University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization:

Constructions and Materials/ Energy

Spring semester, 2010

Open access

Writer:
Mats Leander Mathisen

…………………………………………
(Writer's signature)

Faculty supervisor:
Professor Mohsen Assadi

Title of thesis:

Noise filtering from a nonlinear system by using AANN

Credits (ECTS): 30 points

Key words:

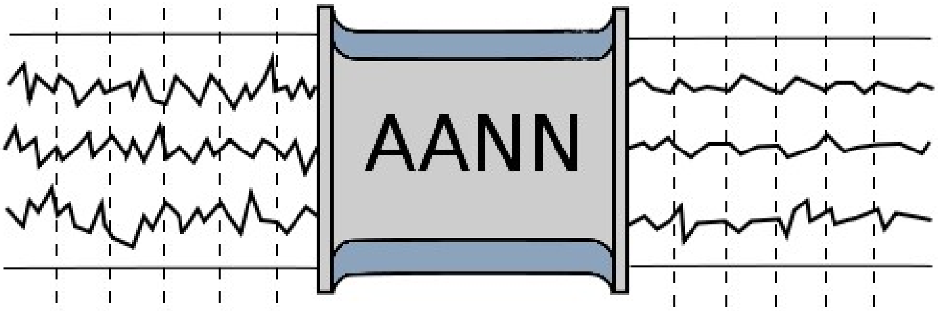
AANN
ANN
Artificial neural networks
Autoassociative neural networks
Gas turbines
Noise filter

Pages: 44

Stavanger, June 9th 2010

Frontpage for master thesis
Faculty of Science and Technology
Decision made by the Dean October 30th 2009

Noise filtering from a nonlinear system by using AANN



Mats Leander Mathisen

University of Stavanger

Abstract

In order to run a gas turbine, the operator (be it human or automatic) needs to monitor the conditions of the various parts inside it. Pressures, temperatures, mass flows, vibrations, power output. These properties all need to be controlled in order to run the gas turbine optimally. And in order for the operator to make the necessary adjustments, sensors inside the gas turbine are needed to monitor said properties.

As the industry drives towards higher efficiencies and lower emissions, the accuracy of these sensor readings inside the gas turbine become more and more important. The objective of this thesis then, was to see how this accuracy could be improved by the use of autoassociative neural networks (AANN), which is a kind of noise filter.

Sensor readings will not be completely accurate, since the technology is not perfect. One problem is something called random noise, meaning sensor measurements that are scattered randomly close to the exact value. A noise filter will take these scattered measurements and move them all closer to the exact value.

It is already known that an AANN can perform this task, and in this thesis the main objective was to find some indication of just how effective it is as a noise filter.

In order to measure how effective a noise filter is, one would ideally need one set of measurements, which are noisy, and one set of corresponding measurements, which are not noisy at all (perfect measurements). Checking the level of noise reduction then would be to first filter the noisy measurements, and then comparing both the filtered and noisy measurements to the perfect measurements.

Such a solution can not be found with real measurements from a gas turbine, because they are never perfect. But if the measurements were calculated using thermodynamic and physics equations, they would not contain noise. They would be completely theoretical, but they would not contain noise.

Synthetic measurements like these were generated by the use of a software which can model gas turbines and calculate theoretical properties for various theoretical scenarios. Noise was then added to these noise free measurements in order to emulate the real gas turbine. And with that, two sets of measurements were available: One set of noisy measurements, and one set of perfect measurements.

With the use of the MATLAB neural network toolbox, these sets of measurements were used to test the effectiveness of an AANN as a noise filter. The noisy measurements were filtered through an AANN, and the filtered and noisy measurements were then compared to the perfect measurements.

Artificial neural networks (ANN), which also have some noise filtering abilities, were also tested this way. But not as extensively as the AANN.

Results showed that there was indeed noise reduction, but not for all the individual parameters in the measurements. For some parameters, the AANN achieved very good noise filtering, but for other parameters there was no effect.

The reason for this is not entirely clear, but earlier two purely mathematical examples were tested in order for the author to familiarize himself with the methodology. And these examples only had two-three parameters; few enough to visualize in graphs (2- and 3 dimensional).

In these two examples, there was found a trend which suggested that an AANN does not filter each parameter individually, but rather all parameters together as if they were one.

The author can not prove this, but he speculates the same principle could apply to measurements that have more than three parameters as well, which means that an AANN might not be very ideal for noise filtering of individual sensors inside a gas turbine.

It the future, it could certainly be interesting to test an AANN on measurements from a real gas turbine. Several conditions would need to be met for such a test to prove useful; like extensive correlations between the different parameters included, and redundant measurements. But it is not unreasonable to assume there would be some reduction of noise.

Preface

This report represents the final product of the author's master thesis, which concludes a two year Masters degree in structural and material science at the University of Stavanger, Norway. The latter part of this Masters degree was centered around energy applications, and one of the classes was an introductory course to gas turbine technology.

With this background, the need for the author to spend time studying gas turbine theory was greatly reduced, and most of the time has been spent on learning to understand, create and utilize artificial neural networks. This has also provided the author with an opportunity to familiarize himself somewhat with the computer program MATLAB.

It has been a nice learning experience with regards to independent work, and the subject matter was indeed quite fascinating.

Technical aspects of this report

The report is divided into four chapters: Theory, Work, Results and Discussion. Each chapter builds on previous chapters, but some relevant information may be included in the appendix at the end of the report. A direct reference to the appendix will be made in the text where appropriate.

A list of notations, abbreviations and a small glossary has been added for easy reference while reading the report.

References to other works are added inside or at the end of paragraphs, indicating that these particular lines or paragraphs consist of information interpreted directly from the references given. In two instances, the references have been added directly to the headline, meaning those sources heavily influence those subchapters.

Whenever MATLAB code is included, it will be in a different font, setting it apart from the rest of the text.

Vectors and matrices are written in bold script, such as for the input matrix; **p_input**.

Acknowledgements

The author would like to thank Phd. students Nikolett Sipöcz and Thomas Palme, as well as Professor Mohsen Assadi at the University of Stavanger for providing the opportunity to work on this thesis.

Also a special thanks to Thomas Palme for all guidance and help. Especially with the theory of artificial neural networks.

And finally, thanks to fellow student Linn Noomi Garborg for the dataset generated in IPSEpro with a model of the Turbec T100 CHP, without which this thesis would not have been possible.

Contents

Preface 1
Contents 2
Nomenclature 3
Introduction 4

1 Theory
1.1 Gas turbine 5
1.1.1 Compressor 6
1.1.2 Combustion 8
1.1.3 Turbine 9
1.1.4 Auxiliary systems 9
1.2 Turbec T100 CHP 10
1.2.1 Modified Trubec T100 CHP 10
1.2.2 Sensors monitoring 10
1.3 Artificial Neural Networks 11
1.3.1 The perceptron 12
1.3.2 Transfer functions 12
1.3.3 Multilayer perceptron 13
1.3.4 Error backpropagation 13
1.3.5 Training algorithms 14
1.3.6 Regression 15
1.3.7 Preprocessing 15
1.3.8 Principal component analysis 16
1.3.9 Autoassociative Neural Networks 16
1.4 Noise reduction 17
1.4.1 Exponential smoothing 17
1.4.2 AANN noise reduction 17
1.4.3 ANN noise reduction 18
1.5 IPSEpro 18
1.5.1 Creating a gas turbine model 18
1.5.2 Shortcomings of the IPSEpro model 18

2 Work
2.1 Theoretical 2D AANN example 19
2.1.1 Acquisition and preprocessing of data 19
2.1.2 Building the AANN 19
2.1.3 Training the network 20
2.1.4 Producing the results 20
2.2 Theoretical 3D AANN example 22
2.3 AANN with data from IPSEpro 23
2.3.1 Clean data 23
2.3.2 Data with noise 24
2.3.3 Data with noise and outliers 24
2.4 AANN with data from Turbec 25
2.4.1 Preprocessing of data 25
2.4.2 Building the AANN model 25
2.4.3 Training and producing results 25
2.5 ANN with data from IPSEpro 26
2.5.1 Preprocessing of data 26
2.5.2 Building the ANN model 26
2.5.3 Training and producing results 26
2.6 ANN with data from Turbec 26

3 Results
3.1 Theoretical AANN examples 27
3.1.1 2D example 27
3.1.2 3D example 27
3.2 AANN with data from IPSEpro 29
3.2.1 Optimal network architecture 29
3.2.2 Level of noise reduction 30
3.2.3 Noise reduction on noisy data with outliers 30
3.3 AANN with data from Turbec 30
3.4 ANN with data from IPSEpro 31
3.4.1 Results for all parameters 31
3.4.2 Level of noise reduction 32
3.5 ANN with data from Turbec 32

4 Discussion
4.1 Shortcomings of methodology used 33
4.2 Evaluation of results 33
4.3 Ideas for future work 33

List of tables and illustrations 34
References 35

Appendix
I Backpropagation example 36
II Various MATLAB code used 38
III The MATLAB nntool 39
IV Additional illustrations 41

Nomenclature

Here are included symbols, abbreviations and selected words/phrases used in this report that might be unfamiliar to the reader. It will hopefully save extra time spent looking them up elsewhere.

Notation

T	- temperature
S	- entropy
p_0	- stagnation pressure
p	- static pressure
C	- velocity
c_p	- specific heat
ρ	- density
R	- universal gas constant

Abbreviations

AANN	- autoassociative neural network
ANN	- artificial neural network
CHP	- combined heat and power
CO	- carbon monoxide
GUI	- graphical user interface
MLP	- multilayer perceptron
MSE	- mean square error
NO_X	- nitric oxides
PMC	- power module controller
TIT	- turbine inlet temperature
TOT	- turbine outlet temperature
UHC	- unburned hydrocarbons

Glossary

correlated - dependent on each other to some extent.

diffusion flame -

dynamic pressure - pressure generated by kinetic energy.

isentropic - constant entropy.

microturbine - compact turbine generating between 100 and 200 kW of electrical energy.

normalize - to convert a set of data, making the values fall between a set of selected boundaries (for example between 1 and -1).

recuperator - heat exchanger which uses hot exhaust to preheat compressed air before it enters the combustion chamber inside a gas turbine.

regression - an approximately best fit relation between several parameters.

stagnation pressure - static- plus dynamic pressure.

static pressure - pressure in the form of potential energy.

stoichiometry - The art of calculating atomic proportions in chemical reactions.

white noise - random noise following a gaussian distribution.

Introduction

Gas turbines will usually have several sensors monitoring their components. These can be temperature sensors, pressure sensors, mass flow sensors, etc. While a gas turbine is running, these sensors are vital to ensure the operator (human or automatic) runs the gas turbine within its specifications. The drive to higher efficiency is also raising the pressures and temperatures in gas turbines, which further increases the challenge to accurate sensor measurements.

For this thesis, the gas turbine in question is a Turbec T100 CHP. It is a microturbine; a small gas turbine which can produce (in this case) 100 kW of electricity and more than 150 kW of heat through an exhaust to water heat exchanger.

Microturbines such as the Turbec T100 CHP are designed to work on autopilot while directly connected to the commercial power grid. A process that requires accurate sensor measurements for the controls to run the system optimally. Such accurate measurements in the hot components of gas turbines (including combustion diagnostics) are recognized as a major need for the assessment of engine component health and performance. And considering the relatively small scale of a microturbine, these measurements will also need to be economical.

There already exist simple linear filters in the form of mathematical algorithms, which provide excellent noise reduction for industrial applications, but they do have certain limitations.[11] It would therefore be interesting to investigate other options.

Using measurements from the Turbec T100 CHP microturbine, it should be possible to train an artificial neural network (ANN), or more specifically an autoassociative neural network (AANN), and apply it to the measurement data. An AANN is a filter which can be used to detect sensor failure, trend shifts or degradation in gas turbines. At the same time it can also provide some degree of noise reduction in the sensor readings.[10,11]

But how much noise reduction? The author has received simulated sensor data of a Turbec T100 CHP gas turbine. These data are *clean*, in the sense that they do not contain any measurement noise. By adding white noise to the data, one obtains a set of noisy data, which will represent the *real* sensor data one could expect from the Turbec T100 CHP.

These *clean* and *real* data provide the author with an opportunity to estimate how much noise reduction an AANN can provide to gas turbine sensors. By simply checking the difference between the *clean* and *filtered* (filtered through the AANN) data in relation to the difference between the *clean* and *real* data, one should be able to get an indication of how much noise reduction the AANN can provide.

Since this simulated sensor data of the Turbec T100 CHP is not a perfect representation of the real engine, it only serves as an indication of plausible success. Sensor data from the real Turbec T100 CHP gas turbine will therefore also be filtered through an AANN to hopefully see if the process actually works.

Whether the filter works on real measurements can be checked by simply plotting the original sensor data on top of the sensor data which has been filtered through the AANN. However, as is the case with all artificial neural networks; good measurement data is essential in order to get good results.

1 Theory

In this chapter there will be provided a short introduction to the basic theory of gas turbines, before exploring in more detail the design of the Turbec T100 CHP. Hopefully, this will provide readers who are not familiar with gas turbines, or perhaps are in need of a short reminder, with enough insight to follow the rest of the report. A thorough understanding of the Turbec might also be important for constructing and analyzing the AANN model later.

An introduction to artificial neural networks then follows, presenting the theory on which the computer program used in this assignment was built. Details of the mathematics involved will not be covered (except for a brief example in Appendix I), but instead an attempt is made to give a general understanding of what an artificial neural network is and can do.

How neural networks may be used for noise filtering is then introduced, as well as compared to a more conventional approach.

And finally, a brief mention of IPSEpro, and how it was used for the simulated run of the Turbec T100 CHP.

1.1 Gas turbines

A simple gas turbine design consists of three main components; compressor, combustion chamber and turbine. Together, these provide a power output in the form of a rotating shaft, which can be utilized directly as mechanical energy or connected to an electric generator.

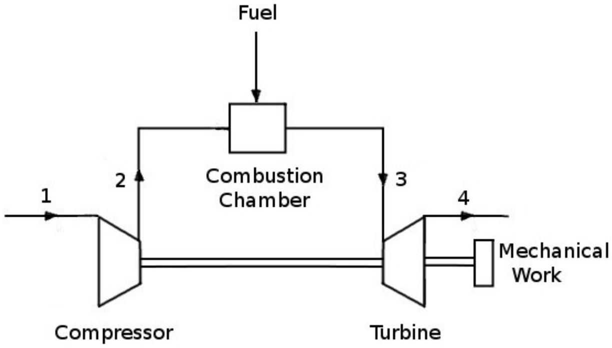


Illustration 1: Open simple cycle gas turbine.

In order for a turbine to produce work, a pressure ratio is needed in the working fluid (usually air). Consider the equation below (assume Z to be negligible) together with illustration 2. The density of air increases when the pressure increases (temperature will also increase). This is done in the compressor. After compression the working fluid is then heated up further in the combustion chamber at ideally constant pressure (there is however always some pressure loss due to friction). This added heat decreases the density in the air towards what it was before compression. When the air then expands through the turbine, more pressure is converted into mechanical energy than if there was no extra heat from combustion. This way the turbine can generate more mechanical power than what is needed to run the compressor.

$$\rho = \frac{p}{ZRT}$$

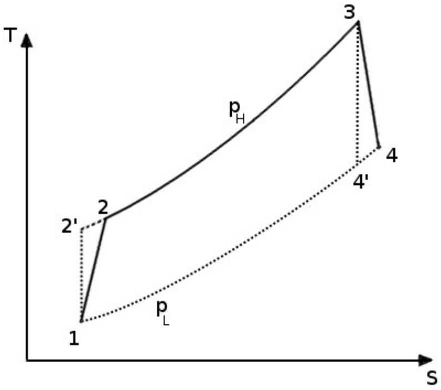


Illustration 2: T-S diagram for open simple cycle gas turbine. 2' and 4' indicate the isentropic states of the working fluid. P_L and P_H are constant pressures.

Compressor pressure ratio, turbine inlet temperature and component efficiencies are the key factors in the simple cycle gas turbine. As can be seen in illustration 3 on the next page, increasing the turbine inlet temperature provides the potential for more work (the area under the graph is bigger), while the pressure ratio determines the relationship between the amount of heat added and the amount of work produced. It can also be seen that a high efficiency in the compressor causes it to draw less power, and a high efficiency in the turbine causes it to produce more work (less entropy generated means less energy lost)..

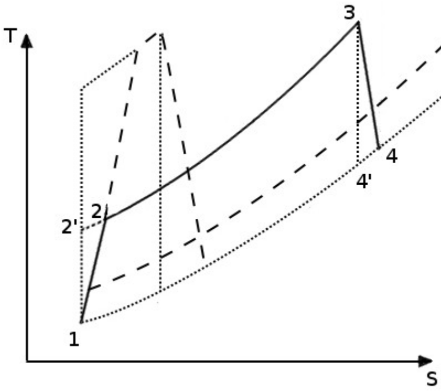


Illustration 3: T-S diagram for open simple cycle gas turbine. 2' and 4' indicate the isentropic states of the working fluid.

Restrictions to maximum temperature are a result of material properties, while restrictions to the pressure ratio depends on the size of the compressor and turbine.

Because the compression, combustion and expansion occurs in three separate components (as opposed to a reciprocating engine), each component can be developed individually, and later linked up in several different ways. It might be desirable to use more than one compressor in order to achieve a higher total pressure ratio. Perhaps it would be practical to have two turbines; one to drive the compressor, and one on a different shaft used to drive the generator. Different applications for gas turbines mean different design solutions may provide an optimal solution.

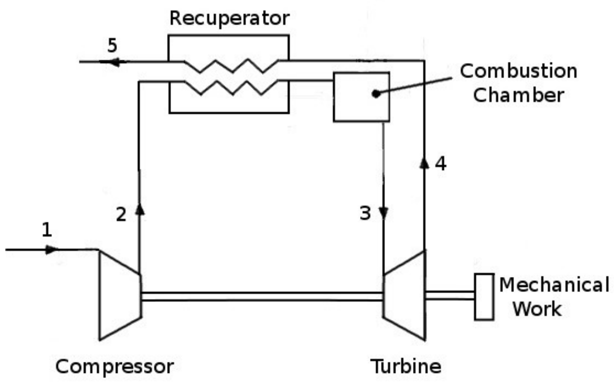


Illustration 4: Open simple cycle gas turbine with recuperator.

An overview of all the different designs in use will not be covered here, but a brief description of the simple cycle gas turbine with recuperator should be mentioned. In illustration 4, it can be seen how the exhaust leaving the turbine is sent through the recuperator, preheating the compressed air. This way less heat needs to be produced in the combustion chamber, offering an increased thermal efficiency.

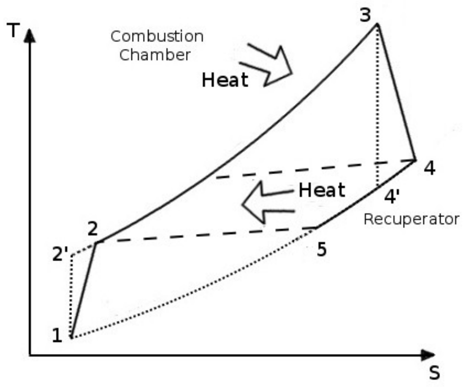


Illustration 5: T-S diagram for an open simple cycle gas turbine with recuperator. The working fluid is first heated up in the recuperator, and then the combustion chamber. 2' and 4' indicate the isentropic states of the working fluid.

The recuperator gas cycle needs a smaller pressure ratio than the simple cycle. If the pressure ratio is too high, the temperature difference between compressor outlet and turbine outlet will be too small for the recuperator to be useful. This can be seen in illustration 5, by imagining that the pressure ratio is so high that the temperature at point 2 is actually higher than the temperature at point 4.

In the T-S diagrams for both simple cycle and cycle with recuperator, notice how the temperature leaving the turbine is higher than the temperature going into the compressor (several hundred degrees in fact). Utilizing this excess heat will increase the thermal efficiency of the system considerably. This can be done by for example using a recuperator, combining the gas turbine with a steam turbine, or simply heating up water.

1.1.1 Compressor

There are two kinds of compressors being used in gas turbines; the centrifugal (radial) compressor, and the axial compressor. Centrifugal compressors are primarily used for small mass flows, while axial compressors are used for large mass flows.

Up to a certain mass flow, centrifugal compressors are more compact. They offer a better resistance to foreign object damage, less susceptibility to loss of performance by build-up of deposits on the blade surfaces, and the ability to operate over a wider range of mass flow at a particular rotational speed.[1] Axial compressors on the other hand offer the potential for higher pressure ratios, and higher mass flows with the same size air inlet area.

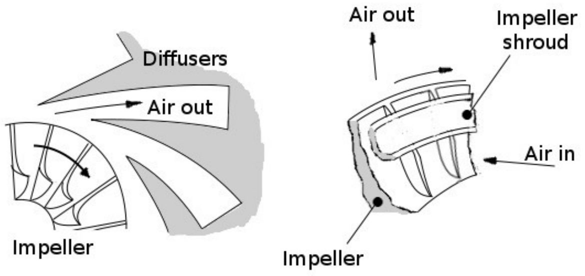


Illustration 6: A centrifugal compressor consists of an impeller, the impeller shroud (part of impeller casing), and diffusers.

The centrifugal compressor consists of a stationary casing and a rotating impeller. Air is drawn in through the impeller eye, and accelerated outward to the fixed diverging passages in the casing. These passages are known as diffusers, and here the air is decelerated with a consequent rise in static pressure.

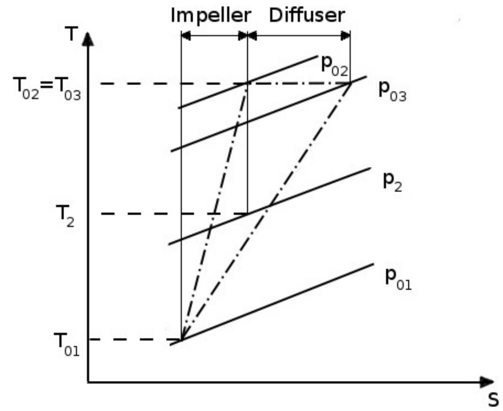


Illustration 7: T-S diagram for working fluid as it passes through a radial compressor. [1]

Taking p_0 to mean stagnation pressure, i.e. total pressure, the T-S diagram in illustration 7 shows how pressure rises through the centrifugal compressor. Stage 1 indicates the impeller eye, stage 2 the impeller tip, and stage 3 the diffuser. Over the impeller there is produced a rise in both static and dynamic pressure, while the diffuser converts some of the extra dynamic pressure to static pressure. Notice the losses in both impeller and diffuser expressed by an increase of entropy.

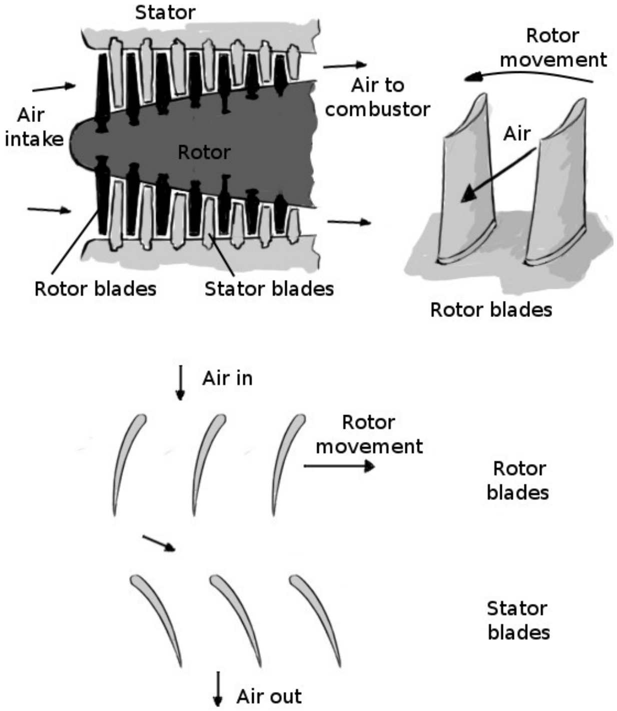


Illustration 8: An axial compressor. Rotor blades are fastened to the rotating shaft, while stator blades are fastened to the stator. One stage of an axial compressor consists of one row of rotor blades and one row of stator blades.

An axial compressor consists of several stages. In every stage, there is a row of rotor blades followed by a row of stator blades. Rotor blades accelerate the air with mechanical energy, while stator blades decelerate the air, converting dynamic pressure to static pressure. This is repeated over as many stages as is needed to produce the desired pressure ratio.

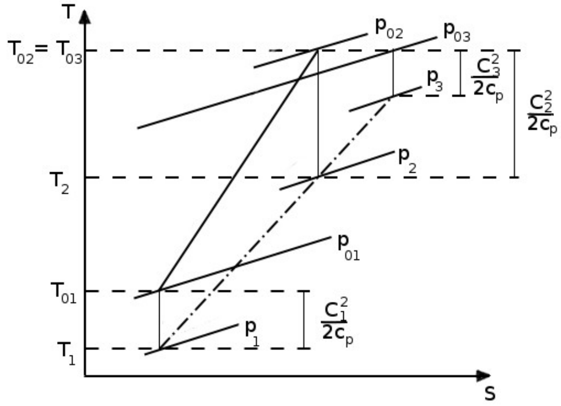


Illustration 9: T-S diagram of working fluid as it passes through an axial compressor. [1]

$$p_0 - p = \frac{C^2}{2c_p}$$

To clarify illustration 9; p_0 is the stagnation pressure (total pressure) and p is the static pressure. Subtracting the static pressure from the stagnation pressure leaves the dynamic pressure. C is the absolute velocity and c_p is the specific heat of the working fluid.

Looking at the T-S diagram for the axial compressor (illustration 9), position 1 lies before the rotor, position 2 between rotor and stator, and position 3 after the stator. This represents one stage in the axial compressor, and it can be seen how the dynamic and static pressure changes through each stage.

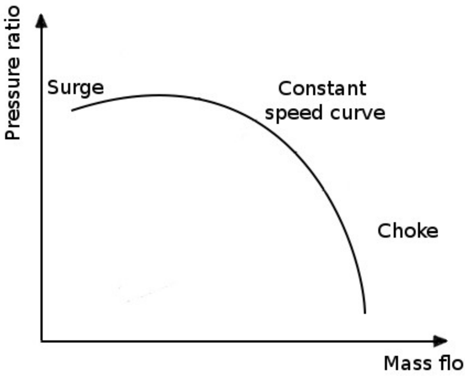


Illustration 10: Theoretical characteristics of a compressor. Working fluid moving at constant speed.

All compressors operate in a specified range between something called surge and choke. If the pressure just outside the compressor inlet is lower than the pressure just inside the compressor inlet, it will prevent air from fully being sucked into the compressor. This phenomenon, called surge, will cause unstable operation, and could be very damaging to the system.

If on the other hand the mass flow is too high, the compressor will not be able to influence the working fluid properly, as friction is stealing all the mechanical energy. This phenomenon is called choke.

1.1.2 Combustion

Combustion is achieved by mixing fuel (gas and/or liquid) with air, and then igniting it. Fuel is first premixed with some of the high pressure air and injected into the air stream at certain speeds. Old gas turbines did not premix the fuel and air, but simply injected the fuel directly into the air stream. This caused a diffusion flame which was very stable.[1] But it also produced emissions due to more incomplete combustion (more CO and UHC generated). This combined with the high flame temperatures also generated high NO_X emissions.

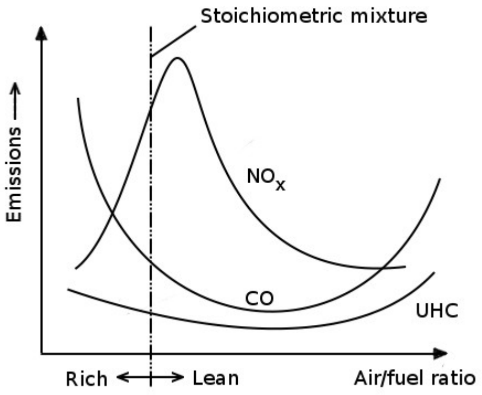


Illustration 11: Dependence of emissions on air/fuel ratio. [1]

$$CH_4 + Air \rightarrow H_2O + CO_2 + N_2$$

The stoichiometric chemical reaction for combustion with CH_4 and air, requires that the number of atoms on both sides of the reaction correspond to each other (assume air only consists of O_2 and N_2). Together with illustration 11, this gives an idea of what is happening in the combustion process.

A dry (low H_20 concentration in air), lean (more air than in the stoichiometric mixture) premix will lower the flame temperature because of the excess air which needs to be heated up. The lowered flame temperature lowers the NO_X emissions. At the same time, the extra O_2 from the excess air thoroughly mixed with the fuel, increases the chances for complete combustion, which reduces CO and UHC emissions.

With regards to combustor design, three options stand out; the annular, the can-type and the cannular combustor configurations. For aircraft engines the annular combustor is used almost exclusively because of its low frontal area and weight for a given volume.[2] Can-type and cannular combustors are heavier and require more space, but this is not necessarily a big problem for industrial applications. And since they consist of several similar cans, design only needs to focus on one can. This may prove economical both in development and maintenance.

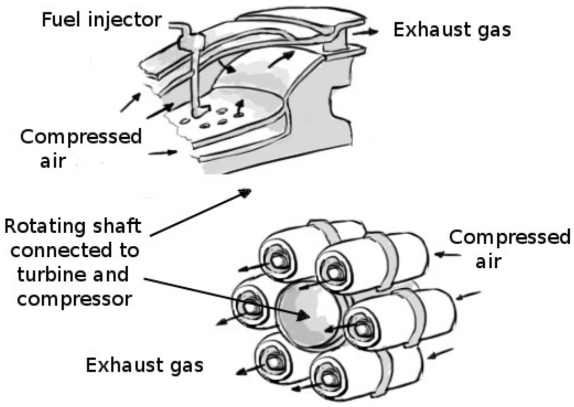


Illustration 12: Upper left: Annular combustor. The combustion chamber wraps all the way around the rotating shaft connected to turbine and compressor. Lower right: Can-type combustor. Each can is a combustion chamber. Can-type combustors need not be positioned around the rotating shaft. Cannular combustors are a hybrid of the two above.

High combustion efficiency, low pressure loss, flame stability and low emissions are the main guidelines for the combustion process. Turbine inlet temperatures also need to be as high as possible, but they are restricted both by material properties and emissions increasing with very high flame temperatures.

1.1.3 Turbine

As with compressors, there are two kinds of turbines; radial and axial turbines. Most gas turbines use the axial flow turbine.[1]

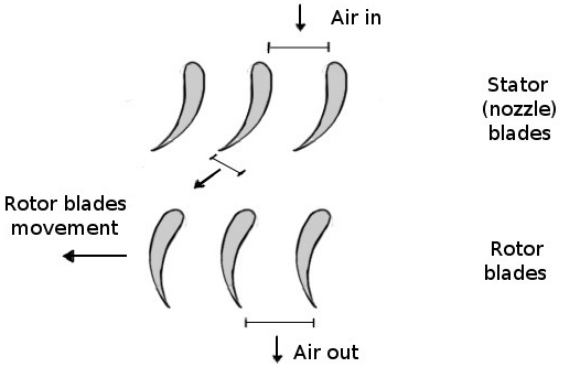


Illustration 13: One stage of an axial flow turbine consists of one row of stator (nozzle) blades, and one row of rotor blades. Axial turbines have fewer (some times only one) stages than axial compressors.

A turbine converts high pressure and temperature to mechanical energy. It is worth mentioning that because of the high temperatures, turbine blades are cooled by ventilation air passing over the turbine disc and blade roots. Small cooling holes inside the individual blades may also provide protection from the high temperatures, and they make turbine blades very expensive to manufacture.

The axial turbine looks much like the opposite of the axial compressor (compare illustrations 8 and 13). It consists of one or more stages. One stage has a row of nozzle blades, and a row of rotor blades. Exhaust from the combustion chamber is accelerated through the nozzle blades and the extra kinetic energy is then converted to mechanical energy in the rotor blades.

Radial turbines look very similar to a reversed compressor, but with nozzle guide vanes replacing the diffuser vanes. There might also be a diffuser at the outlet to reduce the exhaust velocity to a negligible value, which prevents kinetic energy from being wasted.[1]

1.1.4 Auxiliary systems

In addition to the three main components of a gas turbine, there must of course be several auxiliary components to complement the system.

For utilizing the mechanical energy directly, a gearbox might be needed. And for conversion of mechanical energy to electricity a generator is needed.

An air filter is often installed in front of the compressor in order to prevent foreign objects entering the cycle and causing damage to various components. It also helps prevent fouling (particles gathering inside the system, deteriorating components).

Rotating parts often require lubrication and cooling. Subsystems would need to be installed in order to provide this.

A system is needed to regulate the fuel supply, as well as offering a possibility to control the start up, re-ignition and shut down of the gas turbine.

Electronics and sensors need to be installed in order to create a user interface so the engineer can control the entire machine.

1.2 Turbec T100 CHP[3]

The Turbec T100 CHP is a 100 kW microturbine. It includes back to back radial compressor and turbine, combustion chamber, recuperator, electric generator, and exhaust gas to water heat exchanger.

Comparing it to the recuperator cycle mentioned earlier, the only difference is the exhaust gas to water heat exchanger, providing a higher thermal efficiency.

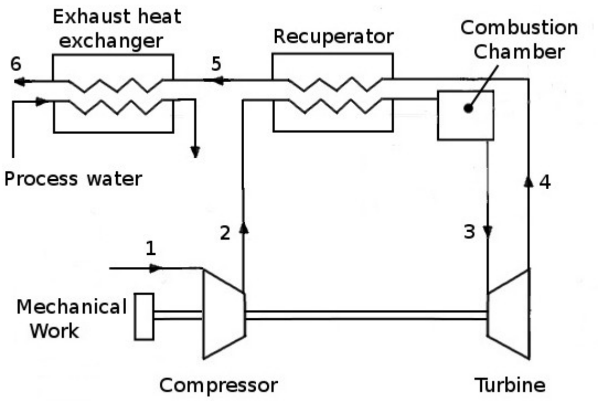


Illustration 14: The Turbec T100 CHP gas turbine. See illustration 5 for a T-S diagram (exhaust to water heat exchanger not included).

Auxiliary systems include air intake and ventilation system, electrical and control system, lubrication oil system, buffer air system, water cooling system for the electronics and generator, emergency stop and fuel gas system.

Ambient air is taken from an outdoor intake and as the airflow enters the CHP unit, it is split into two partial flows. One for combustion air, and one for ventilation of excess heat. There are two air filters, one optional coarse prefilter close to the outdoor intake, and a fine filter close to the the compressor.

Electric power is generated with a rotating permanent magnet, but needs to be rectified and transformed to the preferred frequency. The generator and electrical system is automatically controlled by the PMC (Power Module Controller). In reverse they work as the electric starter for the gas turbine.

Oil is circulated from the bearings to an oil-to-air cooler by a motor-driven pump, providing lubrication. Buffer air is pumped to the sealing system to block lubrication oil from entering the engine. An oil filter separates oil mist from the air.

The Turbec T100 CHP runs on natural gas, and the fuel gas system includes piping, auto shut off valve, filter, fuel block, pressure sensor, fuel control valves and pipes to injectors. If the gas provided has too low a pressure, there is a fuel gas compressor installed to raise the pressure.

Oil pressure and temperature, heat demand, gas pressure and vibrations are monitored by the PMC, running the gas turbine automatically. It starts, stops and supervises the operation, responding to a critical fault by either a normal stop or an emergency stop. Faults are then logged in the system. The author has not been able to learn any specifics on the algorithms used in the PMC.

1.2.1 Modified Turbec T100 CHP

For this thesis, a Turbec T100 CHP stationed at Risavika Gas Centre, Tananger Norway, is being modeled. This CHP unit is identical to the one described above, except for some modifications to the combustion system. A bypass has been installed, making it possible to connect and use a fuel cell to replace the combustion chamber.

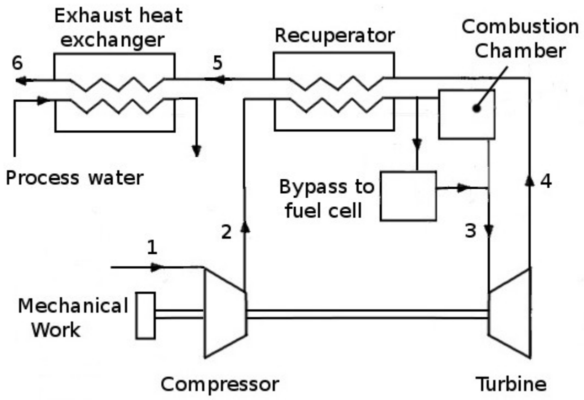


Illustration 15: Modified Turbec T100 CHP gas turbine. A bypass around the combustion chamber has been installed, providing an opportunity to run the gas turbine together with a fuel cell.

1.2.2 Sensors monitoring

The various parameters in a gas turbine can be monitored by several different kinds of sensor technologies. Temperatures, pressures, mass flows, electric power, rotational speed, mechanical work and more are measured continuously.

For temperature measurements, thermocouples are widely used. An electrical conductor (usually a metal) subjected to a thermal gradient will generate a voltage. And different metals subjected to the same thermal gradient will generate different voltages. If these two metals are connected, the temperature can be calculated by measuring the difference in voltages, using a voltmeter.

Pressure measurements can be done using piezoresistive sensors. These consist of a small metal surface subjected to ambient pressure. The variation in pressure causes mechanical deformation in the metal, which alters the resistivity in the metal. By subjecting the metal to a constant electrical current, this change in resistivity can be measured, and converted into pressure measurements.

Electric power can be measured with a combination of voltmeter and ammeter, and mass flows can be calculated from pressure differences. It is important to have an idea where the measurements are coming from in order to determine their accuracy.

1.3 Artificial Neural Networks

Before trying to explain what an artificial neural network is, it might be helpful to introduce some basic elements. The idea of artificial neural networks comes from studying biological neural networks.

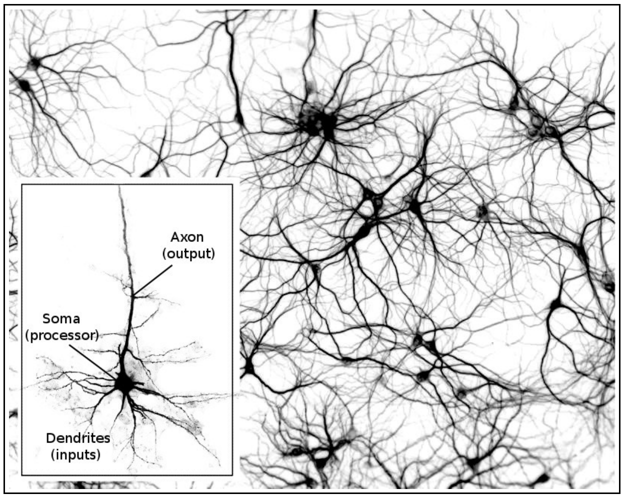


Illustration 16: A biological neural network consisting of several interconnected neurons. This image is a theoretical representation of what a biological neural network might look like.

Looking at a neuron (a nerve cell) commonly found in the human brain, it consists of three main areas; *dendrites*, *soma* and *axon* (this is not entirely accurate, seeing as how there are many different kinds of neurons, but for the sake of simplicity it will be sufficient). *Soma* is the main body, which processes information passing through the neuron. Attached to the *soma* are several *dendrites*, which receive signals (inputs) from other cells. An *axon*, which is much longer than the *dendrites*, is also attached to the soma. This is where the neuron sends out signals (outputs) to other cells. A connection between one neuron and another is called a *synapse*. Now lets look at an artificial neuron.

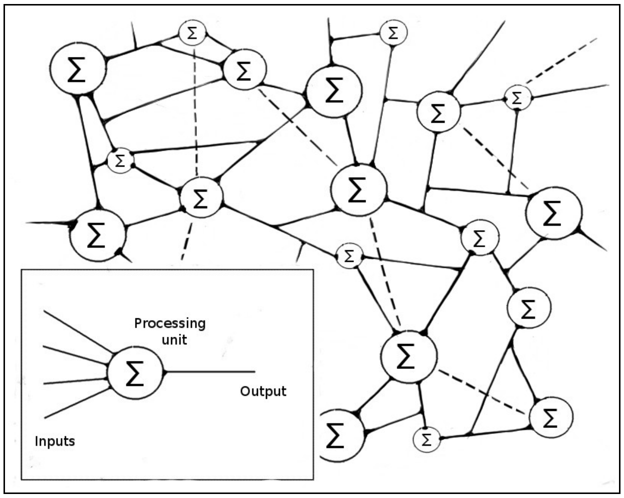
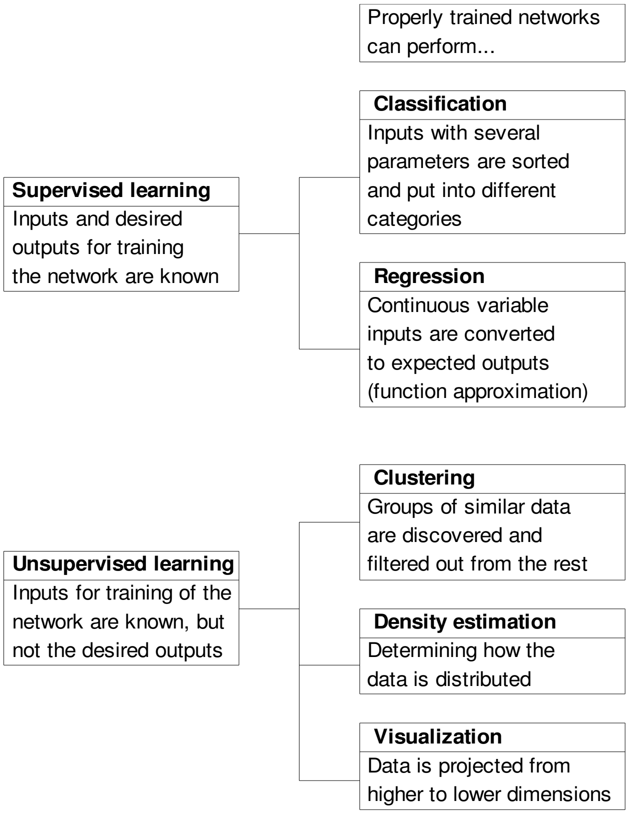


Illustration 17: An artificial neural network consisting of several interconnected neurons. It is not an illustration of an actual neural network, but means to imply the structure may be very complex.

Here we also find three main areas: Inputs, processing unit, and output (again, it can be more complicated than this). Connections between one neuron and others are called synaptic weights.

In general terms, one might describe artificial neural networks as groups of these neurons connected to each other, as well as to input- and output data, for the purpose of pattern recognition. Pattern recognition in this context means the automatic discovery of regularities in data through the use of computer algorithms.[6]

It might be helpful for that last statement to be explored a little further. Regularities in data can be used to achieve several objectives, ranging from simple linear regression or classification, to neural networks learning to play a game of chess. The chart on the next page gives a small introduction to the various possibilities involved.


Properly trained networks can perform...
Supervised learning
Inputs and desired outputs for training the network are known
Classification
Inputs with several parameters are sorted and put into different categories
Regression
Continuous variable inputs are converted to expected outputs (function approximation)
Unsupervised learning
Inputs for training of the network are known, but not the desired outputs
Clustering
Groups of similar data are discovered and filtered out from the rest
Density estimation
Determining how the data is distributed
Visualization
Data is projected from higher to lower dimensions

Seeing as how there are so many different kinds of neural networks, it would now be appropriate to focus specifically on the type of neural networks used to solve the problem explored in this thesis. These are the feedforward networks of layered neurons (multilayer perceptrons).

1.3.1 The perceptron

Originally, the artificial neuron was modeled to function much like it was *known* for a biological neuron to function. This amounted to a series of inputs which were either 1 (*excitors*) or -1 (*inhibitors*). Depending on whether the sum of these inputs exceeded a certain threshold value, an output of either 1 or -1 was activated. In 1949 Hebb introduced the idea that connections between neurons (synapses) could be strengthened or weakened as neurons were or were not stimulating each other when they were active. From his theories came the idea of adjustable synaptic weights, strengthening or weakening the inputs. These synaptic weights are adjusted differently by different learning algorithms.[8]

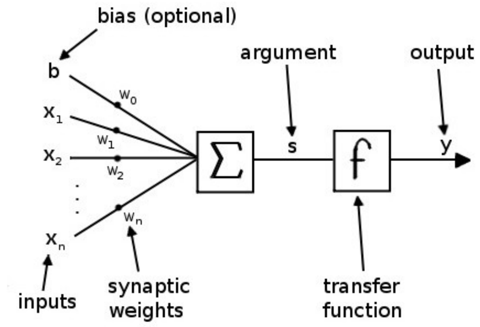


Illustration 18: The perceptron.

Ignore the bias, b, for now (assume b = 0), and notice how the input vector x, and the weight vector w, together become the argument, s.

1.3.2 Transfer functions

Looking at the illustration of the perceptron, one can see that the inputs are summed up into an *argument*, s, which is fed into the transfer function. The transfer function then decides what the value of the output will be.

In the first perceptron, the transfer function was just a threshold function. Judging by the value of the argument, s, the output, y, was either 1, or -1 (an alternative threshold function could give an output of either 1 or 0).

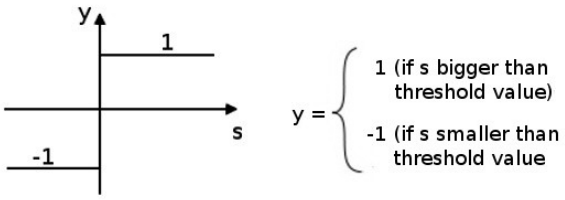


Illustration 19: Threshold function. The first transfer functions had only two possible outputs; 1 or -1 (alternatively 1 or 0).

These threshold functions have later been replaced with other transfer functions which are continuously differentiable. This is a very important property for a transfer function to have, because it opens up for gradient methods to be used in training of the network (more on this in relation to network training). These new transfer functions are the *sigmoidal* and *linear* transfer functions.

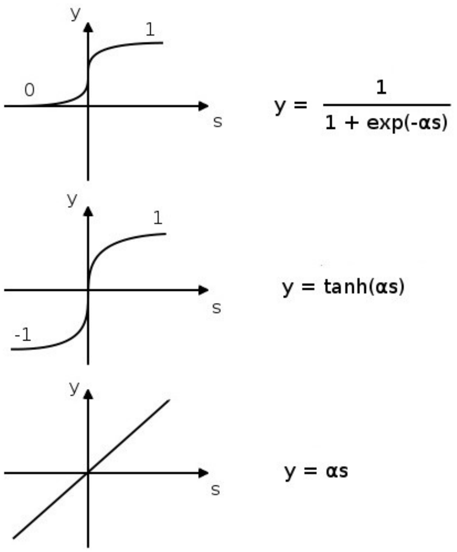


Illustration 20: Continuously differentiable transfer functions. From top to bottom: The logistic transfer function, the tanh transfer function and the linear transfer function. α decides how steep the function is at the origin.[4]

Again, the argument, s, made from the inputs and synaptic weights, will be transformed into the output, y, by the transfer function. But with a continuously differentiable transfer function, the output is no longer 1 or -1. It is now somewhere between 1 and -1. And in the case of a linear transfer function, the output is not constrained by these boundaries.

1.3.3 Multilayer perceptron

A single perceptron in itself does not have any remarkable computational abilities. If one creates a layer of neurons in a feed forward artificial neural network however, the computational abilities are amplified greatly.[8]

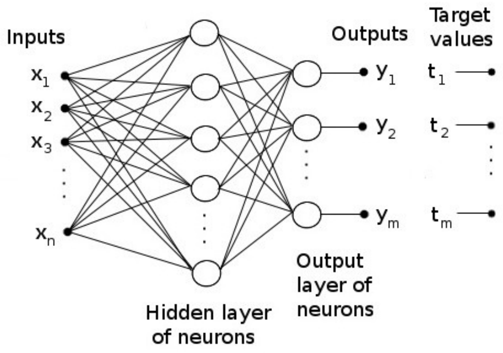


Illustration 21: A multilayer perceptron with one hidden layer.

Inputs are presented to the network on the left side, and through the synaptic weights they give impulses to the neurons in the hidden layer. From this layer, outputs are sent through synaptic weights and onwards to the neurons in the output layer. This layer sends out the network outputs.

Now, as can be seen on the right hand side of illustration 21, target values are introduced. During training, these target values (desired outputs) are compared to the network outputs, and training continues until they are approximately the same. This is achieved by altering the network's synaptic weights.

Since the inputs and target outputs are already provided, and the neurons themselves are just processing elements, this means that the synaptic weights are where the network's *memory* is stored once it is trained.

A multilayer perceptron like this can in theory approximate any nonlinear (or linear for that matter) relationships between n-dimensional input data and m-dimensional target data given that the hidden layer has enough neurons (Hornik-Stinchcombe-White-Cybenko theorem).[8]

Adding a second hidden layer before the output layer can prevent the necessity of using an excessive number of neurons in a single hidden layer. There is also a benefit from adding an extra hidden layer when there are a large number of input parameters. With only one hidden layer, all inputs are connected to all neurons in the hidden layer. But a second hidden layer is not subject to this, and the information could therefore be divided into smaller blocks here and processed more efficiently.[12]

However, more layers decrease processing speed, so this will be for the designer of the network to optimize.[8]

1.3.4 Error backpropagation

Without going too deeply into the mathematics involved, an attempt will now be made to explain how the network can be trained.

Consider the multilayer perceptron in the previous section. There are inputs, synaptic weights, layers of neurons, outputs, and target values. The difference between the network outputs, y, and the target values, t, is called the error, e ($e = y - t$). The objective of training the network then becomes to minimize the error ($e \rightarrow 0$).

If one were to take the mean of all the errors squared, one would have the widely used performance criterion MSE (mean square error). The goal then is to find the minimum MSE.

$$MSE=\frac{1}{2\mathrm{m}}\sum_{i=1}^{m} e_i^2$$

Since the error is a function of the synaptic weights, we can assume that the MSE is also a function of the synaptic weights; MSE = MSE(w). Finding the minimum MSE can then be done by taking the partial derivative of the MSE with regards to the synaptic weights, and setting it equal to zero.

$$\frac{\partial MSE}{\partial w}=0$$

Solving this can only be done numerically (read: approximated through iteration). For every iteration, the synaptic weights are adjusted slightly in the direction of the steepest descent of the gradient. The synaptic weights closest to the outputs are adjusted first, and then the synaptic weights connected to the inputs are adjusted. Hence the expression backpropagation. How big an individual adjustment is, and when to stop the iterations (finish training) depends on several conditions set by the network designer.

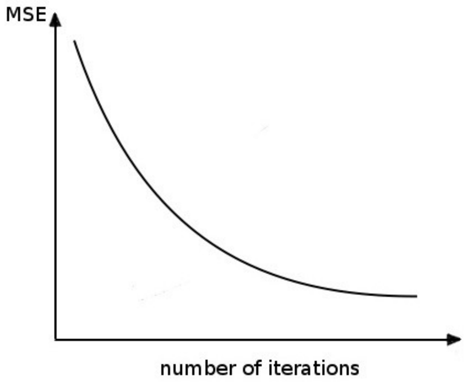


Illustration 22: The mean square error as a function of all synaptic weights, w, calculated by iteration.

Notice that the MSE in the illustration above is a function of all the synaptic weights in the network. The goal of the training is to find the optimal values for every synaptic weight which in combination provide the minimum MSE.

In relation to the process of finding the optimal values for the synaptic weights, there is a certain problem which might occur. Imagine a general function, f(w), where w represents one or more synaptic weights. If one were to take the derivative of f with respect to w, looking for the function minimum, there is a chance one might find a local minimum. This is a problem that needs to be considered when training a network.

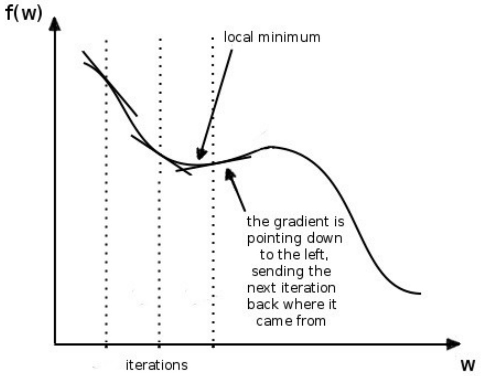

f(w)
local minimum
the gradient is pointing down to the left, sending the next iteration back where it came from
iterations
w

Illustration 23: A function f(w) trying to find the minimum value of w, risks ending up in a local minimum instead of the desired global minimum.

1.3.5 Training algorithms

As could be seen in the previous section, each iteration adjusts the synaptic weights slightly. There are however several different ways to do just this. How big steps should each iteration take? Is there another direction than the steepest descent of the gradient that could be used? Should the weights be adjusted after running all input data through the network, or should they be adjusted continuously for each input?

Training algorithms are designed to minimize calculation effort and get the best result. It should be emphasized that different problems may require different training algorithms for optimal training of the network.

It is a bit difficult to go into detail on the different training algorithms here, seeing as how they are complex numerical matrix calculations. A brief look at the simple backpropagation algorithm could however help give some better understanding of what a training algorithm is.

$$w_{hij}(k+1)=w_{hij}(k)-\eta\frac{\partial g_j(k)}{\partial w_{hij}(k)}$$

In this simple algorithm $w_{hij}(k)$ is a weight connected to two neurons; the ith neuron in the jth layer, and the hth neuron in the layer before the jth layer (see Appendix I for an example of how this algorithm is implemented). ŋ is called the step size, and in more advanced algorithms this is often designed to change as necessary. In this simple algorithm however, it is just a constant. Multiplied with the step size is the gradient, calculated for each individual weight. This last factor decides the direction of the next step.[4]

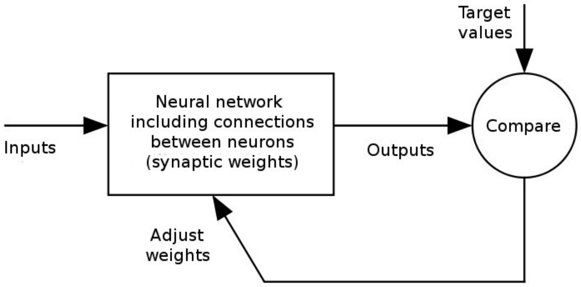

Target values
Neural network including connections between neurons (synaptic weights)
Inputs
Outputs
Compare
Adjust weights

Illustration 24: Flow chart describing the principle behind training algorithms when the target values are known.

A couple of training algorithms have been tested in the networks created for the problems in this report. The problems were attempted solved with autoassociative neural networks, and these networks are used for function approximation. The author ended up using the *Scaled Conjugate Gradient* training algorithm, which is good for function approximation with large datasets.[5]

1.3.6 Regression

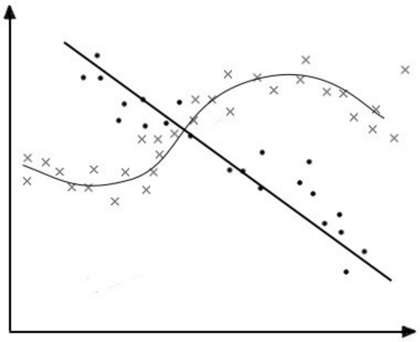


Illustration 25: A linear curve has been drawn to approximate the dots in the diagram. A nonlinear curve has been drawn to approximate the crosses.

Given a set of two dimensional data, one can draw an approximate continuous *best fit* line through it. As seen in illustration 25, this best fit can be both linear and nonlinear.

When training a network, this is sometimes what we are after. A best fit between the input and output data. Whether this is a two dimensional or a higher dimensional dataset does not matter, but a two dimensional dataset is easier to visualize. Therefore, a two dimensional example will be used to explain the phenomenon called *overfitting*.

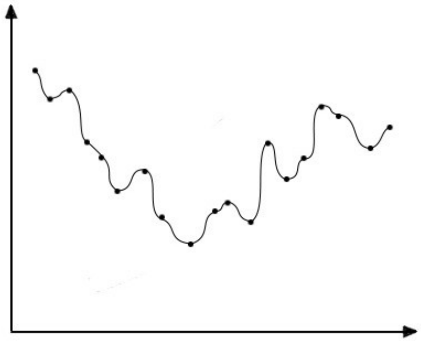


Illustration 26: An example of overfitting.

In this case the network has been overtrained. A solution has been found, but it is not the general solution we were looking for. The same scenario applies to higher dimensional datasets, although that would, again, be difficult to visualize.

Whether overfitting occurs or not is largely dependent on the training data and training algorithm. Avoiding overfitting is a result of the training algorithm being able to separate the relevant information from the noise found in data. And this generalization is ultimately the goal when training an artificial neural network.

1.3.7 Preprocessing

Before the inputs and target values can be used to train the multilayer perceptron, they need to be preprocessed.

The first thing to do is to randomize the datasets. This is done to help prevent the training process getting stuck in a local minimum.

Sample	Input	Target
1	67	8
2	64	7
3	62	7
4	55	6
5	53	7
6	47	4
7	46	4
8	44	4
9	41	3

Sample	Input	Target
3	62	7
5	53	7
1	67	8
6	47	4
7	46	4
2	64	7
9	41	3
4	55	6
8	44	4

Illustration 27: An example of how data is randomized before training. The table on the left is in order, while the table on the right has been randomized.

It is also common to normalize the data before training. This means converting the data to within a specified range (approximately between -1 and 1 is the standard), which makes the network easier to train because the transfer functions are more susceptible to these values.

After the data has been randomized and normalized, it can be divided into three parts; *training* set, *cross-validation* set and *test* set. Usually 60% - 70% of the data is used in the training set, while the rest is equally divided between the cross-validation and test sets.

The training set is used to train the network. This means the weights are adjusted with regards to the data in the training set.

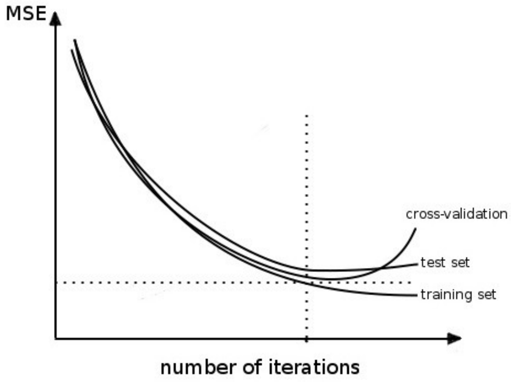


Illustration 28: If the cross-validation set diverges from the training set, that can be an indication of overfitting, and training should be stopped.

If the cross-validation set, using the adjusted weights, deviates too much from the training set, training should stop. This is what the cross-validation set is for. And if the test set corresponds nicely with the training set, that may be an indicator of good generalization.

1.3.8 Principal component analysis

Principal component analysis is a technique for mapping multidimensional data into lower dimensions with minimal loss of data.[9] Given a n-dimensional dataset, it is possible to reduce the number of dimensions to less than n, loosing redundant information in the process.

Linear principal component analysis is a straight forward matrix calculation. It sorts out the parameters and lists them from the most relevant and down to the least relevant with regards to reconstruction. The idea is then that the least relevant can be discarded (the data can be compressed) without significant loss of information.

But if the data is nonlinear, principal component analysis does not provide very good results. Neural networks however, can do this by using an AANN architecture.

1.3.9 Autoassociative Neural Networks

Using three hidden layers in a multilayer perceptron, it is possible to perform a nonlinear principal component analysis of a dataset.[9]

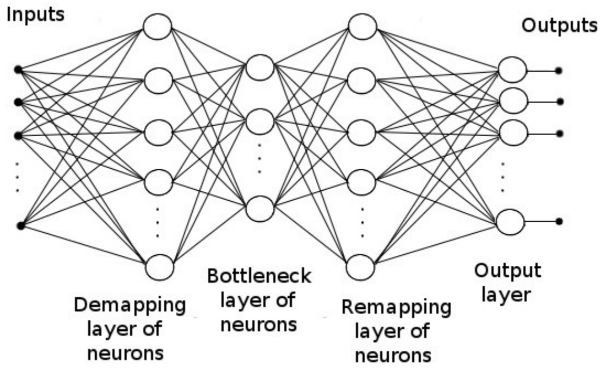

Inputs
Outputs
Demapping layer of neurons
Bottleneck layer of neurons
Remapping layer of neurons
Output layer

Illustration 29: An autoassociative neural network. There must be more input and output parameters than there are neurons in the bottleneck layer.

Looking at the illustration above, notice that if the inputs are used as both inputs and target values, the network will be trained to perform an *is equal to* mapping.

Imagine that this dataset has n dimensions. If the second hidden layer has less than n neurons, it forces the data through a *bottleneck*, removing redundant information. When the data is reconstructed on the other side of the bottleneck, it cannot be completely recovered, because some of the information is missing. However, if the difference between outputs and target values (inputs) is small enough, one can read a less than n-dimensional dataset from the bottleneck layer which is approximately correct.

1.4 Noise reduction

The sensors used to measure temperatures, pressure, mass flow, and other properties, in a gas turbine, will not be 100% accurate. To some extent there will be random noise, and the idea is that this noise follows a normal distribution.

Consider the illustration below. Given a large number of measurements, they will be distributed and focused around the exact value.

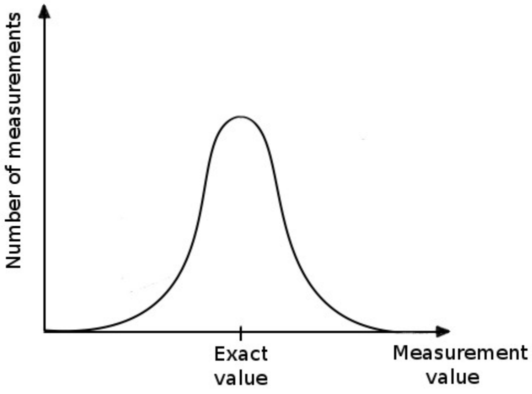


Illustration 30: Normal distribution.

Now, for various reasons it might be desirable to reduce this noise in order to make measurements more accurate. There are some simple linear filters that provide excellent noise reduction, but they do have limitations. Exponential smoothing is one such filter, and it will be used here as an example.

1.4.1 Exponential smoothing

The algorithm itself is quite simple. Let **x** represent the measured (noisy) values, and **s** the estimated (filtered) values. α is a smoothing constant which is set between 0 and 1.

$$s_0 = x_0$$

$$s_t = \alpha x_{t-1} + (1-\alpha) s_{t-1}$$

As shown in illustration 31, the algorithm finds an average which corresponds to the incoming data.

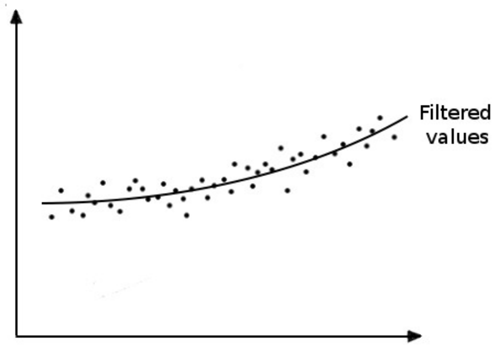


Illustration 31: Data filtered with exponential smoothing.

But if the data should encounter a sharp increase or decrease in value, which could for example be a result of a system malfunction or faulty sensor, the exponential smoothing will not only smooth out the noise. It will also smooth out the shift in the data pattern. Whether this is acceptable or not depends on what tasks the filter is expected to perform.

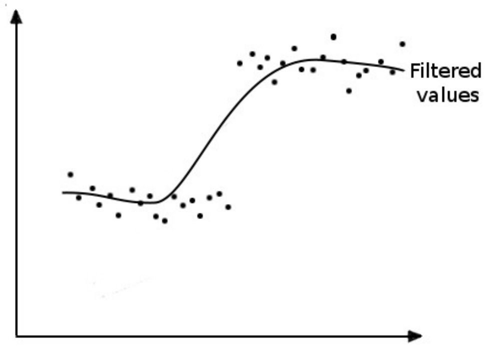


Illustration 32: Data with a trendshift, filtered with the exponential smoothing algorithm.

1.4.2 AANN noise reduction[10]

Consider the network described in section 1.3.9; the autoassociative neural network. If the input parameters are sensor readings from different parts of the gas turbine, it is reasonable to assume that they are to some extent correlated. Meaning if one parameter changes, the other parameters can be expected to change as well. An increase in power to the compressor will result in an increased rotational speed.

The idea of AANN noise reduction is that the network should be trained to recognize these correlations between parameters. Measurement noise is not correlated between the sensors in a gas turbine, because each sensor has its own random noise.

Finding which parameters correlate to each other can be done to some extent (and it should be emphasized that this method might not be useful at all) by analyzing the covariance matrix, **R**, of the training data. Given n input parameters, regardless the number of datapoints, the covariance matrix will be a nxn matrix. If an element in this matrix R_{ij} is zero (or statistically indistinguishable from zero), then the parameters i and j are independent of each other. Rearranging **R** into a block diagonal form reveals the dependency structure between parameters, and each square block of nonzero elements represents a set of mutually correlated variables. There is no benefit derived from introducing two independent groups of variables into a single autoassociative neural network, since no correlations will be found between them.

Another important factor in noise filtering is redundancy. It reduces variance the same way that taking samples containing multiple items reduces variance in statistical quality control.

1.4.3 ANN noise reduction

An ANN can also provide some noise reduction. However, the concept is a bit different from AANN. With an AANN parameters are put into the network and the same parameters come out, filtered. With an ANN certain parameters are used as inputs, and these inputs are then used to estimate a completely different parameter (remember the multilayer perceptron in section 1.3.3).

Assuming the ANN is not overtrained (see section 1.3.6), the estimated output generated should be somewhat noise free, simply because one of the criterea of finding a general solution is that the ANN discards irrelevant information (noise).

1.5 IPSEpro

IPSEpro is a heat and mass balance program developed by SimTech. It is a software package which uses thermodynamic tables and equations to create system models. Several components can be linked together (like for example the components of a gas turbine), and the program will calculate the different parameter values inside the system. Just like an engineer could do by hand using diagrams, tables and thermodynamic equations. The computer will of course do this much faster.

1.5.1 Creating a gas turbine model

Each component is picked out of a library and put down on a flow sheet, and certain design characteristics are specified for all of them individually. The components are then linked together.

After specifying the initial conditions of the working fluid (or working fluids), the program calculates the state of the working fluid in every part of the system. To find out how a small change in one of the components or initial conditions then influences the calculations, the designer only needs to change said conditions.

1.5.2 Shortcomings of the IPSEpro model

A model built with a set of theoretical equations and thermodynamic tables with their own limitations will not be a perfect representation of reality.

The model would also become very complex if one were to include ever single factor that could influence the calculations. Certain auxiliary systems could probably be excluded without influencing the calculations too much, but some accuracy is inevitably lost.

2 Work

In this chapter the methodology used during testing is introduced. It has been divided into six parts, and starts with a theoretical AANN example to provide a step by step overview of how MATLAB can be used to test an AANN. Most of these MATLAB commands are listed with descriptions in Appendix II for easy reference.

Next is another theoretical AANN example, before testing methods used on the IPSEpro dataset and the Turbec dataset are described. The methods used in the various cases are similar to each other. Therefore, instead of repeating these several times, the later parts of the chapter are shorter, mentioning more how one case differs from the others (for example how the method used for the first theoretical AANN example differs from the case with the Turbec AANN).

2.1 Theoretical 2D AANN example

As an introduction to noise filtering using AANN, example datasets were created and used to make networks. This was done for the author to become familiar with AANN in practice, and is written out here to help introduce the concept to the reader. The two parameters used in this first example are listed here.

$$x \in [-2, 2]$$
$$y = x^2 + \exp(x)$$

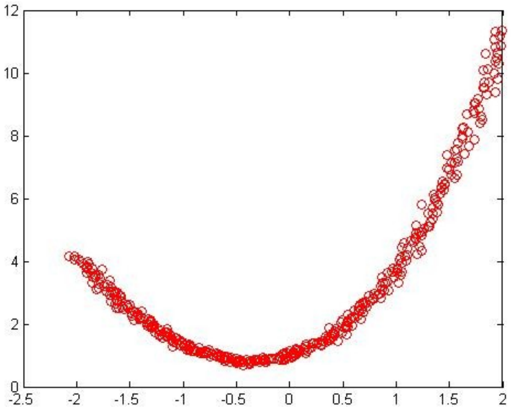


Illustration 33: An example of a system of two interdependent parameters, to each of which has been added some white noise.

2.1.1 Acquisition and preprocessing of data

Generating the datasets for this example is simple in MATLAB. The following code gives numeric values corresponding to the equations above.

```
x = -2:0.01:2;
y = x.*x + exp(x);
```

Adding white noise to each individual parameter is then done using the following commands.

```
xN = x + 0.05*randn(size(x));
yN = y + 0.05*randn(size(y));
```

This adds random (gaussian) noise to each parameter. Here, size(x) is used to tell the *randn* command how many sample points the vector **x** has, while 0.05 affects the size of the variance in the noise. Measurement noise in a sensor found inside a gas turbine system will be simulated this way later, only using different noise values.

With the dataset ready, preprocessing can be done. In this case, that will be randomizing and normalizing the data. These following commands will perform said tasks.

```
shuffle = randperm(length(x));
p_input = [xN(shuffle); yN(shuffle)];
[p_inputn,ps] = mapminmax(p_input,-0.8,0.8);
```

p_input is a 2x401 matrix containing the two randomized parameters with noise. **p_inputn** is a 2x401 matrix containing the data from **p_input** normalized to fall between the values -0.8 and 0.8. This matrix will be used as input for training of the network.

2.1.2 Building the AANN

Building an AANN in MATLAB requires that one builds a custom four layer artificial neural network. The complete code to build such a custom network can be found in Appendix II. Once this code is called it builds a network from the input matrix and the desired number of neurons in the three hidden layers.

```
net = makebottle(p_inputn',[6,1]);
```

Consider this MATLAB expression at the bottom of the previous page. Calling on the makebottle function from the file makebottle.m this way will create a 2:6:1:6:2 autoassociative neural network named net. The number 2 in both input and output comes from the **p_inputn'** matrix, which contains our dataset. The number of neurons in the three hidden layers are specified when calling on the makebottle function.

If this is confusing, a closer look at the makebottle code found in Appendix II along with illustration 29 might be helpful.

2.1.3 Training the network

There is a tool in MATLAB called *nntool* (a more detailed description of the *nntool* has been included in Appendix III of this report). It is activated by typing nntool in the command line. Input data, desired target data (same as input data), and the network created with the makebottle function should be imported into the *nntool*.

When this is done, open the imported network inside *nntool*, and reinitialize the weights. This gives each synaptic weight a random start value before training, which helps prevent the training from getting stuck in a local minimum.

Set the maximum number of iterations desired, and the number of cross-validation errors allowed. In this example these values were set to 1000 and 20. Training is then started by clicking *train network*, and it stops after 1000 iterations or 20 cross-validation errors.

2.1.4 Producing the results

While a network is being trained, MATLAB automatically generates a performance curve, regression curve, gradient curve, and a cross-validation error diagram. These can be useful to examine in order to determine whether the network has been properly trained.

If these plots are acceptable, the network may be exported to the working directory in MATLAB for simulation. There the newly trained network will be exposed to new data, and the outputs produced are the *simulated data*. These simulated data have then hopefully been filtered from white noise.

In order to find the best possible network architecture, the following network architectures were systematically tested. Notice that three of them have two extra hidden layers. This was tested because of the possible benefits mentioned in section 1.3.3 of this report.

2:3:1:3:2
2:4:1:4:2
2:5:1:5:2
2:7:1:7:2
2:10:1:10:2
2:3:5:1:5:3:2
2:3:7:1:7:3:2
2:5:10:1:10:5:2

Of these, the network 2:5:10:1:10:5:2 produced the best results, and shall be examined further here.

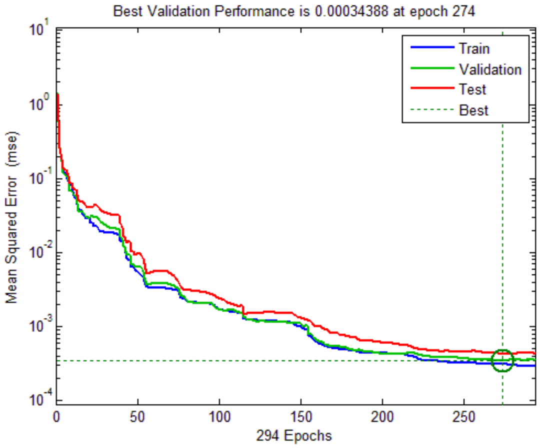


Illustration 34: Performance curve produced while training the 2:5:10:1:10:5:2 network.

The performance curve looks good. Training, cross-validation and test sets are not deviating from each other to any serious extent. Although more training could probably continue in order to lower the MSE, training here stopped after 294 iterations because there were 20 consecutive cross-validation errors. It could be interesting to set the cross-validation error limit higher and achieve a lower MSE, but that could also make the network more susceptible to overfitting. Ultimately it is up to the designer to make a decision, but for this example the current training will be deemed sufficient.

Next is the regression plot in illustration 35. It also looks good, to some extent. All three sets fall almost completely parallel to the target, but the cross-validation and test sets seem to have somewhat clustered data in stead of it being spread out evenly. This graph might be a little confusing, so perhaps it should be clarified a little.

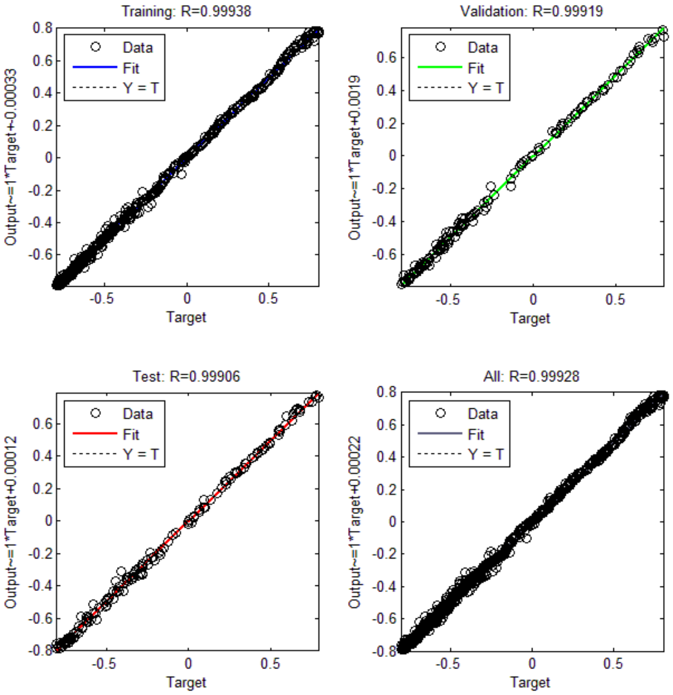


Illustration 35: Regression plot produced while training the 2:5:10:1:10:5:2 network.

Remember section 1.3.6, showing how regression means finding the best fit. This graph indicates how close to regression this example is. Notice how it is not an example with linear regression, and therefore overfitting becomes a concern. It is however, not possible to decide one way or the other just from looking at a regression plot.

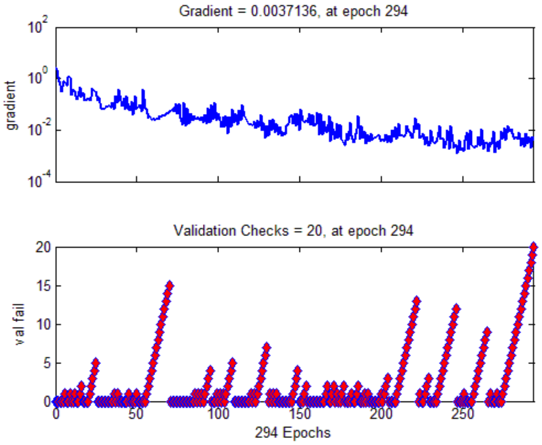


Illustration 36: Gradient curve and cross-validation diagram produced while training the 2:5:10:1:10:5:2 network.

The plot in illustration 36 is pretty straight forward. The gradient fluctuates repeatedly but overall reduces throughout the training. There are some cross-validation errors during parts of the training, which could also be an indication of overfitting.

That concludes the training part of the 2:5:10:1:10:5:2 network. It was then exported into the MATLAB working directory for simulation.

Simulation was done with a new set of data, meaning the same x and y data with fresh noise values generated the same way as before. The following commands produced this new set.

```
x_new = x + 0.05*randn(size(x));
y_new = y + 0.05*randn(size(y));
p_new = [x_new; y_new];
[p_newn,ps] = mapminmax(p_new,-0.8,0.8);
```

As can be seen, the new data is noisy and normalized, but not randomized. It is noisy because the objective here is to filter this noise through simulation. It is normalized because the network was trained with normalized data, and will therefore not work with data that has not been normalized. But it is not randomized. This is because randomization is only necessary to prevent training from getting stuck in a local minimum. After the network is trained, randomization no longer serves a purpose.

Simulation can then be done by feeding **p_newn** into the network, which produces the simulated (filtered) data. The following commands do just this, and then converts the simulated data so that it is not normalized, and can be compared to the noisy (unfiltered) data.

```
a_simn = sim(net,p_newn);
a_sim = mapminmax('reverse',a_simn,ps);
```

Two sets of data are now ready to be compared; unfiltered noisy data, and filtered (hopefully noise free) data. In order to compare them, the difference between each set and the completely noise free data (plain x and y in the form of the matrix **clean_data**) is calculated.

```
N_diff = p_new - clean_data;
a_diff = a_sim - clean_data;
```

These differences are then plotted together, giving an image of how much noise has been reduced through simulation in the AANN.

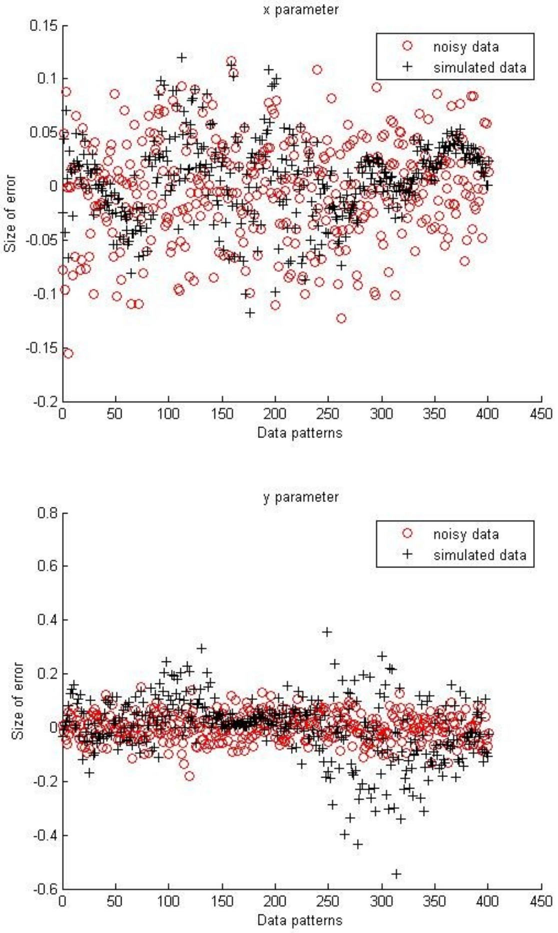


Illustration 37: The unfiltered noisy data compared to the filtered (simulated) data, plotted for both parameters. This is from the 2:5:10:1:10:5:2 network.

From the plotted results there is unfortunately no noise reduction to be seen for the individual parameters. The filtered data also seems less random than the noisy data, which could mean that the network has not been able to find the correlations between the parameters to the extent desired, and has therefore not been able to discard the noise factor as irrelevant to training.

On the other hand, a plot of the filtered x and y values together gives an interesting result. The filtered curve shown in illustration 38 is far less noisy than the curve shown in illustration 33.

Regardless, the results will be interpreted later. The objective here is simply to introduce the methodology which will be used to examine noise reduction by AANN in this report.

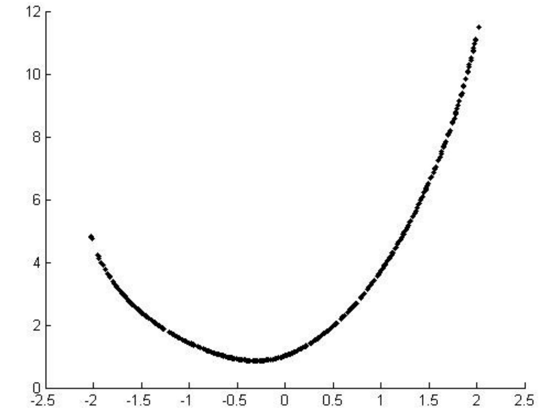


Illustration 38: The x and y parameters plotted together after running them through the 2:5:10:1:10:5:2 network.

2.2 Theoretical 3D AANN example

The second AANN example is a dataset with three parameters; a surface plot. Here three parameters will be interdependent of each other.

$$x, y \in [-2, 2]$$
$$z = x^2 + x\exp(y)$$

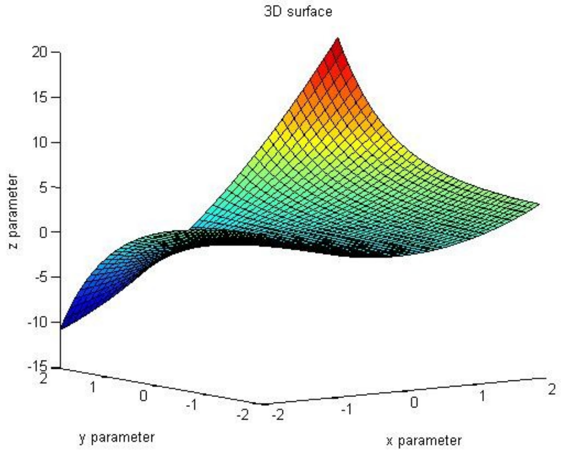


Illustration 39: An example of a system of three interdependent parameters. This is the clean dataset with no noise added to it.

The dataset was created in MATLAB by using the following commands.

```
[x,y] = meshgrid(-2:0.1:2,-2:0.1:2);
z = x.*x + x.*exp(y);
```

Noise was added with the *randn* command, and preprocessing was done in a similar way as to the first example. This means the data was randomized and normalized before training.

For this example, the following network architectures were tested. And again, some of the networks have two extra hidden layers.

3:5:1:5:3
3:7:1:7:3
3:9:1:9:3
3:5:2:5:3
3:7:2:7:3
3:9:2:9:3

3:5:7:1:7:5:3
3:5:10:1:10:5:3
3:5:7:2:7:5:3
3:5:10:2:10:5:3
3:7:15:2:15:7:3

After training, a dataset with new random noise values was filtered through the networks, just like in the earlier example. The results could then be plotted, comparing noisy data with filtered data.

2.3 AANN with data from IPSEpro

The model of the Turbec T100 CHP gas turbine using IPSEpro produced a set of data which consisted of three input parameters, and eight output parameters. There were 13578 sample points taken for each parameter.

1	Input	Ambient Pressure
2	Input	Ambient Temperature
3	Input	Ambient Humidity
4	Output	Mass Flow out Compressor
5	Output	Pressure out Compressor
6	Output	Temperature out Compressor
7	Output	Mass Flow Fuel
8	Output	Temperature out Turbine
9	Output	Pressure out Turbine
10	Output	Generator Power
11	Output	Shaft Power Compressor

Table 1: The eleven parameters included in the IPSEpro dataset; a model of the Turbec T100 CHP gas turbine.

2.3.1 Clean data

An initial examination of the dataset provided some insights which could be useful later. First a look at the range of each individual parameter.

Parameter	MIN	MAX	RANGE
Ambient Pressure [bar]	0.9750	1.0400	**0.0650**
Ambient Temperature [°C]	-15.0000	25.0000	**40.0000**
Ambient Humidity [%]	35.0000	90.0000	**55.0000**
Mass Flow out Comp. [kg/s]	1.9980	1.9989	**0.0009**
Pressure out Comp. [bar]	4.3875	4.6800	**0.2925**
Temperature out Comp. [°C]	118.7315	250.2866	**131.5551**
Mass Flow Fuel [kg/s]	0.0173	0.0179	**0.0007**
Temperature out Turbine [°C]	629.1326	641.8474	**12.7148**
Pressure out Turbine [bar]	1.0140	1.0140	**0.0000**
Generator Power [kW]	70.9793	151.6751	**80.6958**
Shaft Power to Comp. [kW]	-578.8826	-496.7885	**82.0941**

Table 2: Range of each individual parameter in the dataset created with IPSEpro.

Some of the parameters vary significantly over the whole range, while others almost do not change at all. Temperature out of the compressor varies over a range of 131.56 Kelvin with varying ambient temperature and humidity. As expected, this has a noticeable effect on generator power and shaft power to the compressor.

Another aspect of the dataset which was subject to initial examination, is the covariance matrix, **R**, from section 1.4.2. Below can be seen a representation of this matrix where 0 denotes a value indistinguishable from zero, and X denotes a value which is not zero.

	1	2	3	4	5	6	7	8	9	10	11
1	X	X	X	0	X	0	0	X	0	X	X
2	X	X	X	X	X	X	X	X	0	X	X
3	X	X	X	X	X	X	X	X	0	X	X
4	0	X	X	0	0	X	0	X	0	0	X
5	X	X	X	0	X	X	X	0	0	X	X
6	0	X	X	X	X	X	X	X	0	X	X
7	0	X	X	0	0	X	0	X	0	X	X
8	X	X	X	X	X	X	X	X	0	X	X
9	0	0	0	0	0	0	0	0	0	0	0
10	X	X	X	0	X	X	X	X	0	X	X
11	X	X	X	X	X	X	X	X	0	X	X

Table 3: Covariance matrix for the eleven parameters included in the artificial dataset created with IPSEpro. X means a non-zero value, while 0 means a value statistically indistinguishable from zero.

Parameter 9, pressure out of the turbine, appears to be independent of the other parameters (from table 2 it can also be seen that it is actually constant), and can therefore be removed from the training set. All the other parameters however, might be dependent on each other to some extent, and will therefore be trained together in the autoassociative neural network.

2.3.2 Data with noise

As with the dataset in the AANN example at the beginning of this chapter, white noise was also added to each individual parameter in the dataset created with IPSEpro. Thus simulating the measurement noise that comes with the sensors in a real gas turbine. These parameters however will not have the same level of noise. Different parameters are measured with different types of sensors, which again are subject to different levels of accuracy. The table below shows the noise values used.

Parameter	Noise added
Ambient Pressure	± 0.01 bar
Ambient Temperature	± 0.2 °C
Ambient Humidity	± 3 %
Mass Flow out Comp.	± 0.1 kg/s
Pressure out Comp.	± 0.01 bar
Temperature out Comp.	± 0.2 °C
Mass Flow Fuel	± 0.02 kg/s
Temperature out Turbine	± 2 °C
Pressure out Turbine	± 0.01 bar
Generator Power	± 0.5 kW
Shaft Power to Comp.	± 0.5 kW

Table 4: Amount of noise added to IPSEpro dataset.

The data collected from the real Turbec T100 CHP at Risavika provided an opportunity to read what approximate level of noise one can expect from certain sensors. Some of these approximations are listed in the table above (more specifically pressure, temperature and power output). The rest were found by comparing accuracy levels listed in sensor documentation found on the Internet and advice from people with experience from this industry. Certainly not ideal, but the author assumes it is sufficient for the theoretical scenario investigated in this thesis.

These noisy data were collected in a 10x13578 matrix (pressure out of turbine excluded) called **p_input_noise**, before they were randomized and normalized for training.

```
shuffle = randperm(13578);
p_input_noise = [p_input1(shuffle); p_input2(shuffle);  . . .
p_input10(shuffle)];
[p_inputn,ps] = mapminmax(p_input_noise,-0.8,0.8);
```

p_inputn was then used to both build and train the autoassociative neural networks, using the makebottle code and the *nntool* (see Appendix II and III for a more detailed description of these).

In total, twelve network architectures were tested on this dataset. Eleven networks with the ten parameters which were dependent on each other, and one network with all eleven parameters. This last one was tested to verify that the last parameter, pressure out of the turbine, did not have any effect on the noise reduction of the other parameters.

10:12:3:12:10
10:12:5:12:10
10:12:7:12:10

10:15:3:15:10
10:15:5:15:10
10:15:7:15:10

10:20:3:20:10
10:20:5:20:10
10:20:7:20:10

11:20:3:20:11
10:12:15:5:15:12:10
10:15:20:5:20:15:10

Two of the networks that were tested had two extra hidden layers, to see if such an architecture would produce any different results than the three hidden layer networks (see section 1.3.3).

Networks were trained for 2000 iterations with cross-validation error limits set to 200.

Although there is a large compression of data in the bottleneck layer in some of these networks, the networks could be trained without any serious deviations between training set and cross validation set.

2.3.3 Data with noise and outliers

Outliers are not uncommon in measurements taken from a real gas turbine. Their presence may indicate various scenarios are taking place, and it might also be interesting to see how a dataset containing outliers affect the noise reduction provided by an AANN.

This could be done by simply replacing random measurements in the dataset used to train the AANN with outliers, and then run a simulation of the new dataset through the already trained AANN.

2.4 AANN with data from Turbec

Measurements from the Turbec T100 CHP at Risavika gas center were also filtered through an AANN. The parameters however, were not the same as the ones from the IPSEpro model.

1	Generator power [kW]
2	Water temperature [°C]
3	Rotational speed [%]
4	Turbine outlet temperature [°C]
5	Gas pressure [mbar]
6	Oil temperature in [°C]
7	Ambient temperature [°C]
8	Acc [g]
9	Electrical energy produced [MWh]
10	Air filter pressure [Pa]

Table 5: The ten parameters included in the measurements taken from the Turbec T100 CHP gas turbine.

There were 7054 samples taken for each parameter, with an approximate time interval of 51 seconds between each measurement.

2.4.1 Preprocessing of data

The dataset was first put into a 10x7054 matrix. It was then randomized and normalized before training, using these commands.

```
shuffle = randperm(7054);
p_input = [p_input1(shuffle); p_input2(shuffle);  . . .
p_input10(shuffle)];
[p_inputn,ps] = mapminmax(p_input,-0.8,0.8);
```

2.4.2 Building the AANN model

Networks were created with the makebottle code and the input matrix, **p_inputn**. Same method as used with the first AANN example. In this case, the following architectures were tested.

10:12:2:12:10	10:12:15:3:15:12:10
10:12:3:12:10	10:12:15:5:15:12:10
10:12:5:12:10	10:15:20:5:20:15:10
10:15:3:15:10	
10:15:5:15:10	

2.4.3 Training and producing results

Training was done with the *nntool*, using 1000 iterations with cross-validation error limits set to 50. Again, this method is described in detail in Appendix III.

After training, the original dataset was put into a new matrix, **p_new**, which was then normalized and filtered through the networks using these commands.

```
[p_newn,ps] = mapminmax(p_new,-0.8,0.8);
an = sim(net,p_newn);
a = mapminmax('reverse',an,ps);
```

Now, plotting the results. This could not be done the same way as with the earlier datasets. In this case, there was no clean data to check the level of noise. There was only noisy and filtered data.

The filtered data was therefore plotted on top of the noisy data, giving an indication of whether there was any noise reduction. An example of this technique is shown here in illustration 40.

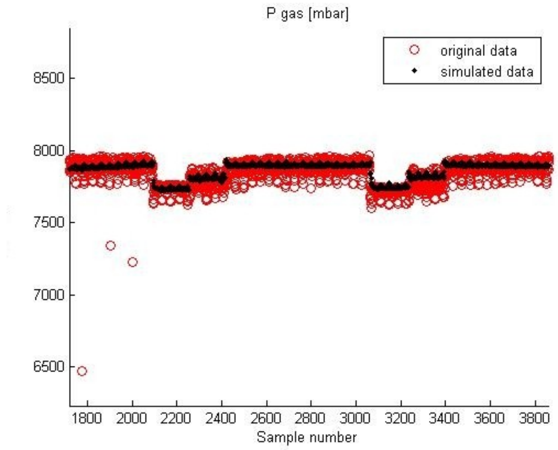


Illustration 40: Gas pressure measurements filtered through 10:12:3:12:10 network. Simulated data means filtered data.

It is only part of the results for this parameter (samples 1800 to 3800), but it shows how filtered data is plotted on top of original measurements. In this case it shows the principle behind the plotting method nicely, because the filtered data looks cleaner than the original data.

Whether the original data here varies because of measurement noise or simply because the gas turbine continually adjusts itself to keep the power output constant, is of course important to consider.

2.5 ANN with data from IPSEpro

As mentioned in section 1.4.3, some noise filtering may also take place using an ANN. This was tested briefly to see if there was any chance it could be more effective than the AANN approach. The methodology used will be presented here.

2.5.1 Preprocessing of data

There were two different approaches to this. The first one taking into account that eight of the parameters in the IPSEpro dataset were calculated, while three of them were inputs (see table 1). For this reason, one could expect each of the eight calculated parameters to be dependent on the three input parameters. For the purpose of constructing the ANN models, the three inputs were put in an input matrix, **p_input**, while the outputs were used each in their own network.

One network would then consist of inputs, one hidden layer, and one output layer (see illustration 21 for an image of the architecture). The inputs were represented by the same input matrix in all networks trained, while there were eight different networks for the eight different outputs. For example, in order to filter the mass flow out compressor parameter, a network was trained with **p_input** as inputs, and mass flow out compressor as the target value. The architecture would then be 3:5:1, or 3:8:1 (the hidden layer can be anything bigger than 3). After the network was trained, feeding **p_input** into the network would then give an estimate (which in this case means filtered) of mass flow out compressor.

Another configuration which was tested was to use ten parameters as inputs and one parameter as output. This was done for eleven networks, one for each parameter. With this configuration, filtering mass flow out compressor meant that the other ten parameters were used as inputs, while mass flow out compressor was the target value. The architecture would then be 10:15:1 (the hidden layer needs to be bigger than 10). This configuration could be easier to train simply because it has more parameters to find correlations with the target value.

All parameters were first randomized, had noise added to them (same noise values as in table 4), and then normalized for training.

2.5.2 Building the ANN model

This was done in much the same way as when building the AANN network. The difference was the architecture. Instead of the AANN architecture, with two hidden layers, one bottleneck layer and one output layer, there was only one hidden layer and one output layer. The author made this by configuring the makebottle code seen in Appendix II, but there are easier ways this simple network architecture can be created. Three network architectures were tested.

3:5:1 3:8:1 10:15:1

2.5.3 Training and producing results

Using the *nntool*, each network (one for each output parameter and network architecture) was trained separately. Networks were trained for 200 iterations with cross-validation error limits of 20.

After training, new datasets similar to the ones used for training were generated and run through the ANN. The filtered data was plotted against the noisy data, same as how it was done with the AANN used on IPSEpro data.

2.6 ANN with data from Turbec

The Turbec measurements were also filtered through an ANN to check whether this could give better results.

Networks were trained with 9 inputs and one output. One network for each parameter (meaning each parameter was the single output in its own network). All parameters were first randomized and normalized for training the same way as was done with the AANN used on Turbec data.

This was done the same way as with the IPSEpro ANN model. But only one architecture was tested here.

Again, the *nntool* was used. Networks were trained for 1000 iterations with cross-validation error limits of 20.

After training, the same dataset used for training was filtered through the ANN. Filtered data was then plotted on top of the original noisy data, same way as was done with the AANN used on Turbec data.

3 Results

In this chapter the results from the AANN and ANN tests are described. Results (plots and tables) have not been included if they did not provide any interesting information, but some extra plots have been included in Appendix IV.

The chapter starts with a brief look at the results from the two theoretical AANN examples, before continuing with the results from the tests done on the IPSEpro and Turbec datasets.

3.1 Theoretical AANN examples

Here are the results from the two theoretical AANN examples introduced in chapter 2. Part of the reason they have been included is because they are easily visualized, as shall be seen.

3.1.1 2D example

Eight networks were tested, with the following number of iterations and MSE. All performance plots were deemed acceptable, and as can be seen in table 6, training was stopped by cross-validation error limits for every network trained (none of the networks made it to 1000 iterations).

Network	MSE	Iterations
2:3:1:3:2	0.027900	140
2:4:1:4:2	0.000390	224
2:5:1:5:2	0.000384	336
2:7:1:7:2	0.000396	245
2:10:1:10:2	0.000374	274
2:3:5:1:5:3:2	0.000304	349
2:3:7:1:7:3:1	0.001400	164
2:5:10:1:10:5:2	0.000312	294

Table 6: Mean square error and number of iterations for each network architecture trained with the 2D dataset.

From the plots of filtered and noisy data, the 2:5:10:1:10:5:2 network was deemed the best network. It gave the plots already shown in illustration 37. As was seen there, the individual parameters did not achieve noise reduction.

Plotting the two filtered parameters together on the other hand (here in illustration 41), shows that on a system level there is indeed noise reduction. If it is difficult to distinguish the two plots from each other in this illustration, it is basically illustration 38 laid on top of illustration 33.

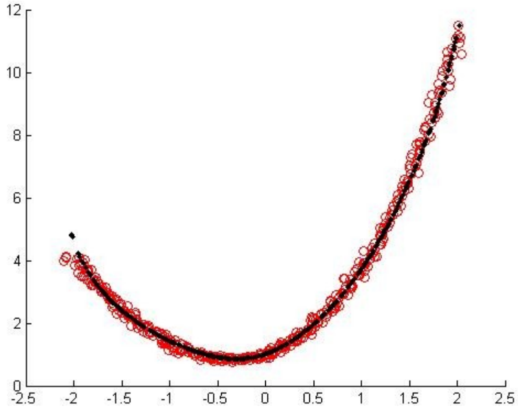


Illustration 41: Filtered x and y values plotted on top of the noisy x and y values. The data was filtered through the 2:5:10:1:10:5:2 network.

For this dataset, two extra hidden layers in the network architecture did provide better results than a corresponding network without the extra layers.

3.1.2 3D example

Eleven networks were tested in this example. Some of the networks did train up to 1000 iterations, but not all of them.

Network	MSE	Iterations
3:5:1:5:3	0.027900	441
3:7:1:7:3	0.039100	245
3:9:1:9:3	0.024500	771
3:5:2:5:3	0.001050	1000
3:7:2:7:3	0.000474	1000
3:9:2:9:3	0.000180	1000
3:5:7:1:7:5:3	0.025100	854
3:5:10:1:10:5:3	0.078600	186
3:5:7:2:7:5:3	0.002240	1000
3:5:10:2:10:5:3	0.000523	533
3:7:15:2:15:7:3	0.000088	1000

Table 7: Mean square error and number of iterations for each network architecture trained with the 3D dataset.

Judging by the plots made from data filtered through each network, the 3:7:15:2:15:7:3 network produced the best results.

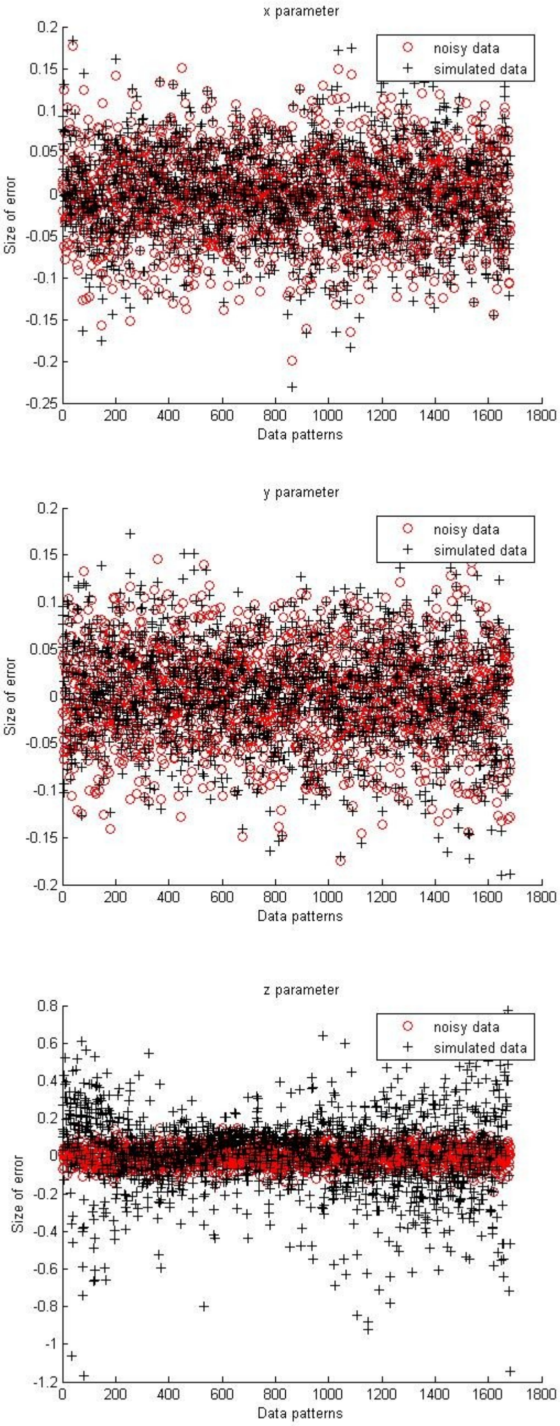


Illustration 42: The unfiltered noisy data compared to the filtered (simulated) data plotted for all three parameters. This is from the 3:7:15:2:15:7:3 network.

There is no indication of noise reduction in any of these three parameters in illustration 42. But if the filtered values are plotted together in a surface plot, also in this example there is some noise reduction on a system level. It is not so easy to see here in illustration 43, but the surface has been smoothed out with the filtered data.

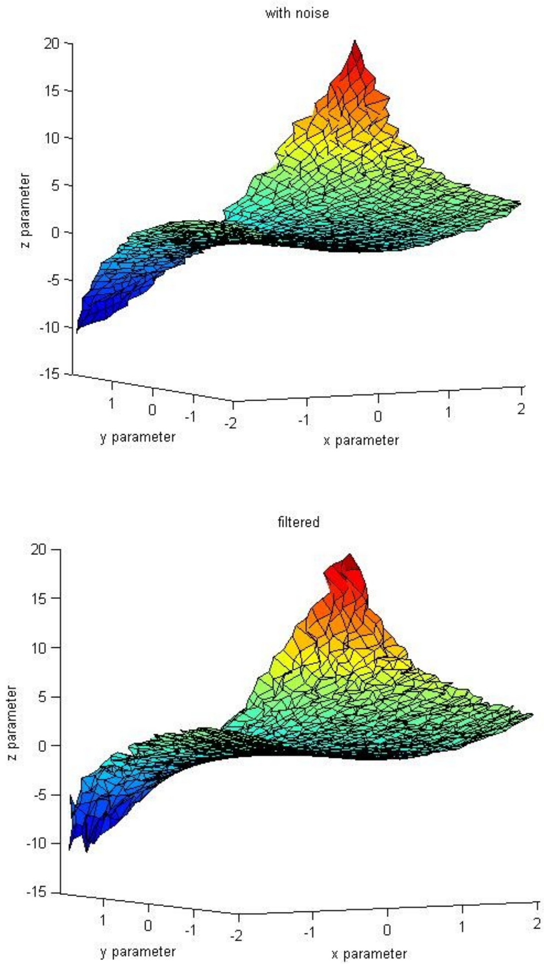


Illustration 43: Surface plots of the original noisy data and the filtered data. The 3:7:15:2:15:7:3 network was used for this plot.

Networks with only one neuron in the bottleneck layer produced noticeably worse results than networks with two neurons in the bottleneck layer.

Also for this dataset, two extra hidden layers in the network architecture did provide better results than a corresponding network without the extra layers.

3.2 AANN with data from IPSEpro

Results from training of AANN networks on the IPSEpro dataset are listed here. Mean square error decreases as the size of the bottleneck layer increases. This is expected, as less information is lost in compression.

Network	MSE	Iterations
10:12:3:12:10	0.01700	2000
10:12:5:12:10	0.01090	2000
10:12:7:12:10	0.00061	2000
10:15:3:15:10	0.01090	2000
10:15:5:15:10	0.00301	2000
10:15:7:15:10	0.00057	2000
10:20:3:20:10	0.01090	2000
10:20:5:20:10	0.00692	2000
10:20:7:20:10	0.00059	2000
11:20:3:20:11	0.01570	2000
10:12:15:5:15:12:10	0.00367	2000
10:15:20:5:20:15:10	0.00692	2000

Table 8: Mean square error and number of iterations for each network architecture trained with the IPSEpro dataset.

3.2.1 Optimal network architecture

Noise reduction was achieved on six of eleven parameters. That includes the parameter excluded from most of the networks trained; pressure out of turbine. The three parameters where there was no noise reduction were ambient humidity, temperature out of compressor and temperature out of turbine.

Five hidden layers in the AANN did not produce any better results than three hidden layers did with regards to noise reduction.

The 10:12:5:12:10 network produced the best noise reduction for those parameters where noise reduction was achieved.

The 11:20:3:20:11 network produced good noise reduction for the previously excluded pressure out of turbine parameter. Other than that it did not produce results much different from the 10:12:5:12:10 network.

Networks with a bottleneck layer of size 3 generally produced better results. This can be illustrated by a good example of overfitting in illustrations 44 and 45. The second plot, where the bottleneck layer has 5 neurons instead of 3, is overtrained.

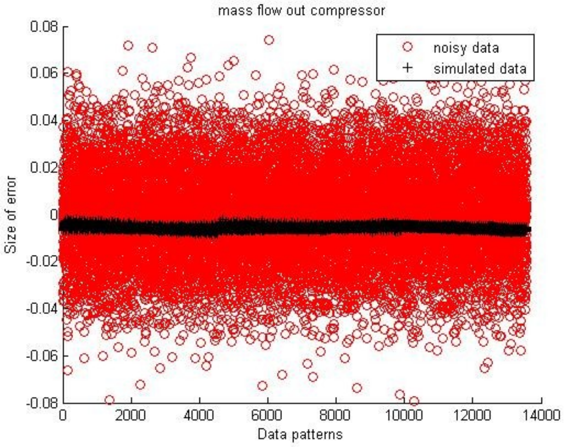


Illustration 44: Mass flow out compressor measurements filtered through 10:20:3:20:10 AANN. Simulated data means filtered data. An example of the general solution.

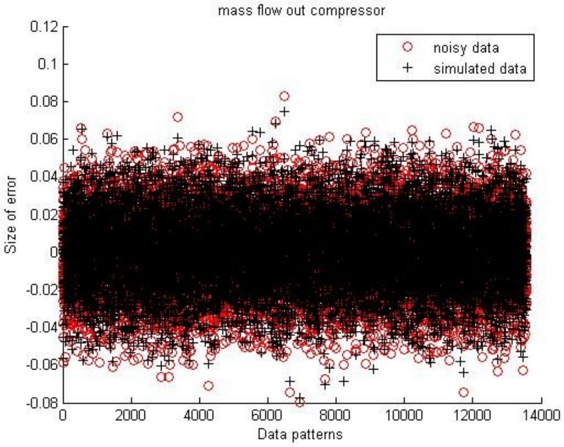


Illustration 45: Mass flow out compressor measurements filtered through 10:20:5:20:10 AANN. Simulated data means filtered data. An example of overfitting.

In order to rule out that the best network architecture (10:12:5:12:10) was not also subject to some overfitting, this network was trained again twice. Once with 1000, and once with 500 iterations. This did not improve upon the results.

3.2.2 Level of noise reduction

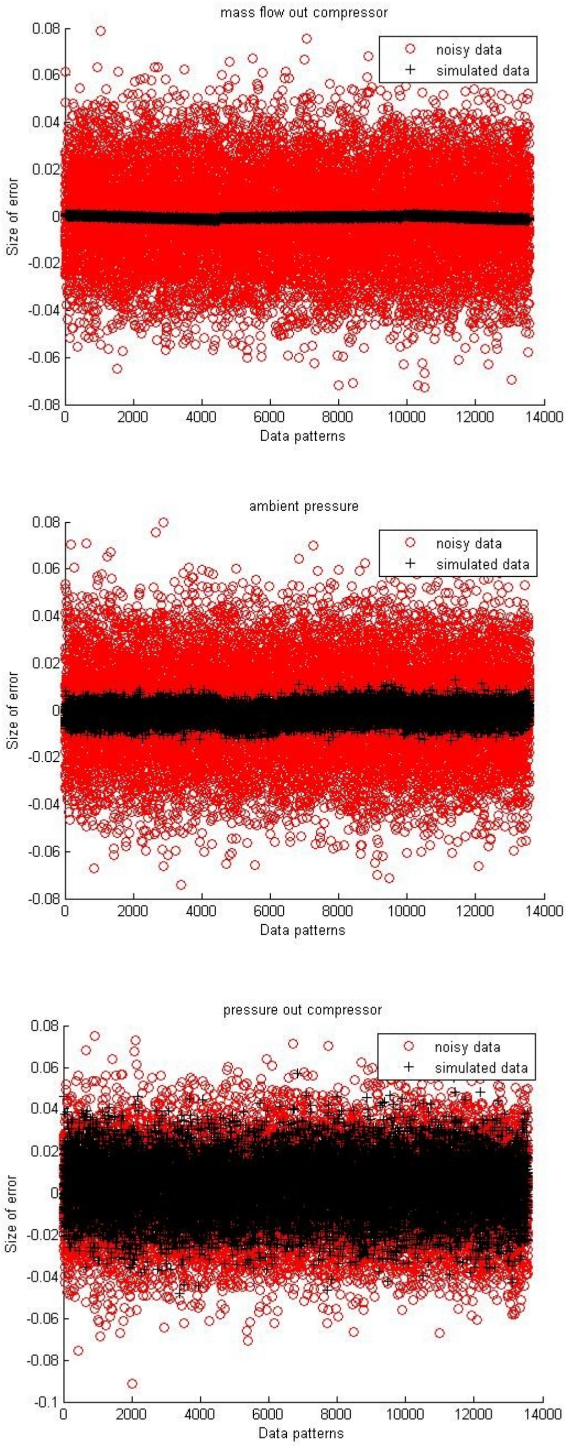


Illustration 46: Parameters filtered through 10:12:5:12:10 network. Simulated data means filtered data.

The seven parameters on which noise reduction was achieved in the 10:12:5:12:10 network, had different levels of noise reduction. This can be shown in illustration 46, which displays examples of high, medium and minimal noise reduction.

Parameter	Level of noise reduction
Ambient Pressure	medium
Ambient Temperature	minimal
Mass Flow out Compressor	high
Pressure out Compressor	minimal
Mass Flow Fuel	high

Table 9: Level of noise reduction for listed parameters filtered through the 10:12:5:12:10 network.

Comparing table 9 with table 2 in section 2.3.1, which lists the range of each parameter, it can be seen that parameters with a range less than 0.01 achieve a high level of noise reduction. Ambient pressure however has a range higher than 0.01, and only achieves a medium level noise reduction. Parameters which achieve minimal or no noise reduction have a range >> 1.

3.2.3 Noise reduction on noisy data with outliers

Because the results from the AANN noise filter was so unsuccessful, there was little point in adding outliers to the dataset. Checking for the effect of outliers on noise reduction when there is little to no noise reduction in the first place will not yield any results worth analyzing.

3.3 AANN with data from Turbec

Filtering the Turbec dataset with an AANN filter did not work. One exception was the gas pressure measurements filtered with the 10:12:3:12:10 network shown in illustration 40, but it was also the only exception. And it was just a part of the data plotted for that parameter. The whole plot can be seen in illustration 47 on the next page.

None of the other parameters in the Turbec dataset came close to anything resembling noise reduction when filtered through an AANN.

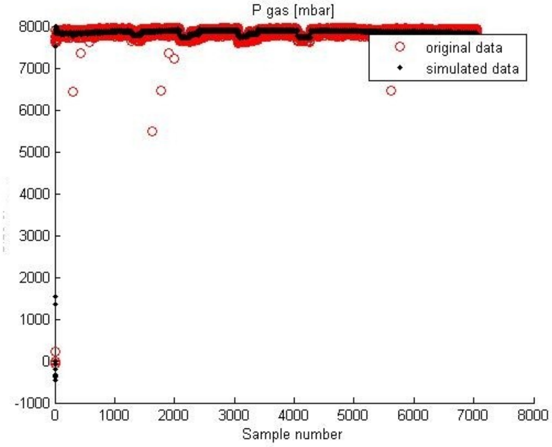


Illustration 47: Gas pressure measurements filtered through the 10:12:3:12:10 network. Simulated data is filtered data.

In table 10 here, it can be seen that each network was trained to 1000 iterations. It might be possible to train the network even further in order to lower the MSE, but the MSE is quite high here for a network that has already been trained to 1000 iterations. Most likely the MSE values will not become much lower than this.

Network	MSE	Iterations
10:12:2:12:10	0.00314	1000
10:12:3:12:10	0.00256	1000
10:12:5:12:10	0.00150	1000
10:15:3:15:10	0.00195	1000
10:15:5:15:10	0.00140	1000
10:12:15:3:15:12:10	0.00449	1000
10:12:15:5:15:12:10	0.00240	1000
10:15:20:3:20:15:10	0.00242	1000

Table 10: Mean square error and number of iterations for each network architecture tested on the Turbec dataset.

Having two extra hidden layers in the AANN architecture for this test did not make any noticeable difference for the gas pressure parameter.

3.4 ANN with data from IPSEpro

Here are presented some results from using ANN for noise filtering. They are not as extensive as with the AANN, but show that ANN also does work on some parameters.

3.4.1 Results for all parameters

Only three of the output parameters achieved noise reduction by use of ANN. Mass flow out compressor, mass flow fuel and pressure out of turbine. The same ones that achieved high level of noise reduction in the AANN noise filter. None of the other parameters achieved noise reduction by use of the ANN, and this is true for all three network architectures.

Network 10:15:1 might arguably be the best network architecture. This because the 3:5:1 and 3:8:1 networks had more problems finding correlations between parameters. An example of this can be demonstrated here with illustrations 48 and 49.

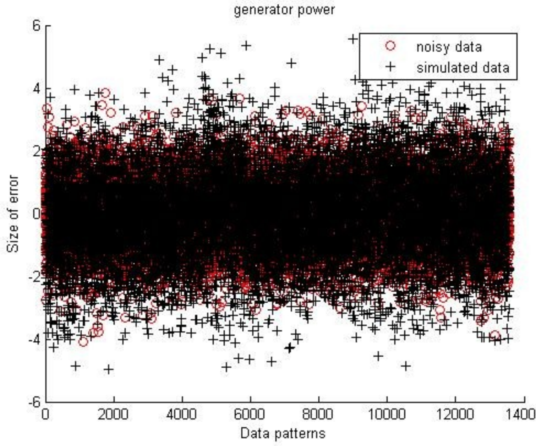


Illustration 48: Generator power measurements filtered through the 10:15:1 network. Simulated data means filtered data.

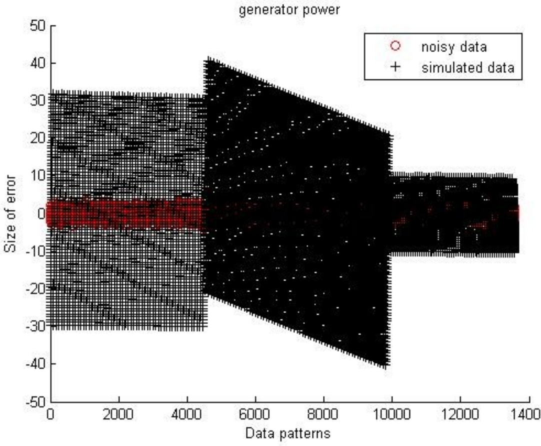


Illustration 49: Generator power measurements filtered through the 3:5:1 network. Simulated data means filtered data.

The 3:5:1 network trained for generator power (result plotted in illustration 49) has completely failed to pick up on any correlations between the three input parameters and the output parameter.

3.4.2 Level of noise reduction

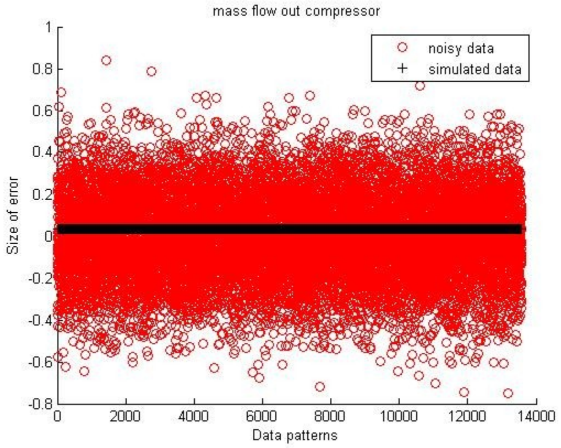


Illustration 50: Mass flow out compressor filtered through 3:5:1 ANN. Simulated data means filtered data.

The three parameters where noise reduction was achieved had a high level of noise reduction. Example of mass flow out compressor is plotted here in illustration 50.

3.5 ANN with data from Turbec

Much like the results of the AANN filter tested on the Turbec dataset, the results of the ANN filter did not work. Not even on the gas pressure parameter which had some positive results with the AANN filter would it work using the ANN filter.

4 Discussion

4.1 Shortcomings in methodology used

Several limitations have affected the results of the work in this thesis. Looking first at the data which was used to represent the Turbec T100 CHP, it is not going to be a completely accurate model. Thermodynamic tables, theoretical equations and a simplified model will deviate from reality.

Further, the author should emphasize that he does not know exactly how the mathematics behind the training of neural networks by use of the scaled conjugate gradient algorithm works. This does leave results open to misunderstanding. It is also difficult to say anything with certainty regarding overfitting and correlations between parameters.

It is not clear to what extent the continuously fluctuating sensor measurements from the Turbec could be considered noise, as opposed to the gas turbine merely correcting itself continuously in order to maintain a constant power output.

The sensor readings from the Turbec had an interval of 51 seconds between measurements. This seems to the author like it undermines the idea of sensor noise, as the state of a gas turbine can change a great deal over the course of 51 seconds.

Another problem that should be mentioned is how the sensors inside a gas turbine will inevitably degrade over time. This in itself could also have an effect on any AANN noise filtering, something which was not explored in this thesis.

4.2 Evaluation of results

Although there was noise reduction (and in some cases quite a lot of noise reduction) achieved through the use of an AANN filter, the results are too inconclusive. In theory the methodology could work, but there are several conditions that must be met first.

The author believes that measurements used for training and monitoring of the gas turbine need to be of parameters which are known to correlate to each other. This is essential in order for a network to focus on these correlations during training, and discard random noise. Such correlations could be ensured by for example theoretical thermodynamic equations.

Redundancy also seems to play a role. Looking at the parameters which did achieve a high level of noise reduction in chapter 3, they all had a very low range over which they changed. This may have made it easier for the network to discard random noise from these parameters.

Regarding the results gotten from the tests run on the Turbec measurement data, the author feels they should all be discarded. This simply because the data used in the tests is not good enough, for reasons which are mentioned in section 4.1.

The two theoretical AANN examples did reveal a trend which speaks against using AANN for noise reduction on individual sensors in a gas turbine (at least against the methodology used in this thesis). While the individual parameters achieved no noise reduction, the system itself (consisting of all parameters put together) did achieve noise reduction. So the filter does not work directly towards filtering individual parameters. It filters them in combination, as if they were one.

The author speculates that the same principle should also apply to a system which consists of more than three parameters, but it is (as far as the author can see) not possible to check this, because there is no way to visualize it.

Using an ANN as a noise filter was less successful than using an AANN. But the use of ANN as a noise filter in this thesis was devoted limited attention, and the mere fact that it did indeed work on some parameters means that it could be worth looking into more deeply.

4.3 Ideas for future work

It might be interesting to test an AANN noise filter on more relevant measurements taken from a Turbec. With a set of measurements taken at short time intervals, from parameters that are known to correlate with each other, it is not unreasonable to assume some level of noise reduction could be achieved. Although there is a good chance it will only work on some of the parameters.

List of tables and illustrations

Table 1 -page 23- Parameters included in IPSEpro data.
Table 2 -page 23- Range for each individual parameter in the IPSEpro data.
Table 3 -page 23- Covariance matrix for IPSEpro data.
Table 4 -page 24- Noise added to IPSEpro data.
Table 5 -page 25- The ten parameters included in the measurements taken from the Turbec.
Table 6 -page 27- Results from AANN on 2D example.
Table 7 -page 27- Results from AANN on 3D example.
Table 8 -page 29- Results from AANN on IPSEpro data.
Table 9 -page 30- Level of noise reduction on selected parameters filtered through AANN.
Table 10-page31- Results from AANN on Turbec data.

Illustration 1 -page 5- Open simple cycle gas turbine.
Illustration 2 -page 5- T-S diagram for open simple cycle gas turbine.
Illustration 3 -page 6- T-S diagram for different open simple cycle gas turbines.
Illustration 4 -page 6- Open simple cycle gas turbine with recuperator.
Illustration 5 -page 6- T-S diagram for open simple cycle gas turbine with recuperator.
Illustration 6 -page 7- Centrifugal compressor.
Illustration 7 -page 7- T-S diagram for centrifugal compressor.
Illustration 8 -page 7- Axial compressor.
Illustration 9 -page 7- T-S diagram for an axial compressor.
Illustration 10 -page 8- Surge/choke diagram.
Illustration 11 -page 8- Dependence of emissions on air/fuel ratio.
Illustration 12 -page 9- Annular and can-type combustors.
Illustration 13 -page 9- One stage of an axial turbine.
Illustration 14 -page 10- The Turbec T100 CHP.
Illustration 15 -page 10- Modified Turbec T100 CHP.
Illustration 16 -page 11- Biological neural network.
Illustration 17 -page 11- Artificial neural network.
Illustration 18 -page 12- The perceptron.
Illustration 19 -page 12- Threshold function.
Illustration 20 -page 13- Differentiable transfer functions.
Illustration 21 -page 13- Multilayer perceptron.
Illustration 22 -page 14- Plot of mean square error.
Illustration 23 -page 14- How gradient method can end up finding a local minimum.
Illustration 24 -page 15- The principle behind training algorithms with known target.
Illustration 25 -page 15- Example of regression.
Illustration 26 -page 15- Example of overfitting.
Illustration 27 -page 16- Example of randomization.
Illustration 28 -page 16- Performance curve with training set, cross-validation set and test set.
Illustration 29 -page 16- AANN architecture.
Illustration 30 -page 17- Normal distribution.
Illustration 31 -page 17- Data filtered by exponential smoothing.
Illustration 32 -page 17- Data with trend shift filtered by exponential smoothing.
Illustration 33 -page 19- 2D AANN example.
Illustration 34 -page 20- Performance curve for 2D AANN example.
Illustration 35 -page 21- Regression plot for 2D AANN example.
Illustration 36 -page 21- Gradient curve and cross-validation diagram for 2D AANN example.
Illustration 37 -page 22- Results plotted for parameters in 2D AANN example.
Illustration 38 -page 22- Filtered 2D AANN example.
Illustration 39 -page 22- 3D AANN example.
Illustration 40 -page 25- Gas pressure measurements filtered through AANN.
Illustration 41 -page 27- Filtered 2D AANN example.
Illustration 42 -page 28- Results plotted for parameters in 3D AANN example.
Illustration 43 -page 28- Surface plots for unfiltered and filtered 3D AANN example.
Illustration 44 -page 29- Mass flow out compressor measurements filtered through 10:20:3:20:10 network.
Illustration 45 -page 29- Mass flow out compressor measurements filtered through 10:20:5:20:10 network.
Illustration 46 -page 30- High, medium and minimal level of noise reduction.
Illustration 47 -page 31- Gas pressure measurements filtered through AANN.
Illustration 48 -page 31- Generator power measurements filtered through 10:15:1 network.
Illustration 49 -page 31- Generator power measurements filtered through 3:5:1 network.
Illustration 50 -page 32- Mass flow out compressor measurements filtered through 3:5:1 network.

References

[1] H.I.H. Saravanamuttoo, G.F.C. Rogers, H. Cohen, P.V. Straznicky, **Gas turbine theory**, 6th edition, Pearson Education Limited, 2009

[2] P.P. Walsh, P. Fletcher, **Gas Turbine Performance**, 2nd edition, Blackwell Publishing, 2004

[3] **T100 microturbine CHP system**, Technical description, version 4.0, Turbec AB

[4] J. C. Principe, N. R. Euliano, W. C. Lefebvre, **Neural and Adaptive Systems Fundamentals Through Simulations**, John Wiley and Sons, Inc. 2000

[5] H. Demuth, M. Beale, M. Hagan, **MATLAB Neural Network Toolbox^{TM} 6 User's Guide**, The MathWorks, Inc. September 2009, http://www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf

[6] C. M. Bishop, **Pattern Recognition and Machine learning**, Springer-Verlag, 2007

[7] G. Dreyfus, **Neural Networks Methodology and Applications**, English edition, Springer-Verlag, 2005

[8] C. G. Looney, **Pattern Recognition Using Neural Networks**, Oxford University Press, Inc. 1997

[9] M. A. Kramer, **Nonlinear Principal Component Analysis Using Autoassociative Neural Networks**, AIChE Journal, Vol. 73, No. 2, pages 233 – 243, February 1991

[10] M. A. Kramer, **Autoassociative Neural Networks**, Computers Chemical Engineering, Vol 16, No. 4, pages 313–328, 1992

[11] R. Ganguli, **Data Rectification and Detection of Trend Shifts in Jet Engine Gas Path Measurements Using Median Filters and Fuzzy Logic**, ASME TURBO EXPO, June 4–7, New Orleans, Louisiana, 2001

[12] D. L. Chester, **Why two hidden layers are better than one**, International joint conference on neural networks, Washington DC, January 15–19, 1990

Appendix

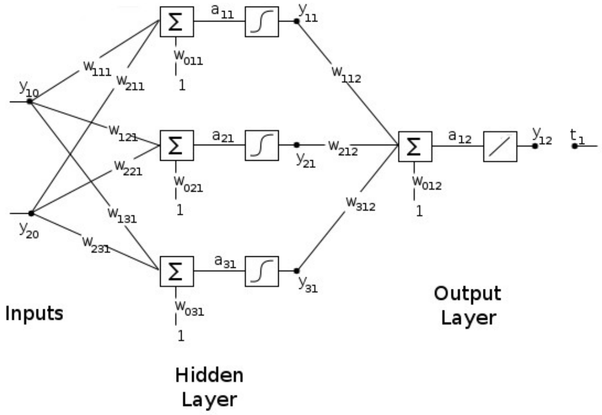
I Backpropagation example

An example of backpropagation calculation was here constructed to provide a more in depth understanding of how the process works. The mathematics are pretty straight forward, but extensive.

In this example there are two input vectors (there is also a bias, b), and one target vector. These will be used to train the network using the backpropagation algorithm.

$$\begin{aligned} \mathbf{b} &= [1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1] \\ \mathbf{y_{10}} &= [1\ \ 2\ \ 4\ \ 6\ \ 7\ \ 10\ \ 11\ \ 14\ \ 14\ \ 15] \\ \mathbf{y_{20}} &= [3\ \ 3.2\ \ 2.9\ \ 3.3\ \ 3.8\ \ 4.1\ \ 4.2\ \ 4.8\ \ 4.7\ \ 4.9] \\ \mathbf{t_1} &= [12.2\ \ 12.1\ \ 11.2\ \ 10.0\ \ 9.8\ \ 7.3\ \ 6.9\ \ 5.8\ \ 5.9\ \ 5.7] \end{aligned}$$

$$\boldsymbol{y_0} = [\boldsymbol{b} \quad \boldsymbol{y_{10}} \quad \boldsymbol{y_{20}}]$$



The feed forward ANN of the backpropagation example. A 2:3:1 network with tansig transfer functions in the hidden layer, and a purelin transfer function in the output layer.

The goal is to update the network weights, using the backpropagation algorithm from section 1.3.5.

$$w_{hij}(k+1) = w_{hij}(k) - \eta \frac{\partial g_j(k)}{\partial w_{hij}(k)} \qquad \text{(I.1)}$$

$w_{hij}(k)$ is a weight connected to two neurons: the i^th^ neuron in the j^th^ layer, and the h^th^ neuron in the layer before the j^th^ layer (see the illustration above). In this case the hidden layer (which has 3 neurons) is the 1st layer, and the output layer is the 2nd layer. ŋ is the step size (a constant which we will set to 0.1), and $g_j(k)$ is the cost function (also known as criterion).

The first part of the process is to initialize the weights with random numbers. This is done to prevent the training getting stuck in a local minimum.

$w_{011} = 0.27$
$w_{111} = -0.41$
$w_{211} = -0.13$
$w_{021} = 0.09$
$w_{121} = 0.74$
$w_{221} = -0.21$
$w_{031} = 0.03$
$w_{131} = -0.88$
$w_{231} = -0.79$

$w_{012} = 0.12$
$w_{112} = 0.54$
$w_{212} = -0.91$
$w_{312} = 0.23$

$$\boldsymbol{w_1} = \begin{bmatrix} w_{011} & w_{021} & w_{031} \\ w_{111} & w_{121} & w_{131} \\ w_{211} & w_{221} & w_{231} \end{bmatrix} \qquad \boldsymbol{w_2} = \begin{bmatrix} w_{012} \\ w_{112} \\ w_{212} \\ w_{312} \end{bmatrix}$$

Calculating the network output for the first iteration (k denotes the current iteration), is then next. The argument to, and output of, each individual neuron will need to be found.

$$\boldsymbol{a_1} = \boldsymbol{y_0} \boldsymbol{w_1} = [\boldsymbol{a_{11}} \quad \boldsymbol{a_{21}} \quad \boldsymbol{a_{31}}]$$

$$\boldsymbol{a_1} = \begin{bmatrix} -0.530 & 0.200 & -3.220 \\ -0.966 & 0.898 & -4.258 \\ -1.747 & 2.441 & -5.781 \\ -2.619 & 3.837 & -7.857 \\ -3.094 & 4.472 & -9.132 \\ -4.363 & 6.629 & -12.009 \\ -4.786 & 7.348 & -12.968 \\ -6.094 & 9.442 & -16.082 \\ -6.081 & 9.463 & -16.003 \\ -6.517 & 10.161 & -17.041 \end{bmatrix}$$

$$\tanh(\boldsymbol{a}_1)=\begin{bmatrix}\boldsymbol{y}_{11} & \boldsymbol{y}_{21} & \boldsymbol{y}_{31}\end{bmatrix}$$

$$\boldsymbol{y}_1=\begin{bmatrix}\boldsymbol{b} & \boldsymbol{y}_{11} & \boldsymbol{y}_{21} & \boldsymbol{y}_{31}\end{bmatrix}$$

$$\boldsymbol{y}_1=\begin{bmatrix}1.000 & -0.485 & 0.197 & -0.996\\ 1.000 & -0.746 & 0.715 & -0.999\\ 1.000 & -0.941 & 0.984 & -0.999\\ 1.000 & -0.989 & 0.999 & -1.000\\ 1.000 & -0.995 & 0.999 & -1.000\\ 1.000 & -0.999 & 1.000 & -1.000\\ 1.000 & -0.999 & 1.000 & -1.000\\ 1.000 & -0.999 & 1.000 & -1.000\\ 1.000 & -0.999 & 1.000 & -1.000\end{bmatrix}$$

$$\boldsymbol{a}_2=\boldsymbol{a}_{12}=\boldsymbol{y}_1\boldsymbol{w}_2$$

$$\boldsymbol{y}_2=\boldsymbol{y}_{12}=purelin(\boldsymbol{a}_{12})=\boldsymbol{a}_{12}$$

$$\boldsymbol{y}_2=\begin{bmatrix}-0.550\\ -1.164\\ -1.514\\ -1.553\\ -1.557\\ -1.559\\ -1.559\\ -1.559\\ -1.559\\ -1.560\end{bmatrix}$$

Having calculated these arguments and outputs, the gradient for each weight can then be calculated. This is done through partial differentiation, and in order to find this differential equation, some expressions will have to be defined.

Cost function, or criterion if you will, is calculated for each layer. N is here the number of samples in the training set (**t**, **b**, $\mathbf{y}_{10}$ and $\mathbf{y}_{20}$), which in this example is 10.

$$g_j=\frac{1}{2\mathrm{N}}\sum_{p=1}^{N}\sum_{i=1}e_{ijp}^2 \qquad \text{(I.2)}$$

The error for the ith neuron in the jth layer, if that layer is the output layer is:

$$e_{ij}=t_h-y_{hj} \qquad \text{(I.3A)}$$

Otherwise, the error for the ith neuron in the jth layer is (with this particular function, use the average of y_{hj} for all N samples):

$$e_{ij}=y_{hj}(k)[w_{hij}(k+1)-w_{hij}(k)] \qquad \text{(I.3B)}$$

Outputs for the jth layer are calculated by putting the arguments for the jth layer into the transfer function, *f*. In our example, *f* is *tanh* for the 1st (hidden) layer, and *purelin* for the 2nd (output) layer.

$$y_{hj}=f(a_{ij}) \qquad \text{(I.4)}$$

Arguments for the jth layer are (remember b_{ij} = 1):

$$a_{ij}=b_{ij}w_{0\mathrm{ij}}+\sum_h w_{hij}y_{h(j-1)} \qquad \text{(I.5)}$$

And so, finally an expression for the gradient of weight $w_{hij}(k)$ can be found, using the delta rule of differentiation.

$$\frac{\partial g_j}{\partial w_{hij}}=\frac{\partial g_j}{\partial e_{ij}}\frac{\partial e_{ij}}{\partial y_{hj}}\frac{\partial y_{hj}}{\partial a_{ij}}\frac{\partial a_{ij}}{\partial w_{hij}} \qquad \text{(I.6)}$$

Each of these partial derivatives have to be calculated for each individual weight. Starting with w_{012}, these become:

$$\frac{\partial a_{12}}{\partial w_{012}}=1 \qquad \text{(see equation I.5)}$$

$$\frac{\partial y_{12}}{\partial a_{12}}=1 \qquad \text{(see equation I.4)}$$

$$\frac{\partial e_{12}}{\partial y_{12}} = -1 \qquad \text{(see equation I.3A)}$$

$$\frac{\partial g_2}{\partial e_{12}} = \frac{1}{10}\sum^{10} e_{12} \qquad \text{(see equation I.2)}$$

The gradient for weight w_{012} is then:

$$\frac{\partial g_2}{\partial w_{012}} = -\frac{1}{10}\sum^{10}(t - y_{12}) = -101.04$$

w_{012} can then be updated using equation I.1 (remember ŋ is set to 0.1):

$$w_{012}(2) = w_{012}(1) - \eta \frac{\partial g_2(1)}{\partial w_{012}(1)} = 10.21$$

Every weight in layer 2 needs to be updated before any of the weights in layer 1 can be updated. This is because all errors in layer 2 need to be combined in order to calculate the gradients in that layer (see equations I.3B and I.2).

Whenever a partial derivative is calculated equal to an output, such as this example with w_{112}, use the average value of y_{11} (**$\mathbf{y_{11}}$** contains 10 samples for each iteration) to calculate the value of the gradient.

$$\frac{\partial a_{12}}{\partial w_{112}} = y_{11}$$

The differential for the *purelin* transfer function is simple (it is 1), but the differential for the *tanh* transfer function is more difficult. Consider this example with w_{111}.

$$\frac{\partial y_{11}}{\partial a_{11}} = \frac{1}{\cosh^2(a_{11})} = \frac{4}{[\exp(a_{11}) + \exp(-a_{11})]^2}$$

Again, use the average value of a_{11} to calculate the value of the gradient.

After all weights have been updated, arguments and outputs for the 2nd iteration should be calculated, and the weights can be updated again.

II Various MATLAB code used

makebottle - Calling this function will create an autoassociative neural network. Thanks to Doug Hundley who posted the original version of this code (meant to perform nonlinear principle component analysis) on the Mathworks forums in 2001. Some small alterations have been made here in order to make it usable for AANN noise reduction. This was used on the R2008a version of MATLAB.

```
function net=makebottle(X,v)
%FUNCTION NET=MAKEBOTTLE(X,V)
%Returns a bottleneck network ready for initialization
%and training. This constructs it in the standard approach,
%where the network has 5 layers,
%
%input-mapping-bottleneck-demapping-output
%
%The input:
%X=data set.  Input as number of points x dimension
%v=vector of 2 numbers for the sizes of the encoding and
%bottleneck layers.
%
%EXAMPLE: Data set X is three dimensions, use 2 nodes in
%the bottleneck layer, and 5 nodes in the other two hidden
%layers so the ending network is 3-5-2-5-3
%
%net=makebottle(X,[5,2]);

net=network;
net.numInputs= 1;
net.numLayers= 4;
net.biasConnect= [1;1;1;1];
net.inputConnect(1)= 1;
net.layerConnect(2,1)= 1;
net.layerConnect(3,2)= 1;
net.layerConnect(4,3)= 1;
net.outputConnect(4)= 1;

[numpts,dim]= size(X);
M= minmax(X');
net.inputs{1}.size= dim;
net.inputs{1}.range= M;
net.layers{1}.size= v(1);
net.layers{3}.size= v(1);
net.layers{1}.transferFcn= 'tansig';
net.layers{3}.transferFcn= 'tansig';
net.layers{2}.size= v(2);
net.layers{2}.transferFcn= 'tansig';
net.layers{4}.size= dim;
net.layers{4}.transferFcn= 'purelin';
net.layers{1}.initFcn= 'initnw';
net.layers{2}.initFcn= 'initnw';
net.layers{3}.initFcn= 'initnw';
net.layers{4}.initFcn= 'initnw';

%Initialize the functions for the network
net.initFcn= 'initlay';
net.performFcn= 'mse';
net.trainFcn= 'trainscg';
net.gradientFcn = 'calcgrad';
net.plotFcns = {'plotperform','plotregression','plottrainstate'};
net.divideFcn = 'divideblock';
```

randomize – Using the randperm command will generate a random sequence. A vector **x**, with 25 samples can be randomized with the following code:

```
shuffle = randperm(length(x));
x_randomized = x(shuffle);
```

The vector **x_randomized** will then contain the samples from vector x in a random sequence.

normalize – The mapminmax command can be used to normalize vectors and matrices. A matrix **X**, containing numbers between ± ∞, can be converted so that the numbers range between ± 1.

```
[Xn, ps] = mapminmax(X, -1, 1);
```

The matrix **Xn** then contains the numbers converted to range between -1 and 1. The mapminmax command can also be used to reverse the effect.

```
X = mapminmax('reverse', Xn, ps);
```

Here **ps** must be the same as it was for the initial normalization

white noise – Random gaussian numbers can be generated with the *randn* command. White noise can be added to the vector **x** with a chosen size of the noise.

```
x_noise = x + 0.5*randn(size(x));
```

This will make **x_noise** a vector almost identical to **x**, only with noise added to it.

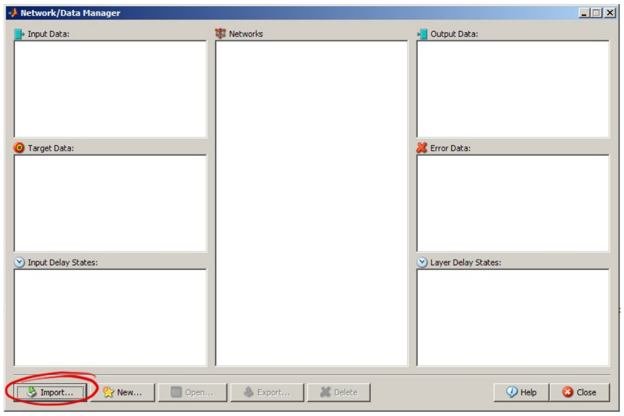
covariance matrix – The command cov will generate a covariance matrix. Matrix **X** contains nxm samples (m samples of n parameters). The following code will generate a nxn covariance matrix.

```
covariance_matrix = cov(X);
```

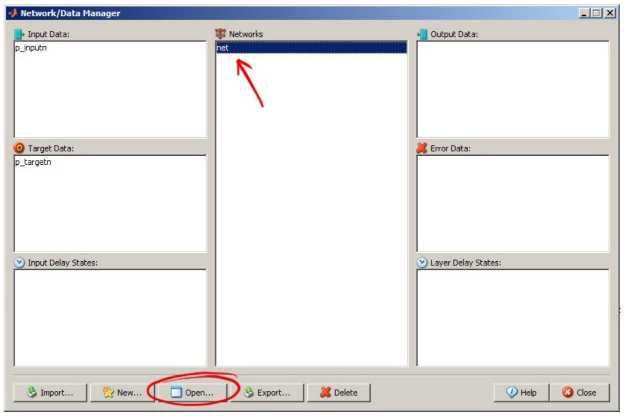
III The MATLAB nntool

This small tutorial was added to show how the *nntool* was used for training the artificial neural networks in this thesis. It is meant to help people who have not used neural networks before, and also serve as a reference should the author ever need to use neural networks in his future work.

To open the *nntool*, type nntool in the MATLAB command line. The following GUI should appear.



Press the *import* button, and import the network (the author created this with the makebottle function before calling the *nntool*), as well as the input and output matrices. A custom network could also be made by pressing the *new* button and building a network that way, but it is too limited to create an AANN.

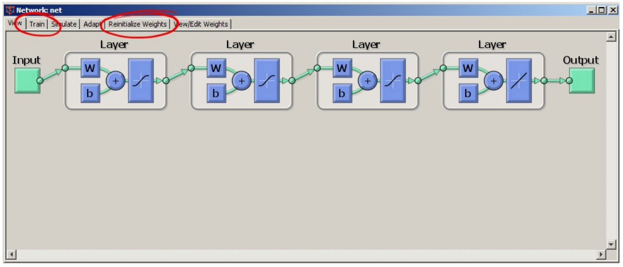


Input Data: p_inputn

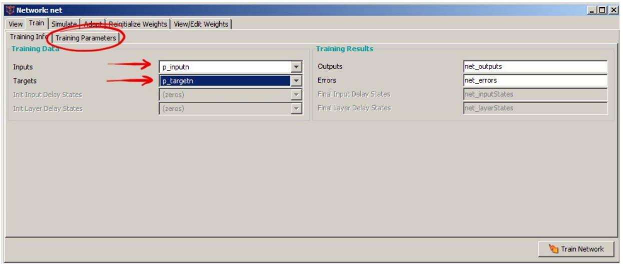
Networks: net

Target Data: p_targetn

After both inputs, outputs and network have been imported, select the network and press the *Open...* button.



This will open a new GUI for the network. The first thing to do here is to open the *Reinitialize Weights* tab, and there pressing the *initialize weights* button. This resets the weights to random values before training, which is important in order to avoid the training algorithm getting stuck in a local minimum. Notice also that one can look at each individual weight by pressing the *View/Edit Weights* tab. After initializing the weights, press the *train* tab.



Training Data

Inputs: p_inputn

Targets: p_targetn

Init Input Delay States: (zeros)

Init Layer Delay States: (zeros)

Training Results

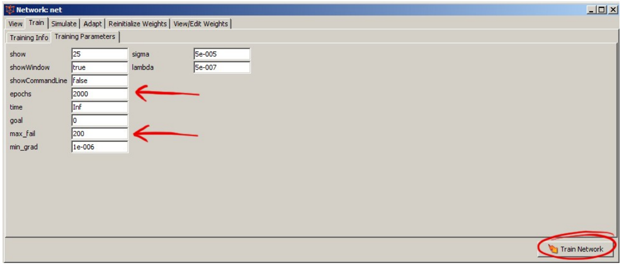
Outputs: net_outputs

Errors: net_errors

Final Input Delay States: net_inputStates

Final Layer Delay States: net_layerStates

Select the imported inputs and outputs, and then press the *Training Parameters* tab.



show: 25

showWindow: true

showCommandLine: false

epochs: 2000

time: Inf

goal: 0

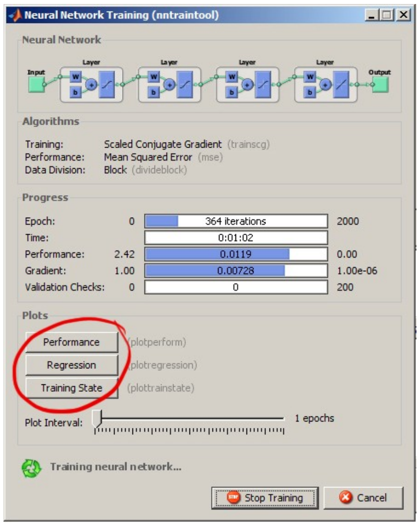
max_fail: 200

min_grad: 1e-006

sigma: 5e-005

lambda: 5e-007

Here one can alter some of the conditions for training. Setting a specific number of epochs decides how many iterations the training algorithm should do before stopping. The number of max_fail decides how many cross-validation deviations the training algorithm will accept before stopping training. After setting the desired training parameters, press the *Train Network* button.



Training: Scaled Conjugate Gradient (trainscg)

Performance: Mean Squared Error (mse)

Data Division: Block (divideblock)

Epoch: 0 — 364 iterations — 2000

Time: 0:01:02

Performance: 2.42 — 0.0119 — 0.00

Gradient: 1.00 — 0.00728 — 1.00e-06

Validation Checks: 0 — 0 — 200

Performance (plotperform)

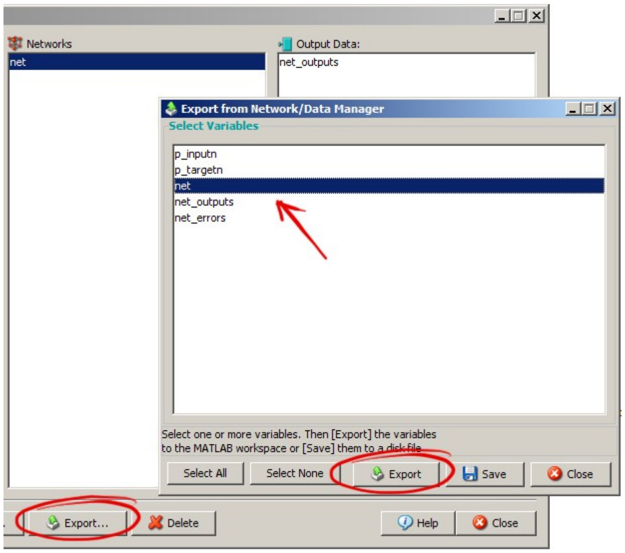
Regression (plotregression)

Training State (plottrainstate)

Plot Interval: 1 epochs

Training neural network...

A new window will show up (unless you altered *showWindow* = *false* in the training parameters). Here the training process can be monitored. One progress bar counts down the number of iterations, another shows the size of the MSE (called Performance here, but Performance could mean something else if a performance criterion different from MSE has been selected). There is also a bar that shows the number of cross-validation deviations. After training is finished (or during for that matter), plots can be made of the performance curve, regression curve and training state, by pressing the respective buttons. If the network designer is satisfied with the training, this window and the network window can now be closed.



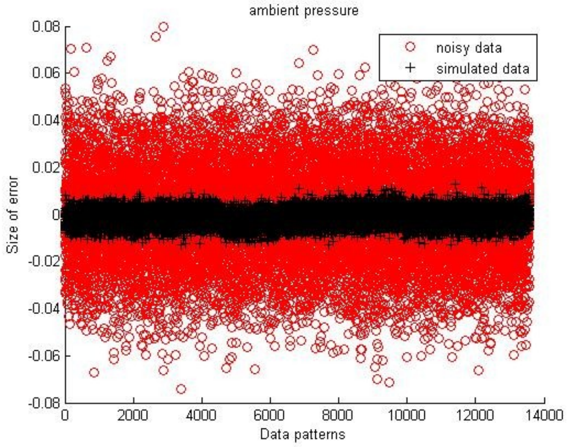
p_inputn
p_targetn
net
net_outputs
net_errors

Select one or more variables. Then [Export] the variables to the MATLAB workspace or [Save] them to a disk file.

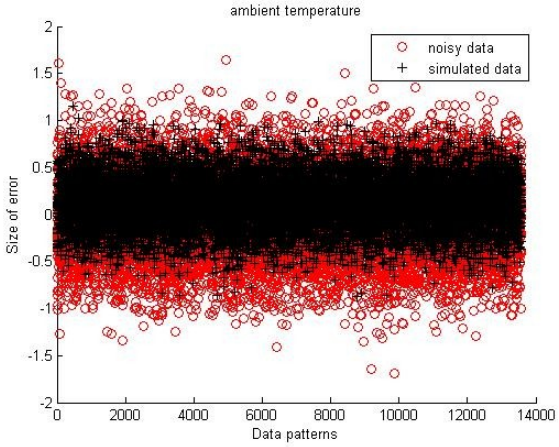
What remains then is to export the newly trained network to the MATLAB workspace where it can be subjected to simulation with new datasets. This is done by first selecting the network and pressing the *Export...* button, and then selecting the network again and pressing another *Export* button.

IV Additional illustrations

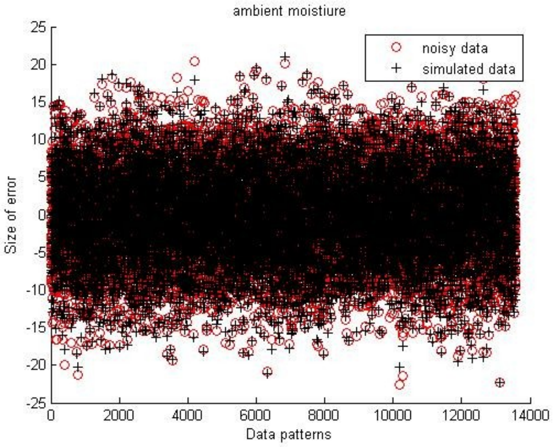
Here are included plots of noise reduction for all ten parameters in the 10:12:5:12:10 network which was used to filter the noisy IPSEpro dataset.



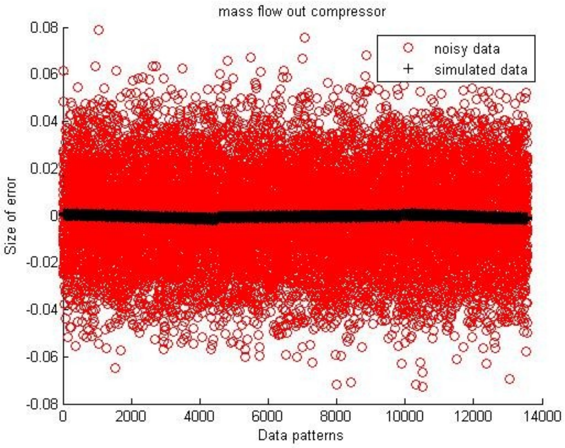
Ambient pressure filtered through 10:12:5:12:10 network. Simulated data means filtered data.



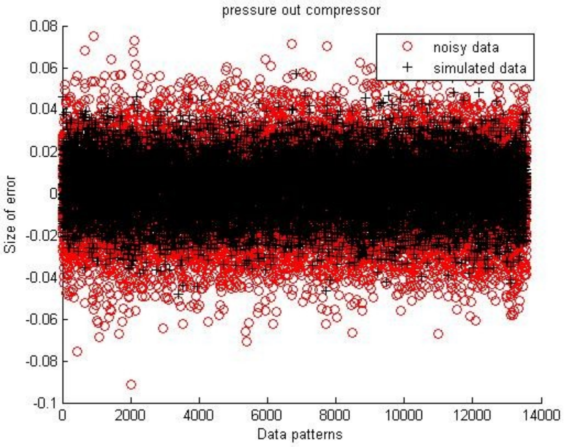
Ambient temperature filtered through 10:12:5:12:10 network. Simulated data means filtered data.



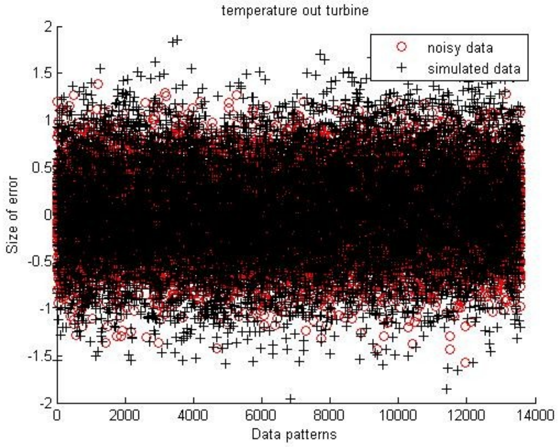
Ambient humidity filtered through 10:12:5:12:10 network. Simulated data means filtered data.



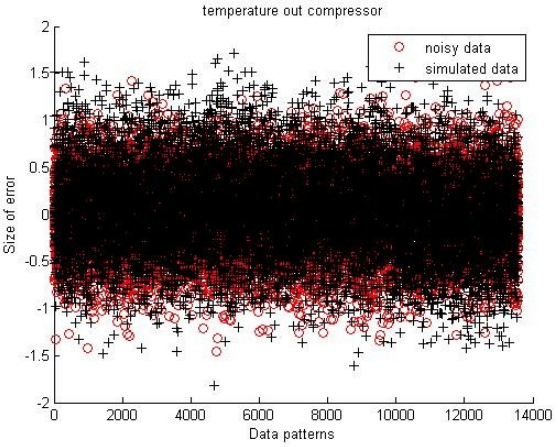
Mass flow out compressor filtered through 10:12:5:12:10 network. Simulated data means filtered data.



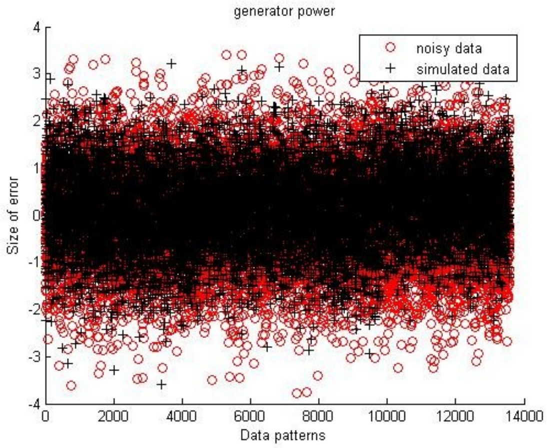
Pressure out compressor filtered through 10:12:5:12:10 network. Simulated data means filtered data.



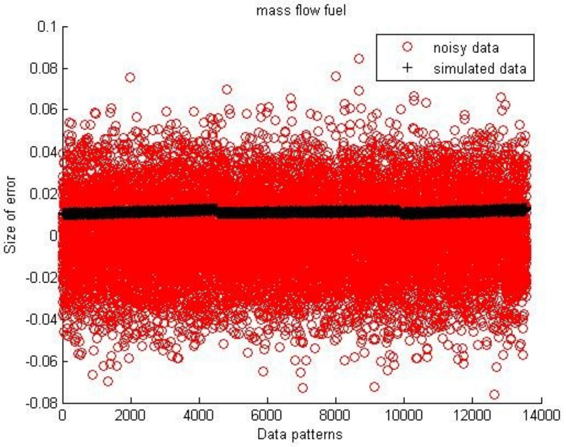
Temperature out turbine filtered through 10:12:5:12:10 network. Simulated data means filtered data.



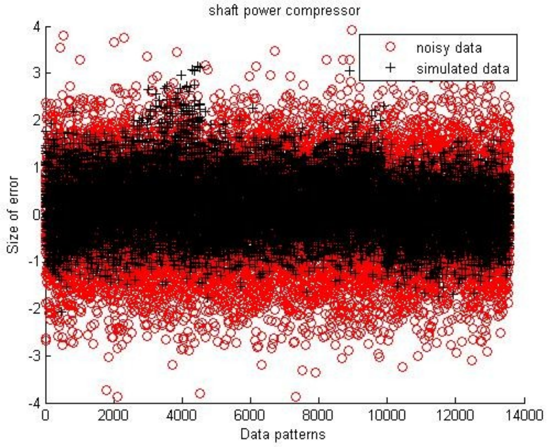
Temperature out compressor filtered through 10:12:5:12:10 network. Simulated data means filtered data.



Generator power filtered through 10:12:5:12:10 network. Simulated data means filtered data.



Mass flow fuel filtered through 10:12:5:12:10 network. Simulated data means filtered data.



Shaft power compressor filtered through 10:12:5:12:10 network. Simulated data means filtered data.