Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-93-23

Comparative Study of Connectionist Simulators

Andreas Dengel, Ottmar Lutzy

Mai 1993

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, SEMA Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Friedrich J. Wendl
Director

Comparative Study of Connectionist Simulators

Andreas Dengel, Ottmar Lutzy

DFKI-RR-93-23

A shorter version of this report will be published in IEEE Expert (1993) [12].

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9003 0).

Comparative Study of Connectionist Simulators

ANDREAS DENGEL and OTTMAR LUTZY

Authors' Abstract — This paper presents practical experiences and results we obtained while working with simulators for artificial neural network, i.e. a comparison of the simulators' functionality and performance is described. The selected simulators are free of charge for research and education. The simulators in test were: (a) PlaNet, Version 5.6 from the University of Colorado at Boulder, USA, (b) Pygmalion, Version 2.0, from the Computer Science Department of the University College London, Great Britain, (c) the Rochester Connectionist Simulator (RCS), Version 4.2 from the University of Rochester, NY, USA and (d) the SNNS (Stuttgart Neural Net Simulator), Versions 1.3 and 2.0 from the University of Stuttgart, Germany. The functionality test focusses on special features concerning the establishment and training of connectionist networks as well as facilities of their application. By exemplarily evaluating the simulators' performance, we attempted to establish one and the same type of back-propagation network for optical character recognition (OCR). A respective quality statement is made by comparing the number of cycles needed for training and the recognition rate of the individual simulators.

Keywords: neural networks, neural network simulation, character recognition, connectionism, computer vision, artificial intelligence

CONTENTS:

1 INTRODUCTION

Over the last years, a lot of research has been done in developing models of neural networks to solve knowledge and recognition problems in reasonable time. Researchers have begun to investigate massively parallel architectures to overcome the problems in conventional symbolic computation. Although parallel processors offer attractive mechanisms to implement large artificial neural networks, most of the models are implemented using simulators on conventional s ingle-processor machines. Many working groups have developed simulation packages providing flexible and standardized tools for designing and testing artificial neural networks. They provide mechanism for creating cells, for their connection, and for defining respective learning rules, activation functions as well as output functions. Additionally, libraries facilitate updating of a net, while graphical interfaces describe intermediate activations and allow user interaction.

In this report, we present the results of testing five artificial neural network simulators. The comparison is outcome of evaluating the applicability of artificial neural nets in optical character recognition. Due to the lack of scoring publicly and commercialy available network software, in a first step, we concentrate on simulators which are free of charge and can be received by anonymous FTP via the Arpa Internet from the corresponding developers or distributors. The simulators tested are:

- **PlaNet**, Version 5.6 [1]
  University of Colorado at Boulder, USA
- **Pygmalion**, Version 2.0 [2]
  Computer Science Department of the University College London, Great Britain
- **RCS (Rochester Connectionist Simulator)**, Version 4.2 [3]
  University of Rochester, NY, USA
- **SNNS (Stuttgart Neural Net Simulator)**, Versions 1.3 and 2.0 [4,5]
  IPVR, University of Stuttgart, Germany

There are various other simulators of similar functionality. In our considerations, we selected free available simulators running on Sun SparcStations under X Window System.

In Section 2, we first give brief descriptions of the individual simulators tested, their components, and general characteristics. In Section 3, results of comparing the simulators' functionality are given. In particular, number of standard functions, hard- and software requirements, and experiences during network design and run of the simulators are reported. Section 4 first describes dimensions and topology of a feed-forward-net taken as a common basis for testing the performance of all simulators considered and then shows individual results of the test. In Section 5, a final discussion is given including a summary description of all simulators in test as well as general recommendations.

2 SIMULATOR SET

2.1 PlaNet

The PlaNet simulator has been developed at the University of Colorado at Boulder which has a long history in developing artificial neural network simulators. The version available for

our test was version 5.6 from January 1991. Other earlier simulators developed at Boulder are StarNet, SunNet, and XNet designed for other Unix environments or other computers. The designers of PlaNet describe their simulator as a tool for constructing, running and examining PDP (parallel distributed processing) or connectionist networks.

The respective software environment is installed using a special *Tutorial* routine. This routine further gives an introduction of how the program should be started and goes through an example session. While working with PlaNet this session can be started in any state just calling the *Tutorial*. Installation time for the whole system is about 20 minutes on a SparcStation. It ends by sending electronic mail to Boulder to confirm a correct installation.

The PlaNet system is composed of two major components:

- the simulator kernel with the "PlaNet" programming language and
- the PlaNet interface, a Unix shell and a graphical window, to run and examine the simulator.

(A) Elements of PlaNet

A connectionist network designed by PlaNet consists of *units* and *connections*. The units are grouped in layers. The definition of a layer generates the respective units. The number of units may be changed dynamically by parameters when starting the net. Each unit is identified by the name of its layer and its number.

The connections between layers or units are equivalent to links in other simulators. A connectionist network may have 64 connections, each of them identified by an individual name. This limit seems to be a weakness when using coarse coding (screening) to connect only parts of two layers. However, the special construct *weight matrix* allows the definition of complex connecting structures, requiring only one single connection. Connections between source and target units can so be defined as *existent* or *non-existent*.

(B) Net Functions and Training

Updating (activating) unit values can be done layer by layer and is described by procedures in the net-program respectively. PlaNet provides a set of special high-level standard operations (e.g. for activation and learning), but also allows an easy definition of new operations and procedures for special applications.

After loading a network, the connections were initialized by a default value of 0.5. Changing this value is possible by appropriate parameter settings. For backpropagation learning, the parameters *learning rate* (η) and *momentum* (α) are used. As reported in the manual, possible values are $\eta = 0.2$ and $\alpha = 0.4$.

Input and target patterns to be given to the network are stored in a file which is called *pattern specification file*. There are two different formats to describe patterns:

- a special digit representation. Two strings composed of digits from '0' ... '9' and the character 'a' represent input and corresponding target patterns. Usually, '0' corresponds to value 0.0 (default *min*), '1' to value 0.1, .. , and 'a' to value 1.0 (default *max*).
- a floating-point representation.

(C) PlaNet Interface

After opening a conventional X-Window, about 50 special commands may be entered to interact with PlaNet. The entire simulation runs in the background. The shell window is also used as output device. There exists a graphical window for displaying various views of the network, layers or connections. Various static pictures can be mounted and combined to a movie for presentation. Training states can be saved in a file to rerun the network later on at characteristic states of knowledge.

The documentations *User's Guide* and *Reference Manual* [1] give a brief, but thorough introduction to the PlaNet system and its functionality. Commands are described by their syntax and respective examples. Additionally, an online helpsystem is integrated into the PlaNet package which is easy to handle and very useful while working with PlaNet.

2.2 Pygmalion

The Pygmalion simulator is a result of an ESPRIT II project, supported by the European Community. Companies and universities involved in the project were Thomson-CSF/DSE, Mimetics S.A., CSELT, CTI, ENS, INESC, LRI, Philips, SEL and UCL.

To run the Pygmalion simulator, Unix and X Window System are needed. The simulator has been tested for various computers such as Sun Sparcstation or DECstations. An installation script allows an easy installation of the simulator. This job takes about 30 minutes on a Sparc-Station.

The ESPRIT project was dropped in 1991 after making Version 2.0 available. The University College London (UCL) has co-ordinated the development. Currently, there is no further support given by UCL (questions send via email were not answered).

The Pygmalion simulator consists of three main components:

- the simulator kernel holds the connectionist net and the functions to work with.
- a programming language called *nC* provides constructs for designing connectionist nets and specific functions for its application. nC is a Pygmalion extension of the programming language C*.
- the Graphic Monitor is the graphical interface of the system for visualizing and running the simulator.

(A) Elements of Pygmalion

In Pygmalion a connectionist net has a hierarchical structure. The top level layers are composed of so-called *clusters*. Clusters again consist of *neurons* which contain *synapses*.

---

* There is also a programming language N which has been developed by Thomson-CSF/DSE and Mimetics S.A.

Neurons correspond to units and synapses to links respectively. The states of a net are represented by the data of the *neuron states* and the *synaptic weights*.

Besides the general feature to consist of input layer, hidden layer and output layer, a Pygmalion net may also be composed of various heterogeneous subnets. All data as well as the net topology are administrated by the Pygmalion simulator at the system level.

(B) Net Functions and Training

When starting the Pygmalion system, a special slot allows for a specification of the hardware on which the simulator should run. This leads to an improved performance when running large connectionist networks. For training, there is only one backpropagation learning algorithm available.

There are two online learning modes available to train a net:

i. the mode *step learn* makes one training cycle with the given pattern.

ii. the mode *learn* trains a pattern many cycles until the error is kept within a given tolerance.

For training more than one pattern a time, a special batch learning tool is contained in the system, but it did not run on our equipment. Patterns may be described in a file by five alternative formats. A pattern file may contain 25 different patterns. If there are more patterns in one file, an error message appears to tell the user that there might be problems.

(C) Graphic Monitor

The Graphic Monitor is the user interface in Pygmalion. It is an user-friendly tool for displaying connectionist nets and working with them. In various windows different components of a net can be visualized in many ways.

The documentation includes the *Installations Manual*, *Pattern Descriptor Files Manual* and the *Graphic Monitor Tutorial*. Other documents are referenced but have not been accessable. The online help system integrated into the simulator gives only very little (or no) support.

2.3 Rochester Connectionist Simulator (RCS)

This simulator has been developed at the University of Rochester, New York. It is the classical system for artificial neural network simulation. It is elaborated as a tool to aid specification, construction and simulation of connectionist networks. In our considerations we use the version 4.2 from October 1989. The simulator may be installed on an Unix computer, but also on a Macintosh. Originally, the simulator was implemented in Lisp, but now is also available in C. Because of several bugs, we needed more than one installation cycle. However, the installation is easy and takes about 45 minutes.

The RCS is composed of two major components:

- the simulator kernel for generating and running connectionist networks. A specialized backpropagation module (BP-Module) provides the capability to operate with many layered backpropagation nets.
- two graphical interfaces: the XGI interface working on a X Window System and the GI interface which runs on other platforms.

(A) Elements of RCS

A RCS connectionist network consists of *units*, *links* and *sites*. Units are simple computational elements which communicate by sending their level of activation via links to other elements (units or sites). There are various types of units provided by the BP-Module, i.e. input, hidden, output, teach as well as fire units. Input- and teach units can be manipulated from outside.

When generating a connectionist net, a *learnsite* is added to the hidden- and output units. It may be considered as a library function which can be redefined by the user.

Links are directed connections between two units, which are initialized during generation. Optional sites can be added to units. In this way sites are preprocessing elements. Therefore, links have no relation to the corresponding unit.

(B) Net Functions and Training

There are three different modes for computing the values of a unit:

- *Synchronous*. Every unit is updated at each simulation step. The output value from a preordered step is used to calculate the actual unit value. This mode is set as a default.
- *Asynchronous*. All or only few of the units are updated at each step [3]
- *Delay*. There is an option for generating a *delay simulator* with a special type of links. Defining the respective delay time (value), the update operation is maintained for some time.

Several update or learning functions are implemented. In addition user-defined functions may be included into the simulator library. The values of the links will be initialized during network generation. A speciality of the RCS is the allocation of data space for the units, which may be changed dynamically.

There is no tool for handling data files to initialize input and teach units. The user must take care of this job on his own and add his own procedures (functions) to the netprogram.

(C) Graphical User Interface GI and XGI

The XGI interface is an extension of the GI interface. XGI uses the X-Window system and is used to operate with the RCS. The *XGI Control Window* is used for building, training and displaying states of the connectionist network. Instructions can be entered by typing or invoking with mouse clicks when noted in a panel.

The documentation *Technical Report 233* [3] is very substantial. More information is given in a *Simulator User Manual* and in an *Advanced Programming Manual* where each instruction

is described twice. Nevertheless, an index would be very useful to read the documentation. The online helpsystem gives only little support.

2.4 Stuttgart Neural Net Simulator (SNNS)

The SNNS has been developed at the Institute for Parallel and Distributed High Performance Systems (IPVR) at the University of Stuttgart, Germany. For our benchmark test, we have considered versions 1.3 and 2.0 of this simulator.

The SNNS is available for various computer architectures which run Unix and X Window System. During installation makefiles for the underlying machine architecture and window system are generated. Sometimes, compile options have to be changed manually because of wrong parameters for optimization. A complete installation of SNNS takes about 30 minutes.

SNNS simulator is a successor to an earlier neural network simulator called NetSim, which uses many ideas of the Rochester Connectionist Simulator.

The Simulator consists of three major components:

- The simulator kernel for designing and running neural nets, for memory management and for providing all operation functions.
- The Nessus compiler, a programming language for generating medium large nets up to about 10^6 units.
- The XGUI graphical user interface which is related to the simulator kernel and the X-Window system for interaction and displaying.

(A) Elements of SNNS

In the SNNS notation, a network of artificial neurons consists of *units*, *links* and *optional sites*. Units are the real computation elements of the net. Usually, there are three different types of units: input units, output units and the hidden units. Other, more specialized unit types are described in [5] Units may be distinguished by number or name. With a parameter called *io-type* the units can be grouped in input, hidden or output layer. Other parameters capture activation value, initial activation, output value, bias value, activation function, output function etc. Additional parameters such as *position* or *subnet* are used for visualization in the XGUI.

Links are directed connections between two units. The direction designates the orientation of the activation transfer. Recursive connections are possible but redundant connections are prohibited. Each link has a *weight* assigned to it. Negative values describe inhibitory connections and positive values excitatory connections respectively. Sites can be compared with dentrites. A unit having a site relation has no direct incoming connection from other units.

(B) Net Functions and Training

When running the SNNS on a single-processor workstation the activation values of the units must be computed in some sequential order. This order is defined in a socalled *update mode*. There are five different update modes implemented:

- *Synchronous.* All units change their activation after each step. The new activation values are computed in an arbitrary order. After the new activation values have been computed, the new output of the units is calculated. This looks like all units fired simultaneously.
- *Random permutation.* The units compute their activation and output sequentially. The order is defined randomly, but each unit is selected only once in a cycle (step).
- *Random.* Same as random permutation, but there is no guaranty for a single updating of units. Instead, units may keep their value or can be updated several times during a cycle.
- *Serial.* The updating order is defined by the ascending unit number specified during net generation. This is the fastest mode.
- *Topological.* The simulator sorts the units with respect to their topology. The activation is computed from input to output. This mode provides good performance for feed-forward nets because many units will reach very early their final output activation, and will not be modified in a later stage.

These modes only are applied to the forward propagation phase from input to output. The backward phase in learning procedures such as backpropagation is not affected.

An important aspect of training an artificial neural network is the question of how to adjust the weights of the links to get the desired system behaviour.

In the SNNS, the weights are initalized with random values. The number of cycles needed is influenced by the initial weight value. The learning algorithms are based on the *Hebb-rule*, the *delta-rule* and the *generalized delta-rule*. Several activation functions are implemented for units and links. In Version 2.0, there are five alternative learning algorithms:

- *Vanilla-Backpropagation*,
- *Backpropagation with momentum*,
- *Quickpropagation*,
- *Counterpropagation* and
- *Backpercolation*.

(C) Programming Language Nessus

Nessus stands for '**Ne**tzwerk **S**pezifikations **S**prache der **U**niversität **S**tuttgart'. It is a tool to describe connectionist nets, their topology and dynamic behaviour at a high level. The compiler generates a netfile with all specifications needed as input for the simulator kernel. Because of bugs in this tool, there is no further support from Stuttgart. Therefore, Version 2.0 does not include Nessus. For generating connectionist networks, other tools have been developed, i.e. *net editor* and *bignet*.

(D) Graphical User Interface XGUI

The user communicates via XGUI with the simulator kernel. It provides mechanism for generating, training, manipulating as well as visualizing a net. Furthermore, generation and

manipulation of patternfiles for net input is possible. Net states are shown in net displays, while error report is displayed in the shell window from which the SNNS has been started. To operate with SNNS there are several pull-up and pull-down windows described in [4,5].

The initial data for input and teach units are stored in file and read into memory to train or test the net. The data may be either integer or floating point values.

In the SNNS *User Manual* [5] all components, functions and instructions are described very clearly. Moreover, the [4] gives a brief overview about connectionist networks in general. The integrated online help system is very useful and gives good support to work with the simulator.

3 FUNCTIONAL COMPARISON OF THE SIMULATORS

For comparing the simulators' functionality, we consider the number as well as the spectrum of facilities they offer for generating and working with connectionist networks. For a comparison concerning performance, we selected a connectionist network running for all simulators. Thus, one and the same topology and underlying functionality provide a common basis for all simulators in test.

Most of the information taken for comparison is extracted from the simulators' manuals and reports [1,2,3,4,5]. Other information concerns our experience while working with the simulators. The results will be presented in tables.

(A) Simulator Functions

Table 1 shows the number of implemented standard functions. These are functions for updating, learning and initializing the net, but also output and site functions.

	PlaNet V.5.6	Pygmalion V.2.0	RCS V.4.2	SNNS V.1.3	SNNS V.2.0
update function	1*	?	3	5	6
learn function	3*	1	2*	2	5
init function	2	1	1	1	2
activation function	1*	?	1*	11	11
output function	2*	?	1*	4	4
site-function	0	?	9	5	5

Table 1: Number of standard functions provided by the simulators.

* More functions can be implemented and easily applied by the user.

(B) Hard- and Software Requirements

Table 2 captures requirements of the simulators to the Hard- and Software on which they should run. In particular, the required operating system and graphical system, the implementation language as well as the ability of the simulators for computer networks and for different CPUs are reported.

	PlaNet V.5.6	Pygmalion V.2.0	RCS V.4.2	SNNS V.1.3	SNNS V.2.0
operating system	Unix	Unix	Unix	Unix	Unix
graphical system	X Window System, Sunview	X Window System	X Window System	X Window System	X Window System
source code	C	C	C	C	C
networking ability	no	yes	no	no	no
alternative CPUs	yes	yes	yes	yes	yes

Table 2: Simulators requirements in hard- and software.

(C) Net Design

Table 3 lists possibilities for a user to design and generate connectionist networks. These are the programming languages used, the existence of a net editor, and the number of possibilities for generating nets.

	PlaNet V.5.6	Pygmalion V.2.0	RCS V.4.2	SNNS V.1.3	SNNS V.2.0
programming language	PlaNet	nC, N*	C, Common Lisp, Scheme**	Nessus	—
net editor	no	yes	yes	yes	yes
net gen. pos.	1	2	2	3	3

Table 3: Benchmarks for design connectionist networks.

(D) Acting with the Simulators

Table 4 contains a number of system services for working with the simulators. Here, we focus on the operating modes, the occurrence of a graphical interface, and the mode of output for controlling the network behaviour.

---

* The programming language N is distributed by Thomson CSF and Mimetics S.A. only free to partners of Pygmalion ESPRIT 2059 and Galatea ESPRIT 5293 projects

** All languages can be used when available on the system.

	PlaNet V.5.6	Pygmalion V.2.0	RCS V.4.2	SNNS V.1.3	SNNS V.2.0
operating mode	system shell	graphical interface	graphical interface	graphical interface	graphical interface
handling	easy with system shell	easy with graphical interface	easy with graphical interface	easy with graphical interface	easy with graphical interface
graphical interface	yes	yes	yes	yes	yes
purpose of graphical interface	net visualization	net visualization and operating	net visualization and operating	net visualization and operating	net visualization and operating
output of net states	system shell, graphical interface	graphical interface	graphical interface	system shell, graphical interface	system shell, graphical interface
range of output	potentials of units and links learning curves, etc.	potentials of units and links	potentials of units	potentials of units and links	potentials of units and links

Table 4: System services while running the simulators.

(E) Filesystem

Each of the simulators requires data to initialize the input and output units. The respective file functions provided by the simulators and further possibilities are listed in Table 5. These are input file formats, capabilities for file handling and filr saving as well as saving of the graphical output of a net. Note, for the RCS, there is no appropriate file input of patterns. The user defines his own file formats as *integer* or *float* for initializing the input units or teach units.

	PlaNet V.5.6	Pygmalion V.2.0	RCS V.4.2	SNNS V.1.3	SNNS V.2.0
input file formats	2	4	—	2***	2
file handling	yes	yes	no	yes	yes
save net state	yes	yes (restricted)	yes	yes	yes
save graphical output	yes	no	yes	no	no

Table 5: System services while running the simulators.

(F) Experiences during Training Connectionist Nets

The entries of the Table 6 have been gained during the performance test. They indicate rendering of learning results, the subjective speed of learning as well as termination of learning in a local minimum.

*** There are several integer and floating-point representation formats for pattern file.

	PlaNet V.5.6	Pygmalion V.2.0	RCS V.4.2	SNNS V.1.3	SNNS V.2.0
rendering of learning	yes*	yes	yes*	yes*	yes*
subj. learning speed	very fast	slow	slow (average)	fast (average)	very fast
termination in local minima	no	no statement	no statement	yes	no

Table 6: Learning benchmarks.

(G) Further Important Criteria

In Table 7, we have determined additional benchmarks important to qualify the alternative simulators. These are the possibility for combining them with other applications, their fault tolerance, and their tolerance against errors in patterns as well as installation time, quality of documentation, and the support we got from the distributors.

	PlaNet V.5.6	Pygmalion V.2.0	RCS V.4.2	SNNS V.1.3	SNNS V.2.0
combination with other applications	yes	no statement	yes (relocateable simulator)	yes (interface functions)	yes (interface functions)
fault tolerance	very little	high (average)	average	very little	little
tolerance against pattern errors	little	little	little	little	little
installation-time	ca. 20 min.	ca. 30 min.	ca. 50 min.	ca. 40 min.	ca. 25 min.
quality of documentation	excellent	bad (not available)	good (very good)	good (old version)	excellent
support	not needed	no answer	no answer	excellent	excellent

Table 7: Additional criteria.

(H) Summary of Functional Benchmarks

Table 8 summarizes our experiences while working with and testing the simulators on a Sun SparcStation.

In particular, all of the functionality listed in aspects (A) to (G) are rated according to criteria excellent (++), good (+), average (o), bad (-), and worse (--). The high requirements of the Pygmalion simulator are caused by the various problems (i.e. time errors and simulator crashes) occuring while storing and loading of trained networks.

---

* Reproducing training (learning) results is not possible, because the presentation of learning pattern happens in random order.

	PlaNet V.5.6	Pygmalion V.2.0	RCS V.4.2	SNNS V.1.3	SNNS V.2.0
(A)	+	-	o	+	++
(B)	average	high	average	average	average
(C)	+	+	o	++	++
(D)	++	o	o (-)	++	++
(E)	++	--	-	+	+
(F)	++	-	-	o	++
(G)	++	o (-)	o	+	++

Table 8: Summary of functional benchmarks.

4 COMPARISON OF SIMULATOR PERFORMANCE

For performance testing, an artificial neural net, having the same topology and functions, is generated for all simulators. Therefore, a three-layered feed-forward-net using a backpropagation learning function for recognizing printed characters is established.

4.1 Dimension and Topology of the Test Net

The input layer consists of 375 units arranged in a 15x25 input matrix. The output layer is limited to 26 units for characters 'A' to 'Z' respectively. For the practical use of a respective net, number and connections of the hidden units is significant.

Literature describes that a factor of 1.5 to 2 times the number of input units may be necessary for hidden units in fully connected networks. This is justified by the tolerance against unknown patterns and the ability to adapt the net respectively. Training a corresponding net, having about 250 hidden units (factor 1.5) and about 225,000 links, is very time-consuming on a sequential computer. Even when reducing the number of hidden units to 100, there are 40.100 connections left to be updated at each cycle.

Several publications about character recognition using artificial neural nets [6,7,8] propose screening or coarse coding on input and hidden layers. This is similar to human perception where visual data is screened on the retina [9,10,11].

For our test net, only the input layer has been screened. All units of a column or a row have been connected to one hidden unit. Therefore, 40 hidden units are needed. They are fully connected to the output layer. Using this topology (375 input units, 40 hidden units and 26 output units), only 1.790 connections have to be updated. This can be done within acceptable time by a sequential computer. Additionally, the net topology has the strength to be invariant against rotations up to 15°. Figure 1 schemes the topology of our test net describing the different layers and their interconnection.

To test the simulators, we take one and the same set of patterns adapted to the corresponding file types. One pattern file consists of ideal binary images of printed characters (values 0 and 1 as unit activation). A second pattern file contains the same pattern but the

zero values are changed by random numbers with random values. A third file contains patterns being optically scanned and normalized to 15x25 image dots.

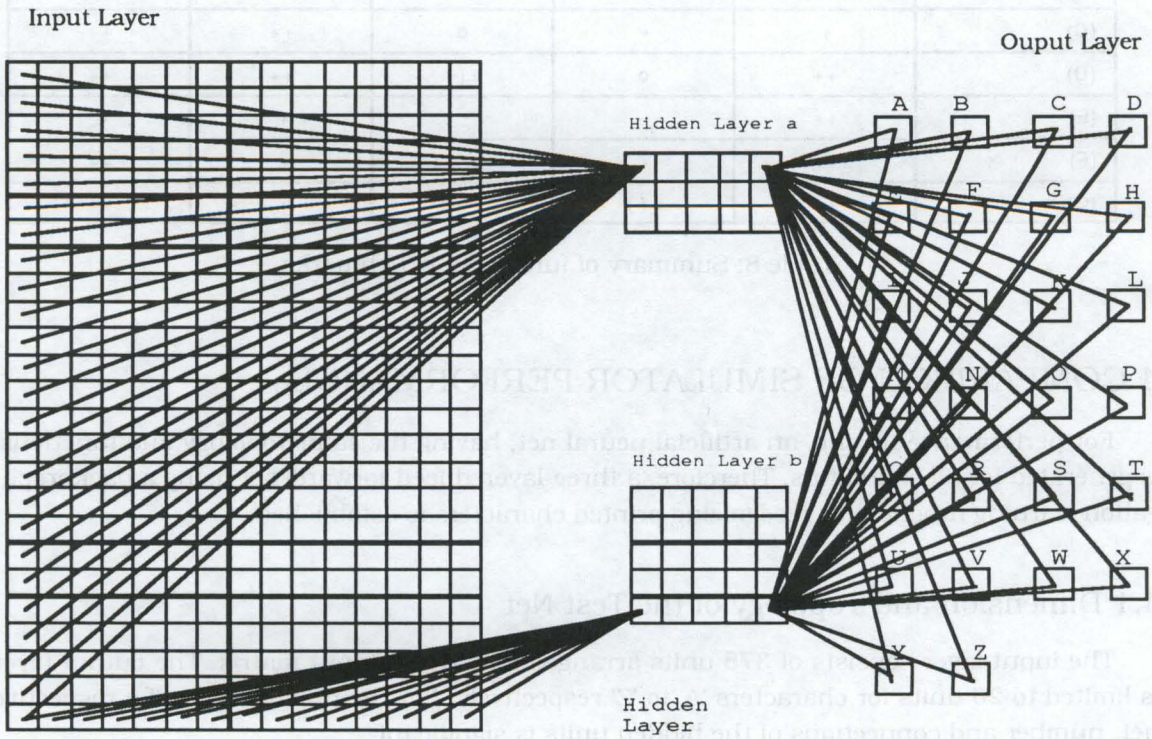

Figure 1: Topology of the test net for character recognition.

4.2 Simulator Benchmarks

While testing the given simulators with the net topology described above, Pygmalion and RCS failed for reasons discussed later in this paper. Statements about the performance of the simulators are based on the number of training cycles needed until a set of patterns was recognized. After 300, 500, and 550 cycles another pattern set was loaded, tested and then trained. Training finished after 650 cycles. At this point, all trained patterns (including small variations) can be well recognized.

The entries of Table 9 are the sum of the square distances between the teaching input and the real output. It is the average error for each output unit considering all input patterns. Because of the failure of Pygmalion and RCS, only PlaNet and SNNS results are shown. Figure 2 illustrates the error curves respectively.

after .. cycles	PlaNet V. 5.6	SNNS V. 1.3	SNNS V. 2.0
50	0.29175	0.33275	0.07931
100	0.13705	0.21871	0.01821
150	0.06591	0.15585	0.00789
200	0.03416	0.12235	0.00454
250	0.02306	0.10882	0.00296
300	0.00980	0.09126	0.00211
	0.11853	0.22364	0.07084
350	0.05790	0.16661	0.01017
400	0.03024	0.13448	0.00424
450	0.02621	0.12619	0.00252
500	0.02350	0.10613	0.00172
	0.01326	0.10053	0.00435
550	0.00496	0.07760	0.00156
	0.00995	0.11245	0.00548
600	0.00647	0.07211	0.00282
650	0.00426	0.06604	0.00140

Table 9: Error table for the simulators during net training.

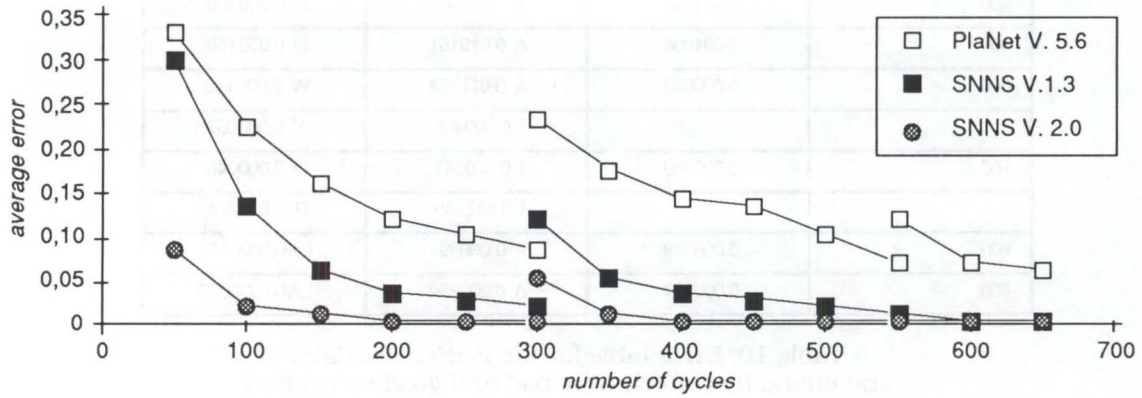

Figure 2: Diagram of error values during training of the net.

Please note, that for obtaining the average error of a new pattern set to be learned, the patterns must be trained for one cycle. This fact leads to the crack points shown in the diagram of Figure 2.

In the following, the performance of the individual simulators are discussed in more detail. In particular, the average error after a certain number of cycles and the characters are reported which are recognized best and worse (or not) by the simulators. All patterns tested were presented in a random order in each cycle.

At first, the ideal printed character set is learned for 300 cycles. In the next step, the optically scanned pattern set is tested and trained. Consequently, after 500 cycles the ideal pat-

tern set is presented again to the net and trained 50 cycles. Finally, the ideal character set with a noisy pattern is loaded and trained. This procedure was applied to all simulators in test (except Pygmalion).

(A) PlaNet

The topology of PlaNet network has the structure as described above. The *learning rate* parameter was initialized by $\eta = 0.2$ and *momentum* by $\alpha = 0.4$ for training the net. Table 10 summarizes the results of the test. The average error designated by the entries in the tables is calculated by averaging the squared error across all output units as well as across all pattern.

after .. cycles	average error	pattern bad recognized	pattern good recognized
50	0.011221	I 0.020758	W 0.001394
100	0.005271	I 0.019649	V 0.000543
150	0.002535	I 0.019265	V 0.000254
200	0.001314	I 0.018766	V 0.000210
250	0.000887	I 0.007742	K 0.000218
300	0.000377	I 0.001365	V 0.000106
		P 0.031288	L 0.000682
350	0.002227	A 0.019335	H 0.000357
400	0.001163	A 0.019302	H 0.000206
450	0.001008	A 0.019161	H 0.000150
500	0.000904	A 0.017764	W 0.000113
		I 0.009405	Y 0.000057
550	0.000189	I 0.000541	U 0.000046
		I 0.011945	D 0.000274
600	0.000249	A 0.000398	L,M 0.000109
650	0.000164	A 0.000259	L,M 0.000079

Table 10: Error table for the PlaNet simulator
and characters which were bad and good recognized.

(B) SNNS

For testing SNNS, there are no structural modifications of the connectionist network. So, it is identical to the PlaNet network. Note that Version 1.3 of SNNS only provides the *Vanilla-Algorithm* for backpropagation learning with one parameter (initialized by a value of $\eta = 0.2$). The errors reported in the Table 11 and 12 correspond to the sum of the square distances between the teaching input and the real output, overall output units summed over the number of all patterns presented. To directly compare these values to those of the PlaNet test, they have to be divided by the number of characters, i.e. 26.

after .. cycles	average error	pattern bad recognized	pattern good recognized
50	0.33275	H,K,O,Q,R *	Y 0.231
100	0.21871	H,O,Q,R *	Y 0.140
150	0.15585	H,O,Q,R *	Y 0.112
200	0.12235	H,O,Q,R *	Y 0.094
250	0.10882	H,O,Q,R *	Y 0.085
300	0.09126	H,O,Q,R *	V 0.079
	0.22364	A,C,H,N,O,Q ,R*	X 0.253
350	0.16661	A,C,H,N,O,Q *	L 0.117
400	0.13448	A,C,H,N,O,Q *	L 0.092
450	0.12619	A,C,H,N,O,Q *	L 0.078
500	0.10613	A,H,O,Q *	L 0.068
	0.10053	H,O,Q,R *	C 0.013
550	0.07760	H,O,Q *	J 0.049
	0.11245	H,O,Q *	S 0.128
600	0.07211	H,O,Q *	T 0.079
650	0.06604	H,O,Q *	T 0.066

Table 11: Error table for the SNNS simulator, Version 1.3, and characters which were recognized as bad and good .

after .. cycles	average error	pattern bad recognized	pattern good recognized
50	0.07931	E 0.759	X 0.156
100	0.01821	E 0.366	W,X 0.091
150	0.00789	E 0.195	W 0.064
200	0.00454	E 0.152	W 0.050
250	0.00296	E 0.131	W 0.041
300	0.00211	E 0.107	W 0.035
	0.07084	A 1.000	T 0.047
350	0.01017	O 0.152	T 0.0.53
400	0.00424	U 0.102	T 0,044
450	0.00252	U 0.080	T 0.038
500	0.00172	U 0.067	T 0.034
	0.00435	A 0.183	P 0.011
550	0.00156	E 0.096	V 0.013
	0.00548	S 0.989	M 0.063
600	0.00282	E 0.097	M 0.053
650	0.00140	E 0.072	V 0.023

Table 12: Error table for the SNNS simulator, Version 2.0, and characters which were recognized as bad and good

Note, characters labeled by a '*' could not be classified (recognized) correctly, e.g. H,O,Q,R after 100 cycles. However, to recognize these characters some additional hundred training cycles are needed. Consequently, all patterns were recognized correctly.

For training Version 2.0, another backpropagation algorithm with *momentum* (μ) and *flat-spot-elimination* (c) parameters is implemented. Initial values are $\eta = 0.2$, $\mu = 0.4$, and $c = 0.1$.

As reported in Table 12, the average error decreases faster than with the *backpropagation* algorithm of version 1.3. All characters were recognized by nearly the same accuracy.

(C) RCS

For testing RCS, the structure of the net must be modified. A second hidden layer with 26 units was added. This additional layer is fully connected to the first hidden layer. Each unit of the second hidden layer is connected to one corresponding output unit. Thus, the network consists of 467 units and 1,816 links.

Another modification concerns the procedure of how patterns are presented to the net. Because RCS does not support any file handling for file input, the test patterns cannot be presented in a random order. An implemented test function similar to the examples given by the RCS did not work for the test net. Therefore, it was impossible to get results of recognition rates for the patterns (characters). However, we have not been able to visualize changes in potential for units of the first hidden layer. Therefore, no exact statements concerning learning cycles are needed and recognition can be made — approximately about 150 - 200 cycles longer than PlaNet or SNNS.

(D) Pygmalion

The Pygmalion only allows the design of fully connected structures. Therefore, it was impossible to generate a screened network. Perhaps other structures can be generated, but there is no document available which gives any information about this.

However, two variants of connectionist networks were generated for our test:

i. input and output like in the other networks, but 80 hidden units

ii. input and output like in the other networks, but 120 hidden units with a fully connected structure.

This implies many more connections to update each cycle and thus, more cpu-time for training.

Although patterns are generated as described in the documentation, an error message is displayed when loading the patterns. It indicates that 26 rows of data are expected as input, but only 25 rows are in the patternfile. Another error message appears if the pattern file contains more than 25 patterns. These messages are ignored because no damage could be detected. Thus, every pattern must be trained on its own. Therefore, the simulator needs further 100 to 200 cycles longer than RCS to learn one pattern.

A batch training tool is integrated into the Pygmalion simulator, but because of an unidentifiable bug, this tool could not be used. As reported in the Pygmalion documentation, there are thousands of cycles necessary for training a net by an entire set of patterns. Moreover, the trained (partial trained) net could not be saved for a later reuse. Time out errors occur when saving and loading the net. Thus, a correct testing of this simulator was impossible.

5 DISCUSSION

For the Pygmalion simulator, it is not possible to give any recommendation because of the problems occuring while testing the simulator and the additional lack in support from the developers. The simulator might be suitable for small connectionist networks. Perhaps the underlying hardware may influence the performance. Unfortunately, we had no possibility to test the simulator on other platforms. For the Pygmalion Version 2.0, the functionality of Version 1.x is only partially implemented.

The Rochester Connectionist Simulator is one of the first connectionist simulators implemented. It is the *standard simulator* in general. It allows defining network functions in C code. In this test, we could not figure out why the RCS needs a second hidden layer to show output values while training the net. We checked all possibilities described to fit this problem, but we had no success. Another issue for criticism is the lack of file functions to handle patterns for input and teach unit initialization.

The PlaNet simulator brags with excellent learning speed. There are only few library functions for updating and learning available. Further functions can be easily written in the Netfile of PlaNet. However, these functions can be specifically used for the net in which they are defined. Designing large connectionist nets, of a complex topology, is a very hard job. Here, some additional support by the system could be given. An integration of the simulator or a connectionist network into other user applications is not easy because documentation does not give details. Generally, PlaNet is a flexible tool which allows user-implemented net functions.

SNNS version 1.3 needs many learning cycles to train a connectionist network. Version 2.0 provides more learning functions. Here, the learning speed is nearly equivalent to PlaNet. The set of functions to operate with SNNS (Version 2.0) is the largest one of the tested simulators. Moreover, user-defined functions in C code can be defined and linked to the system library. There are many different net types supported. With integrated tools for generating nets it is easy to build even complex networks. All interface functions are described in the documentation in an exact manner to be used in other applications.

When changing the network topology, the net must be redesigned in all simulators. There is no possibility in modifying the topology dynamically during runtime. Here, only SNNS supports dynamic change of learning as well as update functions during training and working phase. The other simulators use functions which are described in the Netfiles. Thus, there is no possibility for changing the funtions.

All in all, the simulators tested seem to be valuable for networks of small and middle size. For instance, SNNS may be used for networks up to 10^4 units and 10^6 links, e.g., it requires about 230 MB memory for a net having 10^4 units and $2*10^7$ links.

As a conclusion, we can state, that the performance of the simulators may change with the net structure. Nevertheless, as a result of our comparison, we can only recommend PlaNet and SNNS. The better equipment and faster learning seems to make SNNS the better choice. In September 1991 the SNNS simulator, Version1.3 was awarded the 'German Federal Research Software Prize 1991' by the German Federal Ministry of Science and Education.

REFERENCES

[1] Miyata, Y.: "A User's Guide to PlaNet Version 5.6", The Reference Manual for PlaNet Version 5.6; University of Colorado, Boulder, Computer Science Department, 1991.

[2] Hewetson, M.; et.al.: Pygmalion Neurocomputing Tutorials, University College London, 1991.

[3] Goddard, N.H.; et.al.: "Rochester Connectionist Simulator Technical Report 233", University of Rochester, Computer Science, 1989.

[4] Zell, A.; et.al.: "SNNS Benutzerhandbuch", Bericht Nr. 1/91, SNNS Nessus Handbuch, Bericht Nr. 3/91; Universität Stuttgart Fachbereich Informatik, 1991.

[5] Zell, A.; et.al.: "SNNS User Manual", Version 2.0, Report No. 3/92, Universität Stuttgart Fachbereich Informatik, 1991.

[6] LeCun,Y.; et.al.: "Handwritten Digit Recognition with a Back-Propagation Network"; Neural Information Processing System Volume 2, Morgan Kaufmann, 1990.

[7] Fukushima, K.; Miyake, S.; Ito, T.: Neocognitron: "A Neural Network Model for a Mechanism of Visual Pattern Recognition", IEEE Transaction on System, Man and Cybernetics 13, 1983.

[8] McClelland, J.L.; Rumelhard, D.: "Parallel Distributed Processing", Volume 1 - 3, MIT press, Cambridge, Massachusetts, 1988.

[9] Poggio, Tomaso; Koch, Christof: "Synapses That Compute Motion"; Scientific American 5/87, p 42-48.

[10] Hubel, David H.; Wiesel, Torsten N.: "Brain Mechanisms of Vision"; Scientific American 9/79, p 130-144.

[11] Frisby, John P.: "Sehen", Bertelsmann Club , 1979.

[12] Lutzy, Ottmar; Dengel, Andreas: "A Comparison of Simulators for Artificial Neural Nets", IEEE Expert, 1993.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
D-6750 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-25
Franz Schmalhofer, Ralf Bergmann, Otto Kühn, Gabriele Schmidt: Using integrated knowledge acquisition to prepare sophisticated expert plans for their re-use in novel situations
12 pages

RR-92-26
Franz Schmalhofer, Thomas Reinartz, Bidjan Tschaitschian: Intelligent documentation as a catalyst for developing cooperative knowledge-based systems
16 pages

RR-92-27
Franz Schmalhofer, Jörg Thoben: The model-based construction of a case-oriented expert system
18 pages

RR-92-29
Zhaohui Wu, Ansgar Bernardi, Christoph Klauck: Skeletel Plans Reuse: A Restricted Conceptual Graph Classification Approach
13 pages

RR-92-30
Rolf Backofen, Gert Smolka:
A Complete and Recursive Feature Theory
32 pages

RR-92-31
Wolfgang Wahlster:
Automatic Design of Multimodal Presentations
17 pages

RR-92-33
Franz Baader: Unification Theory
22 pages

RR-92-34
Philipp Hanschke: Terminological Reasoning and Partial Inductive Definitions
23 pages

RR-92-35
Manfred Meyer:
Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment
18 pages

RR-92-36
Franz Baader, Philipp Hanschke:
Extensions of Concept Languages for a Mechanical Engineering Application
15 pages

RR-92-37
Philipp Hanschke: Specifying Role Interaction in Concept Languages
26 pages

RR-92-38
Philipp Hanschke, Manfred Meyer:
An Alternative to H-Subsumption Based on Terminological Reasoning
9 pages

RR-92-40
Philipp Hanschke, Knut Hinkelmann: Combining Terminological and Rule-based Reasoning for Abstraction Processes
17 pages

RR-92-41
Andreas Lux: A Multi-Agent Approach towards Group Scheduling
32 pages

RR-92-42
John Nerbonne:
A Feature-Based Syntax/Semantics Interface
19 pages

RR-92-43
Christoph Klauck, Jakob Mauss: A Heuristic driven Parser for Attributed Node Labeled Graph Grammars and its Application to Feature Recognition in CIM
17 pages

RR-92-44
Thomas Rist, Elisabeth André: Incorporating Graphics Design and Realization into the Multimodal Presentation System WIP
15 pages

RR-92-45
Elisabeth André, Thomas Rist: The Design of Illustrated Documents as a Planning Task
21 pages

RR-92-46
Elisabeth André, Wolfgang Finkler, Winfried Graf, Thomas Rist, Anne Schauder, Wolfgang Wahlster: WIP: The Automatic Synthesis of Multimodal Presentations
19 pages

RR-92-47
Frank Bomarius: A Multi-Agent Approach towards Modeling Urban Traffic Scenarios
24 pages

RR-92-48
Bernhard Nebel, Jana Koehler: Plan Modifications versus Plan Generation: A Complexity-Theoretic Perspective
15 pages

RR-92-49
Christoph Klauck, Ralf Legleitner, Ansgar Bernardi: Heuristic Classification for Automated CAPP
15 pages

RR-92-50
Stephan Busemann: Generierung natürlicher Sprache
61 Seiten

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt: On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-53
Werner Stephan, Susanne Biundo: A New Logical Framework for Deductive Planning
15 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

RR-92-56
Armin Laux: Integrating a Modal Logic of Knowledge into Terminological Logics
34 pages

RR-92-58
Franz Baader, Bernhard Hollunder: How to Prefer More Specific Defaults in Terminological Default Logic
31 pages

RR-92-59
Karl Schlechta and David Makinson: On Principles and Problems of Defeasible Inheritance
13 pages

RR-92-60
Karl Schlechta: Defaults, Preorder Semantics and Circumscription
19 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist: Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi: An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit: GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-05
Franz Baader, Klaus Schulz: Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: On Skolemization in Constrained Logics
40 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: Concept Logics with Function Symbols
36 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer: COLAB: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-09
Philipp Hanschke, Jörg Würtz: Satisfiability of the Smallest Binary Program
8 Seiten

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf: Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-11
Bernhard Nebel, Hans-Juergen Buerckert: Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-12
Pierre Sablayrolles: A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-13
Franz Baader, Karl Schlechta: A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-14
Joachim Niehren, Andreas Podelski,Ralf Treinen: Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster: PLUS - Plan-based User Support Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz: Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-20
Franz Baader, Bernhard Hollunder: Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy: Comparative Study of Connectionist Simulators
20 pages

RR-93-24
Rainer Hoch, Andreas Dengel: Document Highlighting — Message Classification in Printed Business Letters
17 pages

DFKI Technical Memos

TM-91-12
Klaus Becker, Christoph Klauck, Johannes Schwagereit: FEAT-PATR: Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM
33 Seiten

TM-91-13
Knut Hinkelmann: Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter
16 pages

TM-91-14
Rainer Bleisinger, Rainer Hoch, Andreas Dengel: ODA-based modeling for document analysis
14 pages

TM-91-15
Stefan Busemann: Prototypical Concept Formation An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang: Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh: A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer: On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben: The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

TM-93-01
Otto Kühn, Andreas Birk: Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

DFKI Documents

D-92-13
Holger Peine: An Investigation of the Applicability of Terminological Reasoning to Application-Independent Software-Analysis
55 pages

D-92-14
Johannes Schwagereit: Integration von Graph-Grammatiken und Taxonomien zur Repräsentation von Features in CIM
98 Seiten

D-92-15
DFKI Wissenschaftlich-Technischer Jahresbericht 1991
130 Seiten

D-92-16
Judith Engelkamp (Hrsg.): Verzeichnis von Softwarekomponenten für natürlichsprachliche Systeme
189 Seiten

D-92-17
Elisabeth André, Robin Cohen, Winfried Graf, Bob Kass, Cécile Paris, Wolfgang Wahlster (Eds.): UM92: Third International Workshop on User Modeling, Proceedings
254 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-92-18
Klaus Becker: Verfahren der automatisierten Diagnose technischer Systeme
109 Seiten

D-92-19
Stefan Dittrich, Rainer Hoch: Automatische, Deskriptor-basierte Unterstützung der Dokumentanalyse zur Fokussierung und Klassifizierung von Geschäftsbriefen
107 Seiten

D-92-21
Anne Schauder: Incremental Syntactic Generation of Natural Language with Tree Adjoining Grammars
57 pages

D-92-22
Werner Stein: Indexing Principles for Relational Languages Applied to PROLOG Code Generation
80 pages

D-92-23
Michael Herfert: Parsen und Generieren der Prolog-artigen Syntax von RELFUN
51 Seiten

D-92-24
Jürgen Müller, Donald Steiner (Hrsg.): Kooperierende Agenten
78 Seiten

D-92-25
Martin Buchheit: Klassische Kommunikations- und Koordinationsmodelle
31 Seiten

D-92-26
Enno Tolzmann:
Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX
28 Seiten

D-92-27
Martin Harm, Knut Hinkelmann, Thomas Labisch: Integrating Top-down and Bottom-up Reasoning in COLAB
40 pages

D-92-28
Klaus-Peter Gores, Rainer Bleisinger: Ein Modell zur Repräsentation von Nachrichtentypen
56 Seiten

D-93-01
Philipp Hanschke, Thom Frühwirth: Terminological Reasoning with Constraint Handling Rules
12 pages

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter: User Manual of COKAM+
23 pages

D-93-03
Stephan Busemann, Karin Harbusch(Eds.):
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1992
194 Seiten

D-93-06
Jürgen Müller (Hrsg.):
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz Saarbrücken 29.-30. April 1993
235 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-07
Klaus-Peter Gores, Rainer Bleisinger:
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-08
Thomas Kieninger, Rainer Hoch: Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

Comparative Study of Connectionist Simulators

Andreas Dengel, Ottmar Lutzy

RR-93-23
Research Report