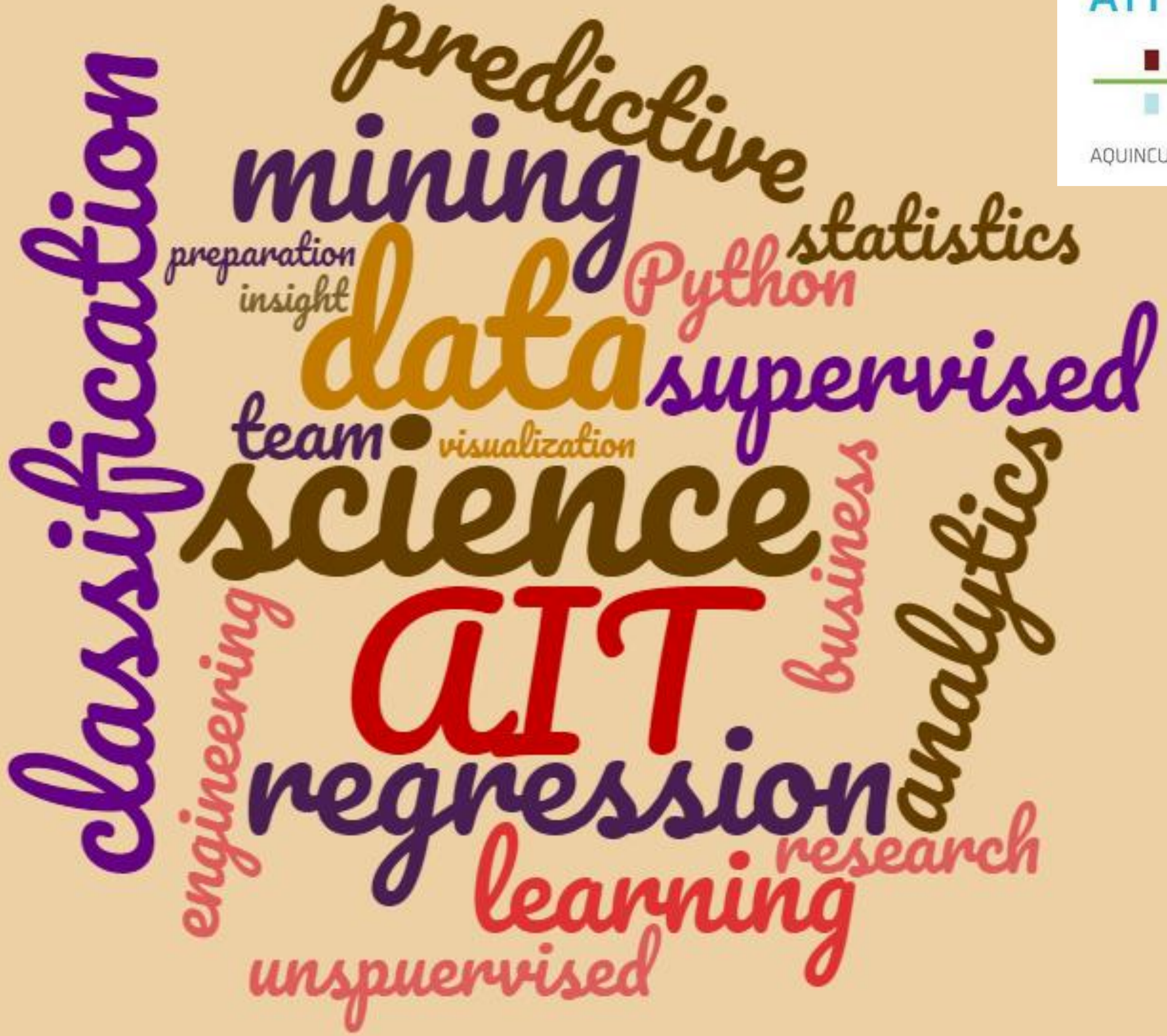
AIT-BUDAPEST
AQUINCUM INSTITUTE OF TECHNOLOGY

Data Science



Dr. Roland Molontay

April 21, 2023.
Neural networks

Schedule of the semester

	Monday midnight	*Tuesday class*	*Friday class*
W1 (02/06)			
W2 (02/13)		**HW1 out**	
W3 (02/20)			
W4 (02/27)	**HW1 deadline + TEAMS**	**HW2 out**	
W5 (03/06)			PROJECT PLAN
W6 (03/13)	**HW2 deadline**	**HW3 out**	
W7 (03/20)			**MIDTERM**
SPRING BREAK		SPRING BREAK	SPRING BREAK
W8 (04/03)	**HW3 deadline**		GOOD FRIDAY
W9 (04/10)	MILESTONE 1		
W10 (04/17)		**HW4 out**	
W11 (04/24)			
W12 (05/01)	**HW4 deadline**		
W13 (05/08)	MILESTONE 2		
W14 (05/15)		**FINAL**	**PROJECT presentations**
W15 (05/22)		**PROJECT presentations**	

Artificial Neural Networks (ANN)

- It can be used for approximating any function
  - So it can solve both classification and regression problems
- The algorithm mimics the functioning of interconnected neurons

Machine learning with neural networks

The three main steps of machine learning (in general):

1. Hypothesis: non-linear hypothesis function
2. Cost function: Usual cost (error) function for classification/regression – misclassification error, logarithmic error, hinge, squared error, absolute error
3. Optimization: Gradient descent, stochastic gradient descent, other minimization methods
    - For neural network the cost function is usually non-convex ➔ it is possible to get stuck in local minima

Feature extraction

- Until now we defined the features by hand
  - Linear: x_i
  - Polynomial: $x_i^2, x_j^3 x_i^2$
  - Other non-linear functions (pl. radial function, logarithm etc.): $\log(x_i)$ etc.
  - Then the model learned linear hypothesis, the weights **θ** (or **w**):

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \theta^T \phi(\boldsymbol{x}) \quad \phi\colon \mathbb{R}^n \to \mathbb{R}^k$$

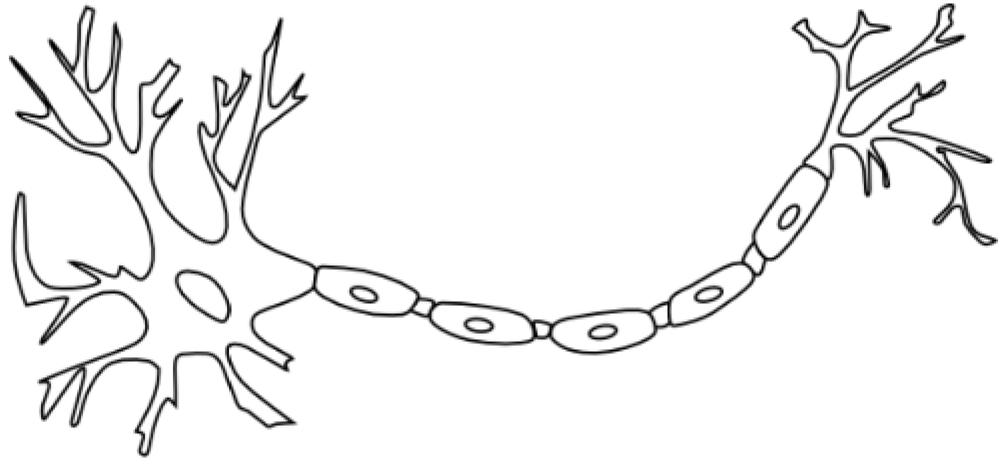
    - where ϕ is a function that transforms the variables, if $k > n$: transformation to higher dimension
- Goal: having an algorithm that is able to extract features automatically

Expert-defined features for hand-written digit recognition, e.g.:

- number of „on" pixels
- average of the vertical coordinates of the „on" pixels
- variance of the horizonal coordinates
- correlation between the horizontal and vertical positions of „on" pixels
- …

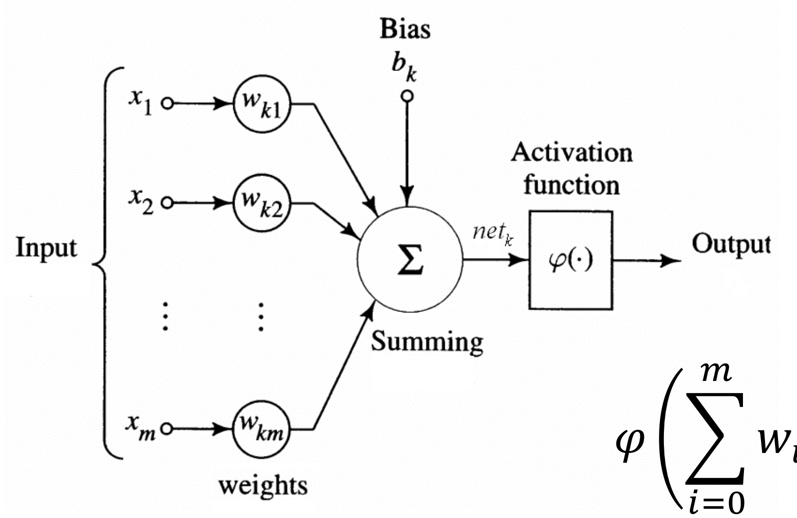
Biological motivation: neuron

- Neurons receive signals via dendrites (usually more dendrites), if the intensity of the stimuli is above a threshold then the neuron generate and propagate an electrical signal (an action potential) and the potential travels along the axon to its connections
  - There are inputs (signals via dendrites)
  - The cell body „process" the inputs
  - The output is the action potential that is transmitted along the axon

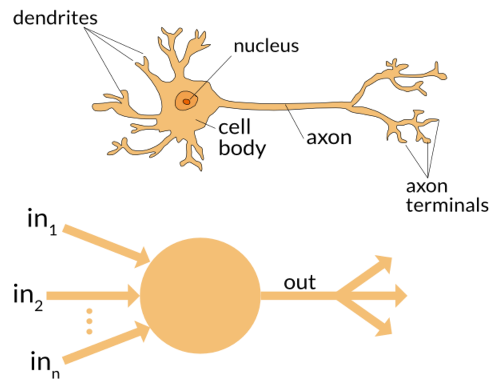


Perceptron

- Input nodes: where the input arrives
  - They correspond to the attributes
  - Special input node is the constant ($x_0 = 1$, bias)
- Weights of the inputs
  - These weights should be learned by the algorithm
- Summing
  - Summation of the weighted inputs
- Activation
  - We apply the activation function to the weighted sum of the inputs
- Output
  - The result of the activation function
  - It corresponds to the class label / target variable



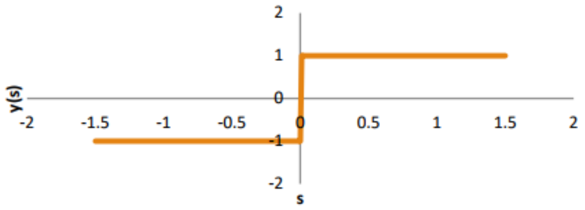
$$\varphi\left(\sum_{i=0}^{m} w_i x_i\right)$$



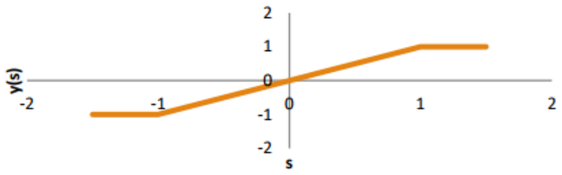
Activation functions

- Sign function
- Piecewise linear function
- Tangent hyperbolic (K=2)
- Sigmoid function
- Hinge

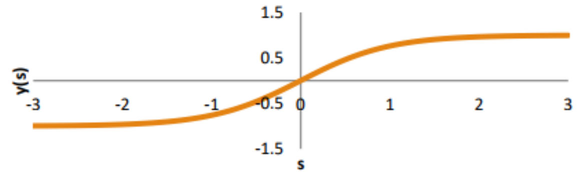
$$y = \begin{cases} 1, & s > 0 \\ -1, & s \leq 0 \end{cases}$$



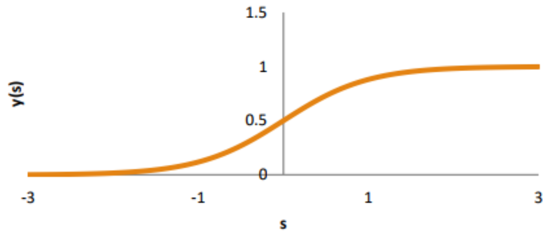
$$y = \begin{cases} 1, & s > 1 \\ s, & -1 \leq s \leq 1 \\ -1, & s \leq -1 \end{cases}$$



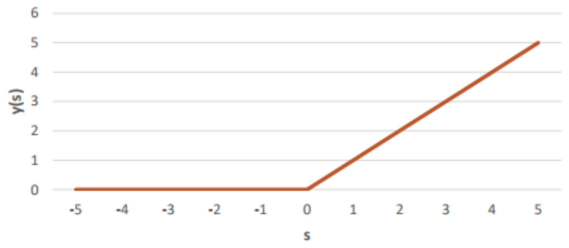
$$y = \frac{1 - e^{-Ks}}{1 + e^{-Ks}}; \; K > 0$$



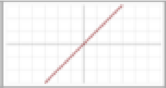
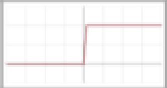
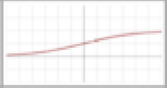
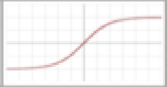
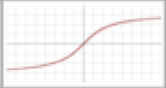
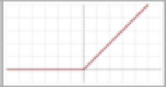
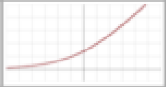
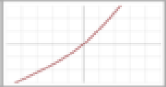
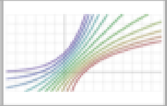
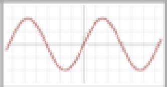
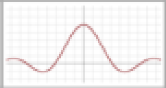
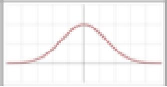
$$y = \frac{1}{1 + e^{-Ks}}; \; K > 0$$



$$y = \max(0, s)$$



Many other possible activation function

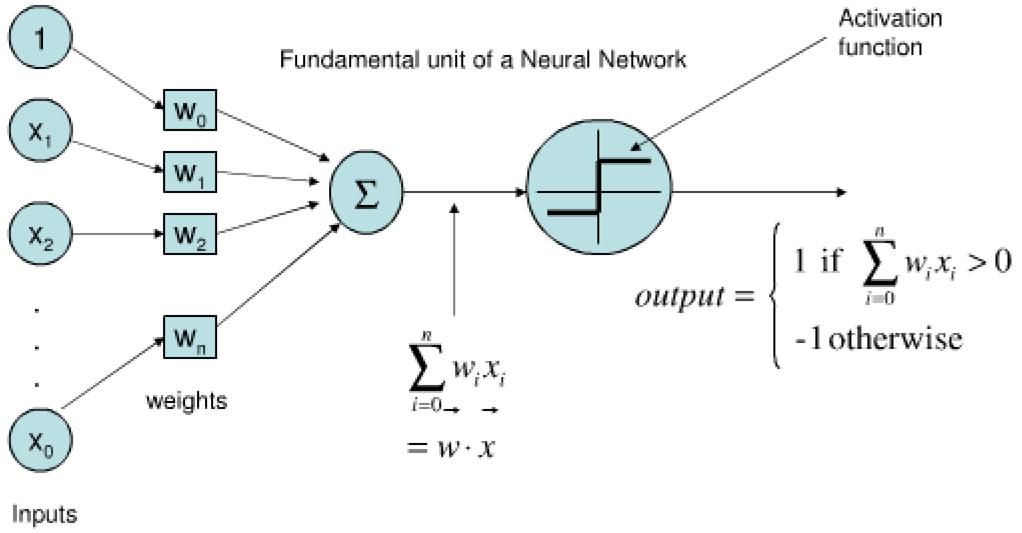
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$
Bent identity		$f(x) = \frac{\sqrt{x^2+1}-1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2+1}} + 1$
SoftExponential		$f(\alpha, x) = \begin{cases} -\frac{\log_e(1-\alpha(x+\alpha))}{\alpha} & \text{for} \quad \alpha < 0 \\ x & \text{for} \quad \alpha = 0 \\ \frac{e^{\alpha x}-1}{\alpha} + \alpha & \text{for} \quad \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1-\alpha(\alpha+x)} & \text{for} \quad \alpha < 0 \\ e^{\alpha x} & \text{for} \quad \alpha \geq 0 \end{cases}$
Sinusoid		$f(x) = \sin(x)$	$f'(x) = \cos(x)$
Sinc		$f(x) = \begin{cases} 1 & \text{for} \quad x = 0 \\ \frac{\sin(x)}{x} & \text{for} \quad x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for} \quad x \neq 0 \end{cases}$
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$

The connection of perceptron to other algorithms

- If the activation function is the sigmoid function then the perceptron algorithm agrees with logistic regression
  - Providing that the cost function is the logarithmic loss

$$\varphi\left(\sum_{i=0}^{n} w_i x_i\right)$$

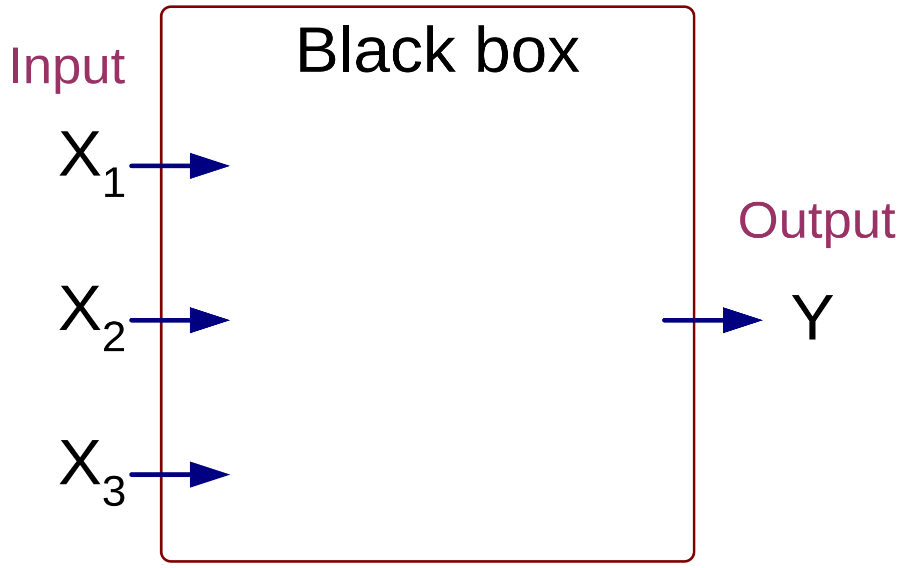
- If the activation function is the identity, then it is the same as linear regression
- Perceptron (with usual activation functions) is a linear algorithm, thus it creates linear decision boundary



Fundamental unit of a Neural Network
Activation function
1
X1
X2
Xn
Inputs
W0
W1
W2
Wn
weights
Σ
$\sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$
$output = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Training perceptron to learn logical function

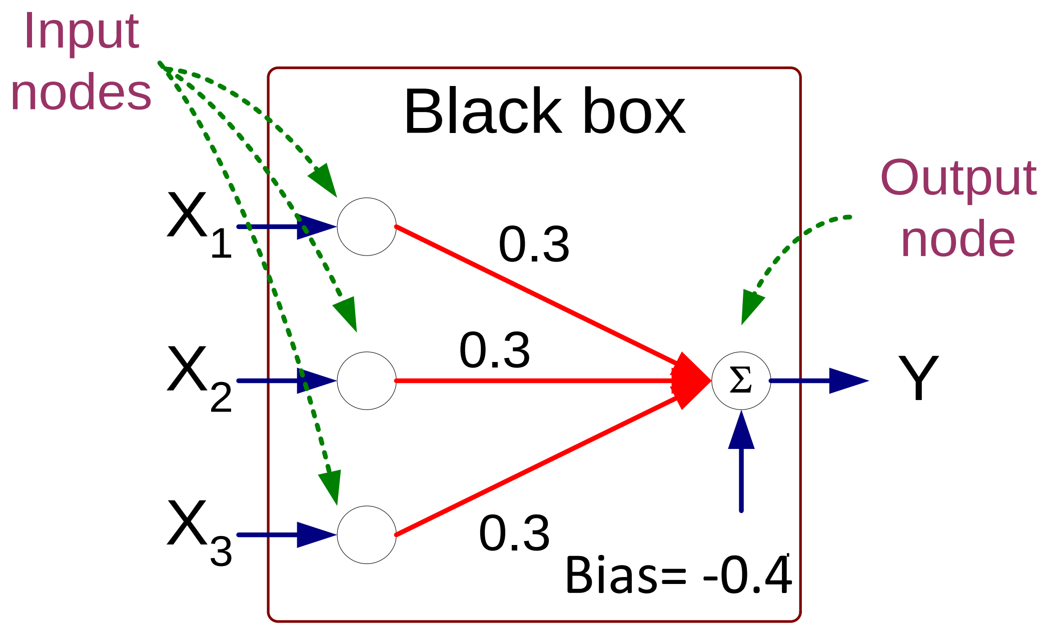
X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



- The output (Y) is 1, if and only if at least two input parameters are 1

Training perceptron to learn logical function II.

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



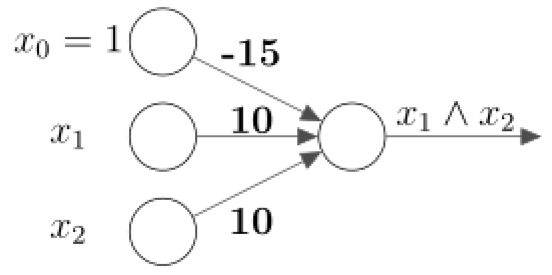
$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

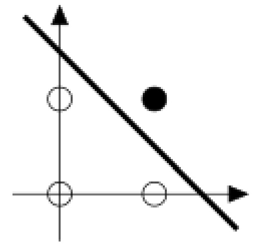
$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ true} \\ 0 & \text{otherwise} \end{cases}$$

Examples – other logical functions

- Similarly to the previous slide, let's find the perceptron that is able to represent AND and OR functions for two input variables
  - x_1 AND x_2
  - x_1 OR x_2
- It is enough to find the equation of the separating line

AND

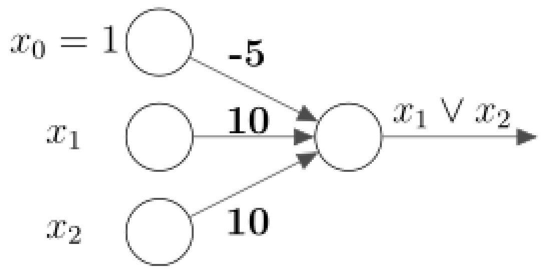


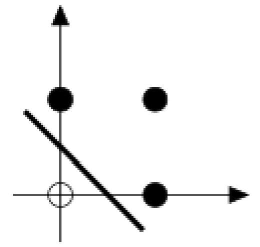


$x_2 = -x_1 + 1{,}5$

$-1{,}5x_0 + x_1 + x_2 = 0$

OR



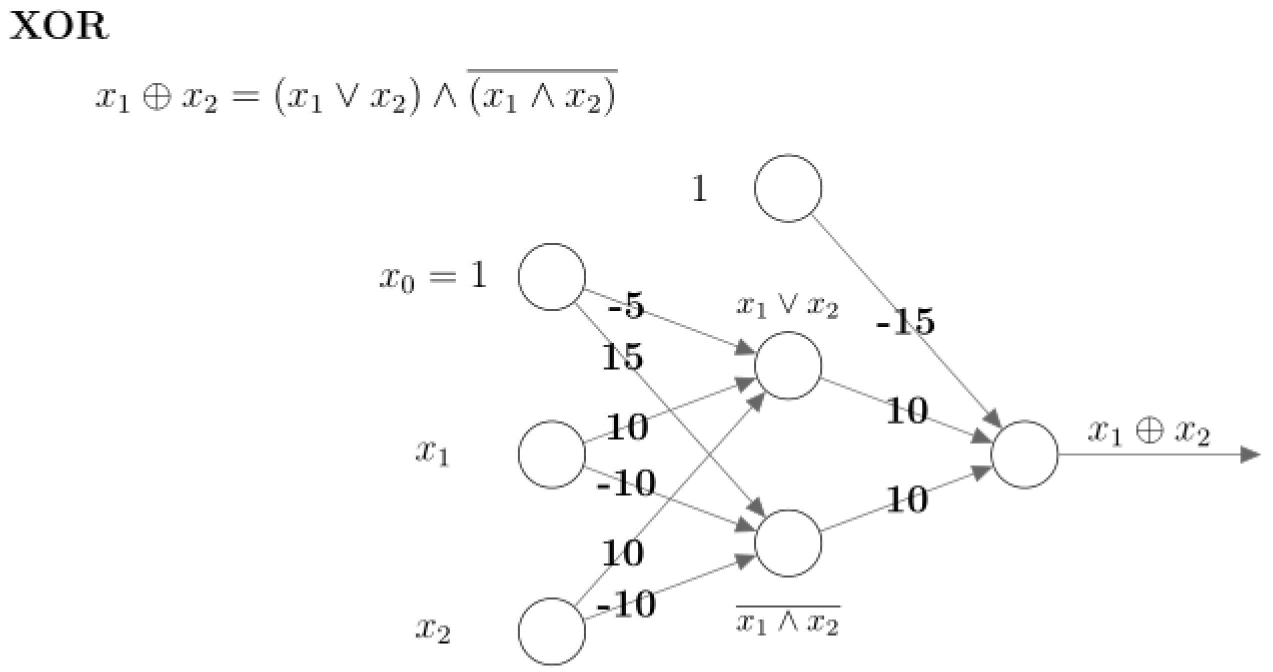


$x_2 = -x_1 + 0{,}5$

$-0{,}5x_0 + x_1 + x_2 = 0$

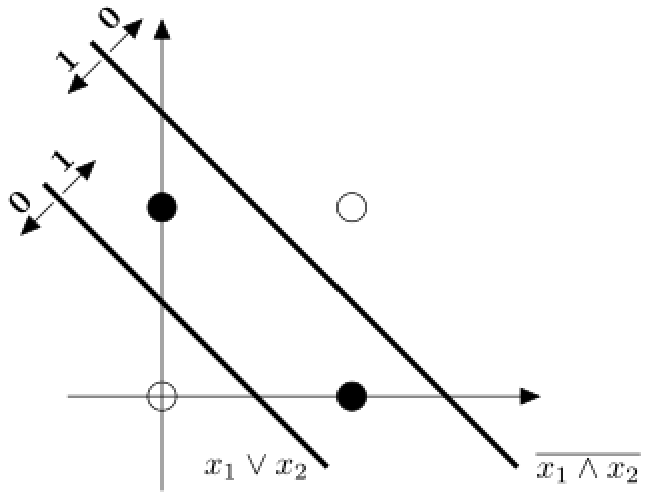
Example – XOR function

- We have seen that XOR is not linearly separable
  - One neuron is not able to represent is
  - Using AND and OR functions, it can be represented
    - Let us combine AND and OR to build a neural network by combining the neurons


XOR
$x_1 \oplus x_2 = (x_1 \vee x_2) \wedge \overline{(x_1 \wedge x_2)}$
1
$x_0 = 1$
x_1
x_2
-5
15
10
-10
10
-10
$x_1 \vee x_2$
$\overline{x_1 \wedge x_2}$
-15
10
10
$x_1 \oplus x_2$

Example– XOR function II. – feature extraction

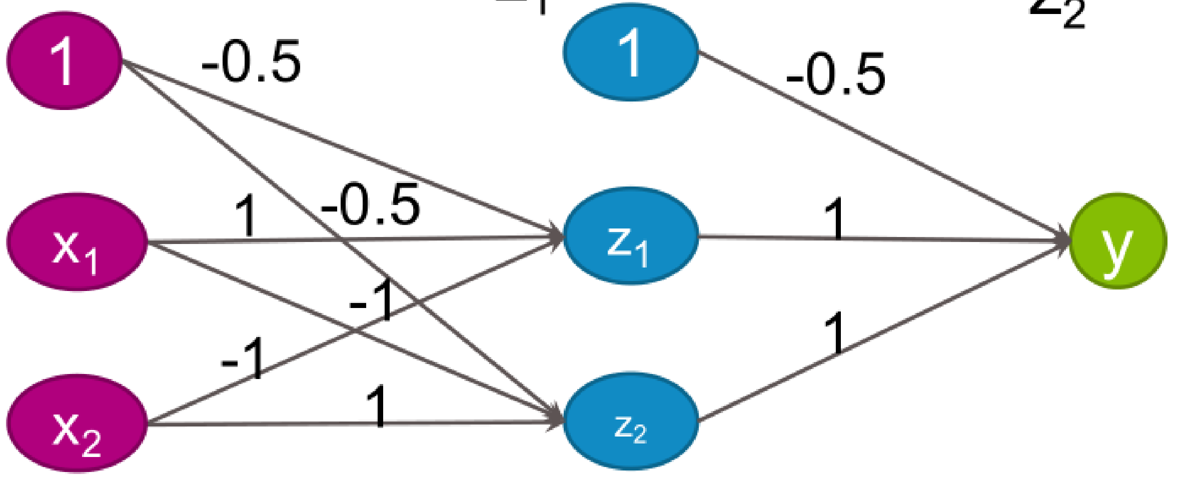
- One neuron was not enough but by using a so-called hidden layer we were able to represent XOR function
- The hidden layer can be considered as new features ($x_1 \lor x_2, \overline{x_1 \land x_2}$) extracted from original features (x_1, x_2)
  - Neural networks with hidden layer(s) are able to extract new features automatically



Example – XOR function III.

- XOR function can be represented by other neural networks as well (with the help of a different logical identity)
- The principle is the same

$$\text{XOR} = \underbrace{x_1 \text{ AND NOT } x_2}_{z_1} \text{ OR } \underbrace{\text{NOT } x_1 \text{ AND } x_2}_{z_2}$$



Problem

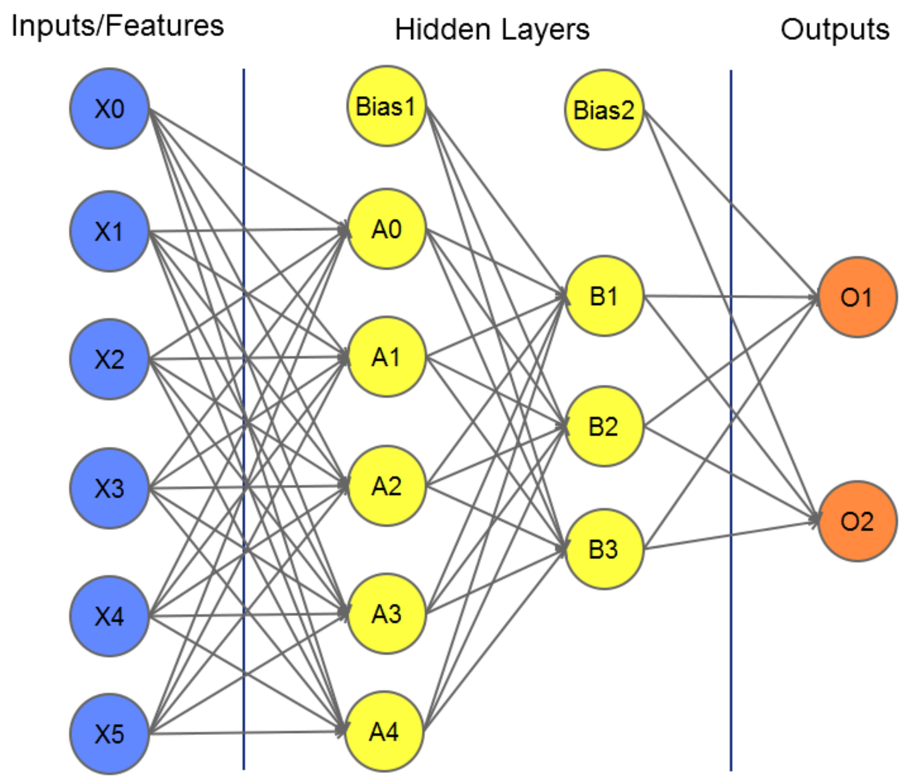
Represent the following logical functions with a perceptron or show that it is not possible to do so. In the latter case, construct a neural network with one hidden layer.

a) A **AND** B **AND** C

b) (A **XOR** B) **AND** (A **OR** B)

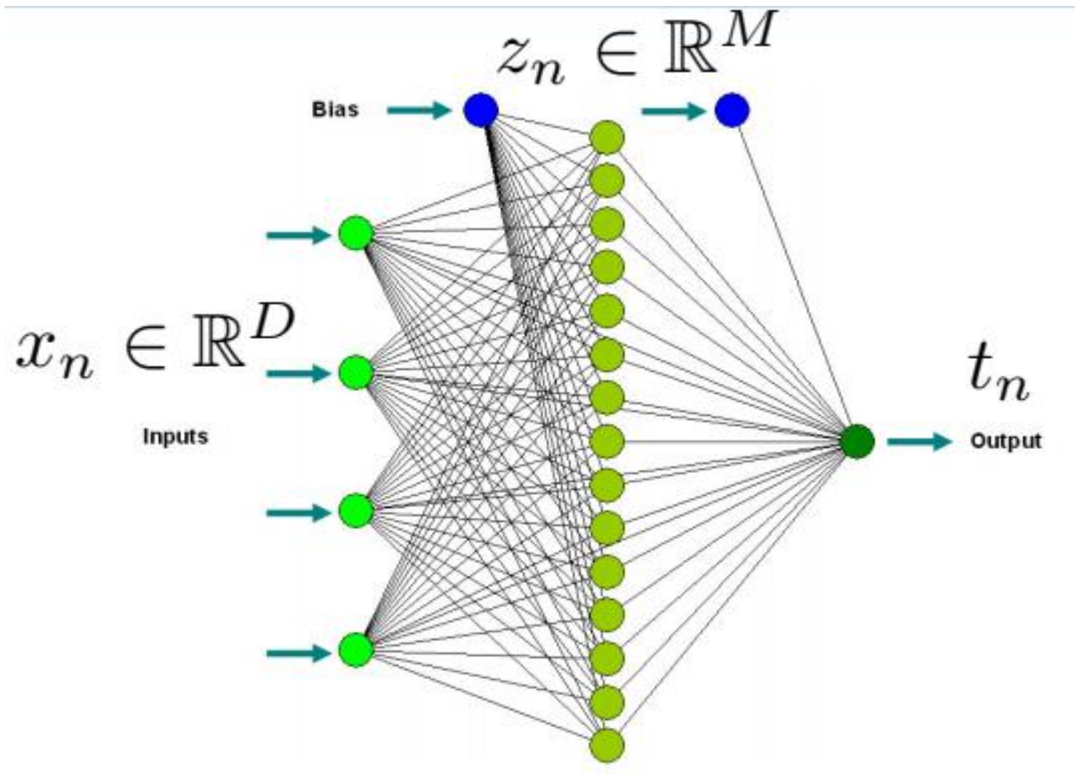
Multi-layer neural networks

- Having more perceptrons (neurons) in multiple layers
- First layer: input nodes
- Last layer: output nodes (one or more)
- Hidden layers: perceptrons
- Perceptrons of neighboring layers are connected
    - The connectedness depends on the architecture of the network (normally: fully connected)
- All perceptrons receive their inputs from the previous layers and propagate their outputs the to the next layer
- Weights correspond to the links (edges)
    - These weights should be adjusted by the learning procedure to minimize the cost function



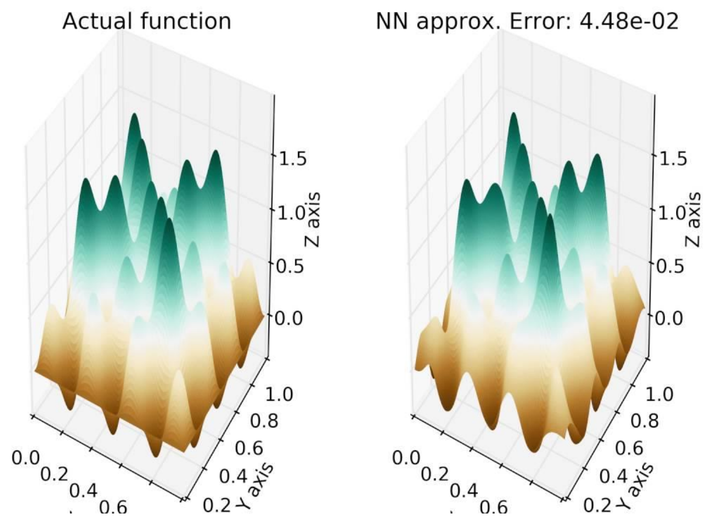
Feed-forward neural network

- Assume that the network is already trained (the weights are set)
- A record arrives to the input nodes and it propagates through the network in a forward direction (from the left to the right) on a layer-by-layer basis
    - The layers process the data
    - It can be considered as extracting new features in each layer
- Finally we obtain the result on the output node(s)



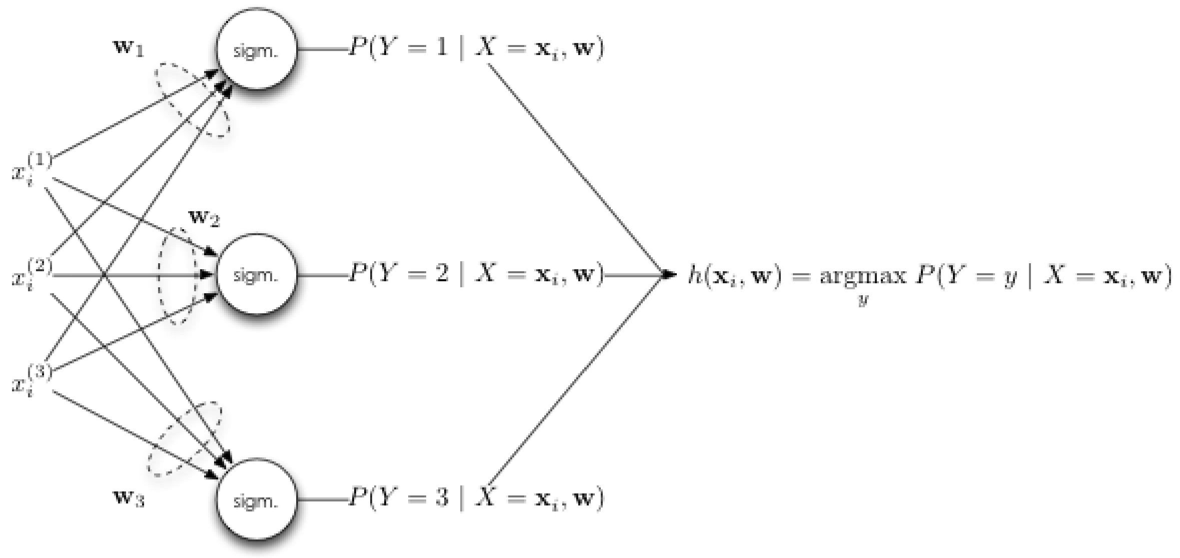
Universal approximation theorem

- A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions in compact subsets of R^n, under mild assumptions on the activation function
  - It is important to note that the theorem states that simple neural networks can represent a wide variety of functions, however, it does not touch upon the algorithmic learnability of the parameters
    - Non-constructive



Multi-class problem

- If there are more possible class labels (number of labels *c*)
- There are more output nodes: the same number as the number of possible labels (*c*)
- An input is assigned to label *i* if the *ith* output is 1 and the others are 0
  - if the possible output values are continuous then we assign the label with largest value on the corresponding output node
- E.g. for character recognition, the number of outputs corresponds to the number of characters



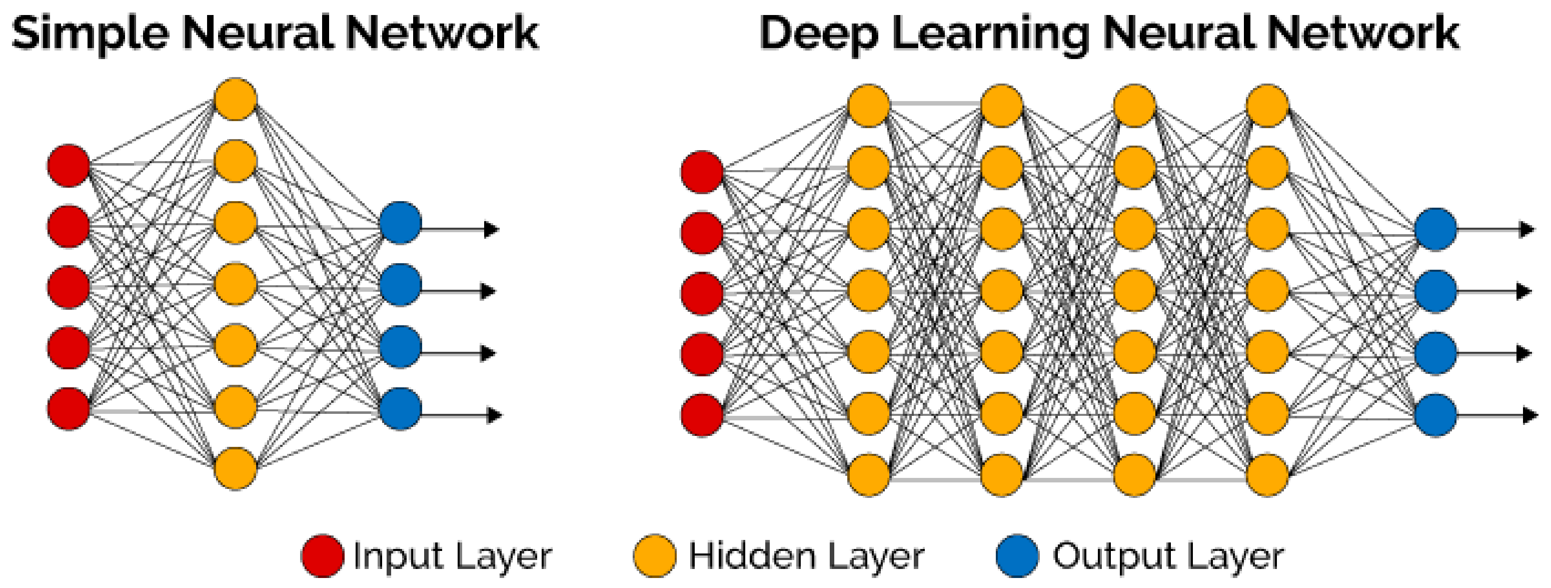
$\mathbf{w}_1$
$\mathbf{w}_2$
$\mathbf{w}_3$
$x_i^{(1)}$
$x_i^{(2)}$
$x_i^{(3)}$
sigm.
sigm.
sigm.
$P(Y=1 \mid X=\mathbf{x}_i,\mathbf{w})$
$P(Y=2 \mid X=\mathbf{x}_i,\mathbf{w})$
$P(Y=3 \mid X=\mathbf{x}_i,\mathbf{w})$
$h(\mathbf{x}_i,\mathbf{w}) = \underset{y}{\operatorname{argmax}}\, P(Y=y \mid X=\mathbf{x}_i,\mathbf{w})$

Architecture of the network

- Number of input nodes arises from the nature of the problem
  - Number of attributes, e.g. for character recognition it is the number of pixels
- Number of output nodes arises from the nature of the problem
- The other parameters can be set freely
  - Number of hidden layers
  - Number of neurons in each layer
- The more complex the network is, the more possible that the network will overfit the data
  - Sometimes it is enough to have a simpler network

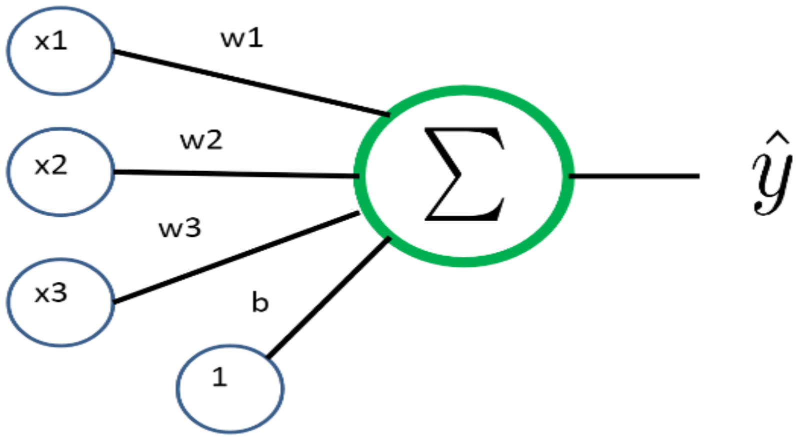
Deep learning

- Deep learning refers to machine learning with deep neural networks, i.e. neural networks with many hidden layers



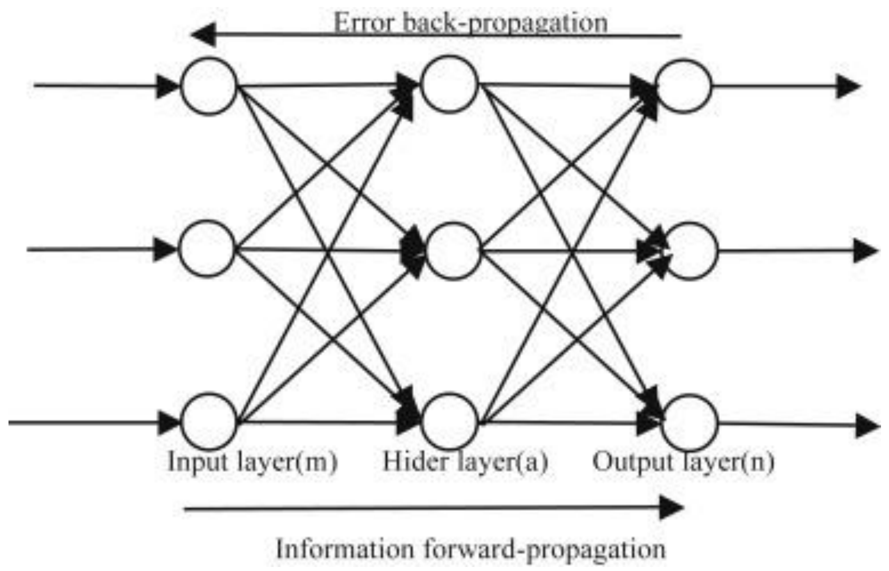
Training a perceptron

- Perceptron is a simple linear model, the training procedure is similar to the usual one:
  - We choose a cost function (MSE, logistic cost)
  - The cost function is minimized (gradient descent, stochastic gradient descent or other optimization method)



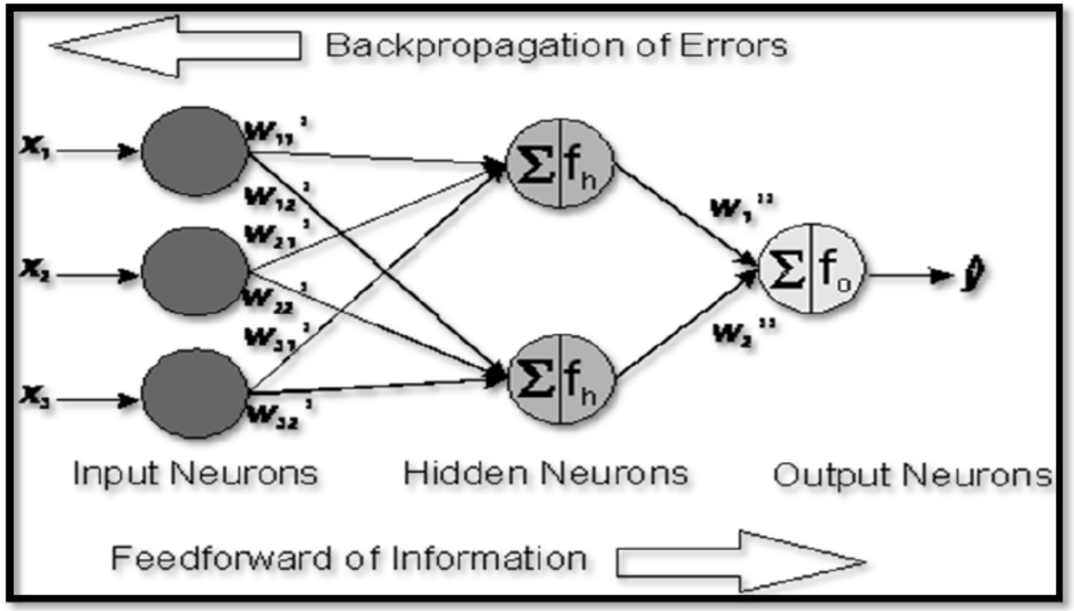
Forward propagation for ANNs

1. The training record arrives on the input nodes
2. Using the current weights on the edges, the record propagates through the network (from the left to the right)
3. We calculate the result on the output node(s) and we compare the output(s) with the actual target value
4. We calculate the cost (e.g. squared error, logarithmic error) based on the difference between the output and the actual target value



Backpropagation

1. Initializing the weights
2. Choosing a (random) record from the training data
3. Using the forward propagation algorithm, the output and the error is calculated
4. Calculating the gradient with respect to the weights
   - See next slides!
5. Updating the weights according to gradient method
6. Start the procedure again with another (random) record and repeat the procedure „until convergence”



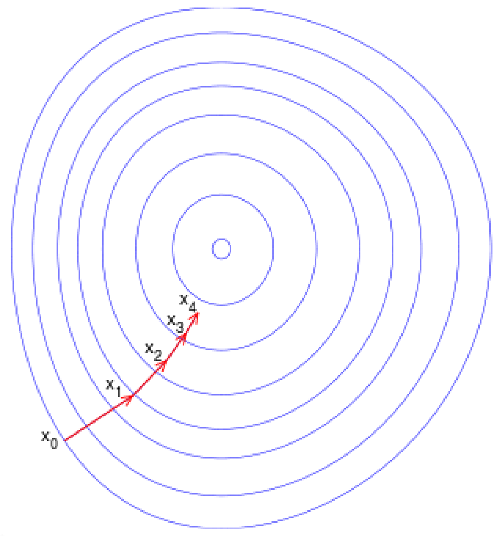
Calculating the gradient – using chain rule

- Reminder: how it worked for simple linear regression:

$$F = (\hat{y} - y)^2$$

$$\hat{y} = \sum_{j=1}^{m} w_j x_j + w_0$$

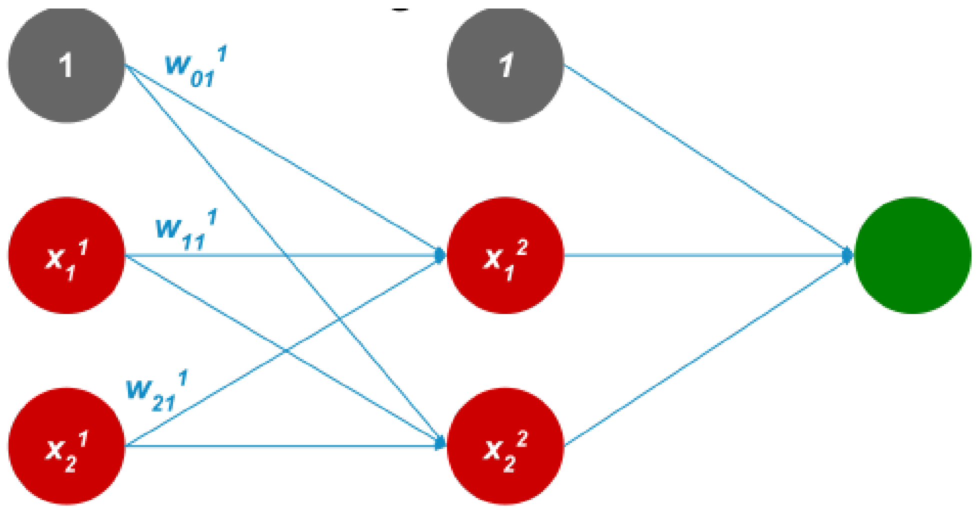
$$\frac{\partial F}{\partial w_j} = \frac{\partial F}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j}$$



Training a multi-layer neural network

- The output of a layer is the input of the following layer
- The weights on the edges should be learned by the model
- Activation function: *g*
- Weights on edges*: w*
- The output of the neurons (nodes) are denoted by *x* and calculated as:

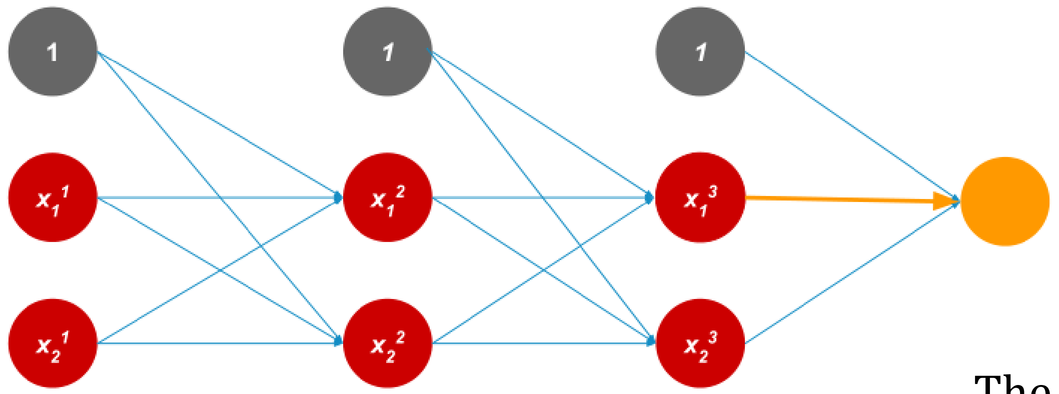
$$x_i^{l+1} = g\left(\sum_{j=1}^{n} w_{ji}^l x_j^l + w_0^l\right)$$



Calculating the gradient for a multi-layer neural netwok

- Using chain-rule:

$F = (\hat{y} - y)^2$



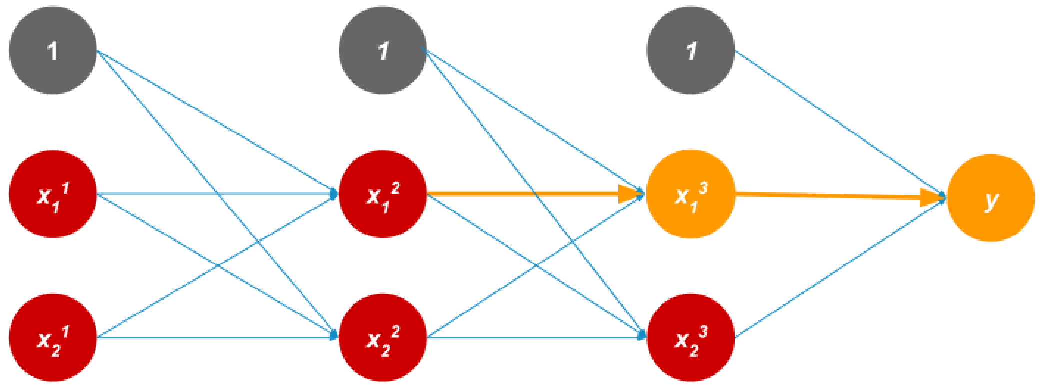
The output:
$\hat{y} = g(w_{01}^3 + w_{11}^3 x_1^3 + w_{21}^3 x_2^3)$

$$\frac{\partial F}{\partial w_{11}^3} = \frac{\partial F}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{11}^3} = 2 \cdot (\hat{y} - \mathrm{y}) \cdot g'(w_{01}^3 + w_{11}^3 x_1^3 + w_{21}^3 x_2^3) \cdot x_1^3$$

where g is the activation function, w_{jk}^l is the weight on the edge from x_j^l to x_k^{l+1}

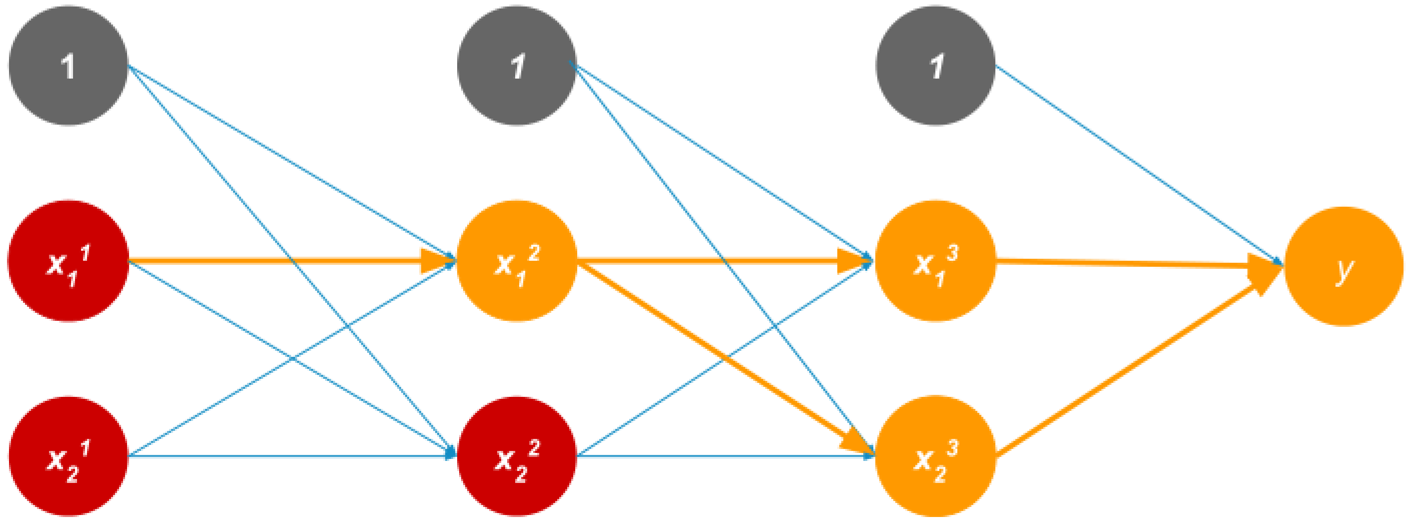
Calculating the gradient for a multi-layer neural network II.

- For „deeper” edges, i.e., for edges that do not effect the output directly, the derivatives calculated recursively



$$\frac{\partial F}{\partial w_{11}^2} = \frac{\partial F}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x_1^3} \frac{\partial x_1^3}{\partial w_{11}^2}$$

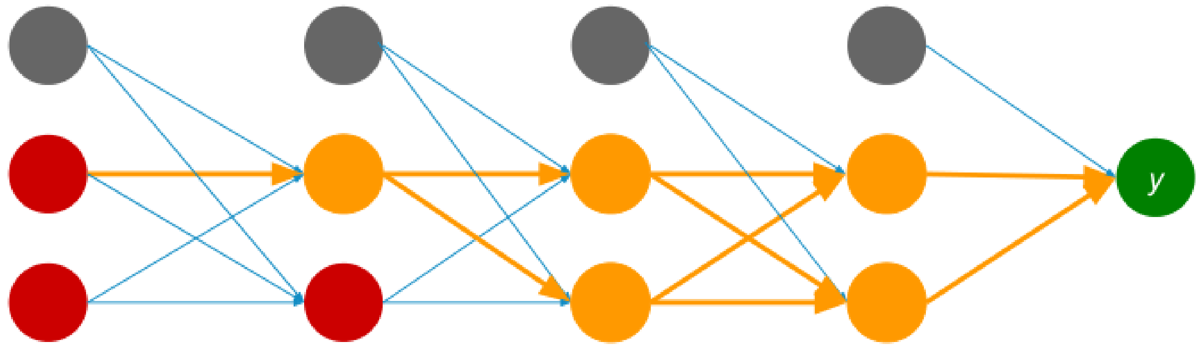
Calculating the gradient for a multi-layer neural network III.



$$\frac{\partial F}{\partial w_{11}^1} = \frac{\partial F}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x_1^3} \frac{\partial x_1^3}{\partial x_1^2} \frac{\partial x_1^2}{\partial w_{11}^1} + \frac{\partial F}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x_2^3} \frac{\partial x_2^3}{\partial x_1^2} \frac{\partial x_1^2}{\partial w_{11}^1}$$

Calculating the gradient for a multi-layer neural network IV.

- Generally, we have to sum up the derivatives for all the possible paths that starts from the end node of the examined edge and terminated in the output node



$$\frac{\partial F}{\partial w_{jk}^{l}} = \sum_{mn\ldots q} \frac{\partial F}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x_m^L} \frac{\partial x_m^L}{\partial x_n^{L-1}} \cdots \frac{\partial x_q^{l+2}}{\partial x_k^{l+1}} \frac{\partial x_k^{l+1}}{\partial w_{jk}^{l}}$$

- Where: L is the number of layers, $m, n, \ldots\ q$ run through the possible values (i.e. number the possible values are the number of neurons in each layer)

Problem

Consider a chain of two neurons. The input of the first neuron is x_1 and the output is $y_1 = ax_1 + b$. The input of the other neuron is x_2 and the output is $y_2 = cx_2 + d$ (the activation function is the identity function in both cases). Connect the two neurons so that the second neuron's input is y_1, i.e. $x_2 = y_1$.

a) Draw this neural network!

b) Give the final output y_2 as a function of x_1!

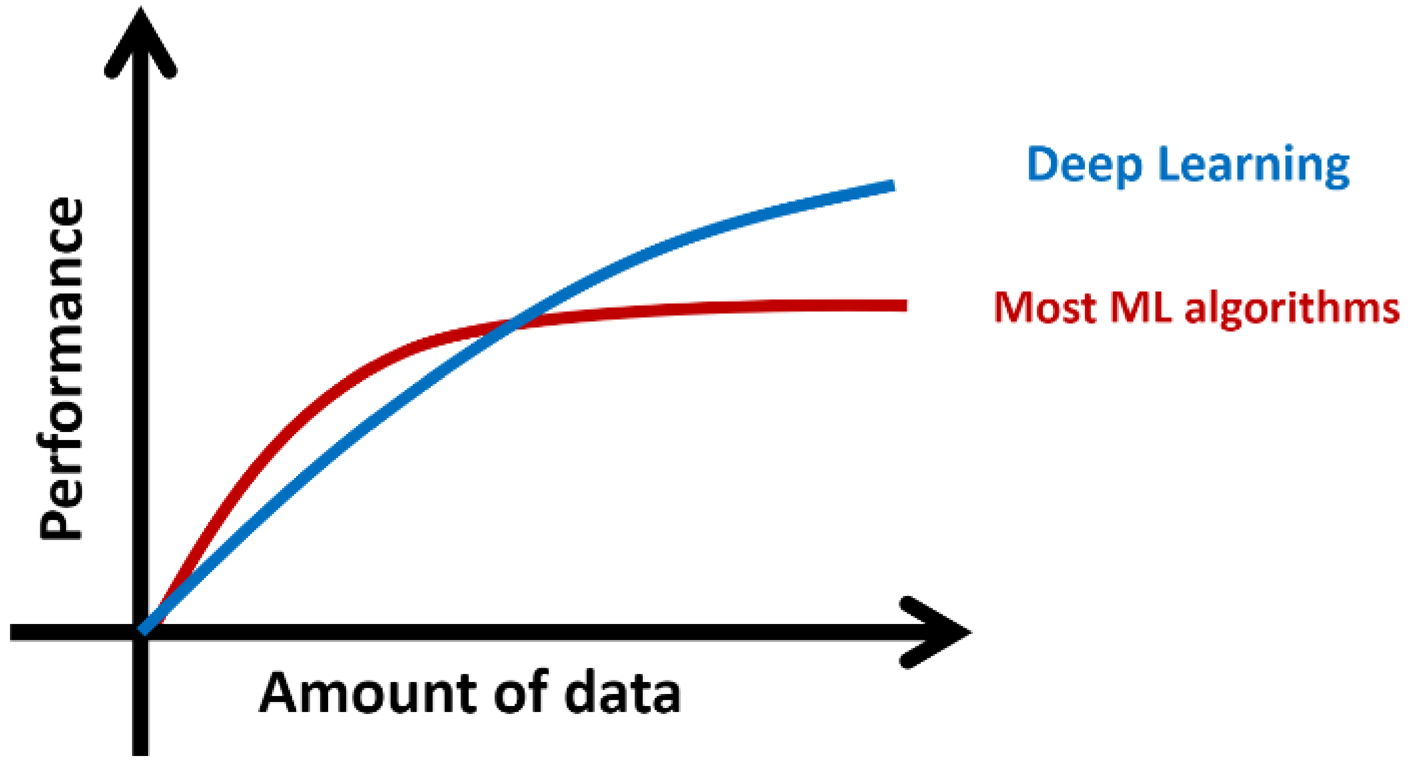
c) Let the input of the ANN be x and the output be y. Using the gradient descent method, show how the weights (a, b, c, d) are updated after one training step if the squared error is minimized.

Advantages/disadvantages of neural networks

- They make it possible to extract features automatically
- They mean significant advancement in:
  - Image recognition
  - Speech recognition
  - Text mining
- They can be parallelized
- Potential room for improvement
  - Increasing amount of data
  - Faster computers

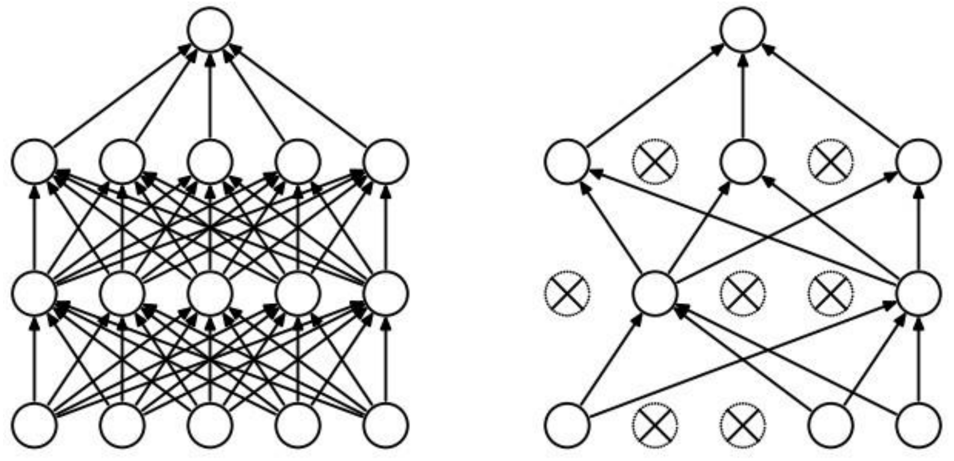
- For good performance (very) big data is necessary
- Computationally expensive
- It is difficult to set the (hyper)parameters, i.e. choose the proper architecture
  - Number of hidden layers, number of neurons, number of edges
- The result is not easily interpretable
  - Difficult to understand what the reason of the prediction is

Deep learning and amount of data



Avoiding overfitting neural networks

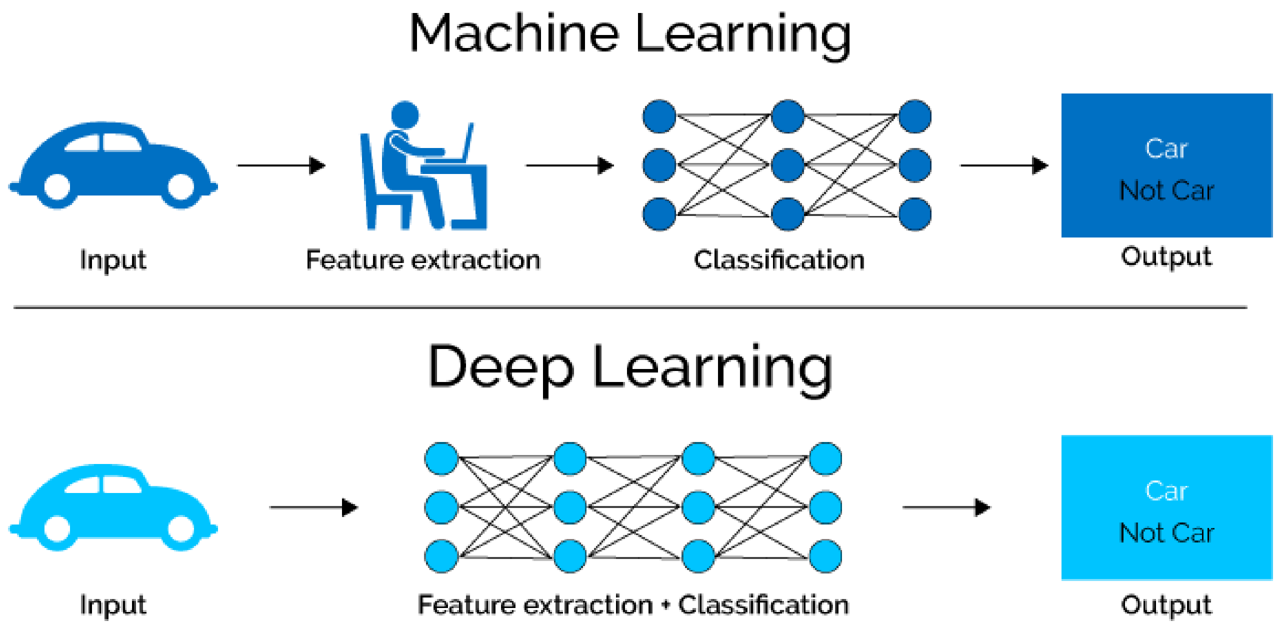
- Adding regularization term to cost function
  - Usually the sum of the squared weights (as before)
- Reducing the number of hidden layers
- Avoiding too complex architectures
- Dropout algorithm (dropping out certain nodes from the network)



(a) Standard Neural Net (b) After applying dropout.

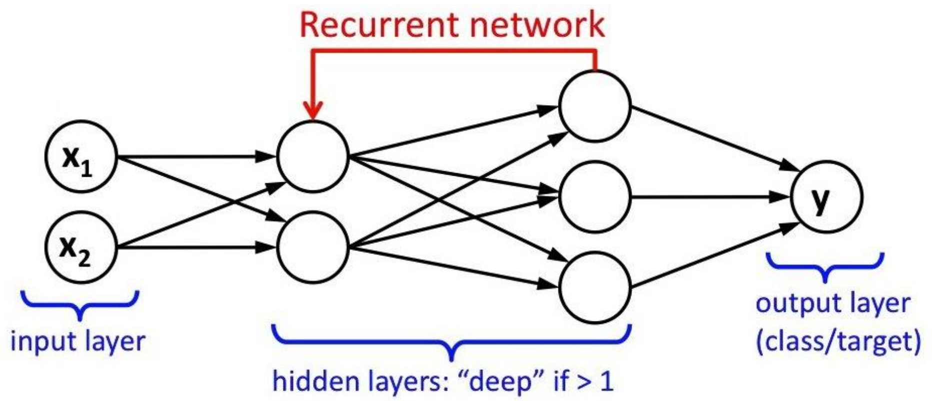
Neural network as a feature extractor

- We can think of the values on the hidden layers as new extracted attributes
- It is also possible to use these new features as inputs of another machine learning algorithm

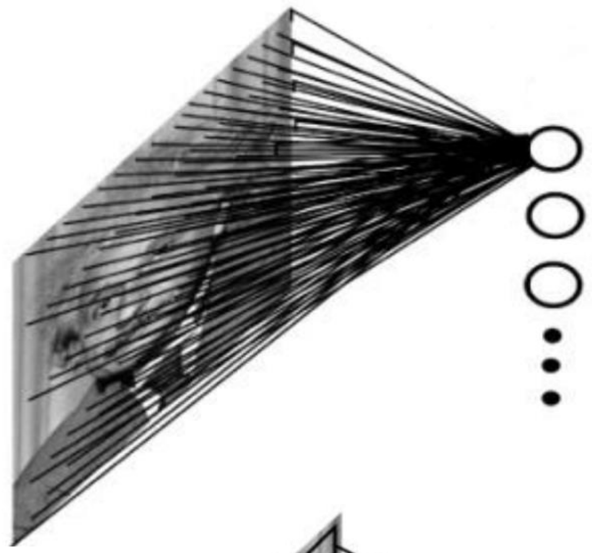


Types of neural networks

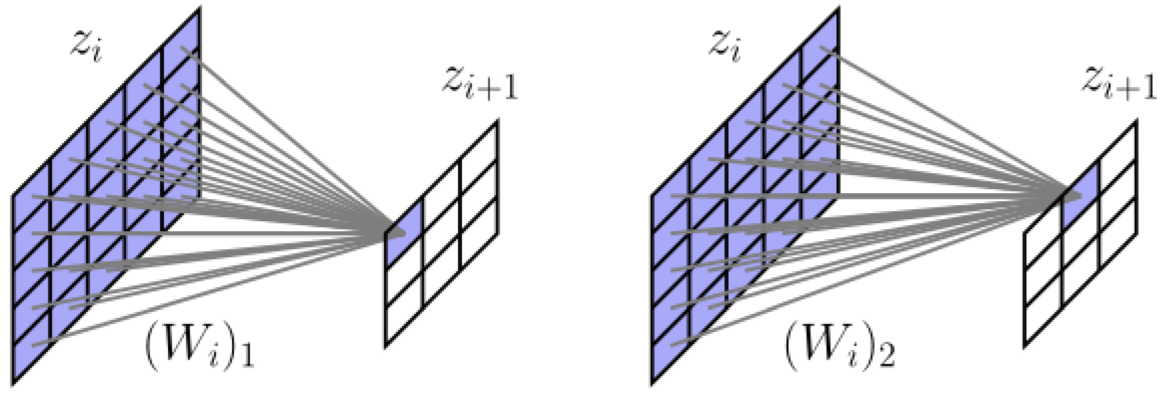
- Fully connected feed-forward neural networks
  - The simplest but not the most popular architecture
  - There are a lot of parameters (edge weights) that we have to optimize for
  - Danger of overfitting
- Recurrent Neural Network – RNN
  - For sequential data it is able to learn the temporal nature of the data
    - E.g. time series prediction, audio mining
  - There is a directed cycle in the network
  - RNN is able to use its own inner memory
- Convolutional Neural Networks – CNN
  - Not all of the connections are drawn, the „proximity”, of the data is also taken into consideration
  - Mainly used for image classification

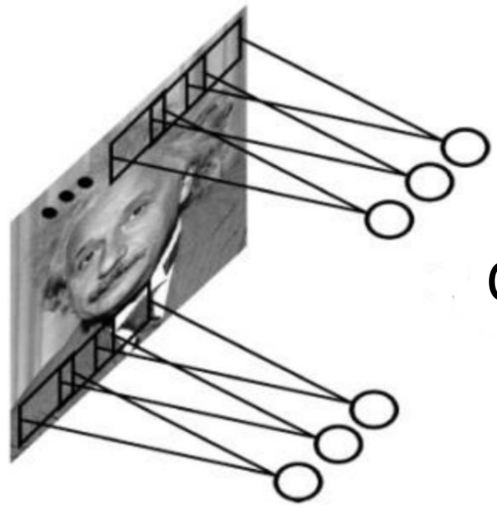


Fully connected vs. convolutional neural networks

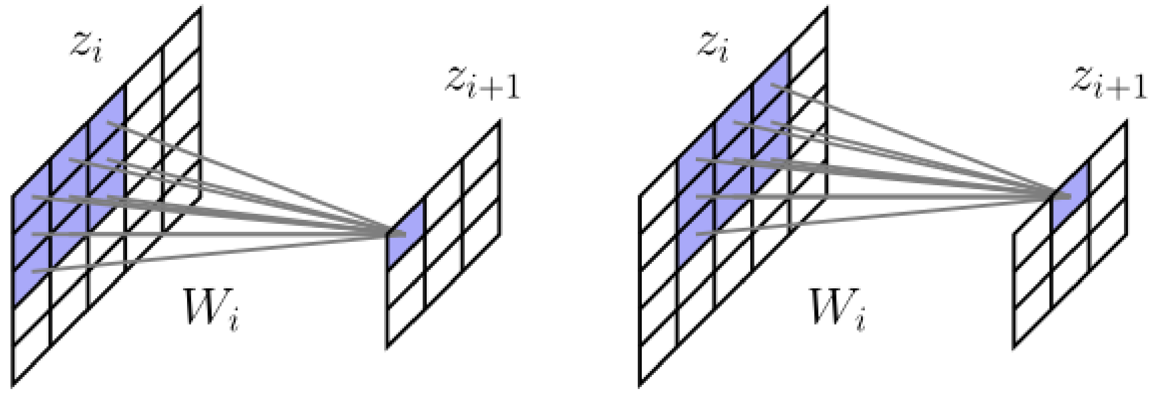


Fully connected

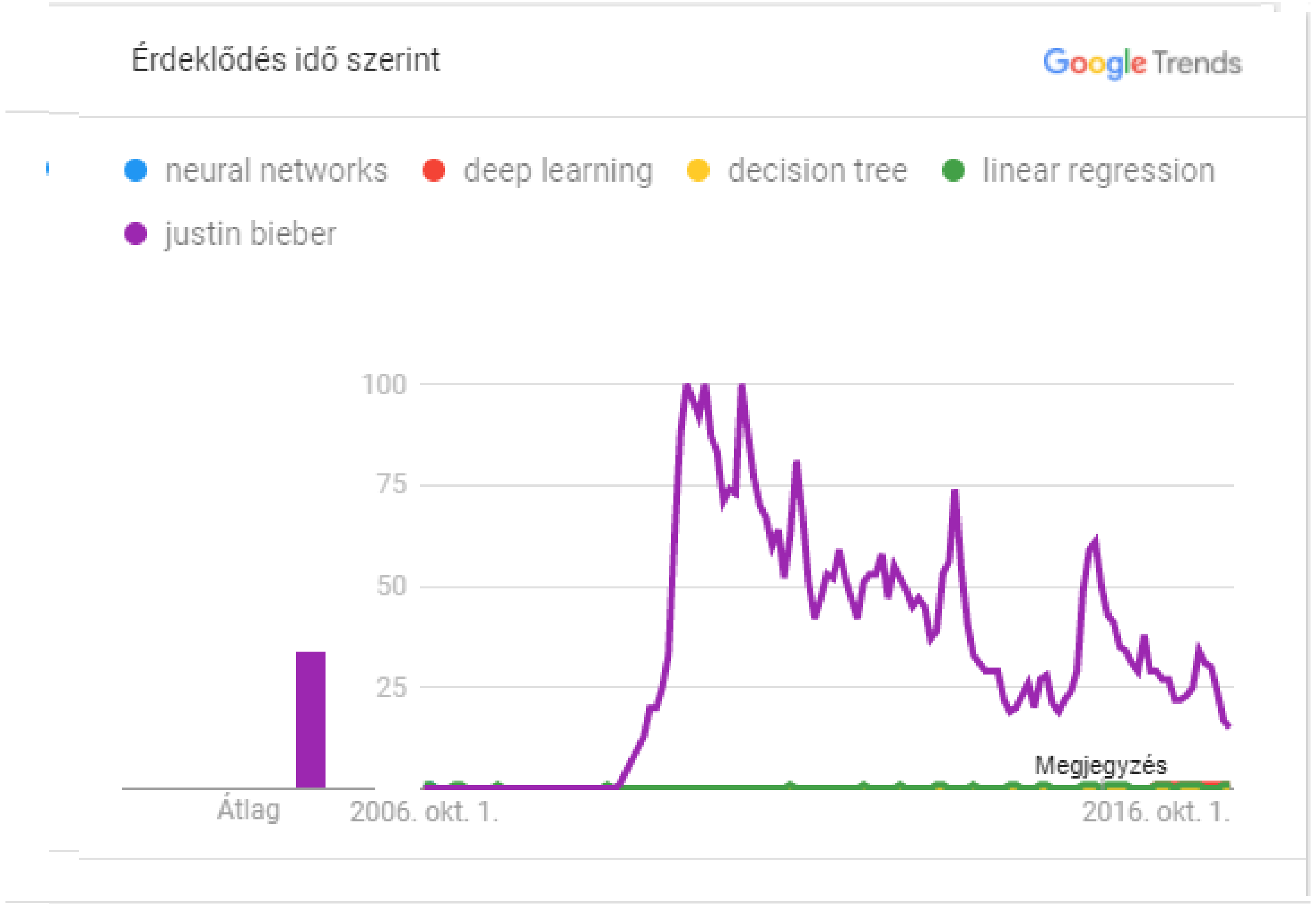




Convolutional



Change in search interest

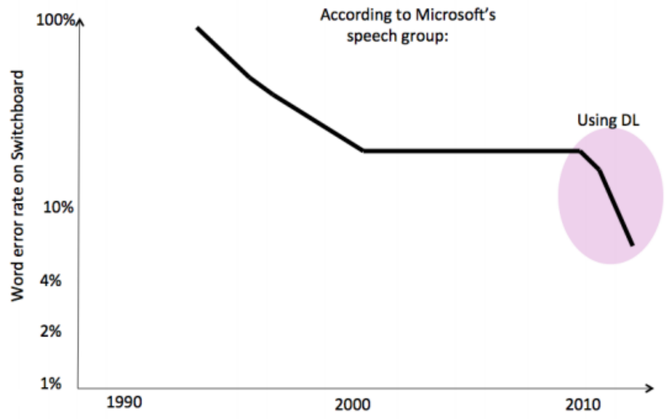


For which problems is it not so practical to use deep learning?

- If we have „traditional” attributes (features that can be easily interpreted by humans), then it is better to start with „traditional” algorithms
  - If we are not satisfied with the results, we can use deep learning to possibly achieve 1-2% performance improvement

For which problem is it practical to use deep learning?

- If the data have attributes that are not easily interpretable (e.g. text, image, audio files) then using deep learning has great potential
  - Image recognition
  - Text mining
  - Audio mining
  - Language processing
- It is practical to use an already trained network for the given problem instead of experimenting with the proper architecture



Try it out: http://deeplearning.cs.toronto.edu

Softwares for deep learning

- Keras
  - Based on Python, Theano and TensorFlow backend, easily readable code, originates from Google
- TensorFlow
- Working with Python (and with some other programming languages as well)

Acknowledgement

- András Benczúr, Róbert Pálovics, SZTAKI-AIT, DM1-2
- Krisztián Buza, MTA-BME, VISZJV68
- Bálint Daróczy, SZTAKI-BME, VISZAMA01
- Judit Csima, BME, VISZM185
- Gábor Horváth, Péter Antal, BME, VIMMD294, VIMIA313
- Lukács András, ELTE, MM1C1AB6E
- Tim Kraska, Brown University, CS195
- Dan Potter, Carsten Binnig, Eli Upfal, Brown University, CS1951A
- Erik Sudderth, Brown University, CS142
- Joe Blitzstein, Hanspeter Pfister, Verena Kaynig-Fittkau, Harvard University, CS109
- Rajan Patel, Stanford University, STAT202
- Andrew Ng, John Duchi, Stanford University, CS229