COMPUTATIONAL INTELLIGENCE
(INTRODUCTION TO MACHINE LEARNING) SS16

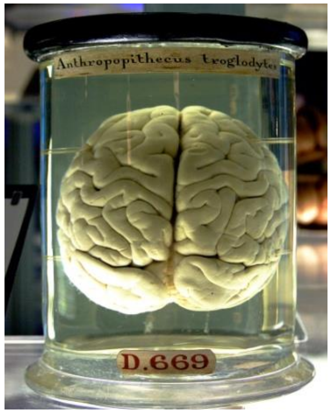
Lecture 4:

- Neural Networks
  - Perceptron
  - Feedforward Neural Network
  - Backpropagation algorithm

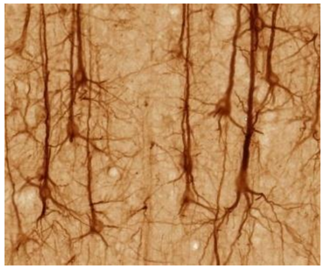
NEURAL NETWORKS MOTIVATION

The brain

- Enables us to do some remarkable things such as:
  - learning from experience
  - instant memory (experiences) search
  - face recognition (~100ms)
  - adaptation to a new situation
  - being creative..



- The brain is a biological neural network (BNN):
  - extremely complex and highly parallel system
  - consisting of 100 billions of neurons communicating via 100 trillions of synapses (each neuron connected to 10000 other neurons on average)

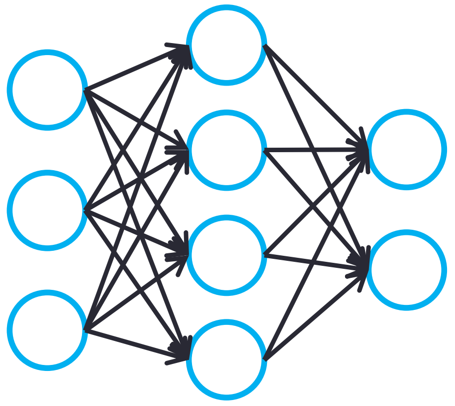


- Mimicking the brain could be the way to build an (human level) intelligent system!

Artificial Neural Networks (ANN)

- ANN are computational models that model the way brain works
- Motivated and inspired by BNN, but still only simple imitation of BNN!

- ANN mimic human learning by changing the strength of simulated neural connections on the basis of experience



- What are the different types of ANNs?
- What level of details is required?
- How do we implement and simulate them?

ARTIFICIAL NEURAL NETWORKS

History and types

History

- 1943 McCulloch and Pitts, Threshold Logic Unit (TLU) - artificial neuron model
- 1949 Hebb’s book “The organization of behavior", learning as synapse modification
- 1957 Rosenblatt, Perceptron (conceptual foundation of SVM)
- 1969 Minsky and Papert, limits of Perceptron (AI winter)
- 1974 Backpropagation algorithm (applied in the context of NN - “renaissance” of NN)
- 1982 Hopfield network (Recurrent Neural Network)
- 1985 Boltzmann machine (stochastic RNN)
- 2010+ Backpropagation algorithm is back (top results, GPUs, deep learning)

What types of ANN there are?

- Feedforward networks ⇦ this lecture
- SOM (Self Organizing Maps , Kohonen network)
- Recurrent networks
  - Hopfield Network
  - Boltzmann Machine
  - Echo State Network
  - Liquid State Machine
  - ...
- Spiking Neural Networks
- ...

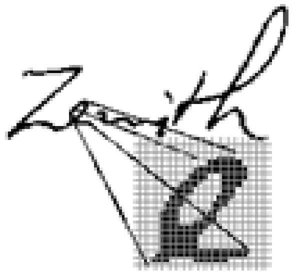
} Master courses NN, PBC

ARTIFICIAL NEURAL NETWORKS

Applications (level of details required?)

Machine learning ANN applications

- Areas of applications:
  - Function approximation, regression analysis
  - Classification
  - Data processing
  - Robotics

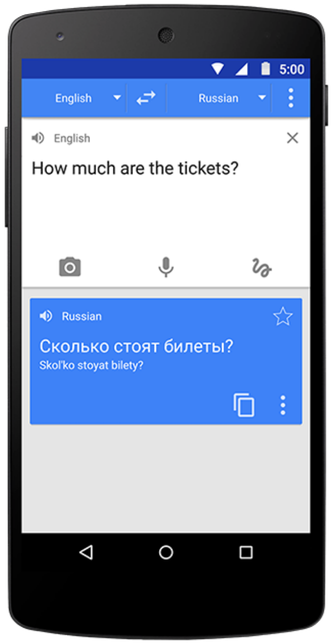


- Examples:
  - Vehicle control, process control
  - Game-playing, decision making
  - Pattern recognition (face identification, object recognition)
  - Sequence recognition (gesture, speech, handwritten text recognition)
  - Medical diagnosis
  - Financial applications (e.g. automated trading systems)
  - Data mining, spam filtering
  - Data compression

Success of Neural Networks

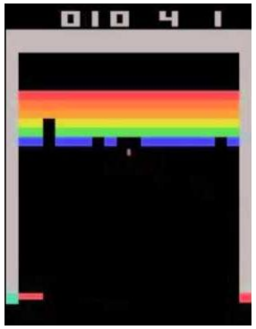
- Since 2012
  - Alpha Go
  - ATARI Games
  - Speech synthesis
  - Google translate

- Since 2005, overcame all other methods in :
  - Image recognition Image Net
  - Speech analysis
  - Text prediction
  - Analysis of brain slices



English
How much are the tickets?

Russian
Сколько стоят билеты?
Skol'ko stoyat bilety?



Other fields using NN

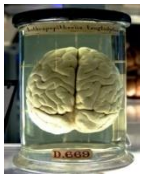
(and often biologically more realistic neuron models)

- Computational neuroscience:
  - Create models of biological neural systems in order to understand how biological systems work
- Cognitive science:
  - Modelling higher(language, problem solving) and lower(vision, speech generation) level reasoning
- Neurobiology:
  - Modelling how the brain works at neuron-level (neural correlates of consciousness) or higher levels(vision, hearing)
- Philosophy:
  - Can human mind and behavior be explained in terms of symbols, or does it require something lower level, like a neuronal based model?

ARTIFICIAL NEURAL NETWORKS

Simulations

Brain vs computer



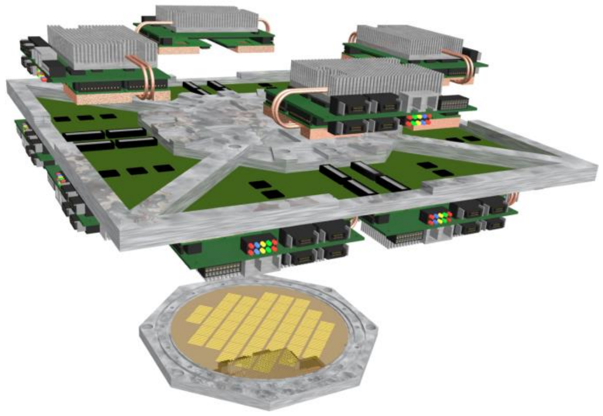
- Hundreds of neuron types
- ~100 billion neurons
- ~100 trillion synapses
- ~ms signal transmission
- Serial, parallel
- Analog
- Robustness
- Energy efficient
- Learns and develops through experience

- 1-2 types of processing unit
- ~10 processors
- ~1 billion transistors
- ~ns signal transmission
- Serial (per core)
- Digital
- Sensitive to failures

How to efficiently simulate ANN?

- Depends on the network size and the level of biological details (neuron/synapse)

- Implementation on:
  - General purpose computers (exploiting GPUs)
  - Supercomputers (Blue Gene, IBM)



- Specialized hardware (analog, digital, hybrid)
  - BSS (Wafer scale)
  - SpiNNaker
  - IBM TrueNorth
  - FPGAs
  - Other neuromorphic hardware

Recent large scale simulations

- 2005+ Blue Brain Project (BBP): Henry Markram at EPFL
  - An attempt to create a synthetic brain by reverse-engineering the mammalian brain down to the molecular level
  - Simulation of rat neocortical column (10,000 neurons and 100 million synapses)
- 2009 D.S. Modha (IBM): simulation of cat cerebral cortex
  - 1 billion neurons, 10 trillion synapses, 100x slower than real time at 147,456 processor Blue Gene supercomputer (cognitive architecture)
- 2012 D.S. Modha (IBM):
  - 530 billion neurons, 137 trillion synapses,1542x slower than real time at 1,572,864 processors of Sequoia supercomputer
- 2012 Chris Eliasmith: Spaun
  - A Large-Scale Model of the Functioning Brain
  - it can perform 8 tasks, 2 million neurons, with plasticity

Recent large scale simulations (2)

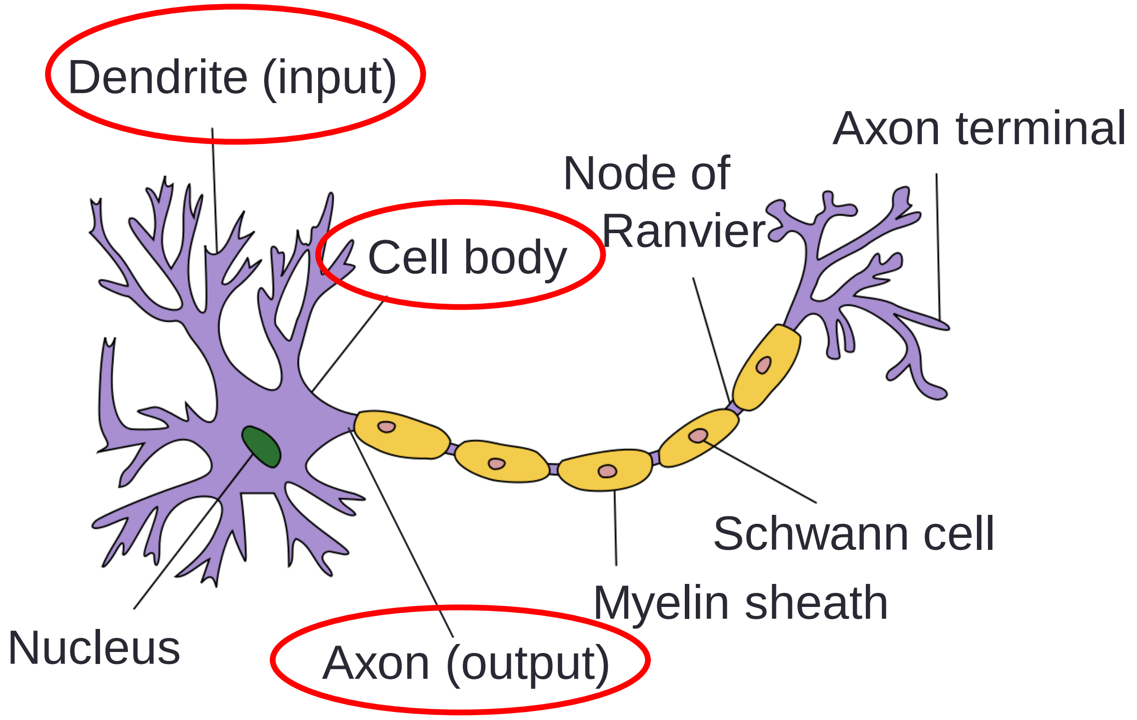
- 2012 Andrew Ng (Google) - deep learning:
  - extract concepts (cat) from YouTube videos
  - 1 million neurons, 1 billion synapses, 16,000 processors
  - “One algorithm hypothesis”, learning

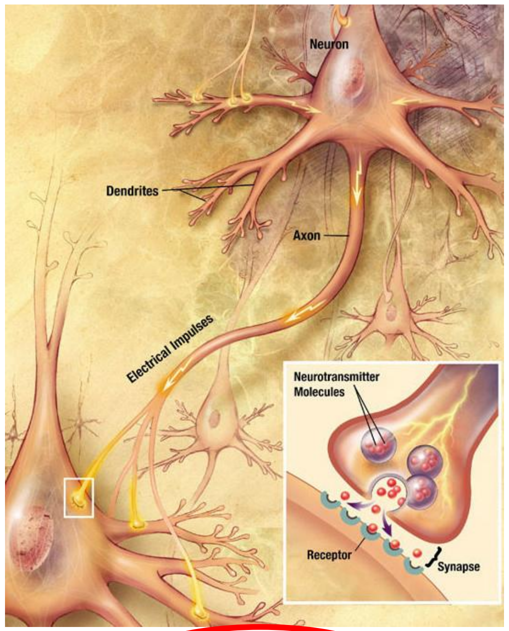
- 2013 Andrew Ng (Google) - deep learning:
  - network size x10 simulated with only 64 GPUs

- 2013+ Human Brain Project (HBP):
  - Aim is to simulate the complete human brain (molecular level) on supercomputers to better understand how it functions
  - www.ted.com/talks/henry_markram_supercomputing_the_brain_s_secrets

ARTIFICIAL NEURAL MODEL

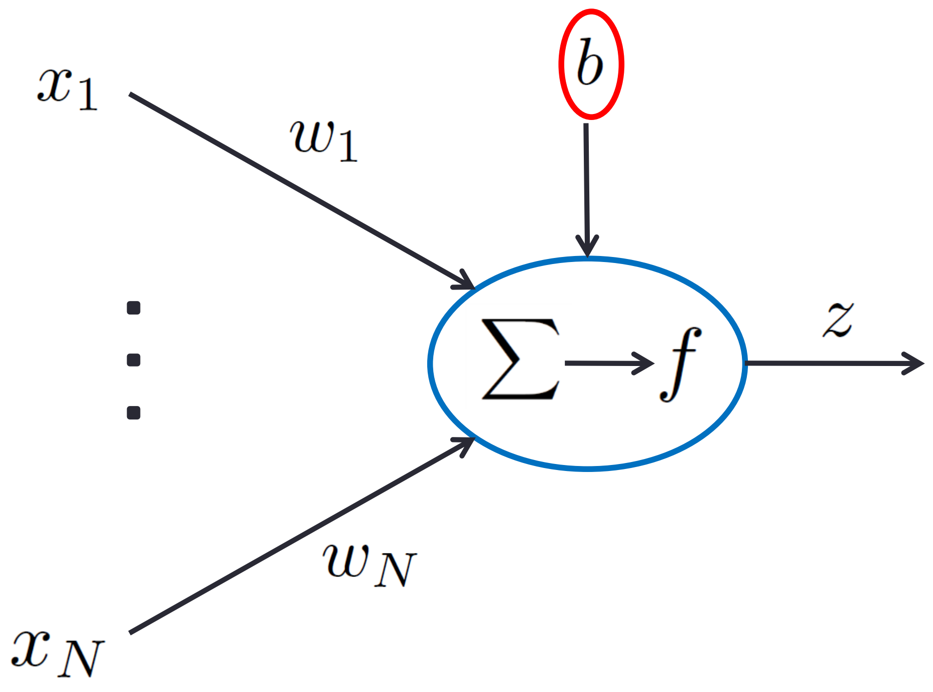
Biological neuron





Synapse

Artificial neuron model



Activation:

$$a = b + \sum_{i}^{N} x_i w_i$$

Output:

$$z = f(a)$$

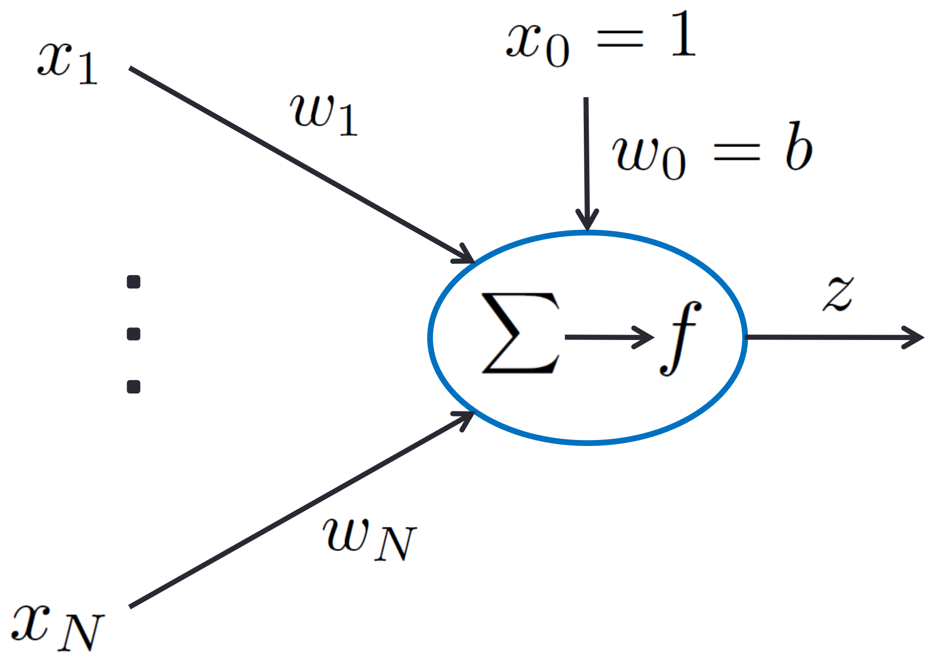
$x_1 \ldots x_N$ - inputs

$w_1 \ldots w_N$ - weights

b - bias (threshold)

f - activation function

Artificial neuron model



$x_1 \dots x_N$ - inputs

$w_1 \dots w_N$ - weights

b - bias (threshold)

f - activation function

Activation:

$$a = \sum_{i=0}^{N} x_i w_i$$

$$a = \boldsymbol{w}^T \boldsymbol{x}$$

Output:

$$z = f(a)$$

$$z = f(\boldsymbol{w}^T \boldsymbol{x})$$

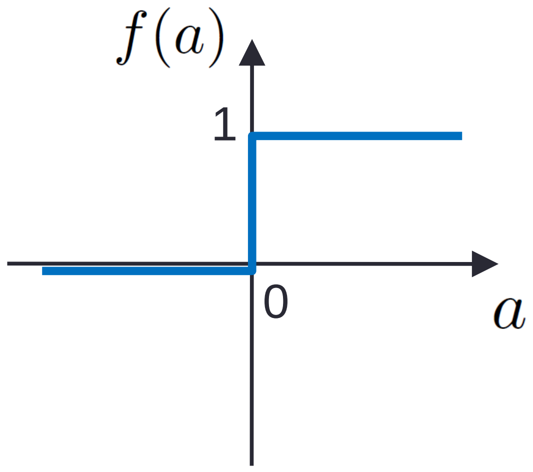
$$\boldsymbol{w} = \underbrace{\begin{pmatrix} b \\ w_1 \\ \vdots \\ w_N \end{pmatrix}}_{\in \mathbb{R}^{N+1}} \quad \boldsymbol{x} = \underbrace{\begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_N \end{pmatrix}}_{\in \mathbb{R}^{N+1}}$$

Activation functions (most common)

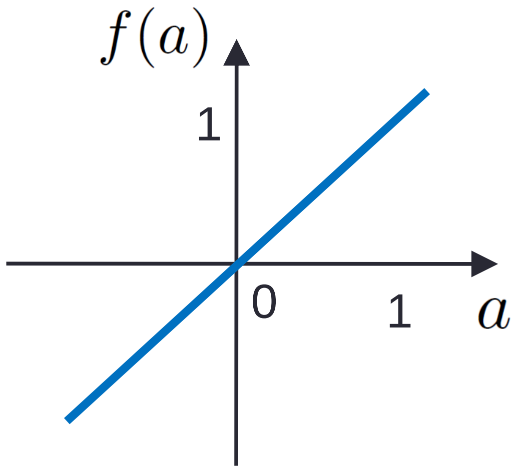
$$f(a) = \begin{cases} 0 & a < 0 \\ 1 & a \geq 0 \end{cases}$$

$$f(a) = a$$

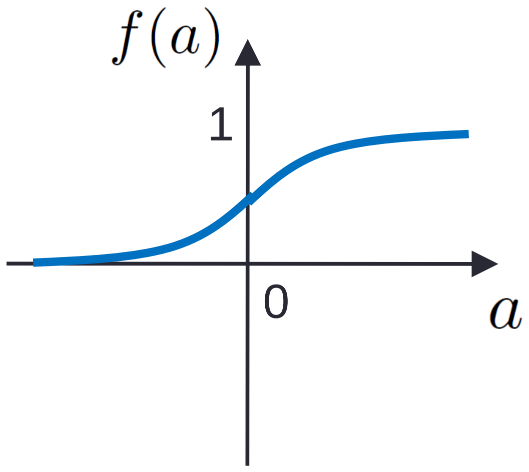
$$f(a) = \frac{1}{1 + e^{-a}}$$



Heaviside step function
Threshold Logic Unit
(TLU) Perceptron
(classification)



Linear function
ADALINE
(linear regression)

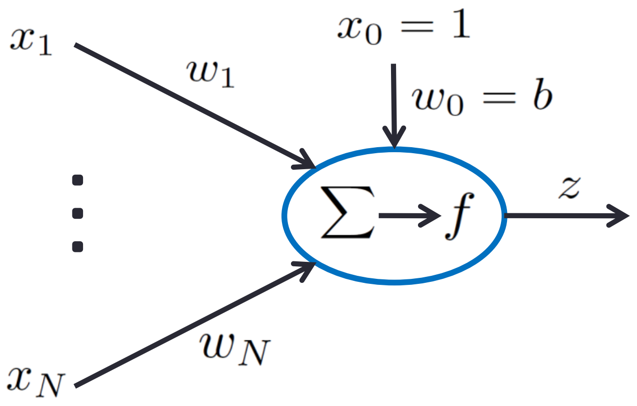


Sigmoid function
Feedforward networks
(nonlinear, classification)

PERCEPTRON

Introduction

- Perceptron is the simplest neural network for classification of linearly separable data – it is a linear (binary) classifier
- It consists of 1 neuron with Heaviside step function as activation function (TLU)



$$a = \boldsymbol{w}^T \boldsymbol{x}$$
$$z = f(a)$$

$$f(a) = \begin{cases} 0 & a < 0 \\ 1 & a \geq 0 \end{cases}$$

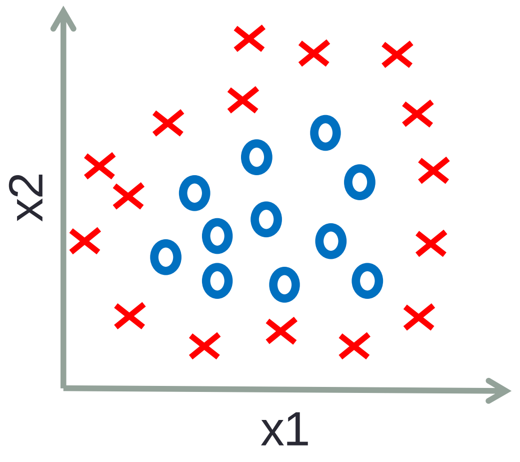
- Learning algorithm by Rosenblatt in 1957:
  - **Theorem**: If training set is linearly separable the algorithm converges
  - If training set is not linearly separable, the algorithm does not terminate!
  - Algorithm allows for online learning: it processes one data sample at a time

Linearly separable data?

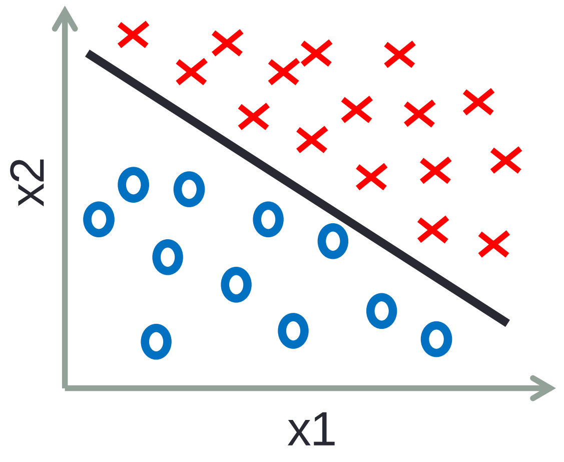
- Consider a training set consisting of *m* samples: $\langle \boldsymbol{x}^{(1)}, y^{(1)} \rangle \ldots \langle \boldsymbol{x}^{(m)}, y^{(m)} \rangle$

Where $y^{(i)} = \begin{cases} 0 & \text{if} \quad \boldsymbol{x}^{(i)} \quad \text{belongs to the class 0} \quad \text{o} \\ 1 & \text{if} \quad \boldsymbol{x}^{(i)} \quad \text{belongs to the class 1} \quad \times \end{cases}$

Example in 2D space: Can you find a linear decision boundary (hyperplane) which separates class 0 from class 1?



NO



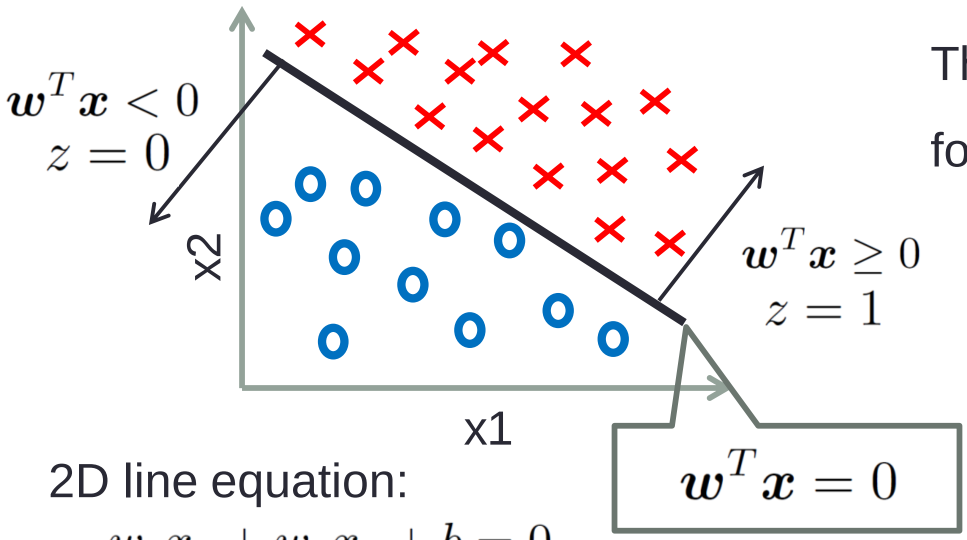
YES

Binary classification

If training data is linearly separable then there exist parameters $\boldsymbol{w}$ defining a decision boundary or a hyperplane (a subspace of one dimension less then its ambient space) such that:

$a = \boldsymbol{w}^T \boldsymbol{x}^{(i)} < 0$ for each $\boldsymbol{x}^{(i)}$ that belongs to class 0 o

$a = \boldsymbol{w}^T \boldsymbol{x}^{(i)} \geq 0$ for each $\boldsymbol{x}^{(i)}$ that belongs to class 1 ×



$\boldsymbol{w}^T \boldsymbol{x} < 0$
$z = 0$
x2
x1
$\boldsymbol{w}^T \boldsymbol{x} \geq 0$
$z = 1$
$\boldsymbol{w}^T \boldsymbol{x} = 0$

2D line equation:

$w_1 x_1 + w_2 x_2 + b = 0$

The output of Perceptron $z = f(\boldsymbol{w}^T \boldsymbol{x})$

for sample $\boldsymbol{x}^{(i)}$ is:

$z = 0$ if $\boldsymbol{x}^{(i)}$ belongs to class 0 o

$z = 1$ if $\boldsymbol{x}^{(i)}$ belongs to class 1 ×

$$f(a) = \begin{cases} 0 & a < 0 \\ 1 & a \geq 0 \end{cases}$$

Learning algorithm

- For each sample $\boldsymbol{x}^{(i)}$ from training set:
  - If sample is classified correctly no change

    $z = 0$ as $\boldsymbol{w}^T\boldsymbol{x}^{(i)} < 0$ and $y^{(i)} = 0$ or

    $z = 1$ as $\boldsymbol{w}^T\boldsymbol{x}^{(i)} \geq 0$ and $y^{(i)} = 1$
  - Otherwise update the weights

    $$\boldsymbol{w} := \boldsymbol{w} + \eta(y^{(i)} - z)\boldsymbol{x}^{(i)}$$

    sample category 0 or 1

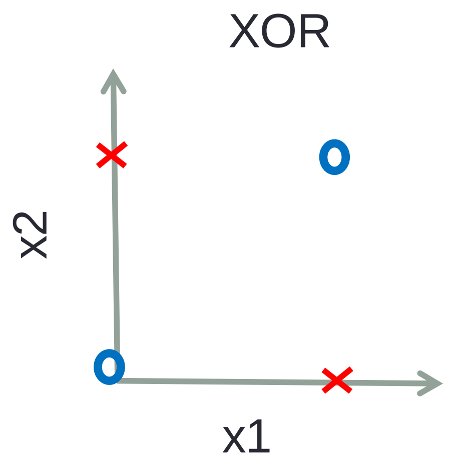
    Perceptron prediction 0 or 1

    η is learning rate

- Repeat until all data is correctly classified or max number of iteration reached (or some other stopping criteria is met, e.g. the number of misclassified samples)

Limitations and extensions

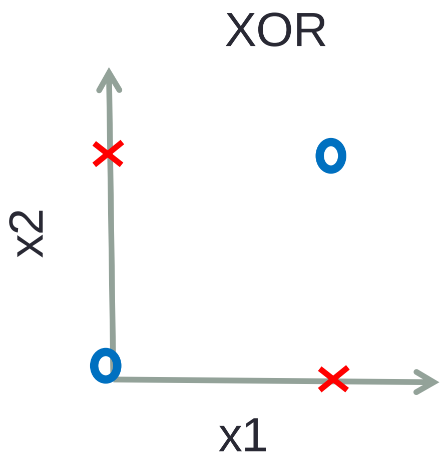
- If data is not linearly separable no linear classifier can classify all data correctly, so neither can the Perceptron
- Note that the best classifier is not necessarily one which classifies all the training data perfectly (overfitting)
- But for Perceptron, if data is not linearly separable it does not converge!



Limitations and extensions

- This was brought up by Minsky and Pepert in 1969 (AI winter)
  - They pointed out that Perceptron can not solve XOR problem

x	y	out
0	0	0
1	1	0
0	1	1
1	0	1

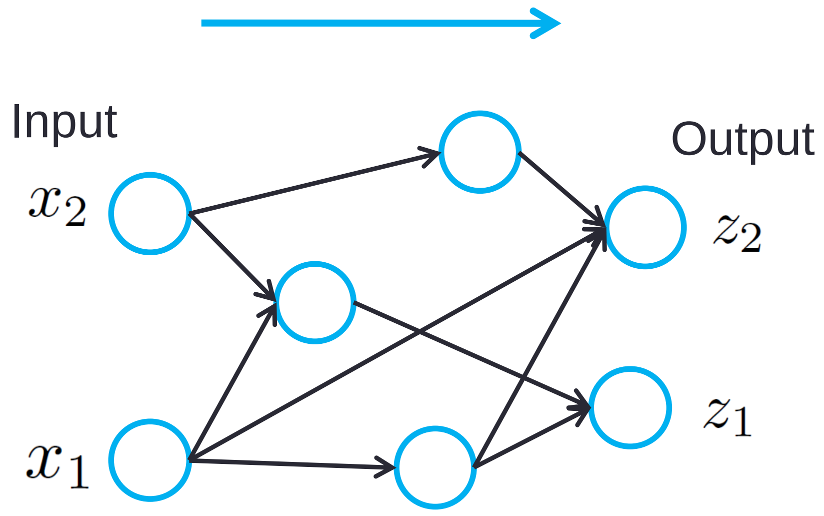


- Extensions:
  - Pocket Perceptron
  - Perceptron of optimal stability
  - Multiclass Perceptron
  - Kernel Perceptron (separate nonlinear data!)

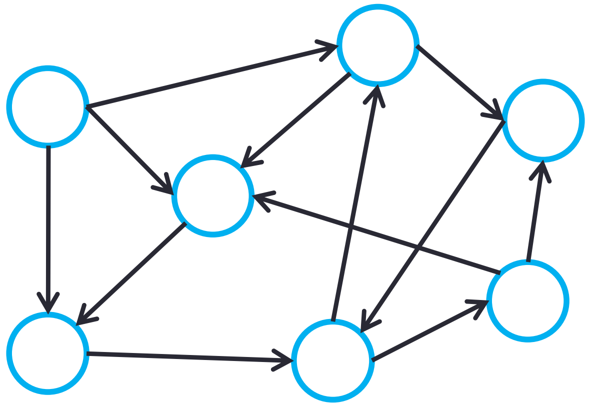
FEEDFORWARD NEURAL NETWORK

Feedforward architecture

- Network with feedforward architecture has no cycles
- The input information is propagated from the input neurons towards the output ones



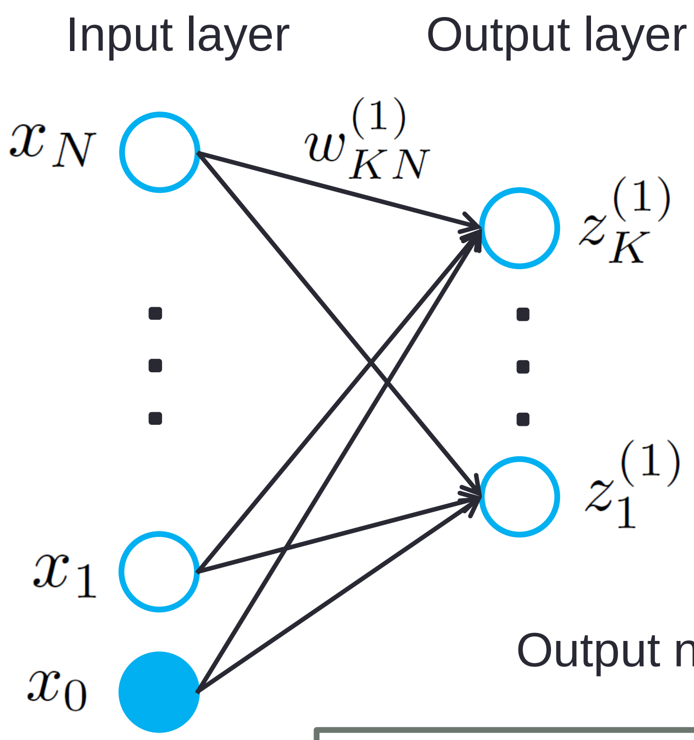
Feedforward architecture



Recurrent architecture

Feedforward layer architecture

- Neurons are organized in layers
- Such networks are also called Multilayer Perceptrons



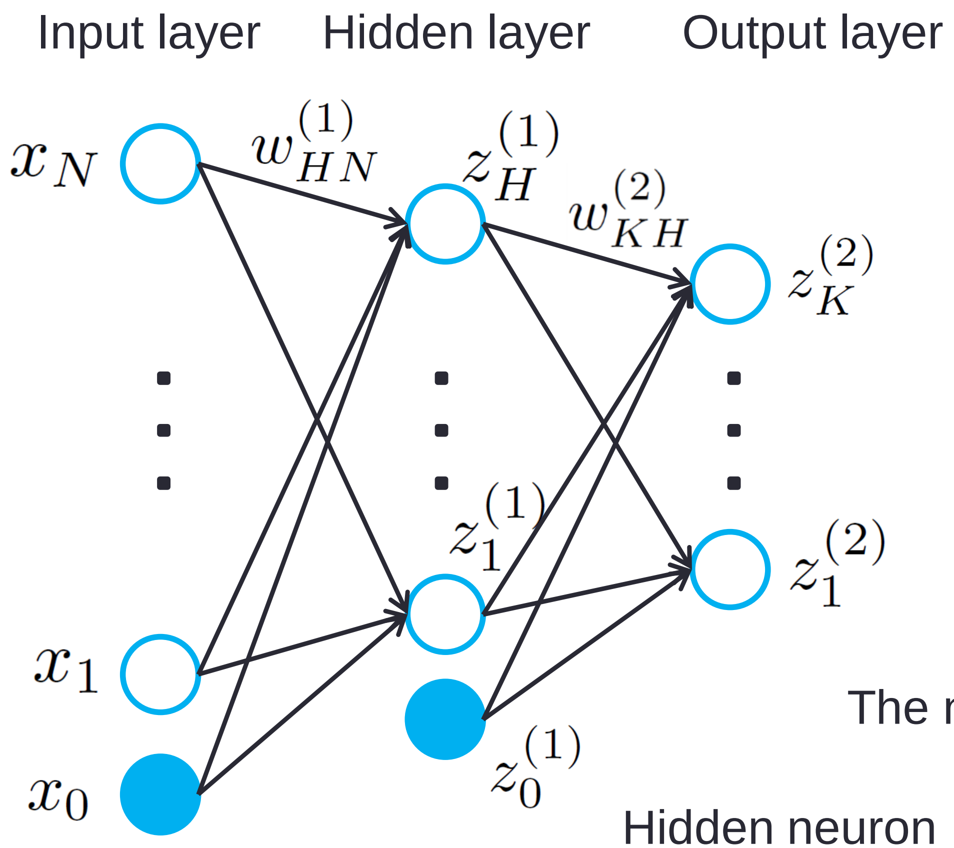
input	$\boldsymbol{x} \in \mathbb{R}^{N+1}$
output	$\boldsymbol{z}^{(1)} \in \mathbb{R}^{K}$
weights	$\boldsymbol{w}^{(1)} \in \mathbb{R}^{K \times N}$

The network implements the function:

Output neuron $$z^{(1)}_k = f^{(1)}_k\left(\sum_{l=0}^{N} w^{(1)}_{kl} x_l\right)$$

Activation function of neuron ***k*** in layer 1

Hidden layers



input	$\boldsymbol{x} \in \mathbb{R}^{N+1}$
output	$\boldsymbol{z}^{(2)} \in \mathbb{R}^{K}$
weights	$\boldsymbol{w}^{(1)} \in \mathbb{R}^{H \times N}$
	$\boldsymbol{w}^{(2)} \in \mathbb{R}^{K \times H}$

The network implements the function:

Hidden neuron $$\boxed{z^{(1)}_j} = f^{(1)}_j\Big(\sum_{l=0}^{N} w^{(1)}_{jl} x_l\Big)$$

Output neuron $$z^{(2)}_k = f^{(2)}_k\Big(\sum_{j=0}^{H} w^{(2)}_{kj} z^{(1)}_j\Big) = f^{(2)}_k\Big(\sum_{j=0}^{H} w^{(2)}_{kj} f^{(1)}_j\Big(\sum_{l=0}^{N} w^{(1)}_{jl} x_l\Big)\Big)$$

Hidden neurons (units)

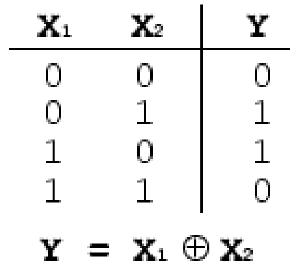
- Are situated in **hidden layers** between the input and the output layers
- They allow a network to learn **non-linear** functions and to represent combinations of the input features
- Given too many hidden neurons, a neural net will simply memorize the input patterns (overfitting)
- Given too few hidden neurons, the network may not be able to represent all of the necessary generalizations (underfitting)

FEEDFORWARD NEURAL NETWORK

Computational power

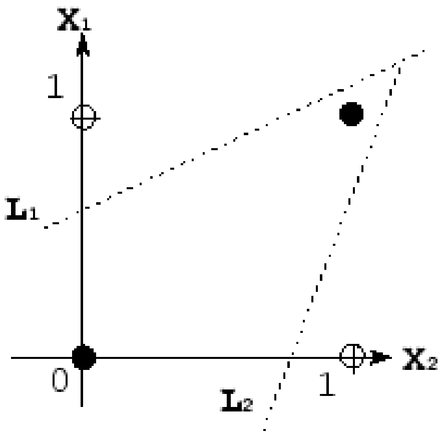
XOR problem

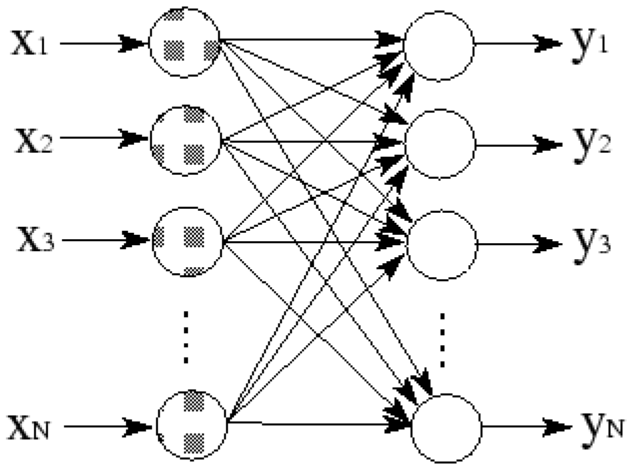
- ANN which solves XOR problem can be easily constructed



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

$Y = X_1 \oplus X_2$



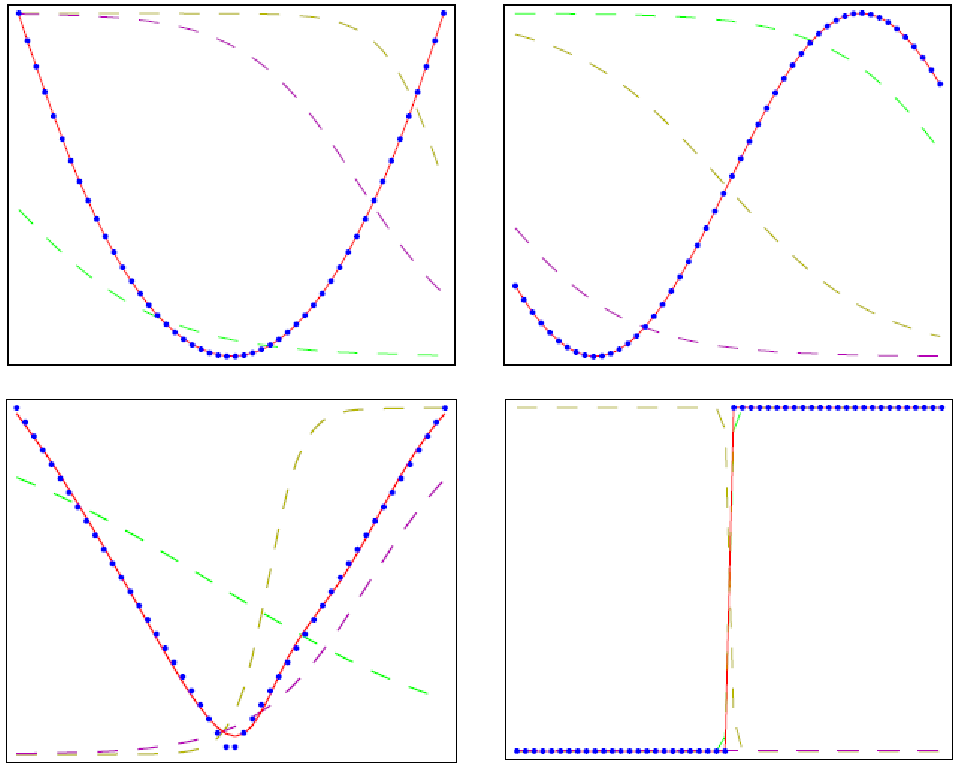


Computational power

- **Boolean functions:**
  - Every Boolean function can be represented by network with single hidden layer, but might require exponential (in number of inputs) hidden units

- **Continuous functions (universal approximation theorem)**:
  - Every bounded continuous function can be approximated with arbitrarily small error, by a network with one hidden layer under mild assumption on activation function (it has to be non-constant, bounded, and monotonically-increasing continuous function, e.g. sigmoid function) [Cybenko1989; Hornik et al. 1989]

Example: function approximation

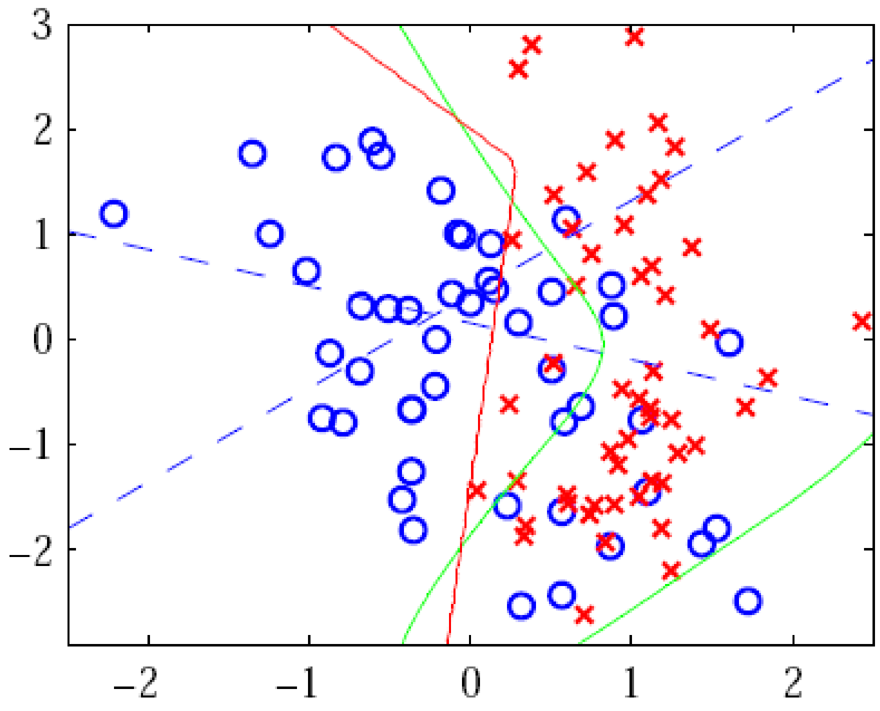


Network with 3 hidden neurons

blue	Target function
red	Approximation
– –	Output of hidden neurons

Example: classification

Neural network can define very complex decision boundaries



Network with 2 hidden neurons

green	Target function
red	Approximation
– –	Output of hidden neurons

BACKPROPAGATION ALGORITHM

Introduction

- Backpropagation algorithm was first applied in the context of NN in 1974 by Paul Werbos (“renaissance” of NN)
- It became famous after book “Parallel Distributed processing” by Rumelhart and McClelland in 1987
- Previously it was not applicable to larger networks, but nowadays due to the increased computation power(GPU) is back and is being used for training of large networks which achieve top results for many problems!
- **Credit Assignment Problem** : determining how the success of a system’s overall performance is due to the various contributions of the system’s components
- Credit Assignment Problem for ANN: assign the “blame” to each hidden neuron for its contribution to output neuron error

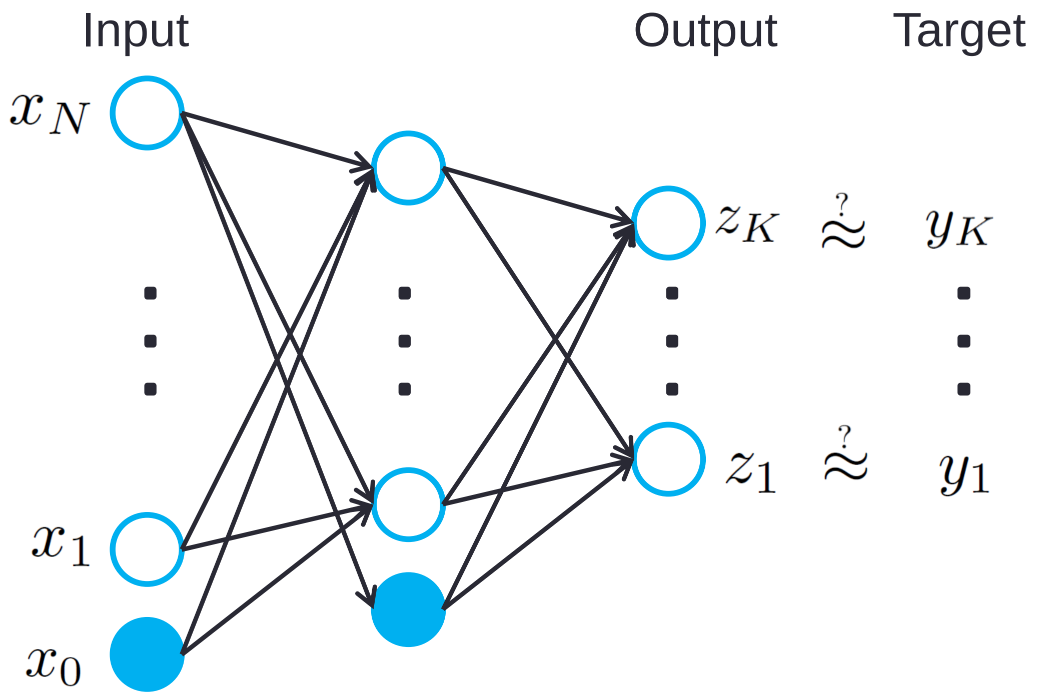
Error measure

- Consider a training set consisting of *m* samples: $\langle \boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)} \rangle \ldots \langle \boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)} \rangle$

- Error of a single output neuron: $e_k = z_k - y_k$
- Classical measure (function) of error:

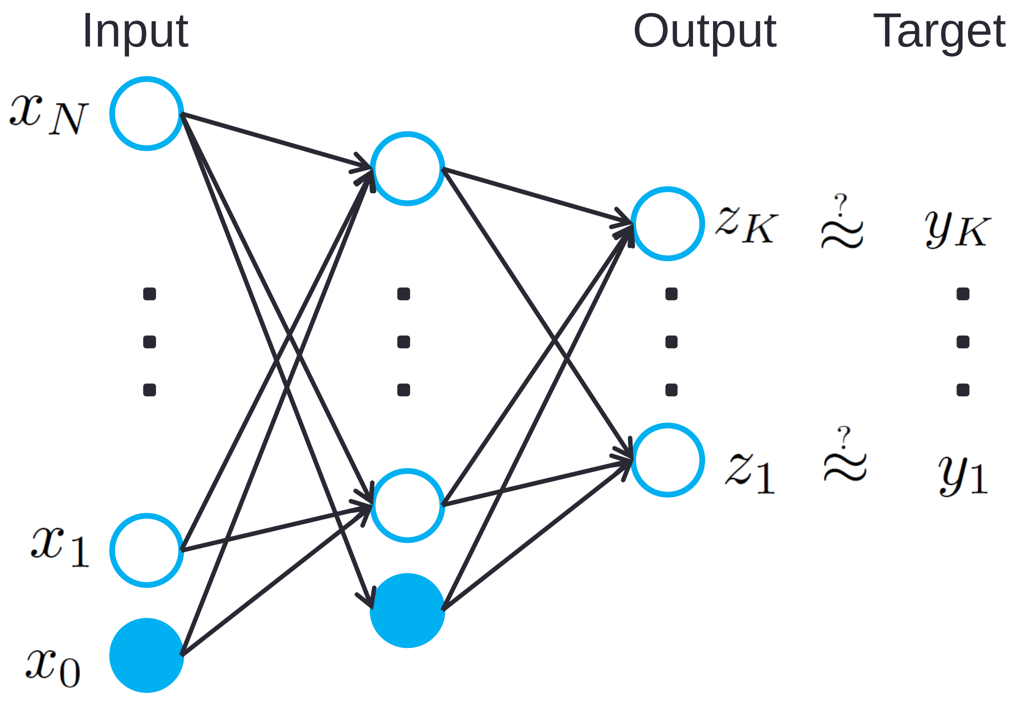
Sum of squared errors: $$E^{(i)} = \frac{1}{2} \sum_{k=0}^{K} e_k^2 = \frac{1}{2} \sum_{k=0}^{K} (z_k - y_k)^2$$

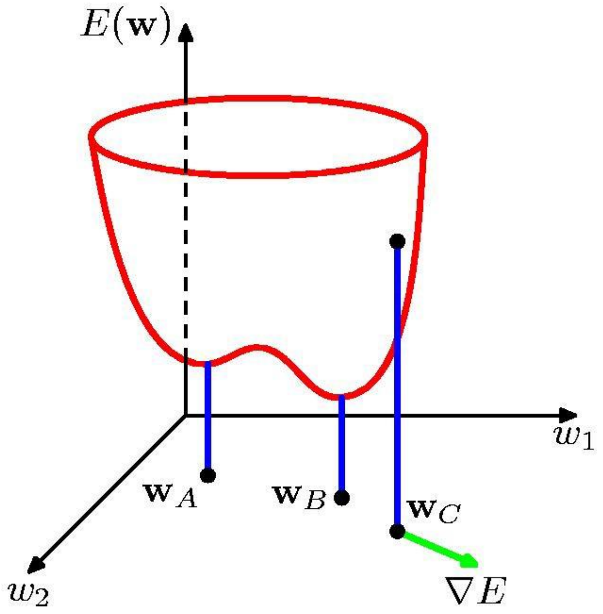
Error for a particular sample



How to choose weights to minimize the error measure?

Learning = minimizing training error



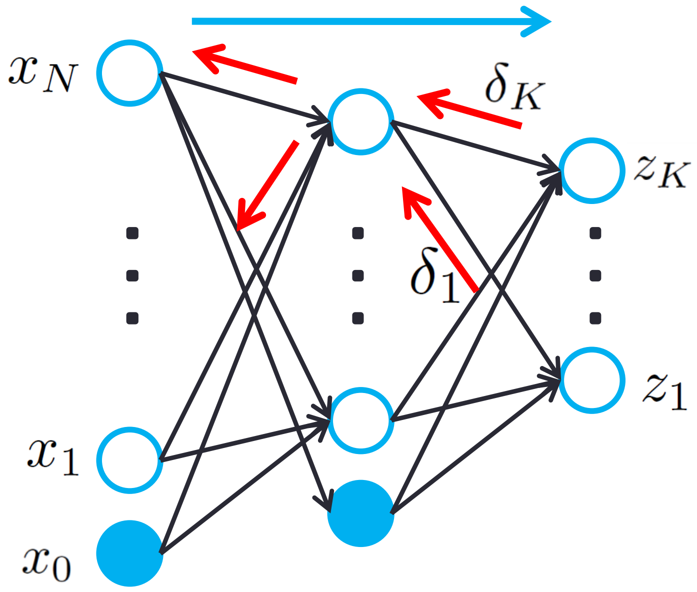


$$E^{(i)} = \frac{1}{2}\sum_{k=0}^{K} e_k^2 = \frac{1}{2}\sum_{k=0}^{K}(z_k - y_k)^2$$

- As no analytic solution is possible use **gradient descent**!
- Other techniques are also possible (adam, l-bfgs, conjugate gradient, Rprop, any optimization technique)

Backpropagation algorithm

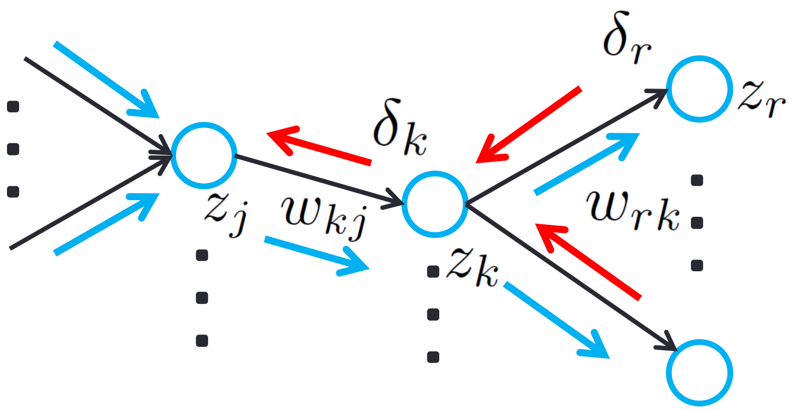
- For learning (update of weights) the gradient of the error function is needed
- The gradient of the error function is calculated by the local exchange of messages in 2 passes:
  - **Forward:** Calculate activations and outputs of all neurons $\boldsymbol{z}$
  - **Backward:** Calculate errors $\boldsymbol{\delta}$ and propagate them back



Algorithm

- **Error gradient calculation:**

$$\frac{\partial E^{(i)}}{\partial w_{kj}} = \delta_k z_j$$



- **Forward transmission**: calculate the output z_j of the neuron j
- **Backward transmission:** calculate the error δ_k of the neuron k
  - **Hidden neuron:**

$$\delta_k = \frac{\partial f_k(a_k)}{\partial a_k} \sum_{r \in \mathrm{post}(k)} \delta_r w_{rk}$$

  - **Output neuron:**

$$\delta_k = \frac{\partial f_k(a_k)}{\partial a_k} e_k$$

Batch vs online learning

- **Batch learning**
  - The error gradient for each sample from training set is calculated and accumulated. The weight update is done after all samples are seen.

$$E = \sum_{i}^{m} E^{(i)} \qquad w_{kj} := w_{kj} - \eta \nabla E$$

- **Online learning**
  - After presentation of each sample i from the training set we use the calculated error gradient for weight update:

$$w_{kj} := w_{kj} - \eta \nabla E^{(i)}$$

  - It can be used when there is no fixed training set (new data keeps coming in)
  - The noise in the gradient can help to escape from **local minimum**

- **Mini-batch (m = 32)** “Stochastic Gradient Descent”

Properties

- The algorithm is executed (usually in epochs during which each sample from the training set is presented to the network) until stopping criteria (e.g. error is smaller then some threshold) is fulfilled

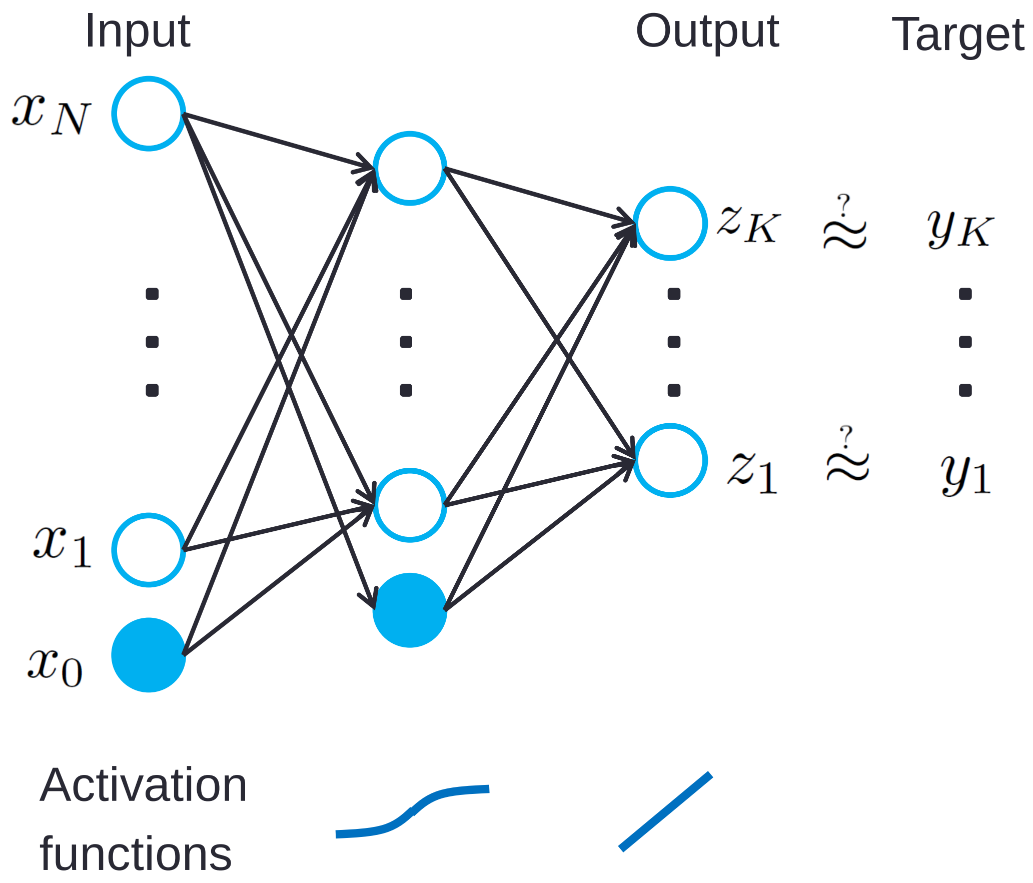
- The results may converge to a local minimum

- The convergence is not guaranteed and is very slow (if obtained)

- Normalization of input vectors is not required, but it can improve the performance

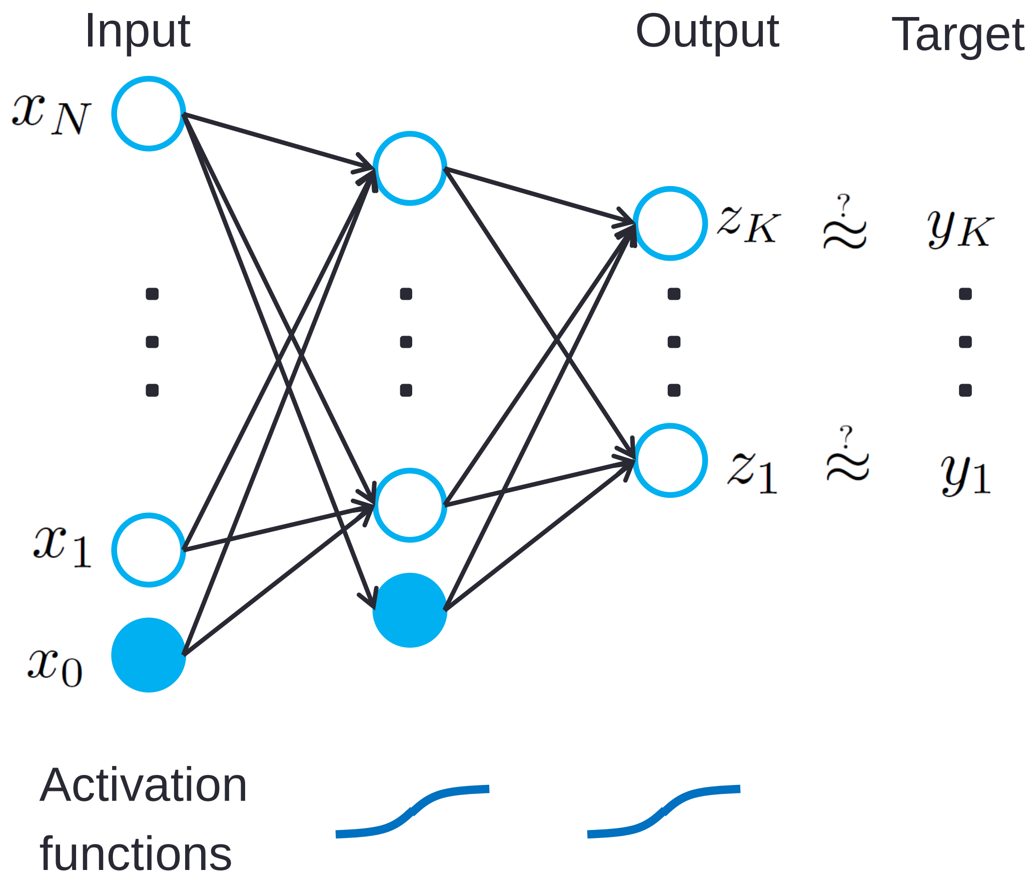
NEURAL NETWORKS EXAMPLE

Regression



- Linear activation function in output layer ensures arbitrary output range

Classification



- Sigmoid activation function in output layer ensures outputs between 0 and 1

Classification: binary vs multiclass

- Binary classification:
  - Round the output of a single neuron (with sigmoidal activation) to 0 or 1 and interpret it as a class 0 or class 1 (as in Perceptron)

- Multiclass classification:
  - Multiple output neurons: use 1-out-of-n encoding for the target value $\boldsymbol{y}^{(i)}$ (one of $y_k^{(i)}$ values is 1, all others are 0)
  - This means there is one output neuron for each class
  - Use a softmax encoding to code the output as probabilities

$$Softmax(z)_i = \frac{e^{z_i}}{\sum_{l=1}^{k} e^{z_l}}$$

Classification: binary vs multiclass

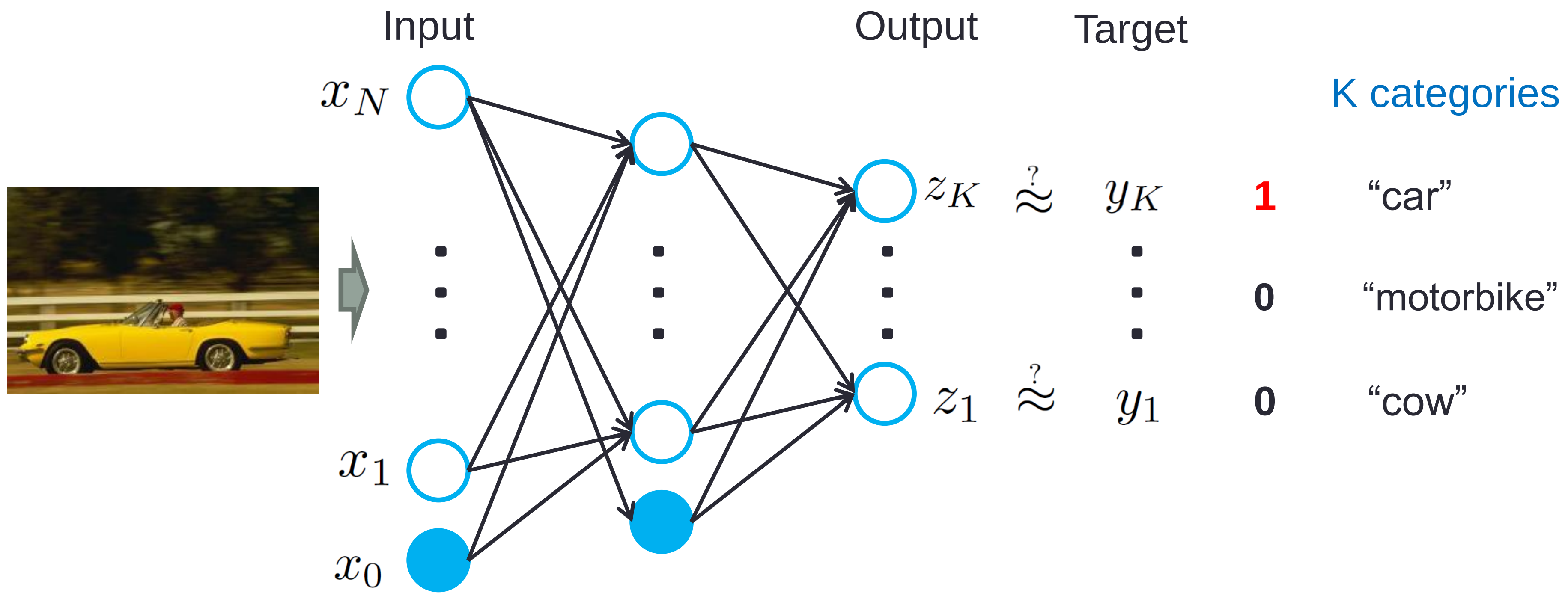
- For example, if the number of classes is 3:

$$\boldsymbol{y}^{(i)} \text{ is one of: } \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix} \text{ or } \begin{pmatrix}0\\0\\1\end{pmatrix}$$

- We want the output of ANN to be:

$$\boldsymbol{z} \approx \begin{pmatrix}1\\0\\0\end{pmatrix}, \quad \boldsymbol{z} \approx \begin{pmatrix}0\\1\\0\end{pmatrix}, \quad \boldsymbol{z} \approx \begin{pmatrix}0\\0\\1\end{pmatrix}$$

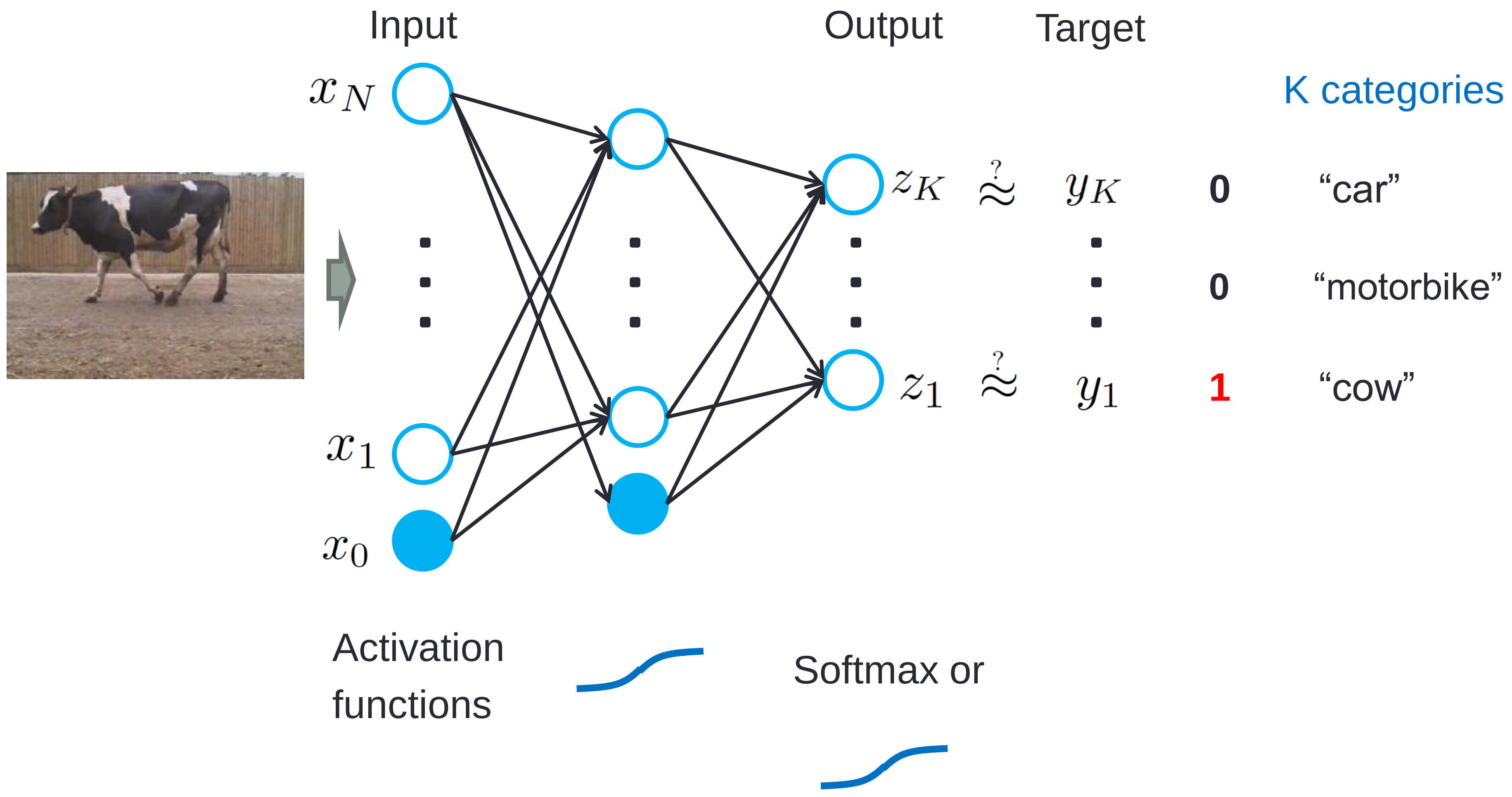
Classification training example



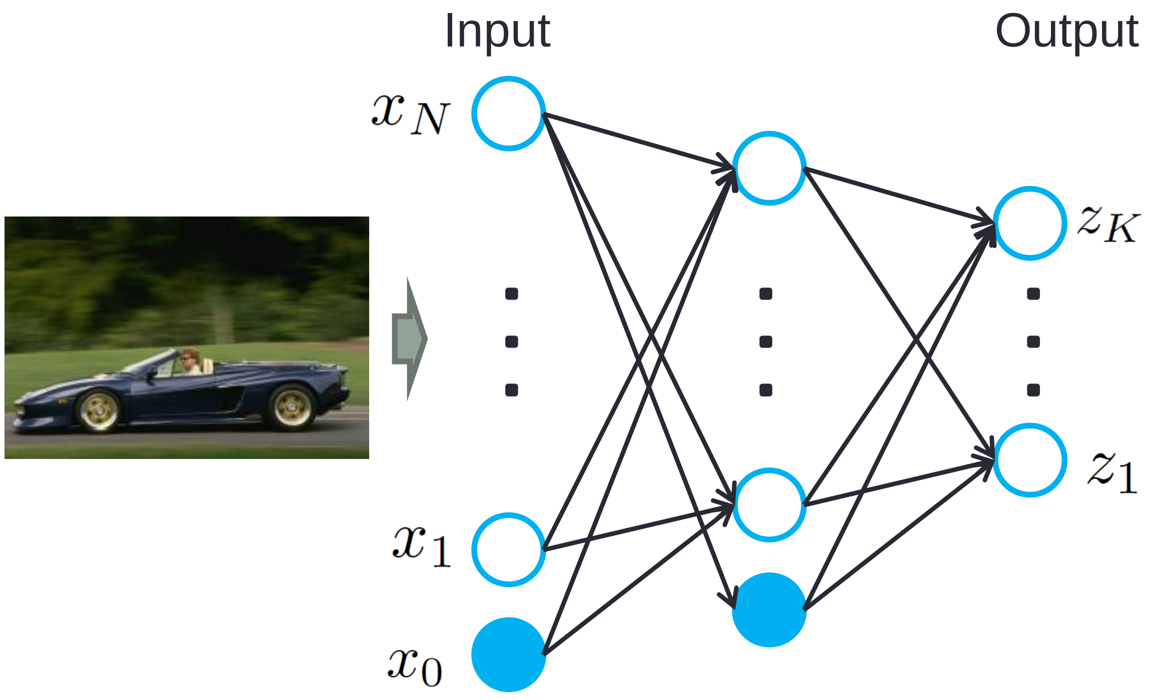
Activation functions

Softmax or

Classification training example



Classification test example (after learning)



K categories

0.9	“car”
0.03	“motorbike”
0.07	“cow”

Activation functions

Softmax or

ARTIFICIAL NEURAL NETWORKS

Properties

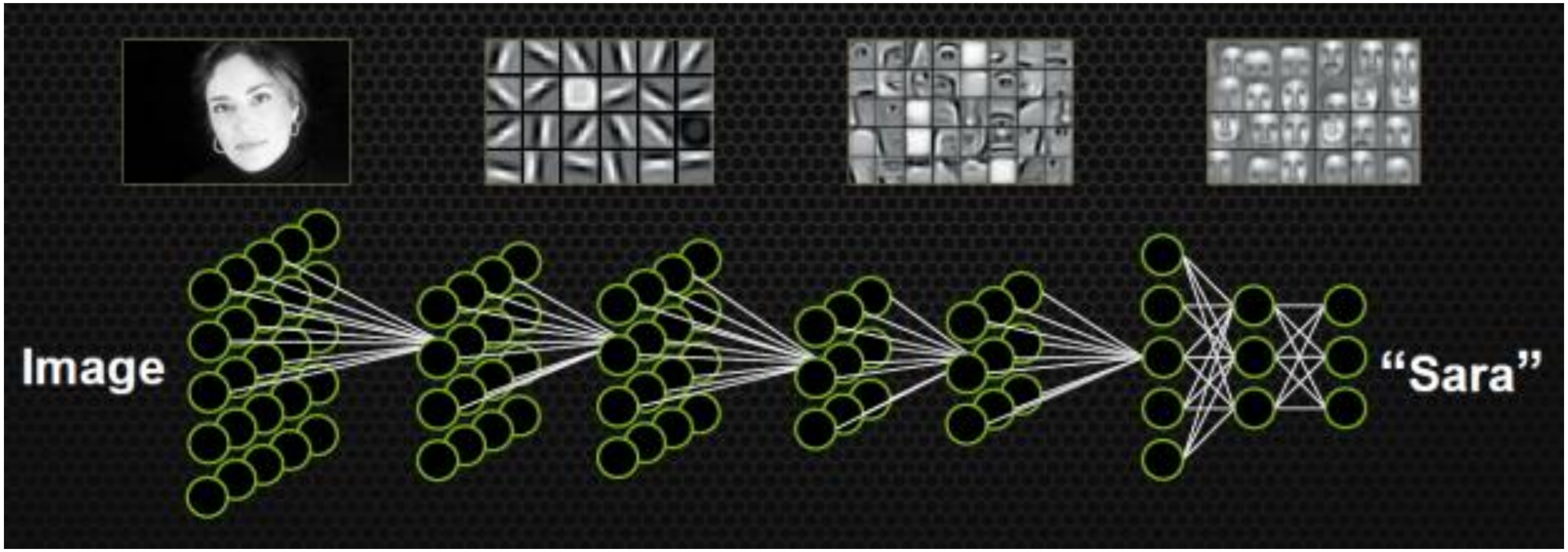
Properties

- Neurobiology analogy
- Adaptive model
- Learns input output mapping
- Learning is quite slow but testing is fast
- Data does not have to be precise and perfect
- Result does not depend on a single network element
- Fault tolerant - robust (redundancy)
- Knowledge is stored implicitly (it is hard to interpret it)

Critic: Black Box interpretation

A neural network learns. But What ?
What are the role of higher layers ?

- Pre-training unsupervised or on another dataset may help

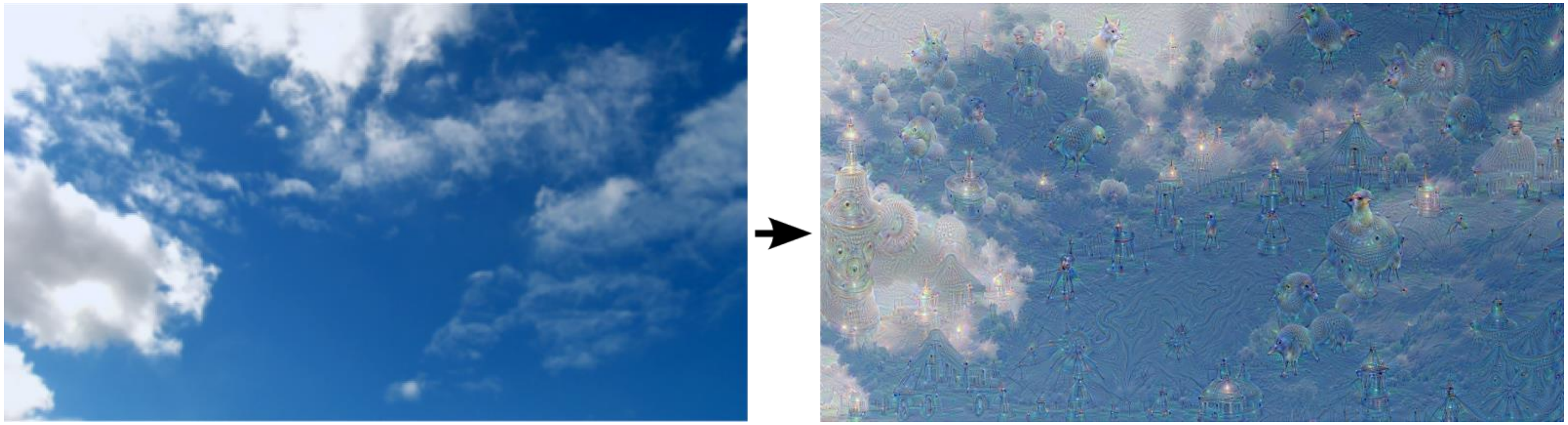


Google Deep Dream

Google Deep Dream

Google Deep Dream

Google Deep Dream



Details at:
- http://googleresearch.blogspot.co.at/2015/06/inceptionism-going-deeper-into-neural.html

Source code available at:
- https://github.com/google/deepdream

SUMMARY (QUESTIONS)

Some questions…

- What types of neural networks there are?
- What are ANN?
- Types of ANN?
- Applications of ANN?
- Artificial neuron model?
- What is an activation function? Types and usage?
- What is Perceptron?
- Convergence properties of Perceptron?
- Binary linear classification with Perceptron?
- Perceptron learning algorithm?
- Limitations of Perceptron?
- Can you use Perceptron to classify nonlinear data?

Some questions (2)…

- What is feedforward architecture?
- What is hidden layer and what is it useful for?
- What function implements ANN with 1 hidden layer with sigmoid activation function?
- Can Perceptron solve XOR? How about Multilayer Perceptron?
- Computational properties of ANN?
- What is Credit Assignment Problem? In context of ANN?
- What is backpropagation algorithm?
- What error function minimizes backpropagation?
- Why is backpropagation algorithm used?
- Weight update rules for output and hidden neurons?
- What are online and batch learning? What is the difference?
- How can one use ANN for classification? And how for regression?
- ANN properties?

What is next?

- Support Vector Machines : a powerful linear classifier
- Kernel methods
- Multiclass classification methods

References

- Waferscale: http://www.kip.uni-heidelberg.de/cms/vision/projects/facets/neuromorphic_hardware/waferscale_integration_system/
- SpiNNaker: https://spinnaker.cs.manchester.ac.uk
- TrueNorth: http://www.research.ibm.com/articles/brain-chip.shtml