Czech Technical University in Prague
Faculty of Information Technology

ASSIGNMENT OF MASTER'S THESIS

Title:	Decision Making and Construction of Trust in Ad-Hoc Networks Using Neural Networks
Student:	Bc. Tatyana Aubekerova
Supervisor:	Ing. Alexandru Moucha, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Systems and Networks
Department:	Department of Computer Systems
Validity:	Until the end of summer semester 2016/17

Instructions

Generally, neural networks are able to estimate and find data dependencies not known before. They are able to make decisions based on some learning performed (samples of good or bad data) and thus they theoretically can be useful in construction of trust in distributed systems.
Study out which properties and types of neural networks could be used to analyze and construct trust in ad-hoc networks. Create a neural network simulator to analyze and discover which parameters make sense for such trust methods. Discuss the results of the simulations and their applicability for trust management in real ad-hoc networks and formulate the conclusions.

References

Will be provided by the supervisor.

L.S.

prof. Ing. Róbert Lórencz, CSc.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague October 15, 2015

Czech Technical University in Prague

Faculty of Information Technology

Department of computer systems

Master's thesis

Decision Making and Construction of Trust in Ad-Hoc Networks Using Neural Networks

Bc. Tatyana Aubekerova

Supervisor: Ing. Alexandru Mihnea Moucha, Ph.D.

14th July 2016

Acknowledgements

I would like to thank my family for support during writing this thesis. Also I am grateful to my supervisor Alex Moucha for providing valuable advices and be supportive during work on this thesis.

Declaration

In Prague on 14th July 2016 . . . .

Citation of this thesis

Aubekerova, Tatyana. *Decision Making and Construction of Trust in Ad-Hoc Networks Using Neural Networks.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Tato práce je věnována oblasti Ad-hoc sítí, konkrétně konceptu důvěry a je záměrená na možnosti využití neuronových sítí do tohoto konceptu. Nejprve jsou uvedeny definice souvisejících pojmů, pak problém je definovan a následuje navrhované řešení. Hlavní náplní práce jsou provedené experimenty, jejich následné vyhodnocení a diskusi.

Klíčová slova ad-hoc sítě, důvěra, neuronovy sítě, Omnet++, FaNN, PRD

Abstract

This work is dedicated to the field of Ad-hoc networks, more specifically to concept of trust and intent to find application of neural network to this concept. First there are given definitions to related terms, then the problem is stated and the suggested solution follows. The main pivot of the thesis are experiments held and their evaluation and discussion.

Keywords ad-hoc network, trust, neural network, Omnet++, FaNN, PRD

Contents

List of Figures

List of Tables

Introduction

Motivation and objectives

Ad-hoc networks are created by nodes, which are routers and end-stations at the same time, thus such networks can function only if nodes cooperate. In case some message was not delivered it is generally hard to tell from the view of sender, which node was responsible. Even if the node have information from other nodes, it is difficult to construct a method which decides if the given node is untrusted. The most important advantage of neural networks(NNs) is their ability to solve problems, for which conventional algorithmic solution does not exist or is too complex.[1] In general, being a conceptional representation of human brain, NNs are suitable for the problem solving in which people are good, but computers are not. They can learn and reorganize itself from experience, adapt to the environment. As trust concept arose from the notion of social trust the assumption is that NNs can be capable of trust evaluation in Ad-hoc networks.

This thesis is structured as follows: first the basic notions related to the research are described, then the formulation of problem and its analysis is presented. The simulator for implementation of chosen scenario and collecting data for NN was created. Description of the experiments with training NN follows and performance and results are analyzed.

State of the Art

In [2] authors suggested to use NN in peer-to-peer (P2P) networks for evaluation of recommendations received from neighbors and determine the trust level of the target node. Then the communication take place and based on the obtained result NN weights are adjusted. After a while NN is trained and can be used for making decisions. Some other studies are using NN for prediction the node behavior and thus intrusion detection [3] or for making routing decisions.[4]

Chapter 1

Basic concepts and definitions

First chapter aims to describe related theory behind the thesis.

1.1 Ad-hoc networks

1.1.1 General notions

Field of wireless rapid-deployed networks have received a lot of attention recently owning to exponential growth and evolution of wireless communication. *Ad-hoc network* is a set of small modules, connecting on-the-fly. Every node plays role of the router and end station at the same time.

Due to high mobility, continuous changes and lack of fixed infrastructure and any central management these networks are prone to have security vulnerabilities. Modules located inside the radio range can communicate directly, otherwise they need to use intermediate nodes to deliver their messages. Therefore network performance highly depends on the cooperation of nodes, thus mechanisms to enforce the collaboration are required. Node not necessary has to be malicious, the case it is selfish is equal to maliciousness.

Challenges, such as deficiency of computational power and energy resources, preserves from using classical key-establishment schemes and central certification authorities. In order to reduce potential threats the concept of trust arose.

1.1.2 Structure and properties of node

Network is created by *nodes*. These are small modules, consisting of antenna and radio transceiver, processing unit, memory and battery. As computations made by one node for the needs of this research are negligible, detailed description of processing unit and memory is out of scope of this paper. Energy consumption is also topic of another research. Thus follows the description of antenna and radio.

For wireless communication antenna is probably the most critical constituent element. *Antenna* is usually a metal device which is a means for sending and receiving data. In other words the antenna is the transitional structure between free-space and a guiding device.The guiding device or transmission line may take the form of a coaxial line or a hollow pipe (waveguide), and it is used to transport electromagnetic energy from the transmitting source to the antenna, or from the antenna to the receiver. [5] Radio transceivers operate in half-duplex, as transmit and receive data at the same time is not possible.

Fundamental parameter of antenna is *radiation pattern*. It is defined as "a mathematical function or a graphical representation of the radiation properties of the antenna as a function of space coordinates".[5] Radiation properties can include power density, field strength, directivity, polarization or phase. The most interesting is the space distribution of related energy, so the node range can be determined. A graph of spatial distribution of power density is called power pattern and usually is plotted in decibels.

Radiation pattern can be isotropic, directional, and omnidirectional. *Isotropic* means the equal radiation in all directions, it is not reachable in practice, but serve as a reference for expressing directive properties for real antennas. *Directional* antenna can send/receive more effectively in some direction. Antenna, whose radiation pattern is non-directional in one plane and directional in orthogonal is called *omnidirectional* and is represented on Figure 1.1).[6] These antennas are used in our research.

Range of antenna is the distance within the node can communicate. It grows with square root of power, which means that to double the communication distance requires four times the power. Communication range defines which nodes will be neighbors, thus have a direct link between them. Set of links determines certain network topology.

1.1.3 Network topology

Network can be viewed as a graph $G = V \times E$, where V is a set of all nodes in the network and E is a set of all links between nodes. Each $G^* \subseteq G$, where $V^* = V$ will give a different *network topology* with the same nodes.

Path between two nodes is a sequence of links which need to be passed to get from one node to another. Start node is called source and the final node is destination. Paths can be found by routing algorithms. So network topology is defined by routing.

1.1.4 Routing

Generally routing protocols for Ad-hoc networks can calculate the routes periodically (proactive) or on-demand (reactive). Due to such disadvantages as high memory demand and slow reaction on restructuring and failures [7],

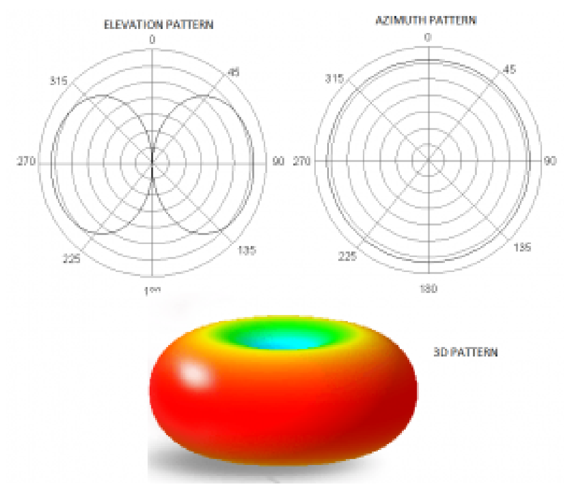

Figure 1.1: Omnidirectional antenna

proactive protocols (OLSR, DSDV) are not commonly used for Ad-hoc networks. Instead, reactive protocols are preferable here. The most prevalent are AODV, DSR, and OR. Their disadvantages are greater latency and flooding the network with route request messages.

In Ad hoc On-demand Distance Vector (AODV) protocol all mobile nodes work collectively to discover a route path from source to destination. An actual data transfer takes place only after the route is established.

There are three types of control messages: RREP (Route Reply), RREQ (Route Request), and RERR (Route Error). To find a path the source broadcasts RREQ packet to the network. On receipt of RREQ, a node sends a RREP packet, if it is the destination or if it has a fresh enough route to the destination, otherwise it just forwards RREQ packet to its neighbors.

On receipt of RREQ message, every node increases hop count by one and on receipt of RREP, intermediate nodes update their route entry with the new data. Whenever a new RREQ, RREP or RERR messages are sent, nodes increase their own sequence number. Higher the sequence number more considered that information. Path with smallest hop-count is chosen [8].

As AODV protocol is the most widespread, and for this research basically any algorithm is suitable which will create topology, AODV was chosen as routing protocol in our simulation.

DSR (Dynamic Source Routing) is similar to AODV in terms, that it is on-demand routing, but it does not rely on routing tables at the intermediate nodes, using the whole path calculated by source.

In OR (Opportunistic Routing), a set of nodes is selected as potential forwarders. The nodes in the selected set will forward the packet according to some criteria after they receive the packet. This group of nodes in OR is usually called a Candidate Set (CS). A priority is assigned to each candidate in the CS.

Candidate priority shows the level of ability of a candidate to act as the next forwarder. The highest priority is given to the candidate that can reach the destination at the lowest cost. This cost could be understood in different terms: for example, distance to the destination in terms of the number of hops, power consumption, the Expected Number of Transmissions (ExNT), and the like.

The candidates that have received the transmitted packet coordinate among each other to decide which of them must forward the packet and which must discard it. This process is usually called candidate coordination. All OR protocols differ in OR metrics, candidate selection algorithm and candidate coordination method[9].

Need to mention that neither AODV nor DSR do not choose a route based on the stability of links but focus on the shortest path leading to sending data through possibly untrusted nodes.

1.2 Concept of trust

Trust in the scope of Ad-hoc networks means a measure of confidence in the fact that a node will behave according to expectations. Trust concept in this case provides a light and feasible way to enhance security of network. Trust establishment schemes are used for different purposes, such as authentication, access control, intrusion detection and secure routing.

There are several different trust methods and models proposed by researches up to now. However, the trust in wireless ad-hoc and sensor networks is still an open problem. There is no standardization or classification, and therefore no specialized books have ever been published.

Prior to experiments and evaluations there is need to determine the concept of trust and agree on its meaning in the context of our research.

Notion of trust originally comes from social disciplines and is defined as a measure of subjective belief regarding the behavior of a some entity [10].

During last years a theory of trust in relation to wireless Ad-hoc networks was gradually created. It discusses the concept and properties of trust and become generally accepted among researchers. Fundamental properties of trust according to this theory are dynamicity, subjectivity, incomplete transitivity, asymmetry and context-dependency [11].

As network infrastructure is not static, nodes join and leave time to time, some nodes can fail, thus network status changes frequently, and trust values should consider these changes. Nodes can have incomplete and partially local information about situation in the network. Hence, trust value should be continuous to represent this dynamics. It definitely can not be binary and if discrete, it at least should be with enough number of grades.

Subjectivity comes from social disciplines and indicates the biased nature of trust evaluations, based on different experiences.

Trust in psychology makes accent on the cognitive process implying that humans acquire trust values from their experiences. In the simulation each node creates its own opinion on the trust of other nodes based on communication experience with that node.

Non-transitivity or incomplete transitivity means that if node A trusts node B and node B trusts node C, that does not imply that node A trusts node C. Although, when using recommendations from other nodes, partial transitivity takes place.

Trust is not a mutual concept. The fact, that a node believes in another's node trustworthiness does not induce its trustworthiness in return.

The last but not least property of trust is context-dependency. Li and Singhal [12] define trust as the belief that an entity is capable of performing reliably, dependably, and securely in a particular case; hence, different levels of trust exist in different contexts. For example, one can trust his doctor advices on the health issues, but do not trust if the doctor makes advice on money managing.

Various metrics are used to compute trust. Metrics should justly reflect the situation in the network with respect to trust calculations. To determine which metrics to use is not a simple task.

Taking into account an economic basis of trust, the selfishness of nodes should be considered, as trust in economics is based on the assumption that humans are rational and strict utility maximizers of their own interest or incentives [13]. Selfishness in network can be measured by packet dropping rate or not responding to route requests.

Trust also imply wiliness to take a risk of become vulnerable or lose data. Hence, the most critical metrics for given network should be investigated. Level of trust is often linked with level of reliability. Reliability in terms of networking means a guarantee of data delivery. So, the metric is packet delivery ratio. Josang et al. [14] and Solhaug et al. [15] conclude that trust is generally neither proportional nor inversely proportional to risk. Risk is closely connected to stake. Even when the trust is strong, if the stake is high the risk will be also great.

On account of trust to be context-dependent, trust metrics should return adequate references for diverse situations, depending on how sensitive the information is.

As the most critical feature of the network is its ability to deliver data, for this research as a trust metrics packet delivery ratio (PDR) was chosen. More in detail this is described in chapter 3.

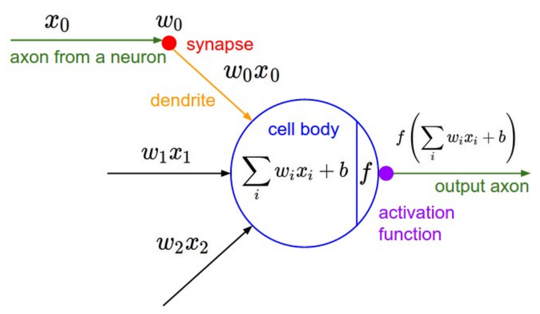

Figure 1.2: Mathematical model of neuron

1.3 Basics of artificial neural networks

1.3.1 Model of neuron

Biological neuron is the prototype and inspiration for mathematical model of artificial neural networks(Figure 1.2).). It has several dendrites(inputs) and one axon(output). Signals from dendrites traverse through neuron body, what in mathematical model is implemented with activation function.

Axon connects to other dendrites via synapses, which have different strength (weights) and thus can be excitatory or inhibitory. First the weighted sum of inputs is calculated and then passed as argument to activation function. Usually bias(b) is used to shift the activation function to the left or right. Generally, if the value of activation function is greater then some threshold, signal is fired to the output. Bias is passed as one of the inputs and changes threshold, so for clarity it is better to set threshold to 0 when using bias.

Most common activation function is Sigmoid (1.1), which is used in multi-layer perceptron networks.

$$S(t) = \frac{1}{1 + e^{-t}} \tag{1.1}$$

1.3.2 Architecture of neural networks

As stated in [16] NN is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes. In essence NN is a structure of connected nodes with some number of inputs and outputs, embedded activation function and each connection has a weight assigned to it. Inspired by biological nervous systems they are mathematical models. NN should be considered when input data is high-dimensional or possibly noisy and the transformation function is unknown.

A lot of different types of NN were created, for example:

- Multi-Layer Perceptron
- Radial Basis Function (RBF)
- Kohonen Features maps
- Other architectures (Hopfield networks, Boltzmann machine, etc.)

Each NN is characterized by:

- model of neurons (details on their inherit properties and functions)
- topology of the network
- learning method

NN consists of input layer, one or more hidden layers and the output layer. Network design include setting the number of neurons in each layer, number of hidden layers, create certain topology and choosing the model of neurons.

Several architectures are distinguished according to signal flow:

- Single-Layer Feed-forward
- Multi-Layer Feed-forward
- Recurrent

In feed-forward networks information always flows in one direction. Recurrent topology allow cycles. There is at least one feed-back connection.

Single-layer networks consist of one input layer and one output layer of processing units. Multi-layer architecture in addition has one or more hidden layers of processing units.

Recurrent networks may or may not have hidden layers. There are also further variations of topology, like short-cut connections, partial connectivity or time-delayed connections.

1.3.3 Learning methods

NN needs to be configured for particular problem. This is usually achieved by process called learning. Learning means adjusting connection weights to get desired output. There is a set of known input/outputs and after feeding the system with input and getting the result alter the connection weights to obtain more fitting output. So node connection strengths known as weights are used to store the learned knowledge.

There are different types of learning: supervised and unsupervised learning. First assumes that data for training contains values of inputs and their corresponding outputs.

At the beginning all weights are initialized randomly. In each training cycle NN is feed up with inputs and after receiving the response from network, it is compared to the correct output and values of weights are adjusted.

There exist different methods of adjusting weights, one of most common is Backpropagation algorithm. It calculates gradient of cost function with respect to weights. Gradient shows how fast the cost changes when the weights change. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the cost function.[17]

Important parameters of this learning method are momentum and learning rate. Momentum helps to avoid local minimum. It simply adds a fraction of the previous weight update to the current one.[18] The learning rate, LR, applies a greater or lesser portion of the respective adjustment to the old weight.[19]

If the network is too large, it has difficulties with learning and after being learned it will tend to overfitting, that results in poor generalization. It means that the network is over learned - it predicts perfectly the results of training instances but cannot deal with new data.

If the network becomes too small, it will not be able to represent the rules needed to learn the problem and it will never gain a sufficiently low error rate.

Another reason for overlearning is due to excessive iterations, too big number of learning epochs. An epoch is a time during which all training instances are used once to update the weights.

The number of hidden layers is also important. Generally speaking, if the problem is simple it is often enough to have one or two hidden layers, but as the problems get more complex, so does the need for more layers.[20]

1.3.4 Limitations of NN

NN can be trained to work with specific problem. And they cannot be retrained if the problem changes, completely new training should be performed. Change of number of inputs or outputs means again a new problem and thus construction of another neural network.

Moreover, number of training instances for each class in classification problem should be exact the same. Training instance should cover the problem set uniformly.

Results of training depends on random initialization, which is some cases may not lead to sufficient training results.

Chapter 2

Statement of objectives

Next chapter specifies the object of the research and states the problem to be analyzed and solved.

2.1 Object of research

The routing in traditional infrastructure-based network is implemented with the help of dedicated devices. In case of infrastructure-less network, the routing is achieved via message forwarding between nodes.

Taking in mind that there are different types of mobile ad-hoc networks, it is necessary specify to which of them this work is related. First, drop all schemes dealing with hierarchical/cluster networks. All nodes have the exact same responsibility and functionality, that means the network is homogeneous.

This topology is very productive for dynamic environment and becomes less efficient as the total number of nodes significantly increases. Thus, our research is related to non-clustered (plain) networks of middle size (~ 20 nodes). Each node has omnidirectional antenna with specified communication range. Each node has some probability of not forwarding data. The *packet delivery ratio (PDR)* of a node is the ratio of packets successfully received to the total sent. PDR is used as metric of trust. Similarly PDR of the path is defined.

2.2 Problem definition

The core of the problem is to detect untrusted nodes. Node is considered to be *untrusted* when its PDR reaches some specified threshold. There could be several reasons for node to make errors and not deliver data. First is traffic congestion. If a node is on the path of great amount of routes and the traffic is intensive, node can be overloaded and will not be able to forward data. This problem can be solved by better topology: creating alternative routes to take

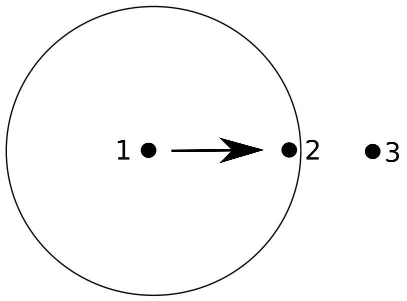

Figure 2.1: Node ranges and possibilities for eavesdropping

off the load from the most used nodes. Several topology control algorithm exist for this purpose.

Another reason can be node itself. Not depending on traffic some node can behave improperly. Cause for this may hide in fault hardware or software or the node can be malicious. This situation is hard to solve, it should be detected and this particular node should be excluded from network communication.

Each node can try to measure and share information about other nodes error rates, but the question is to which extent this information can be trusted. Therefore the problem is the detection of error nodes from outside. This can be done in several ways:

- Eavesdropping
- Measurement by communication
- Detection of path error rate

Die to the fact that radio is a shared medium, each node can see packets in its communication range, even if these packets are not targeted to it. Thus when node sends a packet to its neighbor it can try to eavesdrop if the neighbor forwards the data further. Figure 2.1 shows node 1 and node 3 which cannot communicate directly, therefore they forward their communication through node 2. As node 2 is in communication range of node 1, node 1 after sending the packet turns its network card in promiscuous mode (receiving all traffic even if it is not addressed to this node) and eavesdrops if node 2 will forward the data (Figure 2.2).

This seems to be working, but let us show another example on Figure 2.3.

Again nodes 1 and 3 are not in direct communication range of each other and use node 2 for relaying their data. But in this case if node 1 wants to eavesdrop node 2 it will not obtain relevant information as node 2 adjusted its power to be sufficient to send data to 3, but node 1 became out of the range. From node 1 point of view node 2 never forward any packet.

This is the first disadvantage of eavesdropping, next one is the fact, that being in promiscuous mode node is not able to receive any data. This in final

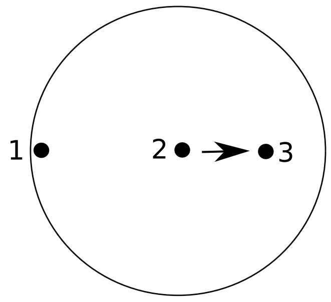

Figure 2.2: Node ranges and possibilities for eavesdropping

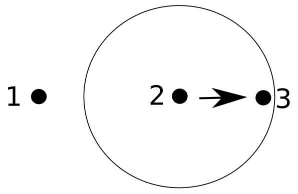

Figure 2.3: Bad possibilities for eavesdropping

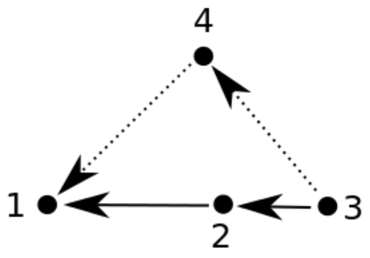

Figure 2.4: Measurement of communication

result collides with methods of access to shared medium (for example carrier sense multiple access with collision avoidance (CSMA/CA))

Next way to detect the error node is measurement of communication(see Figure 2.4). Here node 1 and node 3 want to perform measurement on node 2. First they need to agree about measurement process and have to use some alternative route for that (for example through node 4), which does not have to exist. Even if this is accomplished, another difficulty comes. There is a need to ensure requirements for measuring, in particular, there cannot be any traffic in this part of the network during measurement process, and this is hard, if not to say almost impossible to guarantee.

One more way to detect error rates of nodes is measure of error rates of paths. Later it will be explained on small configuration how this method works.

Chapter 3

Suggested solution

Suppose that some node already has information about the network topology. This means it has routes to all other nodes in the network. Then it can observe PDR of each path. Assuming we have information about PDR of all paths from all nodes, that can be bind with network topology (information about all intermediate nodes to all destinations). The assumption is, that there are dependencies in this data, that are hard to find by conventional algorithms. Searching dependencies in this data can assist in determining PDR of the particular node.

On this stage NN comes into play. Now the task is to determine format of data for NN. The goal of NN is evaluate information about particular node and make decision about trust of this node. Trust, as were mentioned before, is PDR of that node.

Each node participates in different set of paths, which means each node represent separate problem, for which individual NN should be trained. The target node is used by one or more nodes in the network. Each of those nodes calculates the average value of PDR of the paths where the target node is present. This average PDR becomes one of the inputs of NN. From this follows that number of inputs of NN is the number of nodes which are using the investigating node. There are two possible types of output of NN.

First it can determine if the node is trusted or untrusted. Back propagation neural networks can naturally solve two types of problems - classification and regression. The primary goal of this work was detection of untrusted nodes however after performing some experiments it became clear that neural network is able to perform also a regression analysis. This means it is possible to estimate the PDR value of every node in the topology.

While classification typically uses a form of logistic regression in the network final layer to convert continuous data into 0 or 1 – e.g. given someone's height, weight and age you might bucket them as a heart-disease candidate or not – true regression maps one set of continuous inputs to another set of continuous outputs.[21]

3.1 Creation of Data Instances

In this section the creation of data instances which represent the current state of the network will be described. The example situation is on the figure 3.1. It represents very simple topology that could be created for example by AODV routing algorithm and is composed of six nodes (1 to 6). The focused nodes are the middle ones - 5 and 6. These intermediate nodes are responsible for successful delivery of data messages. Both have assigned the PDR (which is unknown in the real world) - the node 5 has PDR 0.4 and node 6 has 0.9. Lets assume that the threshold PDR between trusted and untrusted node is a value of 0.5.

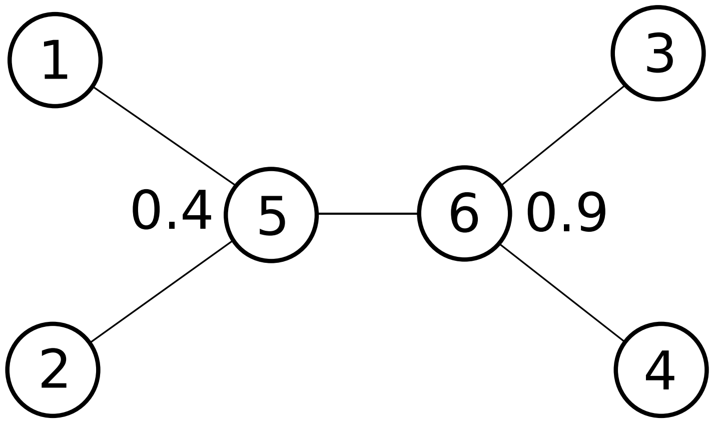

Figure 3.1: Example topology with node PDRs

In this example fifteen different paths can be found. The list of paths is in the table 3.1 (the paths consisted only from one link are excluded).

Path 1	1 - 5 - 2
Path 2	1 - 5 - 6
Path 3	2 - 5 - 6
Path 4	1 - 5 - 6 - 3
Path 5	1 - 5 - 6 - 4
Path 6	2 - 5 - 6 - 3
Path 7	2 - 5 - 6 - 4
Path 8	3 - 6 - 4
Path 9	5 - 6 - 3
Path 10	5 - 6 - 4

Table 3.1: Example of all paths in the topology (one link paths excluded)

The PDR for every path strongly depends on PDRs of the intermediate nodes 5 and 6, because only these two nodes are responsible for the message retransmissions. The PDR of the path can be calculated as a multiplication of PDRs of every intermediate node in the path. The resulting PDRs can be seen in the table 3.2.

Path 1	0.4
Path 2	0.4
Path 3	0.4
Path 4	0.4 * 0.9 = 0.36
Path 5	0.4 * 0.9 = 0.36
Path 6	0.4 * 0.9 = 0.36
Path 7	0.4 * 0.9 = 0.36
Path 8	0.9
Path 9	0.9
Path 10	0.9

Table 3.2: Example of PDR calculation for every path

The PDR can be from this point of view considered as a probability of successful packet delivery. It explains why the PDR values along the path need to be multiplied. The PDR value for every path can be easily measured in every node. The main idea of this solution is that PDR values of every path depends on the underlying topology. Lets consider that these PDR values were measured in some functioning network with the same topology and the original PDR values assigned to intermediate nodes are unknown. What can be said about intermediate nodes if the PDR of every path, where the investigated node presents, is known? In table 3.3 the list of paths according to the presence of the intermediate node can be seen.

Investigated node	Path list
5	Paths 1 - 7
6	Paths 4 - 10

Table 3.3: Paths by investigated node

With the knowledge of path PDRs it is possible to create data instances in which every input is a path PDR with investigated intermediate node. The example of such data instances is in the table 3.4.

5	0.4 0.4 0.4 0.36 0.36 0.36 0.36
6	0.36 0.36 0.36 0.36 0.9 0.9 0.9

Table 3.4: Example of data instance based on all paths

With the knowledge of the underlying topology it can be very easily determined that the PDR of the node 5 is 0.4 and the node 6 has PDR 0.9.

The recognition of the PDR from this data seems to be easy, however the count of paths in the topology depends quadratically on the count of nodes. With more complicated topology it is almost impossible to find correct and unambiguous answer - what is the PDR of the investigated node. Even the question which intermediate node crosses the threshold 0.5 would be almost impossible to answer with a conventional algorithm.

The task seems to be ideal for NN, however the large number of inputs increases the complexity of the learning process. This count of inputs can be decreased by averaging all path PDRs regarding to investigated node on every other nodes that use this node. Only paths that use the investigated node are involved in calculation.

The maximal count of all inputs equals to the count of all nodes in the topology minus one. In the table 3.5 the suggested calculation for all instances is presented. Every row in this table shows the calculation of average from all paths in which the investigated node participates. The middle column holds calculation example for investigated intermediate node 5 and the right one is for node 6. The input values in the left column are calculated values from the table 3.2. It can be seen that the number of inputs for every investigated node is the count of all nodes which use the investigated node for communication.

Investigated nodes $->$	5	6
Node 1:	(0.4+0.4+0.36+0.36)/4	(0.36+0.36)/2
Node 2:	(0.4+0.4+0.36+0.36)/4	(0.36+0.36)/2
Node 3:	(0.36+0.36)/2	(0.9+0.9+0.36+0.36)/4
Node 4:	(0.36+0.36)/2	(0.9+0.9+0.36+0.36)/4
Node 5:	-	(0.9+0.9)/2
Node 6:	(0.4+0.4)/2	-

Table 3.5: The average path PDRs

The resulting instances are shown in the table 3.6. The count of inputs is now small enough and it is still possible to estimate the PDRs of investigated nodes.

Investigated node	Resulting instance
5	0.38 0.38 0.36 0.36 0.4
6	0.36 0.36 0.63 0.63 0.9

Table 3.6: The example of the final instances based on the paths average on every node

The final data instance describes the current state of the network from the nodes point of view. In real network this instance can be very easily created. Every node in the network can calculate the PDR of the communication with any other node from the network. This measured value is in fact the separate path PDR. The topology of the network is known too, therefore it is possible to build the data instance for every investigated intermediate node.

3.2 Training of NN

In the previous section the creation of data instance was demonstrated. The example was simple enough to see that the searched value of PDR is directly presented in the input data. With more complicated topology it is not possible to detect the PDR as simple as it was in this example. Fortunately, the PDRs of every node are involved in the calculation of inputs in the instance. The change of the node PDR results in changes in some inputs depending on the underlying topology. The examples of instances created on the same topology with different PDRs can be seen in tables 3.7 and 3.8. It is not possible to create a conventional algorithm which comprises all possible topology configurations.

Investigated node	Resulting instance
5	0.27 0.27 0.24 0.24 0.3
6	0.24 0.24 0.52 0.52 0.8

Table 3.7: The example of final instances with $PDR_{node5} = 0.3$ and $PDR_{node6} = 0.8$

Investigated node	Resulting instance
5	0.17 0.17 0.14 0.14 0.2
6	0.14 0.14 0.42 0.42 0.7

Table 3.8: The example of final instances with $PDR_{node5} = 0.2$ and $PDR_{node6} = 0.7$

The main goal of this work is to exploit the advantages of NN in order to determine which nodes have crossed the predefined threshold and estimate the PDRs of intermediate nodes (if it is possible). This is an ideal problem to be solved by the NN - many data inputs depending on some hidden system and one output. Since every intermediate node presents one unique problem, the unique NN needs to be assigned to every intermediate node. It means that maximal count of needed NNs is equal to the count of all nodes in the topology and the maximal count of inputs for every instance equals to the count of all nodes minus one.

The basic problem of NN learning process is the need for sufficient number of data instances with known result. It seems to be very complicated to collect convenient data from real topology. Fortunately, the final values in data instances depend only on nodes PDRs and current topology. Since the topology is stable and node PDRs are known, it is possible to artificially create suitable set of data instances with the known output. All node PDRs can be randomly generated and the output value can be calculated based on these PDRs. With this technique a sufficient amount of training data can be generated.

Note: Now the question "where should the NNs be placed" raises. This problem is beyond the scope of this work. In this work only the suitability of NNs for detecting nodes with the small PDR is investigated. Some possibilities will be discussed in the chapter 6.

3.3 In search of solution: wrong ways

The laconic description of suggested solution is a result of extensive thinking and several dead end ways. First I assumed, that I can just take opinions of other nodes for detection if the node is malfunctioning. After setting up an experiment and processing results we realized, that NN had no chance to learn successfully, because we were training it with instances of different problem. Even if the number of opinions in two instances were the same, which means these instances can be used for training particular NN, they belong to different problems, as the constructed situation arises around different target nodes.

Chapter 4

Testing environment

There are several phases of experiment. First involves the network simulation to collect data for NN learning and testing. Next phase is NN learning. And the last and most interesting stage is applying trained models of NN on the testing data.

4.1 Tools used

Network simulation was done in Omnet++ 4.6. OMNeT++ is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators.[22]

Term network here is meant in broad sense, it can be wire, wireless, even queueing networks and specific functionality for example support of sensor, ad-hoc networks, Internet protocols and performance analysis is provided by model frameworks, developed independently. In this research Inet framework was used.

Omnet++ offers users an Eclipse-based IDE, graphical runtime environment and other tools and extensions. Its simulation platform provides user with large set of predefined components and convenient manner to define new one.

Basic building block for simulation is module, and several modules together can create compound module, which can be used in another compound modules without limiting the number of nested modules. Assembling in done using a high-level language NED. Modules communicate by sending messages, which also can be easily constructed by so called message definitions. User describes message contents, and the corresponding C++ code is automatically generated from the definitions.

As for our purposes there is no need in graphical runtime environment luckily has command-line user interface for simulation execution. OMNeT++ runs on Windows, Linux, Mac OS X, and other Unix-like systems. This research was performed in Ubuntu 14.04.

Two implementations of NN were tried: OpenNN library and FaNN.

OpenNN is an open source class library written in C++ programming language which implements neural networks. [23] On high-level is consists of three classes: NeuralNetwork, PerformanceFunctional and TrainingStrategy, relationships between them are follows. A neural network has assigned a performance functional. A performance functional is improved by a training strategy. [24] There is also DataSet class, which represents data for learning NN and obtaining desired model. *Model* is the output NN with adjusted parameters (weights) for particular problem solving. This library is able to perform both classification and regression tasks.

FANN stands for fast artificial neural network library. It is a free open source neural network library, which implements multilayer artificial neural networks in C with support for both fully connected and sparsely connected networks. [25] It is cross-platform, well-documented and easy to use. Several different activation functions are implemented in it. The library has bindings for PHP, Python, Delphi and Mathematica and the library also became accepted in the Debian Linux distribution.[26]

4.1.1 Omnet++ simulation

Realization of this phase took surprisingly the greater part of time. It took a while to get familiar with environment and features it offers. Construction of network for the needs of the experiment faced with choice of relevant modules, which required understanding of their structure and behavior. And this knowledge then used for creation of new module.

Simulation of network (creating topologies) and generating data for experiments (data for learning and testing NN) was done using Omnet++ IDE. Several modules from Inet framework were used. Some of them were adjusted and new one was written specially for simulation. Network is presented on the Figure 4.1.

Structure of node is presented of Figure 4.2. It has two interfaces: wireless NIC (wlan[0]) and loopback interface(lo0), which are stored in interface table.

Status module keeps information is the node is up. Mobility module takes care about node's position. In our simulation for simplicity nodes are not moving and initial position is defined randomly within the set playground.

Routing table stores data about routes: destination, next hop, interface, source (manual, routing protocol) administrative distance, metric and routing protocol specific data.

Network layer performs routing with the help of AODV module. When it receives packet from higher layer it makes a request to routing table, and if the route exists, add control information to the packet and send it NIC. If the route to the destination was not found in te routing table, network layer sends request to AODV to find the route. After AODV found the route, packet can finally be forwarded to lower layer. AODV module was adjusted in such a

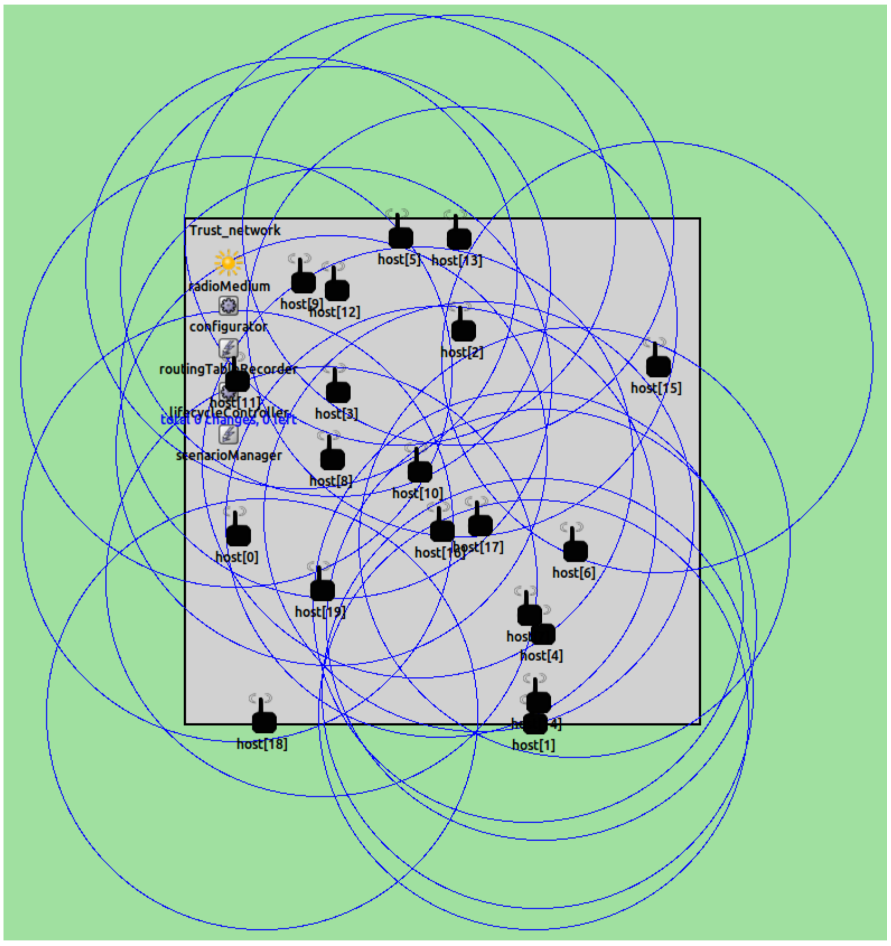

Figure 4.1: Structure of network in Omnet++ simulation

way, that after finding a route to the destination, it inserts the entry to the routing table, setting the route timeout to a number, large enough for the route to be valid till the end of the simulation run.

Main module that implements simulation logic is trafficGenerator. First it sends probe packets to all other nodes to discover the topology. So now the routing table of each node contains routes to all other nodes, specifically addresses of next hop nodes on the route to destination.

On this step there is a check if the topology constructed is connected, meaning that each node has at least one node in its radio range. If it happens that the topology is disconnected, it is dropped and next attempt follows. Next step it to collect information about paths, i.e. identify all intermediate nodes. After this step is accomplished there is all information, needed to perform data generation (synthetic simulation of PRD measurement).

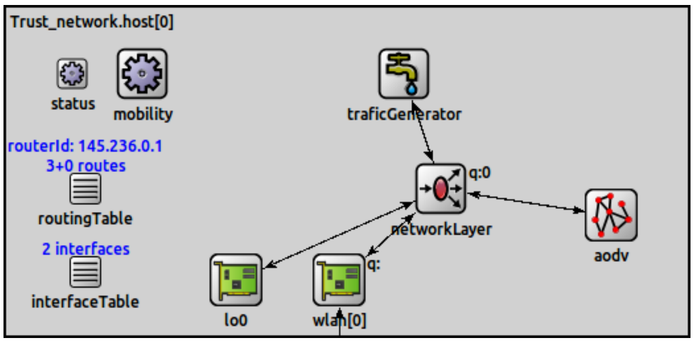

Figure 4.2: Structure of node in Omnet++ simulation

4.2 NN learning and adjusting parameters

After data collected NN training was executed. First try was to use OpenNN. First experience with OpenNN encountered with data loading problem, however this broken reading of an XML file was repaired. Then this application was ready for experiments, but although at the beginning it looked promising, then the experimentation faced with an another bug in implementation and it became clear that batch learning cannot be performed. In this case the problem was hidden deeply in the logic of application and the repair seemed to be too time consuming.

FaNN proved to work and as the name claims, this NN library is really fast. An application which allowed to set all important parameters by command line arguments was created. I did not notice any problem with this library during execution of all experiments.

4.3 Data construction

The Omnet++ was used for the instance generation. There were created five sets every with 100 different topologies. In all sets 5000 instances for every intermediate nodes in all topologies were constructed. The first 1000 instances were used for training data and the rest for testing. The threshold was set to the value of 0.5. There are two types of instances, which differ in the output:

- Negative - the investigated intermediate node has PDR above threshold
- Positive - the investigated intermediate node has PDR below threshold

There is in total one half of positive and one half of negative instances in every set.

Every set has different method of node PDRs construction:

1. all node PDRs are generated by uniform distribution with range from 0.5 to 1.0, positive instances have PDR of the investigated node generated by uniform distribution from range 0.01 to 0.49.
2. all node PDRs are generated by uniform distribution with range from 0.5 to 1.0 and one random node has PDR from range 0.01 to 1.0, positive instances have PDR of the investigated node generated by uniform distribution from range 0.01 to 0.49.
3. all node PDRs are generated by uniform distribution with range from 0.01 to 1.0, positive instances have PDR of the investigated node generated by uniform distribution from range 0.01 to 0.49.
4. all node PDRs are generated by normal distribution with mean equal to 0.75 and standard deviation equal to 0.25, positive instances have PDR of the investigated node generated by uniform distribution from range 0.01 to 0.49.
5. all node PDRs are generated by normal distribution with mean equal to 0.75 and standard deviation equal to 0.25 and one random node has PDR from range 0.01 to 1.0, positive instances have PDR of the investigated node generated by uniform distribution from range 0.01 to 0.49.

Not all of these sets were used for the experiments from chapter 5, however the complete list of results is presented in the Appendix D.

4.4 Data format

The chosen NN implementation FaNN requires two types of data format, first for training data and second for testing data. In both cases there are very simple textual files. Furthermore, there need to be created a special data set for detection and estimation, because they represents different type of problems from the NN point of view (3). I created a script which converts data files constructed by Omnet++ to the proper format required by detection or estimation.

4.4.1 Detection

The data format for training data is little more complicated. On the following example the input with six instances in total can be seen. Every instance has five inputs and exactly one output:

```
6 5 1
0.405549 0.694223 0.440352 0.694223 0.440352
1
0.282362 0.274169 0.271964 0.274169 0.271964
```

```
0
0.540561 0.581132 0.381087 0.581132 0.381087
1
0.233959 0.298616 0.232437 0.298616 0.232437
0
0.713560 0.523443 0.291015 0.523443 0.291015
1
0.271328 0.399818 0.236811 0.399818 0.236811
0
```

In the example every odd instance is positive and every even is negative.

The testing data file format is simpler. Whole instance is inside one line. In the following example ten testing instances with five inputs and one output are presented:

```
0.534804 0.743309 0.530129 0.743309 0.530129 1
0.015796 0.016523 0.013020 0.016523 0.013020 0
0.536254 0.583671 0.530174 0.583671 0.530174 1
0.267581 0.387889 0.251876 0.387889 0.251876 0
0.631034 0.854609 0.704724 0.854609 0.704724 1
0.039097 0.039840 0.028928 0.039840 0.028928 0
0.502005 0.634014 0.450335 0.634014 0.450335 1
0.339783 0.431359 0.216874 0.431359 0.216874 0
0.362893 0.658696 0.579637 0.658696 0.579637 1
0.215415 0.209800 0.176126 0.209800 0.176126 0
```

4.4.2 Estimation

The data for estimation are almost identical to that for detection. The only difference is in output value which is not a binary one. This is the example of training data:

```
6 5 1
0.392869 0.597670 0.333235 0.597670 0.333235
0.641888
0.178368 0.253757 0.162813 0.253757 0.162813
0.342506
0.501104 0.395111 0.216980 0.395111 0.216980
0.594301
0.022631 0.038991 0.022856 0.038991 0.022856
0.045603
0.329764 0.526783 0.415047 0.526783 0.415047
0.558883
0.000576 0.000533 0.000389 0.000533 0.000389
0.000801
```

And the example of testing is here:

```
0.320386 0.499555 0.333538 0.499555 0.333538 0.537222
0.077176 0.091607 0.080138 0.091607 0.080138 0.104488
0.392090 0.439251 0.362681 0.439251 0.362681 0.626878
0.239972 0.181316 0.135093 0.181316 0.135093 0.337902
0.460870 0.475353 0.250051 0.475353 0.250051 0.764832
0.319197 0.339044 0.203384 0.339044 0.203384 0.462905
0.349613 0.524040 0.374240 0.524040 0.374240 0.596328
```

4.5 Data structure

Every set defined in the previous section is constructed by Omnet++ and it has its own folder. After construction it has following structure 4.5.

```
exp0 ................................ The folder with raw data of set 1
├── 0 ....................... The folder with instance files for topology 0
└── 2 .............. Not all topologies were used - number 1 was skipped
    ├── 0.csv ................................. Instance file for node 0
    └── 2.csv ...... Not all nodes are intermediate - number 1 was skipped
```

Figure 4.3: Data structure of the data folder

4.6 Evaluating results

At first a calculation of a performance result needs to be defined. This work is focused on two types of problems - detection and estimation. In detection I am interested in the information: How many instances were successfully evaluated. However in estimation the question is: How are the results close to the correct values.

For detection problem the performance result can be defined as:

$$R_D = 1 - \frac{C_{success}}{C_{all}}, \tag{4.1}$$

where $C_{success}$ is the count of all successfully evaluated instances and C_{all} is the count of all instances. The result R is a number between 0 and 1.

For estimation problem the performance result can be defined as:

$$R_E = 1 - \frac{Sum_{diff}}{C_{all}}, \tag{4.2}$$

where Sum_{diff} is the sum of all differences between estimated PDR and expected PDR. The result R is again the number between 0 and 1.

The performance result is calculated for every instance file. In order to see how successful was the whole experiment it is necessary to calculate some overall values:

- Overall results
- Detailed results

These result types will be described in following sections.

4.6.1 Overall results

This result consists of four values - average, median, minimum and maximum. It is calculated over all performance results of all instance files generated for the experiment (all topologies together).

4.6.2 Detailed results

This is calculated in the same way as the overall result however results are grouped by the inputs count. Values in this result type are: count of inputs, average, median, minimum, maximum and count of instances. The last value is the number of all instances with desired inputs count.

Chapter 5

Experiments

In the previous chapter the process of data instance creation was defined and the suggestion how to train the NN was made. In this chapter a set of experiments which test the performance of NNs on detection nodes with low PDR and on estimation of node PDRs will be presented. The detection uses the NN for classification and estimation uses the regression. For both tasks the same type of NN can be used and training process is almost identical. The only difference is the type of output (which is based on the generated inputs). The learning, detection and estimation parts will be described separately.

5.1 Data for learning

The learning process can be influenced by many factors:

- type of NN
- learning method
- count of layers
- count of neurons
- count of learning epochs
- count of inputs
- count of instances
- quality of instances

Each of these factors presents separate choice and together create huge amount of possible combinations. It would be very hard to investigate every possibility. In the following subsections the reasons for choosing particular values will be given.

5.1.1 Type of NN

This research was bound to operate with available implementations of NN. Writing the own implementation is demanding and time-consuming task and is out of scope if this thesis. As it was already mentioned in subsection 1.3.2 each NN is characterized by:

- model of neurons (details on their inherit properties and functions)
- topology of the network
- learning method

Learning is discussed in next subsection. As for topology, both OpenNN and FaNN are implementations of multilayer feed-forward neural networks. The feed-forward neural network was the first and arguably most simple type of artificial neural network devised.[27] In search of suitable parameters for solution several models of neuron were tested: perceptron with sigmoid activation function and RBF neuron with Gaussian activation function. It turned out that on constructed data perceptrons learn faster and have better performance.

5.1.2 Learning method

FaNN implementation uses Rprop backpropagation algorithm as default. Rprop, short for resilient backpropagation, is a learning heuristic for supervised learning in feedforward artificial neural networks. Rprop is one of the fastest weight update mechanisms.[28]

As it was mentioned in subsection 1.3.3 at the beginning all weights are initialized randomly. In each training cycle NN is feed up with inputs and after receiving the response from network, it is compared to the correct output and values of weights are adjusted. The Backpropagation algorithm calculates gradient of cost function with respect to weights. So gradient is the vector whose components are the partial derivatives of weights. Gradient shows how fast the cost changes when the weights change. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the cost function.[17]

The adaptation-rule works as follows: every time the partial derivative of the corresponding weight changes its sign, which indicates that the last update was too big and the algorithm has jumped over a local minimum, the update-value is decreased [29] by the factor η^-, where $\eta^- < 1$. If the derivative retains its sign, the update-value is slightly increased by a factor of η^+, where $\eta^+ > 1$. The update values are calculated for each weight in the above manner, and finally each weight is changed by its own update value, in the opposite direction of that weight's partial derivative, so as to minimize the total error function. η^+ is empirically set to 1.2 and η^- to 0.5.[28]

Parameters of this learning method such are momentum and learning rate were left with default values, as they showed good performance.

5.1.3 Count of layers

Number of hidden layers should respond to the difficulty of the problem. Generally the harder the problem more hidden layers are required. One hidden layer is sufficient for the large majority of problems. [30] Several experiments showed that one layer is sufficient for this research.

5.1.4 Count of neurons and learning epochs

These two factors are the most variable ones. Several experiments were conducted to figure out the best combination of these two factors. One topology was chosen with one constant PDR configuration. The learning process does not get the same results for the same inputs because the initial weights configuration is randomly generated. Therefore, it was conducted ten detections and ten estimations for every investigated parameter combination. The dependency of result on the count of neurons and learning epochs can be seen in the table 5.1. This experiment revealed the simplicity of the given problem. It is obvious that detection problem can be very easily solved by only one neuron in the hidden layer. The quality of the detection is more influenced by the count of learning epochs. The most interesting results are highlighted. In the table 5.2 there are results for node PDRs estimation.

It is obvious from both tables that the quality of results differs very slightly even for great changes of learning factors. The differences are in fractions of percent. The count of neurons has very negligible influence on results however the value of 5 neurons seems to be slightly better than the other ones.

The count of learning epochs changed the result quality significantly between 1000 and 2000. Other changes increased the quality of results very slightly. The count of learning epochs influences not only the results but the total time of learning process. Therefore the lowest possible count is suggested.

According to the previous observations 5 neurons in the hidden layer and 2000 learning epochs were chosen for the next experiments.

5.1.5 Count of inputs

According the section 3.1 the count of inputs depends on the count of nodes in the used topology. The value 20 was chosen as the count of nodes. This number is the trade-off between two factors:

- The topology for experiments needs to be large enough in order to be more similar to the real ad-hoc networks.
- The topology needs to be small enough to be controlled by manual calculations.

Neurons count \ Epochs count	1000	2000	3000	4000	5000
1	0.9834	0.9838	**0.9842**	**0.9844**	**0.9844**
2	0.9836	**0.9840**	**0.9841**	**0.9842**	**0.9841**
3	0.9836	**0.9840**	**0.9841**	**0.9840**	0.9838
4	0.9837	0.9839	**0.9840**	0.9839	0.9838
5	0.9837	**0.9840**	0.9839	**0.9840**	0.9838
6	0.9836	0.9839	0.9839	0.9839	0.9837
7	0.9836	0.9800	0.9839	0.9838	0.9839
8	0.9837	0.9839	0.9838	0.9839	0.9824

Table 5.1: Quality of results depending on count of neurons and learning epochs - detection. (Best results are highlighted.)

Neurons count \ Epochs count	1000	2000	3000	4000	5000
1	0.9710	0.9727	0.9732	0.9735	0.9735
2	0.9726	0.9738	**0.9742**	**0.9744**	**0.9746**
3	0.9729	**0.9741**	**0.9744**	**0.9747**	**0.9749**
4	0.9729	**0.9740**	**0.9743**	**0.9748**	**0.9747**
5	0.9729	**0.9742**	**0.9744**	**0.9747**	**0.9747**
6	0.9728	0.9739	**0.9744**	**0.9744**	**0.9749**
7	0.9729	0.9739	**0.9742**	**0.9747**	**0.9749**
8	0.9728	0.9738	**0.9742**	**0.9748**	**0.9746**

Table 5.2: Quality of results depending on the count of neurons and learning epochs - estimation

5.1.6 Count of instances

After a few trials it became clear that 1000 training instances is sufficient for all experiments.

5.1.7 Quality of instances

As it was mentioned in the subsection 1.3.4 the set of training instances should uniformly cover the problem set. In this work the problem is created from a combination of node PDRs. The sufficient set of training instances can be obtained by random generation of PDR values. The task is to create instances as different as possible. For the PDR values generation two different distribution - uniform and normal were chosen.

The uniform distribution covers the investigated problem sufficiently, however it is expected that it is to far from reality. In real environment there are many external influences on PDR values, which can cause some extreme configurations which have low probability to be generated by uniform distribution. The idea is to train NN on data instances created with uniform distribution and test it on data instances created with normal distribution. In case of a good results obtained, the NN trained on uniform distribution can have a good performance in real environment.

Note: The range of generated PDR values differs by experiment and type of problem. The concrete ranges will be specified in following sections.

5.2 Detection

The main purpose of detection is to detect nodes which crossed some predefined threshold. In the detection problem it is expected that all node PDRs are higher than threshold until one of nodes is broken and its PDR decreases below threshold. The problem ends with positive detection of this broken node. What happens after the detection is not a part of this work.

In the following sections the experiments will be described. The PDR threshold value for all experiments is set to 0.5. In all experiments 100 different topologies were generated. For every topology and experiment 1000 training instances and 4000 testing instances were created.

5.2.1 Experiment 1

Lets assume that only one node breaks at the time. For this situation there is prepared NN trained on data set 1 (4.3). The trained NN was used on the testing data again from set 1. The overall performance of the NN can be seen in table 5.3.

Average	Median	Minimum	Maximum
0.9878	0.9885	0.9694	0.9975

Table 5.3: Experiment 1 - overall results

In the table 5.4 can be found detailed results of experiment 1.

The performance of detection differs by inputs count. It could be clearly seen that performance NNs with small amount of inputs is the worst and it has the greatest dispersion.

5.2.2 Experiment 2

In this experiment the learned NN from experiment 1 is used, however it classifies testing data from data set 4. This experiment should demonstrate the ability of NN trained on uniform data to work on different type of data. The results are in table 5.5.

The performance is slightly worse than in experiment 1.

Count of inputs	Average quality	Median	Minimum	Maximum	Instances count
3	0.9601	0.9703	0.8588	1.0000	121
4	0.9663	0.9965	0.8268	0.9998	99
5	0.9757	0.9964	0.8798	0.9998	78
6	0.9862	0.9973	0.9035	1.0000	61
7	0.9858	0.9963	0.8985	1.0000	77
8	0.9881	0.9965	0.8915	0.9998	57
9	0.9932	0.9961	0.9513	0.9998	46
10	0.9956	0.9965	0.9743	1.0000	74
11	0.9948	0.9959	0.9743	0.9993	68
12	0.9946	0.9963	0.9065	0.9993	71
13	0.9938	0.9965	0.9398	0.9998	63
14	0.9954	0.9958	0.9835	0.9990	65
15	0.9938	0.9958	0.9380	0.9988	61
16	0.9944	0.9955	0.9628	0.9993	68
17	0.9945	0.9953	0.9813	0.9993	94
18	0.9948	0.9955	0.9820	0.9998	119
19	0.9943	0.9948	0.9830	0.9993	218

Table 5.4: Experiment 1 - detailed results

Average	Median	Minimum	Maximum
0.9781	0.9791	0.9435	0.9967

Table 5.5: Experiment 2 - overall results

5.2.3 Experiment 3

The first two experiments were prepared in advance and their results were expected. During my experiments I tried different combinations of learned NN and type of testing data and I got some surprising result. When I tried the NN trained on data set 4 with data set 1, I got better performance than in experiment 2. The results can be seen in the table 5.6.

Average	Median	Minimum	Maximum
0.9854	0.9862	0.9670	0.9964

Table 5.6: Experiment 3 - overall results

This result is even more surprising because the performance of this NN is slightly better on uniform data than on normal data on which it was trained (table 5.7.

Average	Median	Minimum	Maximum
9822	9834	9514	9974

Table 5.7: Experiment 3 - overall results on normal data

5.2.4 Experiment 4

Because the performance in first thee experiments was surprisingly good, I decided for the last hardest experiment with detection. I stated before that only one broken node in the topology is expected. What if I try the NN trained in the experiment 1 for the topologies with random count of broken nodes? For this experiment there was used a data set 3 on NN trained on data set 1. According the table 5.8 the results are still good enough.

Average	Median	Minimum	Maximum
9696	9699	9326	9957

Table 5.8: Experiment 4 - overall performance

5.2.5 Other experiments with detection

Much more experiments was conducted, however the results were not so interesting, therefore are not commented. Results from other experiments and detailed results are placed in the Appendix D.

5.3 Estimation

Another kind of problem is an estimation of the PDR values in the whole topology. There is no threshold PDR value defined.

5.3.1 Experiment 5

In this experiment the NN trained on data set 3 is used. For testing the same data like for training was used. The resulting performance is shown in the table 5.9.

Average	Median	Minimum	Maximum
9660	9675	9412	9776

Table 5.9: Experiment 5 - overall results

It can be seen that performance is over 95% which is really great for the indirect estimation from averaged values. If we take a look at the detailed

Count of inputs	Average quality	Median	Minimum	Maximum	Instances count
3	0.9275	0.9612	0.8123	0.9818	121
4	0.9375	0.9750	0.8020	0.9822	99
5	0.9478	0.9752	0.8247	0.9838	78
6	0.9613	0.9760	0.8526	0.9818	61
7	0.9632	0.9759	0.8501	0.9823	77
8	0.9647	0.9760	0.8334	0.9832	57
9	0.9726	0.9765	0.8934	0.9830	46
10	0.9752	0.9764	0.9323	0.9813	74
11	0.9769	0.9771	0.9505	0.9823	68
12	0.9750	0.9767	0.8502	0.9812	71
13	0.9746	0.9770	0.8953	0.9828	63
14	0.9770	0.9771	0.9633	0.9823	65
15	0.9731	0.9764	0.8832	0.9818	61
16	0.9766	0.9771	0.9336	0.9822	68
17	0.9764	0.9767	0.9535	0.9820	94
18	0.9770	0.9770	0.9710	0.9832	119
19	0.9757	0.9772	0.6467	0.9828	218

Table 5.10: Experiment 5 - detailed results

results sorted by inputs count in table 5.10, we can notice poor performance on low inputs instances. The behaviour is similar to the experiment 1.

5.3.2 Experiment 6

Previous experiment has a great results on the ideal data. In this experiment the same NN will be tested on data set 4. This experiment should simulate the performance of NN under different conditions. As it was expected the performance is worse than in experiment 5 however it is according table 5.11 still over 90% which gives a chance for similar results in the real environment.

Average	Median	Minimum	Maximum
9249	9263	8679	9773

Table 5.11: Experiment 6 - overall results

Chapter 6

Discussion and possible further work

6.1 Suitability of NN for detection od broken nodes

Generally it appeared that NN is applicable on the problem of trust how it is defined in this thesis.

From the experiment results it can be seen that neural networks, which have small number of inputs demonstrate worse performance. This can be explained by follows. It there are several paths through the same nodes in a row and some untrusted node is among them, it is hard to detect precisely which one. But NN with number of inputs > 5 showed quite impressive quality of output.

It was unexpectedly discovered, that NN trained on data generated with normal distribution cope better with data generated with uniform distribution than with their own data.

6.2 Suitability of NN for PDR estimation

Here it can be observed the same situation when using models with small number of inputs. But imprecise value of PDR has not that critical impact as wrong classification of the node.

6.3 Limitations

Topology changes result in necessity to perform new learning for NN. But learning can be performed quite quickly and with the knowledge of topology any traffic generation is not required. Routing protocol (AODV in our case) is used purely for topology discovery. AODV route lifetime according to AODV

Routing RFC 3561 is network traverse time [31] and thus quite short. For the aims of this research this parameter was adjusted, but generally AODV is not suitable for such kind of experiment where topology better remain unchanged. AODV was chosen as it is the most popular protocol for Ad-hoc networks.

6.4 Future work

Later the research can be extended to data generated with another random distribution or collected from the real traffic simulation. Also another parameters of NN can be tried.

Conclusion

This work aimed to find out if neural networks are applicable to problem of trust in Ad-hoc networks. After stating the objectives and defining the problem there were performed several experiments, that proved the neural network to be able to cope with the problem with good performance. Different parameters of NN were investigated and the optimal ones proposed and their choice is explained. There are limitations to the proposed solution which were discussed in the previous chapter.

Bibliography

[1] Stergiou, C. What is a Neural Network? Available from: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/cs11/article1.html

[2] Azmi, R.; Hakimi, M.; Bahmani, Z. Dynamic Reputation Based Trust Management Using Neural Network Approach. In *IJCSI International Journal of Computer Science Issues*, 2011.

[3] Imana, E. Y.; Ham, F. M.; Allen, W.; et al. Proactive Reputation-Based De f ense f or MANETs Using Radial Basis Function Neural Networks.

[4] Mujica-V, V.; Sisalem, D.; Popescu-Zeletin, R. NEURAL-Trust: Stimulating Cooperation Engagements in Multi-Agents Systems.

[5] Balanis, C. A. *Antenna Theory: Analysis and Design*. Wiley-Interscience, 2005, ISBN 0471714623.

[6] Omnidirectional Antenna Radiation Pattern. Available from: http://www.mpantenna.com/omnidirectional-antenna-radiation-patterns/

[7] Singal, T. *Wireless communications*. Tata McGraw-Hill Education, 2010, ISBN 9781259083501, 595 pp.

[8] Perkins, C.; Royer, E. Ad-hoc on-demand distance vector routing. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, Feb 1999, pp. 90–100, doi:10.1109/MCSA.1999.749281.

[9] Boukerche, A.; Darehshoorzadeh, A. Opportunistic Routing in Wireless Networks: Models, Algorithms, and Classifications. *ACM Comput. Surv.*, volume 47, no. 2, Nov. 2014: pp. 22:1–22:36, ISSN 0360-0300, doi:10.1145/2635675. Available from: http://doi.acm.org/10.1145/2635675

[10] Cook, K. S. (editor). *Trust in Society.* Russell Sage Foundation Series on Trust, 2003.

[11] Cho, J.-H.; Swami, A.; Chen, I.-R. A Survey on Trust Management for Mobile Ad Hoc Networks. *Communications Surveys Tutorials, IEEE*, volume 13, no. 4, Apr 2011: pp. 562–583, ISSN 1553-877X, doi:10.1109/SURV.2011.092110.00088.

[12] Li, H.; Singhal, M. Trust Management in Distributed Systems. *Computer*, volume 40, no. 2, Feb. 2007: pp. 45–53, ISSN 0018-9162, doi:10.1109/MC.2007.76.

[13] MacKenzie, A.; Wicker, S. Game theory and the design of self-configuring, adaptive wireless networks. *Communications Magazine, IEEE*, volume 39, no. 11, Nov 2001: pp. 126–131, ISSN 0163-6804, doi:10.1109/35.965370.

[14] Jøsang, A.; Presti, S. Analysing the Relationship between Risk and Trust. In *Trust Management*, *Lecture Notes in Computer Science*, volume 2995, edited by C. Jensen; S. Poslad; T. Dimitrakos, Springer Berlin Heidelberg, 2004, ISBN 978-3-540-21312-3, pp. 135–145, doi:10.1007/978-3-540-24747-0_11.

[15] Why Trust is not Proportional to Risk. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, April 2007, pp. 11–18, doi:10.1109/ARES.2007.161.

[16] (U.S.), D. N. N. S. *DARPA Neural Network Study.* Virginia: AFCEA International Press, ist ed. edition, 1988.

[17] Backpropagation. Available from: `https://en.wikipedia.org/wiki/Backpropagation`

[18] Momentum and Learning Rate Adaptation. Available from: `https://www.willamette.edu/~gorr/classes/cs449/momrate.html`

[19] Basic Concepts for Neural Networks. Available from: `https://www.cheshireeng.com/Neuralyst/nnbg.htm`

[20] Network Design. Available from: `http://leenissen.dk/fann/wp/help/advanced-usage/`

[21] Using Neural Networks With Regression. Available from: `http://deeplearning4j.org/linear-regression.html`

[22] OMNeT++ Discrete Event Simulator. Available from: `https://omnetpp.org`

[23] OpenNN neural networks. Available from: http://www.opennn.net/

[24] The software model of OpenNN. Available from: http://www.opennn.net/documentation/the_software_model_of_opennn.html

[25] Fast Artificial Neural Network Library. Available from: http://leenissen.dk/fann

[26] Neural Networks Made Simple. Available from: http://fann.sourceforge.net/fann_en.pdf

[27] Types of artificial neural networks. Available from: https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks

[28] Rprop. Available from: https://en.wikipedia.org/wiki/Rprop

[29] Riedmiller, M.; Braun, H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference On*, IEEE, 1993, pp. 586–591.

[30] Panchal, G.; Ganatra, A.; Kosta, Y.; et al. Behaviour analysis of multilayer perceptronswith multiple hidden neurons and hidden layers. *International Journal of Computer Theory and Engineering*, volume 3, no. 2, 2011: p. 332.

[31] Ad hoc On-Demand Distance Vector (AODV) Routing. Available from: https://www.ietf.org/rfc/rfc3561.txt

Appendix A

Acronyms

AODV Ad-hoc on demand distance vector protocol.

CSMA/CA carrier sense multiple access with collision avoidance.

IDE integrated development environment.

NED NEtwork Description.

NIC network interface card.

NN neural network.

P2P peer-to-peer.

PDR packet delivery ratio.

RBF radial basis function.

Rprop resilient backpropagation.

Appendix **B**

Contents of CD

- `readme.txt` .... the file with DVD contents description
- `data` .... the data files directory
  - `FaNN` .... original source codes of the Fast NN
  - `Omnet++` .... the directory with Omnet++ environment
    - `omnetpp-4.6-src.tgz` .... original source of the Omnet++
    - `Project.zip` ..... exported project for constructing data instances
  - `scripts` .... the directory with bash scripts for experiments
  - `data.zip` .... the raw data constructed by Omnet++
  - `models.zip` .... the archive file with data of all learned NN
  - `NN-eclipse` .... the exported eclipse project with source codes for NN
- `src` .... the directory with thesis source codes
- `text` .... the thesis text directory
  - `MastersThesis_Aubekerova.pdf` .. the Diploma thesis in PDF format

Appendix **C**

User manual for experimenting

This manual describes only the data manipulation and calculating results with existing NN which are already trained. The data instances construction and NN training process is not a part of this manual. For the simple experimenting the bash only is required.

In the following list all important scripts from the DVD (content of the directory data/scripts) are briefly described:

- calculateAverageByInputsCount - calculates and shows detailed results in the latex format. Requires only one argument - path to the results directory
- calculateAverageByTopology - calculates and shows overall results in minimal format. Requires only one argument - path to the results directory
- classifyFANN - evaluates detection process with the learned NN on testing data. Requires the path to the directory with testing data, path to the directory with stored NNs and path to the results folder (this will be created automatically)
- convertDataForFANN-classification - converts raw data constructed by Omnet++ into training and testing instances for detection. Requires the path to the directory with raw data and the name of the output directory. (Important note: it creates two directories - the first with suffix "`_learn`" for training data and the second with suffix "`_test`" for testing)
- convertDataForFANN-regression - the same as the script above, it only creates data for estimation.
- learnFANN - executes NN training. It requires path to the directory with training data without suffix "`_learn`" and path to the directory where

will be store data of trained NN. (Important note: The suffix "`_learn`" is not given in argument however it is expected by this script!)

- regressFANN - evaluates estimation process with the learned NN on testing data. Requires the path to the directory with testing data, path to the directory with stored NNs and path to the results folder (this will be created automatically)

In order to start with experiments it is necessary to follow these steps:

1. Extract content of the archive data.zip into scripts directory. This archive contains of five data sets names exp0, exp1, exp2, exp3 and exp4.
2. Extract content of the archive models.zip into scripts directory. This archive contains of five models for detection (with suffix 'c') and five models for estimation (with suffix 'r').
3. Next step is the preparation testing instances. Use scripts 'convertDataForFANN-classification' and 'convertDataForFANN-regression'. Example:

   `./convertDataForFANN-classification exp0 exp0_detection`.

   This will create two directories:

   `exp0_detection_learn` and `exp0_detection_test`.
4. Now you can start experiments. Following example evaluates detection with data set 1 on model trained on data set 2:

   ```
   ./classifyFANN exp0_detection models_exp1_r results_0on1_r
   ```
5. The detailed results can be printed as:

   ```
   ./calculateAverageByInputsCount results_exp0_on_model_exp1_r
   ```
6. The overall results can be printed as:

   ```
   ./calculateAverageByTopology results_exp0_on_model_exp1_r
   ```

Appendix D

More results

D.1 NN trained on data set 1 testing data set 1 - detection

Average	Median	Minimum	Maximum
0.9878	0.9885	0.9694	0.9975

Count of inputs	Average quality	Median	Minimum	Maximum	Instances count
3	0.9601	0.9703	0.8588	1.0000	121
4	0.9663	0.9965	0.8268	0.9998	99
5	0.9757	0.9964	0.8798	0.9998	78
6	0.9862	0.9973	0.9035	1.0000	61
7	0.9858	0.9963	0.8985	1.0000	77
8	0.9881	0.9965	0.8915	0.9998	57
9	0.9932	0.9961	0.9513	0.9998	46
10	0.9956	0.9965	0.9743	1.0000	74
11	0.9948	0.9959	0.9743	0.9993	68
12	0.9946	0.9963	0.9065	0.9993	71
13	0.9938	0.9965	0.9398	0.9998	63
14	0.9954	0.9958	0.9835	0.9990	65
15	0.9938	0.9958	0.9380	0.9988	61
16	0.9944	0.9955	0.9628	0.9993	68
17	0.9945	0.9953	0.9813	0.9993	94
18	0.9948	0.9955	0.9820	0.9998	119
19	0.9943	0.9948	0.9830	0.9993	218

D.2 NN trained on data set 2 testing data set 2 - detection

Average	Median	Minimum	Maximum
0.9864	0.9871	0.9654	0.9976

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9543	0.9660	0.8300	1.0000	121
4	0.9624	0.9965	0.7963	1.0000	99
5	0.9728	0.9967	0.8535	0.9998	78
6	0.9850	0.9965	0.9005	0.9998	61
7	0.9840	0.9958	0.8935	1.0000	77
8	0.9856	0.9963	0.8690	0.9995	57
9	0.9919	0.9961	0.9373	0.9995	46
10	0.9953	0.9968	0.9730	0.9995	74
11	0.9945	0.9959	0.9733	0.9993	68
12	0.9945	0.9965	0.8950	0.9993	71
13	0.9935	0.9960	0.9365	0.9993	63
14	0.9946	0.9953	0.9765	0.9993	65
15	0.9928	0.9950	0.9273	0.9993	61
16	0.9949	0.9960	0.9673	0.9995	68
17	0.9942	0.9957	0.9765	0.9993	94
18	0.9942	0.9948	0.9813	0.9988	119
19	0.9942	0.9950	0.9828	0.9993	218

D.3 NN trained on data set 3 testing data set 3 - detection

Average	Median	Minimum	Maximum
0.9777	0.9792	0.9395	0.9976

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9247	0.9515	0.7640	1.0000	121
4	0.9383	0.9950	0.7433	0.9998	99
5	0.9538	0.9959	0.7795	1.0000	78
6	0.9742	0.9970	0.8343	0.9995	61
7	0.9731	0.9953	0.8160	0.9998	77
8	0.9759	0.9963	0.7888	0.9995	57
9	0.9830	0.9949	0.8740	0.9985	46
10	0.9925	0.9960	0.9478	1.0000	74
11	0.9863	0.9949	0.6143	0.9995	68
12	0.9925	0.9960	0.8203	0.9990	71
13	0.9890	0.9948	0.8800	0.9993	63
14	0.9925	0.9953	0.9575	0.9985	65
15	0.9878	0.9943	0.8545	0.9993	61
16	0.9902	0.9943	0.9158	0.9985	68
17	0.9917	0.9939	0.9570	0.9983	94
18	0.9919	0.9933	0.9660	0.9995	119
19	0.9919	0.9933	0.9665	0.9993	218

D.4 NN trained on data set 4 testing data set 4 - detection

Average	Median	Minimum	Maximum
0.9822	0.9835	0.9514	0.9974

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9364	0.9548	0.7740	1.0000	121
4	0.9472	0.9953	0.7620	0.9998	99
5	0.9620	0.9968	0.8160	0.9995	78
6	0.9789	0.9970	0.8523	0.9998	61
7	0.9796	0.9963	0.8563	1.0000	77
8	0.9811	0.9960	0.8208	0.9998	57
9	0.9894	0.9955	0.9093	1.0000	46
10	0.9945	0.9968	0.9580	0.9995	74
11	0.9938	0.9963	0.9595	0.9993	68
12	0.9938	0.9963	0.8678	0.9995	71
13	0.9921	0.9958	0.9030	0.9988	63
14	0.9947	0.9958	0.9768	0.9988	65
15	0.9913	0.9953	0.9005	0.9998	61
16	0.9935	0.9956	0.9483	0.9988	68
17	0.9934	0.9955	0.9625	0.9988	94
18	0.9936	0.9950	0.9620	0.9993	119
19	0.9936	0.9948	0.9778	0.9993	218

D.5 NN trained on data set 5 testing data set 5 - detection

Average	Median	Minimum	Maximum
0.9816	0.9831	0.9478	0.9975

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9337	0.9548	0.7420	1.0000	121
4	0.9458	0.9958	0.7518	0.9998	99
5	0.9608	0.9969	0.8023	0.9998	78
6	0.9784	0.9970	0.8503	1.0000	61
7	0.9786	0.9960	0.8458	0.9995	77
8	0.9811	0.9960	0.8178	0.9998	57
9	0.9898	0.9957	0.9010	0.9995	46
10	0.9944	0.9958	0.9600	0.9998	74
11	0.9934	0.9960	0.9623	0.9995	68
12	0.9936	0.9963	0.8523	0.9995	71
13	0.9916	0.9960	0.8985	0.9995	63
14	0.9939	0.9953	0.9695	0.9998	65
15	0.9908	0.9948	0.8938	0.9985	61
16	0.9934	0.9953	0.9513	0.9985	68
17	0.9932	0.9950	0.9708	0.9993	94
18	0.9934	0.9948	0.9663	0.9990	119
19	0.9935	0.9943	0.9773	0.9988	218

D.6 NN trained on data set 4 testing data set 1 - detection

Average	Median	Minimum	Maximum
0.9855	0.9862	0.9670	0.9964

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9524	0.9643	0.8183	1.0000	121
4	0.9591	0.9938	0.8008	0.9998	99
5	0.9709	0.9950	0.8460	0.9998	78
6	0.9834	0.9960	0.8960	1.0000	61
7	0.9830	0.9948	0.8968	0.9998	77
8	0.9852	0.9955	0.8840	0.9998	57
9	0.9900	0.9950	0.9073	0.9993	46
10	0.9946	0.9957	0.9755	0.9998	74
11	0.9941	0.9954	0.9583	0.9993	68
12	0.9944	0.9963	0.8873	0.9995	71
13	0.9924	0.9955	0.9198	0.9993	63
14	0.9950	0.9960	0.9768	0.9995	65
15	0.9921	0.9940	0.9235	0.9988	61
16	0.9941	0.9951	0.9500	0.9993	68
17	0.9938	0.9950	0.9755	0.9993	94
18	0.9939	0.9945	0.9668	0.9995	119
19	0.9938	0.9945	0.9780	0.9993	218

D.7 NN trained on data set 1 testing data set 4 - detection

Average	Median	Minimum	Maximum
0.9781	0.9791	0.9435	0.9967

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9320	0.9503	0.7710	1.0000	121
4	0.9434	0.9943	0.7488	1.0000	99
5	0.9581	0.9943	0.8045	0.9998	78
6	0.9755	0.9953	0.8523	0.9995	61
7	0.9735	0.9943	0.8465	1.0000	77
8	0.9778	0.9948	0.8178	0.9988	57
9	0.9852	0.9920	0.9035	0.9990	46
10	0.9902	0.9935	0.9328	0.9990	74
11	0.9892	0.9929	0.9358	0.9998	68
12	0.9905	0.9935	0.8588	0.9980	71
13	0.9867	0.9930	0.8965	0.9998	63
14	0.9906	0.9933	0.9453	0.9983	65
15	0.9866	0.9933	0.8808	0.9980	61
16	0.9886	0.9917	0.9003	0.9988	68
17	0.9896	0.9923	0.9468	0.9978	94
18	0.9904	0.9923	0.9425	0.9990	119
19	0.9897	0.9910	0.9545	0.9990	218

D.8 NN trained on data set 3 testing data set 1 - detection

Average	Median	Minimum	Maximum
0.9715	0.9765	0.9057	0.9969

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9344	0.9518	0.6498	0.9998	121
4	0.9386	0.9883	0.5000	0.9995	99
5	0.9593	0.9889	0.7288	1.0000	78
6	0.9732	0.9938	0.8523	0.9993	61
7	0.9705	0.9890	0.7473	0.9998	77
8	0.9649	0.9925	0.6520	0.9990	57
9	0.9783	0.9901	0.8355	0.9993	46
10	0.9837	0.9926	0.8435	0.9990	74
11	0.9779	0.9914	0.6278	0.9993	68
12	0.9822	0.9923	0.8253	0.9988	71
13	0.9787	0.9885	0.8493	0.9998	63
14	0.9822	0.9885	0.8895	0.9985	65
15	0.9807	0.9898	0.8733	0.9995	61
16	0.9825	0.9892	0.8843	0.9983	68
17	0.9802	0.9891	0.8928	0.9990	94
18	0.9772	0.9855	0.7208	0.9988	119
19	0.9792	0.9853	0.7737	0.9995	218

D.9 NN trained on data set 1 testing data set 3 - detection

Average	Median	Minimum	Maximum
0.9696	0.9699	0.9326	0.9957

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9120	0.9338	0.7683	1.0000	121
4	0.9268	0.9930	0.7565	1.0000	99
5	0.9428	0.9918	0.7688	0.9995	78
6	0.9639	0.9940	0.8085	0.9998	61
7	0.9618	0.9938	0.8078	0.9998	77
8	0.9677	0.9938	0.7350	0.9995	57
9	0.9746	0.9909	0.8535	0.9988	46
10	0.9815	0.9931	0.6805	0.9995	74
11	0.9804	0.9910	0.8053	0.9993	68
12	0.9862	0.9930	0.8190	0.9993	71
13	0.9771	0.9925	0.7623	0.9988	63
14	0.9867	0.9920	0.9140	0.9988	65
15	0.9804	0.9918	0.8433	0.9975	61
16	0.9843	0.9918	0.8713	0.9983	68
17	0.9847	0.9919	0.7500	0.9993	94
18	0.9870	0.9915	0.8820	0.9983	119
19	0.9868	0.9898	0.8743	0.9990	218

D.10 NN trained on data set 3 testing data set 4 - detection

Average	Median	Minimum	Maximum
0.9658	0.9689	0.9058	0.9966

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9079	0.9343	0.6660	1.0000	121
4	0.9225	0.9883	0.5815	0.9998	99
5	0.9443	0.9923	0.7015	0.9998	78
6	0.9645	0.9945	0.7873	0.9995	61
7	0.9595	0.9890	0.7448	1.0000	77
8	0.9614	0.9930	0.6975	0.9995	57
9	0.9744	0.9927	0.8408	0.9985	46
10	0.9805	0.9937	0.8490	0.9990	74
11	0.9753	0.9919	0.5918	0.9993	68
12	0.9824	0.9920	0.8248	0.9990	71
13	0.9763	0.9913	0.8420	0.9998	63
14	0.9818	0.9910	0.8750	0.9995	65
15	0.9770	0.9888	0.8470	0.9993	61
16	0.9792	0.9902	0.8423	0.9993	68
17	0.9799	0.9893	0.8845	0.9973	94
18	0.9786	0.9858	0.7970	0.9990	119
19	0.9796	0.9859	0.8188	0.9990	218

D.11 NN trained on data set 4 testing data set 3 - detection

Average	Median	Minimum	Maximum
0.9741	0.9759	0.9375	0.9965

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9151	0.9408	0.7625	1.0000	121
4	0.9333	0.9930	0.7613	0.9998	99
5	0.9457	0.9958	0.7795	0.9998	78
6	0.9684	0.9958	0.8050	1.0000	61
7	0.9665	0.9948	0.8033	0.9995	77
8	0.9713	0.9945	0.7823	0.9995	57
9	0.9788	0.9946	0.8498	0.9998	46
10	0.9901	0.9952	0.9275	0.9995	74
11	0.9887	0.9944	0.9373	0.9988	68
12	0.9914	0.9958	0.8343	0.9990	71
13	0.9848	0.9943	0.8395	0.9990	63
14	0.9911	0.9950	0.9430	0.9993	65
15	0.9848	0.9925	0.8553	0.9990	61
16	0.9884	0.9935	0.8828	0.9985	68
17	0.9881	0.9930	0.8733	0.9990	94
18	0.9903	0.9930	0.9508	0.9990	119
19	0.9895	0.9925	0.9223	0.9990	218

D.12 NN trained on data set 1 testing data set 1 - estimation

Average	Median	Minimum	Maximum
0.9721	0.9728	0.9616	0.9766

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9571	0.9714	0.8967	0.9811	121
4	0.9602	0.9744	0.8842	0.9792	99
5	0.9658	0.9751	0.9087	0.9799	78
6	0.9708	0.9752	0.9286	0.9788	61
7	0.9721	0.9754	0.9275	0.9799	77
8	0.9727	0.9759	0.9160	0.9795	57
9	0.9752	0.9759	0.9544	0.9779	46
10	0.9760	0.9760	0.9671	0.9793	74
11	0.9761	0.9762	0.9726	0.9793	68
12	0.9753	0.9760	0.9263	0.9789	71
13	0.9754	0.9760	0.9489	0.9785	63
14	0.9760	0.9759	0.9728	0.9790	65
15	0.9755	0.9760	0.9483	0.9790	61
16	0.9759	0.9760	0.9719	0.9781	68
17	0.9760	0.9761	0.9723	0.9802	94
18	0.9759	0.9758	0.9730	0.9800	119
19	0.9758	0.9760	0.9707	0.9796	218

D.13 NN trained on data set 2 testing data set 2 - estimation

Average	Median	Minimum	Maximum
0.9714	0.9724	0.9591	0.9766

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9530	0.9685	0.8742	0.9793	121
4	0.9569	0.9744	0.8555	0.9814	99
5	0.9637	0.9751	0.8929	0.9792	78
6	0.9703	0.9756	0.9216	0.9800	61
7	0.9719	0.9763	0.9197	0.9795	77
8	0.9721	0.9762	0.8982	0.9797	57
9	0.9756	0.9766	0.9497	0.9806	46
10	0.9761	0.9759	0.9703	0.9804	74
11	0.9760	0.9764	0.9708	0.9786	68
12	0.9753	0.9761	0.9196	0.9790	71
13	0.9755	0.9762	0.9454	0.9784	63
14	0.9760	0.9764	0.9729	0.9787	65
15	0.9755	0.9762	0.9433	0.9783	61
16	0.9759	0.9761	0.9686	0.9800	68
17	0.9760	0.9762	0.9704	0.9802	94
18	0.9757	0.9759	0.9610	0.9792	119
19	0.9758	0.9757	0.9727	0.9805	218

D.14 NN trained on data set 3 testing data set 3 - estimation

Average	Median	Minimum	Maximum
0.9661	0.9676	0.9412	0.9776

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9275	0.9612	0.8123	0.9818	121
4	0.9375	0.9750	0.8020	0.9822	99
5	0.9478	0.9752	0.8247	0.9838	78
6	0.9613	0.9760	0.8526	0.9818	61
7	0.9632	0.9759	0.8501	0.9823	77
8	0.9647	0.9760	0.8334	0.9832	57
9	0.9726	0.9765	0.8934	0.9830	46
10	0.9752	0.9764	0.9323	0.9813	74
11	0.9769	0.9771	0.9505	0.9823	68
12	0.9750	0.9767	0.8502	0.9812	71
13	0.9746	0.9770	0.8953	0.9828	63
14	0.9770	0.9771	0.9633	0.9823	65
15	0.9731	0.9764	0.8832	0.9818	61
16	0.9766	0.9771	0.9336	0.9822	68
17	0.9764	0.9767	0.9535	0.9820	94
18	0.9770	0.9770	0.9710	0.9832	119
19	0.9757	0.9772	0.6467	0.9828	218

D.15 NN trained on data set 4 testing data set 4 - estimation

Average	Median	Minimum	Maximum
0.9142	0.9145	0.8498	0.9627

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9118	0.9027	0.8262	0.9850	121
4	0.9136	0.9178	0.8083	0.9855	99
5	0.9233	0.9361	0.8081	0.9856	78
6	0.9216	0.9404	0.7913	0.9849	61
7	0.9139	0.9361	0.5761	0.9850	77
8	0.9111	0.9441	0.7849	0.9844	57
9	0.9005	0.9252	0.6070	0.9854	46
10	0.9237	0.9495	0.7702	0.9850	74
11	0.9106	0.9441	0.7614	0.9857	68
12	0.9131	0.9427	0.7448	0.9854	71
13	0.9242	0.9567	0.7629	0.9844	63
14	0.9051	0.9426	0.7580	0.9843	65
15	0.9102	0.9432	0.7518	0.9837	61
16	0.9078	0.9511	0.7249	0.9852	68
17	0.9221	0.9537	0.7510	0.9858	94
18	0.9126	0.9531	0.7448	0.9851	119
19	0.9079	0.9542	0.6782	0.9854	218

D.16 NN trained on data set 5 testing data set 5 - estimation

Average	Median	Minimum	Maximum
0.9191	0.9202	0.8706	0.9629

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9084	0.9048	0.8154	0.9863	121
4	0.9084	0.9233	0.7610	0.9861	99
5	0.9220	0.9361	0.8264	0.9864	78
6	0.9275	0.9504	0.7936	0.9860	61
7	0.9313	0.9505	0.7856	0.9864	77
8	0.9224	0.9501	0.7769	0.9856	57
9	0.9393	0.9612	0.7746	0.9863	46
10	0.9274	0.9543	0.7733	0.9858	74
11	0.9208	0.9423	0.7670	0.9857	68
12	0.9103	0.9521	0.5367	0.9867	71
13	0.9211	0.9502	0.7591	0.9852	63
14	0.9235	0.9603	0.7609	0.9864	65
15	0.9033	0.9401	0.5710	0.9857	61
16	0.9187	0.9504	0.7563	0.9859	68
17	0.9206	0.9536	0.7582	0.9871	94
18	0.9141	0.9596	0.7528	0.9864	119
19	0.9228	0.9588	0.7512	0.9860	218

D.17 NN trained on data set 4 testing data set 1 - estimation

Average	Median	Minimum	Maximum
0.9251	0.9251	0.8644	0.9720

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9293	0.9248	0.8566	0.9946	121
4	0.9284	0.9333	0.8029	0.9954	99
5	0.9376	0.9430	0.7977	0.9956	78
6	0.9323	0.9473	0.8215	0.9947	61
7	0.9238	0.9462	0.5946	0.9948	77
8	0.9251	0.9456	0.8116	0.9947	57
9	0.9122	0.9223	0.6264	0.9942	46
10	0.9319	0.9585	0.7885	0.9945	74
11	0.9192	0.9514	0.7905	0.9947	68
12	0.9217	0.9452	0.7892	0.9944	71
13	0.9351	0.9638	0.7726	0.9943	63
14	0.9146	0.9525	0.7831	0.9946	65
15	0.9202	0.9489	0.7748	0.9932	61
16	0.9173	0.9550	0.7309	0.9947	68
17	0.9317	0.9640	0.7820	0.9952	94
18	0.9220	0.9609	0.7764	0.9957	119
19	0.9172	0.9594	0.7002	0.9952	218

D.18 NN trained on data set 1 testing data set 4 - estimation

Average	Median	Minimum	Maximum
0.9526	0.9537	0.9344	0.9610

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9278	0.9475	0.8279	0.9690	121
4	0.9334	0.9584	0.8146	0.9656	99
5	0.9417	0.9587	0.8514	0.9663	78
6	0.9510	0.9594	0.8855	0.9638	61
7	0.9522	0.9588	0.8820	0.9679	77
8	0.9526	0.9580	0.8552	0.9664	57
9	0.9567	0.9587	0.9189	0.9646	46
10	0.9588	0.9594	0.9458	0.9654	74
11	0.9593	0.9600	0.9507	0.9648	68
12	0.9586	0.9602	0.8821	0.9647	71
13	0.9574	0.9585	0.9108	0.9659	63
14	0.9589	0.9599	0.9494	0.9643	65
15	0.9577	0.9593	0.9073	0.9658	61
16	0.9589	0.9591	0.9438	0.9641	68
17	0.9588	0.9592	0.9476	0.9662	94
18	0.9588	0.9591	0.9513	0.9655	119
19	0.9588	0.9591	0.9512	0.9642	218

D.19 NN trained on data set 3 testing data set 1 - estimation

Average	Median	Minimum	Maximum
0.9581	0.9600	0.9278	0.9714

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9171	0.9542	0.7719	0.9771	121
4	0.9288	0.9670	0.7742	0.9778	99
5	0.9403	0.9702	0.7892	0.9848	78
6	0.9535	0.9697	0.8361	0.9766	61
7	0.9536	0.9694	0.8253	0.9779	77
8	0.9569	0.9705	0.7970	0.9772	57
9	0.9665	0.9698	0.9130	0.9819	46
10	0.9680	0.9705	0.9341	0.9760	74
11	0.9697	0.9706	0.9507	0.9785	68
12	0.9693	0.9712	0.8668	0.9766	71
13	0.9677	0.9705	0.9055	0.9775	63
14	0.9690	0.9710	0.9328	0.9774	65
15	0.9675	0.9707	0.9147	0.9765	61
16	0.9695	0.9712	0.9434	0.9820	68
17	0.9683	0.9703	0.9267	0.9762	94
18	0.9688	0.9698	0.9419	0.9771	119
19	0.9669	0.9700	0.5363	0.9770	218

D.20 NN trained on data set 1 testing data set 3 - estimation

Average	Median	Minimum	Maximum
0.9444	0.9459	0.9136	0.9658

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9012	0.9447	0.7283	0.9819	121
4	0.9150	0.9585	0.7248	0.9828	99
5	0.9251	0.9546	0.7612	0.9798	78
6	0.9464	0.9648	0.8010	0.9820	61
7	0.9421	0.9608	0.7747	0.9855	77
8	0.9440	0.9603	0.7566	0.9783	57
9	0.9498	0.9607	0.8556	0.9779	46
10	0.9576	0.9626	0.8981	0.9806	74
11	0.9581	0.9634	0.9150	0.9764	68
12	0.9584	0.9613	0.8463	0.9772	71
13	0.9529	0.9571	0.8677	0.9792	63
14	0.9569	0.9607	0.9130	0.9759	65
15	0.9508	0.9560	0.8491	0.9815	61
16	0.9528	0.9544	0.9069	0.9738	68
17	0.9531	0.9558	0.8956	0.9767	94
18	0.9547	0.9571	0.9085	0.9798	119
19	0.9524	0.9542	0.8988	0.9766	218

D.21 NN trained on data set 3 testing data set 4 - estimation

Average	Median	Minimum	Maximum
0.9420	0.9449	0.9084	0.9581

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.8965	0.9291	0.7474	0.9629	121
4	0.9093	0.9523	0.7548	0.9638	99
5	0.9216	0.9566	0.7713	0.9680	78
6	0.9354	0.9543	0.8034	0.9616	61
7	0.9369	0.9561	0.8005	0.9617	77
8	0.9406	0.9568	0.7813	0.9623	57
9	0.9503	0.9557	0.8718	0.9625	46
10	0.9533	0.9573	0.8872	0.9625	74
11	0.9536	0.9557	0.9045	0.9616	68
12	0.9550	0.9578	0.8359	0.9632	71
13	0.9518	0.9563	0.8552	0.9633	63
14	0.9547	0.9563	0.9223	0.9617	65
15	0.9517	0.9560	0.8619	0.9628	61
16	0.9551	0.9570	0.9032	0.9621	68
17	0.9536	0.9558	0.8891	0.9623	94
18	0.9540	0.9553	0.9263	0.9628	119
19	0.9522	0.9555	0.5396	0.9631	218

D.22 NN trained on data set 4 testing data set 3 - estimation

Average	Median	Minimum	Maximum
0.9250	0.9263	0.8679	0.9773

Count of inputs	Average	Median	Minimum	Maximum	Instances count
3	0.9022	0.9082	0.7807	0.9962	121
4	0.9118	0.9209	0.7711	0.9957	99
5	0.9247	0.9480	0.7868	0.9963	78
6	0.9260	0.9485	0.8163	0.9958	61
7	0.9198	0.9414	0.4620	0.9955	77
8	0.9227	0.9511	0.8115	0.9954	57
9	0.9111	0.9336	0.5864	0.9946	46
10	0.9361	0.9599	0.8060	0.9946	74
11	0.9267	0.9616	0.7950	0.9955	68
12	0.9285	0.9562	0.7632	0.9953	71
13	0.9395	0.9658	0.7860	0.9947	63
14	0.9222	0.9569	0.7907	0.9941	65
15	0.9261	0.9572	0.7826	0.9942	61
16	0.9241	0.9618	0.6584	0.9950	68
17	0.9377	0.9600	0.7888	0.9956	94
18	0.9306	0.9651	0.7726	0.9955	119
19	0.9256	0.9611	0.7307	0.9949	218