WeST People and Knowledge Networks | Institute for Web Science & Technologies

UNIVERSITÄT KOBLENZ · LANDAU

Machine Learning and Data Mining WS21/22

"9 Neural Networks"

Dr. Zeyd Boukhers

@ZBoukhers

Institute for Web Science and Technologies

University of Koblenz-Landau

January 5, 2022

From previous lecture

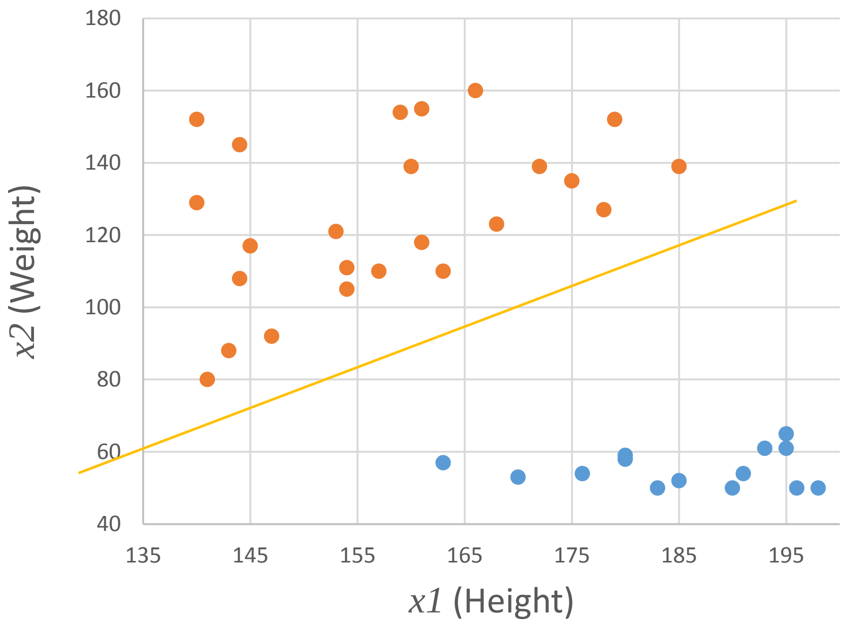
- Linear regression
  - Least squares function
  - Optimization
- Linear classification
  - Perceptron classifier
  - Support Vector Machine (SVM)
  - Optimization

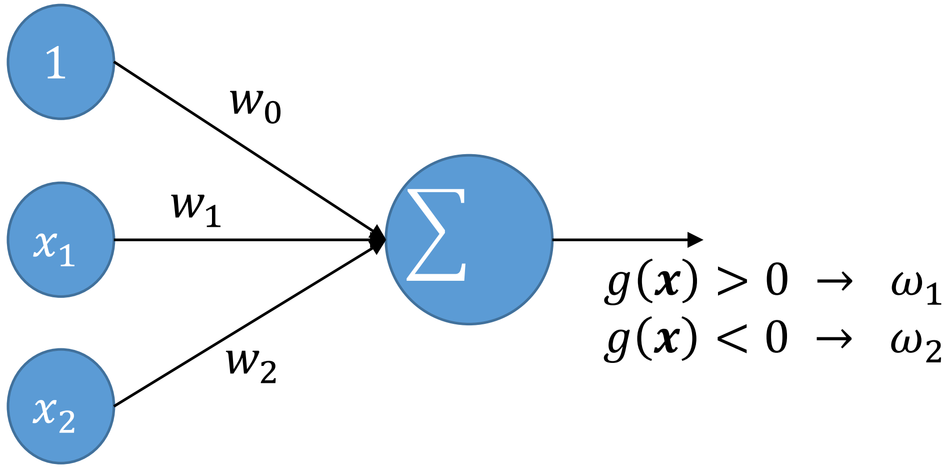
Agenda

- From Perceptron classifier to Multi-Layer Perceptron
  - XOR problem
  - Activation function
  - Two-Layer Perceptron
  - Gradient Decent
  - Backpropagation

Perceptron classifier

- Perceptron: the aim is to learn a hyperplane $g(\boldsymbol{x}) = w_0 + \boldsymbol{w}^T\boldsymbol{x}$ that separates the classes.
- $g(\boldsymbol{x}) = w_0 + w_1x_1 + w_2x_2$

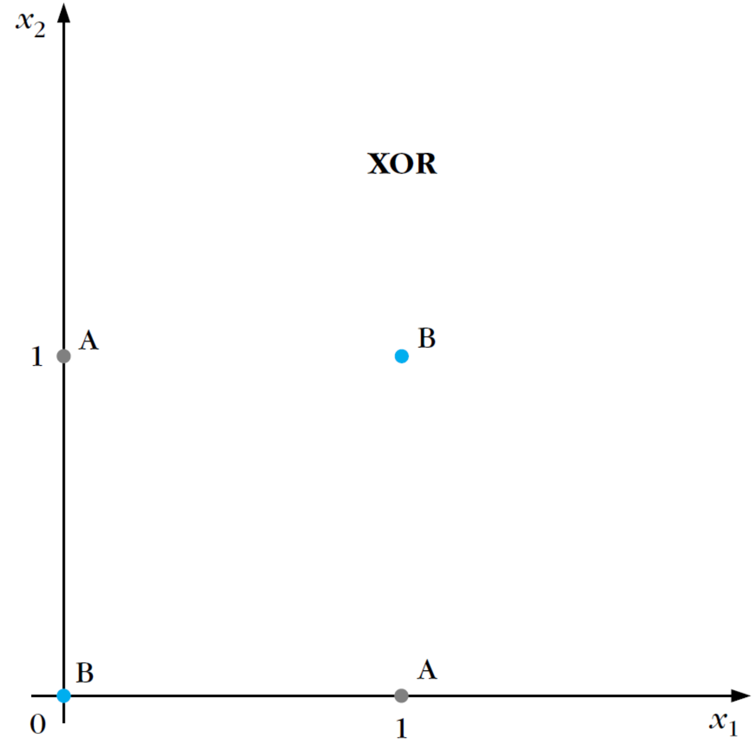




XOR problem

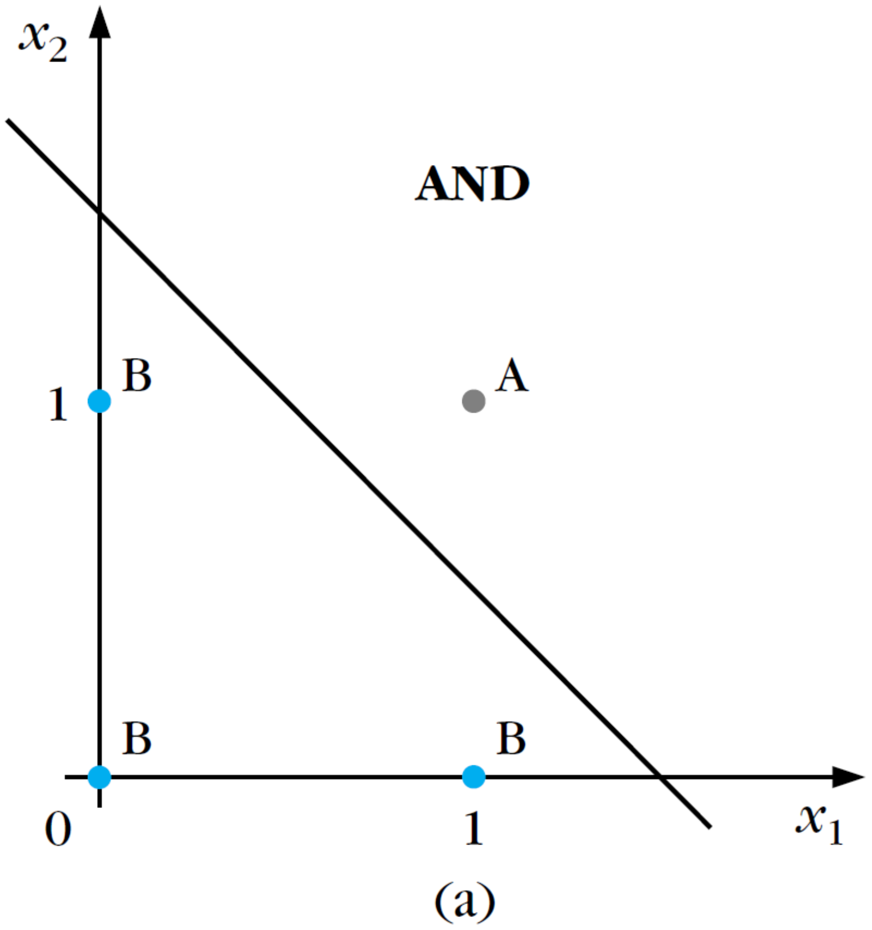
- The Exclusive OR (XOR) Boolean function is a typical example of such a problem.
- Given the values of the input binary data $\boldsymbol{x} = [x_1, x_2]^T$, the output of the function is either 0 or 1, corresponding to the classes ω_1 and ω_2, respectively.

x_1	x_2	XOR	Class
0	0	0	$\omega_1(B)$
0	1	1	$\omega_2(A)$
1	0	1	$\omega_2(A)$
1	1	0	$\omega_1(B)$



Source: Theodoridis, S., & Koutroumbas, K. "Pattern recognition. " Fourth Edition, 9781597492720, 2008

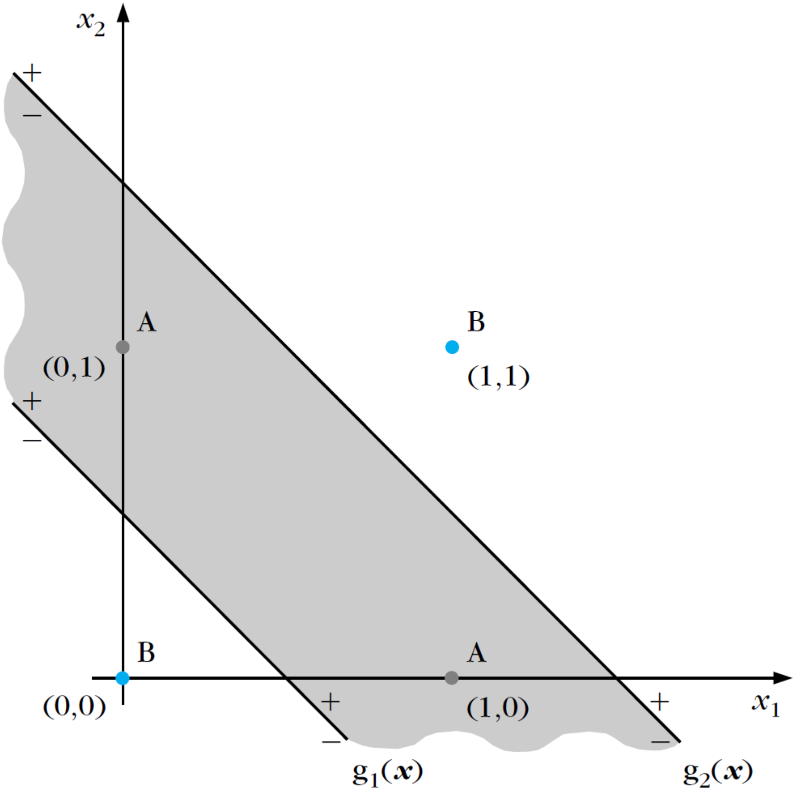
XOR problem



Source: Theodoridis, S., & Koutroumbas, K. “Pattern recognition. ” Fourth Edition, 9781597492720, 2008

XOR problem

- Now the classes can be separated with the two lines: $g_1(\boldsymbol{x}) = g_2(\boldsymbol{x}) = 0$
  - A is to the right $(+)$ of $g_1(\boldsymbol{x})$ and to the left $(-)$ of $g_2(\boldsymbol{x})$
  - B is to the right of both lines.
  - or to the left of both lines.

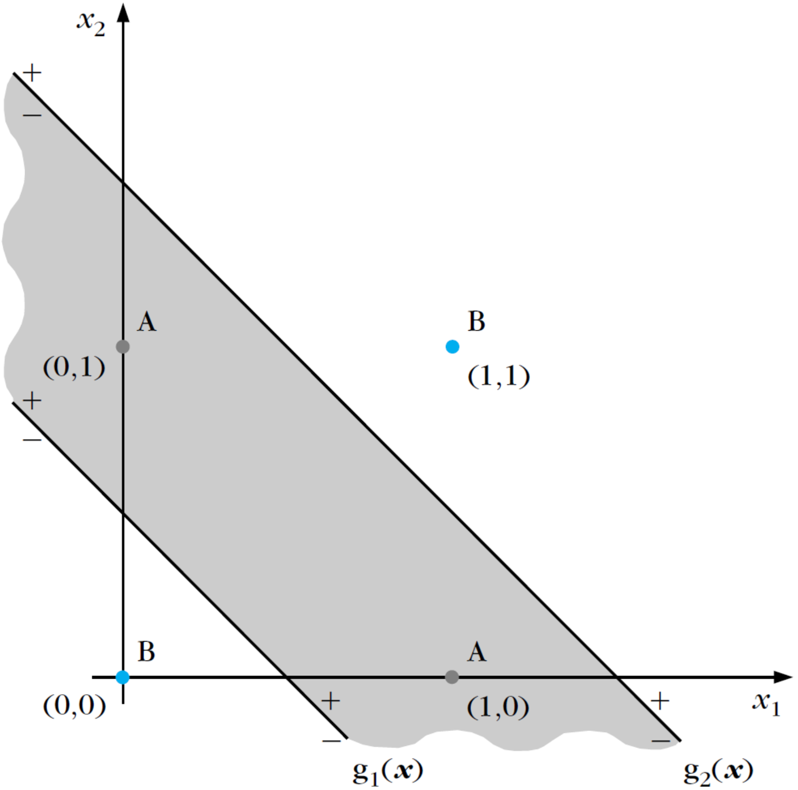


➢The problem has been attacked
In *two successive* phases.

Source: Theodoridis, S., & Koutroumbas, K. “Pattern recognition. ” Fourth Edition, 9781597492720, 2008

XOR problem

- In the first phase, the position of x is computed w.r.t each line.
- In the second phase, the results of the previous phase are combined to find the position of x w.r.t both lines.
  - outside or inside the shaded area.

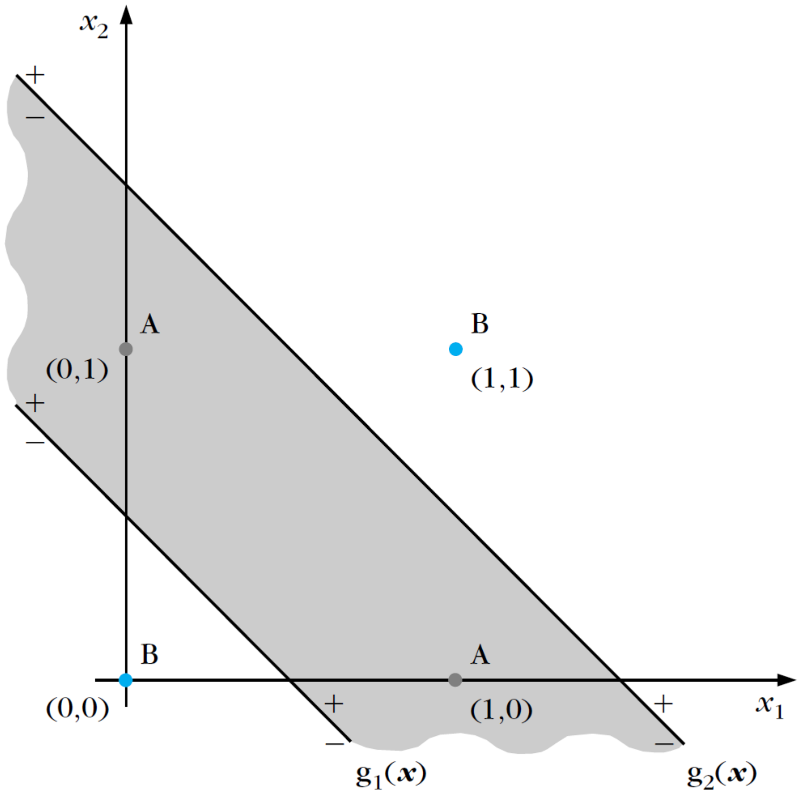


Source: Theodoridis, S., & Koutroumbas, K. “Pattern recognition. ” Fourth Edition, 9781597492720, 2008

XOR problem

- In the first phase, two perceptrons are adopted with input x_1 and x_2 and appropriate synaptic weights.
- $y_u = f\big(g_u(\boldsymbol{x})\big), u = 1,2$
  - $f(.)$ is the activation function.
  - $f(\boldsymbol{x}) = \begin{cases} 1 \text{ if } g(\boldsymbol{x}) \geq 0 \\ 0 \text{ if } g(\boldsymbol{x}) < 0 \end{cases}$

Phase 1				Phase 2	
x_1	x_2	y_1	y_2	XOR	Class
0	0	0	0	0	$\omega_1(B)$
0	1	1	0	1	$\omega_2(A)$
1	0	1	0	1	$\omega_2(A)$
1	1	1	1	0	$\omega_1(B)$

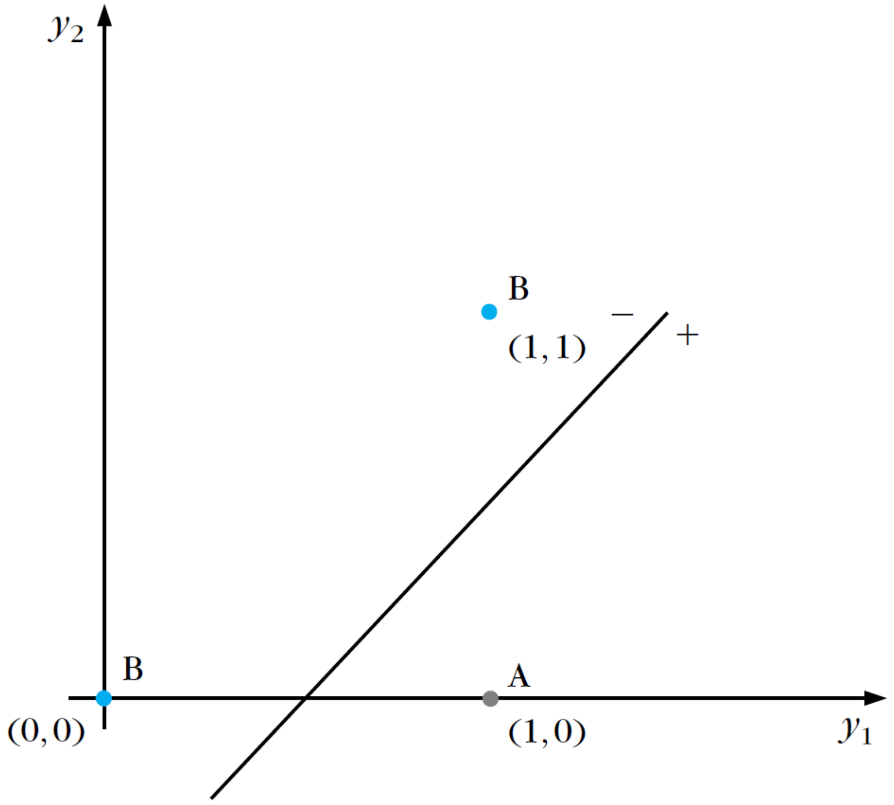


Source: Theodoridis, S., & Koutroumbas, K. “Pattern recognition. ” Fourth Edition, 9781597492720, 2008

XOR problem

- In the first phase, the input vector $\boldsymbol{x}$ is mapped to a new one $\boldsymbol{y} = [y_1, y_2]$.
- In the second phase, the decision is made based on the transformed data.
  - $\boldsymbol{y} = [0,0\,] \rightarrow \omega_1$
  - $\boldsymbol{y} = [1,1\,] \rightarrow \omega_1$
  - $\boldsymbol{y} = [1,0\,] \rightarrow \omega_2$

Phase 1				Phase 2	
x_1	x_2	y_1	y_2	XOR	Class
0	0	0	0	0	$\omega_1(B)$
0	1	1	0	1	$\omega_2(A)$
1	0	1	0	1	$\omega_2(A)$
1	1	1	1	0	$\omega_1(B)$



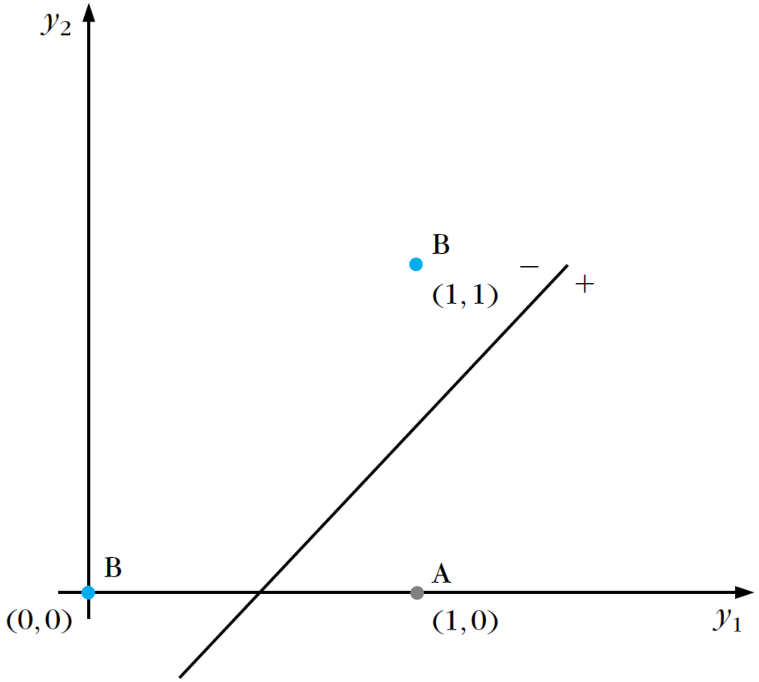
Source: Theodoridis, S., & Koutroumbas, K. "Pattern recognition. " Fourth Edition, 9781597492720, 2008

XOR problem

- What happens?
  - The first phase transforms (by mapping $\boldsymbol{x}$ to $\boldsymbol{y}$) the nonlinearly separable problem to a linearly separable one.
- This multilayer architecture is a generalization of the perceptron.

➢ *two-layer perceptron.*

Phase 1				Phase 2	
x_1	x_2	y_1	y_2	XOR	Class
0	0	0	0	0	$\omega 1(B)$
0	1	1	0	1	$\omega 0(A)$
1	0	1	0	1	$\omega 0(A)$
1	1	1	1	0	$\omega 1(B)$



Source: Theodoridis, S., & Koutroumbas, K. "Pattern recognition. " Fourth Edition, 9781597492720, 2008

Activation function

- Why is it important?
  - Without it, the Multi-Layer Perceptron (MLP) is just a composition of successive linear functions.

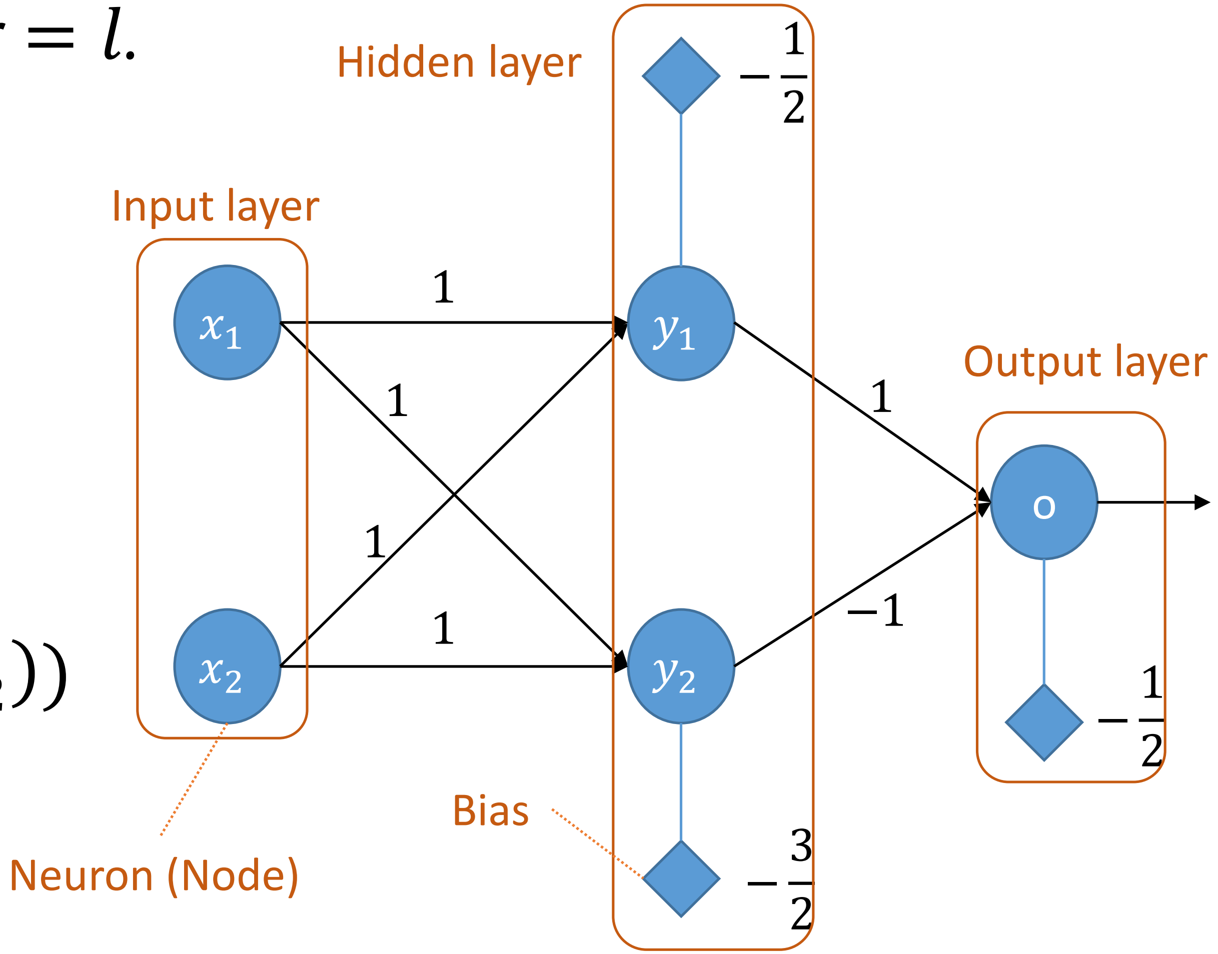
  ➢ A linear function.

✓ A non linear function (activation function) that introduces non-linearity into the model.

"If the activation functions of all the hidden units in a network are taken to be linear, then for any such network we can always find an equivalent network without hidden units." [9] (Bishop 2006)

Two-layer perceptron

- It is also called "*two-layer feedforward neural network*".
- The number of nodes in the input layer $= l$.
- $g_1(\boldsymbol{x}) = x_1 + x_2 - \frac{1}{2} = 0$
- $g_2(\boldsymbol{x}) = x_1 + x_2 - \frac{3}{2} = 0$
- $g(\boldsymbol{y}) = y_1 - y_2 - \frac{1}{2} = 0$
- $g(\boldsymbol{y}, \widetilde{\boldsymbol{w}}) = f(g_1(\boldsymbol{x}, \widetilde{\boldsymbol{w}}_1)) + f(g_2(\boldsymbol{x}, \widetilde{\boldsymbol{w}}_2))$



Two-layer perceptron

- $g(\boldsymbol{y}; \widetilde{\boldsymbol{w}}) = f(g_1(\boldsymbol{x}; \widetilde{\boldsymbol{w}}_1)) + f(g_2(\boldsymbol{x}; \widetilde{\boldsymbol{w}}_2))$
  - Where $\widetilde{\boldsymbol{w}}_j = [\boldsymbol{w}_j{}^T, b_j], \quad \forall j$
  - Let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_l]$
- $g_1(\boldsymbol{X}; \widetilde{\boldsymbol{w}}_1) = \boldsymbol{w}_1{}^T \boldsymbol{X} + b = [1,1]^T \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} - \frac{1}{2} = \begin{bmatrix} 0 & 1 & 1 & 2 \end{bmatrix} - \frac{1}{2} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{3}{2} \end{bmatrix}$
- $f(g_1(\boldsymbol{x}, \widetilde{\boldsymbol{w}}_1)) = \begin{bmatrix} 0 & 1 & 1 & 1 \end{bmatrix}$

Compute $f(g_2(\boldsymbol{x}, \widetilde{\boldsymbol{w}}_2))$
Compute $f(g(\boldsymbol{y}, \widetilde{\boldsymbol{w}}))$

- Let $\widetilde{\boldsymbol{W}} = [\widetilde{\boldsymbol{w}}_1, \widetilde{\boldsymbol{w}}_2]$

$$f(g(y)) = f\left(\widetilde{\boldsymbol{w}}^T f\left(\widetilde{\boldsymbol{W}}^T \boldsymbol{x}\right)\right)$$

Multi-layer perceptron

- The architecture of *two-layer perceptron* can be generalized
  - to l-dimensional input vectors and
  - to more than two neurons in the hidden layer and
  - to more than one neuron in the output layer
  - to more than two layers

Learning

- In other words, how to find the weights?
  - Linear regression $\rightarrow$ Linear programming (linear optimization)
  - SVM $\rightarrow$ Quadratic programming
  - Perceptron $\rightarrow$ Iterative optimization (Non-linear function)
    - Gradient descent.

  - MLP ?
    - Gradient descent
    - Stochastic gradient descent

Maximum likelihood estimation

- Given N training samples, let:
  - $\boldsymbol{\theta}$ be the set of parameters (to be found)
  - $p(\hat{y}_i|\boldsymbol{x}_i;\boldsymbol{\theta})$ be the likelihood of the true label of $\boldsymbol{x}_i$

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p(\hat{y}_i|\boldsymbol{x}_i;\boldsymbol{\theta})$$

$$= \sum_{i=1}^{N} \log(p(\hat{y}_i|\boldsymbol{x}_i;\boldsymbol{\theta}))$$

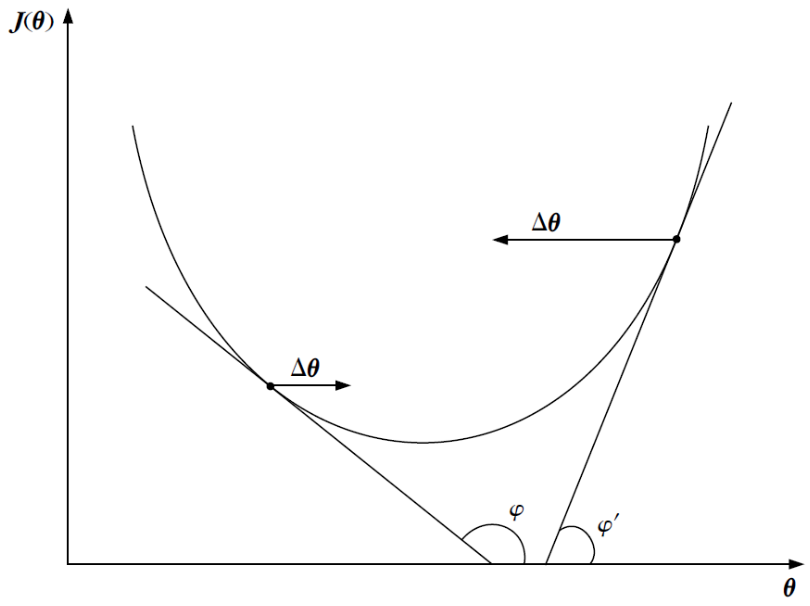
Maximum likelihood estimation

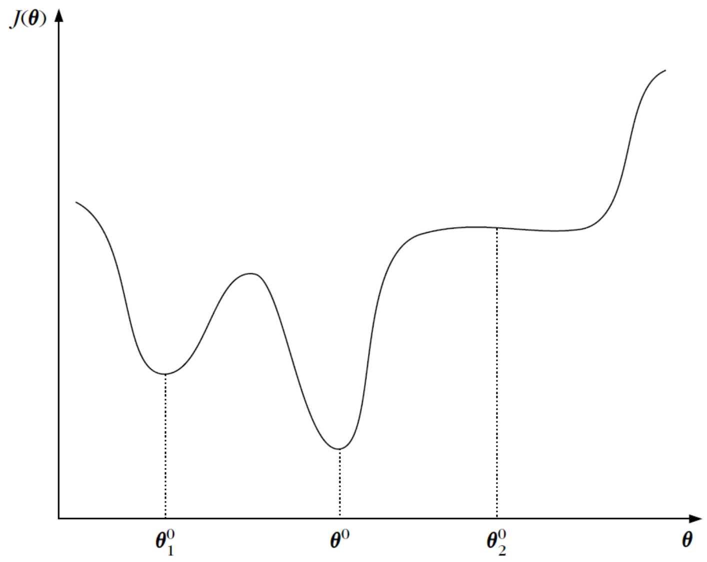
- Maximizing the log likelihood of a categorical distribution (classification) corresponds to minimising the cross-entropy between the approximated distribution and the true distribution.
  - $H(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_i y_i \log(\hat{y}_i)$
  - $H(\hat{\boldsymbol{y}}, \boldsymbol{y}) \neq H(\boldsymbol{y}, \hat{\boldsymbol{y}})$

- Maximising the log of a continuous distribution (regression) corresponds to minimising the mean squared error between the approximated mean and true mean.

Gradient descent

- Goal: finding $\boldsymbol{\theta}$ that minimizes $J(\boldsymbol{\theta})$
- How: Iterative process
  - $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{old} + \Delta\boldsymbol{\theta}$
  - $\Delta\boldsymbol{\theta} = -\rho \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{old}}$
  - $\boldsymbol{\theta}_{new}$ is chosen in the direction that decreases $J(\boldsymbol{\theta})$
  - ρ is the learning rate
    - High $\rho \rightarrow$ big step
    - Low $\rho \rightarrow$ small step
  - Convergence: when the gradient $= 0$

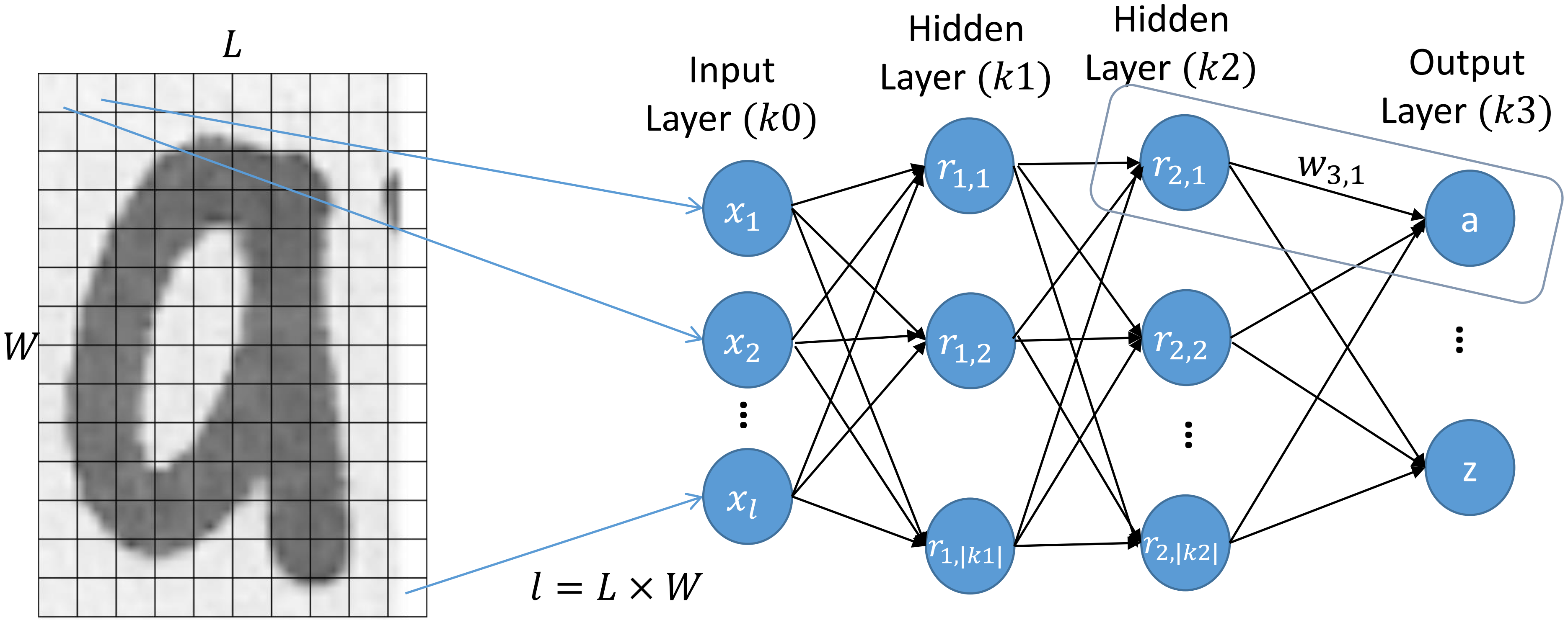




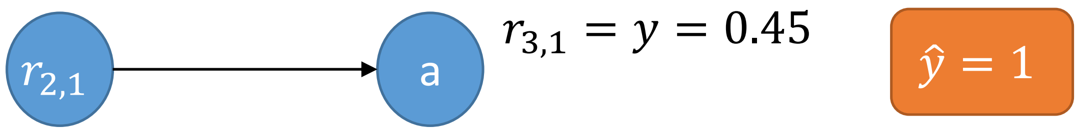
Source: Theodoridis, S., & Koutroumbas, K. “Pattern recognition. ” Fourth Edition, 9781597492720, 2008

How to build MLP?

- Given the architecture below, where all neurones employ the same activation function (sigmoid)
- We assume that N training samples $(\boldsymbol{x}_i, \hat{y}_i), i = 1, \ldots, N$ are available.

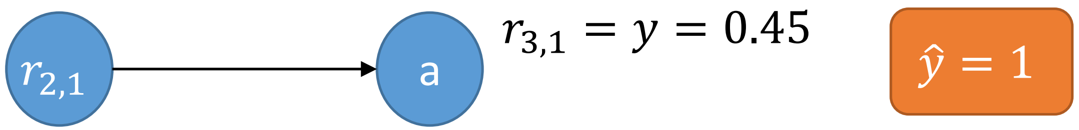


Learning (Optimization)



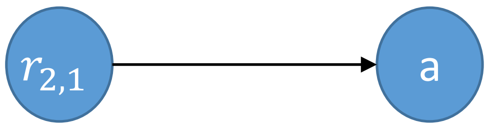
- Since the output is no longer scalar, it will be $\boldsymbol{y} = \left[y_1, y_2, \dots, y_{|k3|}\right]^T$
  - The label will be $\widehat{\boldsymbol{y}} = \left[\hat{y}_1 = 1, \hat{y}_2 = 0, \dots, \hat{y}_{|k3|} = 0\right]^T$
- The goal is to minimize the cost function $J = \sum_{i=1}^{N} \varepsilon(\mathbf{y}_i, \widehat{\boldsymbol{y}}_i)$
  - For all training samples.
  - Let $\varepsilon(.,.)$ be the MSE: $\varepsilon(\mathbf{y}_i, \widehat{\boldsymbol{y}}_i) = \sum_{j=1}^{k3} \left(\mathrm{y}_{i,j} - \hat{y}_{i,j}\right)^2$

Learning (Optimization)



- Let's consider one sample, one neurone per layer:
  - $\varepsilon(y_{i,1}, \hat{y}_{i,1})$ depends on $y_{i,1}$(or $r_{3,1}$)
  - Let get rid of i
  - $y_1 = r_{3,1} = f(g(r_{2,1})) = \sigma\left(g(r_{2,1})\right) = \sigma(r_{2,1}w_{3,1} + b_{3,1})$
    - For simplicity, w_0 is substituted with b
  - $f(g(r_{2,1}))$ depends on $g(r_{2,1})$
  - $g(r_{2,1})$ depends on:
    - $r_{2,1}$
    - $\tilde{\boldsymbol{w}}_{3,1}$ (including the bias)
      - This means the weights of the first neuron in the third layer.

Learning (Optimization)



$r_{3,1} = y = 0.45$

$\hat{y} = 1$

- By the chain rule:

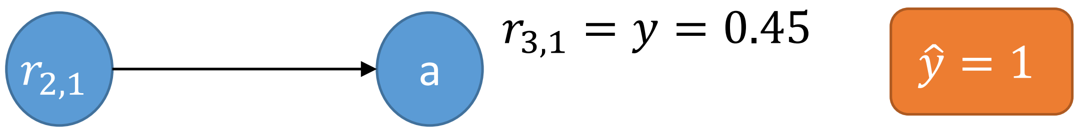
$$\frac{\partial \varepsilon(y_1, \hat{y}_1)}{\partial w_{3,1}} = \frac{\partial \varepsilon(y_1, \hat{y}_1)}{\partial f\big(g(r_{2,1})\big)} \frac{\partial f\big(g(r_{2,1})\big)}{\partial g(r_{2,1})} \frac{\partial g(r_{2,1})}{\partial w_{3,1}}$$

  - $\frac{\partial \varepsilon(y_1,\hat{y}_1)}{\partial f\left(g(r_{2,1})\right)} = 2(y_1 - \hat{y}_1)$
  - $\frac{\partial f\left(g(r_{2,1})\right)}{\partial g(r_{2,1})} = \sigma'\left(g(r_{2,1})\right)$
  - $\frac{\partial g(r_{2,1})}{\partial w_{3,1}} = r_{2,1}$

Remember:

$$\Delta w_{3,1} = -\rho \left.\frac{\partial \varepsilon(y_1, \hat{y}_1)}{\partial w_{3,1}}\right|_{w_{3,1}=w_{3,1_{old}}}$$

Learning (Optimization)



- In the subsequent iteration:

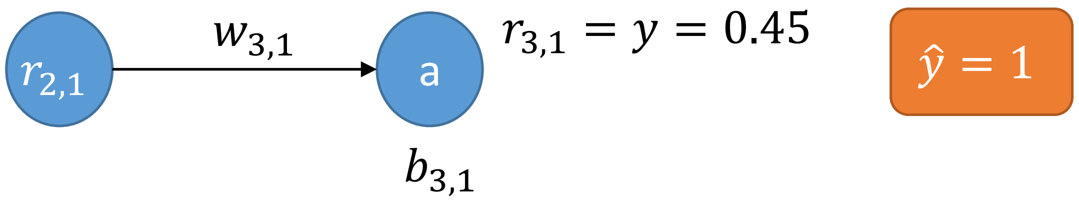
$$w_{3,1}(new) = w_{3,1}(old) - \rho\, 2(y_1 - \hat{y}_1)\sigma'\left(g\left(r_{2,1}\right)\right) r_{2,1}$$

- For N samples:

$$w_{3,1}(new) = w_{3,1}(old) - \rho \sum_{i=1}^{N} 2(y_{i,1} - \hat{y}_{i,1})\sigma'\left(g\left(r_{2,1}\right)\right) r_{2,1}$$

We can do the same thing for b

Learning (Optimization)

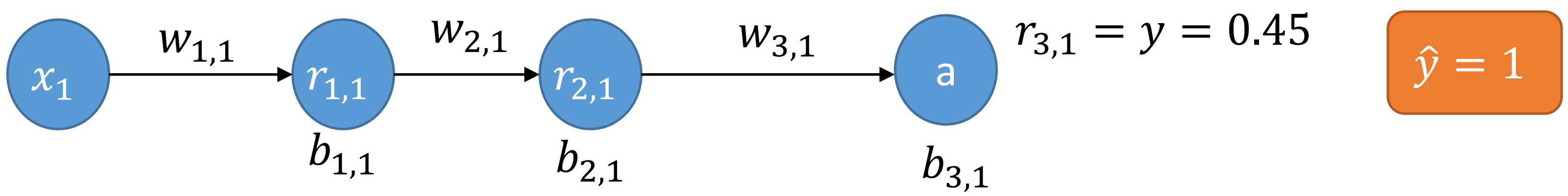


- Since we don't have only one sample but N samples:

$$\frac{\partial \varepsilon\left(y_{i=1:N,1}, \hat{y}_{i=1:N,1}\right)}{\partial w_{3,1}} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \varepsilon\left(y_{i,1}, \hat{y}_{i,1}\right)}{\partial w_{3,1}}$$

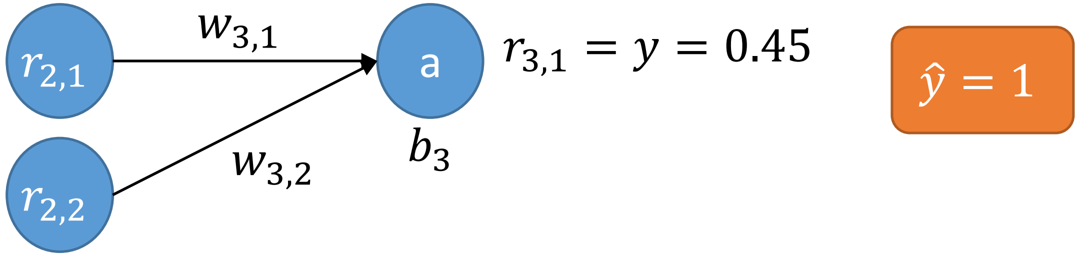
- $\varepsilon(y_1, \hat{y}_1)$ is also sensitive to $r_{2,1}$:
    - $\frac{\partial \varepsilon(y_1,\hat{y}_1)}{\partial r_{2,1}} = \frac{\partial \varepsilon(y_1,\hat{y}_1)}{\partial f\left(g(r_{2,1})\right)} \frac{\partial f\left(g(r_{2,1})\right)}{\partial g(r_{2,1})} \frac{g(r_{2,1})}{\partial r_{2,1}}$
    - $\frac{\partial g(r_{2,1})}{\partial r_{2,1}} = w_{3,1}$

Backpropagation



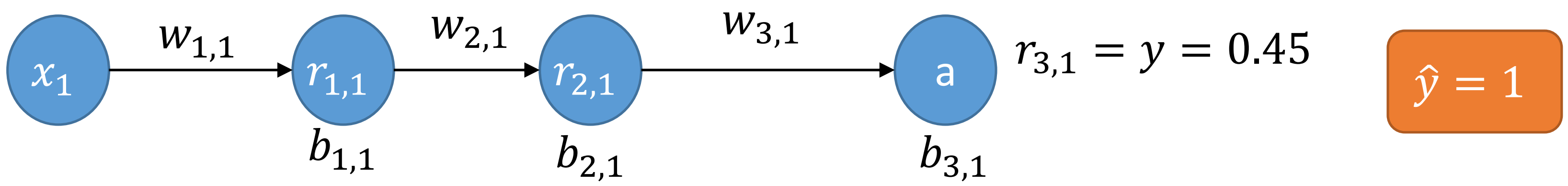
- But the cost function is also sensitive to previous layers:
  - $\varepsilon(y_1, \hat{y}_1)$ depends on y_1(or $r_{3,1}$)
  - $y_1 = r_{3,1} = f\left(g(r_{2,1})\right) = \sigma\left(g\left(r_{2,1}\right)\right) = \sigma\left(r_{2,1}w_{3,1} + b_{3,1}\right)$
  - $f\left(g(r_{2,1})\right)$ depends on $g(r_{2,1})$
  - $g\left(r_{2,1}\right)$ depends on $r_{2,1}$ and $\tilde{\boldsymbol{w}}_{3,1}$
  - $r_{2,1} = f\left(g(r_{1,1})\right)$
  - $f\left(g(r_{1,1})\right)$ depends on $g(r_{1,1})$
  - $g(r_{1,1})$ depends on $r_{1,1}$ and $\tilde{\boldsymbol{w}}_{2,1}$

Learning (Optimization)



- We have two neurones in $k2$:
  - $y_1 = r_{3,1} = f\big(g(r_{2,1}, r_{2,2})\big) = \sigma\big(\sum_{j=1}^{2}(r_{2,j}w_{3,j}) + b_3\big)$
  - We generalize this for any number of neurones:
  - $r_{k,u} = \sigma\left(\sum_{j=1}^{|k-1|}(r_{k-1,j}w_{k,j}) + b_{k,u}\right)$

Learning (Optimization)



- $\nabla \varepsilon\left(y_{i=1:N,1}, \hat{y}_{i=1:N,1}\right)_{\widetilde{\boldsymbol{W}}} = \left[\frac{\partial \varepsilon\left(y_{i=1:N,1}, \hat{y}_{1:N,1}\right)}{\partial w_{1,1}}, \frac{\partial \varepsilon\left(y_{1:N,1}, \hat{y}_{1:N,1}\right)}{\partial b_{1,1}}, \ldots, \frac{\partial \varepsilon\left(y_{1:N,1}, \hat{y}_{1:N,1}\right)}{\partial w_{3,1}}, \frac{\partial \varepsilon\left(y_{1:N,1}, \hat{y}_{1:N,1}\right)}{\partial b_{3,1}}\right]^T$

Summary

Summary

- From Perceptron classifier to Multi-Layer Perceptron
  - XOR problem
  - Activation function
  - Two-Layer Perceptron
- Learning (Optimization)
  - Gradient Decent
  - Backpropagation

Thank you!

Zeyd Boukhers

E-mail: Boukhers@uni-koblenz.de
Phone: +49 (0) 261 287-2765
Web: Zeyd.Boukhers.com

University of Koblenz-Landau
Universitätsstr. 1
56070 Koblenz