Salzburg Interest Group on Integrated Systems

siGis-004-1995 March 15, 1996

Object-Oriented Analysis of Biological Neural Networks using OMT

Technical Report siGis-004-1995

Reinhold Huber
reini@cosy.sbg.ac.at

Roland Schwaiger
rschwaig@cosy.sbg.ac.at

Correspondence to:

Universität Salzburg
Institut für Computerwissenschaften und Systemanalyse
Jakob Haringer Straße 2
5020 Salzburg
Austria

Object-Oriented Analysis of Biological Neural Networks using OMT

Reinhold Huber
reini@cosy.sbg .ac.at

Roland Schwaiger
rschwaig@cosy.sbg .ac.at

Abstract

The main purpose of this paper is to provide a new point of view of Artificial Neural Networks. Since the Object Oriented Modeling method provides a one-to-one mapping of the real world it is preferable for Artificial Neural Networks which are derived from real biological Neural Networks. We start emphasizing the advantages of the Object Oriented approach over a functional point of view. We proceed with an introduction to the mathematical background of Artificial Neural Networks. Subsequently we give an insight to the biological neuron. This section is intended to motivate the concluding section which is on the analysis of Artificial Neural Networks and their properties using the Object Oriented Modeling Technique as introduced by Rumbaugh.

1 Introduction

In this paper we are going to throw light on the software (SW) engineering part of artificial neural network (ANN) design. We will concentrate on object oriented (OO) modeling techniques, because OO modeling delivers a one-to-one mapping of the real-world objects. Hence, the model is easier to understand and handle. This is just one reason for prefering the OO paradigm over some other techniques (points of view). More arguments are presented in a separate section.

To appreciate the advantages of the OO paradigm over traditional methods a short survey of the latter is given. Furthermore, we need a method to visualize the result of the modeling procedure. Therefore, we review the Object Modeling Technique (OMT) of *Rumbaugh et.al.* ([RBP+91]).

After a short introduction to ANNs we will present two different designs for ANNs. A discussion of advantages and disadvantages of the introduced models is included.

1.1 What do we need Software Engineering for?

According to *B. Meyer* ([Mey88]),

> "The principal aim of SE is to help produce *quality SW*."

Before we proceed, we have to clarify what is meant by *quality SW*.

Note 1.1 *The quality of SW is determined by two factors:*

1. **External** *factors, s.a. speed or ease of use.*
2. **Internal** *factors, s.a. beeing modular, or readability.*

There is a clear dependency of the external factors on the internal factors ([Mey88]). Furthermore, the internal factors are the key to ensuring that external factors are satisfied!

Some more examples for external factors are ([Mey88],[PB93]): Correctness, Robustness, Reliability, User-Friendlyness, Efficiency, Portability.

In this paper we will concentrate on the analysis methods devised for SW development and hence, addressing the external factors given above. Many methods have been developed to help the SW engineer to reach the goal of designing quality SW. These methods depend on the different "philosophies" of how to "see" and model problems. Various points of views towards problems are possible ([Hic93]):

1. Procedural point of view,

2. Declarative point of view,
3. Functional point of view,
4. OO point of view.

The first three points of view are refered to as the *traditional* methods (views). The last item is known to as the *object oriented* approach.

2 Overture

2.1 Review of Traditional SW Development Methods

In this section we will concentrate on the first of the traditional methods, namely the procedural method. In Figure 1 the *waterfall model* as an example for a procedural SW life cycle model is shown.

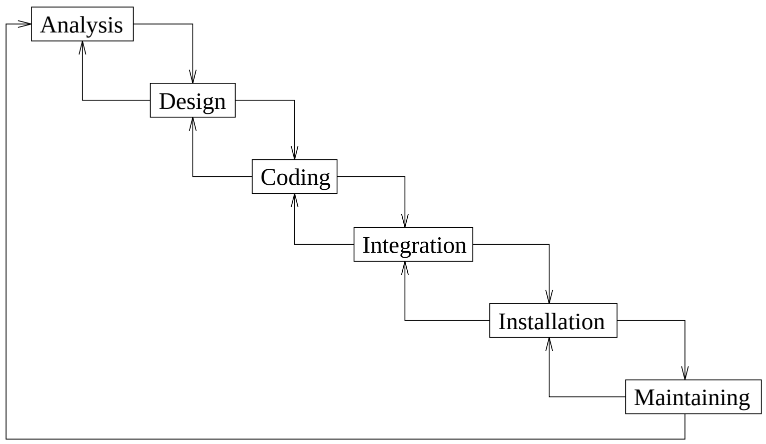

Figure 1: The Waterfall Model, from [Hic93].

2.1.1 The classical sequential SW Life-Cycle-Method

The traditional approach to designing SW after the Life-Cycle-Method is based on the "top-down" decomposition of a specified problem ([PB93]). That means, we start from the initial problem and try to refine and split the problem until the wanted complexity of partial problems is reached[1].

$$\begin{aligned} \mathcal{P} &= \cup_{i_1} P_{i_1}(i_1 = 0, \ldots, n_1), \\ P_{i_1} &= \cup_{i_2} P_{i_1 i_2}(i_2 = 0, \ldots, n_2), \\ &\vdots \\ P_{i_1 i_2 \ldots i_{m-1}} &= \cup_{i_m} P_{i_1 i_2 \ldots i_m}(i_m = 0, \ldots, n_m), \end{aligned}$$

where the complexity of the subproblems $P_{i_i \ldots i_m}$ is small enough to be handled[2].

The basic components of the SW system to be designed are the solutions to the partial problems given above. One can see the obvious problem of this approach. Once the problem is split into its subproblems and the system design and implementation is finished it is quite hard to change the functionality and structure of the problem solution without having side effects on the whole system[3], (see Figure 2).

[1] This type of analysis is called *structured analysis*.

[2] The flow of data from one subproblem to the other can be shown in a graph which is called the *data flow diagram*. An accompanying type of diagram shows the relation between the data entities. It is called the *entity relationship diagram*.

[3] Of course the level of splitting has an influence, too.

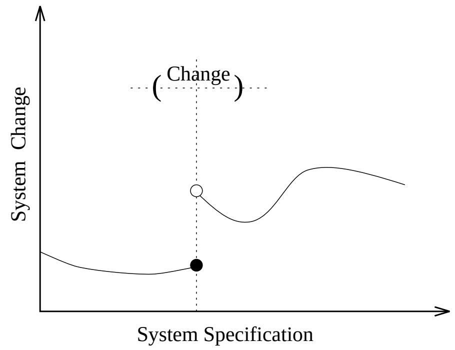

Figure 2: The side effects of changing the system specification, from [Mey88].

To use a mathematical analogy: The change of the problem solution (system specification) can be viewed as a local (small) change in the abscissa of a graph. With the classical sequential SW Life-Cycle-Method a change in the system can refere to a discontinuity in the ordinate of the graph ([Mey88]).

Note: This SW design method is based on operations.

2.2 Introduction to Object Orientation

In general *Object Engineering* consists of the following components: OO Analysis (OOA), OO Design (OOD), OO Programming (OOP), and Management and Documentation of OO Projects ([Bru94]). We will concentrate on the first two components.

The SW Life Cycle for OO SW looks different from the waterfall model described in the previous section. In OO SW design analysis and design are intertwined. In Figure 3 we show one possible model[4] for the OO SW Life Cycle.

2.2.1 OOA

The purpose of the analysis step in building SW systems is to understand the problem, build relationships, and put the system together (possibly using "scenarios"). From an OO point of view this means that real world objects and their relations have to be identified.

2.2.2 OOD

In contrast to the method explained in the previous section

> "OO Design (OOD) is the method which leads to SW architectures based on the objects every system or subsystem manipulates ([Mey88]),"

more precisely

> "OO Design (OOD) is the construction of SW systems as structured collections of abstract data types implementations. ([Mey88]),"

and hence its basis are structural, static, *data* aspects of a system, despite to a functional or procedural point of view which is very common to programmers.

2.2.3 Difference between OO and Traditional Modeling Techniques

To get an idea of the difference between traditional methods and the OO approach to SW design *Wirth's* [5] formula will be used ([Bru94]). Basically Wirth's formula says:

[4] The Raoid Prototyping model is not necessarily an OO-model

[5] In 1976 Niklaus Wirth called one of his books "Algorithms + Data Structures = Programs" [Wir76]

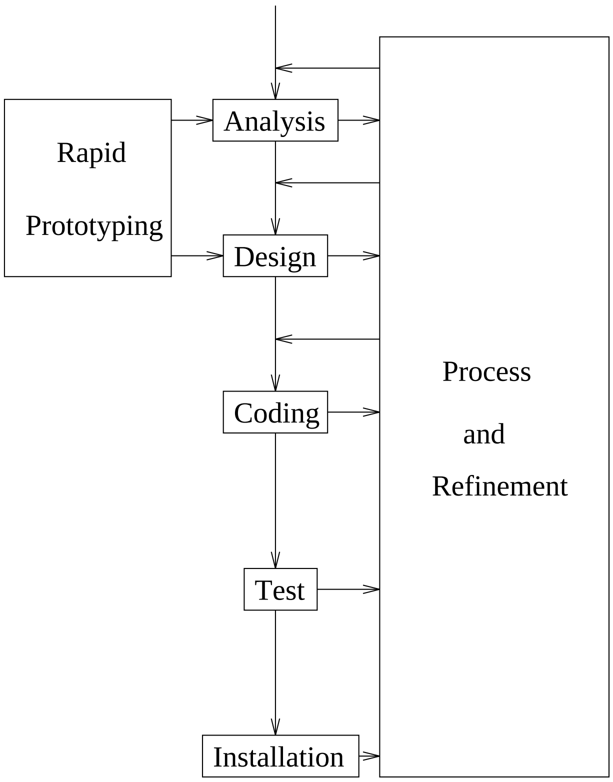

Figure 3: The Rapid Prototyping Model, from [Hic93].

$$\text{Program} = \text{Algorithms} + \text{Data Structures}$$

Traditional methods view this formula like:

$$\text{Program} = (\text{Algorithms}) + (\text{Data Structures})$$

Intermediate modeling steps view it like:

$$\text{Program} = (\text{Algorithms} + \text{Data Structures})$$

Finally, the OO modeling point of view of Wirth's formula is:

$$\text{Object} = (\text{Algorithms} + \text{Data Structures}),$$
$$\text{Program} = (\text{Object} + \text{Object} + \dots)$$

To state it in a more formal way ([Dun93]):

> "Imagine the class of expression $f(x)$, where f is a function and x is its argument. Then the *abstraction of the function* is the variation of the argument x (over its domain) for a certain function f. The *abstraction of the object* is the variation of all functions f, applicable to the object x."

2.2.4 OO Terms

Class: Describes a group of objects with similar attributes, common methods, common relationship to other objects, and comman semantics.

Object: Is the instantiation of a class.

Attributes: Are the data of a class.

Method: Is the algorithmic part of an object. It operates on the data of the object.

Encapsulation: Is the separation of the external aspects of an object from the internalimplementation details of the object.

Polymorphism: Methods and functions that have the same name but have different behaviors for different objects.

Generalization: Is the relationship between a class, the *superclass*, and one or more refined versions, the *subclass(es)*, of it. Each subclass is said to *inherit* the features of its superclass.

Inheritance: Refers to the mechanism of sharing attributes and methods using the generalization relationship.

2.3 Introduction to ANN

This section is intended to give a short introduction to ANN. In fact, in the example given below we will concentrate on socalled *Feed Forward Networks*. Although we will not consider the biological background of this topic we would like to point the interested reader to some usefull resources, s.a. [Tho93], [ST83].

Artificial neural networks, or neural networks are information processing systems which are composed of basic units (neurons, cells). These units are connected by directed links (connections). Via these links information is sent from unit to unit and distributed all over the network.

We will use the definition of *D. Nauck et al.* ([NKK94]) to give a formal introduction to ANNs:

Definition 2.1 *An* **artificial neural network** *is a tupel* (U, W, A, O, NET, ex), *where:*

(i) U *is a finite set of* neurons.

(ii) W *the network structure (*weights*), is a mapping from* $U \times U$ *into* $\mathbf{R}$, *i.e.* $W : U \times U \to \mathbf{R}$.

(iii) A *is the union of all the* A_u, *which are mappings from* $\mathbf{R}^3$ *into* $\mathbf{R}$, *i.e.* $A_u : \mathbf{R}^3 \to \mathbf{R}$. A_u *is called the* activation function *for each* $u \in U$ *with* activation value a_u. a_u *can be defined as*

$$a_u = A_u\left(a_u^{old}, net_u, ex_u\right).$$

net_u and ex_u are explained below. If we like to indicate the layer to which the activation function and value belongs we use a superscript s, i.e. A_u^s and a_u^s, respectively. a_u^{old} denotes the activation value of unit u prior to the actual time step. In our thesis we will sometimes use f_u^s or f instead of A_u^s for the activation function.

(iv) O *is the union of all the* O_u, *which are mappings from* $\mathbf{R}$ *into* $\mathbf{R}$, *i.e.* $O_u : \mathbf{R} \to \mathbf{R}$, *and is called the* output function *for each* $u \in U$ *with* output value $o_u = O_u(a_u)$. *Again superscript s may indicate the layer of the unit, i.e. o_u^s and O_u^s.*

(v) NET *is a union of the mappings* net_u, *which are mappings from a subset* $U' \subseteq U$ *into* $\mathbf{R}$, *i.e.* $net_u : (\mathbf{R} \times \mathbf{R})^{U'} \to \mathbf{R}$. *It is called the* input function *of the neuron. Superscript s indicates the layer.*

(vi) ex_u *is an external input function,* $ex_u : U \to \mathbf{R}$. *Within the category of possible external inputs to a unit there is a special one namely the* threshold *of a unit denoted by* θ_u. *In biological neurons the threshold determines the limit for spiking and not spiking. Once the neuron spikes encoded information is sent to connected neurons.*

To adopt a network to a given problem, it is necessary to present input examples to the net. Depending, if the network is of an unsupervised or a supervised type [Hay94] output examples do not have to, have to be presented respectively. The network extracts the main features of the examples and is able, in general, to generalize to not shown inputs. The way the network extracts information from the examples is called learning. Thus a so-called learning rule determines the way a network adopts.

2.4 An Example: Feed Forward Neural Networks (FFNN)

For the better understanding we will give an illustrative example of an ANN following the definition given in section 2.3. The following problem is known as the XOR or parity problem. Assume that the truth table for the XOR problem is given as:

Input		Output
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

An example of an ANN which is able to "learn" the mapping given in the table above is seen in figure 4. As can be seen the network is organized in layers which are called according to their visibility, input, output and hidden layer. All the following explanations are related to figure 4.

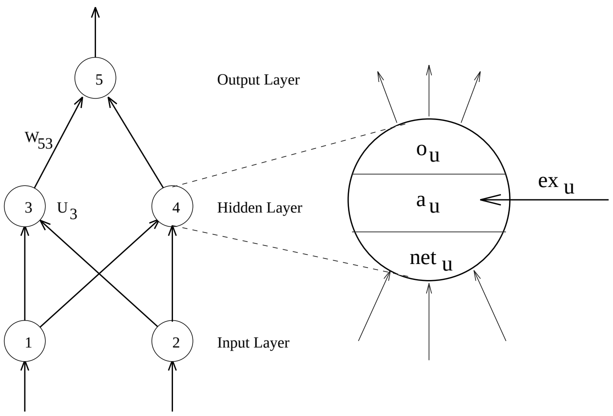

Figure 4: An example of an ANN topology, which is known to solve the XOR problem.

(i) First we have to define a finite set of neurons. $U = \{u_1, \ldots, u_5\}$, where u_1 and u_2 are the input units, u_3 and u_4 are hidden units and u_5 is the output unit.

(ii) Next the network structure W will be given. It can be visualized in a matrix, the so-called connection matrix:

$$W = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ w_{31} & w_{32} & 0 & 0 & 0 \\ w_{41} & w_{42} & 0 & 0 & 0 \\ 0 & 0 & w_{53} & w_{54} & 0 \end{pmatrix}$$

Here w_{ij} denotes the weight for the connection from unit u_i to unit u_j.

(iii) There are a lot of possibilities to define the activation function of a unit, e.g.

$$A_u\left(a_u^{old}, net_u, ex_u\right) = \frac{1}{1 + e^{-(net_u + \theta_u)}}.$$

This function is continuous and therefore all concatenations of A_u's from input to output layer will yield a continuous function [Heu90].

(iv) Usually the output function O_u is the identical mapping, i.e. $O_u(a_u) = a_u$. The output of the neuron is determined by the activation function.

(v) At this stage all the incoming signals with respect to unit u_u are collected. The incoming signals o_i are weighted by the weights w_{ui} of the connections from some unit $u_i \in U'$ to unit u_u. An example for an input function may be:

$$net_u = \sum_{i=1}^{n} w_{ui} o_i.$$

To be more specific: In figure 4 unit u_3 receives input from unit u_1 and u_2. Hence its net input is $net_3^1 = w_{31}o_1 + w_{32}o_2$.

(vi) In our example ex_u determines the shift of the activation function on the input axis.

The input/output examples for the XOR problem are

$$\{(x_p, t_p) \,|p = 1, \ldots, n_p\} = \{((0,0),0), ((0,1),1), ((1,0),1), ((1,1),0)\}.$$

As we mentioned above, the associated learn rule for this type of ANN is called "Error Back-Propagation" rule. During the learning phase the ANN is presented pairs of input-/output-vectors. After the learning phase the ANN should be able to associate the given and novel input-/output-pairs, with respect to the given context. For unknown inputs the ANN should produce outputs according to the required approximation criteria.

More formally the modification of each of the weights in the network can be written as

$$w_{ji}^{new} = w_{ji}^{old} + \triangle_p w_{ji},$$

where

$$\triangle_p w_{ji} = \eta \delta_{pj} o_{pi}.$$

η is called the learn rate and o_{pi} is the ith component of the actual output. This formula states the general form of the learn rule and differs in the δ_{pj}. For an output unit it is given by

$$\delta_{pj} = (t_{pj} - o_{pj}) f_j'(net_{pj}),$$

where t_{pj} is the wanted output of the ANN. f denotes the activation function. For a hidden unit δ_{pj} looks like

$$\delta_{pj} = f_j'(net_{pj}) \sum_k \delta_{pk} w_{kj}.$$

Finally, we want to summarize the learn cycle of a Feed-Forward Network:

1. Select an input/output pair from input/output pair pool.
2. Present the input vector to the ANN.
3. Calculate the error of the propagation.
4. Apply modifications to the free parameters of the ANN by back-propagating the error.
5. If not all input/output pairs in the pool have been selected, go to the first step.
6. Repeat this procedure until the overall error is small enough or the ANN is not able to learn the associations.

2.5 Introduction to OMT

In this section we are going to give a brief introduction to the modeling technique introduced by *Rumbaugh* [RBP+91] et al.

The OMT methodology consists of several phases:

Analysis: During this step the problem and its domain are investigated. This step is concerned with devising a precise, concise, understandablei, and correct model of the real world. A written *problem statement* serves as a starting point. Concurrently to the design progress the dialog between customer and system designer improves the initial statement in an evolutionary manner. This iterative process is repeated until the problem statement describes the problem without defiencies. The output from the analysis step is a formal model consisting of three aspects. First, the *objects* with their relations (Object Model). Second, the interactions among the objects (Dynamic Model). Third, the data transformations of the system (Functional Model) [6].

- The *object model* describes the static structure of the objects in a system and their relations to other objects. It is visualized in an *object diagram*, where the nodes are classes (objects) and the edges are the relationships.
- The *dynamic model* describes those aspects of a system concerned with time and the sequencing of operations. Starting the dynamic analysis one has to look for *events* and *states*. Usually *scenarios* of typical and exceptional sequences are prepared. For each active object a *state diagram* should be provided. In a state diagram the nodes are the states and the edges are the transitions between the states caused by events.
- The *functional model* describes the computations - data value transformations - within the system. For each input value or output value generated by the system a *data flow diagram* (DFD) is constructed. A DFD consist of processes that transform data (nodes), data flows that move data (edges), actor objects that produce and consume data (nodes), and data store objects that store data passively (nodes).

System Design: During this phase the overall structure of the system is determined. The results of the analysis phase are the essential input to this process.

Object Design: The object model from the analysis stage is elaborated. During this phase the focus is shifted from the application concept to the computer concept. Subsystems are packed into modules.

Implementation: The final phase of the cycle. During this stage the focus has shifted completely to the computer concept.

2.5.1 The Object Model

The object model describes the structure of objects in a system (their identity, their relationship to other objects, their attributes, and their operations) and is visualized in an object diagram.

There are various ways how to model an ANN. It depends on the point of view of the designer. If the designer basically concentrates on units as the basic building blocks of an ANN model he completely neglects the existence of cell layers or cell groups. These aspects are considered in the "layer(group)"point of view. In Figure 5 a simple class diagram is shown.

The layers are charaterized in the following way: Layers consist of units. These units are not connected within the layer. A layer is either an input-, hidden-, or output-layer. A layer posesses methods to manipulate the units.

[6] Any OOA notation must at least have symbols for classes, objects, inheritance, and communication between objects. A lot of different notations are available for OOA, see [Hru94] and [BS94] for a comparison of the different notations. We will concentrate on the *Object Modeling Technique* introduced by Rumbaugh et.al. ([RBP+91]) and use the Feed-Forward ANN, introduced in the previous section, as an illustrative example.

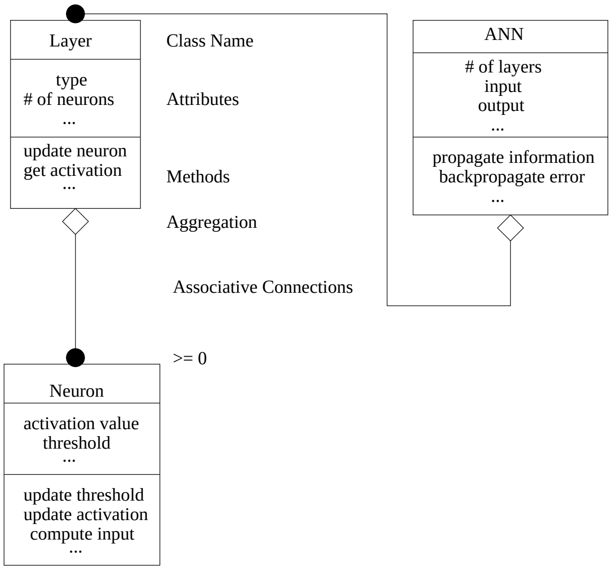

Figure 5: The Class ANN, Layer, and Unit.

We have to notice that Figure 5 is on the class level. For an object diagram one has to replace the boxes with rectangular corners by boxes with rounded corners and substitute the concrete values for the attributes.

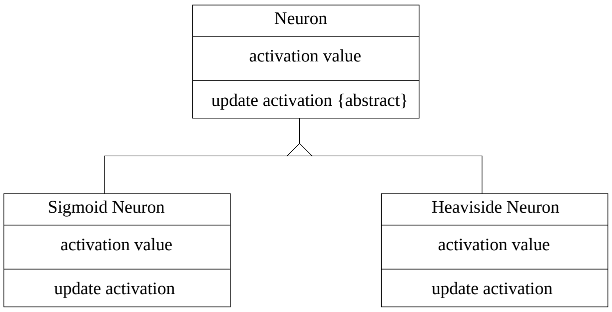

Figure 6: The Inheritance Scheme.

Of course, the definition of the unit in Figure 5 is a very narrow one. In Figure 6 an example of a possible abstract unit class and its ancestors are given.

2.5.2 The Dynamic Model

The dynamic model describes those aspects of a system concerned with time and the sequencing of operations. Starting the dynamic analysis one has to look for *events*. Then the permissable event sequence is summarized in a state diagram. In Figure 7 a possible dynamic model is given for the learning cycle in a Feed-Forward Network.

2.5.3 The Functional Model

The functional model describes those aspects of a system concerned with transformations of values (data flow diagram (DFD)). A DFD consists of processes that transform data, data flows that move data, actor objects that produce and consume data, and data store objects that store data passively.

The following steps have to be performed in order to generate the functional model([RBP+91], p.180):

1. Identify input and output values.
2. Build data flow diagrams showing functional dependencies.

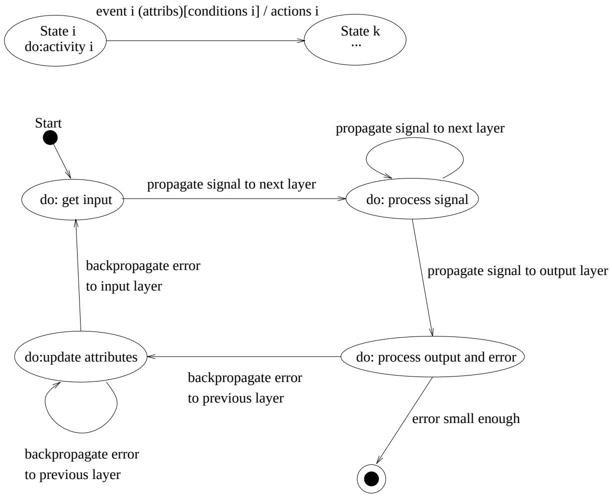

Figure 7: The Dynamic Model.

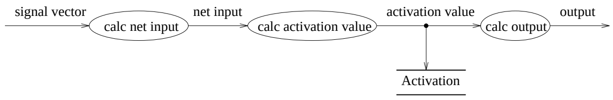

Figure 8: The Functional Model of the class unit.

3. Describe functions.
4. Identify constraints.
5. Spezify optimization criteria.

In Figure 8 the functional model of a unit is given. Data is collected at the entry point of the unit and several values are calculated. The activation is stored for further processing.

3 OO-Analysis of Biological Neural Networks

3.1 Existing Models of Biological Neural Networks

Many different simulation tools for ANNs are available at the time [Rit94]. Unfortunately, most of them are very limited by the used model, learning algorithms, and extendibility. Refering to [Lin94] the most severe disadvantages are:

- No or a too specialized building block abstraction.
- No simulator is able to apply different training algorithms generically to arbitrary functional architectures.
- Most of them require knowledge of C or C++, if one wants to extend it.
- **No simulator is really object-oriented.**

In the same paper the SESAME tool - an object-oriented approach - is presented. The basic building blocks are either neurons, whole layers of neurons, neural nets or even full applications. In the used notation nodes or blocks perform computations, whereas the arcs or connections specify the data dependencies. Figure 9 shows an example for a multilayer architecture in SESAME:

As can be seen from Figure 9 the notation is not consistent with the Object Model Notation suggested in [RBP^{+}91]. The used point of view is rather a functional point of view. Boundaries of the basic elements, neurons in this case, cannot be easily identified. The used modeling approach is similar to the Structured Analysis/Structured Design (SA/SD) method. Although OMT and SA/SD

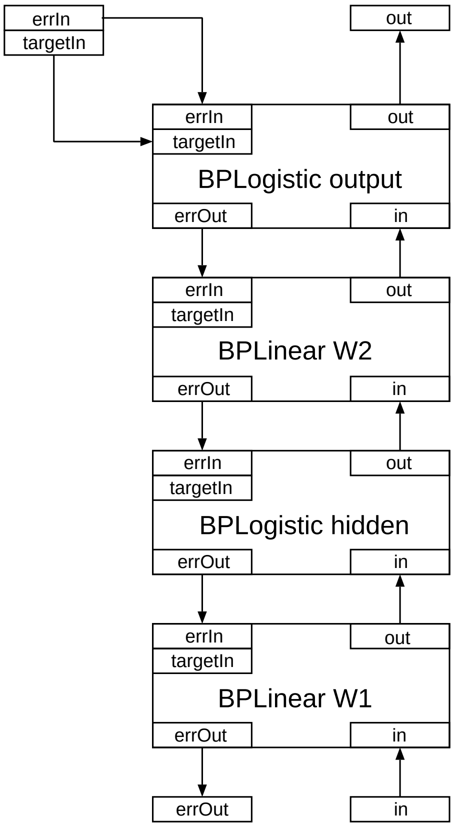

Figure 9: Block diagram for a multilayer net architecture in SESAME.

have much in common, the difference is primarily a matter of style and emphasis. In SA/SD the functional model dominates, the dynamic model is next important and the object model is the least important.

The next subsections are heading for an ANN model using a more intuitive notation - an object oriented description.

3.2 A Model for Biological Neural Nets

Although the existing ANN paradigm used in Computer Science is derived from a natural neural network model, the modeling used in most cases is a very simple representation of the real biological neuron and its behavior. Before we proceed to perform a modeling of ANNs, we take a look at the biological neuron and information processing within it.

3.2.1 Structure of the Biological Neuron

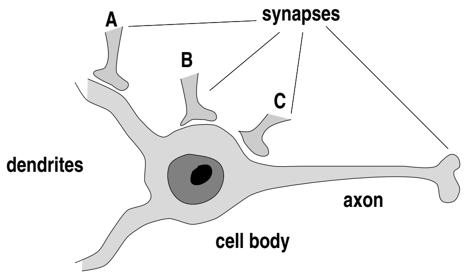

Figure 10: A biological neuron.

To understand the biological neuron we start with exploring its properties. Figure 10 depicts the basic elements of a typical neuron. More detailed pictures are given in physiological books such as [BS91], [ST83] or [Tho93].

A biological neuron constitutes of three parts - *dendrites*, *soma* and *axon*. The tree-like dendrites are connected to the cell body or *soma*, where the cell nucleus is located. The *axon* which forms a single long fiber is extending from the soma. To connect a neuron's axon to more than one other neuron the axon may branch. On the end of each branch a *synapse* forms the transmitting junction to other neurons. Depending on the location of the synapse three different kinds of synapses are distinguished (refer to Figure 10):

A: axo-dendritic synapse,

B: axo-somatic synapse,

C: axo-axonic synapse.

Due to [ST83] a further distinction for axo-axonic synapses can be made into proximal axo-axonic synapses, as shown in Figure 10, and distal axo-axonic synapses, e.g. the first axon hits the second axon next to its terminal synapse. A typical neuron in the cerebral cortex has literally thousands of synapses from other neurons [Tho93].

There are basically two types of synapses: chemical and very fast electrical. Most of the synapses in the mammalian brain are chemical synapses. Vesicles are responsible for releasing neuro transmitters into the synaptic cleft. The transmitter molecules diffuse across the synaptic cleft and attach to specific chemical receptor molecules on the surface of the postsynaptic target cell. Three types of chemical synapses are known, round, flat and dense-core vesicle synapses. Depending on the type of the synapse different transmitters are infused into the cleft.

Round vesicle synapses are thought to act excitatory, i.e. the target neuron activity is increased. Flat vesicle synapses act inhibitory, i.e. the target activity is decreased. Dense-core synapses act either excitatory or inhibitory.

The terminals of axo-dendritic synapses contain round vesicles, therefore, they act excitatory. Inhibitory synapses, i.e. synapses in which the terminals contain flattended vesicles, cluster on the cell body, especially near the initial segments, where the axon leaves the cell body. In general, axo-somatic and axo-axonic synapses are inhibitory.

3.2.2 Activation and Transformation

A neuron can be stimulated by external impact on its connected sensors, e.g. a change in light intensity in the case of a light-receptor. In the soma of the recepting neuron a difference in the electrical potential, called electric potential, is created. Due to *adaption* the sensoric potential is decreasing although the stimulus is remaining constant.

In the axon the sensoric potential causes the process of *depolarization*. The depolarization process again results in periodic peaks in the activation signal. The signal transmitted can be seen as frequency-coded - the higher the value of the stimulating signal the longer and denser are the burst of peaks in the activity. The activity of the neuron is governed by the *all-or-nothing* principle, i.e. the neuron fires, if the potential of activity is greater than a given threshold. After firing the neuron depolarizes.

Figure 11 gives an idea how the the initial stimulus is transfered to the signal which is further transmitted through the neuron's axon.

Depending on the type of receptor the transfer function between the stimulus and the activity looks different. Sensors measuring pressure or stretching for instance have a nearly linear transfer function. In general, the frequency of the activation F and the value of the Stimulus S are related by

$$F = kS^n$$

where k is a constant which denotes the slope of the function. In the case of $n = 1$ we get the pressure sensor mentioned above. If $n > 1$ we get the behaviour of sensors measuring pain. The case of $n < 1$ concerns our senses, e.g. sensors located in the eye's retina.

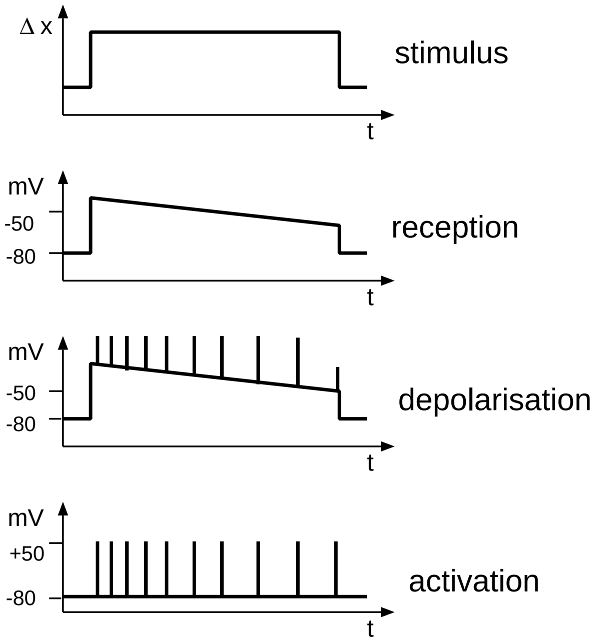

Figure 11: Transformation of a stimulus into activation

3.2.3 Nets of Neurons

A stimulus affects thousands of sensors at once. This information is transmitted through the nervous system. Figure 12 shows a portion of a neural net. If the neuron in the middle is activated the emitted signal influences the activation of all connected neurons, e.g. all neurons of the next *group* or layer. This effect is called *divergence* and results in a loss of localization ability of a stimulus. On the right side of Figure 12 the distribution of the activity caused by the stimulus is shown.

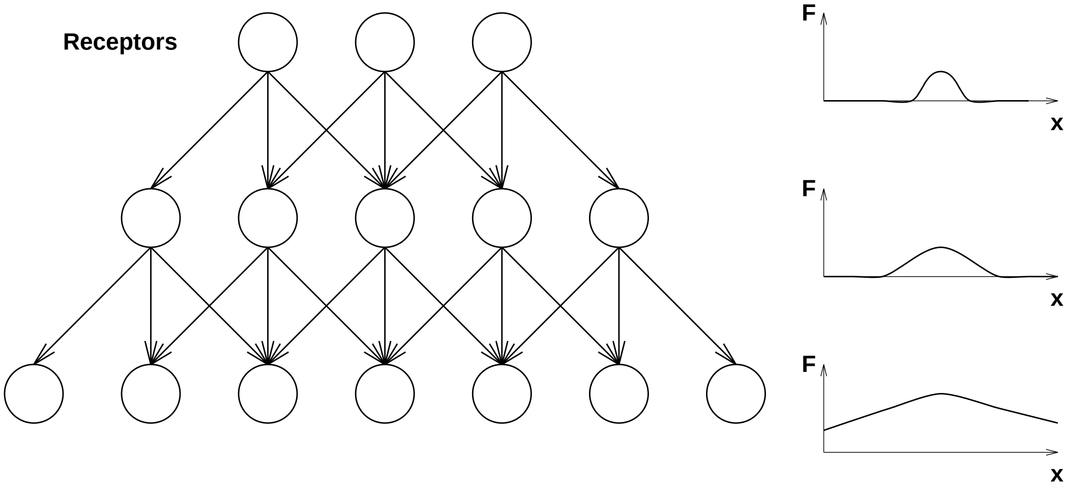

Figure 12: Divergence in a Neural Net

Another effect is *convergence* which appears if more than one recepting neuron is stimulated. The following neurons get input from more than one receptor which results in the inability to recognize the number of stimuli.

Fur further discussions suppose brain functions like the ones described above. In this case a few stimuli would activate the whole brain. Fortunately, inhibitory connections disturb this behaviour. In Figure 13 the shaded neurons represent inhibitory acting neurons. These neurons are called *interneurons*. The shaded connections leaving from the shaded neurons employ negative feedback which causes inhibition of the connected neurons. This kind of inhibition is called *Lateral inhibition*. Again, on the right of Figure 13 the activity distribution is shown.

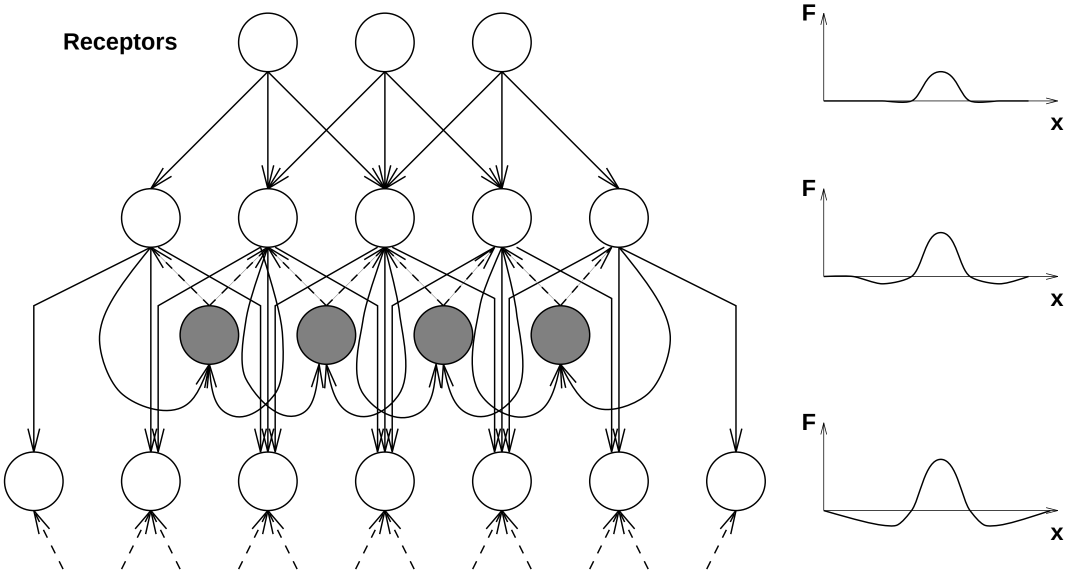

Figure 13: Lateral Inhibition in a Neural Net

The shaded neurons are neurons acting inhibitory and the dashed arrows denote the inhibitory connections. A zone of inhibition surrounding the stimulus causes the effect of contrast enhancement which is especially important in visual systems.

3.2.4 Learning in Neural Nets

The most famous learning algorithm for Neural Nets is *Hebb's rule*. This rule states that an association between two neurons is emphasized, if and only if the pre- and postsynaptic neuron are activated at the same time. Another learning mechanism was discovered by Tauc and Kandel [KH92] who showed that a synaptic association can also be reinforced if the postsynaptic neuron is not activated. There is strong biological evidence that both mechanisms are existent in mammalians.

3.3 Modeling using OMT

In the last subsection we provided an insight into the biological neuron and it was also the first step in OMT - the initial description of the problem. The problem statement is followed by the object model.

3.3.1 The Object Model

We have to start by identifying object classes and remove awkwardly chosen and redundant classes. The remaining classes are:

- dendrites
- soma
- axon
- synapse
- neuron (biological or artifical)
- group or layer
- neural net (biological or artifical)

- stimulus

Isolated words have too many interpretations, so a data dictionary has to be prepared:

dendrite: part of the neuron from where signals acting excitatory are transmitted to the soma.

soma: central part of the neuron where the transformation of the stimulus to activity is performed.

axon: part of the neuron which transmits the frequency representing the neuron activity to its terminal part, the synapse.

synapse: transmitting junction of the axons which connects to parts of other neurons.

neuron: processing element, consisting of dendrites, soma, axon and synapes.

group: a set of neurons, usually a layer.

neural net: a set of groups of neurons.

stimulus: external physical entity - s.a. pressure, light, noise, etc.

For the object model the relevant associations are "Consists of" and "Connected to" relations. Figure 14 shows the object model.

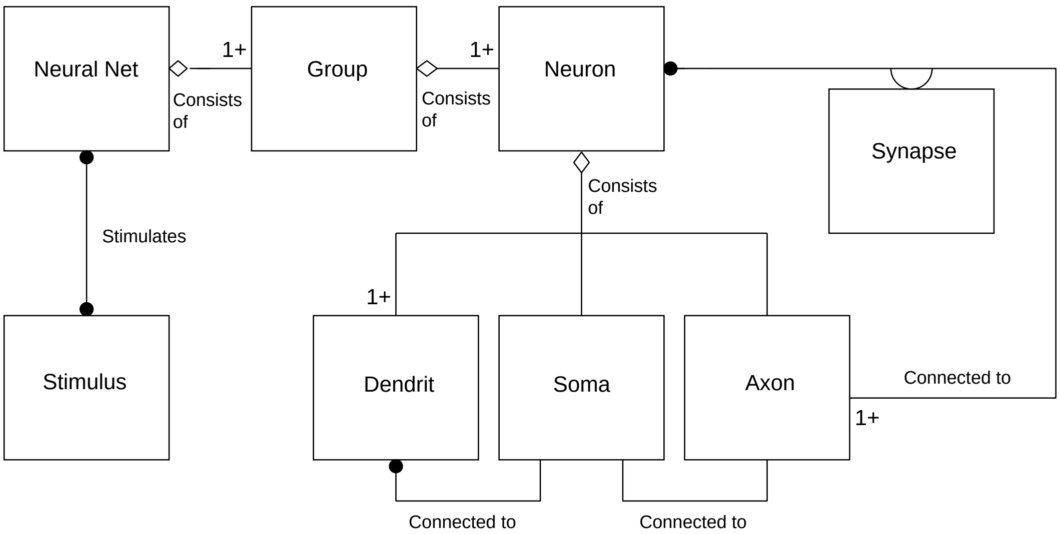

Figure 14: Initial Object Diagram of an ANN

We proceed with identifying attributes. Arributes are properties of individual objects. Furthermore, associations should be used to show any relationships between objects, i.e. synapses are modeled as link attributes between a neuron and a axon object.

Attributes should be modeled as qualifiers, if they depend on a particular context, i.e. if an attribute selects among objects in a set it is good design to model it as a qualified association. Figure 15 shows the model extended with attributes.

Further refinement is achieved by using inheritance, either through generalization (bottom-up) or specialization (top-down). In the example, refinement is performed for the "group" class. Specialization is done for the three similar modeled components which are aggregated to the neuron class. Figure 16 shows the object model so far.

The object model should be iterated. Weaknesses may occur in a later step in the analysis, or in the design step.

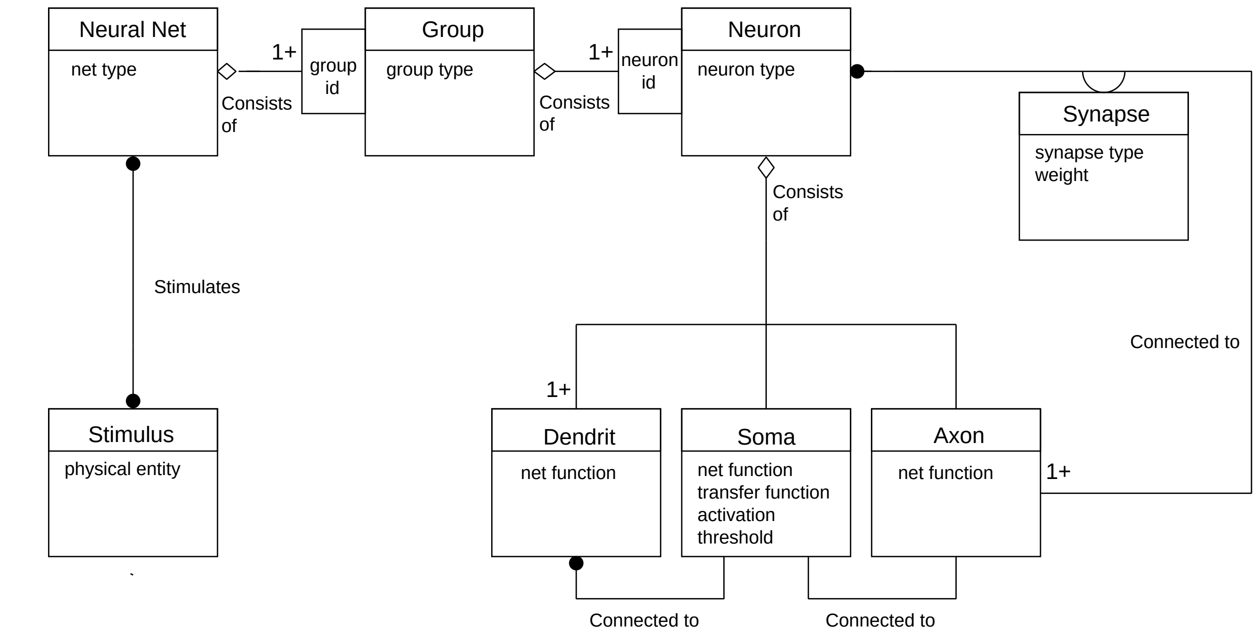

Figure 15: ANN Model with Attributes

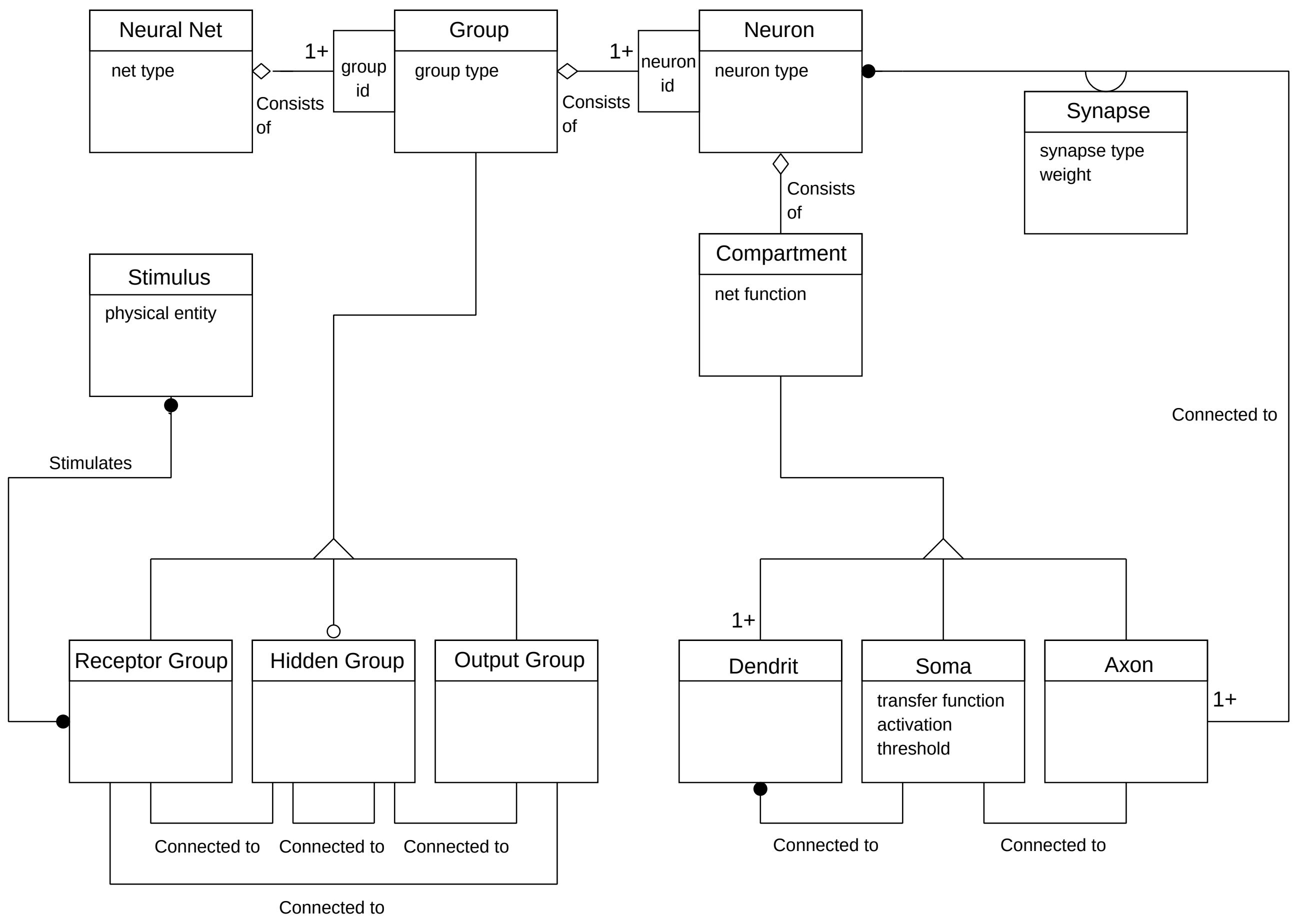

Figure 16: ANN Model with Attributes and Inheritance

3.3.2 The Dynamic Model

The dynamic model shows the time-dependent behaviour of the system. Basically, there are two different scenarios in neural networks: first, response to external stimuli and - especially important in ANNs - second, the learning or training of the net. The dynamic model given in the previous section (depicted in Figure 7) characterizes only one possible scenario, the learning phase of a backpropagation ANN.

We try to give a general ANN dynamic model, where backpropagation is covered by a special case. In Figure 17 a state diagram for general trained ANNs is shown.

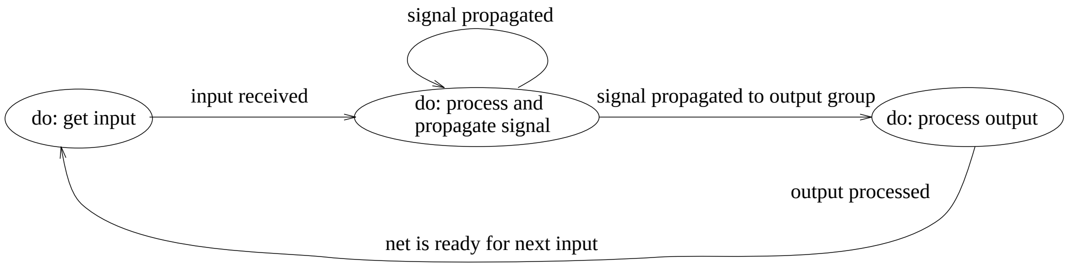

Figure 17: State Diagram for ANNs

Depending on the chosen network paradigm we get different dynamic models for the learning of the ANN. Figure 18 shows the dynamic model for backpropagation learning in Feed-Forward networks and Hebb learning in Kohonen networks. The learning algorithms are given in [Roj91].

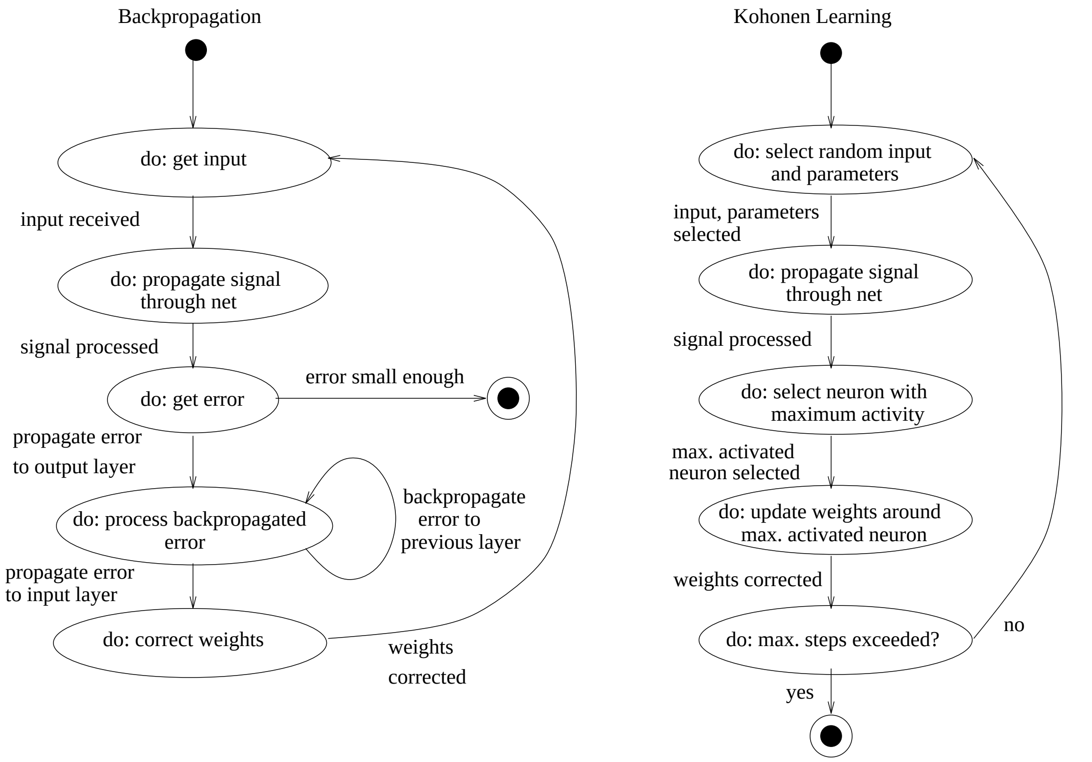

Figure 18: Dynamic Model for Learing Algorithms in ANNs

3.3.3 The Functional Model

The last and least important model is the functional model, where dependencies of values and functions which relate them are shown. Data flow diagrams are used to visualize functional dependencies. Figure 19 shows the DFD for the neuron class.

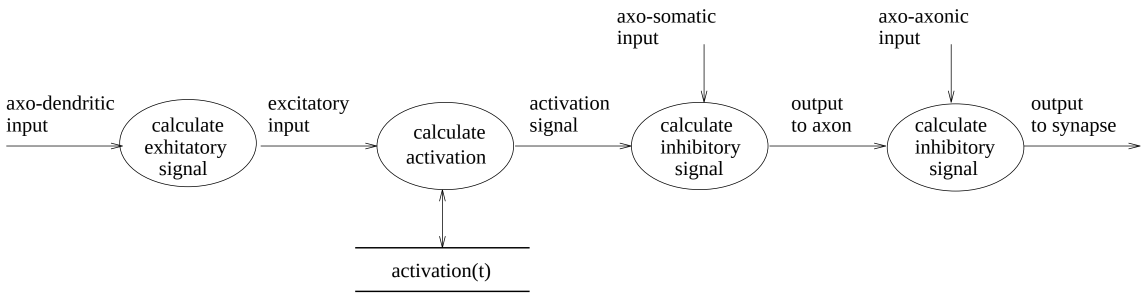

Figure 19: Functional Model for the class neuron

Individual functions may be described by different means, we choose pseudocode descriprions, i.e. for the function "calculate excitatory signal" and "calculate activation".

calculate activation (input, threshold, time)
  if input > threshold
  calculate activation depending on adaption through time.

excitatory signal (axo-dendritic input)
  calculate $\sum_i w_i o_i$, where w_i are the synaptic strengths (weights) and o_i are the axo-dendritic input vector elements.

4 Summary and Outlook

In this paper we have given a truly OOA of biological neurons and networks. We have shown that our analysis covers the traditional ANN models as well as siginificant extensions. Our OOA of the biological system, which is given in the three main components - the object model, the dynamic model, and finally the functional model - is independent of particular network topologies and learning paradigms.

Acknowledgements

We would like to thank Maria Waldhör for her valuable comments on the biologically oriented sections and Helmut A. Mayer for reviewing this paper.

References

[Bru94] D.E. Brumbaugh. *Object-Oriented Development.* Wiley, 1994.

[BS91] N. Birbaumer and R.F. Schmidt. *Biologische Psychologie.* Springer, 1991.

[BS94] H. Balzert and W. Stein. Worin unterscheiden sich die Objektorientierten Methoden. *OBJECTspektrum*, pages 23–28, 1994.

[Dun93] R.H. Dunn. *Software-Qualität.* Hanser, 1993.

[Hay94] S. Haykin. *Neural Networks. A Comprehensive Foundation.* MacMillan, 1994.

[Heu90] H. Heuser. *Lehrbuch der Analysis: Teil 1.* B.G. Teubner Stuttgart, 1990.

[Hic93] A. Hickersberger. *Der Weg zur Objekorientierten Software.* Hüthig, 1993.

[Hru94] P. Hruschka. Objektorientierte Analyse und Designprinzipien. *OBJECTspektrum*, pages 16–22, 1994.

[KH92] Eric R. Kandel and Robert D. Hawkins. Molekulare Grundlagen des Lernens. *Spektrum der Wissenschaft*, pages 66–76, November 1992.

[Lin94] A. Linden. An Object-Oriented Dataflow Approach for Better Designs of Neural Net Architectures. In M. Mozer et.al., editor, *Proceedings of the 1993 Connectionist Models Summer School*, pages 397–404, Hillsdale, New Jersey, 1994. Lawrence Erlbaum Associates, Inc.

[Mey88] B. Meyer. *Object-Oriented Software Construction.* Prentice Hall, 1988.

[NKK94] Detlef Nauck, Frank Klawonn, and Rudolf Kruse. *Neuronale Netze und Fuzzy-Systeme.* Vieweg, 1994.

[PB93] G. Pomberger and G. Blaschek. *Software Engineering.* Hanser, 1993.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[Rit94] N. Ritter. Neural Network Simulators. Technical Report 94/1, Edwith Cowan University, 1994.

[Roj91] Raul Rojas. *Theorie der neuronalen Netze.* Springer, 1991. ISBN 3-540-56353-9.

[ST83] Robert F. Schmidt and Gerhard Thews, editors. *Physiologie des Menschen.* Springer, 1983.

[Tho93] Richard F. Thompson. *The Brain. A Neuroscience Primer.* W.H. Freeman, 1993.

[Wir76] Niklaus Wirth. *Algorithms + Data Structures = Programs.* Prentice-Hall, 1976.