Realisation of Smart Home applications with Lego Mindstorms and neural networks

Kerstin Strecker[1]

[1] University of Göttingen, Goldschmidtstr. 7, 37077 Göttingen, Germany
kerstin.strecker@informatik.uni-goettingen.de

Abstract. In this article, we present a teaching sequence to introduce the topic area of neural networks with the help of Lego robots. Using the Lego system, we implement applications in the area of home automation in a large doll's house. With the help of neural networks, the home control system "learns" the behaviours of its residents and makes it individually configurable for each resident. The learning scenario enables differentiating tasks, which can be selected by the students themselves and thus allow all students to be successful.

The lesson introduces the functionality of perceptrons and small multilayer neural networks on the basis of logical expressions.

Keywords: Neural Networks, Lego Mindstorms, Smart Home

1 The Smart Home as a classical "physical computing" application

The Smart Home [1] [2] is a tried-and-tested example application in physical computing [3], as it integrates a range of simple and real-life examples that can be implemented successfully by all pupils.

Examples of the type:

- when it gets hot, the ventilator switches on automatically.
- when it gets dark, the light switches on automatically.
- if a burglar steps on the touch sensor in the doormat, an alarm goes off.
- when the sun shines, the awnings are extended automatically.
- …

are comprised in informatics terms of a query of whether the sensor value is exceeded or falls short, and an action that follows. Sometimes, several sensor values can be logically related:

- if grandmother has left the cooker on and it gets dark outside and she goes to bed (touch sensor in bed), then the alarm goes off.

In a demonstration of the finished Smart Home, the examples are very impressive, but they are not (from experience) too complex in informatics terms to prevent all pupils from implementing them successfully. Another motivating factor is that pupils can select their own problems (under the condition that the sensors and actors are known), as the given sensors and actors limit the number of ideas to implementable concepts.

This enables internal differentiation as well as a range of complexity with regard to Smart Home solutions [2].

The target group with which this teaching scenario has been tested repeatedly [2][4] comprises pupils in secondary level 1, who have little previous experience with the creation of their own programs. The learning scenario is thus also a good introduction to algorithms [5].

In this article, we want to expand the example of home control with physical computing systems (here: Lego Mindstorms [6]) to include neural networks. As such, the home control should become individually configurable. To illustrate: In the traditional Lego Smart Home, the algorithm: when it gets light, the window is opened automatically, may have been implemented. In combination with the neural networks, the Smart Home system “learns” that the resident always opens the window when it is light outside, and carries out this action autonomously after some time. In this way, it corresponds with the above-described algorithm. However, the resident may also open the window every time it gets dark. This would lead to another configuration of the neural network, according to which the window is always opened when it is dark outside. In the traditional Smart Home application, this would require the implementation of a new algorithm.

For our Smart Home application with neural networks, we use the Lego-NXT Robot [6], Lego bricks and the Enchanting software [7] as a physical computing system. In the following, we will intentionally use very simple, non-complex networks, whose training phase can be easily understood manually. In this way, we are providing an introduction to the topic area of neural networks, which can be consolidated at a later date. We shall not consider complex networks and are also consciously reducing the implementation into a programming language didactically. With this reduction however, the learning scenario is also suitable for secondary level 1, a point in time at which some programming skills are likely to already exist.

2 Our example scenario

Our Smart Home application is an automatic window control system. This example should be viewed as representative for many other Smart Home applications. The window is to be opened and closed by means of a motor. On the one hand, this should be possible manually by using a touch sensor. The state of the window can be adjusted with the touch sensor. If the window is open, it can be closed by using the touch sensor. If the window is closed, it can be opened using the touch sensor.

Our neural network should be able to open and close the window automatically depending on the preference of the resident. To do this, it processes the input values of two sensors: a sound sensor and a light sensor. For simplicity’s sake, there are only two different input values for each sensor. The sound sensor can measure loud or quiet. The light sensor can measure light or dark. Both sensors are attached to the outside wall of the house.

In order to motivate the integration of a neural network, we consider the following example scenario.

Take the house of the Smith family: It has several bedrooms, in which the same new intelligent window closing system is to be installed. However, the residents of our house all have very different preferences.

Uncle Peter always sleeps with the window open, regardless of how noisy or quiet it is outside. But he always keeps the window closed during the day, because he is at work and a closed window seems more secure to him. Above all, he doesn't want any insects to fly into his room during the day.

Mum Kate also sleeps with the window open, regardless of how noisy or quiet it is outside. During the day however, she opens the window even if it is loud outside, because that's when the children play in the garden and Mum Kate wants to be able to hear them. The windows are only kept shut during the day when it is quiet outside.

Grandma Emmy has her very own peculiarities. She opens the window during the day when it's noisy so she can hear her grandchildren playing and the birds singing. On quiet days, the window is kept closed. She likes her peace and quiet during the night. As such, she only opens the window at night when it is quiet outside. If it is noisy outside, however, she closes the window at night because she cannot sleep otherwise.

Dad Eric always works night shifts. For this reason, he keeps the window closed at night because he's not at home. Dad Eric sleeps during the day. But he only opens the window during the day when it's quiet outside. If not, the window is kept closed so he can sleep in peace.

3 The integration of a neural network

We represent an artificial neuron as follows (see Fig. 1):

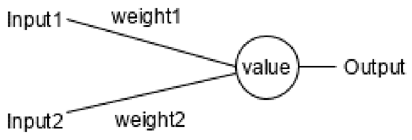


Fig. 1. Representation of a neuron

The following applies:

$$\text{Output} = \begin{cases} 1, & \text{if } (\text{Input1} * \text{weight1} + \text{Input2} * \text{weight2}) \geq \text{value} \\ -1, & \text{if } (\text{Input1} * \text{weight1} + \text{Input2} * \text{weight2}) < \text{value} \end{cases} \quad (1)$$

We want to implement a small neural network, with which simple logical operations can be realised. The network should have only binary inputs and outputs, but have multiple layers, for example to also realise an XOR operation [8]. In order to approach the multi-layer neural network, an initial implementation can include only single neurons/perceptrons, then the necessity of multi-layer networks can be motivated in a task, and finally small multi-layer networks can be used (cf. [9]).

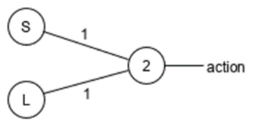
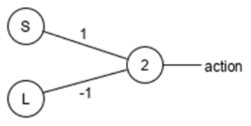
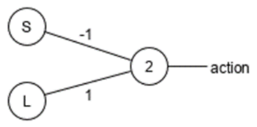
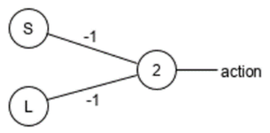
We only choose binary inputs and outputs so that the pupils can manually retrace the adjustment of weights in our small neural networks. In our learning scenario, we use the sound sensor and the light sensor. All sensors should yield binary values, the pupils can decide the thresholds themselves so that the sensors yield the values **-1** or **1**.

The following coding should apply:

- sound sensor S: S corresponds to S = **1** (loud), $\bar{S}$ corresponds to S = **-1** (quiet)
- light sensor L: L corresponds to L = **1** (light), $\bar{L}$ corresponds to L = **-1** (dark)

In this way, we can represent all logical AND operations that can be formed from L and S (in the following referred to as minterms) as perceptrons [10]:

Table 1. minterms

$S \wedge L$:		$S \wedge \bar{L}$:	
$\bar{S} \wedge L$:		$\bar{S} \wedge \bar{L}$:	

If the sum of inputs multiplied by the corresponding weights in a neuron is larger than or equal to the threshold of the neuron, then the neuron yields the **output 1**, **in all other cases -1**. For the output neuron of the final layer we define:

- Output (action) = 1 => Window open
- Output (action) = -1 => Window closed

A perceptron with two input points, which is connected to the two sensors S and L, can be implemented and also trained. We will not go into any more detail at this point, as the implementation results directly from a reduction of the implementation of a multi-layer network which we will describe in detail later, or which can be looked up in [9]. For the opening of his window, Dad Eric wants the configuration L ∧ $\bar{S}$. Here, we would only require one perceptron that we would need to train accordingly. If, however, we tried to find the suitable weights to implement Grandma Emmy's wishes, we would fail. Grandma Emmy's wishes are namely coded as follows:

$$output = \begin{cases} 1 \text{ (window open)} & if\ (S \wedge L) \vee (\bar{S} \wedge \bar{L}) \\ -1 \text{ (window closed)} & else \end{cases} \tag{2}$$

From this, we can derive the necessity of using multi-layer networks, as Grandma Emmy's configuration corresponds to the logical term NOT-XOR.

Our small neural network should be able to realise simple logical operations. We continue to use two input variables and a total of three neurons, which we want to connect as follows (see Fig. 2):

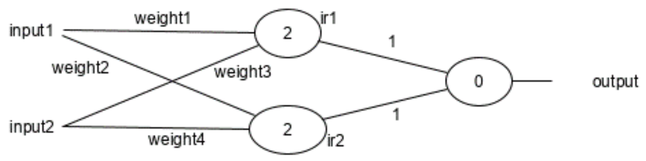


Fig. 2. Neural Network

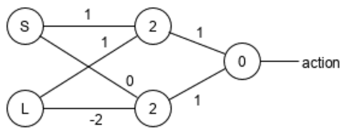
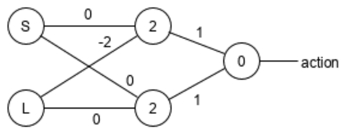
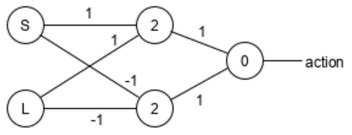
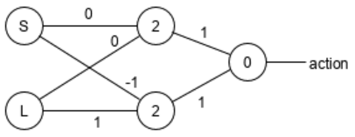
A few words on the use and legitimation of this type of network (cf. multiple adaptive linear neuron [11]). In our example, we are using two sensors with binary values. Our network should be able to model all possible functions from these two variables, that is 16 possible functions. The final layer in our network simulates an OR operation. Let us consider the 16 possibilities in more detail:

- Our function consists of two minterms linked to OR => in accordance with the table 1, the two minterms can be modelled with the two neurons and are linked to OR in the second layer (example: Grandma Emmy and Uncle Peter)
- Our function consists of a minterm => weight3 and weight4 are set to 0, weight1 and weight2 according to the minterm as per table1 (example: Dad Eric)
- Our function consists of three minterms linked to OR => of these, at least two terms can be summarised to S, L, $\overline{S}$ or $\overline{L}$. This can be modelled in one of the neurons in the intermediate layer. The remaining third term can be modelled with the other neuron (example: Mum Kate)
- Our function is always -1 => set all weights to 0
- Our function is always 1 => set weight3 and weight4 to 0, set weight1 to the value 2 and weight2 to the value -2.

The network in Figure 2 is thus sufficient for any “desired window opening” of our residents. In didactic terms, however, we also have the advantage that the weights in the final layer remain constant, and that during the learning phase, we only have to consider changes to the weights in the first layer. Processes such as backpropagation [12][13] etc. therefore do not have to be considered; the pupils can decide themselves how the weights have to be adjusted during the learning phase, but more on this later. A further advantage is that the network learns very quickly. Later on, our learning rate will be 0.5, and thus the network usually “learns” with only a few training runs. This is also important from a didactic point of view, as a neural network can lose its meaningfulness to pupils if they later have to manually adjust the window too often.

First, we translate the requirements of our residents into neural networks (whereby there can be several options). This is also a possible initial practice task for pupils, which can enable a reflective consideration of the training of our neural networks later on.

Table 2. Neural Networks Family Smith

 S L 1 1 0 -2 2 2 1 1 0 action Mum Kate	 S L 0 -2 0 0 2 2 1 1 0 action Uncle Peter
 S L 1 1 -1 -1 2 2 1 1 0 action Grandma Emmy	 S L 0 0 -1 1 2 2 1 1 0 action Dad Eric

4 The implementation of our Smart Home system with neural networks

For the implementation, we separate our problem into three stages. First, we look at the mechanics. To do so, we equip a window with a motor and open and close it by using a touch sensor, which we test.

In the second stage, we implement the neural network, but set the weights by hand in the source code. In this way, we can realise and test the control for Grandma Emmy or Mum Kate or Uncle Peter.

In the third stage, the weights should adjust automatically and our Smart Home solution should be equally suitable for all residents. For this, the neural network should control the window motor. If, however, a resident is unsatisfied and subsequently adjusts the window manually using the touch sensor, the weights should change; thus, the neural network should "learn" the behaviour of our residents.

The "learning" of our neural network is comprised quite deliberately **only** of the change to the weights at the edges. The threshold should remain as unchanged as the number of neurons or the existence of links. For the pupils however, a systemisation stage of the topic should make clear that this represents a didactic reduction, and that a neural network can, in principle, be changed in other ways during the running time.

The code that implements all three stages is described in the following.

4.1 Explanation of the implementation

First, we will explain the use of variable names in the following figure (Fig. 3):

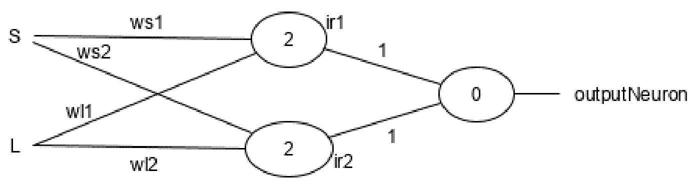


Fig. 3. use of variable names

The sound sensor yields the value **S = 1** if it is loud outside, and **S = -1** if it is quiet outside. This is encapsulated in the block: "`get value sound sensor`"

The light sensor yields the value **L = 1** if it is light outside, and **L = -1** if it is dark outside. This functionality is also encapsulated in the block: "`get value light sensor`"

The weights on the incoming edges of the upper neuron of the first layer are named ws1 and wl1. The neuron yields the result **-1** or **1**, which is stored in the variable ir1. (ir1 stands for "interim result 1"). The following applies: If the sum of inputs multiplied by the associated edge weights is larger than or equal to the threshold, then the output is 1 (the neuron fires), in all other cases it is -1.

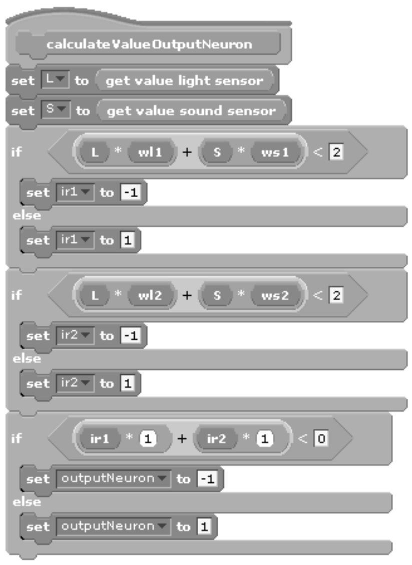
In this case, mathematically stated:

$$\mathrm{ir1} = \begin{cases} 1, & \text{if } (\mathrm{S} * \mathrm{ws1} + \mathrm{L} * \mathrm{wl1}) \geq 2 \\ -1, & \text{else} \end{cases} \tag{3}$$

$$\mathrm{ir2} = \begin{cases} 1, & \text{if } (\mathrm{S} * \mathrm{ws2} + \mathrm{L} * \mathrm{wl2}) \geq 2 \\ -1, & \text{else} \end{cases} \tag{4}$$

$$\mathrm{outputNeuron} = \begin{cases} 1, & \text{if } (\mathrm{ir1} * 1 + \mathrm{ir2} * 1) \geq 0 \\ -1, & \text{else} \end{cases} \tag{5}$$

The corresponding code is (see Fig. 4):



```
calculateValueOutputNeuron
set L to get value light sensor
set S to get value sound sensor
if ((L * wl1) + (S * ws1)) < 2
    set ir1 to -1
else
    set ir1 to 1
if ((L * wl2) + (S * ws2)) < 2
    set ir2 to -1
else
    set ir2 to 1
if ((ir1 * 1) + (ir2 * 1)) < 0
    set outputNeuron to -1
else
    set outputNeuron to 1
```

Fig. 4. Code “output neuron”

At the start of the program, all weights are set to zero and the light source of the light sensor is “switched off”.

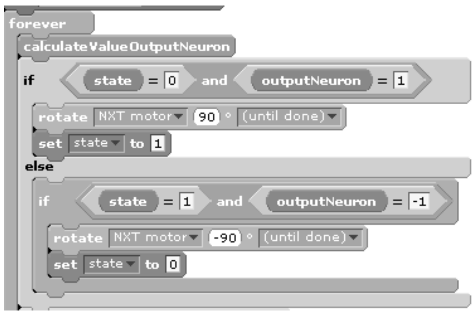
As we want to be able to open and close our window manually using the touch sensor on the one hand, and automatically on the other hand, a further variable state is introduced. If **state=0**, this means that the window is currently closed. Variable **state=1** means that the window is currently open. At the start of the program, the variable state is set to zero. The code for the automatic window control is shown in Figure 5.

Let us now consider the manual control. The window can be opened and closed manually with the touch sensor. If the touch sensor is activated and the window is open, it will be closed. If the touch sensor is activated and the window is closed, then it is opened. For this, we use our variable state. However, if the window needs to be opened or closed manually, this means that our neural network is not configured according to the requirements of the residents. The weights thus need to be adjusted.

If the window is closed and then opened manually, the weights have to be increased so that it would actually have been opened automatically.

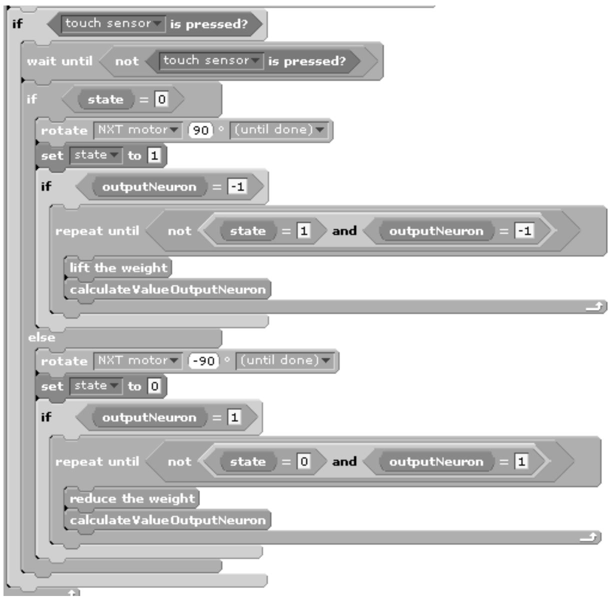
If the window is open and then closed manually, the weights have to be decreased, because it should actually have closed automatically (Figure 6).

We carry out several runs with the same settings of input variables until the weights are set for this one case. This also decreases the number of training runs.



```
forever
  calculateValueOutputNeuron
  if state = 0 and outputNeuron = 1
    rotate NXT motor 90 ° (until done)
    set state to 1
  else
    if state = 1 and outputNeuron = -1
      rotate NXT motor -90 ° (until done)
      set state to 0
```

Fig. 5. Code “automatic window control”



```
if touch sensor is pressed?
  wait until not touch sensor is pressed?
  if state = 0
    rotate NXT motor 90 ° (until done)
    set state to 1
    if outputNeuron = -1
      repeat until not state = 1 and outputNeuron = -1
        lift the weight
        calculateValueOutputNeuron
  else
    rotate NXT motor -90 ° (until done)
    set state to 0
    if outputNeuron = 1
      repeat until not state = 0 and outputNeuron = 1
        reduce the weight
        calculateValueOutputNeuron
```

Fig. 6. Code “manual control”

Finally, let us consider the blocks “lift the weight” and “reduce the weight” (see Fig. 7). The learning rate was set to 0.5 at the start of the program, and is stored in the variable “learning rate”.

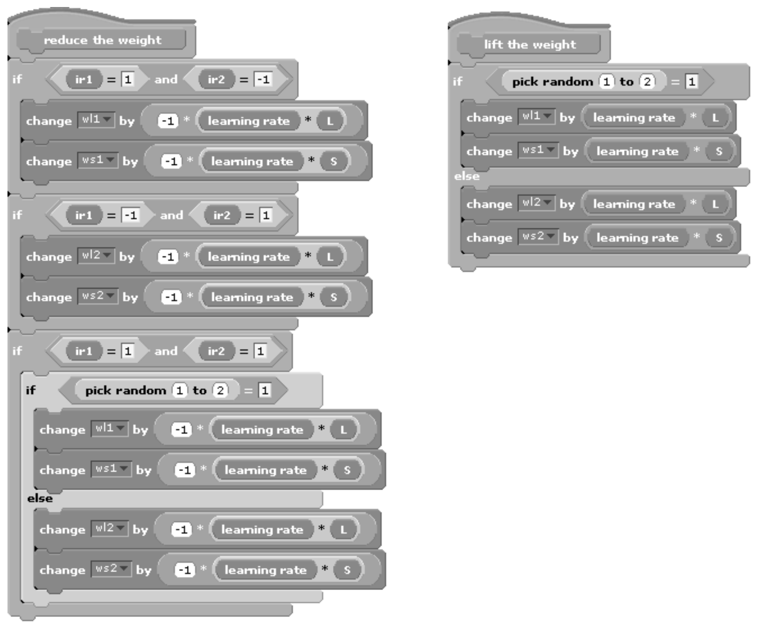
If the window is open and is closed manually, then the weights have to be decreased. Remember: The window is open if the value of outputNeuron=1. It should have been -1, otherwise the window would not now have to be manually closed. If the value of outputNeuron=1, there are three possibilities:

- ir1=1 and ir2=-1
- ir1=-1 and ir2=1
- ir1=1 and ir2=1

In the first two cases, the “faulty” neuron is identified in the intermediate layer (the one yielding the value of 1) and either of the weight pairs ws1/wl1 or ws2/wl2 have to be decreased in accordance with the input value S or L (if S or L is negative, the weights have to be increased, which occurs implicitly through the multiplication of S or L with the learning rate). In the third case, it is not clear which is the “faulty” neuron, which is why one of the two neurons are selected at random and the weights of the first layer of this neuron are changed.
The process for increasing the weights is similar.

If the window is closed and is opened manually, then the weights have to be increased. Remember: The window is closed if the value of outputNeuron = -1. It should have been 1, otherwise the window would not now have to be manually opened.

However, the value of outputNeuron can only be -1 if both neurons in the intermediate layer yield the value -1, that is, ir1 = ir2 = -1. The “faulty” neuron cannot be identified. Thus, one of the two neurons is selected at random. The weights of the first layer are increased for this neuron. (If S or L is negative, the weights have to be decreased, which occurs implicitly through the multiplication of S or L with the learning rate).



```
reduce the weight
if ir1 = 1 and ir2 = -1
    change wl1 by -1 * learning rate * L
    change ws1 by -1 * learning rate * S
if ir1 = -1 and ir2 = 1
    change wl2 by -1 * learning rate * L
    change ws2 by -1 * learning rate * S
if ir1 = 1 and ir2 = 1
    if pick random 1 to 2 = 1
        change wl1 by -1 * learning rate * L
        change ws1 by -1 * learning rate * S
    else
        change wl2 by -1 * learning rate * L
        change ws2 by -1 * learning rate * S
```

```
lift the weight
if pick random 1 to 2 = 1
    change wl1 by learning rate * L
    change ws1 by learning rate * S
else
    change wl2 by learning rate * L
    change ws2 by learning rate * S
```

Fig. 7. Code “adjusting weights”

5 Learning aims

Our learning scenario describes an introduction to the functionality of artificial neural networks. Our neural networks implement simple logical functions, in accordance with [10][11].

Using a motivating example scenario with everyday relevance and the opportunity for a haptic, enactive experience with technical systems, the pupils learn the following:

- The pupils can describe and implement the functionality of a perceptron
- The pupils can name the limitations of a perceptron
- The pupils can describe and implement the functionality of a small, multi-layer network
- They recognise that the implementation of neural networks represents a deterministic algorithm, which they are familiar with
- The pupils can understand and reflects on the adjustment of weights of a neural network through manual feedback
- They recognise that due to a training phase, particular parameters of the algorithm during the runtime are adjusted, and the algorithm can thus be configured differently
- They recognise that the parameter adjustment (depending on feedback) can be different, and therefore future results of the algorithm are in accordance with the parameter settings
- They can identify a previously unfamiliar problem-solving strategy, which is that one and the same algorithm is used for different applications, in that it is trained by the adjusting of parameters and thus configured
- The pupils recognise that the algorithm does not achieve a level of consciousness or similar, such as might be assumed for "artificial intelligence".

In this article, there is no survey of learning groups before and after the lesson to confirm the learning aims. This is due to the current Corona crisis and will be carried out as soon as possible.

6 Didactic categorisation, conclusion and outlook

Our learning scenario is intended to offer an introduction to the basic concepts of neural networks. As such, it aims to show that the "learning" of neural networks is equivalent, among other things, to the adjustment of particular parameters (here: the weights) during the runtime on the basis of feedback.

The author views the benefits of physical computing to include an enactive dealing with informatics and the creation of operable, impressive products, even if these are not particularly complex in informatics terms. In this way, even less able pupils can be successful. At the same time, the Smart Home offers a learning environment that is motivating and relatable to the lives of the pupils. With many small functions (automatic window opening, automatic ventilation control, …) that can be integrated into a Smart Home, pupils can choose their own tasks, which increases their pride in the later product [5]. The pupils experience themselves as self-efficient and constructive. In addition, the Smart Home offers many different implementation options (as described in Chapter 1), which can also be creatively implemented without neural networks. Within a framework of internal differentiation, even weaker pupils can achieve results that can be carried out and presented. Even if neural networks are integrated, a simpler implementation can include only the application of perceptrons [10]. This also leads to operable, functioning products (Smart Homes), but is not quite as comprehensive and complex as multi-layer neural networks in terms of programming. This aspect also contributes to internal differentiation. For heterogeneous learning groups, there are the following complexity levels that all deliver impressive products: At the first three levels, the functionality is implemented directly:

- A pure activation of motors (open window) without consideration of sensors
- The activation of motors dependent on sensor values (if it gets hot, the window opens)
- The activation of motors dependent on links between sensor values (if it is hot and quiet, the window opens)

At the next levels, the system "learns" from the behaviour of the user:

- The implementation of a perceptron with a change of weights if the user controls it manually (training the perceptron so that the window opens when it is dark and quiet)
- The implementation of a multi-layer network with a change of weights if the user controls it manually (training the network so that the window opens if it is dark and quiet or light and noisy)

In this article, the focus is on neural networks that are very small, but nonetheless meaningful to pupils. On the basis of these small networks, the basic functionality can be taught.

References

1. Strecker, K. (2009): Informatik für Alle - wie viel Programmierung braucht der Mensch. Dissertation. University of Göttingen.
2. Modrow, E. & Strecker, K. (2011): PuMa II. LOG IN: Vol. 31, No. 1. LOG IN Verlag.
3. Przybylla, M. & Romeike, R. (2013): Physical Computing im Informatikunterricht. In: Breier, N., Stechert, P. & Wilke, T. (Hrsg.), INFOS 2013: Informatik erweitert Horizonte - 15. GI-Fachtagung Informatik und Schule. Bonn: Gesellschaft für Informatik e.V.
4. Gramm, A., (2014). Portfolioarbeit im Informatikunterricht. LOG IN: Vol. 34, No. 1. LOG IN Verlag
5. Modrow E. & Strecker, K. (2016): Didaktik der Informatik. DeGruyter-Oldenbourg Verlag
6. LEGO Homepage, https://www.lego.com/de-de/categories/robots-for-kids, last accessed 2020/09/07.
7. Enchanting Homepage, https://enchanting.robotclub.ab.ca/tiki-index.php, last accessed 2020/09/07.
8. Rashid, T. (2017): Neuronale Netze selbst programmieren. Ein verständlicher Einstieg mit Python. O'Reilly
9. Strecker, K. (2020). Robby lernt – aber nicht alles! Eine Einführung in die Funktionsweise von Perzeptren. LOG IN: Vol. 40, No. 1. Berlin: LOG IN Verlag
10. Minsky, M. & Papert, S. (1969): Perceptrons: An Introduction to Computational Geometry. MIT Press.
11. Mehrotra, K., Mohan, C. & Ranka, S. (1996): Elements of Artificial Neural Networks. MIT Press.
12. Rojas, R. (1996): Neural Networks - A Systematic Introduction. Springer-Verlag
13. Goodfellow, I., Bengio, Y. & Courville, A. (2016): Deep Learning. MIT Press