Artificial Neural Network in Developing Software Project Telemetry Metrics

Himanshu Sharma
CS Department, Aligarh College Of Engineering & Technology, Aligarh, Uttrar Pradesh, India
sharma_himanshu86@rediffmail.com

***Abstract*—Software development is very slow process, expensive process & error prone usually resulting in the products with a huge number of problems which cause serious and major mistakes in usability, reliability, and performance. To overcome this problem, software measurement provides a systematically and empirical-guided approach to control and Improve software development process. However, due to high cost linked with "metrics collection" and the difficulties in "metrics decision-making," measurement is not universally adopted by software organizations.**

However Software Project Telemetry is still one of the finest Solutions to this problem. The Conventional approach in software project telemetry is to use few automatic sensors to collect all metrics & further decision making is done based on them. But the main problem comes when it becomes very difficult to classify the collected metrics data & that's why if we need trained intelligent sensor based software project telemetry, it is ideal to use artificial neural network.

***Keywords*—software project telemetry, metrics collection, metrics decision making, artificial neural network**

I. Introduction

Software Development is a slow process, hours of work is needed to frame any software .The process is expensive because it requires support resources & lot of efforts. Also this process is error prone because of the logic & problem solution writing is complex, while writing long codes it is natural to be mistaken.

To overcome all these problems, and making software development quick and easy the computer itself needed to be smart. And the method should be opt by which one can guide the software development process, by giving direction, how mistakes not done and the code writing become quick also. For all this the whole software development process should be overlooked by a smart computer process. Now the point is how a computer process make smart-"if the computer can learn" ?

For a computer to become smart ,You can either write a totally fixed program for it – or you can also enable the computer to learn on its own. As we know the Living beings do not have any programmer writing a program for developing their own skills, which then only has to be executed definitely. They learn by themselves – without having the previous knowledge from any external impressions – and thus they can solve problems better than any computers today. What qualities are needed to achieve such a brilliant behavior for devices like computers? Can such features be adapted from biology?

II. NEURAL NETWORKS

There are a lot of problem categories that cannot be solved as an algorithm. Problems those depend on many subtle factors, for example the purchase & sale price of a real estate which our brain can calculate approximately. Without an algorithm one computer cannot do the same. Therefore the question to be asked is definitely: How do we learn to explore such kind of problems?

Exactly – we learn; a capability today's computers obviously do not have. Humans have a brain that can fairly learn. Computers have some processing units and memory. They together allow the computer to perform most complex numerical calculations in a short time, but they are not adaptive. If we compare computer and brain, we will note that, theoretically, the computer should be more powerful than our brain: It comprises 10^8 transistors with a switching time of 10^{-8} seconds. The brain contains 10^{12} neurons, but these only have a switching time of about 10^{-3} seconds. The largest part of the brain is working continuously, while the largest part of the computer is only passive data storage. Thus, the brain is parallel and therefore

	Brain	**Computer**
No. Of Processing Units	$\approx 10^{12}$	$\approx 10^{8}$
Type of Processing Unit	Neurons	Transistors
Type of Calculation	Massively Parallel	Usually Serial
Data Storage	associative	Address Based
Switching Time	$\approx 10^{-3}$sec	$\approx 10^{-8}$sec
Possible Switching Operation	$\approx 10^{14}$ per sec	$\approx 10^{20}$ per sec
Actual Switching Operation	$\approx 10^{12}$ per sec	$\approx 10^{10}$ per sec

Table 1.1: The (flawed) comparison between brain and computer at a glance

Performing close to its theoretical maximum, from which the computer is orders of magnitude away (Table 1.1). Additionally, a computer is static - the brain as a biological neural network can reorganize itself during its "lifespan" and therefore is able to learn, to compensate errors and so forth.

Within this text I want to outline how we can use the said characteristics of our brain for a computer system.

So the study of artificial neural networks is motivated by their similarity to successfully working biological systems, which - in comparison to the overall system - consist of very simple but numerous nerve cells that work massively in parallel and (which is probably one of the most significant aspects) have the capability to learn. There is no need to explicitly program a neural network. For instance, it can learn from training samples or by means of encouragement - with a carrot and a stick, so to speak (reinforcement learning)

III. BIOLOGICAL NEURAL NETWORKS

How do biological systems solve problems? How does a system of neurons work? How can we understand its functionality? What are different quantities of neurons able to do? Where in the nervous system does information processing occur? So here a short biological overview of the complexity of simple elements of neural information processing followed by some thoughts about their simplification in order to technically adapt them.

A neuron is nothing more than a switch with information input and output. The switch will be activated if there are enough stimuli of other neurons hitting the information input. Then, at the information output, a pulse is sent to, for example, other neurons.

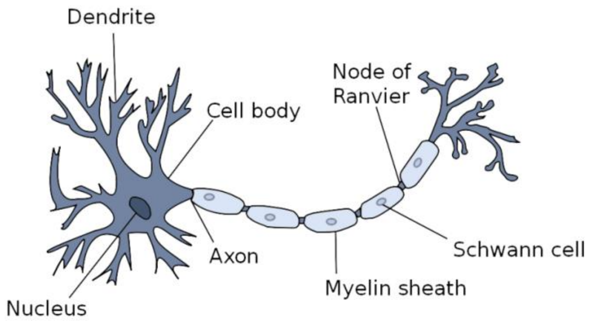


Figure 3.1: Illustration of a biological neuron with the components discussed in this text.

A. Dendrites collect all parts of information

Dendrites branch like trees from the cell nucleus of the neuron (which is called soma) and receive electrical signals from many different sources, which are then transferred into the nucleus of the cell. The amount of branching dendrites is also called dendrite tree.

B. In the soma the weighted information is accumulated

After the cell nucleus (soma) has received a plenty of activating (stimulating) and inhibiting (diminishing) signals by synapses or dendrites, the soma accumulates these signals. As soon as the accumulated signal exceeds a certain value (called threshold value), the cell nucleus of the neuron activates an electrical pulse which then is transmitted to the neurons connected to the current one.

B. The axon transfers outgoing pulses

The pulse is transferred to other neurons by means of the axon. The axon is a long, slender extension of the soma. In an extreme case, an axon can stretch up to one meter (e.g. within the spinal cord). The axon is electrically isolated in order to achieve a better conduction of the electrical signal (we will return to this point later on) and it leads to dendrites, which transfer the information to, for example, other neurons. So now we are back at the beginning of our description of the neuron elements. An axon can, however, transfer information to other kinds of cells in order to control them.

IV. ARTIFICIAL NEURAL NETWORKS

A technical neural network consists of simple processing units, the neurons, and directed, weighted connections between those neurons. Here, the strength of a connection
(or the connecting weight) between two neurons i and j is Definition(Neural network). A neural network is a sorted triple (N, V, w) with two sets N , V and a function w, where N is the set of neurons and V a set $\{(i, j)|i, j \in N\}$ whose elements are called connections between neuron i and neuron j .
The function $w : V \rightarrow R$ defines the weights, where $w((i, j))$, the weight of the connection between neuron i and neuron j , is shortened to $w_{i,j}$. Depending on the point of view it is either undefined or 0 for connections that do not exist in the Network.
So the weights can be implemented in a square weight matrix W or, optionally, in a weight vector W with the row number of the matrix indicating where the connection begins, and the column number of the matrix indicating, which neuron is the target. Indeed, in this case the numeric 0 marks a non-existing connection. This matrix representation is also called Hinton diagram.

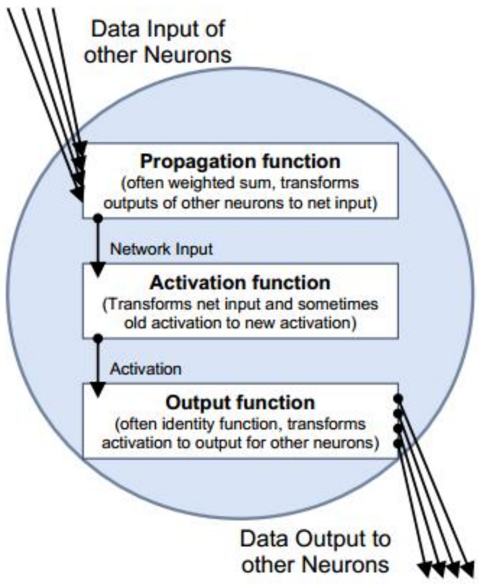

Data Input of other Neurons
Propagation function
(often weighted sum, transforms outputs of other neurons to net input)
Network Input
Activation function
(Transforms net input and sometimes old activation to new activation)
Activation
Output function
(often identity function, transforms activation to output for other neurons)
Data Output to other Neurons

Figure 4.1: Data processing of a neuron. The activation function of a neuron implies the threshold value.

A. Connections carries information that is processed

by neurons

Data are transferred between neurons via connections with the connecting weight being either excitatory or inhibitory. The definition of connections has already been included in the definition of the neural network.

B. The propagation function converts vector inputs to scalar network inputs

Looking at a neuron j , we will usually find a lot of neurons with a connection to j , i.e. which transfer their output to j . For a neuron j the *propagation function* receives the outputs oi1, . . . , o_{in} of other neurons i1, i2, . . . , in (which are connected to j), and transforms them in consideration of the connecting weights $w_{i,j}$ into the network input net_j that can be further processed by the activation function. Thus, the *network input* is the result of the *propagation function*.

Definition (Propagation function and network input). Let I = $\{i_1, i_2, \dots, i_n\}$ be the set of neurons, such that $\forall z \in \{1, \dots, n\} : \exists W i_Z, j$. Then the network input of j , called net_j , is calculated by the propagation function f_{prop} as follows:

$$net_j = f_{prop}(o_{i1}, \dots, o_{in}, w_{i1,j}, \dots, w_{in,j}) \quad ------- \quad (3.1)$$

Here the *weighted sum* is very popular: The multiplication of the output of each neuron i by $w_{i,j}$, and the summation of the results:

$$net_j = \textstyle\sum_{i\in I} (o_i * w_{i,j}) \quad - -------- \quad (3.2)$$

C. The activation is the "switching status" of a neuron

Based on the model of nature every neuron is, to a certain extent, at all times active, excited or whatever you will call it. The reactions of the neurons to the input values depend on this *activation state*. The activation state indicates the extent of a neuron's activation and is often shortly referred to as *activation*. Its formal definition is included in the following definition of the activation function. But generally, it can be defined as follows:

Definition (Activation state / activation in general). Let j be a neuron. The activation state a_j, in short activation, is explicitly assigned to j , indicates the extent of the neuron's activity and results from the activation function.

D. Neurons get activated if the network input exceeds their threshold value

Near the threshold value, the activation function of a neuron reacts particularly sensitive. From the biological point of view the threshold value represents the threshold at which a neuron starts firing. The threshold value is also mostly included in the definition of the activation function, but generally the definition is the following:

Definition (Threshold value in general). Let j be a neuron. The *threshold value* Θj is uniquely assigned to j and marks the position of the maximum gradient value of the activation function.

E. The activation function determines the activation of a neuron dependent on network input and threshold value

At a certain time – as we have already learned – the activation a_j of a neuron j depends on the previous activation state of the neuron and the external input.
Definition (Activation function and Activation). Let j be a neuron. The activation function is defined as

$$a_j(t) = f_{act}(net_j(t), a_j (t-1), \Theta j) \quad \dots\dots\dots\dots \quad (3.3)$$

It transforms the network input net_j , as well as the previous activation state $a_j(t-1)$ into a new activation state $a_j(t)$, with the threshold value Θ playing an important role, as already mentioned.

Unlike the other variables within the neural network (particularly unlike the ones defined so far) the activation function is often defined globally for all neurons or at least for a set of neurons and only the threshold values are different for each neuron. We should also keep in mind that the threshold values can be changed, for example by a learning procedure. So it can in particular become necessary to relate the threshold value to the time and to write, for instance Θ_j as $\Theta_j(t)$ (but for reasons of clarity, I omitted this here). The activation function is also called *transfer function*.

F. An output function may be used to process the activation once again

The *output function* of a neuron j calculates the values which are transferred to the other neurons connected to j. More formally:
Definition (Output function). Let j be a neuron. The output function

$$f_{out}(a_j) = O_j \quad -------- \quad (3.4)$$

calculates the output value O_j of the neuron j from its activation state a_j.Generally, the output function is defined globally, too. Often this function is the identity, i.e. the activation a_j is directly output **:**

$$f_{out}(a_j) = a_j \text{ , so } O_j= a_j \quad -------- \quad (3.5)$$

Unless explicitly specified differently, we will use the identity as output function within this text.

G. Learning strategies adjust a network to fit our

needs

Since we will address this subject later in detail and at first want to get to know the principles of neural network structures, I will only provide a brief and general definition here:
Definition(General learning rule). The *learning strategy* is an algorithm that can be used to change and thereby train the neural network, so that the network produces a desired output for a given input.

V. SOFTWARE PROJECT TELEMETRY

Software Project Telemetry is a project management technique that uses software sensors to collect metrics automatically and unobtrusively. It then employs a domain-specific language to represent telemetry trends in software product and process metrics.

Project management and process improvement decisions are made by detecting changes in telemetry trends and comparing trends between different periods of the same project. Software project telemetry avoids many problems inherent in traditional metrics models, such as the need to accumulate a historical project database and ensure that the historical data remains comparable to current and future projects.

It addresses the *"metrics collection cost problem"* through highly automated measurement machinery: software sensors are written to collect metrics automatically and unobtrusively. It addresses the *"metrics decision-making problem"* through a domain-specific language designed for the representation of telemetry trends for different aspects of software development process.

VI. METRICS COLLECTION

The metrics collection is the core thing by which we can overlook and guide the software development process. Through this metric collection all the patterns of the mistakes in the software development that are usually done by code writers are identified and suggested to make correct. Now there are two methods by which these metrics are collected.

A. Sensor-based Data Collection

In software project telemetry, metrics are collected *automatically* by sensors that *unobtrusively* monitor some form of state in the project development environment. Sensors are pieces of software collecting both process and product metrics.

Software process metrics are the metrics that assist in monitoring and controlling the way software is produced. Sensors collecting process metrics are typically implemented in the form of plug-ins, which are attached to software development tools in order to continuously monitor and record their activities in the background. Some examples are listed below:

- A plug-in for an IDE (integrated development environment) such as Visual Studio, and Eclipse. It can record individual developer activities automatically and transparently, such as code editing effort, compilation attempts, and results, etc
- A plug-in for a version control system, such as Clear Case, CVS, and SVN. It can monitor code check-in and check-out activities, and compute *diff* information between different revisions.
- A plug-in for a bug tracking or issue management system, such as Bugzilla, and Jira. Whenever an issue is reported or its status is updated, the sensor can detect such activities and record the relevant information.
- A plug-in for an automated build system, such as Cruise Control. It can capture information related to build attempts and build results.

Software product metrics are the metrics that describe the properties of the software itself. Sensors collecting product metrics are typically implemented as analyzers for software artifacts. These analyzers usually need to be scheduled to run periodically in order to acquire the continual flow of metrics required by telemetry streams. To automate these tasks, one can use a *Schedule tasker* job, or run them as tasks in automated build system. Some examples are listed below:

- An analyzer that parses program source code to compute size or complexity information.
- An analyzer that parses the output of existing tools, such as Clover, and JBlanket , and converts them to a data format that can be used by software project telemetry**.**

There are many other possibilities. One can even imagine an exotic sensor that retrieves project cost and payroll information from a company's accounting database, if extraction of such information is permitted by the company policy. The point is: no matter what the sensor does and regardless of its implementation details, a sensor-based approach collects metrics *automatically* and *unobtrusively* in order to keep data collection cost low, so that developers are not distracted from their primary tasks - developing software products instead of capturing process and product metrics. This sensor-based approach eliminates the chronic overhead in metrics collection. While setting up sensors might require some effort, once they are installed and configured, sensor data collection is automatic. This contrasts with traditional data collection techniques, such as the paper-and-pencil based approach used in PSP/TSP, or the tool-supported approach used in LEAP, PSP Studio, and Software Process Dashboard. These approaches require constant human intervention or developer effort to collect metrics. Even in the case of the tool-supported approach, the developer still cannot escape the chronic overhead of constantly switching back and forth between doing work and telling the tool what work is being done**.**
The fact that chronic overhead is eliminated from sensor-based metrics collection not only reduces the technology adoption barrier, but also makes it feasible for software organizations to

apply measurement to a wide range of development activities and products in order to get a comprehensive quantitative view of development processes**.**

Admittedly, the sensor-based approach does come with some restrictions:

- A sensor must be developed for each type of tool we wish to monitor. This is a one-time cost. Once the sensor is developed, it can be used by different software development organizations for different projects. The Collaborative Software Development Lab has already developed a repository of over 25 sensors for commonly-used tools.
- Some metrics may not be amenable to automated data collection. An example is software development effort. While it is feasible to instrument an IDE to automatically get information such as how many hours a developer has spent on writing code, it is almost impossible to construct a sensor that knows how much total effort a developer has contributed to a project. For instance, two developers might be discussing the design of a system in the hallway. It is almost impossible to collect this type of effort in an automated way. It is still an open research question whether *all* important metrics can be captured by sensors or not. However, this research takes a more pragmatic view: it is only concerned with whether sensors can collect *sufficient* metrics so that software project telemetry has decision-making value for project management and process improvement.

B. *Telemetry Language and Telemetry Constructs*

Many interesting issues in software project management involve understanding the relationship between different measures. For example, we might be interested in seeing whether an increased investment in code review pays off with less unit test failures, and/or increased coverage, and/or less defects reported against the reviewed modules. Such questions require comparing a set of metrics values over time. The telemetry language provides a mechanism that facilitates interactive exploration of relationships between metrics. The language has the following syntax:

```
Streams  <Stream  Name>  (<ParameterList>)  =
{<DocumentationString>,<Expression> };
Y-axis

<YAxisName>(<Parameter>)={label,'integ
er|double|auto',lowerBound,upperBound
};
Chart  <Chart  Name>  (<ParameterList>)  =
{<ChartTitile>,<StreamReferences> };
Report            <Report            Name>
(<ParameterLilst>)                       =
{<ReportTitle>,<ChartReferences> };
```

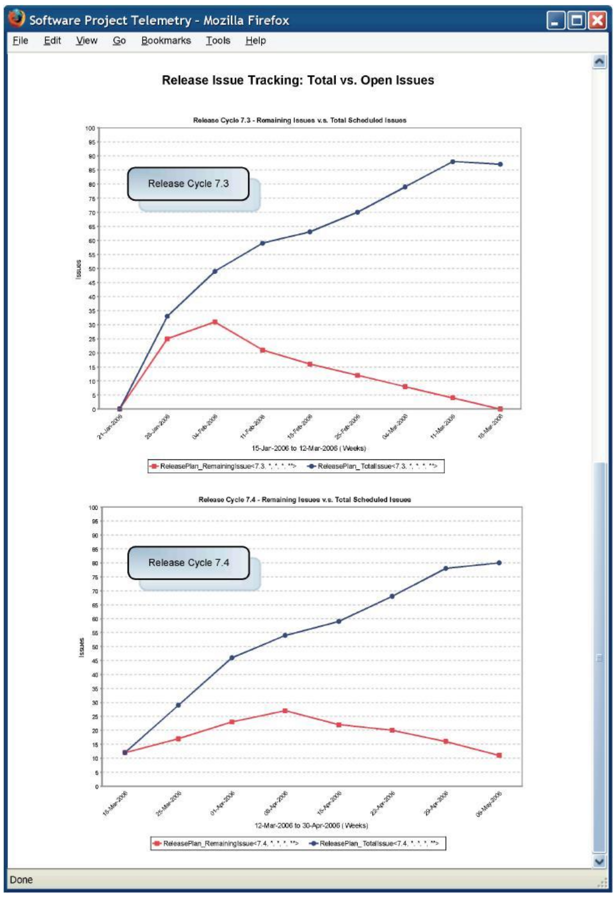
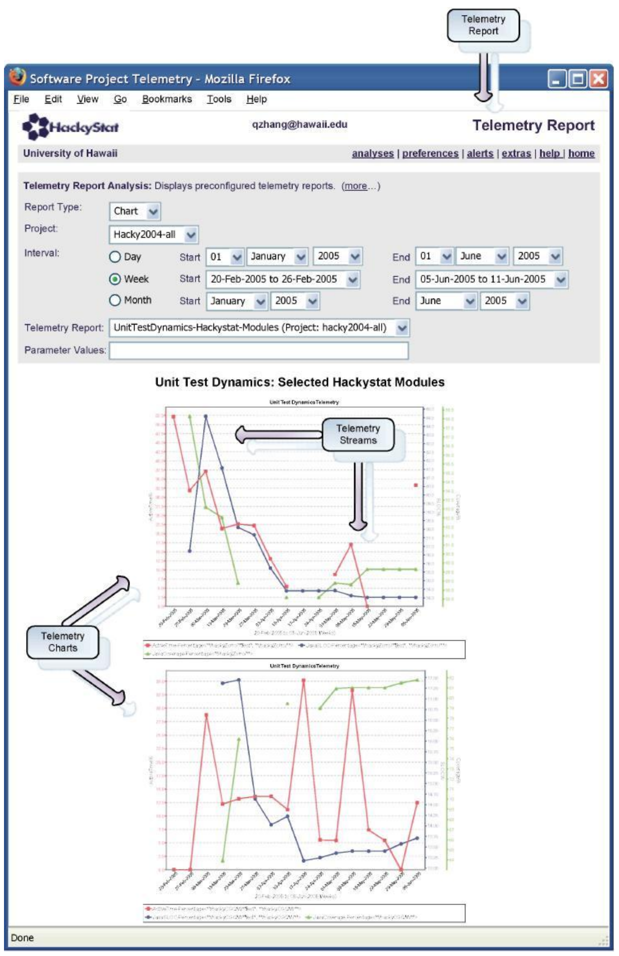


Figure 6.1 Release Issue Tracking: Total vs. Open Issues

In essence, a telemetry report is a named set of telemetry charts that can be generated for a specified project over a specified time interval. The goal of a telemetry report is to discover how the trajectory of different process and product metrics might influence each other over time, and whether these influences change depending upon context. A telemetry chart is a named set of telemetry streams. The goal of a telemetry chart is to display the trajectory of one or more process or product metrics over time.


Telemetry Report
Software Project Telemetry - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
HackyStat
qzhang@hawaii.edu
Telemetry Report
University of Hawaii
analyses | preferences | alerts | extras | help | home
Unit Test Dynamics: Selected Hackystat Modules
Telemetry Streams
Telemetry Charts
Done

Telemetry Report Analysis: Displays preconfigured telemetry reports. (more...)

Report Type: Chart

Project: Hacky2004-all

Interval:
- Day Start 01 January 2005 End 01 June 2005
- [✓ Week] Start 20-Feb-2005 to 26-Feb-2005 End 05-Jun-2005 to 11-Jun-2005
- Month Start January 2005 End June 2005

Telemetry Report: UnitTestDynamics-Hackystat-Modules (Project: hacky2004-all)

Parameter Values:

Figure 6.2 Telemetry Report Analysis

The *y-axis* construct is used to specify the vertical axis of a telemetry chart. Note, however, that a telemetry chart definition does not include the information about its horizontal axis, because such information can be automatically inferred from the time interval over which the telemetry analysis is performed. A telemetry stream is a sequence of a single type of software process or product metrics.

VII. CONCLUSIONS

So how *artificial neural network contributes in software project telemetry* As we discussed previously the approaches in this field of software project telemetry are Sensory based data collection & Language constructs. Both of these approaches are limited because of the lacking of learning factor that's why use of artificial neural network can sufficiently improve the required results.

The Purpose of this study is to improve software development process; The method is Software Project Telemetry which is only based on metrics approach & has two main approaches Sensory based data collection or Language constructs but both approaches lacks in learning metrics for collection & decision making. That's why there is a scope of using Artificial Neural Networks, as Artificial Neural Network can help in developing these metrics & metrics decision making well.(fig 7.1)

The Software Project Telemetry is an emerging field of computer science and there is a lot of possibility of renowned researches in this, as Artificial Neural Network is one of the best approach to apply intelligence in any software that's why these two fields are the focus of attraction for the future researchers.

ACKNOWLEDGMENT

I like to express our special thanks and gratitude to Dr. Anand Sharma, HOD, CS Department , Aligarh College of Engineering & Technology, Aligarh,(Uttar Pradesh) to inspire me to write Some Work. Also I like to Thank Dr.Vinod Sharma, Director, Aligarh College of Engineering & Technology, Aligarh,(Uttar Pradesh) to provide us Platform where we can grow in this fast paced professional environment with limited abilities.

REFERENCES

[1] David Kriesel, "A brief introduction to: Neural Networks, (2005)

[2] Philip Johnson , Qin Zhang-Improving software development process & project management with software project telemetry (2005)

[3] Dr. Qadri Hamarsheh -Neural Networks & Fuzzy Logic

[4] Kenji Suzuki, Artificial Neural Networks-Methodological Advances & Biomedical Applications (2011)

[5] Hristev, R. M. "The ANN Book" ,Edition-1,(1998)