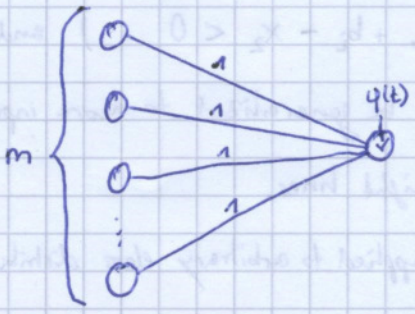
Sheet 2

exercise 1.1)

1. logical or :

To perform a logical OR operation on an input vector $z \in \{0,1\}^m$, we choose the weight vector $w \in \mathbb{R}^m$ to be a vector of 1s ($w = (\underbrace{1,1,1,\dots,1,1}_{m})$) and the activation function to be a step function

$$\varphi(t) = \begin{cases} 0 & \text{if } t < 0.5 \\ 1 & \text{if } t \geq 0.5 \end{cases}$$



2. check if input $z \in \{0,1\}^m$ exactly coincides with $c \in \{0,1\}^m$

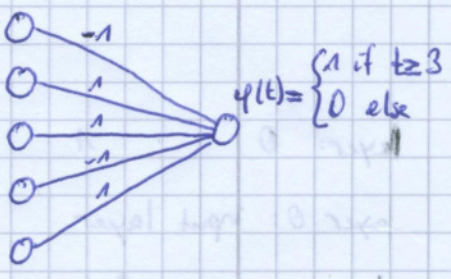
Here we choose our weight vector $w \in \mathbb{R}^m$ to be

$$w_i = \begin{cases} 1 & \text{if } c_i = 1 \\ -1 & \text{if } c_i = 0 \end{cases} \quad \text{or} \quad w = 2c - 1$$

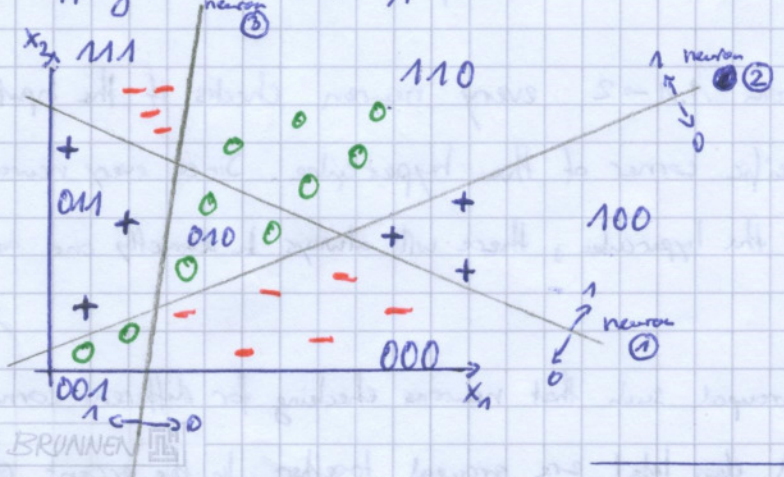
and again a step function as activation

$$\varphi(t) = \begin{cases} 1 & \text{if } t \geq \|c\|_1 \\ 0 & \text{else} \end{cases} \quad \text{with the 1-norm } \|.\|_1$$

For $c = (0,1,1,0,1)$, our network e.g. looks like:



3. mapping to corners of hypercube



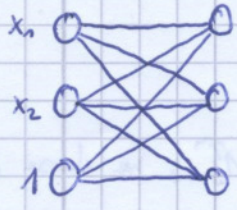
The decision boundary of each neuron can be described by a straight line

$$f_i(x_1) = a_i x_1 + b_i .$$

Given the parameters a_i, b_i $(i = 1,2,3)$ the neural network which

Note that input data can not be mapped to the corner 101 for this choice of decision boundaries.

maps the input data to the corner of a hypercube looks like



with the weight matrix $W = \begin{pmatrix} a_1 & -1 & b_1 \\ a_2 & -1 & b_2 \\ a_3 & -1 & b_3 \end{pmatrix}$

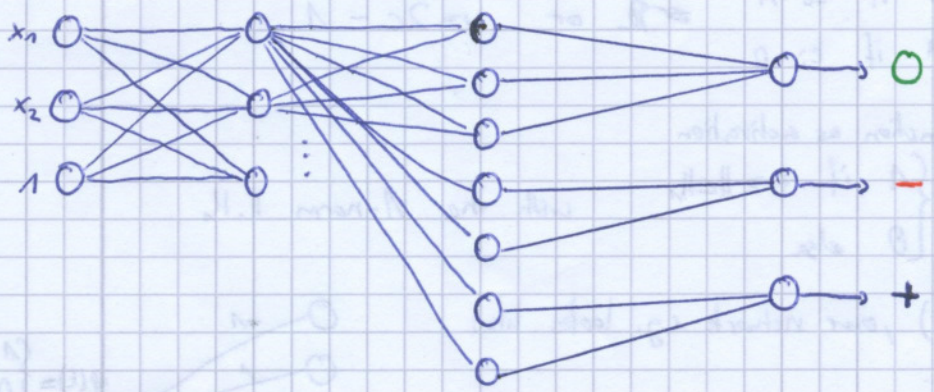
and the activationfunction $\varphi(t) = \begin{cases} 1 & \text{if } t \leq 0 \\ 0 & \text{if } t > 0 \end{cases}$

In words, that means, that a neuron outputs "1" if the input is above its decision boundary ($a_i \cdot x_1 + b_i - x_2 < 0$) and "0" if the input is below.

This can easily be generalized to more input dimensions by using hyperplanes as decision boundaries instead of straight lines.

It can also be applied to arbitrary class distributions by using more decision boundaries, i.e. more neurons.

exercise 1.2) For the given example, the 3-layer network would look as follows:



layer: 0 1 2 3

layer 0: input layer

layer 1: maps the input data to the corners of a hypercube (according to 1.1→3). The number of neurons is given by the required number of decision boundaries to separate the data.

layer 2: using the architecture from 1.1→2 every neuron checks if the input has been mapped to a specific corner of the hypercube. Since every neuron checks for a different corner of the hypercube, there will always be exactly one active neuron in this layer.

The neurons should be grouped such that neurons checking for different corners with the same associated class label are grouped together. In the present case,

the first 3 neurons check for $(0,0,1)$, $(0,1,0)$, $(1,1,0)$, neurons 4,5 check for $(1,1,1)$, $(0,0,0)$ and neurons 6,7 check for $(0,1,1)$, $(1,0,0)$.

In general, the number of neurons in this layer is given by $H_2 = 2^{H_1}$.

In this case, we only have $H_2 = 7 = 2^{H_1} - 1$, since the corner $(1,0,1)$ "does not exist".

layer 3: In this layer, there is one neuron for each class label (here for $-, 0, +$).

~~and each neuron checked~~ Each of these neurons is then only connected to the neurons of the previous layer which checked for corners of the hypercube associated with the corresponding class label.

Using the logical OR from $1.1 \to 1$, each neuron then detects if one of its preneurons in layer 2 is active, i.e. if the data has been mapped to a corner which is associated with its class label ~~and then responds~~ and responds a "1" in this case and a "0" else.

problems)

- the number of neurons in layer 2 grows exponentially with the number of required decision boundaries
- the generalization to an independent test set only works, if the data are grouped in the same way as the training data, i.e. the test set must have the same decision boundaries

  → the network tends to overfitting