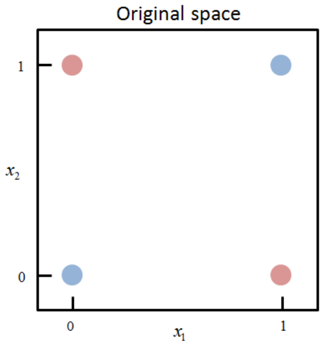
Deep Learning

XOR-Problem



Learning goals

- Example problem a single neuron can not solve but a single hidden layer net can

EXAMPLE: XOR PROBLEM

- Suppose we have four data points

$$X = \{(0,0)^\top, (0,1)^\top, (1,0)^\top, (1,1)^\top\}$$

- The XOR gate (exclusive or) returns true, when an odd number of inputs are true:

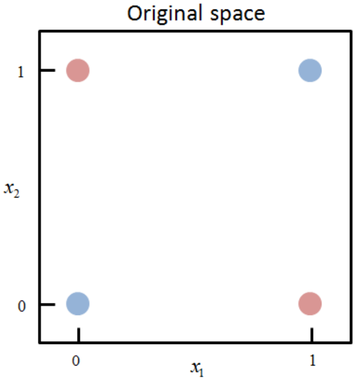
x_1	x_2	**XOR** $= y$
0	0	0
0	1	1
1	0	1
1	1	0

- Can you learn the target function with a logistic regression model?

EXAMPLE: XOR PROBLEM

- Logistic regression can not solve this problem. In fact, any model using simple hyperplanes for separation can not (including a single neuron).

- A small neural net can easily solve the problem by transforming the space!



EXAMPLE: XOR PROBLEM

- Consider the following model:

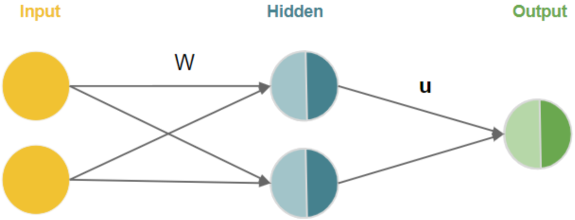


Figure: A neural network with two neurons in the hidden layer. The matrix $\mathbf{W}$ describes the mapping from $\mathbf{x}$ to $\mathbf{z}$. The vector $\mathbf{u}$ from $\mathbf{z}$ to y.

EXAMPLE: XOR PROBLEM

- Let use ReLU $\sigma(z) = \max\{0, z\}$ as activation function and a simple thresholding function $\tau(z) = [z > 0] = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$ as output transformation function. We can represent the architecture of the model by the following equation:

$$\begin{aligned} f(\mathbf{x} \mid \boldsymbol{\theta}) &= f(\mathbf{x} \mid \mathbf{W}, \mathbf{b}, \mathbf{u}, c) = \tau\left(\mathbf{u}^\top \sigma(\mathbf{W}^\top \mathbf{x} + \mathbf{b}) + c\right) \\ &= \tau\left(\mathbf{u}^\top \max\{0, \mathbf{W}^\top \mathbf{x} + \mathbf{b}\} + c\right) \end{aligned}$$

- So how many parameters does our model have?
  - In a fully connected neural net, the number of connections between the nodes equals our parameters:

$$\underbrace{(2 \times 2)}_{W} + \underbrace{(2 \times 1)}_{\mathbf{b}} + \underbrace{(2 \times 1)}_{\mathbf{u}} + \underbrace{(1)}_{c} = 9$$

EXAMPLE: XOR PROBLEM

$$\text{Let } \mathbf{W} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},\ \mathbf{b} = \begin{pmatrix} 0 \\ -1 \end{pmatrix},\ \mathbf{u} = \begin{pmatrix} 1 \\ -2 \end{pmatrix},\ c = -0.5$$

$$\mathbf{X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix},\ \mathbf{XW} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix},\ \mathbf{XW} + \boldsymbol{B} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

Note: $\mathbf{X}$ is a $(n \times p)$ design matrix in which the *rows* correspond to the data points. $\mathbf{W}$, as usual, is a $(p \times m)$ matrix where each *column* corresponds to a single (hidden) neuron. $\boldsymbol{B}$ is a $(n \times m)$ matrix with $\mathbf{b}$ duplicated along the rows.

$$X = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \qquad W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

EXAMPLE: XOR PROBLEM

$$\text{Let } \mathbf{W} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},\ \mathbf{b} = \begin{pmatrix} 0 \\ -1 \end{pmatrix},\ \mathbf{u} = \begin{pmatrix} 1 \\ -2 \end{pmatrix},\ c = -0.5$$

$$\mathbf{X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix},\ \mathbf{XW} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix},\ \mathbf{XW} + \boldsymbol{B} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

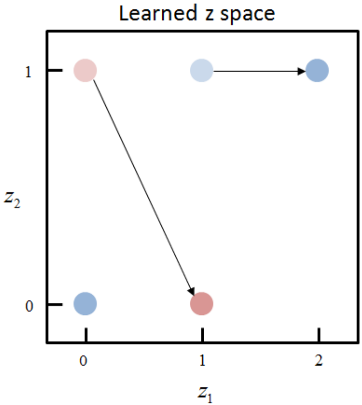
$$\boldsymbol{Z} = \max\{0, \mathbf{XW} + \boldsymbol{B}\} \quad = \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

- Note that we computed all examples at once.

EXAMPLE: XOR PROBLEM

- The input points are mapped into transformed space to

$$\boldsymbol{Z} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

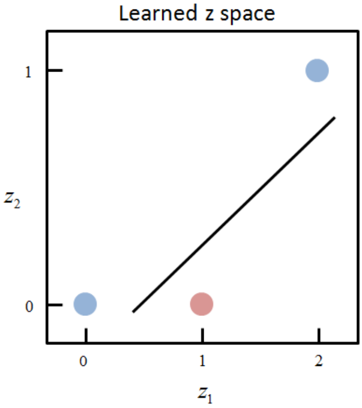


EXAMPLE: XOR PROBLEM

- The input points are mapped into transformed space to

$$\boldsymbol{Z} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

which is easily separable.



EXAMPLE: XOR PROBLEM

- In a final step we have to multiply the activated values of matrix $\boldsymbol{Z}$ with the vector $\mathbf{u}$ and add the bias term c:

$$f(\mathbf{x} \mid \mathbf{W}, \mathbf{b}, \mathbf{u}, c) \quad = \quad \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \end{pmatrix} + \begin{pmatrix} -0.5 \\ -0.5 \\ -0.5 \\ -0.5 \end{pmatrix} = \begin{pmatrix} -0.5 \\ 0.5 \\ 0.5 \\ -0.5 \end{pmatrix}$$

- And then apply the step function $\tau(z) = [z > 0]$. This solves the XOR problem perfectly!

x_1	x_2	**XOR** = y
0	0	0
0	1	1
1	0	1
1	1	0

NEURAL NETWORKS : OPTIMIZATION

- In this simple example we actually “guessed” the values of the parameters for **W**, **b**, **u** and *c*.
- That won’t work for more sophisticated problems!
- We will learn later about iterative optimization algorithms for automatically adapting weights and biases.
- An added complication is that the loss function is no longer convex. Therefore, there might not exist a single minimum.