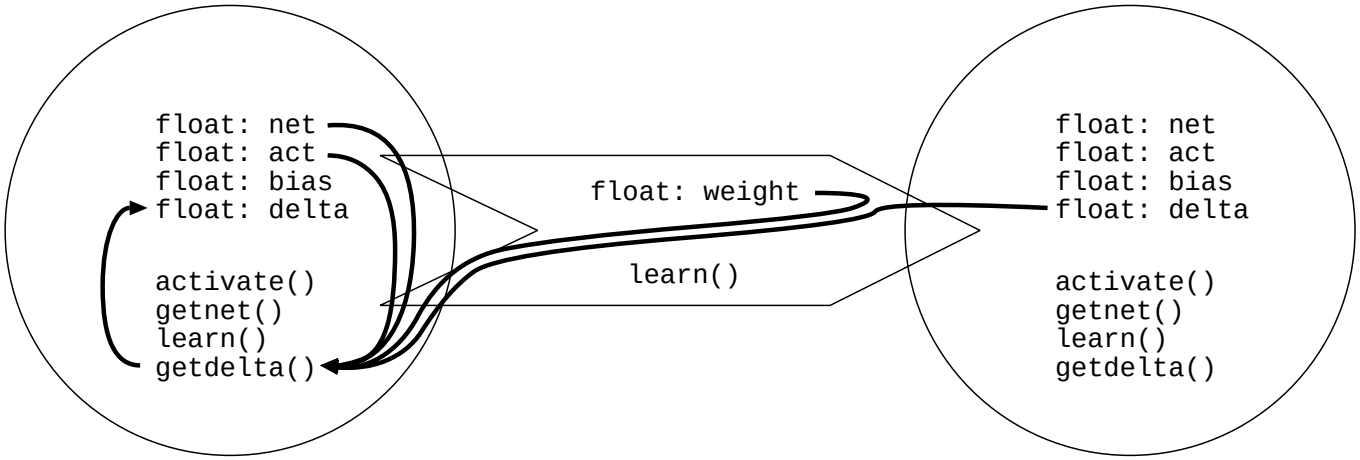

Figure 1: Error Back Propagation

To reflect the dynamical structure of a neural network core one needs to have some tool to describe when and how procedures have to be executed. We represent this "control signal connection" by a dashed line-icon with an arrow head indicating the direction.

When, as a reaction to an event triggered by another object, not only a procedure is executed but also the state of another object is changed, one can say that a data signal was sent. For this we use an full line-icon with an arrow head. This kind signals could be labeled by the involved event and the sent data. Control signals could be interpreted as data signals with empty data, thus the dashed version for representing them.

Attributes and Events Data elements are specified in a pseudo-code notation of the following format: **`data-type : data-name = data-value`**, where the value part often is missing when specifying a data attribute of a neural element. When labeling data connections (information flows) only the name part might be provided if anything else is obvious from the context. Figure 1 shows an example by depicting the attributes of two units and their link in an backpropagation network, as well as their dependencies.

For a procedure we use the following notation: **`proc-name (proc-param) = proc-body`** The function body – normally omitted when specifying a procedure only as a procedural attribute – iswritten in a C-like syntax or using mathematical symbols.

Procedures are triggered either by an implicit or an explicit event. A procedure thus depends on a certain event and itself can trigger other events needed by other units.

We suggest the following notation for events: < **`event-name`** > If a procedure is triggered by some events or if it creates events we describe this in the following way: |**`events`** >**`proc-name`** respectively **`proc-name`**< **`events`**|

where **events** is a sequence of events. If these events are separated by commas (",") this indicates an "and" combination of the corresponding conditions, if a semi-colon is used (";") an "or" operation is indicated.

There are some additional features of the graphical design tool which cannot be explained here due to the limitations in space (for those details see [Wiklicky, 1992]). Fig. 2 in section 4 shows an example depicting a backpropagation network, showing the use of events and signals, as well as of meta-icons.

2.3.2 NSpec

Graphical description of neural networks, as exercised with the help of NGraph can be extremely helpful and easy to understand. However, many important aspects cannot be specified by graphical means to a satisfying degree. For instance, the depiction of algorithms (e.g. learning rules) as well dependencies along the inheritance hierarchy, is extremely difficult if not impossible. Therefore, in a subsequent phase of NEUFODI, a formal specification language for neural networks, called NSpec, was developed. While NGraph was envisioned mainly for description of neural networks in an easy-to-understand way, NSpec was designed to be suitable for formal detailed specification.

One of the basic principles of NSpec is object oriented design (see also above). To stay as close as possible to the basic conceptual definition of a neural network, the following basic entities were introduced:

- **Data Attributes**: Local Variables, Signals.
- **Functional Attributes**: Functions and Procedures, Processes.
- **Elements**: Attributes, Structure of Aggregates.
- **Interfaces** between elements.

Further specific features of NSpec are: hierarchical topological structuring of elements, signals and events to implement concurrency, and assertion techniques to specify abstract requirements. Basically, a notation very reminiscent of C++ ([Stroustrup, 1987]) was chosen. This was done to make NSpec more readable and acceptable, since a large number of neural network researchers are proficient in C or C++. It should be emphasized, however, that NSpec is not intended as a programming language but as a formal tool for specification. Some aspects therefore go beyond C++ or any other programming language. For more details we refer to [Wiklicky *et al.*, 1992b] or [Wiklicky, 1992] where a complete syntax definition of NSpec is given.

Basic structure In NSpec we distinguish between atomic elements, e.g. units or links, and aggregates (see above). To describe an atomic element one simply has to specify all its attributes, such as functions and data structures.

Each attribute is defined, together with their types, in a way very similar to variable and procedure definition in C. Moreover, functions and procedures attached to a module are defined like object-oriented methods in C++ with the name `<object-name>::<func-name>`. Inheritance of modules can be expressed by an `is <object-name>` statement. Declaration or definition bodies are embraced by curly brackets.

Signal exchange and concurrency To describe the concurrency aspects of neural networks we use a kind of signal exchange model to express it. Signals are usually attached to neural network elements. From the point of view of a certain element there are two kinds of signals: incoming and outgoing signals. In particular, signals possess a certain (data) type which specifies the possible values which may be transmitted via a particular signal channel. To represent pure control signals (also call events), the data type `void` is also admissible.

Signals in general are declared like simple data variables, in particular it is also possible to define whole sets of signals. For specifying processes we need expressions to indicate the sending and receiving of a signal. For this a similar (reverse) notation is used. Some elements are connected by identifying some incoming and outgoing signals.

- **Incoming signal:** `data-type <signal|`
- **Receive a signal:** `|signal>;`
- **Outgoing signal:** `data-type |signal>`
- **Send a signal:** `<signal|;`

Abstract Specification Most of the constructs of NSpec which are related to conventional (concurrent) programming seem to suffice for describing neural networks. But for a more abstract analysis one is often interested in formulations where a neural network algorithm is not described as a concrete process, but instead only the desired result is stated. For example, we could be confronted with the problem of formalizing a statement like "select the winning unit", without knowing how to realize such a selection process.

The specification statements about the characteristics are formulated according to a syntax which resembles very much predicate calculus in logic (which therefore also is the basis for the semantics of these statements). Specification, in particular, are based on certain predicates describing the characteristics of neural network entities. For example a winner-take-all (WTA) layer might be specified in abstract way, using an `assert` statement, as follows.

```
elem unit is { float act; };
```

```
elem wta_layer is set of unit assert {
        exists T in time :
        exists w in wta_layer :
        forall l in wta_layer :
                { at t <! T : w.act >= l.act } and
                { at t >! T : w.act == 1.0 and l.act == 0.0}
};
```

The last part of the following section makes use of NSpec and thus provides the reader with a much more extensive example of its application in specification.

3 Comparison and Standardization

After having defined an overall description and specification framework for neural networks, we now discuss some important issues concerning the use of such a framework. The first is the comparison between different networks.

For practitioners who apply neural networks to their problems a comparative appraisal of different network types or paradigms is of vital importance. In particular, they need to know

- what distinguishes network type X from network type Y, which would justify a switch from one to the other in case of unsuccessful application of X. For instance, how are radial basis function networks ([D.S. Broomhead, 1988], [Moody and Darken, 1988]) different from multilayer perceptrons such that the former could be employed more successfully than the latter.

- what are the commonalities of network types X and Y so as to permit an aspect of X (e.g. a learning rule) to be applied to Y. For instance, a comparative description should make clear that gradient descent learning (usually applied to multilayer perceptrons) can also be applied to radial basis function networks (see, for instance, [A.J. Robinson, 1988]).

- what dimensions of different neural network types can be combined so as to develop new and more powerful types. For instance, the description should make clear that selforganizing feature maps could be enhanced with hidden layers so as to make the mapping capabilities from the input to the map with respect to similarity constraints more general.

Of course, a standardized description can provide only half of the necessary knowledge to permit these kinds of reasoning. With each dimension of description, ideally theorems or rules would have to be attached as to what

consequence a variation of that dimension would have on practical applications. For instance, to decide whether to apply gradient descent learning to radial basis function networks one needed to know in what cases it would be successful, or what influence the difference between RBFNs and MLPs has on gradient descent (such as the observation that if RBFNs are not initialized properly, gradient descent will not find a solution due to the spatially constrained decision regions of each RBFN hidden unit). Nevertheless, a formalized description will be a necessary prerequisite for the overall goal of arriving at a comparative overview of neural networks.

One major spin-off of the whole endeavor described in this paper should already have come clear from this discussion. A general description framework of neural networks eases the way for novel developments (such as the mentioned RBFNs with gradient descent or feature maps with hidden units), since it frees the field from the categorical thinking in terms of the most popular paradigms (inasmuch, for instance, backpropagation appears to be inextricably tied to multilayer perceptrons for so many).

3.1 The Impossibility of Uniqueness of Description

One major aspect considerably impeding the whole task of description and standardization of neural networks is the fact that the definition of a unique and canonical description of networks with respect to all interesting aspects is impossible. In other words, it is not possible to find a standard of description such that all relevant similarities and differences between networks are uniquely expressed. The reason for this is simple. Every neural network contains algorithms (such as the learning rule, or the process defining the overall order of updates) and the problem of equivalence of algorithms is undecidable. As a consequence, one can always find a description which makes two arbitrary network types more similar to each other than to any other type. In other words, there is no absolute point for any neural network type in the space of possible types.

This issue has already become evident in the above discussion about whether links should be described as separate objects or not. Another illustrative example is the following. Suppose one wanted to decide whether competitive learning (e.g. in [Grossberg, 1976]) is more similar to Hopfield networks or to perceptrons. One important dimension might be whether each network is recurrent or not. One possible description of the competitive layer in CL could be:

```
elem wta_layer
is array of unit {
        get_winner <|> (|);
};
```

```
proc wta_layer::get_winner <|> (|)
is {
        set of int won; // the set of winner indices
        float max;      // the maximum activation so far

        won = wta_layer[1];
        max = wta_layer[1].act;

        for (i = 2; i =< #wta_layer; i++) {
                if (wta_layer[i] >! max) {
                        // a new maximum found
                        won = [wta_layer[i]];
                        max = wta_layer[i].act;
                } else  {
                        // probably at least as good?
                        if (wta_layer[i] == max) {
                                won += i;
                        }
                }
        }

        wta_layer.<loser|;      // all elements loose first
        wta_layer[won].<winner|; // but some can recover
};
```

In this description, a process, called "get-winner" is associated with the layer, which itself is *not* recurrent. Thus, according to this dimension, the network would be more similar to a perceptron than to a Hopfield network. Logical extensions would be networks like counterpropagation (where the competitive layer is the hidden layer of an MLP-type network, [Hecht-Nielsen, 1987]) or CL with hidden units.

On the other hand, an equally justifiable description would be this:

```
elem wta_unit
is unit {
        float <output|;         // current activation signal
        array of float |input>; // activation of others

        ask <|output> (act|);
        answer <input|> (act|act);
};
```

```
proc wta_unit::ask <|output> (act|)
is {
        <output| = act;
};

proc wta_unit::answer <input|> (act|act)
is {
        for (i=1; i =< #input; i++) {
                if (act <! |input[i]>) {
                        // we have lost
                        act = 0.0;
                        return; // exit!
                }
        }
        // we did not lose, so we are among the winners
        act = 1.0;
};
```

with connections defined as follows

```
conn default (wta_unit u, wta_unit v)
is {
        conn u.<output| to v.|input>[];
};
```

Thus, we can define a WTA layer as a set of **wta_unit**s which are fully connected with each other (but not with themselves).

```
conn full (array of unit layer)
is {
        wta_unit u;

        pardo (u in layer) {
                conn u, layer - u by default;
        }
};

elem wta_layer
is set of unit {
        conn self by full;
};
```

In other words, WTA is realized through interconnections via which units can ask other units whether they are more higly activated or not. The main

difference to the above description is that the process leading to WTA is not defined globally for the whole layer but is achieved through interactions between single units via interfaces (connections). Of course another such view (perhaps even more common) is to have "regular" connections for activation spreading with large negative weights so as to achieve high competition (leading to one winner remaining active).

In this description, the layer is clearly recurrent and directly comparable to the one layer in a Hopfield network, which would thus be the more similar of the two. A logical extension could be "soft competitive learning" ([G. Dorffner, 1993b]), where not only the winner remains active, or "mutiple winner CL", where there can be more than one winner.

The question which of the two descriptions is more canonical or the underlying one cannot be answered. Both descriptions have to be equally accepted, the choice as to which is more appropriate depends on the context. Therefore, any comparative description has to be defined with certain presuppositions. In other words, standardized description will need to be defined by deciding upon one out of many possible forms of description, depending on the requirements of a certain background. Thus, standardization can only provide for one possible view of neural networks, which – if agreed upon – can serve as a seed for other views for researchers with different requirements. Novel views can then be introduced by defining how dimensions of the original standard map onto dimensions of the new standard, thus leaving the framework intact and the deviations tractable. Such a mapping, however, need not be bijective (??), in that every network in one view has a direct analogon in the other view.

Coming back to the above example, it could be decided that the standardized view of CL is the second one (depicting it as being recurrent). In this view, CL would share recurrency with the Hopfield network. If, for instance for reasons of implementation, the other view becomes preferable, it can be introduced by defining the mapping

Layer in Framework1: connections always lead to one single active unit

<->

Layer in Framework2: process 'get-winner'

One can see that this mapping works for these two cases, but not necessarily for any generalizations in either framework. For instance, there are competitive layers which cannot be described as winner-take-all. On the other hand, processes might be introduced in framework 2 (e.g. 'Let the five most active units fire maximally') which might not be translatable into a layer with recurrent connections.

3.2 An example: A comparative description of feed-forward networks

In this section we describe a more elaborated example, illustrating the use of the description framework, as well as the issues raised above. We present a possible standard for describing the family of non-recurrent (i.e. feedforward) neural networks, for example, multilayer perceptrons (MLP) and radial basis function networks (RBFN). Only units and links are defined here, the larger structure of the network, very similar in all cases, was omitted for simplicity.

3.2.1 View 1: Propagation rules and activation functions

We begin by describing hidden units:

```
elem ff-unit
{
        float net;
        float act;
        float bias;

        set of float <input|;
        float |output>;

        float activation (float x);

        forward <input|output> (bias|net,act);
        init <insamples|> (|bias);
};

elem bp-unit
is ff-unit
{
        float delta;
        float error;

        set of float <indelta|;
        float |outdelta>

        float derivative (float x);

        backward <indelta|outdelta> (net,bias|error);
        learn <|> (delta,bias|bias);
};
```

This description specifies that each feedforward unit possesses three local variables, namely the net input (net), the activation (act), and a bias. Furthermore, a set of input data and a output value are defined. Finally, one function (activation function) and two processes (forward propagation of ativations, and initialization) are associated with the unit. Introducing the initialization process stresses the fact that the weights of all feedforward networks, not only of radial basis function networks, can be initialized on the basis of training samples ([Smyth, 1992], [G. Dorffner, 1993a]).

From this unit, a specialization **bp-unit** is defined which possesses additional variables, functions and processes to permit error back propagation and gradient descent learning.

The involved processes are of the following simple kind:

```
proc ff-unit::forward <input|output> (bias|net,act)
is {
        net = sum(|input>);
        <output| = act = activation(net,bias);
};

proc bp-unit::backward <indelta|outdelta>
                       (net,bias|error)
is {
        error = sum(|indelta>);
        <outdelta| = delta
                   = error * derivative(net,bias);
};

proc bp-unit::learn <|> (delta,bias|bias)
is {
        bias = eta*delta + bias;
};
```

The involved functions can have different forms. It is only necessary that **derivative** is the first derivative of the activation function **activation**. Only those functions and processes are defined at this point that are common to all feedforward networks. The remaining ones will be defined with particular specializations below.

Output units are a little bit different, since they may also receive a certain **target** signal which determines their **error** directly. This signal is provided by the environment.

```
elem ff-out-unit
        is ff-unit
{
```

```
        float <target|;
        backward <target|outdelta> (net,bias|error);
};

elem bp-out-unit
  is ff-out-unit
{};

proc bp-out-unit::backward <target|outdelta>
                        (net,bias|error)
is {
        error = |target> - act;
        <outdelta| = delta
                   = error * derivative(net+bias);
};
```

In this standard description we introduce links as separate objects. Like above, we distinguish between standard feedforward links, and links with the additional functionalities for gradient descent (backpropagation). The major parts of learning and initialization (specified in detail below) happens in these links, which otherwise serve for propagating and weighting signals between units.

```
elem ff-link
{
        float weight;

        float <input|;
        float |output>;

        forward <input|output> (weight|);
        init <insamples|> (|weight);
};

elem bp-link
        is ff-link
{
        float <indelta|;
        float |outdelta>;

        backward <indelta|outdelta> (weight|);
        learn <input,delta|> (weight|weight);
};
```

```
proc bp-link::learn<input,indelta|> (weight|weight)
is {
        weight = eta*<input|*<indelta| + weight;
};
```

From this general definition of a feedforward neural network, several specializations (or, since some attributes are specified for the first time, *instantiations*) can be derived. The two most well known are multilayer perceptrons with backpropagation, and radial basis function networks:

Multilayer Perceptron (MLP):

```
elem mlp-unit
   is bp-unit
{}

elem mlp-link
   is bp-link
{}

func mlp-unit::activation(float x, float b)
is { return 1/(1+exp(x+b)) }

func mlp-unit::derivation(float x, float b)
is { return (activation(x,b)*(1-activation(x,b)) }

proc mlp-link::forward <input|output> (weight|)
is {
       <output| = weight * |input>;
   }

proc mlp-link::backward <indelta|outdelta> (weight|)
is {
      <outdelta| = weight* |indelta>;
   }
```

Radial Basis Function Network (RBFN):

```
elem rbfn-unit
   is ff-unit
```

```
{}

elem rbfn-link
   is ff-link
{}

func rbfn-unit::activation(float x, float b)
is { return exp (-x*x) }

proc rbfn-link::forward <input|output> (weight|)
is {
      <output| = (weight - |input>) * (weight - |input>);
   }

proc rbfn-link::init <insample|> (|weight)
is {
      weight = |insample>;
   }
```

But as the description strongly suggests, other instantiations would equally be possible – and as it turns out, both of them are viable (see [A.J. Robinson, 1988], [Smyth, 1992], [G. Dorffner, 1993a]):

MLP with initialization:

```
elem mlp-link::init <insample+, insample-|> (|weight)
is {
      weight = |insample+> - |insample->;
   }

elem mlp-unit::init <insamples+, insamples-|> (|bias)
is {
       bias = 0.5 * sum (insamples+[i]*insamples+[i]
                         - insamples-[i]*insamples-[i]);
   }
```

RBFN with gradient descent:

```
elem rbfn-link1
   is bp-link
{}
```

```
func rbfn-link1::derivation (float x)
is { return -2 * x * exp(-x*x) }

proc rbfn-link1::backward <indelta,input|outdelta> (weight|)
is {
       <outdelta| = (weight - |input>) * |indelta>;
   }
```

The first is an extension to **mlp-unit** introducing an initialization procedure in direct analogy to the radial basis function network (in this case, the weights and the bias are set based on a pair of positive and negative data sample, such that the hyperplane forms a Voronoi tesselation of the two points - see [Smyth, 1992]). The second is a variation of **rbfn-unit** including gradient descent learning (see [A.J. Robinson, 1988]). We see from this example, that the chosen view presents MLPs and RBFNs as rather similar, differing mainly through different forward propagation processes and activation functions. This view gives riuse to easy extensions and combinations.

3.2.2 View 2: Basis functions

Feedfoward networks can also be represented from a different point of view, namely by viewing them as approximators based on weighted sums of basis functions ([D.S. Broomhead, 1988]). In this view, there would be no separate link objects, and propagation rule and activation function would be combined into one basis function:

```
elem ff-unit1
{
        float act;
        array of float params;
        float bias;

        set of float <input|;
        float |output>;

        float basis(array of float x, array of float p);

        forward <input|output> (bias|net,act);
  init <insamples|> (|bias);
};

elem rbfn-unit2
is ff-unit1
{}
```

```
proc rbfn-unit2::forward <input|output> (param|act)
is { <output| = act = basis(<input|,params); }

func rbfn-unit2::basis (array of float x, array of float p)
is { return exp (- ||x - p||^2)) }
```

Here, both net input and activation function are combined into one basis function over vectors. In the case of RBFNs its a Gaussian of the Euclidean distance of the two vectors (input values and parameters **params**). MLP units could be defined similary using the sigmoid of the dot product instead. This view now suggests other similarities than the previous ones, such as to ourier series if expressed as a neural network. F From this we see the influence a chosen view on standardization can have.

3.2.3 Summary

In this section we have discussed the importance of formal comparison between neural networks, namely for pointing out essential differences and for easing novel neural network developments. We have also discussed the impossibility of canonical neural network description and the need for choosing one particular view for specifications. An example has illustrated these issues and also has demonstrated the viability of NSpec for specification and comparison.

4 Specification of neural networks

Another field where a standardized framework is of crucial importance is specification of novel neural networks. By this we mean the unambiguous description of a newly defined paradigm or type such as to make results reproducible, and to define the important differences to other previously published ones. This should be done for one of at least two reasons; either for clean scientific conduct in introducing novel results, or for clear decisions concerning copyright or patents.[2] A description framework like the one introduced above can be considered as the basis for specification (hence the name "specification language").

Tow issues are of special importance in this context. One concerns the question as to what constitutes a separate network type or paradigm and

[2] by discussing this issue we do not want to imply that we do not have any moral doubts or reservations about patents of neural network algorithms or architectures, which, in our opinion, can considerably encroach on free scientific exchange of ideas

not just a variation of the same type or paradigm. The second one concerns the complete picture of a neural network, as defined above. To specify a network with all respects means to also include the role of the environment, and processes controling the interaction between layers, and also between the network and the environment. The follwing two sections discuss these issues.

4.1 Paradigms and Concrete Networks

In our context "paradigm" will mean some kind of an abstract model or ideal archetype representing a whole class of neural networks, like for example *Feed Forward Networks* or *Boltzmann Machines*, and not any concrete realization of such a conceptual model.

It is important to realize that the aim of developing a specification of neural networks is very different from designing a programming language for neural networks. Many different approaches for this problem already exist, but they are "merely" well defined tools for building neural network computer programs. Usually the set of expressible systems, meaning the possible networks that can be described by these languages is very limited. Another problem is that they need to be languages for machines and are therefore hard to work with for humans. Although programming languages, together with their compilers, are exact definitions of neural networks they cannot normally express the main aspects of a paradigm. I.e. in developing a network specification method we are not really interested in running the developed nets. We want to describe and compare different networks and paradigms. Therefore, we also need to clarify the relation between the two. Both paradigm and network are expected to be completely formally described. The description of a network must be clearly derivable from the paradigm description.

In the same way as we can think of classes of neural networks, i.e. of neural network core paradigms, we can also think of certain classes of environments. By abstracting from concrete problems or tasks we arrive at relevant problem classes, e.g. from speech data to time sequences of a certain type. The same should also be true—maybe in a somehow modified way—for the I/O component of neural networks. As a result of this consideration we find that for all three aspects of neural networks (see section 2) several distinct levels of *concretization* exist. In our framework for artificial neural networks we distinguish different levels of abstraction in which a smaller or greater number of values of parameters or structure elements have been fixed. For example: in a conceptual model of the core of neural networks we find at least the following levels of abstraction:

0. The **State** of a neural network (e.g. initial state).

1. The implemented **Network** (e.g. a hardware realization of a network).

2. A concrete **Architecture** (e.g. a 4-2-4 feed forward network).

3. An abstract **Paradigm** (e.g. Back-Propagation).

4. A general **Framework** for the core (e.g. the one in section 2).

A similar scheme can be set up for the other basic entities. In a theory of the environment component of neural networks the following levels can be distinguished:

0. A concrete **Pattern** for a neural network (e.g. a digital satellite picture).

1. The **Collection** of patterns actually realized (e.g. time sequence).

2. A concrete **Application** (e.g. consumption forecasting).

3. An abstract **Environment Model** (e.g. Markov chains).

4. A general **Framework** of the environment (e.g. the one in section 2).

It is not a trivial issue to establish such clearly defined and separated levels of conretization. Additional criteria may be used to establish an even finer structure. For example, one could argue that in the case of a framework for the core of neural networks one should have an additional conceptual level between paradigms and architectures which would correspond to the concept of *feed forward networks using back propagation*. For more details on this discussion, see [Prem, 1991].

The important point in these considerations lies in the observation that there are such different levels and that there is indeed a clear conceptual difference between the problem of describing and classifying neural network paradigms and of specifying and maybe also implementing concrete neural network architectures in a concrete computer environment. It is exactly this distinction which is the main difference between the approach we were following in NEUFODI and most other known approaches (like for example in the other neural network related ESPRIT projects Pygmalion or Galatea project), which concentrated in particular on the description of a concrete implementation of a neural network. In this sense, NEUFODI has been more concerned with the exact description and classification of paradigms and general concepts than in developing yet another programming language for neural networks.

For copyright and patent issues it is necessary to develop more abstract mechanisms to decide if something is just a variation of another known mechanism or not. Thus to overcome the wide-spread "Look and Feel" around different paradigms introduced in literature, more abstract specification techniques would be of scientific and commercial interest. NSpec in this respect also intends to serve as a basis for this kind of question.

4.2 Complete specification

Besides the structural definition, algorithmic aspects of neural networks are very important in developing an adequate specificaton technique. As opposed to their classical way of being depicted as a graph-like structure, neural networks cannot be fully accounted for by describing them as static objects. Not only are there relevant dynamic processes during a network's life cycle which can change the overall architecture, but also the algorithmic part of *any* learning rule must be clearly defined so as to enable an implementation of a specified network paradigm. An example for the first case (changing the architecture) is the creation or deletion of new units during learning. In fact, the whole network operation is controlled by algorithms and it sometimes is not clear as to whether a specific part of a paradigm is implemented through its structure or by means of an algorithmic replacement of this structure.

An example is given in fig.2, specifying the dynamics of backpropagation. The complete learning algorithm is not given by specifying the formulas alone, but also by specifying the order in which layers get updated and at which points the formulas are applied. In NGraph this is depicted by an event/signal flow diagram. For instance, the fact that the weights leading to the hidden layer are changed after error back propagation is expressed by the signal `<learn2>` the hidden layer has to wait for before it itself emits the signal `<update>` to the link objects.

4.3 Summary

To summarize, a neural network paradigm must be described by taking the core neural network, the environment and the I/O components all into account. Besides these aspects, different levels of concretization exist which all may independently be worth describing. In addition, a clear specification method must possess the possibility of expressing algorithms in addition to all elements of a neural network. Combining these elements of a neural network yields a complete description to any desired level of detail. The example has also demonstrated the viablity of NGraph to depict complete learning processes.

5 Conclusion

In this paper we have introduced a framework for formal description and specification of neural networks, developed within the European (ESPRIT) project NEUFODI. Two tools, one for graphical description (NGraph), one for formal specification (NSpec) were briefly described, and subsequently used in examples. In the context of this framework, several important issues concerning specification and standardization of neural networks were discussed.

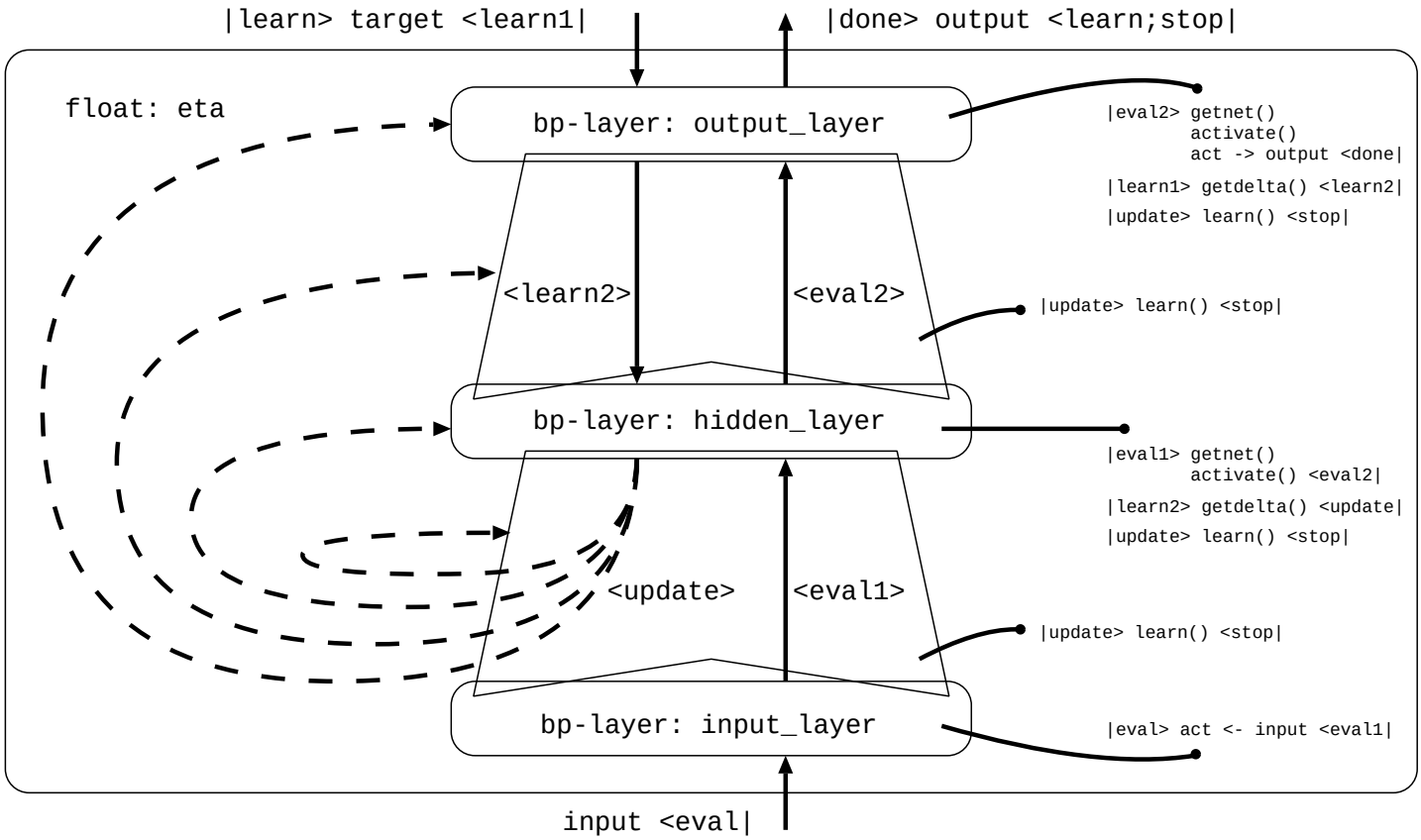

Figure 2: The dynamics of a 3-Layer Perceptron

First, the impossibility of canonical description, such that all similarities between network types are clearly visible, was discussed. A consequence is that for standardization always one out of many possible views have to be chosen. An extensive annotated example, describing and comparing different feedforward networks from two views, was presented. It illustrated the helpfulness of formal comparison for easy new developments.

Secondly, two issues of specification were discussed. The first concerned the clear definition of what a neural network paradigm is, as opposed to a variation of another type. We proposed to apply a conceptual hierarchy, ranging from the most general framework down to an actually implemented network and its state, in solving the problem. The second issue delat with completeness of specification including global algorithms and processes (e.g. the order of layer updates, and the times of pattern presentations), something often overlooked in literature. The general framework introduced earlier, and the the two tools NGraph and NSpec, support the specification of those important procedural aspects of neural networks.

By introducing these issues and describing the approach chosen within NEUFODI we aimed at contributing to the important field of neural network standardization. We stronlgy believe that standardization is badly needed in a field of great terminological confusions and so many "reinventions of the wheel."

Acknowledgment

This work was done as part of the Esprit-II Project No.5433: NEUFODI – Neural Networks for Forecasting and Diagnosis Applications. The partners involved in this project are: Austrian Research Institute for Artificial Intelligence (Vienna), Babbage Institute for Knowledge and Information Technology (Ghent), Elorduy, Sancho y CIA, S.A. (Bilbao), Labein (Bilbao), and Lyonnaise des Eaux (Compiégne). The Austrian contribution is supported by a grant from the Austrian Industrial Research Promotion Fund, Project No. 2/282.

We particularly thank Prof. Robert Trappl for his generous support for the work of the neural network group.

References

[Agha, 1986] Gul A. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems.* MIT Press, Cambridge, Massachusetts – London, England, 1986.

[A.J. Robinson, 1988] F. Fallside A.J. Robinson, M. Niranjan. *Generalisinf the Nodes of the Error Propagation Networks.* Technical Report, Cambridge University, Engineering Dept., 1988.

[Anderson and Rosenfeld, 1988] James A. Anderson and Edward Rosenfeld, editors. *Neurocomputing.* Volume 1: Foundations of Research, MIT Press, Cambridge, Massachusetts – London, England, 1988.

[Bottou and Gallinari, 1990] Léon Bottou and Patrick Gallinari. A framework for the cooperation of learning algorithms. In *Advances in Neural Information Processing Systems 3 [Lippmann* et al.*, 1991]*, pages 781–788, Denver, 1990.

[Bottou and Gallinari, 1992] Léon Bottou and Patrick Gallinari. A unified formalism for neural net training algorithms. In *Proceedings of the International Joint Conference on Neural Networks*, pages IV–7 – IV–12, Baltimore, 1992.

[Derot *et al.*, 1989] Benoit Derot, Philippe Escande, and Catherine Moulinoux. Nacre: A neuron-oriented programming environment. In *Proceedings of Neuro-Nimes '89*, pages 183–200, Nimes, France, 1989.

[D.S. Broomhead, 1988] D. Lowe D.S. Broomhead. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[Fiesler and Caulfield, 1992] Emile Fiesler and H. John Caulfield. *Neural Network Formalism.* Technical Report (posted on **neuroprose**), Institut Dalla Molle d'Intelligence Artificell Perceptive (IDIAP), Martigny, Suisse, 1992.

[G. Dorffner, 1993a] G.Porenta G. Dorffner. On using feedforward neural networks for clinical diagnostic tasks. *submitted for publication*, 1993.

[G. Dorffner, 1993b] T. Schoenauer G. Dorffner. Unsupervised learning of simple speech production based on soft categorization. *Eeckman, Bower (eds.): Computation and Neural Systems 92*, 1993.

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison–Wesley, Reading, Massachusetts, 1989.

[Golden, 1988] R.M. Golden. A unified framework for connectionist systems. *Biological Cybernetics*, 59:109–120, 1988.

[Goles and Martinez, 1990] Eric Goles and Servet Martinez. *Neural and Automata Networks.* Volume 58 of *Mathematics and Its Applications*, Kluwer Academic Publisher, Dordrecht, 1990.

[Grossberg, 1976] Stephen Grossberg. Adaptive pattern classification and universal recording: I. Parallel development and coding of neural feature detectors. *[Anderson and Rosenfeld, 1988]* and *Biological Cybernetics*, 23:121–134, 1976.

[Hatley and Pirbhai, 1987] Derek J. Hatley and Imtiaz A. Pirbhai. *Strategies for Real-Time System Specification.* Dorset House Publishing, New York, 1987.

[Hecht-Nielsen, 1987] Robert Hecht-Nielsen. Counterpropagation networks. In *Proceedings of the IEEE International Conference on Neural Networks*, pages II–19 – II–32, San Diego, 1987.

[Hecht-Nielsen, 1990] Robert Hecht-Nielsen. *Neurocomputing.* Addison–Wesley, Reading, Massachusetts, 1990.

[Hoare, 1985] Charles Anthony Richard Hoare. *Communicating Sequential Processes.* Prentice Hall, Englewood Cliffs, NJ, 1985.

[Knuth, 1983] Donald E. Knuth. *The Art of Computer Programming — Sorting and Searching.* Volume 3, Addison-Wesley, Reading, Massachusetts, 1983.

[Kohonen, 1982] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *[Anderson and Rosenfeld, 1988]* and *Biological Cybernetics*, 43:59–69, 1982.

[Lippmann *et al.*, 1991] Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors. *Advances in Neural Information Processing Systems 3*, Morgan Kaufmann, San Mateo, CA., 1991.

[Marcadé *et al.*, 1990] Erik Marcadé, Frédéric Canut, Nicolas Revault, and Catherine Moulinoux. *N: A Language Dedicated to Neural Algorithm Design.* Technical Report D-110-2, PYGMALION (Esprit Project No.2059), 1990.

[Meyer, 1988] Bertrand Meyer. *Object-Oriented Software Construction. International Series in Computer Science*, Prentice Hall, New York – London, 1988.

[Milner, 1980] Robin Milner, editor. *A Calculus of Communicating Systems.* Volume 92 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin – New York, 1980.

[Moody and Darken, 1988] John Moody and Christian Darken. Learning with localized receptive fields. In *Proceedings of the 1988 Connectionist Models Summer School [Touretzky* et al.*, 1989]*, pages 133–143, 1988.

[Prem, 1991] Erich Prem. *A Description Framework for Solving the "Theory Problem" in Connectionism.* Master's thesis, Universität Wien, 1991.

[Rumelhart and McClelland, 1986] David E. Rumelhart and James L. McClelland. *Parallel Distributed Processing.* Volume 1: Foundations, MIT Press, Cambridge, Massachusetts – London, England, 1986.

[Simpson, 1990] Patrick K. Simpson. *Artificial Neural Systems.* Pergamon Press, New York, 1990.

[Smyth, 1992] S. Gavin Smyth. Designing multilayer perceptrons from nearest-neighbor systems. *IEEE Transactions on Neural Networks*, 2(3):329–333, 1992.

[Stroustrup, 1987] Bjarne Stroustrup. *The C++ Programming Language.* Addison Wesley, Reading, Massachusetts, 1987.

[Toffoli and Margolus, 1987] Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines – A new environment for modelling.* MIT Press, Cambridge, Massachusetts – London, England, 1987.

[Touretzky *et al.*, 1989] David S. Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski, editors. *Connectionist Models – Summer School 1988*, Morgan Kaufmann, San Mateo, CA., 1989.

[Wiklicky, 1992] Herbert Wiklicky. *Synthesis and Analysis of Neural Networks — On a Framework for Artificial Neural Networks.* PhD thesis, University of Vienna – Technical University of Vienna, September 1992.

[Wiklicky *et al.*, 1992a] Herbert Wiklicky, Thierry Denœux, Javier Olarte, Santiago Rementeria, and Veerle Walravens. A tripartite framework for artificial neural networks. In *International Conference on Artificial Neural Networks*, September 1992.

[Wiklicky *et al.*, 1992b] Herbert Wiklicky, Veerle Walravens, Santiago Rementeria, Alberto Lafuente, Javier Olarte, Stéphane Canu, and Thierry Denœux. *A Framework for Artificial Neural Networks.* Technical Report Deliverable D1, NEUFODI (Esprit-II Project No.5433), August 1992.