Bauhaus-University Weimar
Faculty of Media

Single Layer Artificial Neural Networks in Home Automation

Supervisor:
PD Dr. Andreas Jakoby
Submitted by:
Arne Osterthun

Preface

This thesis was submitted for one of the project modules required for the bachelor degree in computer science and media at Bauhaus University Weimar and was supervised by
PD Dr. Andreas Jakoby. The idea for this thesis was inspired by the dispute with the enera project and general interest in machine learning with artificial neural networks. This project investigates the capabilities of artificial neural networks in home automation.

Contents

1 Introduction

1.1 Current State of Home Automation Technology

Despite the fact home automation is not that new of a research area, the industry just recently started to create a marked for it in a noteworthy scale. However their where a number of products available for a long time, the technology was not advanced enough to really gain exeptence. Nevertheless, todays home automation technology is getting much more usable and consequently many companies are trying to gain market share. Sadly most companies are just trying to to gain a bigger market share by developing products that conform to the current state of research. Subsequently their is a lack of innovation and team work in the market. In result of this fast marked expansion most developers missed to maintain a standard to improve system compatibility. Thats also why most companies have there one proprietary software that stops free developers from improving and innovating home automation in an open source way. openHab is probably one of the biggest open source platforms currently available on the market. Unlike other product manufacturers Loxone is also providing a big online documentary. Also HomeKit is providing a software standard for app developers and accessory manufacturers. Additionally their is Homematic which provides an XML API for developers to use.

1.2 What to Expect from Home Automation

Currently three general use cases of home automation technology become apparent. As a matter of fact reduction of power wastage, enhancing the quality of home owners lives and supporting elderly and disabled persons [1] are the most researched use cases for home automation. Specifically research in the field of Ambient Assistant Living (AAL) is generating some insights to realizing home automation with artificial intelligence and big data. Especially Ambient Intelligence (AmI) like Ricardo Costa describes as a digital environment that not only is able to work with present data but also is sensitive, adaptive and responsive to the needs of peoples [2]. In contrast the current technology requires a lot of configuration work in order to achieve home automation that does not even respond to changes in the users behavior. Of course it is even more complex for more than one user in a single environment. The idea of using artificial neural networks to get rid of this configuration work isn't new (1998)[3]. The essential question is whether there even is a repetitive behavior in humans everyday life. To test this hypothesis this project developed a basic artificial neural network and trained it with data gathered by the homematic components provided by EWEs TechLab.

1.3 Neural Networks in Home Automation

Home automation offers, as described, a wide range of opportunities to enhance our lives. But despite of the fact that home automation is around for quite some time now it hasn't catched on fully till today.

There are multiple reasons for that. Beginning costs are still one of the biggest obstacles in home automation. Moreover the emerging costs to configure and later reconfigure a home automation system hold people back from getting in to home automation. Another reason is the complexity that most older systems (that still provide the biggest scope of functions) bring. In the end there just is a fear that holds people back from emerging into home automation systems because they can not really assess the amount of work they have to do.

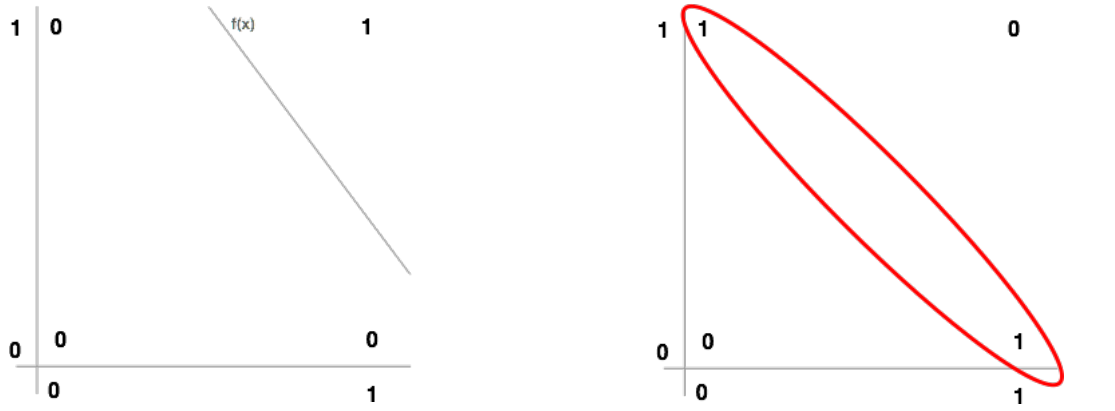
Therefore artificial neural networks could be used to get rid of the hassle of programming a home automation system. Because the system is able to learn the needs of the user by reading out the installed sensors there is no need to configure it at all. It just learns along with the changing needs of the user.

The big challenge is to get the learning data for the networks learning algorithm. However data can be read from the sensors quite easy, it is not that easy to create a coalition to time. Moreover this project is trying to explore the capabilities of two layer networks at first to show that even that basic structures have an enormous potential.

2 General Approach

The general approach to the ANN was to obtain a very simple architecture. Accordingly the challenge was to break the ANN down to the most simplest form, which is a one layer network. Therefore there has to be a linear function $f(x)$ that is able to separate the different kinds of output states.

For instance the output states of an AND gate are linear separable as shown in Fig. 1a. In contrast the output states of an XOR gate are not linear separable as shown in Fig. 1b. Of course it would be nice to have a proof that the input data for the neural network wont ever produce an XOR like behavior. In this case the number of possible input - and related output - states is not assessable moreover the field of application and the behavior of possible installed sensors and actuators which also could be spitefully and therefore does not allow a basic proof. In the following it is shown that common sensors and actuators will not create an XOR behavior. At first the network is trying to learn the relation between sensors and actuators. Every actuator is related to a specific amount of sensors. Therefore there is a neural



(a) Output states of an logic AND gate

(b) Output states of an logic XOR gate

Fig. 1: Linear separability of logic gate output states

network for every actuator that could learn the specific dependencies. For this case we will look at one of thus specific neural networks that models the relation between some sensors and one actuator. To create an XOR like behavior their have to be to sensors that trigger a change in the actuator for itself but not in combination with all other sensors. In general common relations between sensors and actuators will not behave like that. However their may be cases that could create a behavior like that. Nevertheless this project focuses on working with single layer networks to find out in what extend thus can represent a home automation system. Accordingly we are just fine with the knowledge that most combinations will not harm our network. Fig. 2 shows the final structural layout all later created networks will conform to.

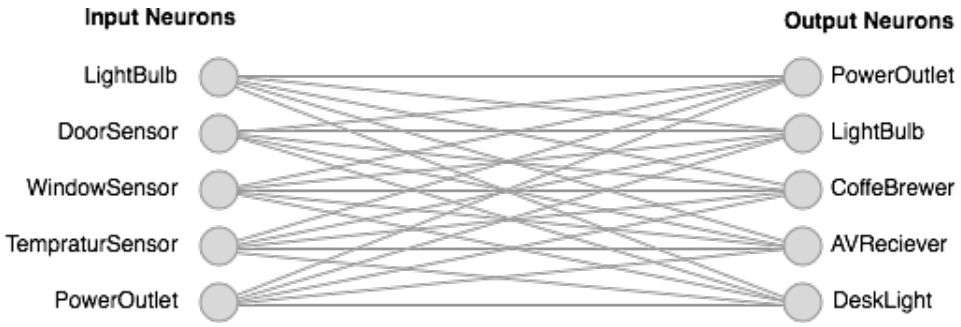


Fig. 2: Final ANN structure

3 Neural Networks in General

The concept of artificial neural networks (ANN) is a model in machine learning that is base on the humans central nervous system. ANNs mostly are used to solve problems that are hard to solve with regular rule based programming. The most common application of ANNs is pattern recognition. The basic concept of ANNs is to approximate functions on the basis of a large amount of data points. This determined function then should be able to divide the data points into different clusters of output states.

To mimic the behavior of a human brain artificial neurons or so called units are connected by weights that will be modified in the process of learning. This weights represent the importance of a connection between to neurons.

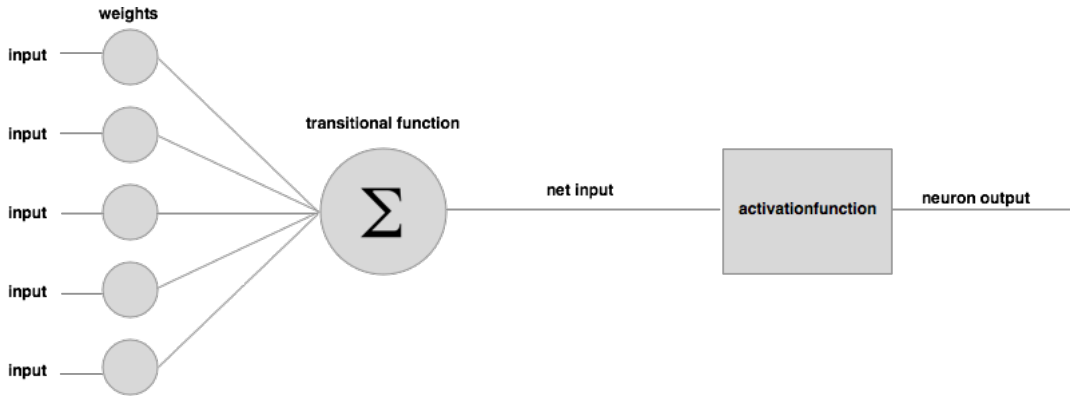


Fig. 3: Basic artifical neuron layout

As shown in Fig. 3 a neuron represents a complex structure that turns its inputs into an output by passing them through multiple computation steps.

At first the neuron calculates an overall neuron input value. Therefore the neuron gathers the input values and multiplies them with their weights. This assures that inputs with a higher weight - inputs that are more important - are represented by a bigger input value. To get the overall input value all calculated input values get added together as shown in Eq. 1.

$$\sum_{i=1}^{n} net_j = x_i \dot{w}_i \tag{1}$$

After calculating the net input the neuron determines its output value by passing the net input through its activation function. The activation function could be any kind of function. Very common choices for an activation function are step functions, linear functions and sigmoid functions.

3.1 Hard Limit Functions

Hard limit functions can be used to map inputs in certain intervals to constant outputs. That could be particularly helpful if output values have to be in a certain format and can't be modified afterwards.

$$\varphi(x) = \begin{cases} 1 & : x \geq 0 \\ 0 & : x < 0 \end{cases} \tag{2}$$

Eq. 2 is a simple hard limit function that maps all inputs greater or equal to 0 to 1 and all inputs less than 0 to a 0.

3.2 Piecewise Linear Functions

Piecewise linear functions allow a more specific definition of the activation function. In contrast to hard limit functions piecewise linear functions allow the use of learning algorithms that need the derivative activation function to calculate weight changes in a much more convenient way.

$$\varphi(x) = \begin{cases} 1 & : \text{if} \quad x \geq \frac{1}{2} \\ x + \frac{1}{2} & : \text{if} \quad -\frac{1}{2} < x < \frac{1}{2} \\ 0 & : \text{if} \quad x \leq -\frac{1}{2} \end{cases}$$

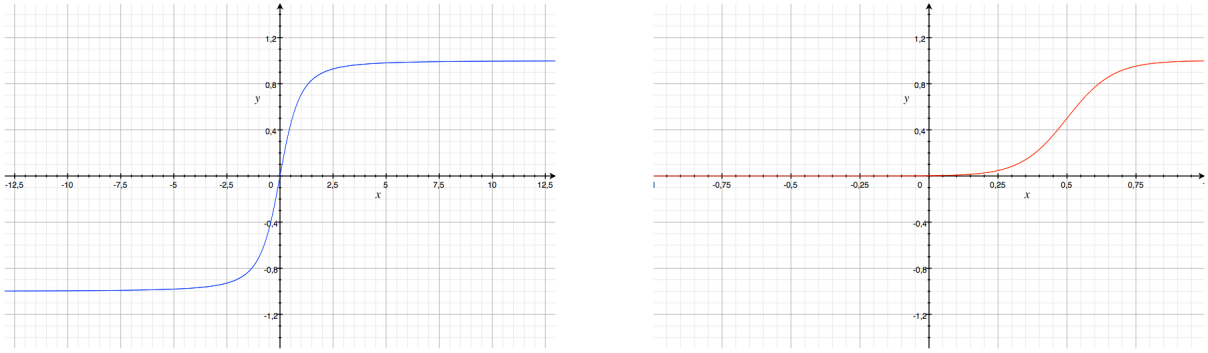
3.3 Sigmoid Functions

The sigmoid function probably is the most common activation function. That is because of the derivability that allows the use of learning algorithms like back propagation. There also is the opportunity of using a sigmoid function in substitution for the hard limit function.

$$\varphi(x) = \frac{x}{\sqrt{1 + x^2}} \tag{3}$$

In this project sigmoid functions are used as activation functions. As it turned out the standard sigmoid function (Fig.: 4a) did not fit the expectations in term of the value range. To get the sigmoid function to fit the requirements it was modified from Eq. 3 to Eq. 4 (Fig. 4b).

$$\varphi(x) = \frac{1}{\sqrt{1 + e^{(-x \cdot 12 + 6)}}} \tag{4}$$



(a) Standard sigmoid function (b) Custom sigmoid function

Fig. 4: Modification of the sigmoid function

4 Learning by Backpropagation

The back propagation learning algorithm is one of the most popular learning algorithms for artificial neural networks. It is based on the delta rule which uses gradient descent to calculate a global minimum for the error function.

$$E = (t - y)^2$$

where:

- t is the expected output
- y is the actual output
- E is the error value

The error function is a mapping from the actual outputs to the calculated error for that output in regard to the expected value. So the error function can be displayed as a parabola - for simple one layer networks - with actual outputs on the x-axis and the error value on the y-axis.

For thus simple networks the minimal error can easily be found by calculating the parabolas minimum. But at this stage the neural network is not able to learn yet. To enable the learning functionality the weights the actual output values depend on have to be included in the calculation. Therefore each weight has to be displayed on a separate horizontal axis with the error value still on the vertical axis. Now the error function is represented as a parabolic bowl which contains a minimum that depends on the input weights.

$$y = x_1 \cdot w_1 + x_2 \cdot w_2$$

where:

- x is the neuron input
- w is the neuron input weight
- y is the actual output

The gradient descent algorithm is one of the algorithms that can be used to calculate the minimum for the parabolic bowl. Basically the algorithm calculates the steepens of the function for the surrounding points and then defines the point with the greatest steepens as the new current point.
To use gradient descent in the propagation algorithm the activation functions derivative is needed. To get rid of the exponents while differentiating the squared error function is extended by the factor $\frac{1}{2}$.

$$E = \frac{1}{2}(t - y)^2$$

The neurons output value (o_j) is defined by putting the neurons input in the chosen activation function:

$$o_j = \varphi(\sum_{i=1}^{n} x_i \dot{w}_i)$$

For this particular ANN Eq. 4 was chosen as activation function. Because a sigmoid function was chosen there is a very nice derivative for that activation function:

$$\frac{\partial \varphi}{\partial x}(x) = \varphi(x)(1 - \varphi(x)) = \frac{1}{\sqrt{1 + e^{(-x \cdot 12 + 6)}}} \cdot (1 - \frac{1}{\sqrt{1 + e^{(-x \cdot 12 + 6)}}})$$

To calculate the final weight changes for each of the neurons it is important to differentiate between the different types of neuron types. Because we are working with single layer networks the weight changes for neurons in the output layer directly depend on the actual and expected output for the neuron. Therefore the error for one neuron is pretty simple to calculate:

$$\delta_j = \varphi'(\sum_{i=1}^{n} x_i \dot{w}_i)(j_i - y_i)$$

Finally the complete Eq. 5 to calculate the weight change ends up looking like this:

$$
\begin{aligned}
\Delta w_{ij} &= -\eta \frac{\partial E}{\partial w_{ij}} \\
&= \eta \delta_j x_i \\
&= \eta \varphi'(\sum_{i=1}^{n} x_i \dot{w}_i)(j_i - y_i) x_i \\
&= \frac{1}{\sqrt{1 + e^{(-\sum_{i=1}^{n} x_i \dot{w}_i \cdot 12 + 6)}}} \cdot (1 - \frac{1}{\sqrt{1 + e^{(-\sum_{i=1}^{n} x_i \dot{w}_i \cdot 12 + 6)}}})
\end{aligned}
\tag{5}
$$

To optimize the learning behavior of the neural network it is important to choose the optimal learning rate. If the learning rate is to high the algorithm could miss the minimal error for quite some time and therefore could be even slower then a lower learning rate. The learning rate can be imagined as a sampling rate which has to be on point to really extract the information from a signal. To illustrate the learning rates impact on the learning behavior Fig. 5 is showing the learning curves for a neural network with 20 input and 40 output neurons with initial weights of zero.

In this ideal conditions with none changing input and output values it could be shown that a higher learning rate also results in a overall faster learning behavior. To investigate further this test was repeated with fault values on three of the neurons to simulate a more realistic learning scenario. The results are shown in Fig. 6.

To get a greater picture on the learning behavior we also tested the algorithm with fourteen fault values which is shown in Fig. 7. To get a better view on the results the overall net error was calculated for every test case. To get the net error each of the error functions was integrated to calculate the area below the graphs. The results are shown in table 1.

The experiment has shown that the learning rate should be lower for networks with many changing input values and higher for networks with relatively equal input values. Furthermore the experiment has shown that higher learning rates are able to quickly calculate weights that are close to the minimal error but fail to calculate the minimal error because of the high fluctuation. Because the network probably has to work with many changing input values but also is expected to get to the minimal error has fast as possible we choose 0,7 as our learning rate.

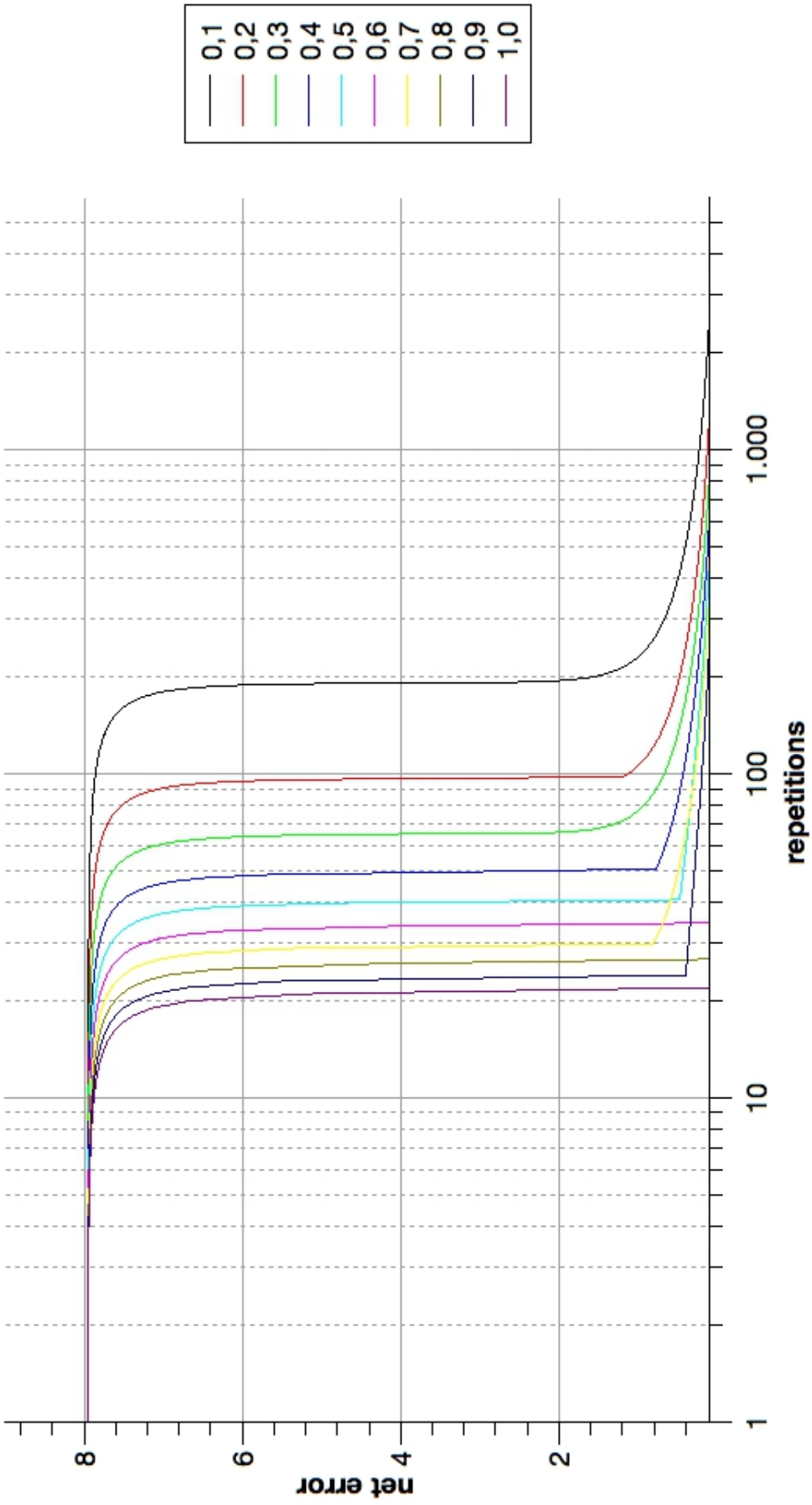


Fig. 5: Learning Behaviour with Different Learning Rates

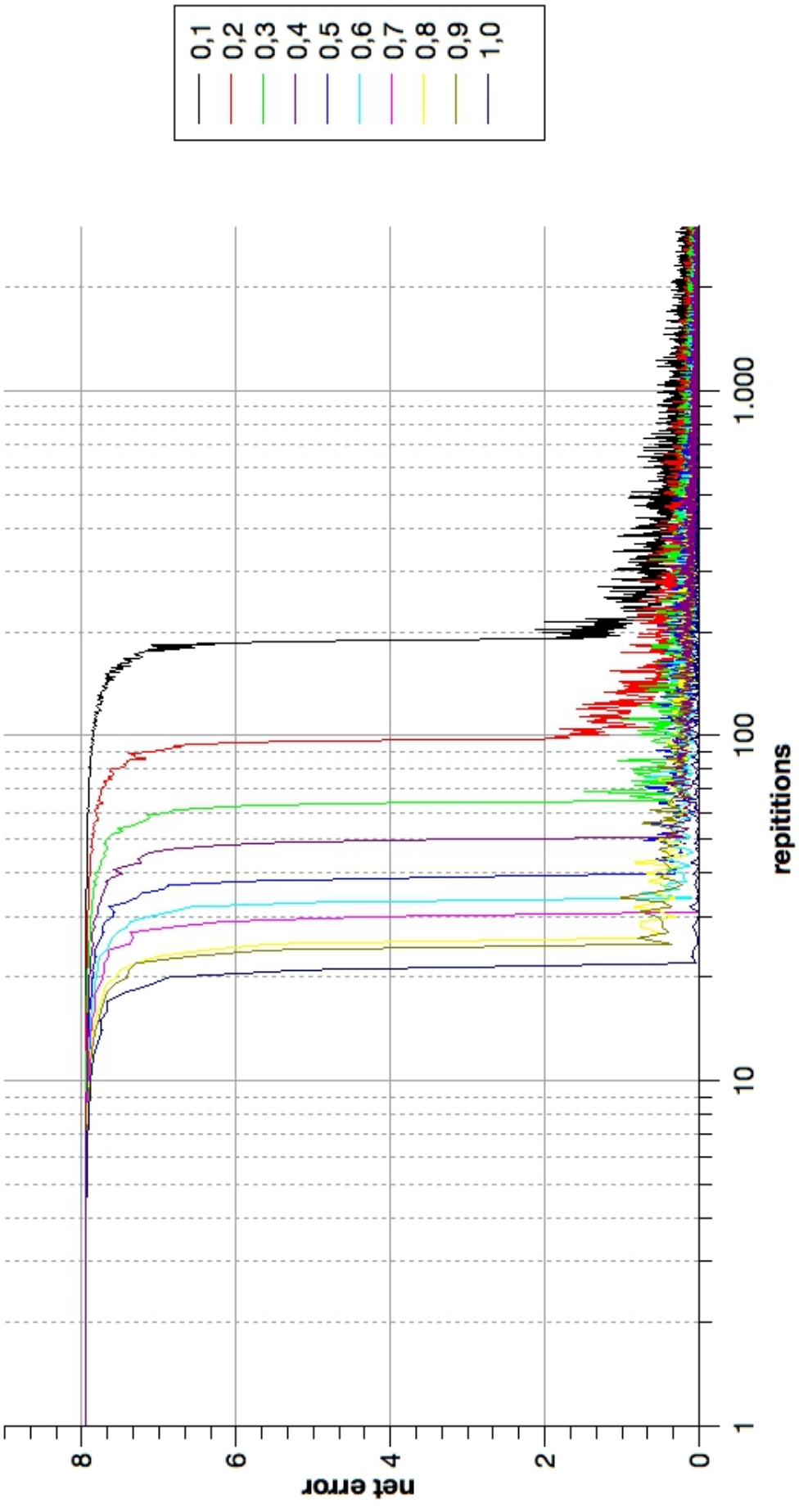


Fig. 6: Learning Behavior with Different Learning Rates and Three Random Inputs

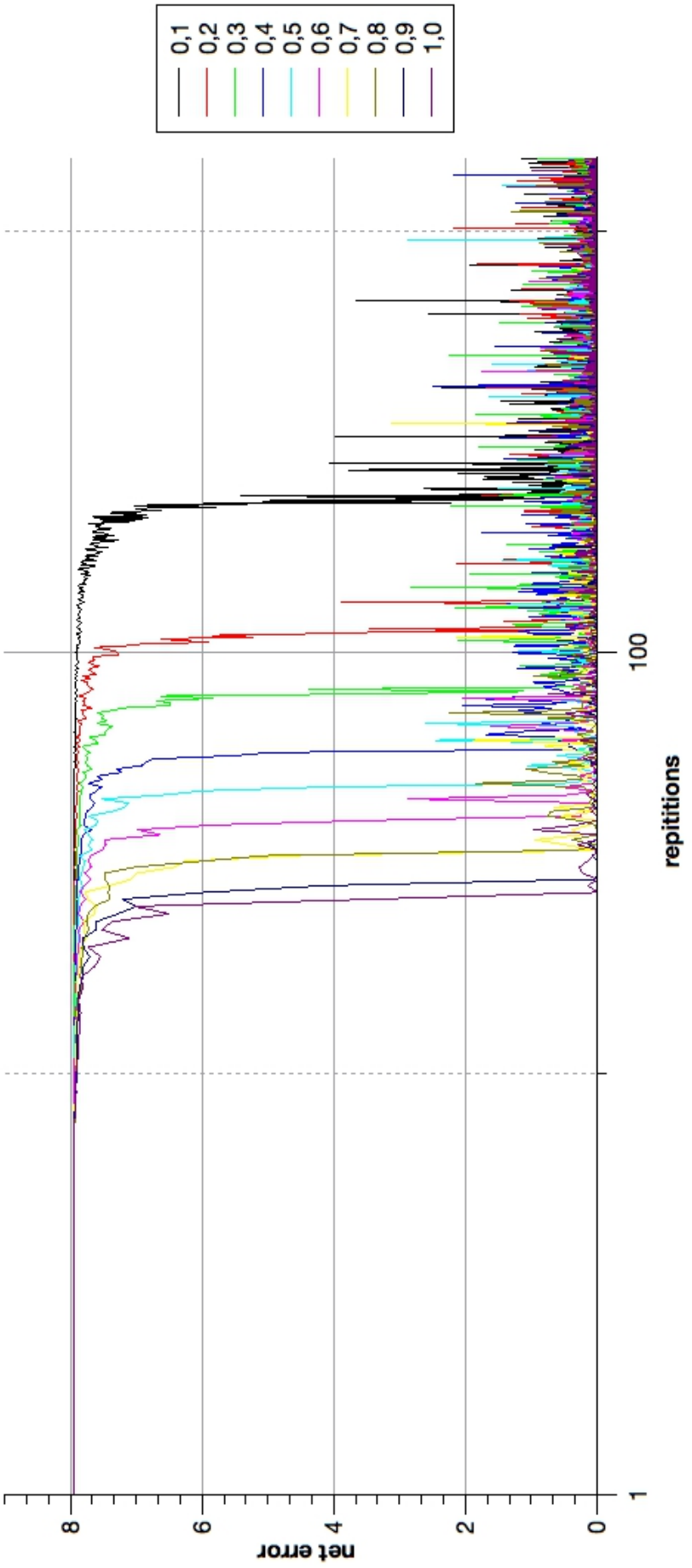


Fig. 7: Learning Behavior with Different Learning Rates and Fourteen Random Inputs

η	overall net error
0,1	2.531,40
0,2	1.358,73
0,3	908,88
0,4	716,15
0,5	579,99
0,6	421,29
0,7	273,34
0,8	196,42
0,9	171,94
1,0	284,68

η	overall net error
0,1	2.164,53
0,2	1.165,22
0,3	756,04
0,4	529,29
0,5	454,24
0,6	367,61
0,7	195,59
0,8	302,28
0,9	281,52
1,0	176,04

η	overall net error
0,1	2.167,91
0,2	1.127,65
0,3	797,59
0,4	644,79
0,5	437,35
0,6	391,22
0,7	264,53
0,8	338,12
0,9	211,03
1,0	231,30

Tab. 1: Relation between learning rate and overall net error

5 Obstacles

As common there where quite a few obstacles in the course of the project. For instance the selection of a home automation system. The condition was to use a system that provides sensors and actuators that could be installed in every home without a lot of construction work. The first and obvious choice was to use Apple HomeKit framework. However we tried to use HomeKit at first we also realized that there is not a lot of supporting hardware for HomeKit because of its recent release. Subsequently we had to switch the home automation system in the middle of the project. That resulted in a complete rewrite of the hardware interface for the app. Also the Homematic system was quite hard to implement in Swift because of poor HTTP and XML parsing support by Swift. However there where nice open source frameworks which added these features to Swift.

6 Implementation

The implementation goal was to create a prototype that could connect to a home automation system and use the systems data to run a neural network. In the beginning of the project Apple released their HomeKit API for Swift. Because the HomeKit API was quite easy to implement we decided to use Swift as programming language for the project. That meant the prototype had to be an iOS app. Sadly HomeKit could not be used because of the small amount of devices that would support it. Consequently a new home automation system had to be found. In cooperation with EWE Homematic was chosen as our new home automation system. Additionally the EWE provided a set of Homematic devices for testing purpose. Moreover Homematic is more or less open source and therefore has a open source community that developed an XML plug-in that allowed us to read out all device informations as XML data. At this point the implementation of the neural network was quite advanced so we had to continue using Swift as our programming language. Subsequently some third party frameworks had to be added to the project to enable Swift to perform syntacticly elegant HTTP requests and XML parsing. The frameworks used are Alamofire for elegant HTTP requests and EVReflection,XMLDictionary and AlamofireXMLToObjects for XML parsing. To store data on the iPhone we used Apples CoreData framework which allows elegant access to the underlying SQLLite database.

To create the neural network a new `IHNeuralNetwork` object gets created. Additionally the neural network layers get created by instantiating two `IHLayer` objects. After that the app sends an HTTP request to the Homematic central to download the XML data that describes all connected devices (`devicelist.cgi`). To get the related data points for each device another HTTP request is send that downloads an XML file that contains all channels and data points for one device (`state.cgi`). Sadly the Homematic XML plug in does not support differentiation between sensors and actors. Accordingly the data points are assigned to the layers manually. A data point is represented by the class `PDatapoint`. The `PDatapoint` conforms to the `NNDevice` protocol that requires a `read()` and a `write(value: Any)` method to enable interaction with the device. Accordingly one could also use a different home automation system as long as its representation class conforms to `NNDevice`.
Because of inheritance limitations in Swift a data point is also represented as `IHNeuron` for all neural network calculations. To connect the neural network with the devices `IHNeuron` holds the related `PDatapoint` object as a class member.

After the neural network is created there are two main methods included in the `IHNeuralNetwork`

class. Firstly the `feedForeward(inputValues:[float_t], context : NSManagedObjectContext)` allows to feed an array of input values - that could be gathered from the input layers `IHNeuron`s `PDatapoint` sub objects - through the neural network. Additionally their is a `backPropagation(expectedValues: [float_t], context: NSManagedObjectContext)` method that also takes an array of float values that could be gathered from the actors `PDatapoint` objects.

A neural network can be load from the database by calling the `IHNeuralNetwork`s static `loadFromPersistentStorage(id: Int, contex: NSManagedObjectContext)` method. That method also loads all related sub objects for that specific `IHNeuralNetwork` object.

6.1 Homematic XML API

To get a better understanding for the Homematic XML API I am going to give a brief overview for the generated XML files. As described the first HTTP request triggers the `devicelist.cgi` script. The produced XML is shown in Listing 1. The device list contains three entities:

deviceList The root entity to get a well formed XML document

device Represents a device that is connected to the Homematic central. Devices are described by its attributes name, address, unique id (ise_id), the type of interface, the type of device and the configuration status.

channel Represents a channel that is provided by a device. Channels are described by its attributes name, type, address, unique id (ise_id), direction (UNKNOWN,SENDER,RECIVER), related device.

Listing 2 describes the state of a specific device. Therefore it contains the actual data points that represent the functions and sensor values of the device.

state The state entity is the root entity to make the document well formed.

datapoint The data point entities represent the actual data for connected devices. They also are identified by a unique id which is later needed to identify the data point again to read out or set the current value.

The `state.cgi` is also used to directly read out the data for a given data point. Therefore the `state.cgi` can be called with a data point id as GET parameter. In this case the `state.cgi`

only response with a data point that contains the the id and the actual value as shown in Listing 3.

```
<deviceList>
        <device name="HMIP-eTRV_000393C98D1B9D" address="000393C98D1B9D" ise_id="1387" interface="HmIP-RF"
            device_type="HMIP-eTRV" ready_config="true">
                <channel name="HMIP-eTRV_000393C98D1B9D:0" type="30" address="000393C98D1B9D:0" ise_id="1388"
                    direction="UNKNOWN" parent_device="1387" index="0" group_partner="" aes_available="false"
                    transmission_mode="AES" visible="true" ready_config="true" operate="true"/>
                <channel name="HMIP-eTRV_000393C98D1B9D:1" type="17" address="000393C98D1B9D:1" ise_id="1403"
                    direction="SENDER" parent_device="1387" index="1" group_partner="" aes_available="false"
                    transmission_mode="AES" visible="true" ready_config="true" operate="true"/>
        </device>
</deviceList>
```

Listing 1: devicelist.cgi

```
<state>
        <device name="HMIP-eTRV_000393C98D1BAC" ise_id="1238" unreach="false" config_pending="">
                <channel name="HMIP-eTRV_000393C98D1BAC:1" ise_id="1254">
                        <datapoint name="HmIP-RF.000393C98D1BAC:1.ACTUAL_TEMPERATURE" type="
                           ACTUAL_TEMPERATURE" ise_id="1256" value="24.600000" valuetype="4" valueunit=
                           "" timestamp="1472655119"/>
                        <datapoint name="HmIP-RF.000393C98D1BAC:1.SET_POINT_TEMPERATURE" type="
                           SET_POINT_TEMPERATURE" ise_id="1269" value="12.000000" valuetype="4"
                           valueunit="" timestamp="1472655119"/>
                        <datapoint name="HmIP-RF.000393C98D1BAC:1.SWITCH_POINT_OCCURED" type="
                           SWITCH_POINT_OCCURED" ise_id="1270" value="false" valuetype="2" valueunit=""
                            timestamp="1472655119"/>
                        <datapoint name="HmIP-RF.000393C98D1BAC:1.VALVE_ADAPTION" type="VALVE_ADAPTION"
                            ise_id="1271" value="" valuetype="2" valueunit="" timestamp="0"/>
                        <datapoint name="HmIP-RF.000393C98D1BAC:1.VALVE_STATE" type="VALVE_STATE"
                           ise_id="1272" value="4" valuetype="16" valueunit="" timestamp="1472655119"/>
                        <datapoint name="HmIP-RF.000393C98D1BAC:1.WINDOW_STATE" type="WINDOW_STATE"
                           ise_id="1273" value="0" valuetype="16" valueunit="" timestamp="1472655119"/>
                </channel>
        </device>
</state>
```

Listing 2: state.cgi

```
<state>
        <datapoint ise_id="1269" value="4.500000"/>
</state>
```

Listing 3: state.cgi

7 Conclusion

The projects goal was to show how artificial neural networks could help to create a better user experiences for home automation systems. After the creation of a neural network that would fit our needs the project was able to show that simple one layer neural networks are able to recognize patterns in user behavior. Real world functionality still has to be further tested because of the lack of functionality in terms of cache control in the Homematic system used in this project. Also the project has shown that single layer networks only are able to recognize patterns that exist while the actuator changes its state. To recognize patterns that also rely on data from the past the network needs more layers to save thus data sets internally. In the end the project has shown that single layer artificial neural networks definitely could help to support the configuration of home automation systems.

8 Future Plans

In the future the project is going to do research on multi layer artificial neural networks in home automation. Also the system will be re-factored to create a more user friendly API. In the end the Loxone in-home grid home automation system is going to be implemented to feed the neural network.

List of Figures

References

[1] Amit Badlani and Surekha Bhanot. “Smart home system design based on artificial neural networks”. In: Proc. of the Word Congress on Engineering and Computer Science. 2011.

[2] Ricardo Costa et al. “Ambient assisted living
”. In: 3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008. Springer. 2009, pp. 86–94.

[3] Michael C Mozer. “The neural network house: An environment hat adapts to its inhabitants”. In: Proc. AAAI Spring Symp. Intelligent Environments. 1998, pp. 110–114.