Evolutionary Design of Neural Architectures -- A Preliminary Taxonomy and Guide to Literature

Karthik Balakrishnan
Iowa State University

Vasant Honavar
Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/cs_techreports

Part of the Artificial Intelligence and Robotics Commons

Recommended Citation
http://lib.dr.iastate.edu/cs_techreports/26

This Article is brought to you for free and open access by the Computer Science at Digital Repository @ Iowa State University. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of Digital Repository @ Iowa State University. For more information, please contact digirep@iastate.edu.
EVOLUTIONARY DESIGN OF NEURAL ARCHITECTURES
— A Preliminary Taxonomy and Guide to Literature

Karthik Balakrishnan & Vasant Honavar
Artificial Intelligence Research Group
CS TR #95-01
January 1995
Abstract

This report briefly motivates current research on evolutionary design of neural architectures (EDNA) and presents a short overview of major research issues in this area. It also includes a preliminary taxonomy of research on EDNA and an extensive bibliography of publications on this topic. The taxonomy is an attempt to categorize current research on EDNA in terms of major research issues addressed and approaches pursued. It is our hope that this will help identify open research questions as well as promising directions for further research on EDNA. The report also includes an appendix that provides some suggestions for effective use of the electronic version of the bibliography.

1 Introduction

Artificial neural networks [Rumelhart & McClelland, 1986, Grossberg, 1988, Gallant, 1993, Kung, 1993, Haykin, 1994, Honavar & Uhr, 1994] offer an attractive paradigm for the design and analysis of adaptive, intelligent systems for a broad range of applications in artificial intelligence and cognitive modeling for a number of reasons including: potential for massively parallel computation, robustness in the presence of noise, resilience to the failure of components, amenability to adaptation and learning (by the modification of computational structures employed), and resemblance (albeit superficial) to biological neural networks (brains).
Despite much research activity in this area which has led to the discovery of several significant theoretical and empirical results and the development of important practical applications over the past decade(s), the design of artificial neural networks (ANN) for specific applications, under given sets of design constraints (for example, those dictated by particular computer technologies) is, to a large extent, a process of trial and error, relying mostly on past experience with similar applications. Furthermore, the performance (and cost) of ANN on particular problems is critically dependent, among other things, on the choice of primitives (neurons or processing elements), network architecture and the learning algorithm(s) used. For example, many of the popular learning algorithms used in ANN essentially search for a suitable setting of modifiable parameters (also called weights) within an otherwise a-priori specified network topology under guidance from sample inputs (training samples) from the task environment. Clearly, in order for this approach to succeed, the desired setting of parameters must in fact exist within the space being searched (which in turn is constrained by the choice of network topology) and the search algorithm used must in fact be able to find it [Honavar & Uhr, 1993]. Even when a suitable setting of parameters can be found using such an approach, the ability of the resulting network to generalize on data not seen during learning [Kung, 1993, Haykin, 1994] or the cost of the hardware realization of the network (as measured by its size, power consumption etc.) may be far from optimal. These factors make the process of ANN design difficult. In addition, the lack of sound design principles constitutes a major hurdle in the development of large scale ANN systems for a wide variety of practical problems.

Thus, techniques for automating the design of neural architectures for particular classes of problems under a wide variety of design and performance constraints are clearly of interest. Motivated by this, some researchers have recently begun to investigate constructive or generative neural network learning algorithms that extend the search for the desired input-output mapping to the space of appropriately constrained network topologies by incrementally constructing the required network [Gallant, 1993, Honavar & Uhr, 1993, Honavar, 1994]. Against this background, a logical next step is the exploration of more powerful techniques for efficiently searching the space of network architectures.

Thus, the design of neural architectures presents us with a challenging multi-criterion optimization problem, namely, that of searching for neural architectures (and learning algorithms) for particular classes of applications (e.g., 2-and-3-dimensional visual pattern classification, diagnosis, language processing, grammar inference, signal detection, sensor fusion, inductive and deductive inference, adaptive control, planning, robotics, etc). Evolutionary algorithms [Holland, 1975, Fogel et al, 1966, Goldberg, 1989, Koza, 1992, Michalewicz, 1992] offer an attractive and relatively efficient, randomized opportunistic approach to search for near-optimal solutions in a variety of problem domains. The design of efficient neural architectures for specific classes of problems under given sets of design and performance constraints is therefore a natural candidate for the application of evolutionary algorithms.
In addition to the development of new and powerful techniques for neural engineering, exploration of evolutionary approaches to the design of neural architectures is likely to shed light on important open problems in artificial intelligence, cognitive modeling, and biology — for, no theory of cognition can be complete without a detailed specification of the mechanisms that explain the evolution of structures and processes (computational or otherwise) that exhibit intelligence.

At an abstract level, the neural network design problem is essentially an instance of the program synthesis task not very different from that explored by researchers in artificial intelligence [Uhr, 1963, Uhr, 1973] and genetic programming [Koza, 1992, Fogel et al, 1966]. Such techniques offer an attractive (and often, the only) approach to finding solutions to complex problems where neither the shape (detailed structure) nor the size of the solutions is known in advance. What distinguishes evolutionary design of neural architectures from such general methods for program synthesis is the commitment to a particular class of computing structures, namely massively parallel, highly interconnected networks of relatively simple processing elements (or ANN) as opposed to, say, LISP programs. Given the equivalence of several models of computation (including lambda calculus of LISP programs, Turing machines, Post productions on which the knowledge-based expert systems are based), the choice of any given model is mostly a matter of convenience. Some models are better suited to expressing certain classes of computations than others and may have some cost/performance advantages over others given particular implementation media. It is well known that virtually any program can be realized by an artificial neural network (even a rather simple one such as that built from threshold logic units or McCulloch-Pitts neurons [McCulloch & Pitts, 1943]). Artificial neural networks represent a particularly attractive model of computation for a broad range of problems in artificial intelligence and cognitive modeling [Honavar & Uhr, 1994, Honavar, 1994, Honavar & Uhr, 1995]. Furthermore, the study of designs for such neural architectures provides a useful research strategy for exploring answers to many fundamental questions of interest in neuroscience including: What are the programs of the brain? How are they realized given specific sets of design and performance constraints? How does the brain get wired or programmed through learning or evolution to compute the things that it does?, etc.

This report is an attempt to put together a fairly up-to-date bibliography and a tentative taxonomy of current research on the evolutionary design of neural architectures. It does not provide a comprehensive review of the vast amount of research in this area. The reader is referred to [Weiss, 1990, Yao, 1993, Honavar, 1995, Balakrishnan & Honavar, 1995b] for such reviews. The primary purpose of the taxonomy presented here is to categorize current research in evolutionary design of neural networks in a manner that would be helpful in bringing out the similarities and differences among different approaches to problems in this area. It is our first attempt to identify some key research problems and approaches to their solution within a common framework (provided by the taxonomy). We feel that the time is ripe for such an undertaking given the vast number and diversity of recent research papers on this topic. It is our hope that this effort will help us (and
other researchers) identify and formulate open research questions as well as topics that need further exploration. Needless to say, given the fast-moving nature of this field, the taxonomy that is presented here would almost certainly evolve and get more refined as new research directions are explored.

2 Evolutionary Design of Neural Architectures

The key processes in the evolutionary approach to the design of neural architectures are shown in Figure 1.

![Diagram of evolutionary design process](Figure 1)

Evolutionary algorithms [Holland, 1975, Fogel et al, 1966, Goldberg, 1989, Koza, 1992, Michalewicz, 1992] are loosely modeled on processes that appear to be at work in biological evolution and the working of the immune systems. Central to such evolutionary systems is the idea of a population of *genotypes* that are elements of a high dimensional search space. For example, in simple genetic algorithms [Goldberg, 1989], *genotypes* are binary strings of some fixed length (say n) that code for points in an n-dimensional Boolean search space.

More generally, a genotype can be thought of as an arrangement (e.g., a string) of *genes*, where each gene takes on values from a suitably defined domain of values (also known as *alleles*). Each genotype encodes for typically one, but possibly a set of *phenotypes* or candidate solutions in the domain of interest — in our case, a class of neural architectures. Such encodings might employ genes that take on numeric values for a few parameters or complex symbol structures that are transformable into phenotypes (in this case, neural networks) by appropriate *decoding* processes. This decoding process...
may be extremely simple or fairly involved (as appears to be the case in mammalian brain development). These resulting neural networks (i.e., the phenotypes) may also be equipped with learning algorithms that train it using the environmental stimuli or simply evaluated on the given task (assuming the weights of the network are also determined by the encoding/decoding mechanism). This evaluation of a phenotype determines the fitness of the corresponding genotype.

The evolutionary procedure works on a population of such genotypes, preferentially selecting genotypes that code for high fitness phenotypes and reproducing them. Genetic operators such as mutation, crossover, inversion etc., are used to introduce variety into the population and to sample variants of candidate solutions represented within the current population. Thus, over several generations, the population gradually evolves toward genotypes that correspond to high fitness phenotypes.

Over the past decade, several researchers have investigated the use of evolutionary search techniques for automating the design of artificial neural networks. Most of these studies have been exploratory in nature and much systematic work is needed to identify the strengths and limitations of the various approaches. More importantly, sound scientific principles need to be discovered, to guide the development and application of evolutionary techniques in the design of artificial neural architectures for specific classes of problems (given a variety of design and performance constraints).

3 Towards a Taxonomy of Evolutionary Design of Neural Architectures

This section develops a preliminary taxonomy of research on evolutionary approaches to the design of artificial neural networks. The proposed taxonomy is suggested by what we see as the key problems explored and solutions proposed in a number of publications on this topic. In turn, the taxonomy provides a way of organizing the published literature on EDNA into a bibliography, that together with the taxonomy, provides a useful guide to literature on this topic. (See Appendix I for details on the organization and use of the bibliography).

3.1 Genotype Representation

The question of how a neural architecture is represented as a genotype is critical to the working of an evolutionary ANN design system. The representation or encoding used not only determines the classes of neural architectures that could possibly evolve but also constrains the choice of the decoding process. For example, if the problem at hand requires the discovery of neural networks with a recurrent structure, in order to ensure non-zero probability of success, the encoding scheme must be expressive enough to describe recurrent neural architectures, and the decoding mechanism must be capable of transforming such a description into an appropriate recurrent network (phenotype).
A broad range of choices (with varying degrees of efficiency, robustness, etc.) may exist for both the encoding and decoding schemes.

An examination of encoding and decoding schemes employed by numerous researchers in their work suggests that genotype representations may be broadly classified into two categories based on the decoding effort that is needed:

- **Direct Encodings** that require little effort to decode i.e., the transformation of the genotype into a phenotype is rather trivial. An example of such an encoding is a *connection matrix* that precisely and directly specifies the architecture of the corresponding neural network.

- **Indirect Encodings** that require considerable decoding effort in the construction of a phenotype. An example of such an encoding is one that uses rewrite rules to specify a set of construction rules that are recursively applied to yield the phenotype. Indirect encodings used in the literature fall into two broad categories — *grammatical* encodings (cellular grammars, graph grammars, genetic/biological grammars, geometric grammars etc) and *others* (genetic programming paradigm i.e., LISP programs, development models etc).

Other categorizations of encoding schemes are possible including: *deterministic* versus *stochastic*, *hierarchical* versus *non-hierarchical*, *modular* versus *non-modular*, etc. The interested reader is referred to [Balakrishnan & Honavar, 1995a] for a characterization of the properties of genetic representations of neural architectures.

### 3.2 Network Topology or Structure of the Phenotypes

It is well known that the success of a neural architecture in solving a particular problem (or class of problems) critically depends on the network topology (or the structure of the phenotype in our evolutionary design system). For example, a purely *feed-forward* neural network is incapable of discovering or responding to temporal dependencies in its environment; a *recurrent* network is needed for this task. Similarly, *non-linear* decision boundaries cannot be discovered by 1-layer networks (e.g., perceptrons); multi-layer perceptrons are required. It is therefore useful to classify work in evolutionary design of neural architectures based on the classes of neural network topologies that are evolved. Neural network topologies may be classified into two broad types:

- **feed-forward** networks (those without feedback loops); and

- **recurrent** networks (those with feedback loops).

Each of the two basic types of topologies may be further classified as networks that are — *multi-layered*, *strictly layered*, *randomly connected*, *locally connected*, *sparsely connected*, *fully connected*, *regular*, *irregular*, *modular*, *hierarchical* etc.

The choice of the target class of network structures dictates the choice of genetic representation. This is very much analogous to the choice of the *knowledge representation*
scheme for problem solving using state-space search in artificial intelligence. Thus, much can be gained in terms of search efficiency by restricting the search space based on a-priori knowledge about the properties of the solution set. In evolutionary design of neural architectures, one way to do this is to choose a sufficiently restricted class of network topologies based on knowledge of the problem domain. Without such bias, much of the search effort can be wasteful.

Additional requirements on the structure of the solution to a neural network design problem may be dictated by cost and/or performance considerations in the context of a particular technology. For example, a VLSI designer would prefer locally connected, fault-tolerant, highly modular neural network structures over globally connected, non-modular ones [Mead, 1989, Uhr, 1984, Honavar & Uhr, 1989, Kung, 1993].

Hence, network structure or topology, can be effectively used to classify different approaches to the evolutionary design of neural architectures.

3.3 Variables of Evolution

Neural architectures are typically specified in terms of the topology (or connectivity pattern), functions computed by the neurons (e.g., threshold, sigmoid, etc.) and the connection weights (or, a learning algorithm that sets the weights) [Rumelhart & McClelland, 1986, Kung, 1993, Gallant, 1993, Haykin, 1994]. A more complete description of a neural architecture requires the specification of coordination and control structures and learning structures (among other things) [Honavar, 1990, Uhr, 1990, Honavar & Uhr, 1990, Honavar, 1994, Honavar, 1994, Honavar, 1995]. Virtually any subset of these variables are candidates to be operated on by evolutionary processes. For example, a system \( A \) might evolve the network connectivity as well as the weights (while maintaining everything else constant) whereas a system \( B \) might only evolve the connectivity, relying on a perhaps more efficient local search for weights within each network. The time/performance tradeoffs for the two systems, on the given problem, will be different, making the choice of variables subjected to evolution, an extremely critical factor.

In addition to the network connectivity and the weights, one might evolve a learning algorithm, control or regulatory functions, the functions computed by various neurons, distribution of different types of neurons, relative densities of connections, parameters (and or processes) governing the decoding of a genotype into a phenotype, and so on. Thus, the variables subjected to evolution, form a useful classification scheme for work in the area of evolutionary design of neural architectures.

3.4 Application Domain

Much of the work on evolutionary design of neural architectures is driven by particular engineering applications (e.g., design of networks for pattern classification, control, robot navigation, sensor fusion, VLSI implementations of problem-specific neural architectures, etc.) and/or specific scientific hypothesis (e.g., explanations for evolution of
certain classes of modular organization or prevalence of particular network structures and functional units in mammalian brains across different species, etc.). In this context, attempts to understand and characterize to the extent possible, the relationship between properties of classes of application problems and those of the ANN used in its solution are of great theoretical and practical interest. Classification of evolutionary approaches to the design of neural architectures based on their focus on particular application domains is a step in this direction.

3.5 An Illustrative Example of the Use of the Proposed Taxonomy

This section demonstrates through an example, the use of the EDNA taxonomy outlined in the previous section.

Our example is drawn from [Miller et al, 1989]. In this scheme, the architecture of a network of $N$ units is represented by a connectivity constraint matrix, $C$ of dimension $N \times (N + 1)$, with the first $N$ columns specifying the constraints on the connections between the $N$ units, and the final column denoting the constraint on the threshold bias of each unit.

Each entry $C_{ij}$ of the connectivity constraint matrix indicates the nature of the constraint on the connection from unit $j$ to unit $i$ (or the constraint on the threshold bias of unit $i$ if $j = N + 1$). In their implementation, the authors allow each matrix entry to take on one of two possible values — 0 denoting the absence, and 1 indicating the presence — of a trainable connection between the corresponding units.

Given the connectivity constraint matrix, the genotype is constructed by concatenating the rows of the matrix, to yield a bit-string of length $N \times (N + 1)$. This is illustrated in Figure 2.

The genetic algorithm maintains a population of such bit-strings (each of length $N \times (N + 1)$). Each string codes for a network architecture. A fitness-proportional selection scheme chooses parents for reproduction — crossover swaps rows between parents, while mutation randomly flips bits in the connectivity constraint matrix with some low, pre-specified probability.

To determine the fitness of a genotype the architecture it encodes is first constructed by providing connections corresponding to matrix entries of 1, and not making any connections for the 0 entries (all feedback connections are ignored even though they may be specified in the genotype — thus this system evolves pure feed-forward networks). The connections in the network are then set to small random values, and trained for a fixed number of epochs using the back-propagation learning rule. The total sum squared error $E$ of the network, at the end of the learning phase, is then used as the fitness measure, with low values of $E$ corresponding to better learning of the task and hence a higher fitness label for the corresponding genotype.

The authors report results for the XOR problem (2-bit parity), the four-quadrant problem (a generalization of the XOR problem from binary inputs to real inputs) and
the pattern copying problem (the network simply copies the input onto the output — note that hidden units are not required for this task).

Now let us see how this approach fits into the EDNA taxonomy that was outlined in section 3.

This system uses a constraint matrix to specify the network connectivity. The genotype is a simple concatenation of the rows of this matrix. Decoding the genotype to construct the phenotype is a trivial process, requiring little more than reading of the matrix elements. Thus, as per our taxonomy, this system uses direct encoding.

Further, feedback connections are ignored by the decoding process hence only networks with feed-forward topology are evolved by this system.

Also, the only variable subjected to evolution is the connectivity (or the topology) of the network, the parameters (connection weights) are determined separately by the back-propagation procedure and are not subject to evolution.

Finally, the system was tested on three toy problems — XOR, four-quadrant problem and the pattern copying problem, which form its application domain.

The example discussed above illustrates the use of the proposed taxonomy to categorize current research on EDNA. It is our hope that it will also help identify open research problems and new research directions in this area.
4 Epilogue

This report has attempted to: motivate research on evolutionary approaches to the design of neural architectures (EDNA); classify current research in this area in terms of a preliminary taxonomy; and provide a guide to literature on EDNA in the form of a bibliography. Appendix I explains the organization of the bibliography and offers suggestions for using it effectively.

We have made every attempt to make the bibliography as complete as possible, however, we are almost certain that it contains gaps in its coverage of the literature on EDNA. We would welcome updates, additions and suggestions for keeping it up-to-date.

Though the bibliography contains over 300 entries, we have been able to label only a small fraction of them, primarily because we lack detailed knowledge of the contents of many of the publications included in it. We would appreciate input from the readers to help us completely label all the entries in the bibliography.

Acknowledgements

The entries in this bibliography were obtained either through our personal forays into the literature, or by researchers on our GANN\(^1\) mailing list. In particular, we would like to thank David Moriarty, Nikolaus Almassy, Frederic Gruau, Michel Olivier, Robert Smith, Filippo Menczer, Georg Thimm, Dan Adler, Peter Dudey, Dave Opitz, Larry Yaeger, Martin Mandischer, Jari Vaario, Russell Anderson, Riccardo Poli, Peter Hancock, Javier Marin, Frank Amos, Egbert Boers, Spyros Kazarlis and Johannes Schäfer for sending us relevant bibliographic references. We have also drawn on earlier bibliographies on this topic [Rudnick, 1990, Schaffer, 1994].

We are grateful to National Science Foundation for its support of our research through a grant (IRI-9409580) to Vasant Honavar.

---
\(^1\)GANN: Genetic Algorithms and Neural Networks Mailing List.
To subscribe, send e-mail to — gann-request@cs.iastate.edu with “subject” subscribe.
References


[Honavar & Uhr, 1993] Honavar, V. & Uhr, L. Generative Learning Structures and Processes for Generalized Connectionist Networks. *Information Sciences* (Special Issue on Artificial Intelligence and Neural Networks) 70 75-108.


Appendix I
Organization and Use of the EDNA Bibliography

The bibliography is organized (primarily) according to the taxonomy developed in section 3. The individual entries in the bibliography are labeled according to the various categories in the taxonomy (see below for details), making it easier to locate bibliographic entries that deal with specific aspects of EDNA. The following sections explain how to obtain an electronic version of the bibliography along with some suggestions for its effective use.

Obtaining an Electronic Version of the Bibliography

The compressed .bib file containing the bibliography is available through anonymous ftp at ftp.cs.iastate.edu as pub/gann/bib/gann.bib.Z. The file pub/gann/bib/README contains useful tips on how to search the bibliography using key words etc.

Alternatively, the bibliography may be accessed on the World-Wide Web (WWW) using a suitable WWW browser such as Mosaic via the URL:

http://www.cs.iastate.edu/~gannadm/homepage.html

Organization of the Bibliography

This bibliography is an attempt to put together as complete a collection as possible of publications on the evolutionary design of neural networks, organized roughly according to the taxonomy outlined in section 3. Each entry in the bibliography has been labeled depending on how that work addresses each of the four issues raised in section 3 — genotype representation, network topology, variables of evolution and application domain. In addition, we allow an additional label based on network type, mainly for pragmatic reasons — to facilitate search for bibliographic entries corresponding to specific popular network types like multi-layer perceptrons or Boltzmann machines or Hopfield networks etc. Thus, each label has 5 components, each component having the format:

<Key> = {<set of values>}

The five keys we use are — encoding, topology, evolves, applications and network, one for each of the issues listed above. For example, if the label of an entry in the bibliography includes the string "topology = {recurrent}"", then the entry deals with the evolution of recurrent networks. Similarly the string "evolves = {parameters}”, indicates that the corresponding entry describes an EDNA system with the network parameters (weights etc.) as the variables of evolution, and so on.

Currently, the bibliography is organized to support search on the basis of specified values for each of the 5 keys which can (at present) take on values from the domains specified below. Note that new values (and even keys) can be easily added to this system.

- encoding — direct, indirect, classifier systems, L-systems, graph grammars, grammar based, cellular encoding, LISP programs, developmental.
- **topology** — feed-forward, recurrent, multi-layered, locally connected, modular, randomly connected, regular, irregular, strictly layered, general (both feed-forward and recurrent).

- **evolves** — parameters, connectivity, topology, learning rule, size, construction rules, receptive fields, problem decomposition, feature detectors.

- **applications** — parity, encoder-decoder, contiguity, optimization, controller, simulated world, pattern recognition, speech recognition, image recognition, cognitive maps, temporal pattern recognition, game playing, function approximation, face recognition, digit recognition, language recognition, neurobiological modeling, data compression, travelling salesperson problem, regular language inference, binary mapping, time-series, pattern copying, four-quadrant.

- **network** — perceptron, multi-layer perceptron, self-organizing, kohonen, boltzmann, product-unit, radial basis function, counterpropagation, jordan, elman.

This labeling scheme provides a simple and effective way to search for bibliography entries that fall in any of the categories in the taxonomy outlined in section 3. The procedure for searching the bibliography and extracting relevant entries is outlined in the sections that follow.

### Locating Specific Information in the Bibliography

The labeling scheme outlined in the previous section makes it possible to locate entries in the bibliography that fall in one or more categories specified by the values of the keys. For example, all entries that deal with feed-forward neural networks will have as part of their label, the string “`topology = {feed forward}`”, while those that deal with recurrent ones will have the label — “`topology = {recurrent}`”. One of the simplest (although not the most convenient) ways to use the bibliography is to use a standard text editor and perform a pattern search for whatever is of interest. For example, if one is interested in work on evolving recurrent networks, he/she can simply search for entries that contain as part of their labels, a pattern of the form — “`topology = {recurrent}`” using virtually any text editor (e.g., `vi` or `emacs`).

The drawback of this approach is that unless the search string provided is exactly like the string in the label, the search query will fail. For example, search for — “`topology = { recurrent}`” will fail owing to the additional spaces in the search query string.

This problem becomes particularly critical for labels which have keys with multiple values. For example, any work dealing with evolving neural networks using the genetic programming paradigm (LISP programs), will have the `encoding` key labeled as — “`encoding = {indirect, LISP programs}`”. Here, one needs to specify the complete string in order to search effectively (partial string searches, like — ”`encoding = {LISP programs}`”, will fail to produce the desired response).
These problems can be overcome by using a more sophisticated search facility, in particular, something that works with \texttt{.bib} files. One such utility — \texttt{bibview} — provides all the functions we desire and more importantly, is available as a free, public domain utility on the internet. The next subsection provides pointers to obtaining and using \texttt{bibview} to perform efficient searches in the bibliography.

**Obtaining and Using \texttt{bibview}**

\texttt{Bibview} is available by anonymous ftp from: \texttt{sunsite.unc.edu}, as the file \texttt{pub/packages/T\!e\!X/biblio/bibtex/utils/bibview-2.0/bibview-2.1.tar.gz}. You will need to uncompress the file using \texttt{gunzip}, extract the files using \texttt{tar}, and install \texttt{bibview} on your system. The package works as is, on HP 9000/710 running HP/UX, DECstation 2100 and MicroVax II running ULTRIX and on SUN running SunOS 4.1. For other platforms, one might have to make specific changes to the makefile or contact Armin Liebl (liebla@informaatik.tu-muenchen.de) for help.

Once you have \texttt{bibview} running, you can load your \texttt{.bib} file by clicking on \texttt{File} and selecting \texttt{Open}. But before you do this, remember to turn off the \texttt{Automatic Check} option under \texttt{Options} (since quite a few of the entries in this bibliography have incomplete information which would otherwise cause \texttt{bibview} to complain of \textit{consistency error}). Once the \texttt{.bib} file is successfully loaded you can:

- \textit{List} all the bibliographic entries in the file and then \textit{Sort} them in whichever order you would like (a whole set of candidate options are available)
- \textit{Print} out the bibliography as a \LaTeX{} document to be incorporated in other \LaTeX{} documents
- \textit{Search} for specific references
- \textit{Add New} entries etc.

Since the other options are self-explanatory, we will briefly explain the steps involved in searching. For example, to list all entries on recurrent networks, the steps required are:

1. Start \texttt{bibview}
2. Click on options, and turn \textit{Automatic Check} off
3. Click on \texttt{File} and \texttt{Open} the required \texttt{.bib} file
4. Once the file is successfully loaded, click on \texttt{Search}
5. Now, to locate all entries on recurrent networks, go down to the \texttt{Userdefined Fields} and specify \texttt{topology} in the keyword section and \texttt{recurrent} in the corresponding value section
6. Click on \texttt{Start search}
7. `bibview` will return a list of entries from the bibliography that matches your input query, in this case, all entries *labeled* with recurrent networks.

Note that filling in more than one keyword section while searching will result in compound searches. For other capabilities of `bibview`, please refer to the README and CHANGES files accompanying the `bibview` utility. *Please bear in mind that all entries in this bibliography have not yet been completely labeled, hence there may be entries that may not show up in a search (although it is in the bibliography).*
The EDNA Bibliography

References


25


[MD] Michael McInerney and Atam P. Dhawan. Use of Genetic Algorithms with Back Propagation in Training of Feed-Forward Neural Networks. preprint from the Author; don’t “no” about publication.


World Congress on Neural Networks 93, pages III564 – III567. WCNN93, 1993.


