Moderne Methoden der Datenanalyse

Depth and activation?

Roger Wolf

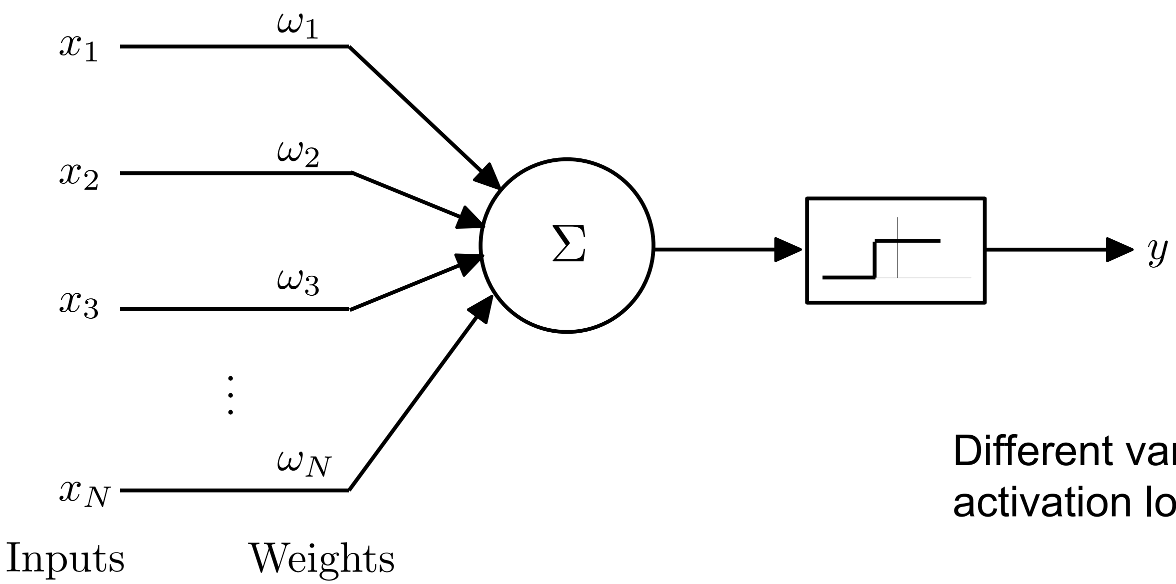
30. June 2023

Content of this lecture

- Recap: perceptron → multilayer perceptron (MLP).
- MLP as representation of Boolean functions.
- MLP as representation of arbitrary contours.
- MLP training:
    - Perceptron learning rule.
    - Not linearly separable tasks and activation functions – revisted.

Recap: perceptron

- Last time we discussed the Boolean perceptron with a simple step function as **activation function** (here shown for real-valued *input features*):



$x_1 \dots x_N \in \mathbb{R}$

$\omega_1 \dots \omega_N \in \mathbb{R}$

Unit fires, if $\sum_i \omega_i x_i \geq T$

Different variations to express the activation logic:

$$\sum_i \omega_i x_i \geq T\,,$$

$$\sum_i \omega_i x_i - T \geq 0\,,$$

$$\theta\left(\sum_i \omega_i x_i - T\right) \quad \text{(Heavyside function)}$$

Recap: activation function

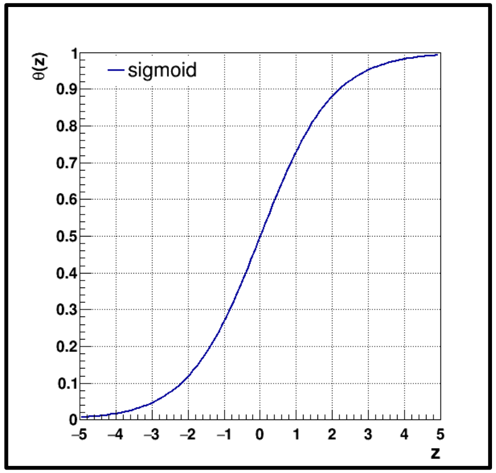
- We discussed that the activation function could be any function.
- A few popular examples are given below:

Rectified linear unit (**ReLU**):

$$\theta(z) = \begin{cases} z & z \geq 0 \\ 0 & \mathrm{sonst} \end{cases}$$

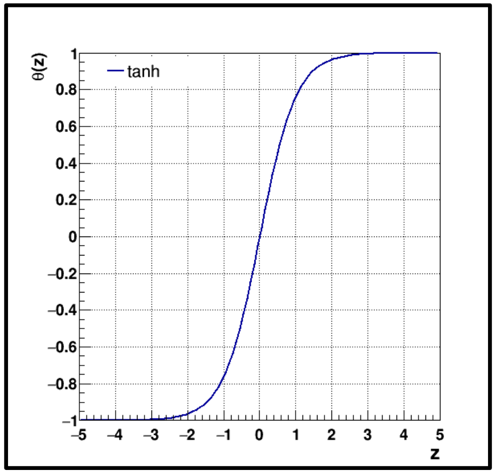
Sigmoid:

$$\theta(z) = \frac{1}{1 + \exp(-z)}$$



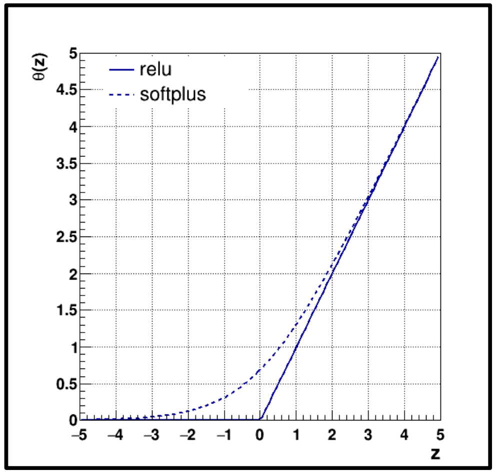
tanh:

$$\theta(z) = \tanh(z)$$



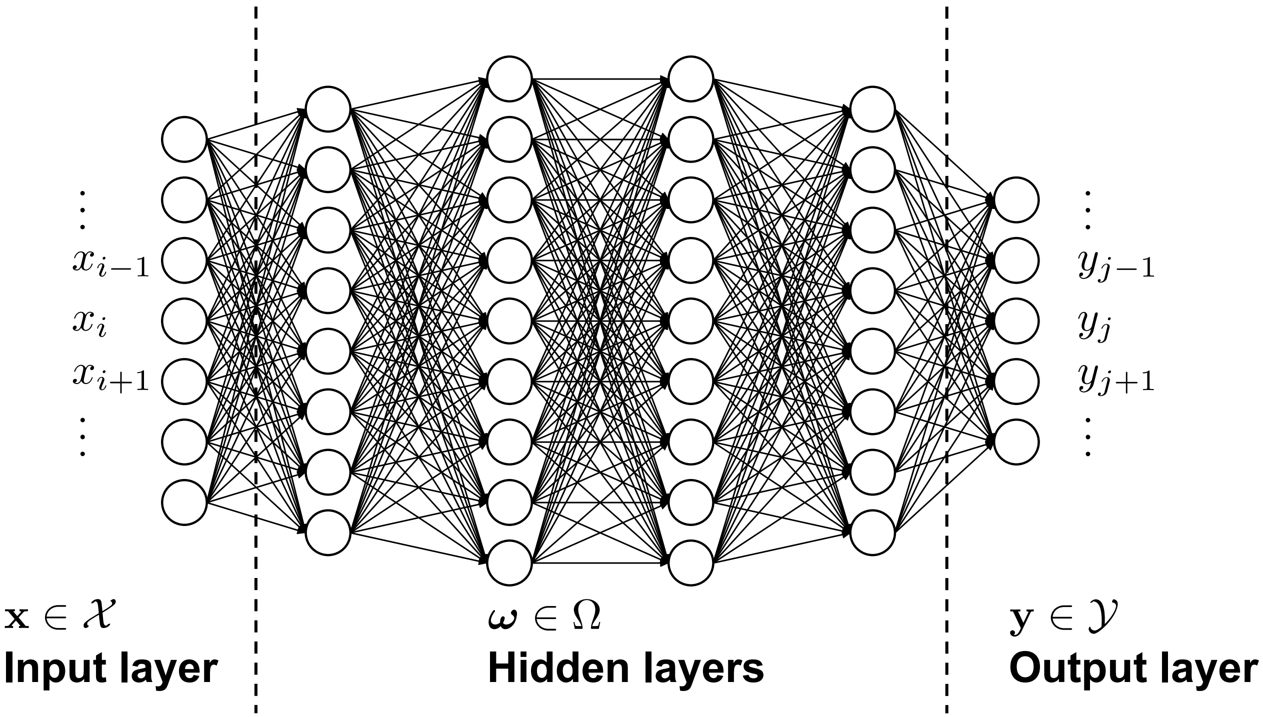
Softplus:

$$\theta(z) = \log(1 + \exp(z))$$



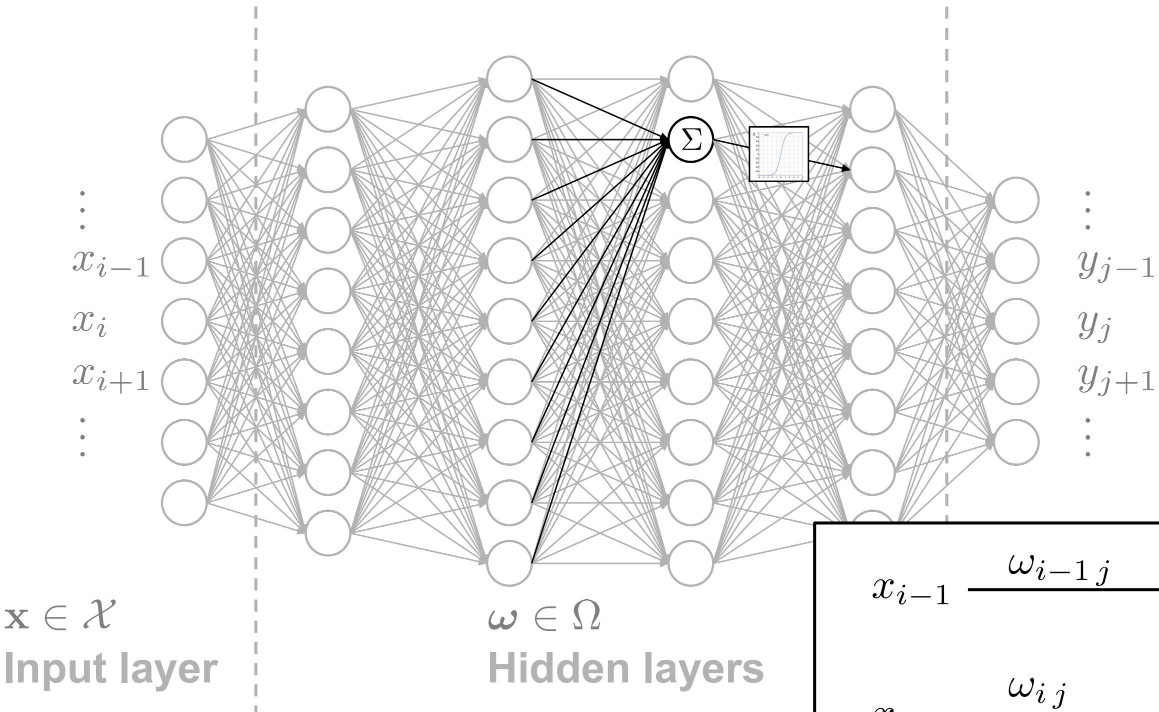
Recap: multilayer perceptron (MLP)

- We have discussed that the ability to express Boolean relations increases when using more than one perceptron, organized in layers → **multilayer perceptron (MLP)**:

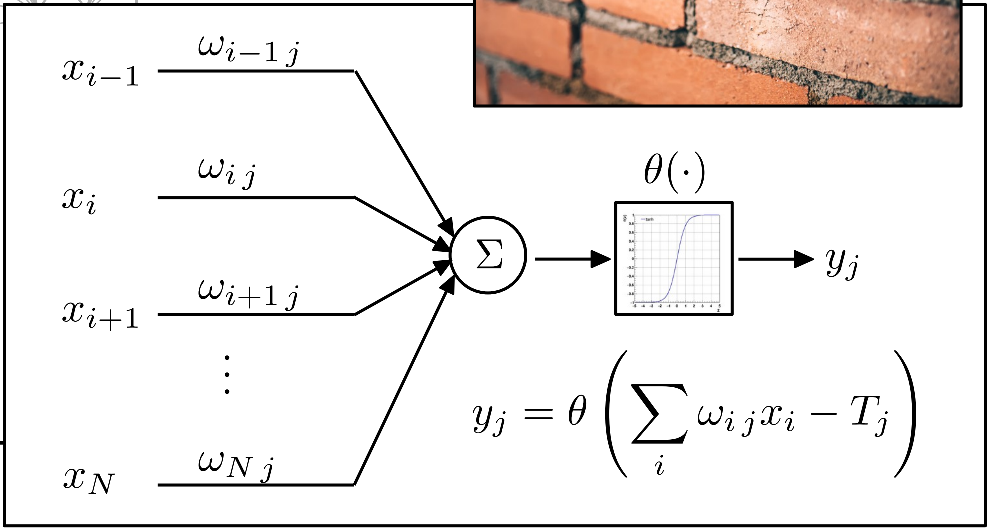


The fully-connected feed-forward NN

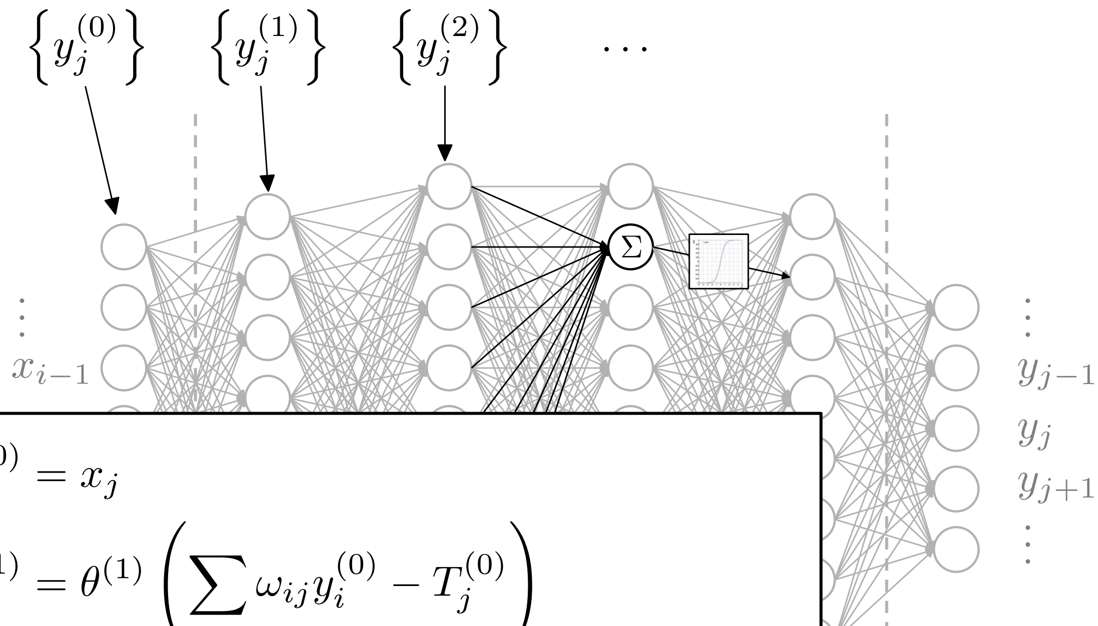
- **Fully-connected**: All nodes of consecutive layers are *connected* with each other.
- **Feed-forward**: Inputs are propagated only in *forward* direction.



Building blocks are single perceptrons (also called **nodes**):



Mathematical notation



$$y_j^{(0)} = x_j$$

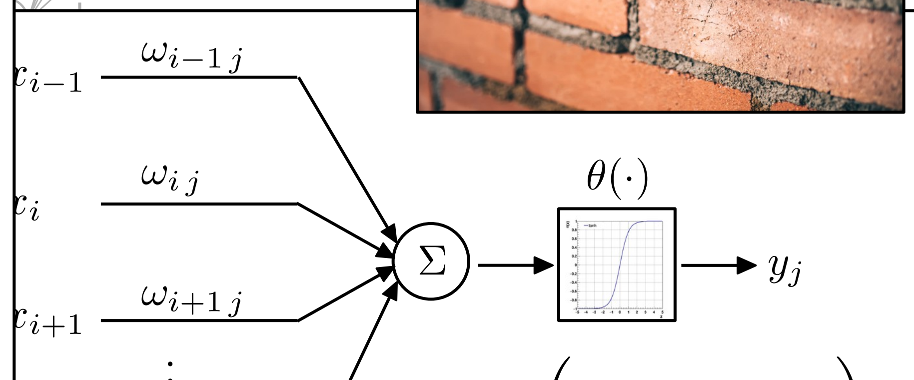
$$y_j^{(1)} = \theta^{(1)}\left(\sum_i \omega_{ij} y_i^{(0)} - T_j^{(0)}\right)$$

$$y_j^{(2)} = \theta^{(2)}\left(\sum_i \omega_{ij} y_i^{(1)} - T_j^{(1)}\right)$$

$$\vdots$$

$$y_j^{(k)} = \theta^{(k)}\left(z_j^{(k)}\right) \qquad z_j^{(k)} = \sum_i \omega_{ij} y_i^{(k-1)} - T_j$$

Building blocks are single perceptrons (also called **nodes**):



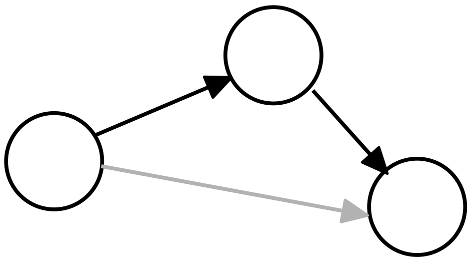
$$y_j = \theta\left(\sum_i \omega_{i\,j} x_i - T_j\right)$$

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

Depth

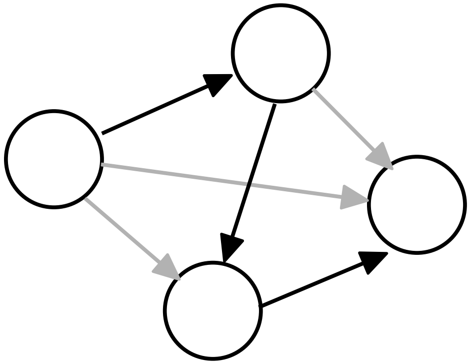
- A feed-forward NN can be understood as a **directed graph** of depth d.
- A directed graph has *sources* and *drains*. The depth of a graph is the longest path between a source and a drain.

Example 1:



Depth? – 2

Example 2:



Depth? – 3

- An NN with a depth of $d > 2$ (i.e. an NN with more than 2 hidden layers) we call *deep*.

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

The MLP as representation of Boolean functions

- Last time we have discussed that an MLP can represent *any arbitrary Boolean function*.
  How many hidden layers does the MLP minimally require to have this quality?

The MLP as representation of Boolean functions

- Last time we have discussed that an MLP can represent *any arbitrary Boolean function*.
  How many hidden layers does the MLP minimally require to have this quality? – 1.

The MLP as representation of Boolean functions

- Last time we have discussed that an MLP can represent *<u>any arbitrary Boolean function</u>*. How many hidden layers does the MLP minimally require to have this quality? – 1.

- **Proof**: any Boolean function can be expressed in the form of a **truth table**.

x_1	x_2	x_3	x_4	x_5	y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Arbitrary example

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

The MLP as representation of Boolean functions

- Last time we have discussed that an MLP can represent *any arbitrary Boolean function*. How many hidden layers does the MLP minimally require to have this quality? – 1.

- **Proof**: any Boolean function can be expressed in the form of a **truth table**.

x_1	x_2	x_3	x_4	x_5	y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Arbitrary example

Disjunctive normal form (DNF):

$$y = \overline{x}_1\overline{x}_2 x_3 x_4 \overline{x}_5 \vee \overline{x}_1 x_2 \overline{x}_3 x_4 x_5 \vee \overline{x}_1 x_2 x_3 \overline{x}_4 \overline{x}_5 \vee$$

$$x_1\overline{x}_2\overline{x}_3\overline{x}_4 x_5 \vee x_1\overline{x}_2 x_3 x_4 x_5 \vee x_1 x_2 \overline{x}_3\overline{x}_4 x_5$$

The MLP as representation of Boolean functions

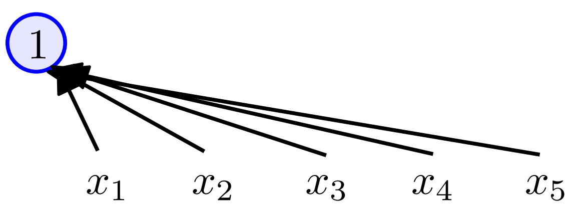
- Last time we have discussed that an MLP can represent *any arbitrary Boolean function*.
  How many hidden layers does the MLP minimally require to have this quality? – 1.

- **Proof**: any Boolean function can be expressed in the form of a **truth table**.

x_1	x_2	x_3	x_4	x_5	y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Arbitrary example

$$y = \underbrace{\overline{x}_1\overline{x}_2 x_3 x_4 \overline{x}_5}_{1} \vee \overline{x}_1 x_2 \overline{x}_3 x_4 x_5 \vee \overline{x}_1 x_2 x_3 \overline{x}_4 \overline{x}_5 \vee$$
$$x_1 \overline{x}_2 \overline{x}_3 \overline{x}_4 x_5 \vee x_1 \overline{x}_2 x_3 x_4 x_5 \vee x_1 x_2 \overline{x}_3 \overline{x}_4 x_5$$



The MLP as representation of Boolean functions

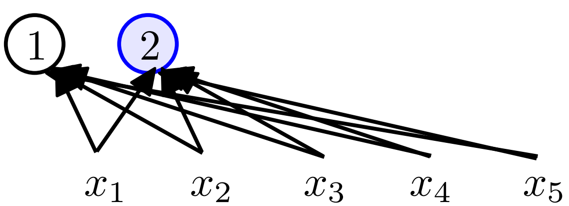
- Last time we have discussed that an MLP can represent *any arbitrary Boolean function*. How many hidden layers does the MLP minimally require to have this quality? – 1.

- **Proof**: any Boolean function can be expressed in the form of a **truth table**.

x_1	x_2	x_3	x_4	x_5	y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Arbitrary example

$$y = \underbrace{\overline{x}_1\overline{x}_2x_3x_4\overline{x}_5}_{1} \vee \underbrace{\overline{x}_1x_2\overline{x}_3x_4x_5}_{2} \vee \overline{x}_1x_2x_3\overline{x}_4\overline{x}_5 \vee$$
$$x_1\overline{x}_2\overline{x}_3\overline{x}_4x_5 \vee x_1\overline{x}_2x_3x_4x_5 \vee x_1x_2\overline{x}_3\overline{x}_4x_5$$



The MLP as representation of Boolean functions

- Last time we have discussed that an MLP can represent *any arbitrary Boolean function*. How many hidden layers does the MLP minimally require to have this quality? – 1.

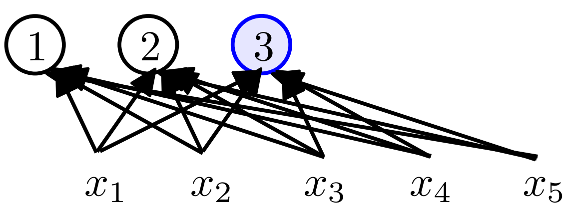
- **Proof**: any Boolean function can be expressed in the form of a **truth table**.

x_1	x_2	x_3	x_4	x_5	y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Arbitrary example

$$y = \underbrace{\overline{x}_1\overline{x}_2x_3x_4\overline{x}_5}_{1} \vee \underbrace{\overline{x}_1x_2\overline{x}_3x_4x_5}_{2} \vee \underbrace{\overline{x}_1x_2x_3\overline{x}_4\overline{x}_5}_{3} \vee$$

$$x_1\overline{x}_2\overline{x}_3\overline{x}_4x_5 \vee x_1\overline{x}_2x_3x_4x_5 \vee x_1x_2\overline{x}_3\overline{x}_4x_5$$



The MLP as representation of Boolean functions

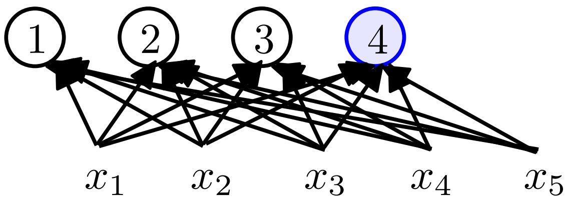
- Last time we have discussed that an MLP can represent *any arbitrary Boolean function*. How many hidden layers does the MLP minimally require to have this quality? – 1.

- **Proof**: any Boolean function can be expressed in the form of a **truth table**.

x_1	x_2	x_3	x_4	x_5	y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Arbitrary example

$$y = \underbrace{\overline{x}_1\overline{x}_2x_3x_4\overline{x}_5}_{1} \vee \underbrace{\overline{x}_1x_2\overline{x}_3x_4x_5}_{2} \vee \underbrace{\overline{x}_1x_2x_3\overline{x}_4\overline{x}_5}_{3} \vee \underbrace{x_1\overline{x}_2\overline{x}_3\overline{x}_4x_5}_{4} \vee x_1\overline{x}_2x_3x_4x_5 \vee x_1x_2\overline{x}_3\overline{x}_4x_5$$



The MLP as representation of Boolean functions

- Last time we have discussed that an MLP can represent *any arbitrary Boolean function*. How many hidden layers does the MLP minimally require to have this quality? – 1.

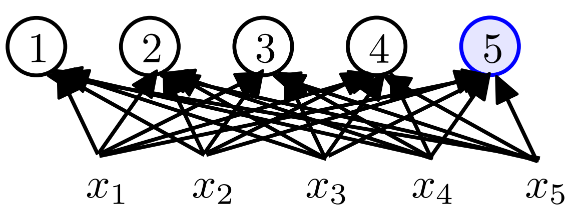
- **Proof**: any Boolean function can be expressed in the form of a **truth table**.

x_1	x_2	x_3	x_4	x_5	y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Arbitrary example

$$y = \underbrace{\overline{x}_1\overline{x}_2x_3x_4\overline{x}_5}_{1} \vee \underbrace{\overline{x}_1x_2\overline{x}_3x_4x_5}_{2} \vee \underbrace{\overline{x}_1x_2x_3\overline{x}_4\overline{x}_5}_{3} \vee$$

$$\underbrace{x_1\overline{x}_2\overline{x}_3\overline{x}_4x_5}_{4} \vee \underbrace{x_1\overline{x}_2x_3x_4x_5}_{5} \vee x_1x_2\overline{x}_3\overline{x}_4x_5$$



The MLP as representation of Boolean functions

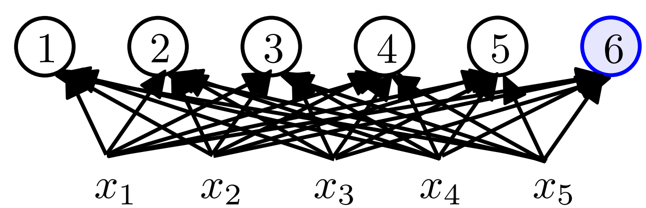
- Last time we have discussed that an MLP can represent *any arbitrary Boolean function*. How many hidden layers does the MLP minimally require to have this quality? – 1.

- **Proof**: any Boolean function can be expressed in the form of a **truth table**.

x_1	x_2	x_3	x_4	x_5	y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Arbitrary example

$$y = \underbrace{\overline{x}_1\overline{x}_2x_3x_4\overline{x}_5}_{1} \vee \underbrace{\overline{x}_1x_2\overline{x}_3x_4x_5}_{2} \vee \underbrace{\overline{x}_1x_2x_3\overline{x}_4\overline{x}_5}_{3} \vee \underbrace{x_1\overline{x}_2\overline{x}_3\overline{x}_4x_5}_{4} \vee \underbrace{x_1\overline{x}_2x_3x_4x_5}_{5} \vee \underbrace{x_1x_2\overline{x}_3\overline{x}_4x_5}_{6}$$



The MLP as representation of Boolean functions

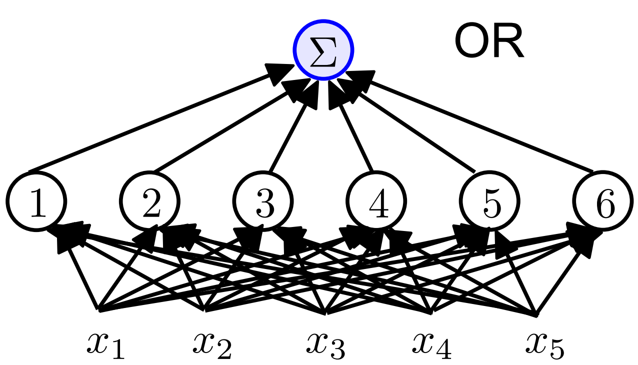
- Last time we have discussed that an MLP can represent *any arbitrary Boolean function*. How many hidden layers does the MLP minimally require to have this quality? – 1.

- **Proof**: any Boolean function can be expressed in the form of a **truth table**.

x_1	x_2	x_3	x_4	x_5	y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Arbitrary example

$$y = \underbrace{\overline{x}_1\overline{x}_2x_3x_4\overline{x}_5}_{1} \vee \underbrace{\overline{x}_1x_2\overline{x}_3x_4x_5}_{2} \vee \underbrace{\overline{x}_1x_2x_3\overline{x}_4\overline{x}_5}_{3} \vee \underbrace{x_1\overline{x}_2\overline{x}_3\overline{x}_4x_5}_{4} \vee \underbrace{x_1\overline{x}_2x_3x_4x_5}_{5} \vee \underbrace{x_1x_2\overline{x}_3\overline{x}_4x_5}_{6}$$



The MLP as representation of Boolean functions

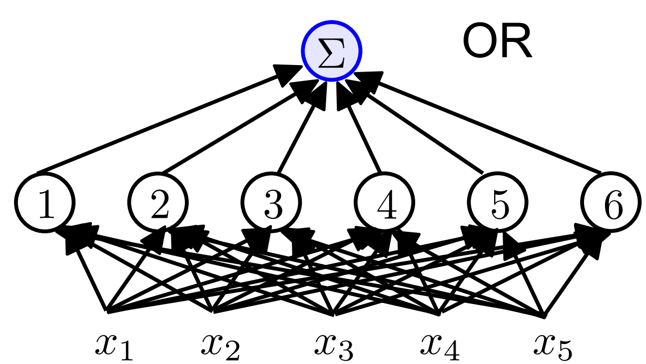
[Without proof] For N inputs the number of required nodes can grow exponentially (exp(N)). Giving the NN more depth, it is possible to reduce the number of nodes down to $\log_2(N)$!

x_1	x_2	x_3	x_4	x_5	y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Arbitrary example

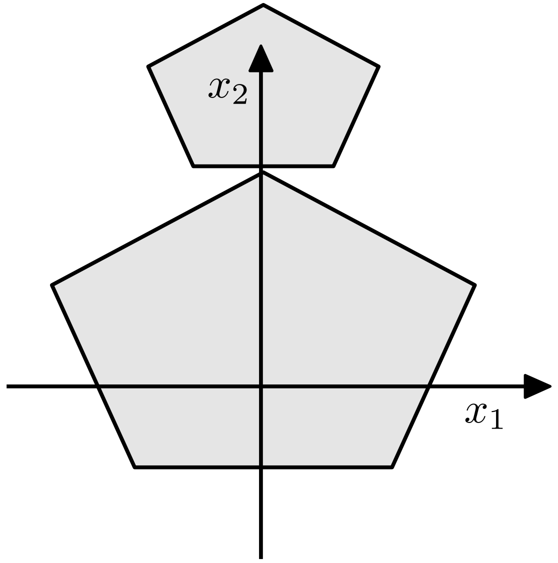
$$y = \underbrace{\overline{x}_1\overline{x}_2x_3x_4\overline{x}_5}_{1} \vee \underbrace{\overline{x}_1x_2\overline{x}_3x_4x_5}_{2} \vee \underbrace{\overline{x}_1x_2x_3\overline{x}_4\overline{x}_5}_{3} \vee$$

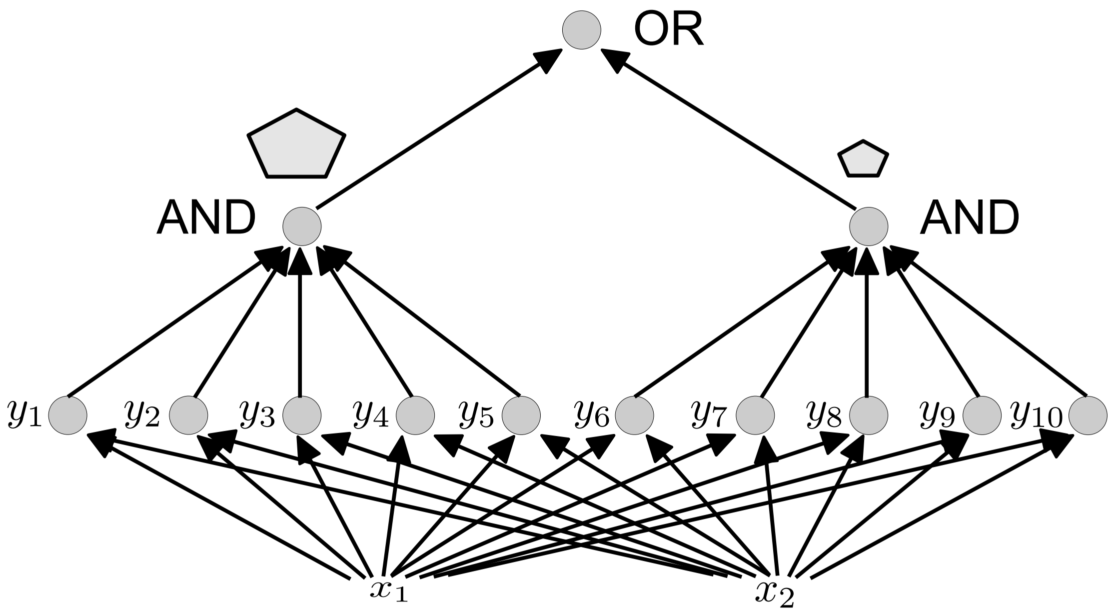
$$\underbrace{x_1\overline{x}_2\overline{x}_3\overline{x}_4x_5}_{4} \vee \underbrace{x_1\overline{x}_2x_3x_4x_5}_{5} \vee \underbrace{x_1x_2\overline{x}_3\overline{x}_4x_5}_{6}$$



The MLP as classifier

- We have discussed that an MLP can approximate any ***arbitrary boundary*** (like the one shown below) with ***abitrary precision***.





- For this we have used two hidden layers.

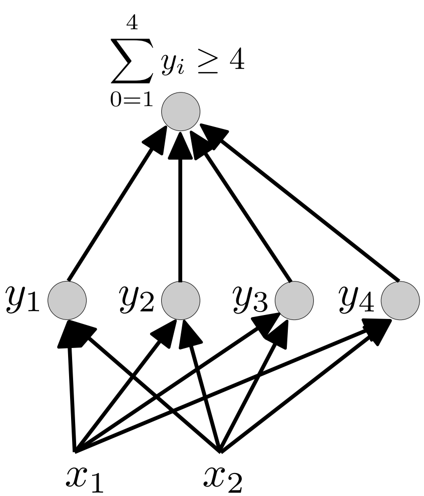
- **Question**: Do you think that this can be done in general[1] with only **one single hidden layer**? If yes – how? If no – why not?

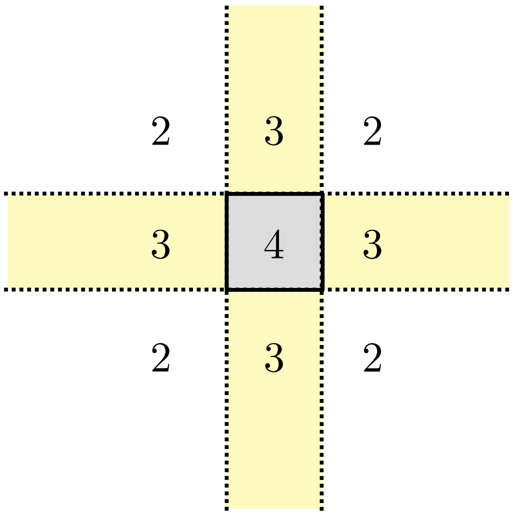
The MLP as classifier

- **Answer**: it is possible, but a general construction is more complicated as you might think.

The MLP as classifier

- **Answer**: it is possible, but a general construction is more complicated as you might think.
- The root cause is the difficulty to represent an arbitrary convex boundary with an MLP:
- A contour with **N=4 bounds** can be represented by an MLP with a single hidden layer as shown (for a single pentagon) e.g. on slide 15.

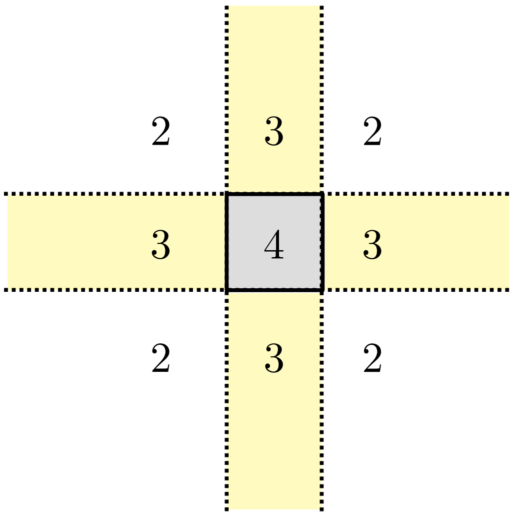




MLP with 4 nodes

The MLP as classifier

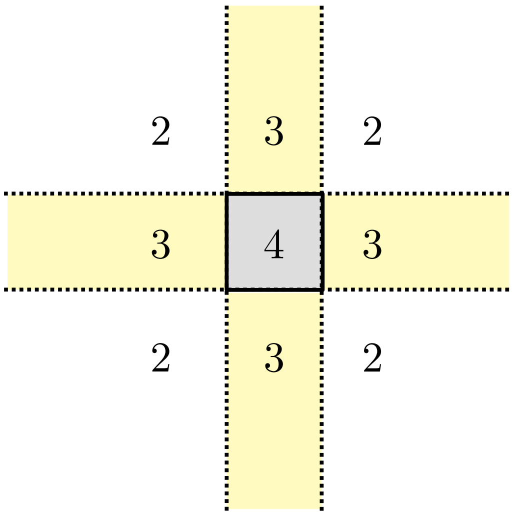
- **Answer**: it is possible, but a general construction is more complicated as you might think.
- The root cause is the difficulty to represent an arbitrary convex boundary with an MLP:
- A contour with **N=4 bounds** can be represented by an MLP with a single hidden layer as shown (for a single pentagon) e.g. on slide 15.
- What prevents us from filling an arbitrary contour with small squares? — there is an unbound area with MLP output $0 < y < N$, i.e. when filling the boundary with squares a distinction between „in- and outside the contour“ is impossible!



MLP with 4 nodes

The MLP as classifier

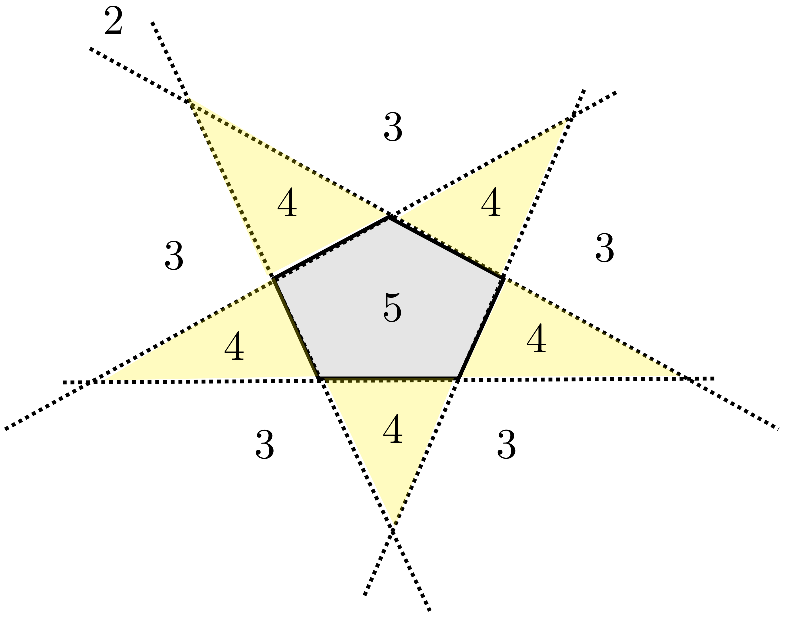
- **Answer**: it is possible, but a general construction is more complicated as you might think.
- The root cause is the difficulty to represent an arbitrary convex boundary with an MLP:
- A contour with **N=4 bounds** can be represented by an MLP with a single hidden layer as shown (for a single pentagon) e.g. on slide 15.
- What prevents us from filling an arbitrary contour with small squares? — there is an unbound area with MLP output $0 < y < N$, i.e. when filling the boundary with squares a distinction between „in- and outside the contour“ is impossible!
- This issue can be mitigated by moving on to more bounds.



MLP with 4 nodes

The MLP as classifier

- **Answer**: it is possible, but a general construction is more complicated as you might think.
- The root cause is the difficulty to represent an arbitrary convex boundary with an MLP:
- A contour with **N=5 bounds** can be represented by an MLP with a single hidden layer as shown (for a single pentagon) e.g. on slide 15.



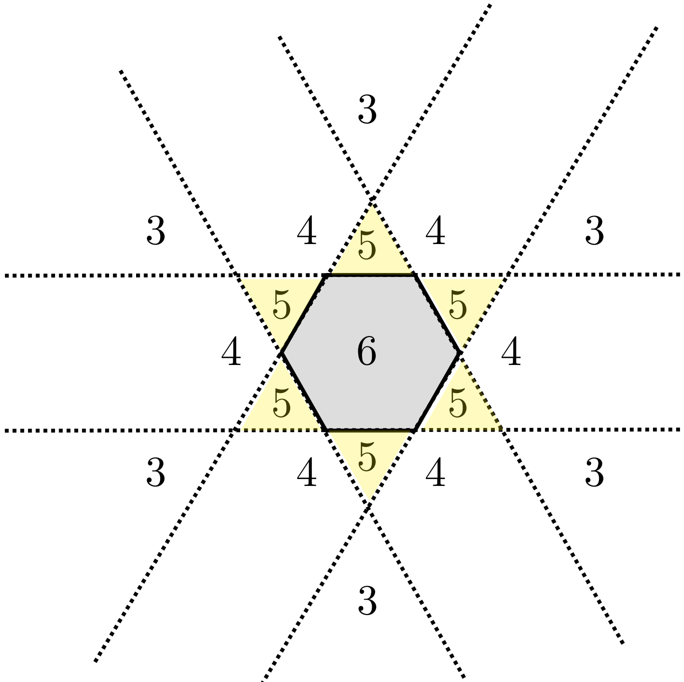
MLP with 5 nodes

The MLP as classifier

- **Answer**: it is possible, but a general construction is more complicated as you might think.

- The root cause is the difficulty to represent an arbitrary convex boundary with an MLP:

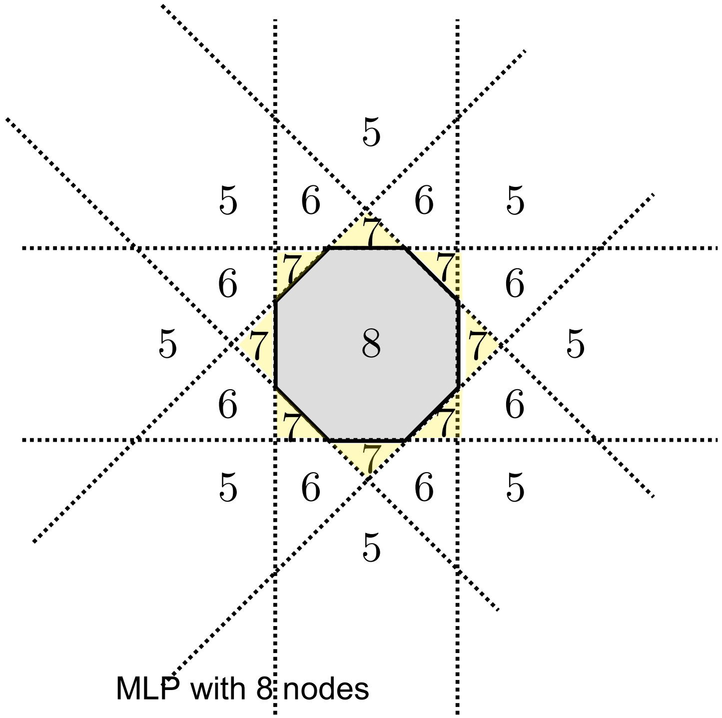
- A contour with **N=6 bounds** can be represented by an MLP with a single hidden layer as shown (for a single pentagon) e.g. on slide 15.



MLP with 6 nodes

The MLP as classifier

- **Answer**: it is possible, but a general construction is more complicated as you might think.
- The root cause is the difficulty to represent an arbitrary convex boundary with an MLP:
- A contour with **N=8 bounds** can be represented by an MLP with a single hidden layer as shown (for a single pentagon) e.g. on slide 15.



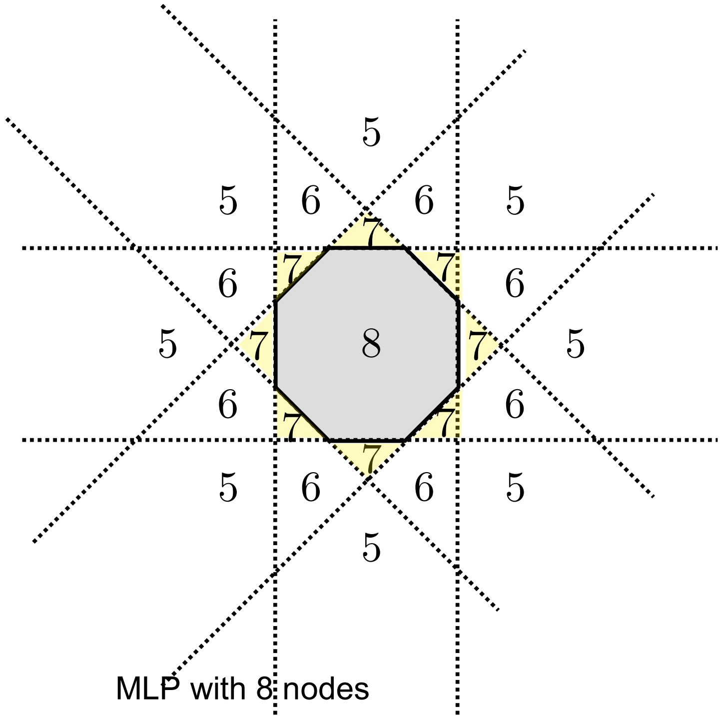
MLP with 8 nodes

The MLP as classifier

- **Answer**: it is possible, but a general construction is more complicated as you might think.

- The root cause is the difficulty to represent an arbitrary convex boundary with an MLP:

- A contour with **N=8 bounds** can be represented by an MLP with a single hidden layer as shown (for a single pentagon) e.g. on slide 15.



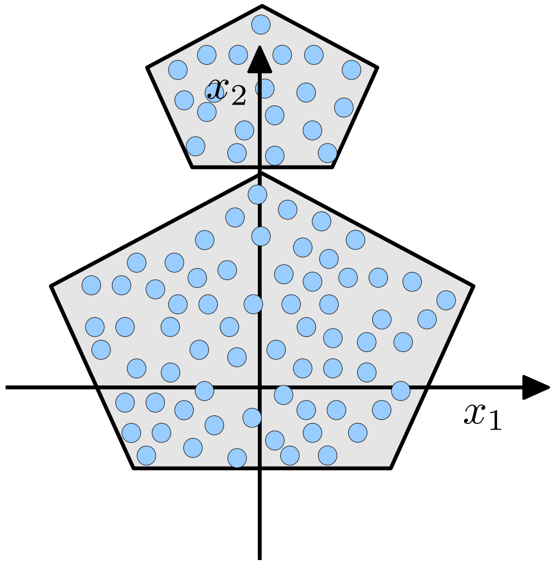
- For $N \rightarrow \infty$ bounds this procedure results in a cylinder, inside of which y can be normalized to a *const.*, while outside it will drop to 0 exponentially.

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

The MLP as classifier

- **Answer**: it is possible, but a general construction is more complicated as you might think.

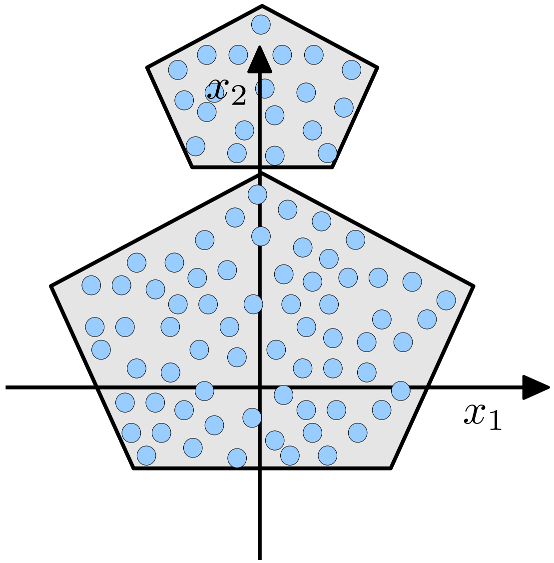
- In this way any arbitrary contour can be approximated with arbitrary precision, with an MLP with a single hidden layer by filling the contours with cylinders. The perceptrons of the hidden layer are added in the output layer as demonstrated, e.g., on slide 15:



The MLP as classifier

- **Answer**: it is possible, but a general construction is more complicated as you might think.

- In this way any arbitrary contour can be approximated with arbitrary precision, with an MLP with a single hidden layer by filling the contours with cylinders. The perceptrons of the hidden layer are added in the output layer as demonstrated, e.g., on slide 15:

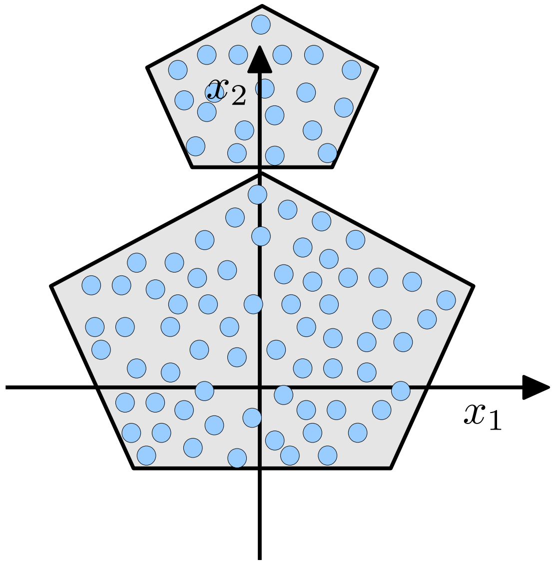


- **[-]** An MLP with a single hidden layer may require an infinite number of nodes.

The MLP as classifier

- **Answer**: it is possible, but a general construction is more complicated as you might think.

- In this way any arbitrary contour can be approximated with arbitrary precision, with an MLP with a single hidden layer by filling the contours with cylinders. The perceptrons of the hidden layer are added in the output layer as demonstrated, e.g., on slide 15:



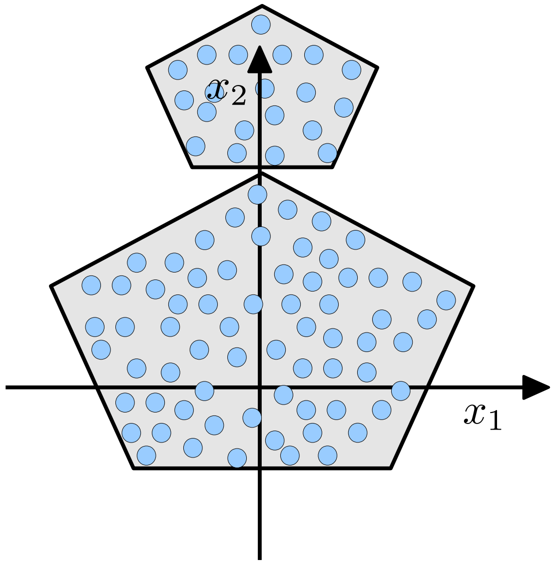
- **[-]** An MLP with a single hidden layer may require an infinite number of nodes.

- **[+]** Also here depth can allow a significant reduction of the required nodes[2]. How many hidden nodes for the example on the left?

The MLP as classifier

- **Answer**: it is possible, but a general construction is more complicated as you might think.

- In this way any arbitrary contour can be approximated with arbitrary precision, with an MLP with a single hidden layer by filling the contours with cylinders. The perceptrons of the hidden layer are added in the output layer as demonstrated, e.g., on slide 15:



- **[-]** An MLP with a single hidden layer may require an infinite number of nodes.

- **[+]** Also here depth can allow a significant reduction of the required nodes[2]. How many hidden nodes for the example on the left? – **12**: 10 in the first + 2 in the second hidden layer.

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

Summary

- **PRO**: It is possible to approximate any arbitrary **Boolean function** (slides 9–14), **boundary** (slides 15–19), or **real-valued function** (equivalent to slides 15–19) to arbitrary precision with an _MLP with only one single hidden layer_.

Summary

- **PRO**: It is possible to approximate any arbitrary **Boolean function** (slides 9–14), **boundary** (slides 15–19), or **real-valued function** (equivalent to slides 15–19) to arbitrary precision with an *MLP with only one single hidden layer*.

- **CON**: A solution with only one single hidden layer in general may require an **infinite amount of nodes**.

Summary

- **PRO**: It is possible to approximate any arbitrary **Boolean function** (slides 9–14), **boundary** (slides 15–19), or **real-valued function** (equivalent to slides 15–19) to arbitrary precision with an *<u>MLP with only one single hidden layer</u>*.

- **CON**: A solution with only one single hidden layer in general may require an **infinite amount of nodes**.

- **PRO**: The number of required nodes can be significantly reduced when exploiting **depth** for an MLP → *depth matters*.

Summary

- **PRO**: It is possible to approximate any arbitrary **Boolean function** (slides 9–14), **boundary** (slides 15–19), or **real-valued function** (equivalent to slides 15–19) to arbitrary precision with an <u>*MLP with only one single hidden layer*</u>.

- **CON**: A solution with only one single hidden layer in general may require an **infinite amount of nodes**.

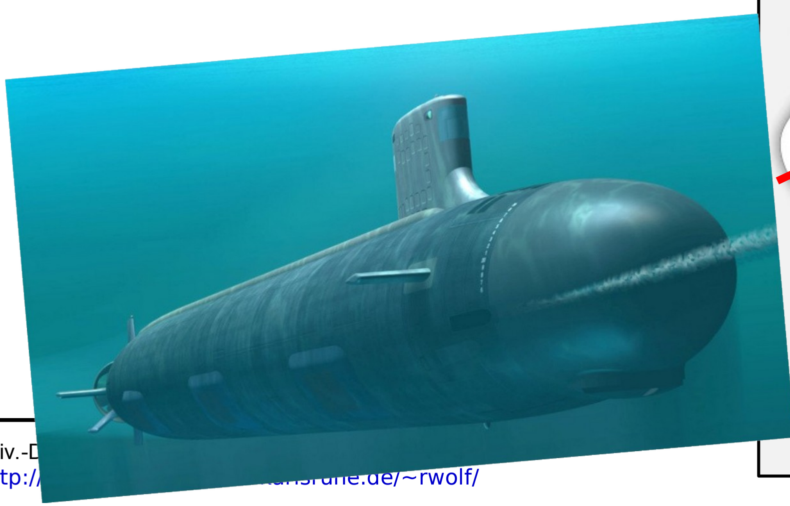
- **PRO**: The number of required nodes can be significantly reduced when exploiting **depth** for an MLP → *depth matters*.

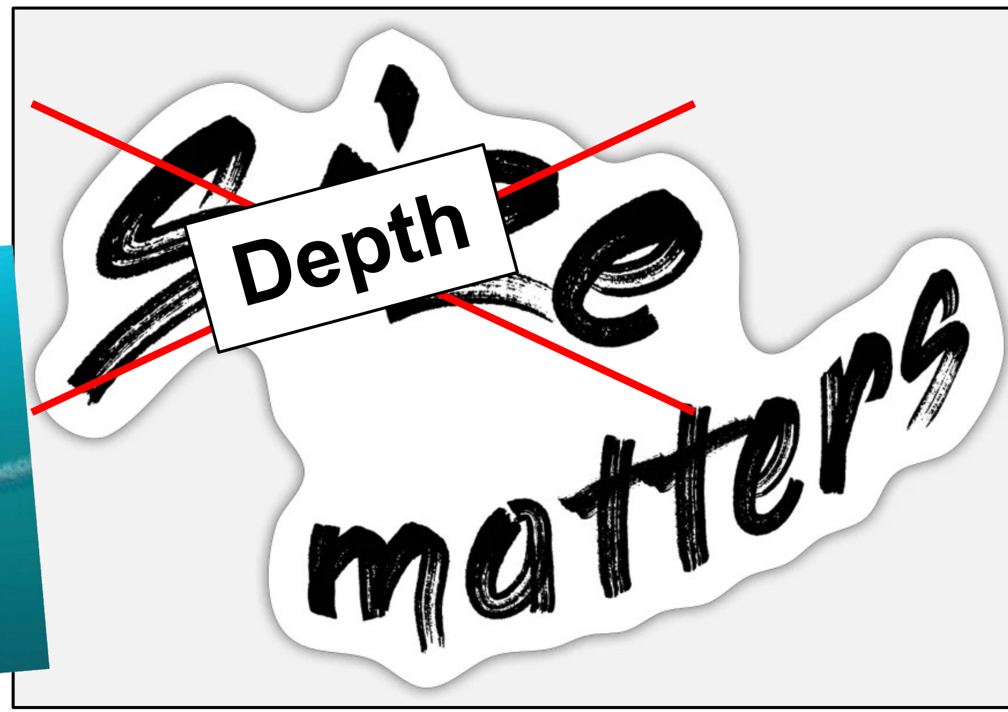
Summary

- **PRO**: It is possible to approximate any arbitrary **Boolean function** (slides 9–14), **boundary** (slides 15–19), or **real-valued function** (equivalent to slides 15–19) to arbitrary precision with an *MLP with only one single hidden layer*.

- **CON**: A solution with only one single hidden layer in general may require an **infinite amount of nodes**.

- **PRO**: The number of required nodes can be significantly reduced when exploiting **depth** for an MLP → *depth matters*.



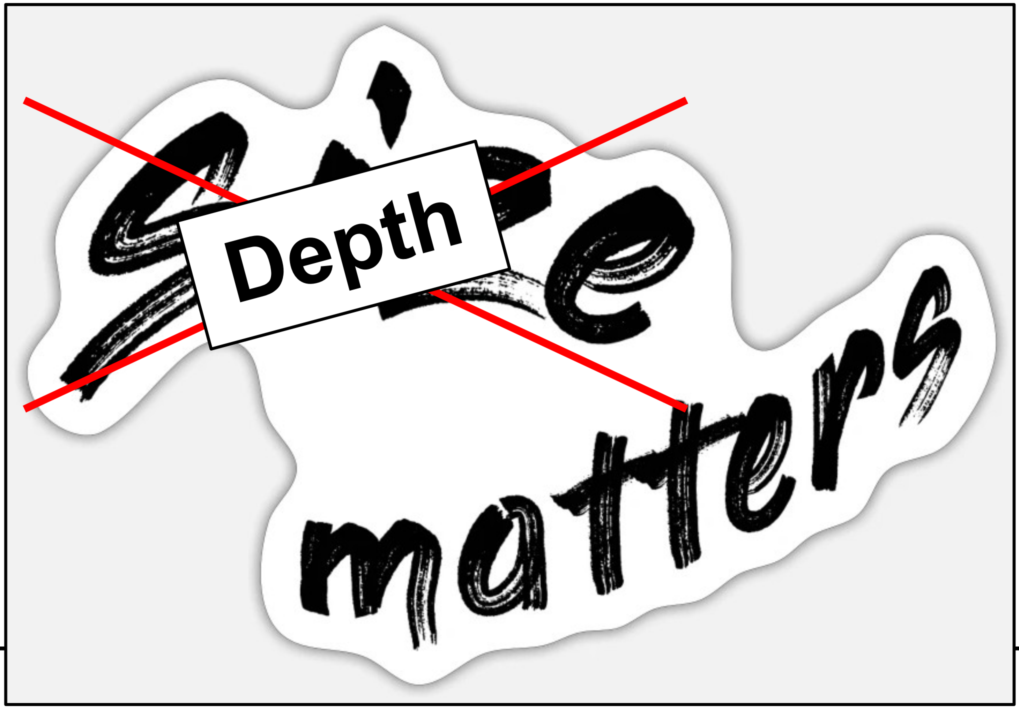


Summary

- **PRO**: It is possible to approximate any arbitrary **Boolean function** (slides 9–14), **boundary** (slides 15–19), or **real-valued function** (equivalent to slides 15–19) to arbitrary precision with an <u>*MLP with only one single hidden layer*</u>.

- **CON**: A solution with only one single hidden layer in general may require an **infinite amount of nodes**.

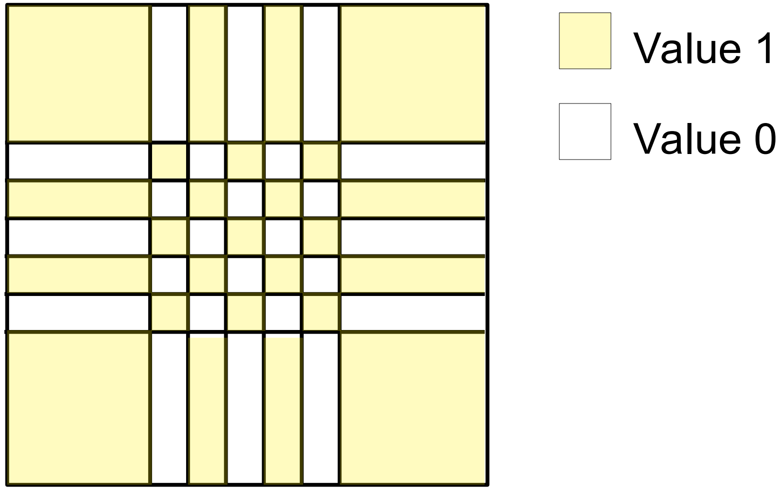
- **PRO**: The number of required nodes can be significantly reduced when exploiting **depth** for an MLP → *depth matters*.



- Deep NNs are ***more expressive*** than shallow NNs.

Deep vs. shallow NNs – Exercise –

- Check out the following example:

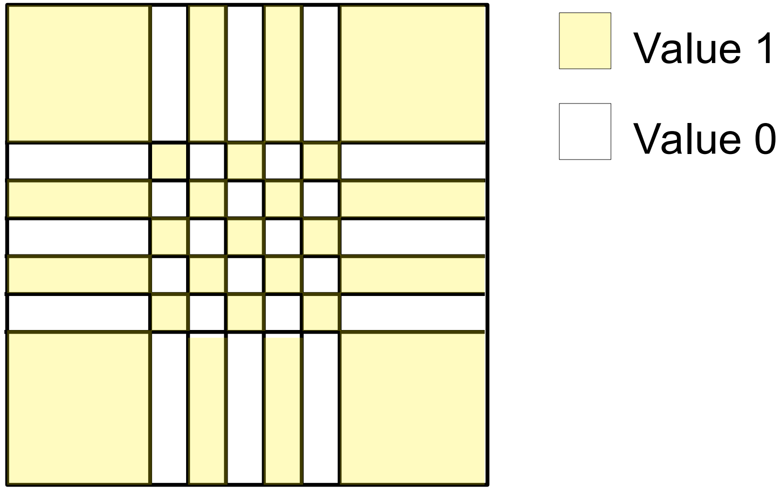


NB: Ignore the bounding box and assume the chess-board structure to extend to infinity beyond the box.

- How many hidden nodes do we need to represent this contour with an NN with a <u>single hidden layer</u>?

Deep vs. shallow NNs – Exercise –

- Check out the following example:

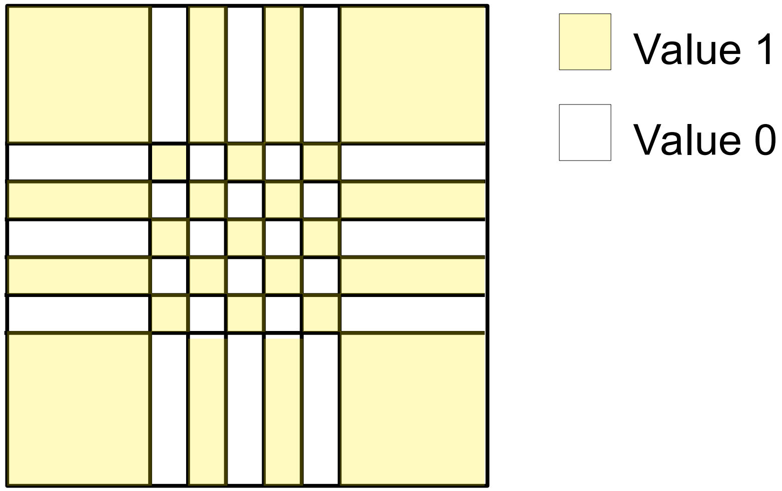


NB: Ignore the bounding box and assume the chess-board structure to extend to infinity beyond the box.

- How many hidden nodes do we need to represent this contour with an NN with a <u>single hidden layer</u>? – **infinitely many!**

Deep vs. shallow NNs – Exercise –

- Check out the following example:



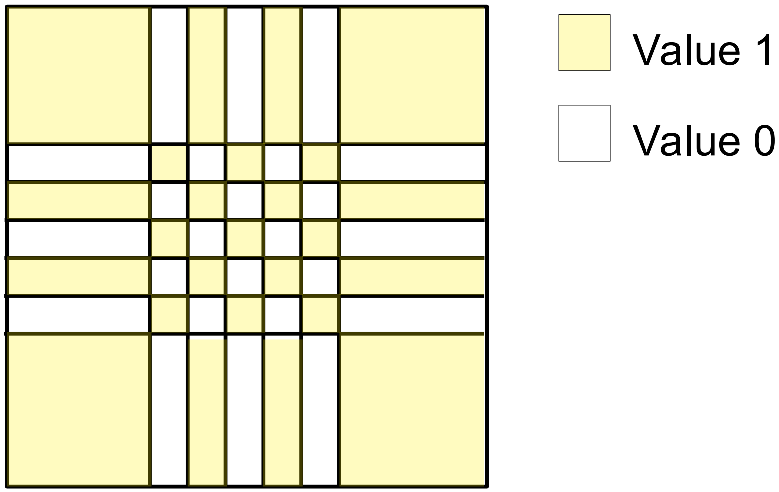
NB: Ignore the bounding box and assume the chess-board structure to extend to infinity beyond the box.

- How many hidden nodes do we need to represent this contour with an NN with a single hidden layer? – **infinitely many!**

- How many hidden nodes do we need to represent this contour with an NN with **two** hidden layers?

Deep vs. shallow NNs – Exercise –

- Check out the following example:



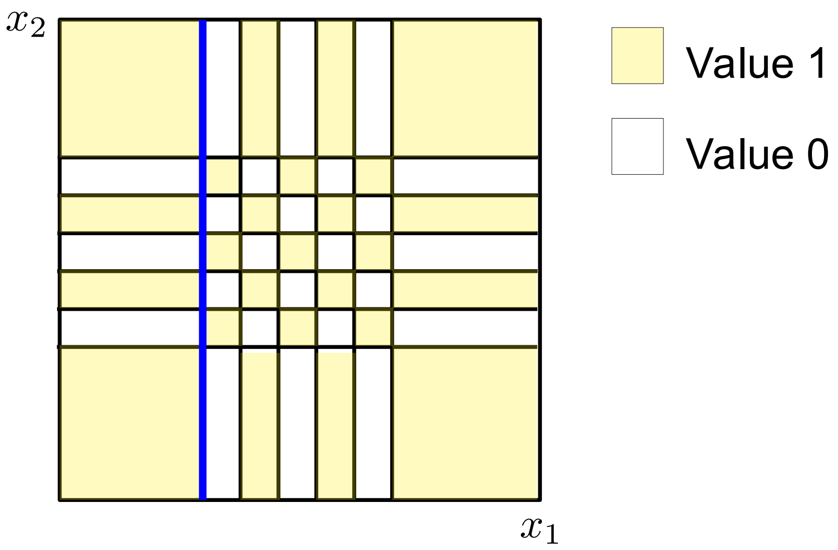
NB: Ignore the bounding box and assume the chess-board structure to extend to infinity beyond the box.

- How many hidden nodes do we need to represent this contour with an NN with a single hidden layer? – **infinitely many!**

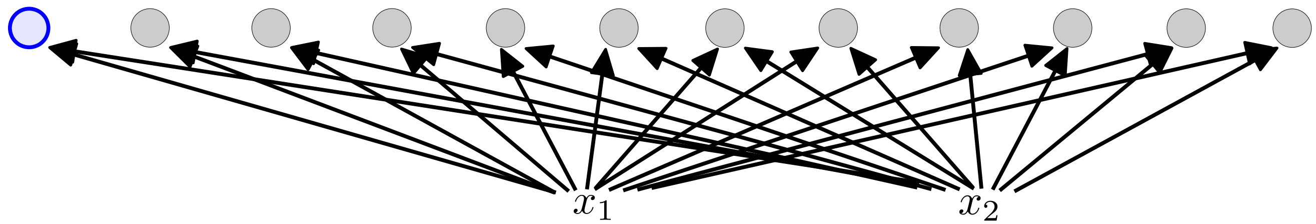
- How many hidden nodes do we need to represent this contour with an NN with **two** hidden layers? – **61**! How do we get to this number?

Deep vs. shallow NNs – Exercise –

- Check out the following example:

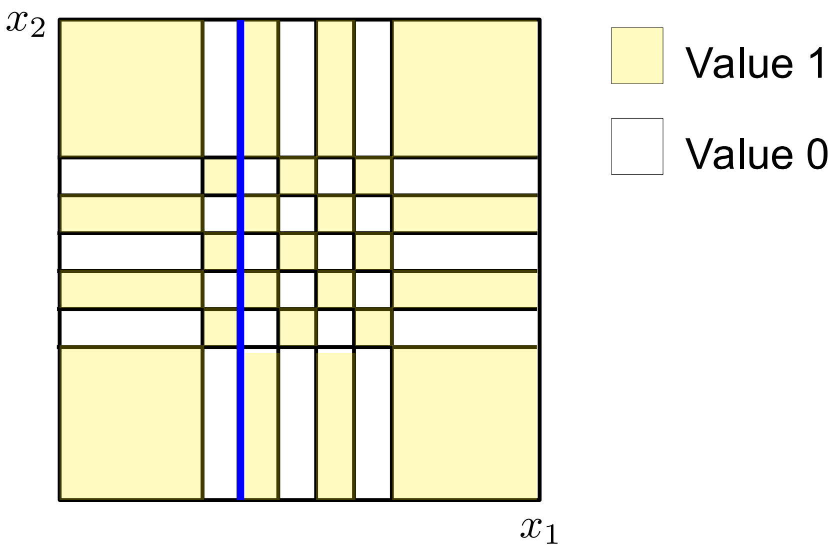


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

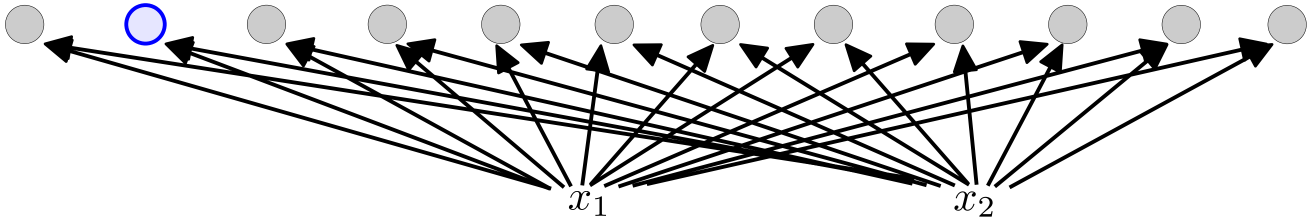


Deep vs. shallow NNs – Exercise –

- Check out the following example:

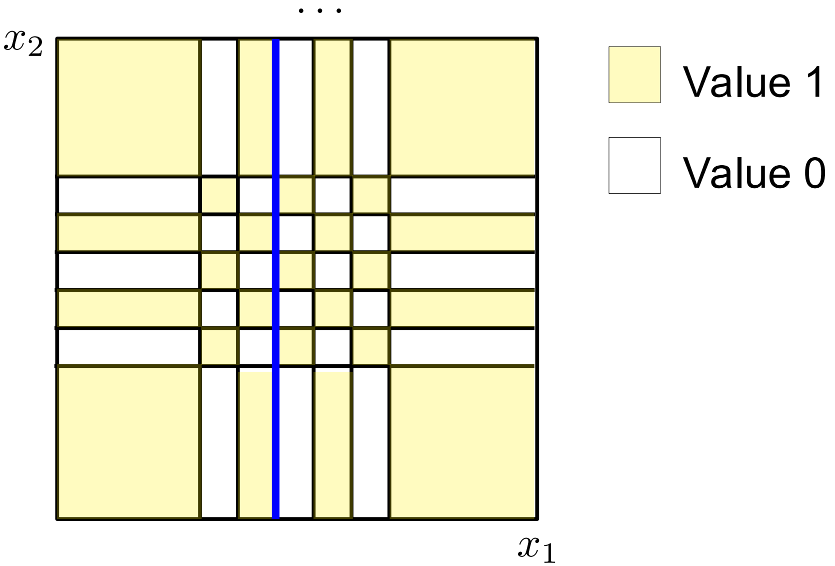


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

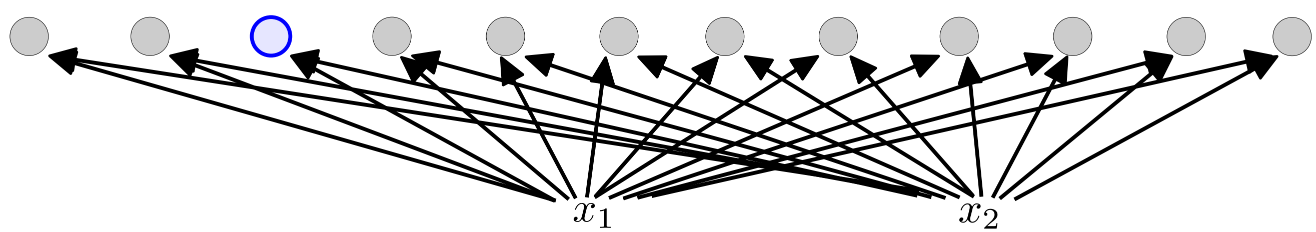


Deep vs. shallow NNs – Exercise –

- Check out the following example:

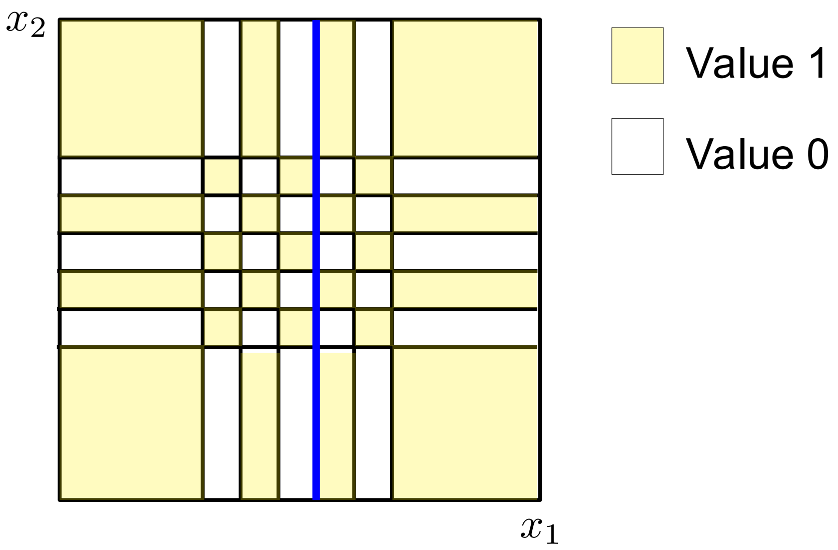


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

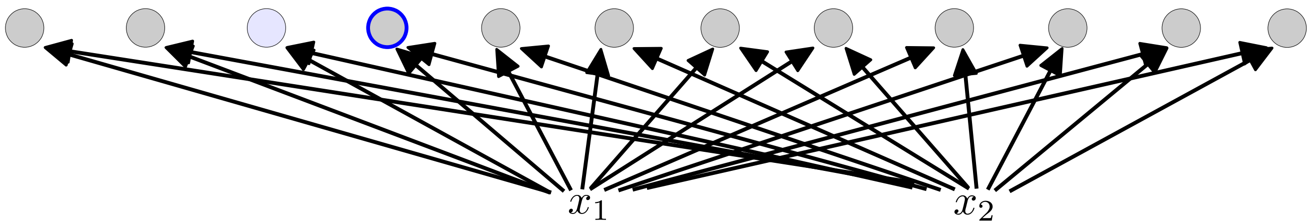


Deep vs. shallow NNs – Exercise –

- Check out the following example:

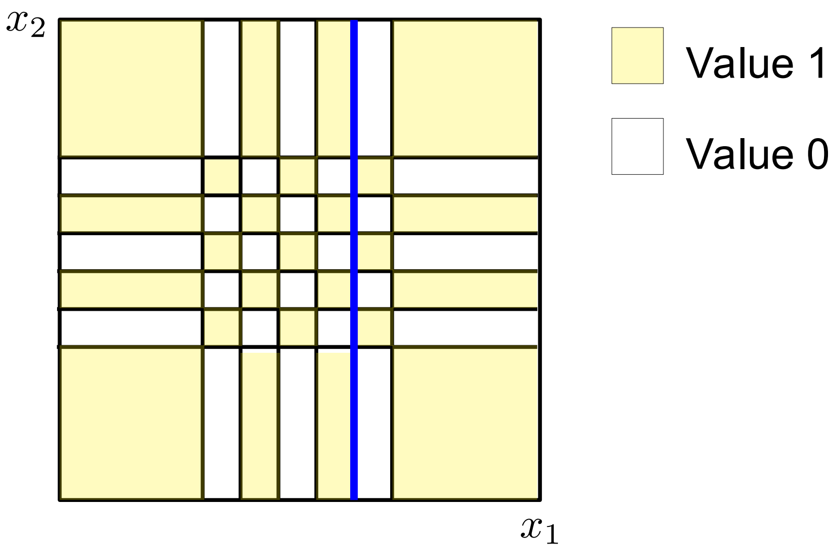


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

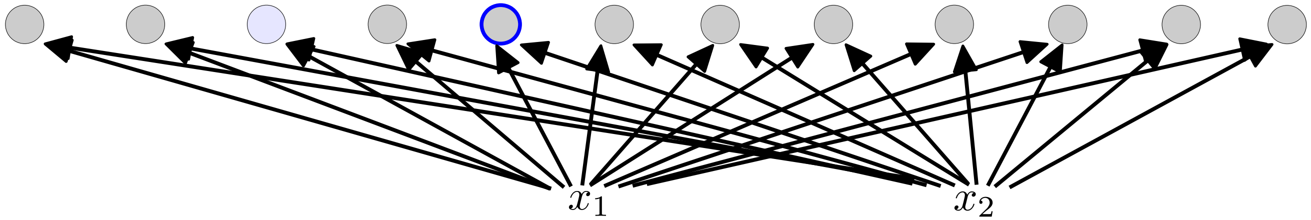


Deep vs. shallow NNs – Exercise –

- Check out the following example:

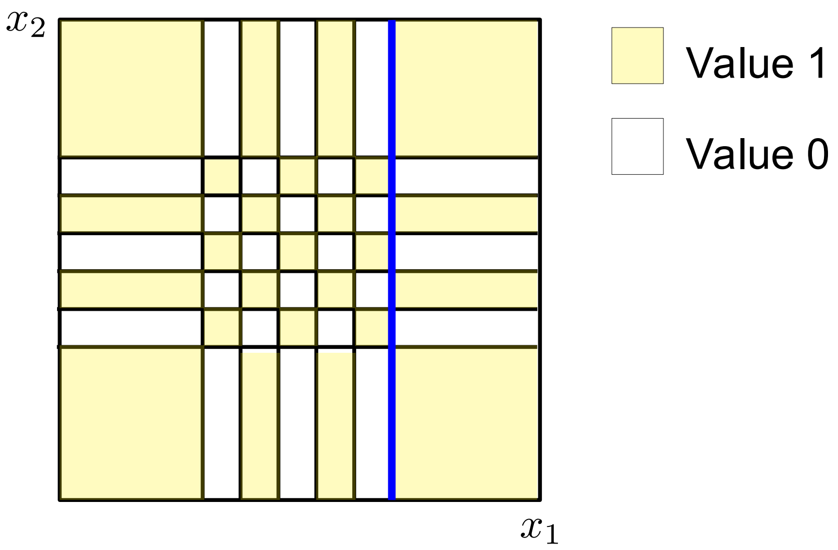


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

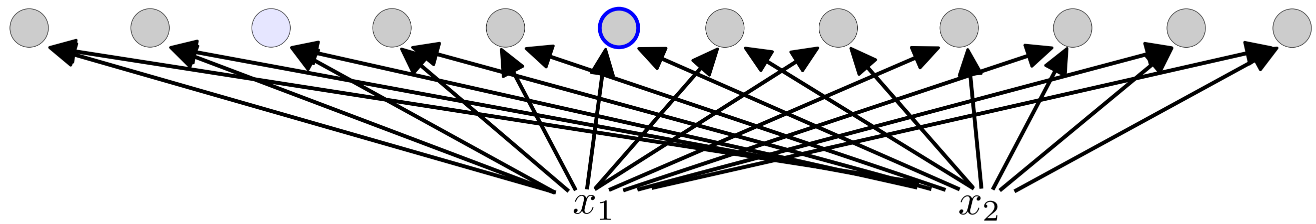


Deep vs. shallow NNs – Exercise –

- Check out the following example:

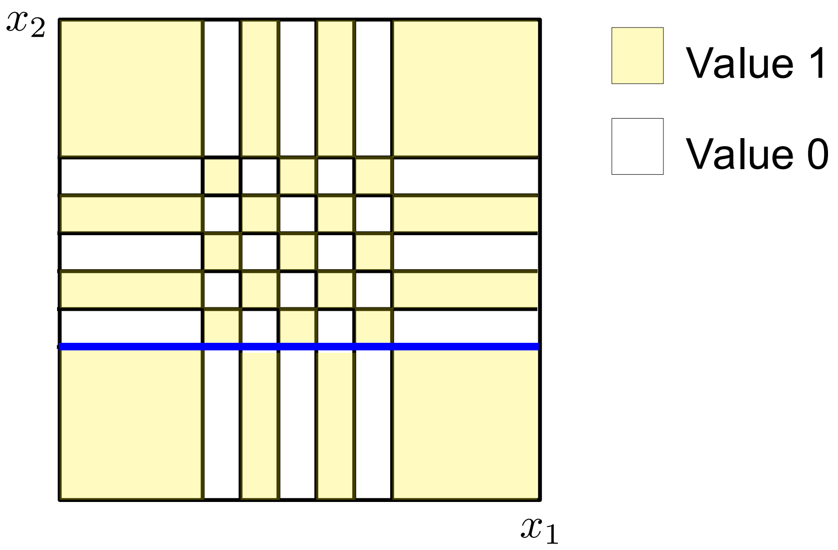


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

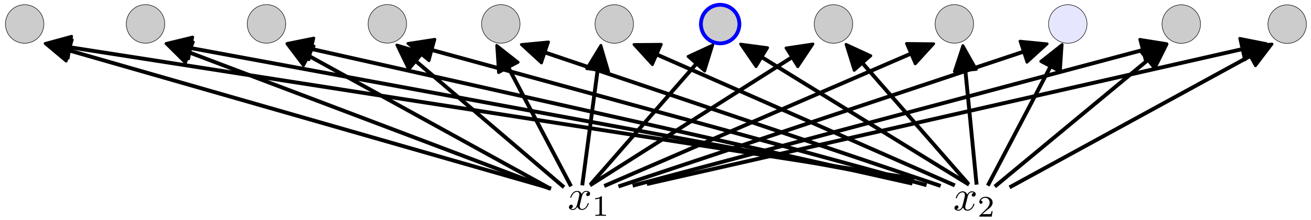


Deep vs. shallow NNs – Exercise –

- Check out the following example:

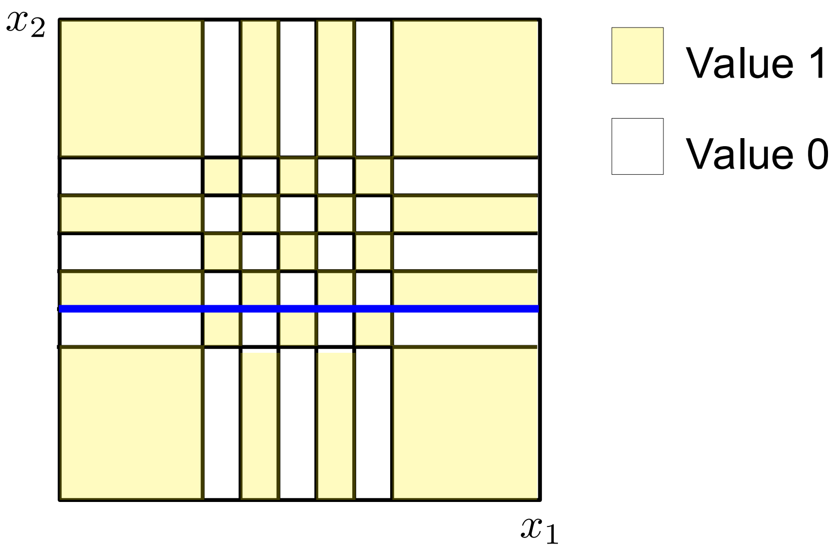


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

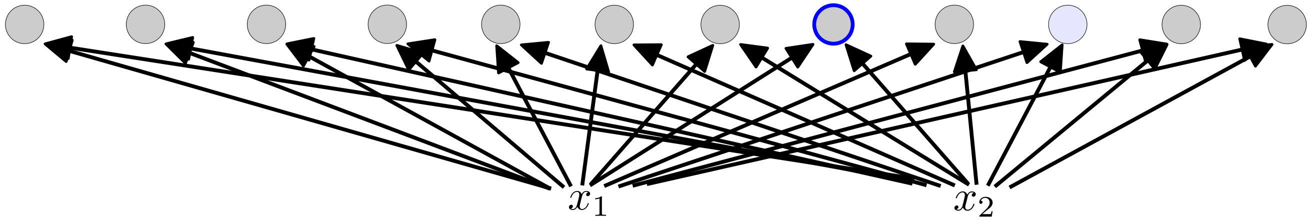


Deep vs. shallow NNs – Exercise –

- Check out the following example:

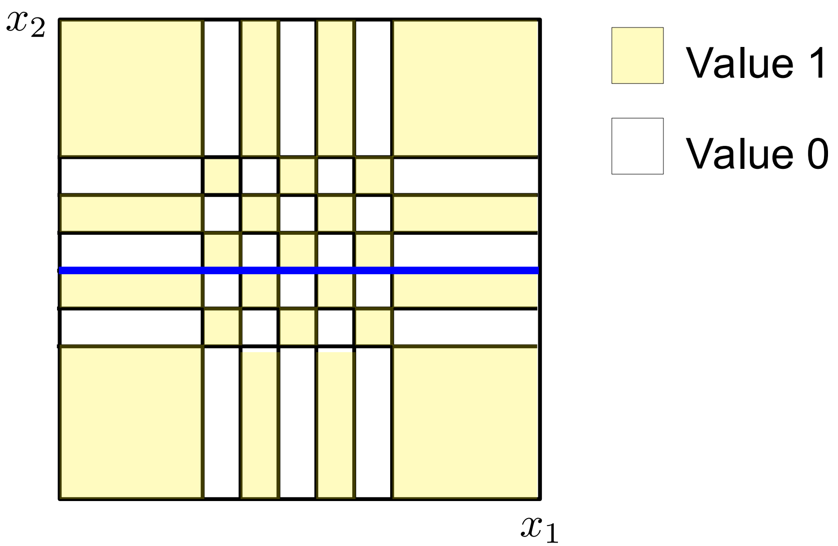


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

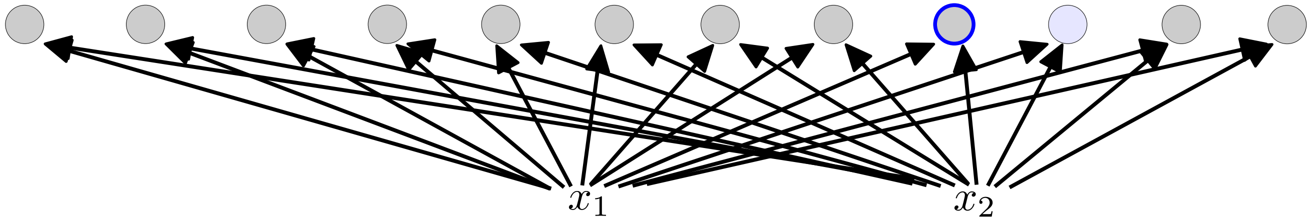


Deep vs. shallow NNs – Exercise –

- Check out the following example:



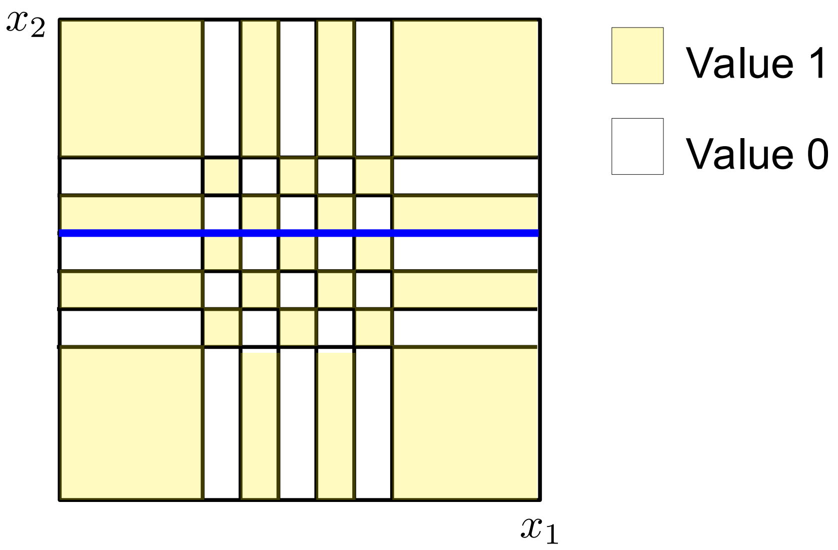
- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.



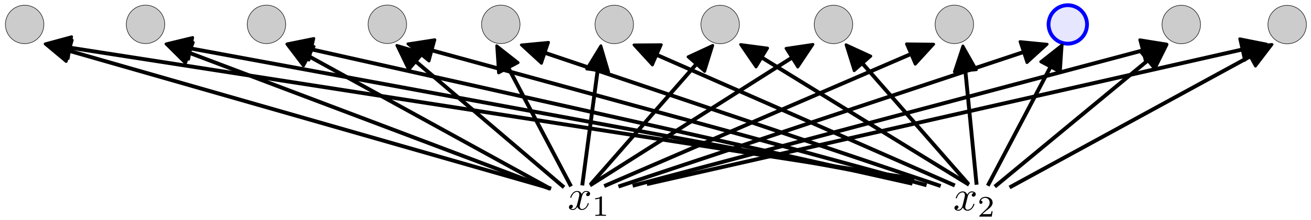
Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

Deep vs. shallow NNs – Exercise –

- Check out the following example:

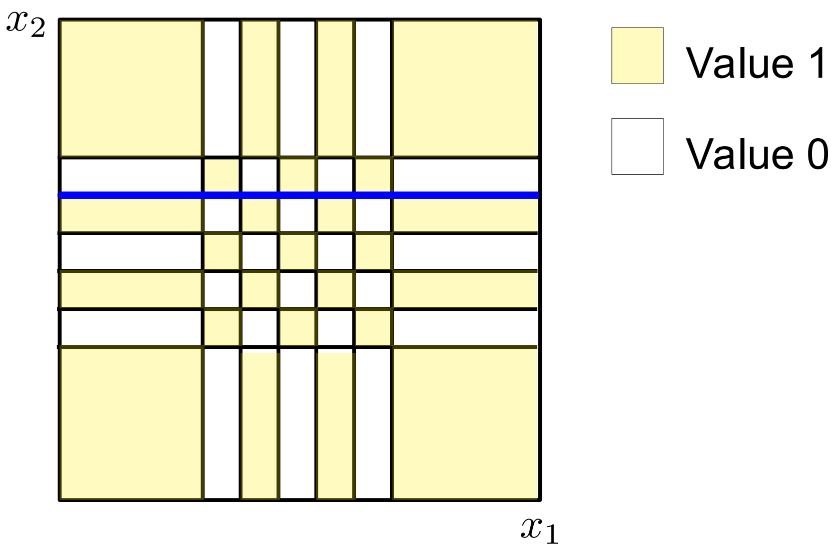


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

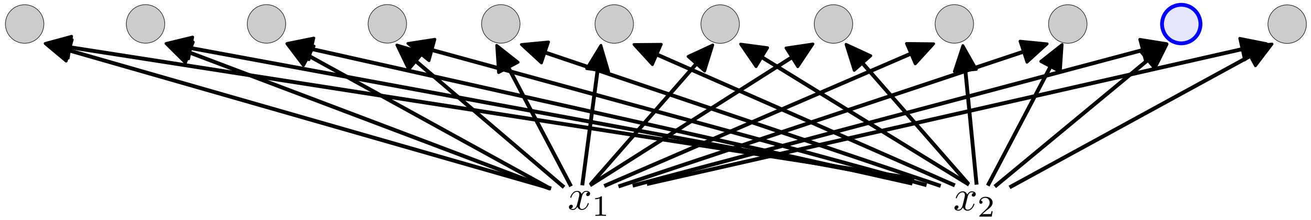


Deep vs. shallow NNs – Exercise –

- Check out the following example:

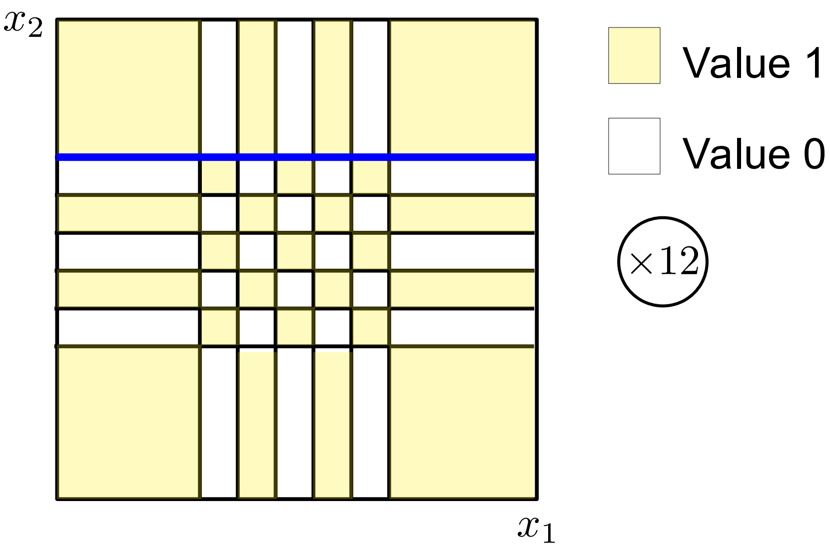


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

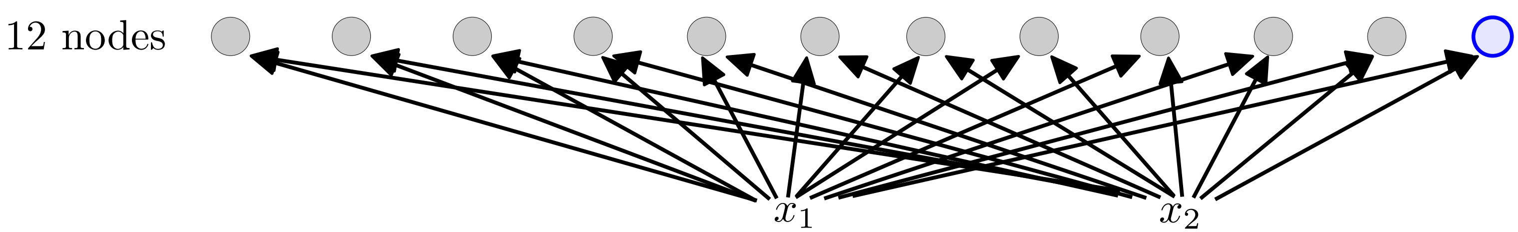


Deep vs. shallow NNs – Exercise –

- Check out the following example:

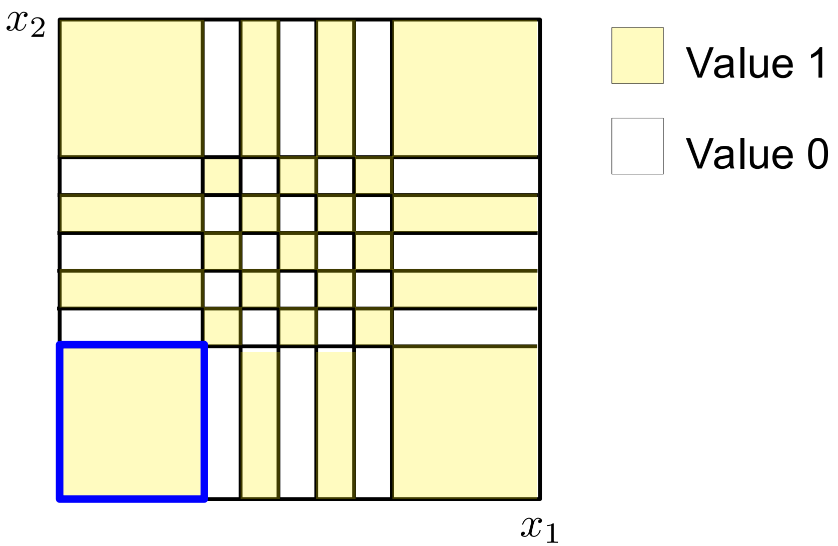


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

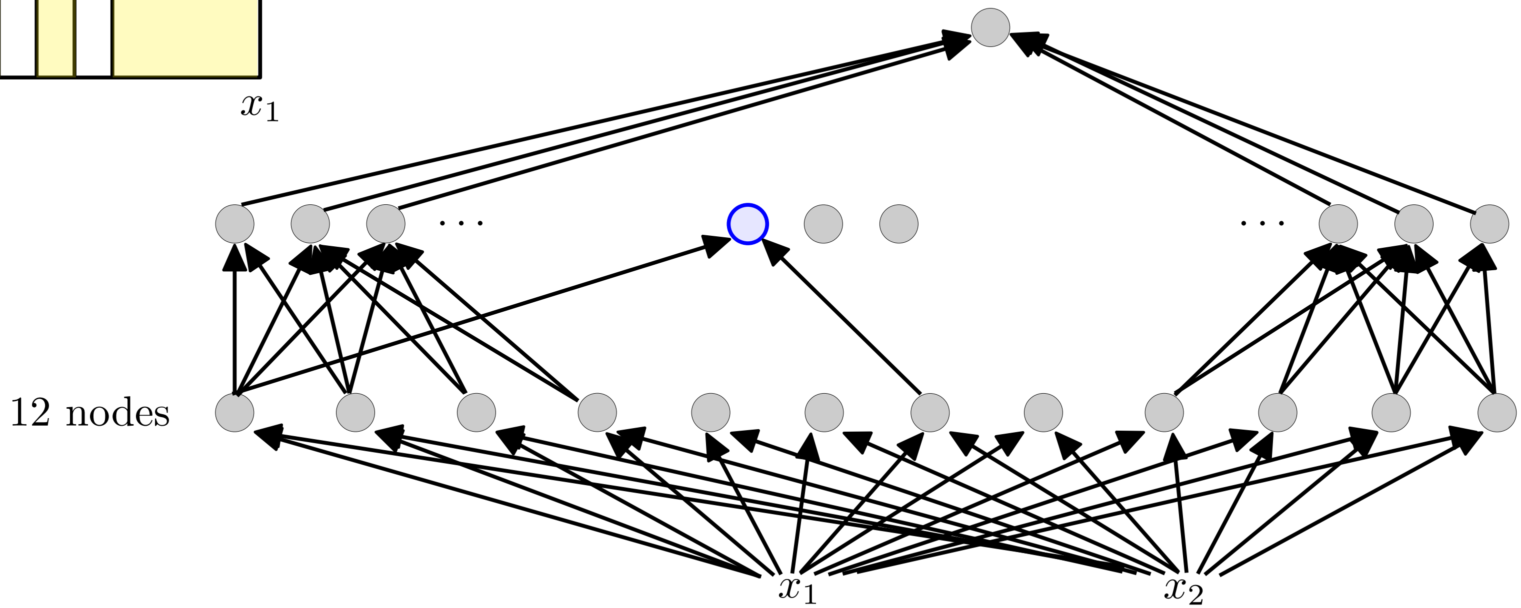


Deep vs. shallow NNs – Exercise –

- Check out the following example:

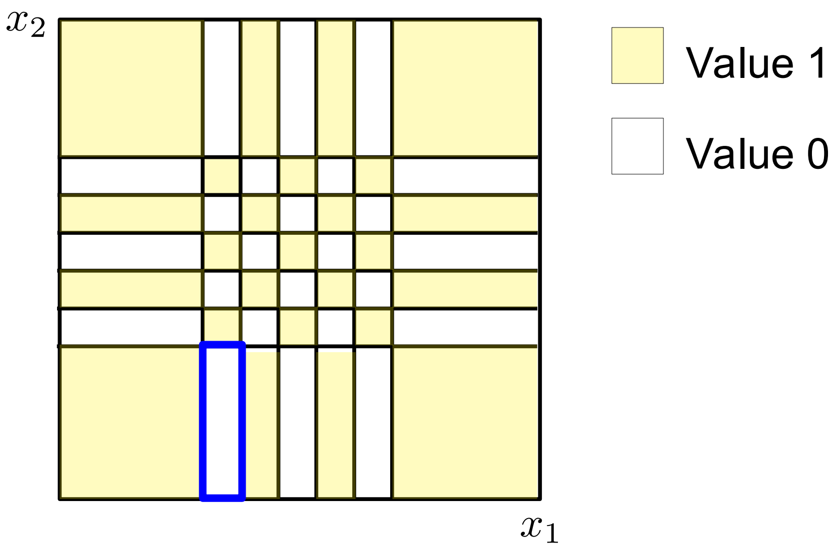


- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.
- In the second hidden layer $7 \times 7 = 49$ boxes are combined from the output of the first layer.



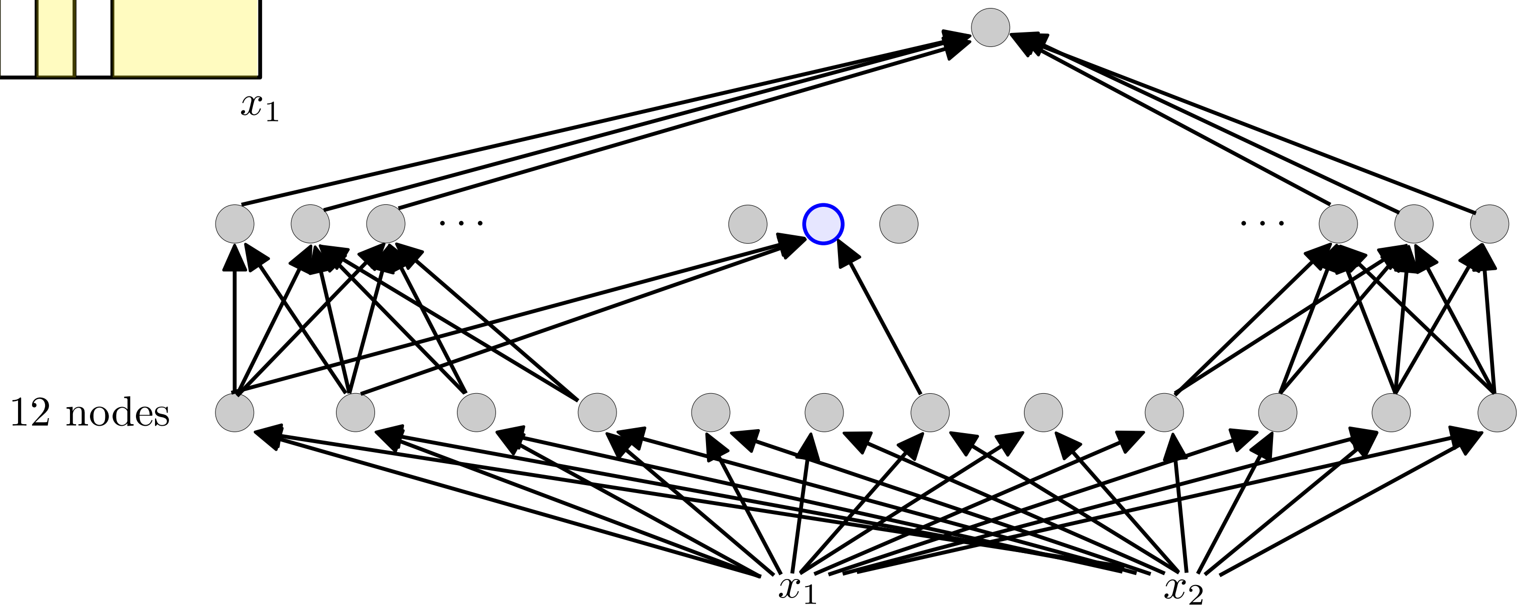
Deep vs. shallow NNs – Exercise –

- Check out the following example:



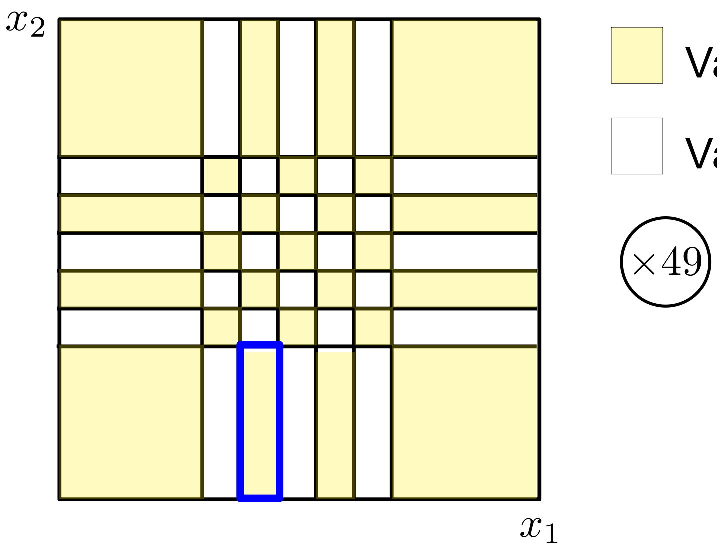
- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

- In the second hidden layer $7 \times 7 = 49$ boxes are combined from the output of the first layer.



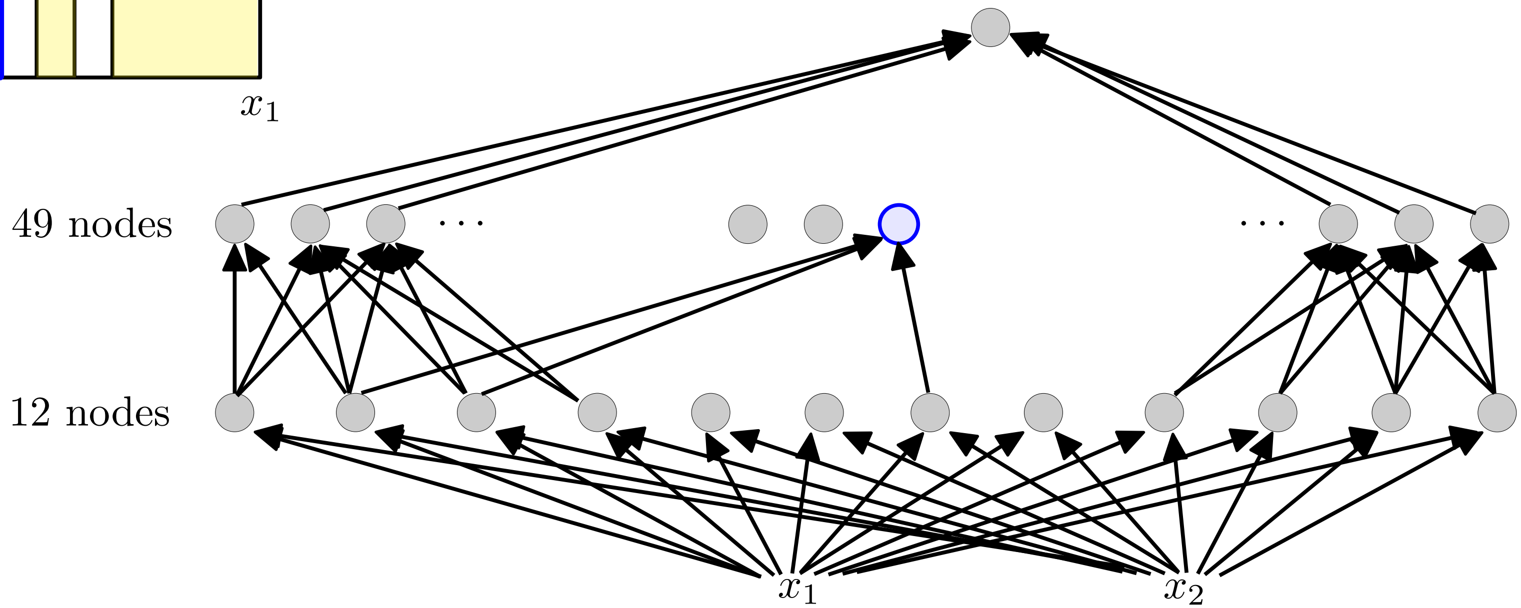
Deep vs. shallow NNs – Exercise –

- Check out the following example:



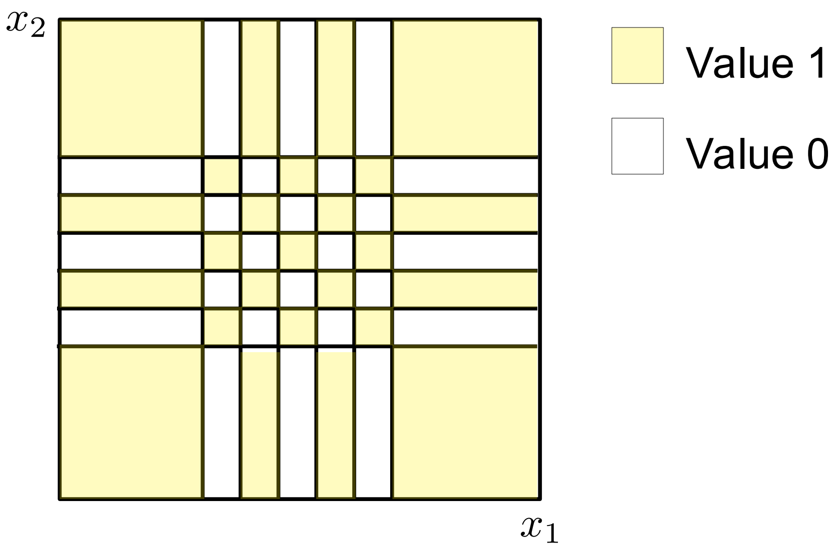
- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

- In the second hidden layer $7 \times 7 = 49$ boxes are combined from the output of the first layer.



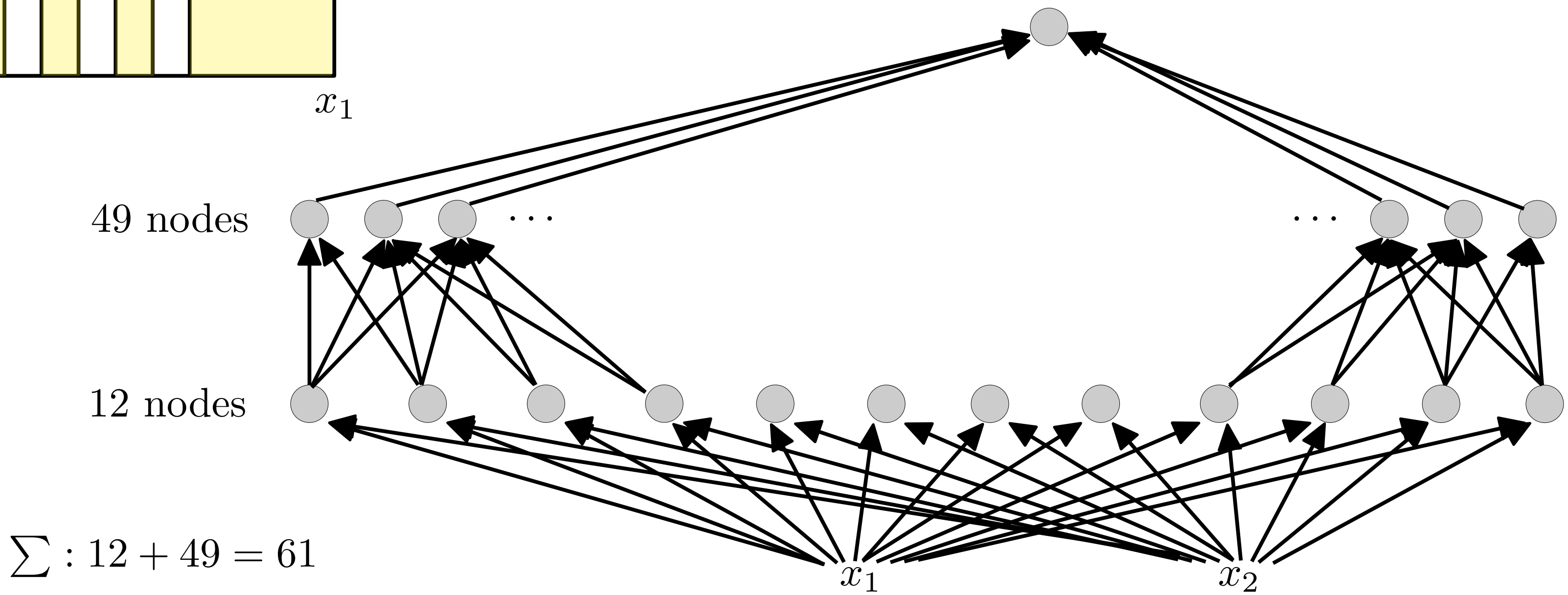
Deep vs. shallow NNs – Exercise –

- Check out the following example:



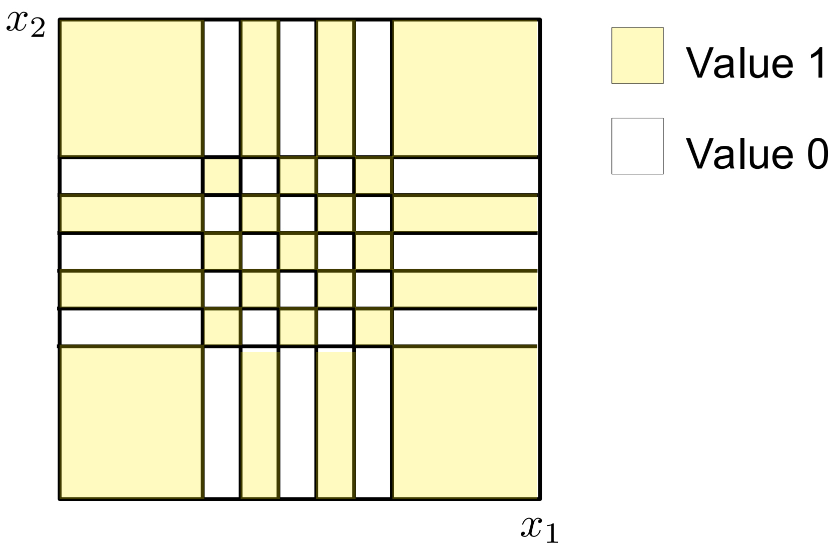
- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

- In the second hidden layer $7 \times 7 = 49$ boxes are combined from the output of the first layer.



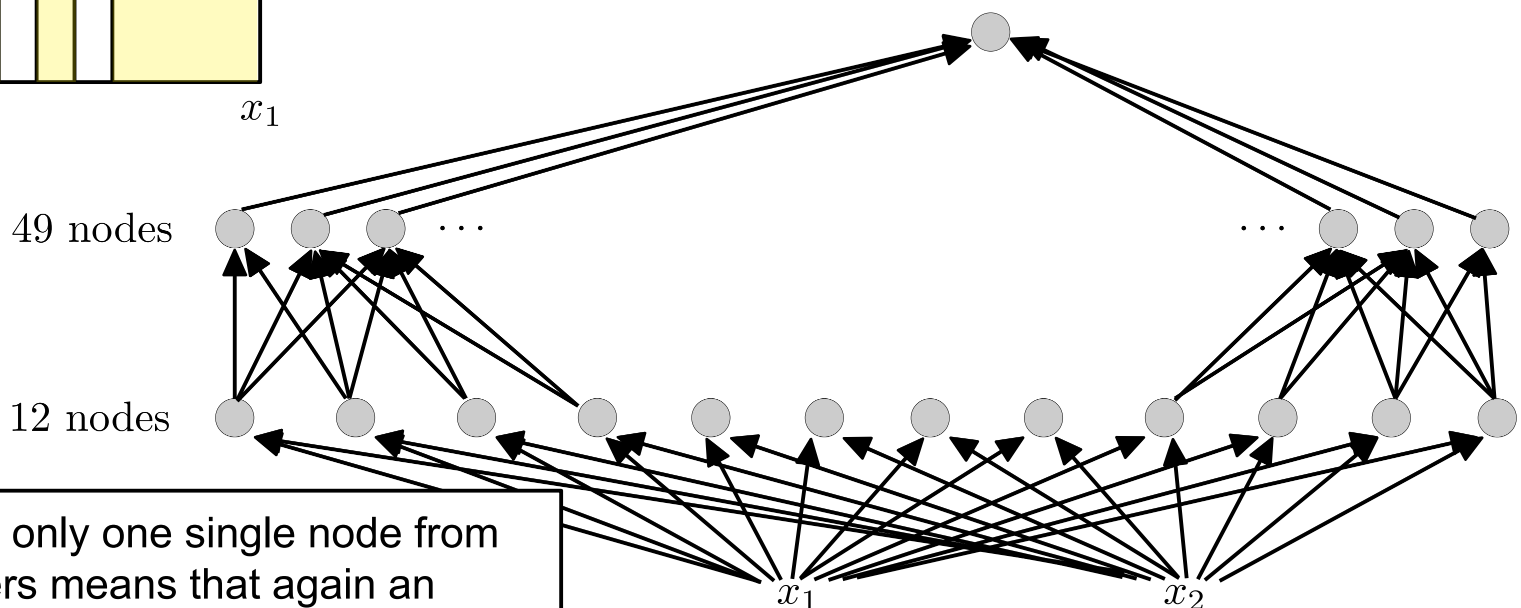
Deep vs. shallow NNs – Exercise –

- Check out the following example:



- The contour consists of 12 lines, which can be represented by 12 nodes in the first hidden layer.

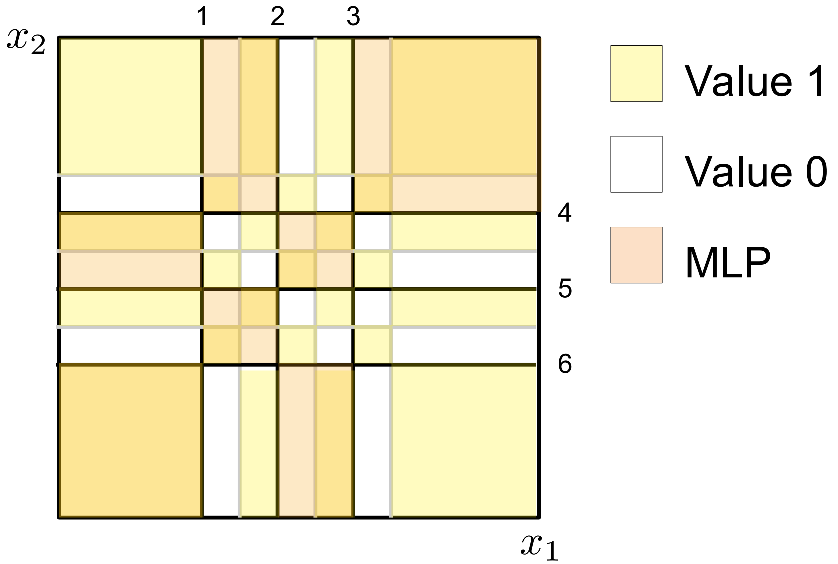
- In the second hidden layer $7 \times 7 = 49$ boxes are combined from the output of the first layer.



NB: Removing only one single node from one of the layers means that again an infinite number of nodes is required and/or the contour can only be approximated.

Whish meets reality

- Check out the following example:



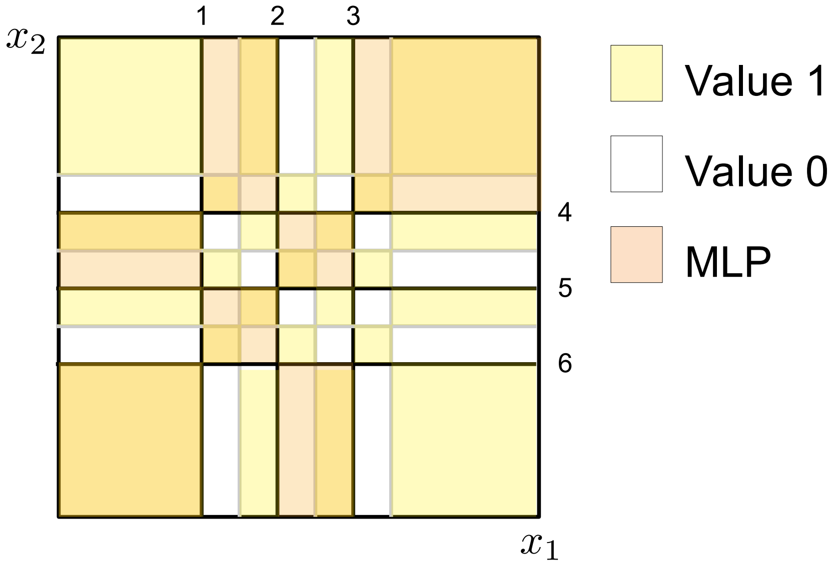
- Imagine you don‘t have 12 but only 6 nodes at hand in the first layer, which you arrange as shown on the left.

- Also this arrangement requires an infinite number of nodes. Why?

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

Whish meets reality

- Check out the following example:

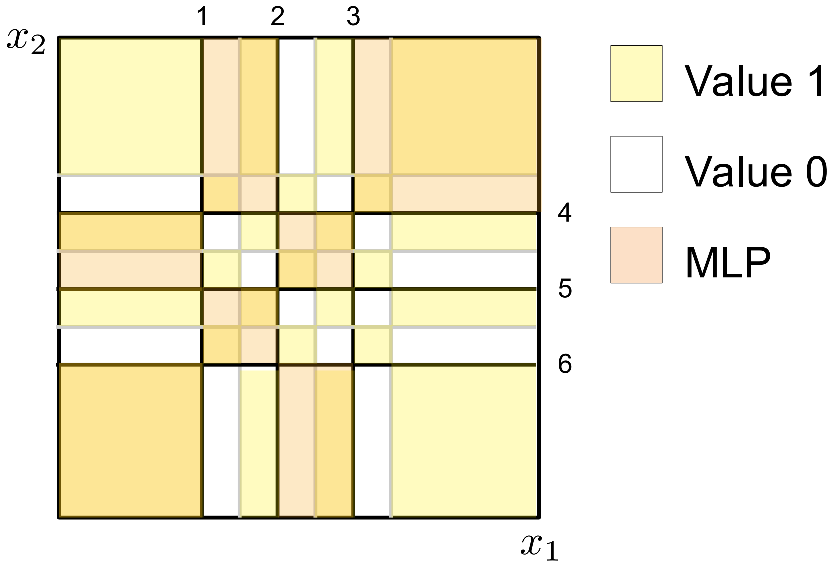


- Imagine you don't have 12 but only 6 nodes at hand in the first layer, which you arrange as shown on the left.

- Also this arrangement requires an infinite number of nodes. Why? – After passing the first layer you don't know any more where, e.g. inside an orange box, a sample is located.

Whish meets reality

- Check out the following example:



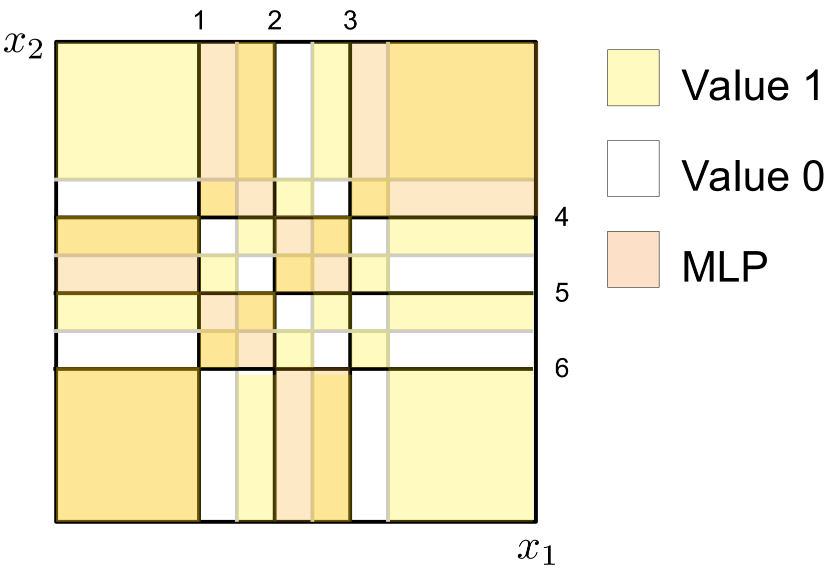
- Imagine you don't have 12 but only 6 nodes at hand in the first layer, which you arrange as shown on the left.

- Also this arrangement requires an infinite number of nodes. Why? – After passing the first layer you don't know any more where, e.g. inside an orange box, a sample is located.

- Root cause is the use of a step function as activation function in our example.

The role of the activation function

- Check out the following example:



- Imagine you don't have 12 but only 6 nodes at hand in the first layer, which you arrange as shown on the left.
- Also this arrangement requires an infinite number of nodes. Why? – After passing the first layer you don't know any more where, e.g. inside an orange box, a sample is located.
- Root cause is the use of a step function as activation function in our example.

Choosing an activation function for the nodes that preserves the information where, relative to the hyperplane boundary, a sample is located (slide 4) allows **processing of information in proceeding layers**. This works the better, the better the activation function supports the transmission of this information through consecutive layers.

Take a break?

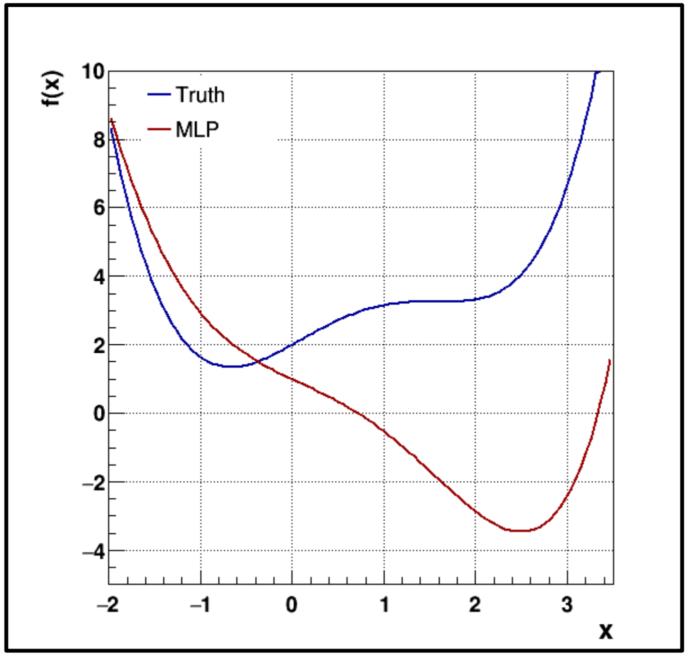
Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

Machine learning

- Up to now we did not touch the actual ML part: the question how to determine the weights and thresholds of the MLPs to fulfill their tasks.

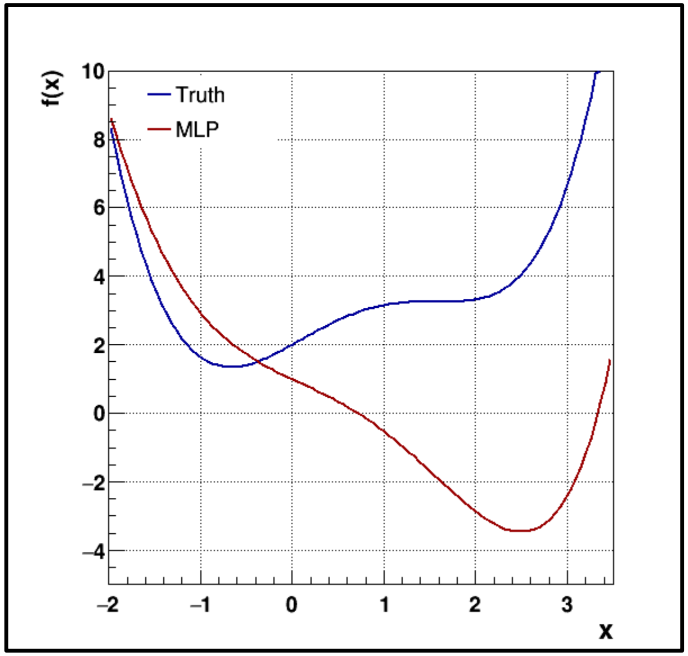
Truth vs. prediction

- Assume the MLP should represent the blue function shown on the right (→ **truth**).



Truth vs. prediction

- Assume the MLP should represent the blue function shown on the right ($\rightarrow$ **truth**).
- Random choice of the weights $\{\omega_{ij}\}$ might result in the red curve, shown on the right ($\rightarrow$ **prediction**).

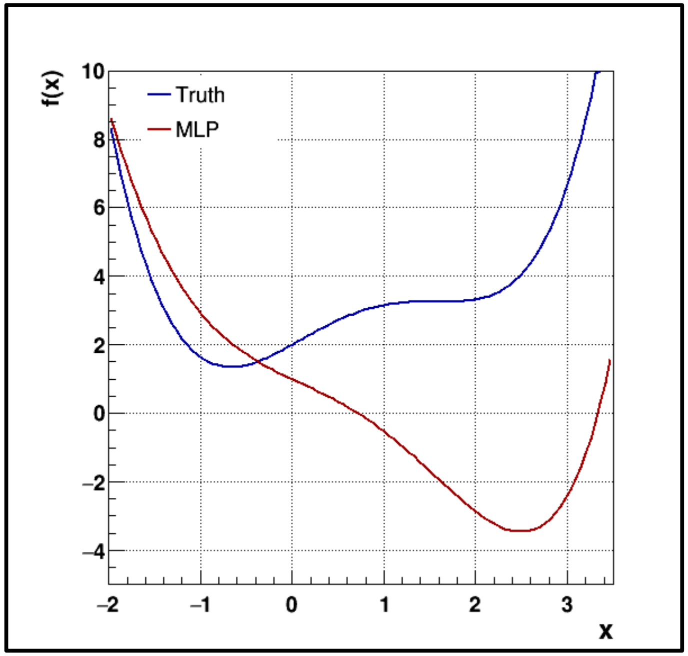


Truth vs. prediction

- Assume the MLP should represent the blue function shown on the right ($\rightarrow$ **truth**).

- Random choice of the weights $\{\omega_{ij}\}$ might result in the red curve, shown on the right ($\rightarrow$ **prediction**).

- **Task:** Adapt the weights such that the red curve approaches the blue one as closely as possible.



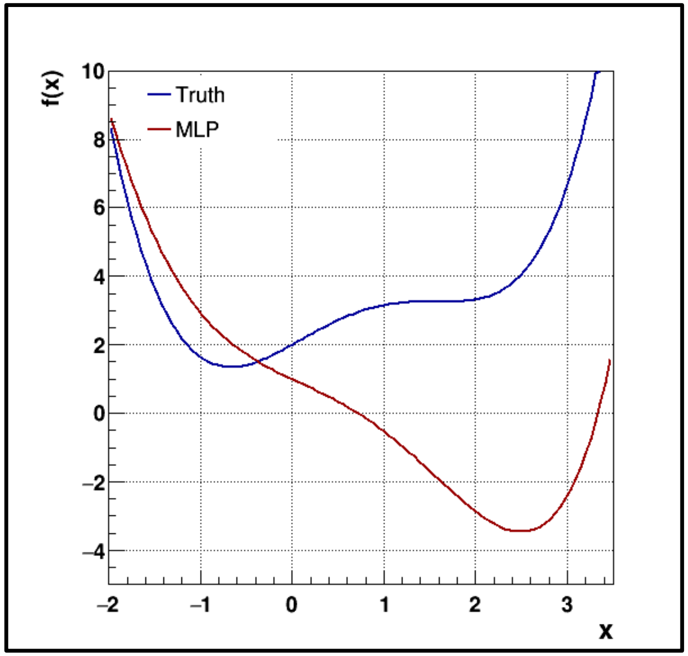
Truth vs. prediction

- Assume the MLP should represent the blue function shown on the right ($\rightarrow$ **truth**).

- Random choice of the weights $\{\omega_{ij}\}$ might result in the red curve, shown on the right ($\rightarrow$ **prediction**).

- **Task:** Adapt the weights such that the red curve approaches the blue one as closely as possible.

- To solve this task mathematically we will quantify the difference between the two curves with the so-called **loss or cost function**.



Sample and training

- **In general we don‘t know the truth**. We have to infer it from a sample hoping that the sample is *representative* of the ground truth (→ learning by example).

Sample and training

- **In general we don‘t know the truth**. We have to infer it from a sample hoping that the sample is *representative* of the ground truth (→ learning by example).

- Learning by example we call **training**.

Sample and training

- **In general we don‘t know the truth**. We have to infer it from a sample hoping that the sample is *representative* of the ground truth (→ learning by example).

- Learning by example we call **training**.

- To be representative the sample should catch all relevant characteristics of the truth. Individual properties of the sample (a.k.a. fluctuations) should not influence the training (→ generalization property).

Sample and training

- **In general we don‘t know the truth**. We have to infer it from a sample hoping that the sample is *representative* of the ground truth (→ learning by example).

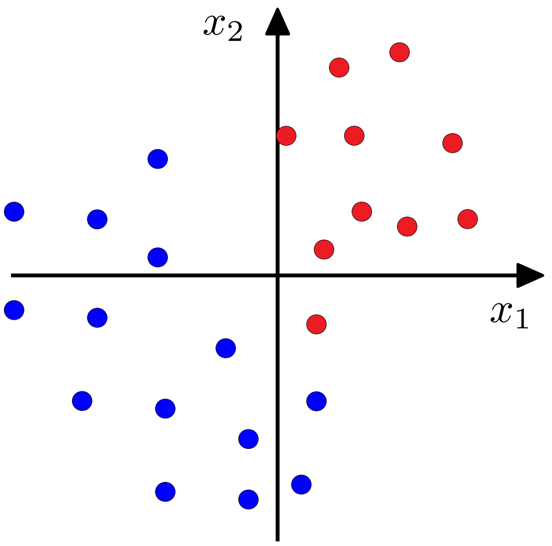
- Learning by example we call **training**.

- To be representative the sample should catch all relevant characteristics of the truth. Individual properties of the sample (a.k.a. fluctuations) should not influence the training (→ generalization property).

- If an MLP has so many trainable parameters that it can pick up on fluctuations of the sample it runs into the issue of overfitting (→ overtraining). If the MLP has too few parameters it might not be *expressive* enough to do the job (→ underfitting).

The perceptron learning rule

- **Historic example**: Training of a single Boolean perceptron to separate two classes with the help of labeled examples (here represented by points with different color) [1]:

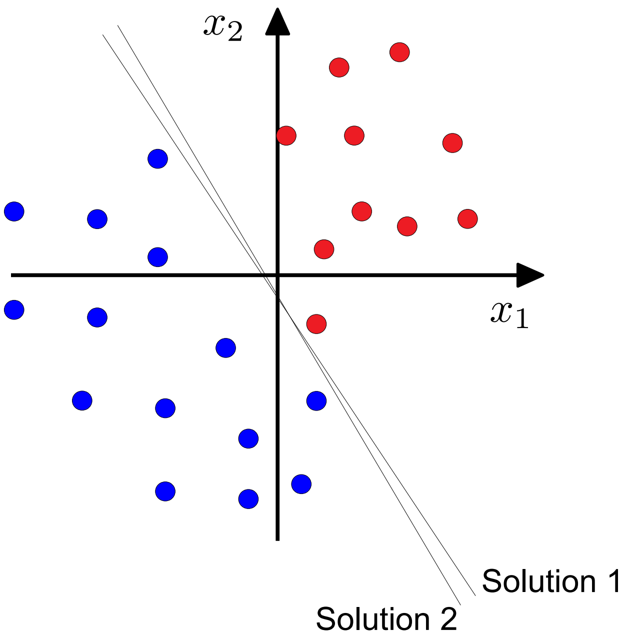


- **Task**: Determine the weights of the perceptron such that the red points (with values 1) and the blue points (with values 0) are separated.

[1] This is the historic example by Frank Rosenblatt, from the 1960ies.

The perceptron learning rule

- **Historic example**: Training of a single Boolean perceptron to separate two classes with the help of labeled examples (here represented by points with different color) [(1)]:



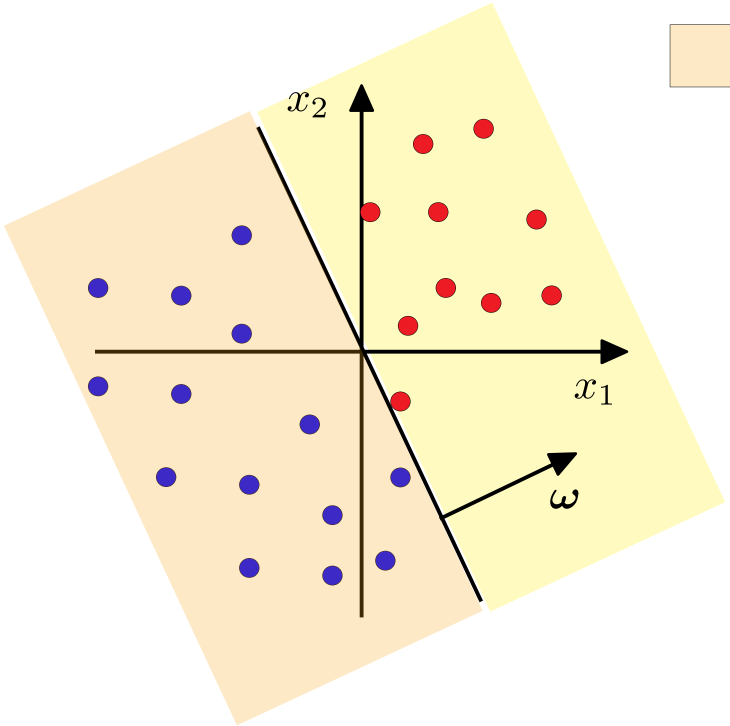
- **Task**: Determine the weights of the perceptron such that the red points (with values 1) and the blue points (with values 0) are separated.

- **NB**: The solution is ambiguous. The root cause of this is that the sample does not cover the complete space over x_1 and x_2.

(1) This is the historic example by Frank Rosenblatt, from the 1960ies.

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall\mathbf{x}$ in the plane (=on the boundary).



$\boldsymbol{\omega}\cdot\mathbf{x} < 0$ — $\boldsymbol{\omega}\cdot\mathbf{x} = 0$ — $\boldsymbol{\omega}\cdot\mathbf{x} > 0$

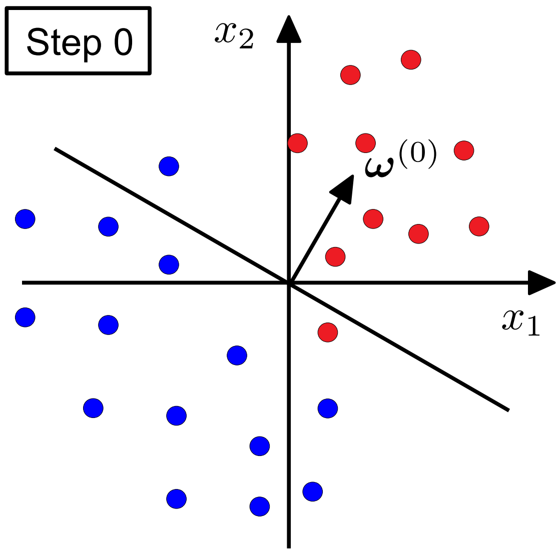
Algorithm:

- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall \mathbf{x}$ in the plane (=on the boundary).



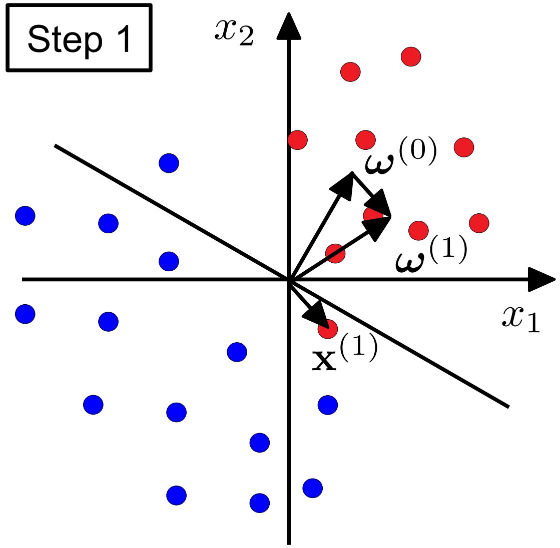
Algorithm:

- Initialize weights randomly. Step 0
- Only update for examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall \mathbf{x}$ in the plane (=on the boundary).



Algorithm:

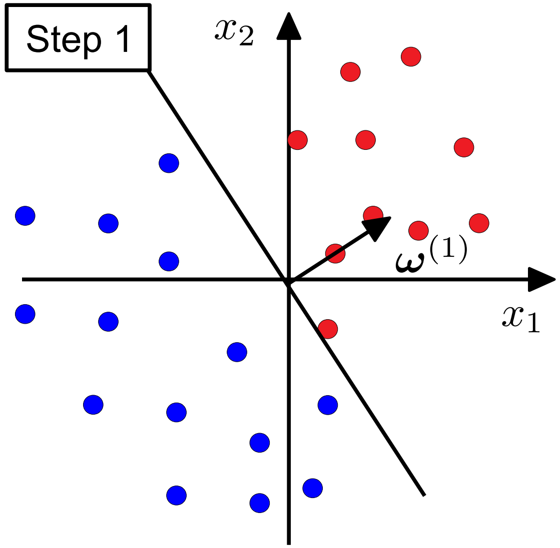
- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \rightarrow \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Step 1

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall \mathbf{x}$ in the plane (=on the boundary).



Algorithm:

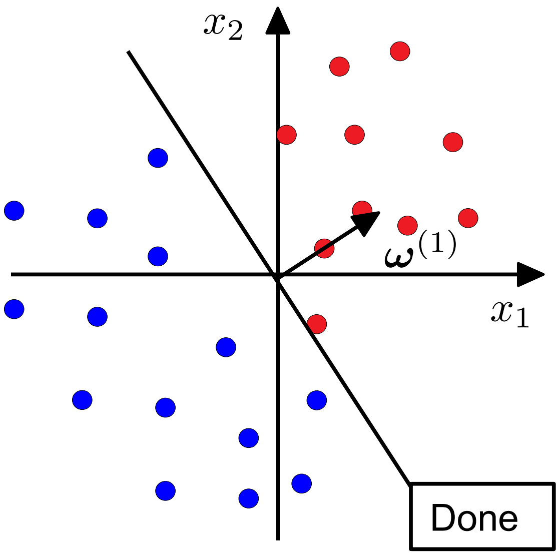
- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Step 1

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall \mathbf{x}$ in the plane (=on the boundary).



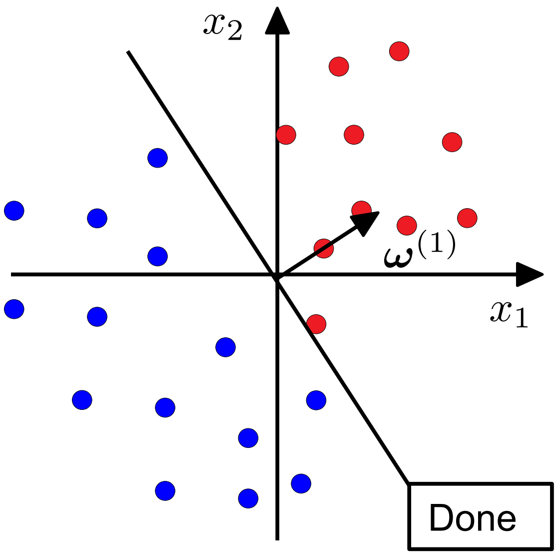
Algorithm:

- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall \mathbf{x}$ in the plane (=on the boundary).



NB: In the backup you can find the same algorithm played through w/ the blue points.

Algorithm:

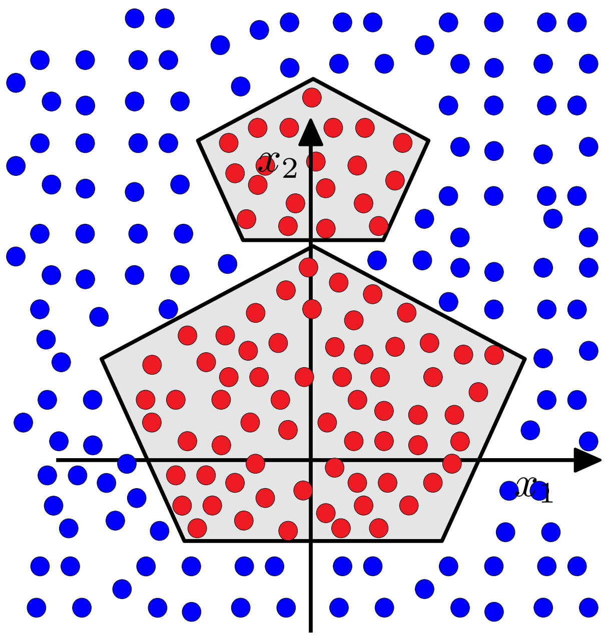
- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

- **NB**: Rosenblatt could show that a single logic perceptron for linearly separable tasks always converges to the correct solution after a *finite number* of steps.

The perceptron learning rule – limitations

- The attempt to apply the same learning rule to more complex problems fails. **Q**: Why?



Algorithm:

- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following learning rule:

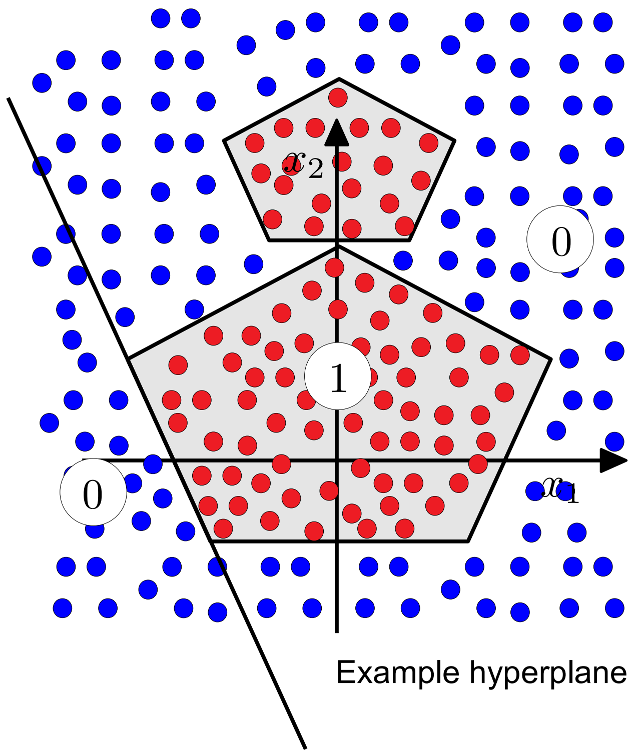
$$\boldsymbol{\omega}^{(k)} \rightarrow \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

The perceptron learning rule – limitations

- The attempt to apply the same learning rule to more complex problems fails. **Q**: Why?

- **A**: For the given example the samples are **not linearly separable**, i.e. one cannot draw a line to separate the blue from the red points!



Algorithm:

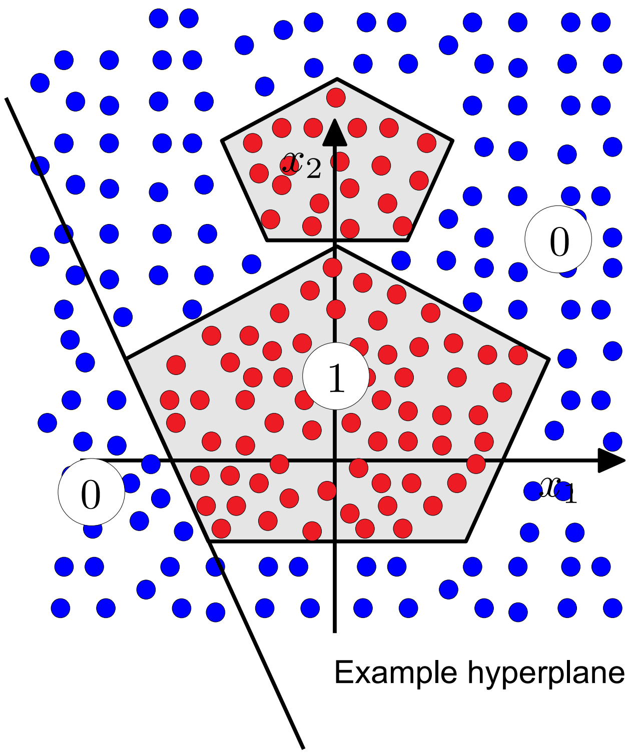
- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

The perceptron learning rule – limitations

- The attempt to apply the same learning rule to more complex problems fails. **Q**: Why?

- **A**: For the given example the samples are **not linearly separable**, i.e. one cannot draw a line to separate the blue from the red points!



Algorithm:

- Initialize weights randomly.
- Only update for examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

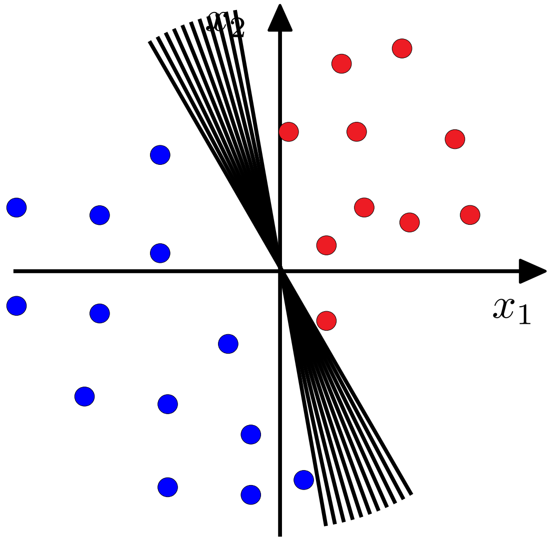
- **NB**: The problem can still be solved. One could re-label the values 0 and 1 for each hyperplane, but the complexity of the problem would still grow exponentially.

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

The role of the activation function – revisited

- The issue of the perceptron learning rule originates from the use of the Heavyside step function as activation function:

- Small changes of the weights of the perceptron have **no influence** on its output.

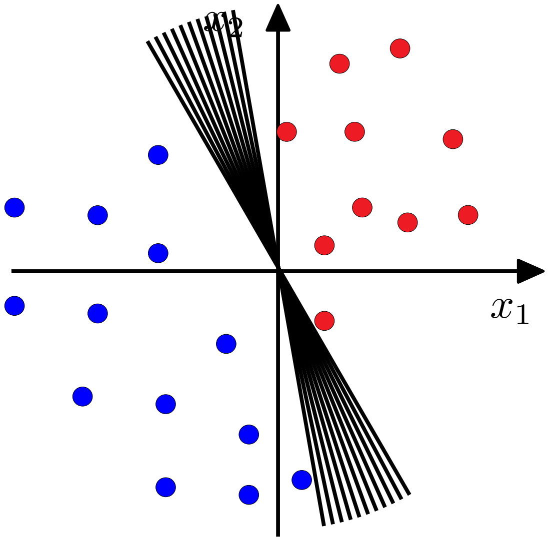


Each of these curves leads to the same output.

The role of the activation function – revisited

- The issue of the perceptron learning rule originates from the use of the Heavyside step function as activation function:

- Small changes of the weights of the perceptron have **no influence** on its output.



Each of these curves leads to the same output.

- **Solution**: Introduce a continuous activation function.

- This turns the NN output function differentiable in *any* variable:

$$y^{(k)}(\boldsymbol{\omega},\mathbf{x}) = \theta(z(\boldsymbol{\omega},\mathbf{x})) \qquad z^{(k)} = \sum_i \omega_i y_i^{(k-1)}$$

$$\frac{\mathrm{d}y^{(k)}}{\mathrm{d}\omega_i} = \frac{\mathrm{d}y^{(k)}}{\mathrm{d}z^{(k)}}\frac{\mathrm{d}z^{(k)}}{\mathrm{d}\omega_i} = \theta'(z^{(k)})\,y_i^{(k-1)}$$

$$\frac{\mathrm{d}y^{(k)}}{\mathrm{d}x_i} = \frac{\mathrm{d}y^{(k)}}{\mathrm{d}z^{(k)}}\frac{\mathrm{d}z^{(k)}}{\mathrm{d}x_i} = \theta'(z^{(k)})\,\omega_i$$

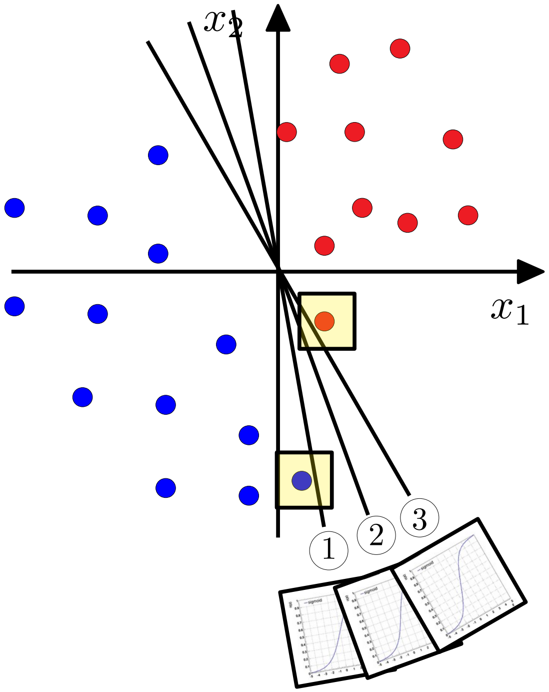
- This allows checking what changes in $y^{(k)}$ one obtains from small changes in ω.

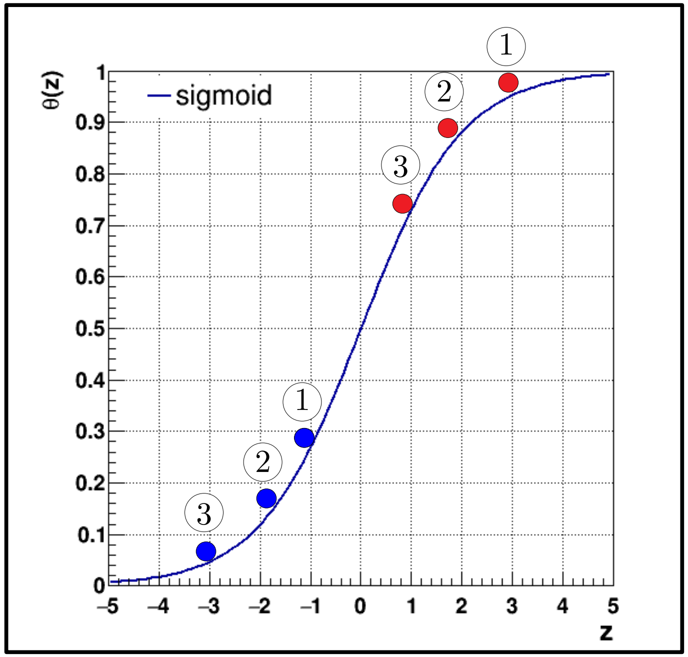
The continuous activation function

- The issue of the perceptron learning rule originates from the use of the Heavyside step function as activation function:

- **Example** sigmoid function:

Wrongly predicted samples

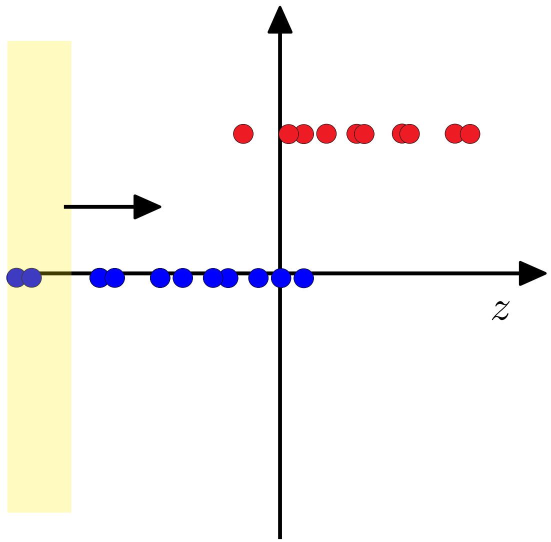


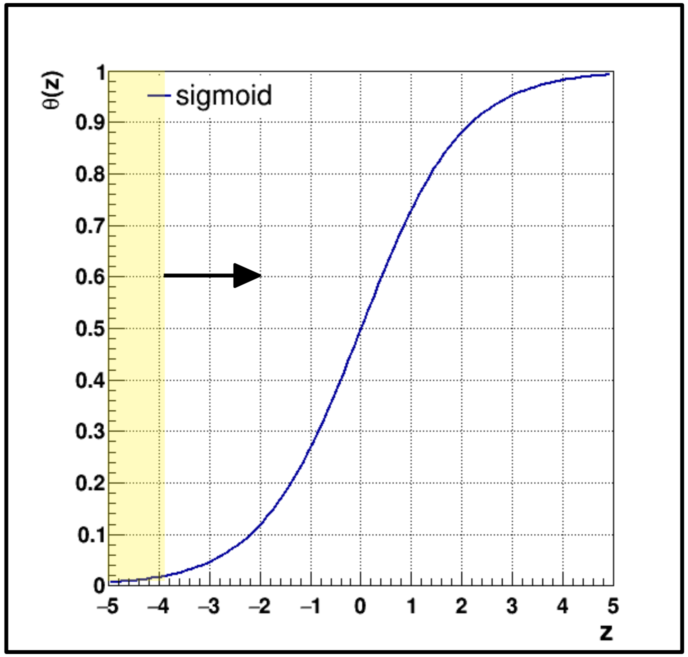


- Gradual changes of the weights lead to gradual changes of the NN output.

Not linearly separable tasks

- We take the sigmoid function as an example of addressing not linearly separable tasks, here demonstrated with a 1d example:

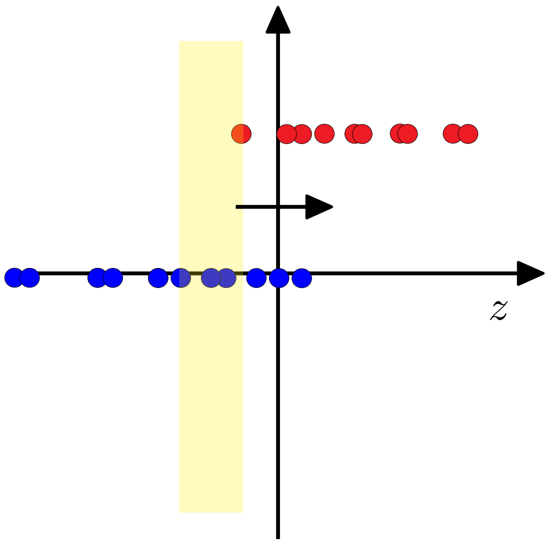


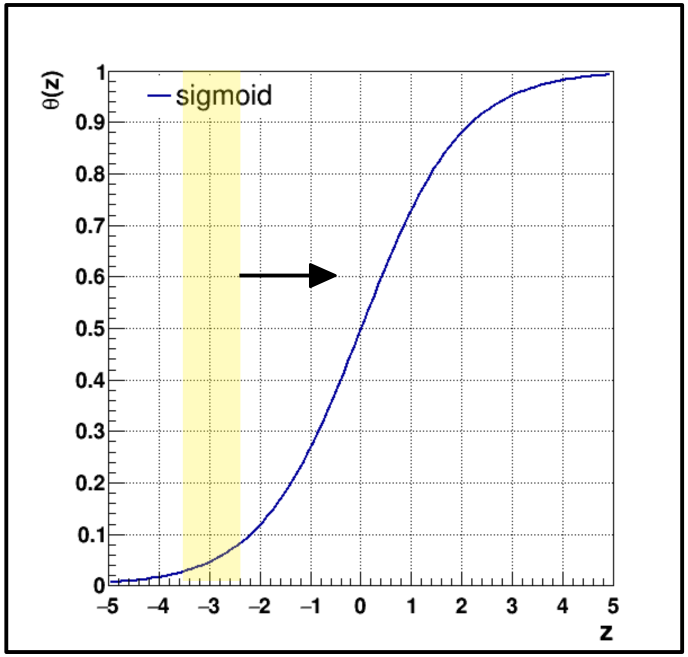


Imagine running through z with a window of constant width and counting the relative fractions of blue and red samples in that window. The sigmoid function represents to a good approximation the PDF to observe red.

Not linearly separable tasks

- We take the sigmoid function as an example of addressing not linearly separable tasks, here demonstrated with a 1d example:

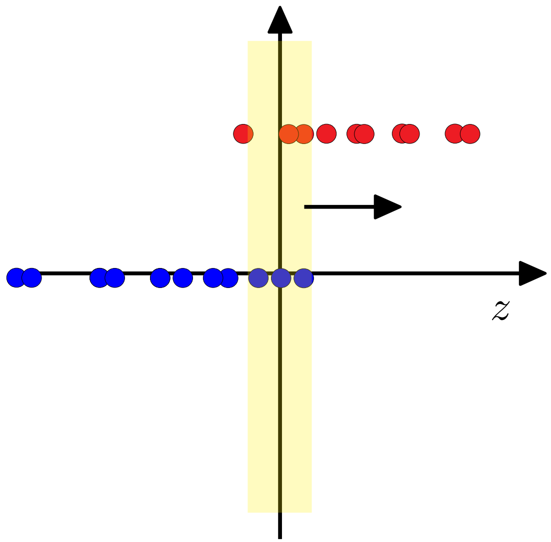


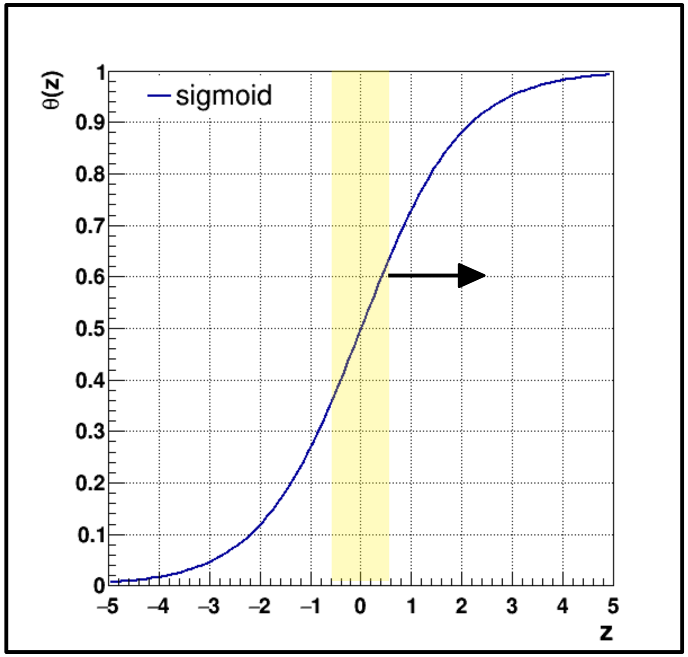


Imagine running through z with a window of constant width and counting the relative fractions of blue and red samples in that window. The sigmoid function represents to a good approximation the PDF to observe red.

Not linearly separable tasks

- We take the sigmoid function as an example of addressing not linearly separable tasks, here demonstrated with a 1d example:

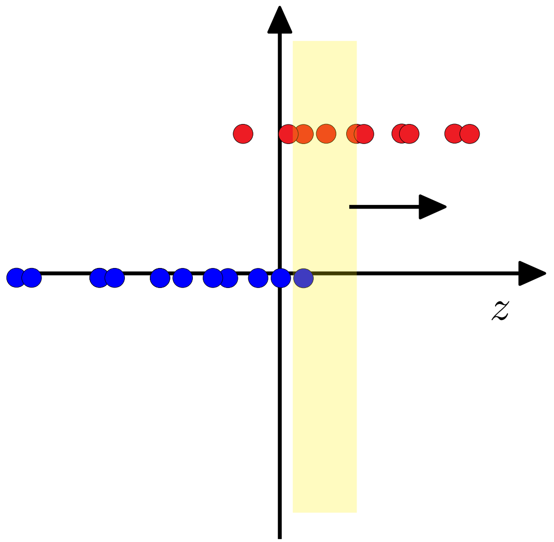


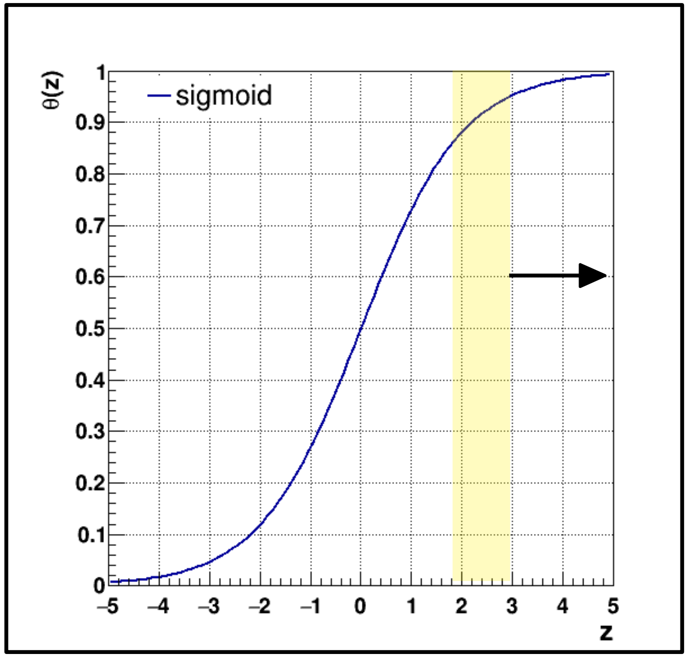


Imagine running through z with a window of constant width and counting the relative fractions of blue and red samples in that window. The sigmoid function represents to a good approximation the PDF to observe red.

Not linearly separable tasks

- We take the sigmoid function as an example of addressing not linearly separable tasks, here demonstrated with a 1d example:

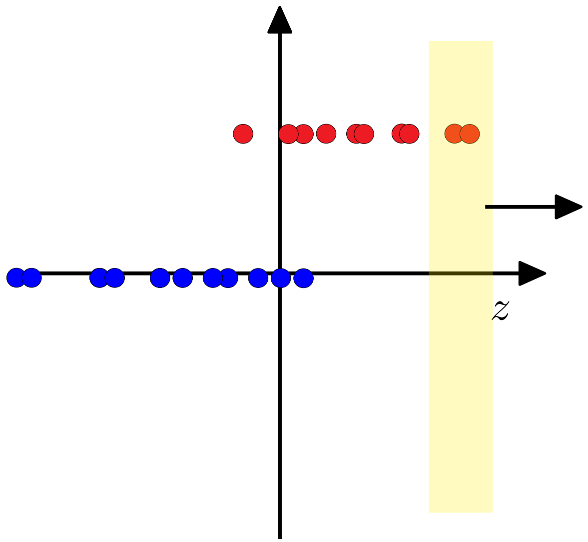


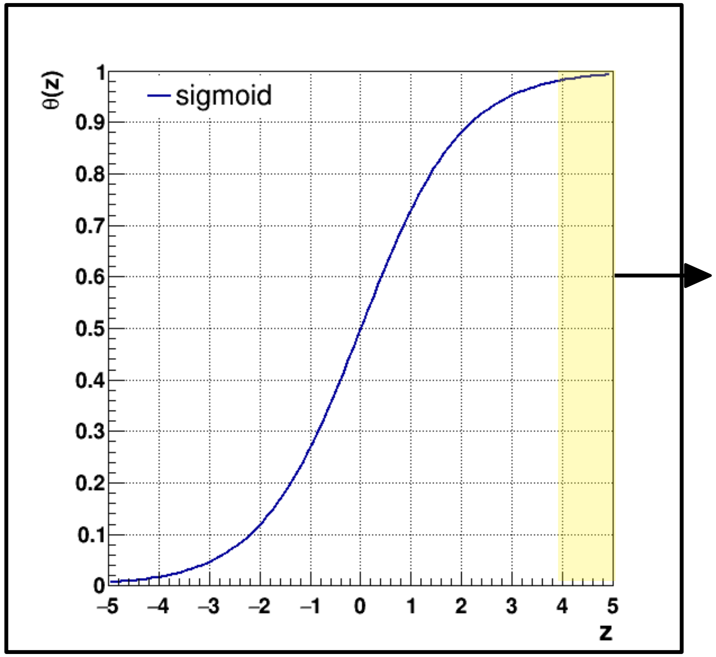


Imagine running through z with a window of constant width and counting the relative fractions of blue and red samples in that window. The sigmoid function represents to a good approximation the PDF to observe red.

Not linearly separable tasks

- We take the sigmoid function as an example of addressing not linearly separable tasks, here demonstrated with a 1d example:





Imagine running through z with a window of constant width and counting the relative fractions of blue and red samples in that window. The sigmoid function represents to a good approximation the PDF to observe red.

Summary

- Continuous activation functions allow transmission of information through hyperplanes.

- Gradual changes of the weights lead to gradual changes of the NN output. In this way the minimization task can in principal be solved analytically!

- **NB**: To solve the minimization task you also need the differentiability of the loss function.

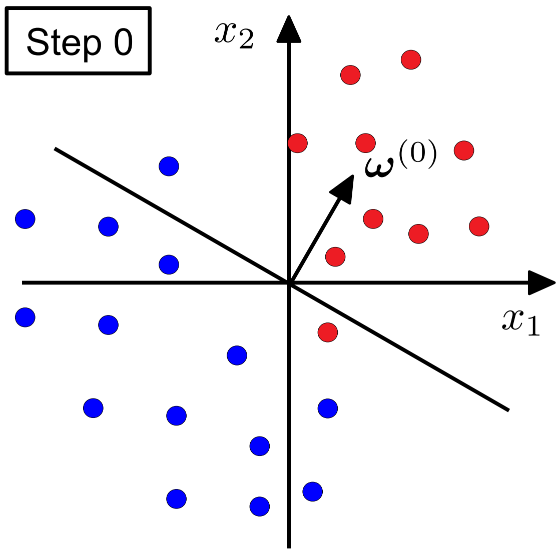
- For practical reasons neither the loss, nor the NN output function need to have a total derivative. It is sufficient if a gradual change in feature space leads to a gradual change in the NN output and loss function.

- An example of an activation function, which is not fully differentiable in all points of its input space is ReLU (slide 4).

Backup

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall \mathbf{x}$ in the plane (=on the boundary).



Algorithm:

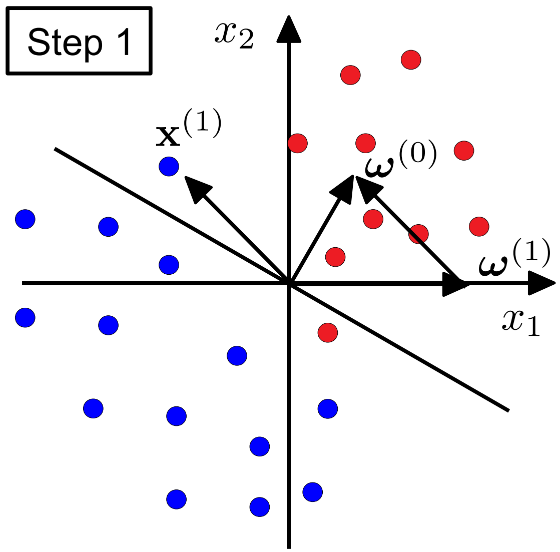
- Initialize weights randomly. Step 0
- Only update only examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall \mathbf{x}$ in the plane (=on the boundary).



Algorithm:

- Initialize weights randomly.
- Only update only examples w/ wrong predictions.
- For those, apply the following learning rule:

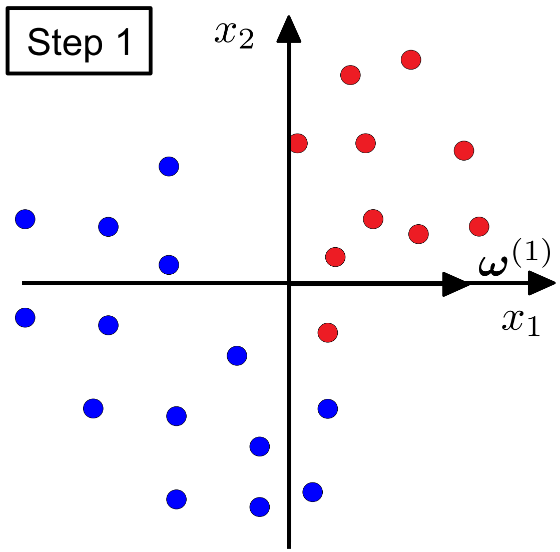
$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Step 1

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall \mathbf{x}$ in the plane (=on the boundary).



Algorithm:

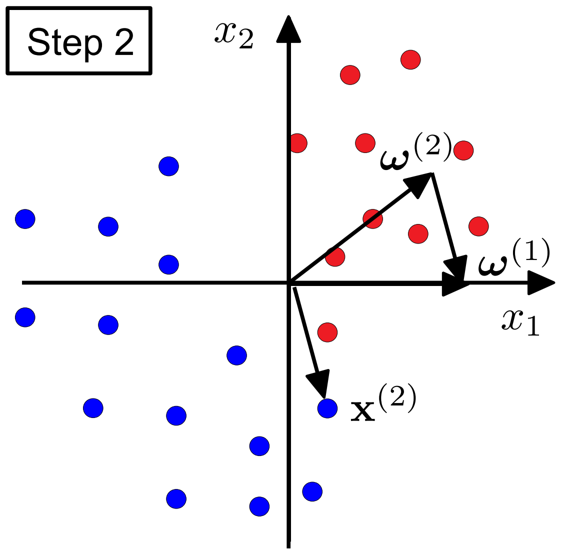
- Initialize weights randomly.
- Only update only examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \rightarrow \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Step 1

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall \mathbf{x}$ in the plane (=on the boundary).



Algorithm:

- Initialize weights randomly.
- Only update only examples w/ wrong predictions.
- For those, apply the following learning rule:

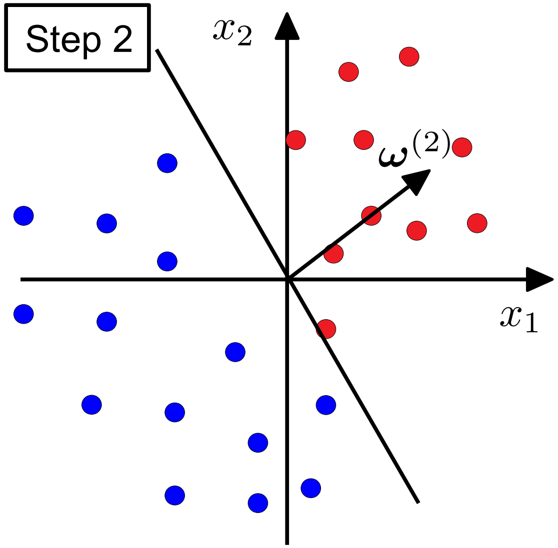
$$\boldsymbol{\omega}^{(k)} \to \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Step 2

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall \mathbf{x}$ in the plane (=on the boundary).



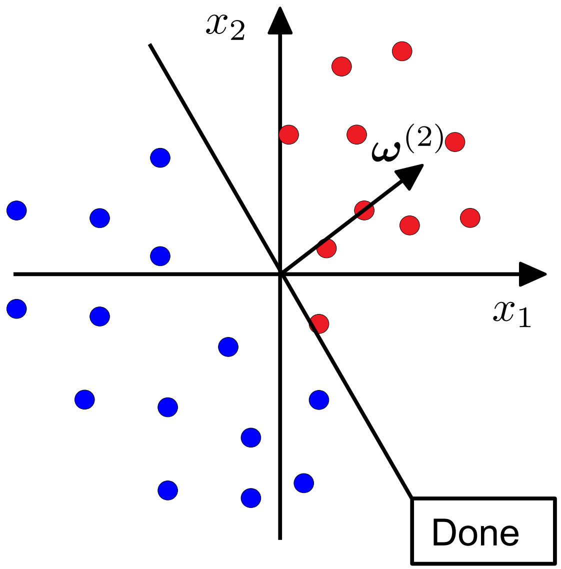
Algorithm:

- Initialize weights randomly.
- Only update only examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \rightarrow \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases} \quad \boxed{\text{Step 2}}$$

The perceptron learning rule

- **Solution**: Hyperplane in Hessian canonical form $\sum_i \omega_i x_i = 0$ i.e. $\boldsymbol{\omega} \perp \mathbf{x} \quad \forall\mathbf{x}$ in the plane (=on the boundary).



Algorithm:

- Initialize weights randomly.
- Only update only examples w/ wrong predictions.
- For those, apply the following learning rule:

$$\boldsymbol{\omega}^{(k)} \rightarrow \boldsymbol{\omega}^{(k+1)} = \begin{cases} \boldsymbol{\omega}^{(k)} + \mathbf{x}^{(k)} & \text{if red} \\ \boldsymbol{\omega}^{(k)} - \mathbf{x}^{(k)} & \text{if blue} \end{cases}$$

Notes on the lecture

- [1] In the previous lecture we have demonstrated that for the pentagon **only five** and not infitely many notes are necessary! This was the case, since for the pentagon the corners are connected by straight lines. In the slides following this annotation a general solution is discussed.

- [2] The difference to [1] is that here there is no alternative to filling the disconnected contours with dots.

Priv.-Doz. Dr. Roger Wolf
http://ekpwww.physik.uni-karlsruhe.de/~rwolf/