Fundamentals of Machine Learning
Exercise 02

Johannes Sindlinger, Daniel Knorr, Marvin Hanf

2023/05/22

1 Hand-Crafted Network

First, we start by building the single neurons as specified in the exercise. One could also add the critical point $z' = 0$ to the value of 1, which would need adjusted bias. For the general model described later, it doesn't make any differences, since the decision planes are the same.

1. logical OR:
   Weights $w = [1]^D$
   Bias $b = 0$
   $z' = w \times z - b$
   Activation Function:
   $$\varphi(z') = \begin{cases} 1 & \text{if } z' > 0 \\ 0 & \text{else} \end{cases}$$
2. masked logical OR:
   Weights $w = c$
   Bias $b = 0$
   $z' = c \times z - 0$
   Activation Function:
   $$\varphi(z') = \begin{cases} 1 & \text{if } z' > 0 \\ 0 & \text{else} \end{cases}$$
3. perfect match:
   Weights $w = c$
   Bias $b = sum(z)$
   $z' = c \times z - sum(z)$
   Activation Function:
   $$\varphi(z') = \begin{cases} 1 & \text{if } z' = 0 \\ 0 & \text{else} \end{cases}$$

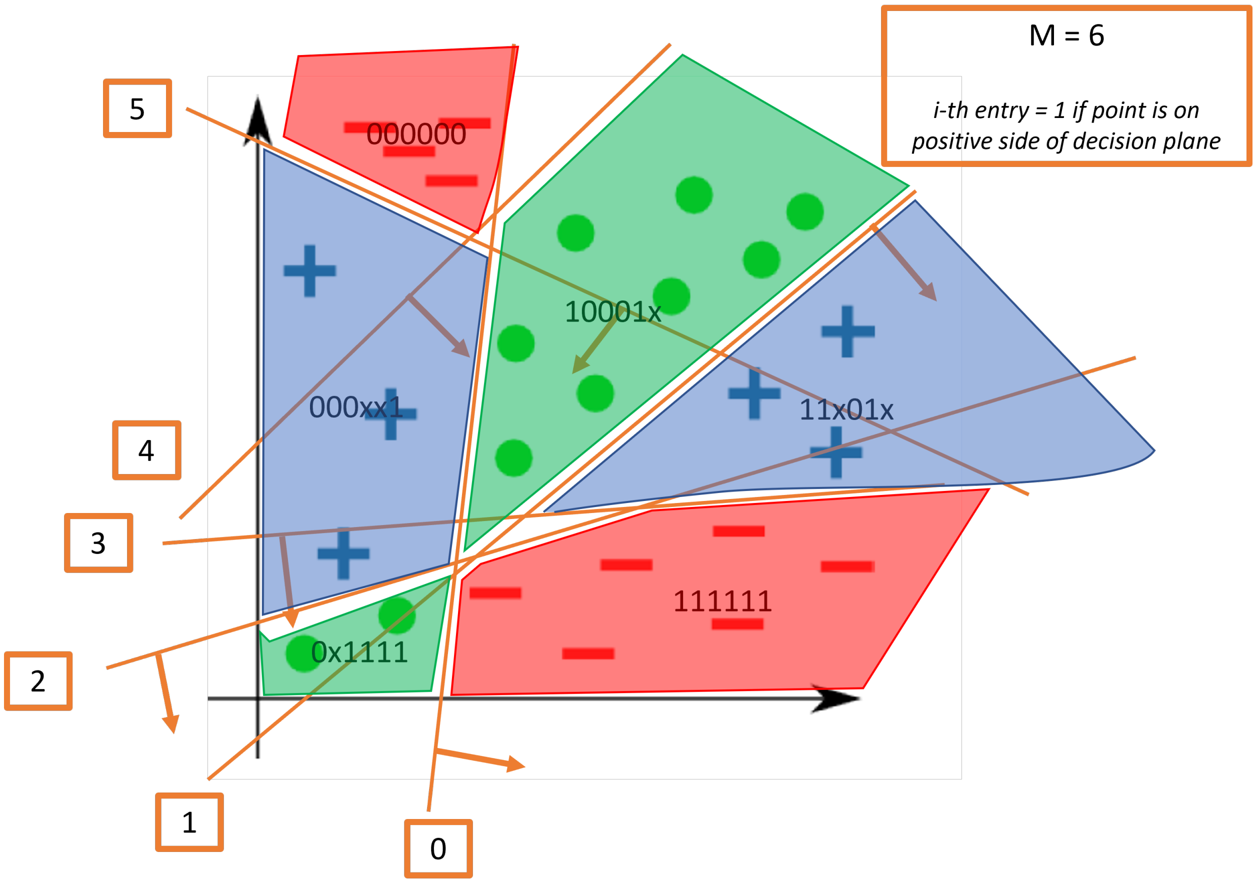
M = 6
i-th entry = 1 if point is on positive side of decision plane
5
4
3
2
1
0
000000
10001x
000xx1
11x01x
111111
0x1111

Figure 1: Decision regions

One might build a network based on these neurons by the following. Figure 2 illustrates the three layer network:

1. The first layer spreads the different regions of the vector space towards a higher dimensional vector space, concretely into points of a cube in the space $\mathbb{R}^M$. Looking at Figure 1 one might obtain that the region of the four red minus at the top left are mapped to the vector $(000000)^T$. This step is done by selecting the weights according to the lines, which are drawn in Figure 1. So overall, the combination of all the lines provide a unique assignment of the different clusters. As one also can see, some clusters are spreading across multiple regions, which results in "don't care"-values. These affect the third layer and masked logical OR step.

2. The second layer maps the binary vector representation to a unique identifier. Overall, there are 2^M different possible vectors, so the second layers contains 2^M neurons. Each neuron performs perfect matching, so we get for each of the 2^M options the information whether the corresponding region in Figure 1 is currently hit.

3. The last layer contains three neurons which produce one-hot encoding of the correct

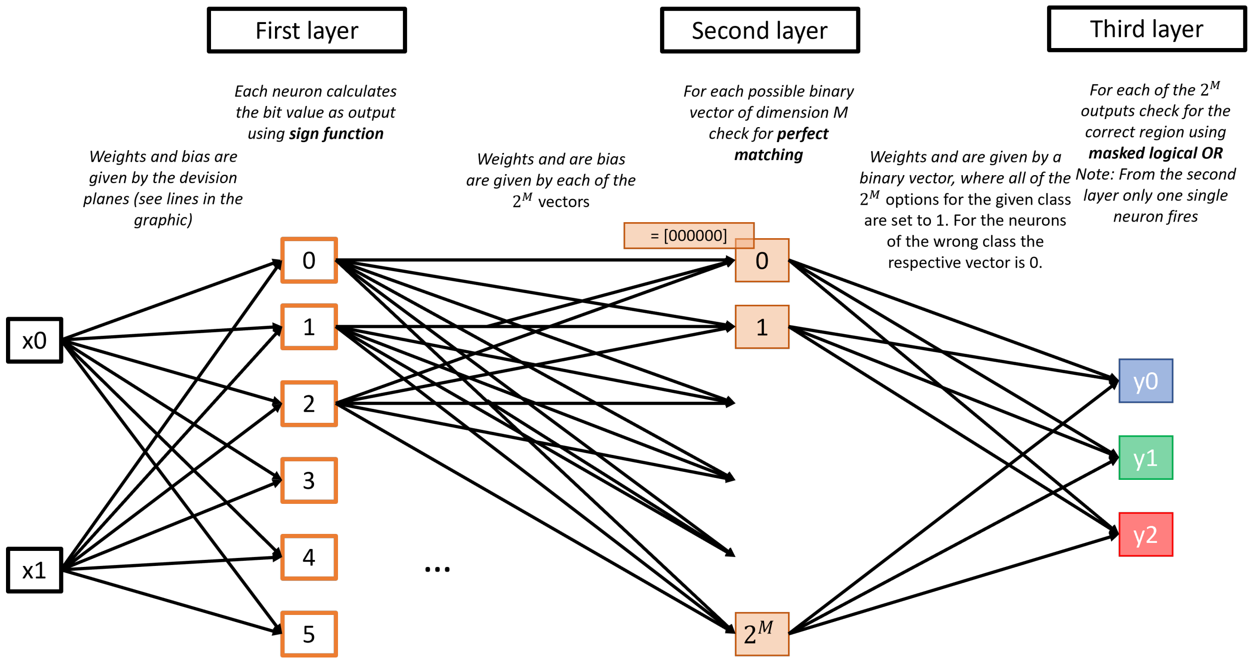
First layer
Second layer
Third layer
Each neuron calculates the bit value as output using **sign function**
Weights and bias are given by the devision planes (see lines in the graphic)
Weights and are bias are given by each of the 2^M vectors
For each possible binary vector of dimension M check for **perfect matching**
Weights and are given by a binary vector, where all of the 2^M options for the given class are set to 1. For the neurons of the wrong class the respective vector is 0.
For each of the 2^M outputs check for the correct region using **masked logical OR**
Note: From the second layer only one single neuron fires
= [000000]
x0
x1
0
1
2
3
4
5
...
2^M
y0
y1
y2

Figure 2: Three layer network

category. Each neuron corresponds to one of the categories of the model. The final mapping is done by connecting all the 2^M neurons of the second layer with each output neuron. Always exactly one of the 2^M neurons in the second layer will be fired, so there will emerge a 2^M vector with a single 1 entry. This output vector of the second layer will be checked against another 2^M which contains entries of 1's for all corresponding vectors of the original regions, but only for the correct output label. Don't care values will be considered for both values 0 and 1. For the output neurons of the wrong categories, the vector to compare is set to 0, so they will never be fired.

The described procedure generally works for infinite dimensions of the input and different distributions. However, since the described mechanism highly depends on the number of decision regions / clusters within the data, the efficiency decreases drastically. Especially considering the dimension of 2^M neurons within layer two makes this procedure unpractical and endless to learn.

Furthermore, this method results in overfitting of the data. As one can see in Figure 1, the procedure draws really specific lines between each cluster, which might not generalize at all. If we put some unseen data in this, the probability of getting the right classification is quite low, especially when increasing dimensions and the disordering of clusters.

2 Linear Activation Function

Assuming we have a neural network with depth $L > 1$ and linear activation function, e.g. the identity function. The for the final layer it yields:

$$y_l = Z'_{l-1} \cdot B_{l-1} + b_{l-1}$$

where Z_{l-1} is the weight matrix, B_{l-1} is the output of the penultimate layer and b_{l-1} is the bias vector. Since we have the identity as activation function, we can recursively reformulate the equation.

$$y_l = (Z_{l-1} \cdot (Z_{l-2} \cdot ... \cdot (Z_1 \cdot Z_0 + b_1) + ... + b_{l-2}) + b_{l-1}$$

Reformulating gives us:

$$y_l = (Z_{l-1} \cdot ... \cdot Z_1) \cdot Z_0 + (Z_{l-1} \cdot ... \cdot Z_2 \cdot b_1 + Z_{l-1} \cdot ... \cdot Z_3 \cdot b_2 + ... + b_{l-1})$$

In overall, this is also a linear function of Z_0, so we can combine the weights of $Z_1 ... Z_{l-1}$ and $b_1 ... b_{l-1}$ and get the following.

$$Z' = Z_{l-1} \cdot ... \cdot Z_1$$
$$b' = Z_{l-1} \cdot ... \cdot Z_2 \cdot b_1 + Z_{l-1} \cdot ... \cdot Z_3 \cdot b_2 + ... + b_l$$

Finally, we get as a single linear model:

$$y_l = Z' \cdot Z_0 + b'$$