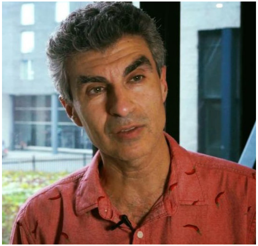
Machine Learning

– winter term 2016/17 –

Chapter 07: Neural Networks I

Prof. Adrian Ulges
Masters "Computer Science"
DCSM Department
University of Applied Sciences RheinMain

Resources for the next Chapters



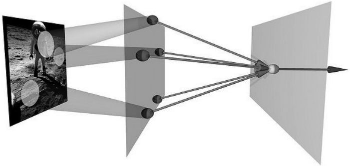
Online Book 1: The nice one

- Nielsen: "Neural Networks and Deep Learning"
  `http://neuralnetworksanddeeplearning.com`

Online Book 2: The tough one

- Goodfellow, Bengio, Courville: "Deep learning"
  `https://www.deeplearningbook.org`

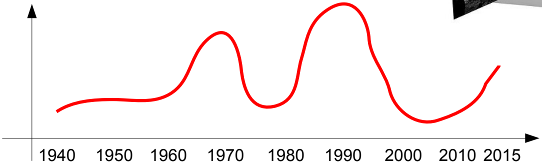
(Artificial) Neural Networks.... image: [1]

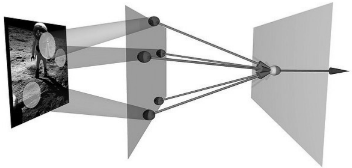


- ... are one of the core topics of **artificial intelligence** (AI)
- ... are **biologically inspired**: They are mathematical models simulating the process of "thinking" in living organisms' brains
- ... are **graphs** (or *networks*) of interlinked atomic processing units (*neurons*)
- ... are not programmed but define their own behavior entirely through the graph's links and weights. These are adjusted by **training**.
- ... are covered by two research fields
  1. **computational neuroscience**
     (goal: better understanding of biological processes)
  2. **machine learning / AI (here)**
     (goal: solving practical data analysis problems)

Neural Networks....

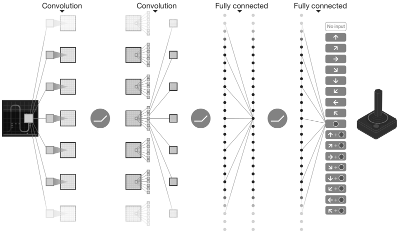
- ... have a turbulent **history**

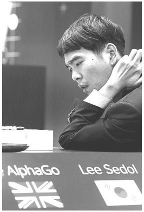


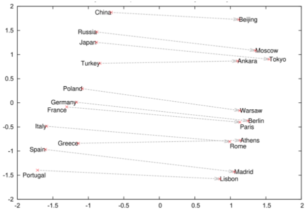


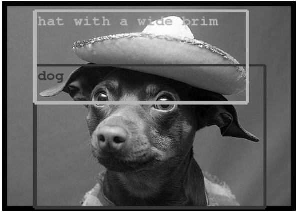
- ... exist in many **variants**
  - multi-layer perceptron (*in this lecture*)
  - convolutional neural networks (*in this lecture*)
  - Boltzmann machines
  - RBF networks
  - recurrent neural networks
  - LSTM networks
  - ...
- ... are the most intensely machine learning model these days (**"deep learning"**)

Deep Learning Applications images from [5] [6] [2] [4] [7]







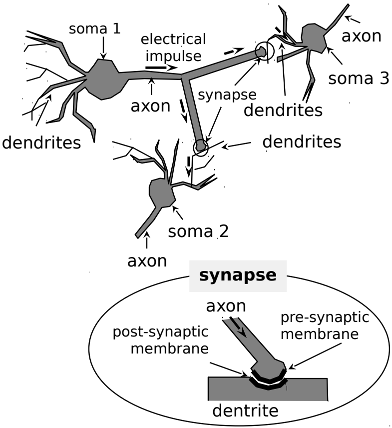


Neural Networks: Biological Inspiration

Neurons are brain cells that send and receive electrical impulses.

Neurons: Terminology

- **Cell body (*soma*)**: center of the neuron, with a diameter of $5 - 100\mu m$
- **axon**: thin, long (up to 1m) nerve fibre, splits and transmits impulses to other neurons
- **dendrites**: small, branch-like outgrowths that receive impulses and transmit them to the soma
- **synapses**: electro-chemical link between two neurons. Transfers signal from axon to dendrites via transmitter substances.



From Biological to Artificial Neural Networks

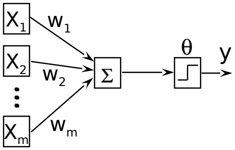
- Biological neural networks are (much) more complex than artificial ones
  - **connectivity**: 10^{11} neurons, each connected with $\approx 7,000$ others
  - **adaptivity**: number and connections of neurons change over time
- Biological neural networks are **asynchronous** and compute with rather **low frequency** (1 KHz)
- The **circuit layout** is unknown

Learning

- Learning happens at synapses, by changing the **transmitter dose**
  - **sensitization**: more transmitter, enhancing the signal
  - **desensitization**: less transmitter, supressing the signal

Artificial Neural Networks

- **weighted sum + activation function**
- **learning** happens by adapting weights
- **memory** happens by storing weights



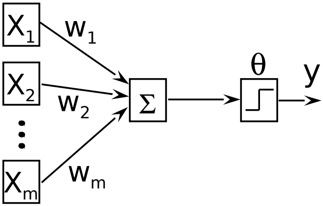
Outline

The McP-Neuron

- The **historically oldest** (1943) neuron by McCulloch & Pitts
- strongly simplified model of biological neurons
- **no learning** yet

Definition

- input
  $\mathbf{x} = (x_1, ..., x_d) \in \{0, 1\}^d$
- output $y \in \{0, 1\}$
- weights $\mathbf{w} = (w_1, ..., w_d) \in \{-1, 1\}^d$
  - $w_j = 1$: stimulating *(="anregend")*
  - $w_j = -1$: inhibitory *(= "hemmend")*
- threshold $\theta \in \mathbb{R}$



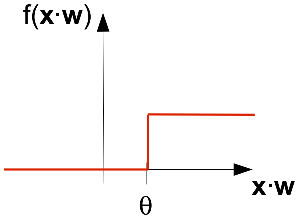
The McP-Neuron

Computational Model

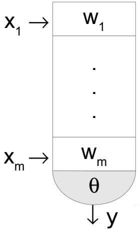
- compute the scalar product between input signal $\mathbf{x}$ and weights $\mathbf{w}$, namely $\mathbf{w} \cdot \mathbf{x}$
- apply an **activation function** f *(here, a step function)*, obtaining the output y

$$
\begin{aligned}
y &= f(\mathbf{w} \cdot \mathbf{x}) \\
&= \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq \theta \\ 0 & \text{else} \end{cases}
\end{aligned}
$$

- The McP-Neuron is a **function** $\phi^{\mathbf{w},\theta} : \{0,1\}^d \to \{0,1\}$ with parameters $\mathbf{w}, \theta$
- If $\phi^{\mathbf{w},\theta}(\mathbf{x}) = 1$, we say the neuron "is activated", or the neuron "fires"



The McP-Neuron

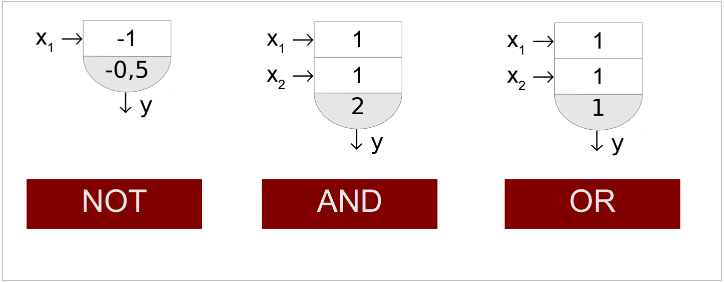


McP-Neurons: Graphical Representation

- input **x**
- weights **w**
- threshold θ

What can McP-Neurons do?

- We can model logic gates using McP-neurons



From McP-Neurons to the Perceptron

McP-Neurons: Limitations

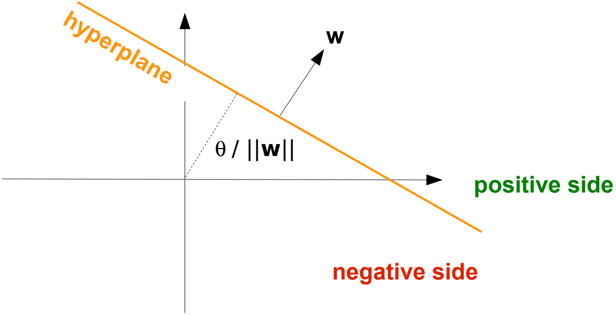
- only boolean inputs/weights/outputs
- only binary activation functions
- no learning

Extensions: The Perceptron

- real-valued weights and inputs $\mathbf{w}, \mathbf{x} \in \mathbb{R}^d$
- We will introduce a learning algorithm, the **Delta-rule**

Perceptron: Graphical Interpretation

- The parameters $\mathbf{w}, \theta$ define a **hyperplane**
- $\mathbf{w} \cdot \mathbf{x} \geq \theta$: $\mathbf{x}$ is on the **positive side**
- $\mathbf{w} \cdot \mathbf{x} < \theta$: $\mathbf{x}$ is on the negative side



Outline

1. Neurons

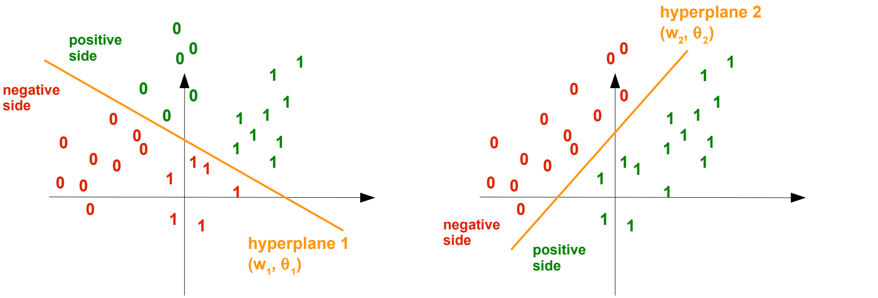
2. Training Neurons

3. The Capacity of Neurons

4. Neural Networks

Perceptron Training: Basics

- A neuron's function (or *behavior*) $\phi^{\mathbf{w},\theta}$ is determined by the parameters $\mathbf{w}$ and θ
- Given labeled training data: How do we find the *best* parameters / the *best* hyperplane?



bad parameters:
error rate 12 / 30 = 40%

good parameters:
error rate 0%

Perceptron Training: Vector Augmentation

Before learning, we simplify notation using a simple trick called **vector augmentation**:

Idea

- **omit** the additional **threshold parameter** θ and include it in the **weight vector**

So far...

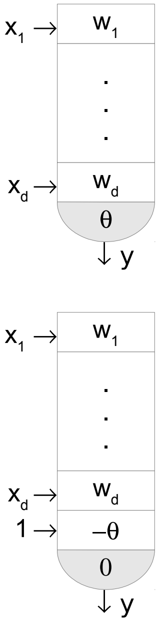
- $\mathbf{x} := (x_1, ..., x_d), \mathbf{w} := (w_1, ..., w_d)$
- fire if $\mathbf{w} \cdot \mathbf{x} \geq \theta$ (or $\mathbf{w} \cdot \mathbf{x} - \theta \geq 0$)

Now

- $\mathbf{x} := (x_1, ..., x_d, 1), \mathbf{w} := (w_1, ..., w_d, -\theta)$
- fire if $\mathbf{w} \cdot \mathbf{x} \geq 0$

Remark

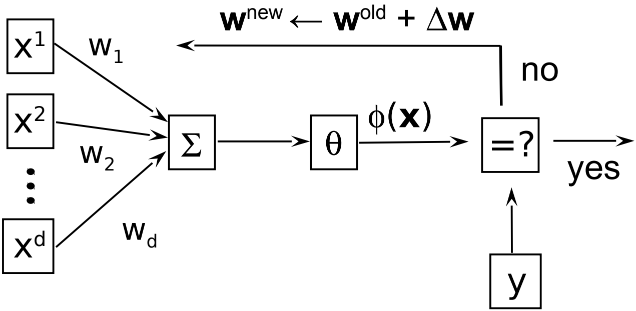
This is just a change of notation (θ is not written separately any more), not of behavior!



Perceptron Training

Learning happens by an **iterative optimization**. We start with random values for $\mathbf{w}$ and θ, and in each iteration we ...

- ... **classify a training sample** $\mathbf{x}$ and compare the result $\phi(\mathbf{x})$ with the targeted result y
- If necessary, we **correct the neuron's parameters** such that the output $\phi(\mathbf{x})$ is adapted towards the desired output y



Perceptron Training: The Delta Rule

How do we compute the 'right' update $\Delta\mathbf{w}$? There are different strategies. The most famous one is the **Delta Rule**:

$$\triangle\mathbf{w} = \lambda \cdot \Big(y - \phi(\mathbf{x})\Big) \cdot \mathbf{x}$$

Remarks

- We call λ the **learning rate**
- Note: If $\mathbf{x}_i$ is classified correctly, $\Delta\mathbf{w} = \mathbf{0} \rightarrow$ no update

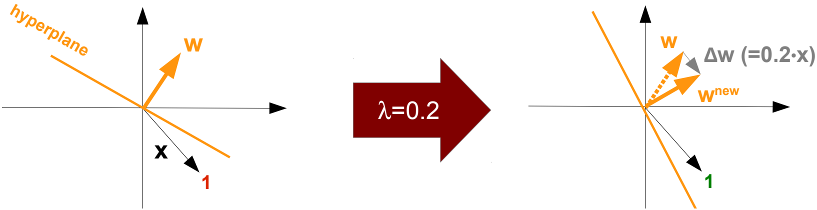
function train_perceptron($\mathbf{x}_1, ..., \mathbf{x}_n, y_1, ..., y_n, \lambda$) :
    initialize $\mathbf{w}$ randomly
    until convergence:
        choose a random training sample $(\mathbf{x}, y)$ (with $y \in \{0, 1\}$)
        compute $\phi(\mathbf{x})$ // classify $\mathbf{x}$
        if $\phi(\mathbf{x}) <> y$:

$$\mathbf{w} := \mathbf{w} + \underbrace{\lambda \cdot \Big(y - \phi(\mathbf{x})\Big) \cdot \mathbf{x}}_{\Delta\mathbf{w}}$$

The Delta Rule: Motivation

$$\triangle \mathbf{w} = \lambda \cdot \Big(y - \phi(\mathbf{x}) \Big) \cdot \mathbf{x}$$

- Why does the Delta rule work?
- We illustrate what happens when a training sample is misclassified ...



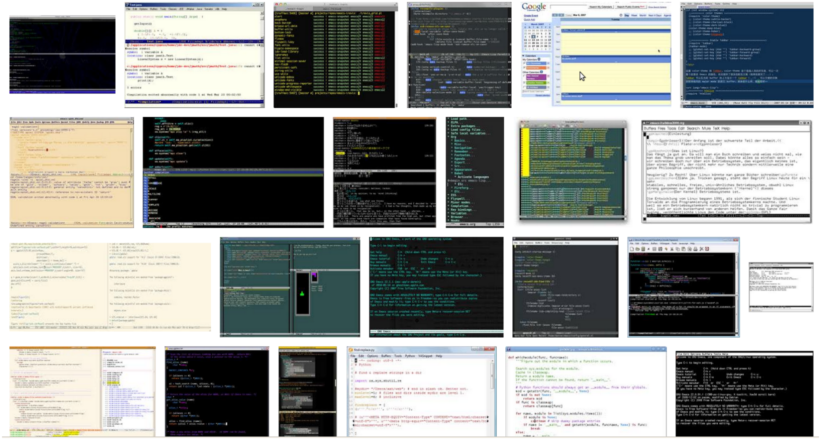
The Delta Rule: Motivation

Formal Motivation

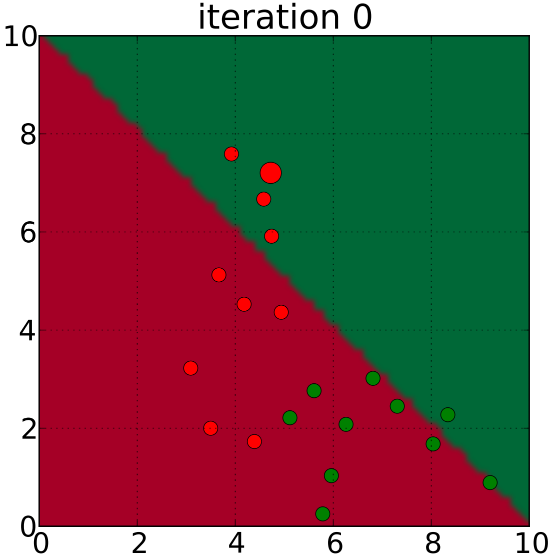
The Delta Rule: Motivation

Formal Motivation

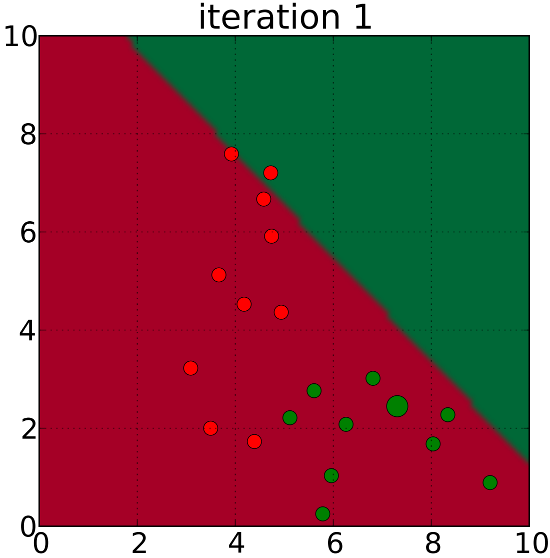
The Delta Rule: Code Example



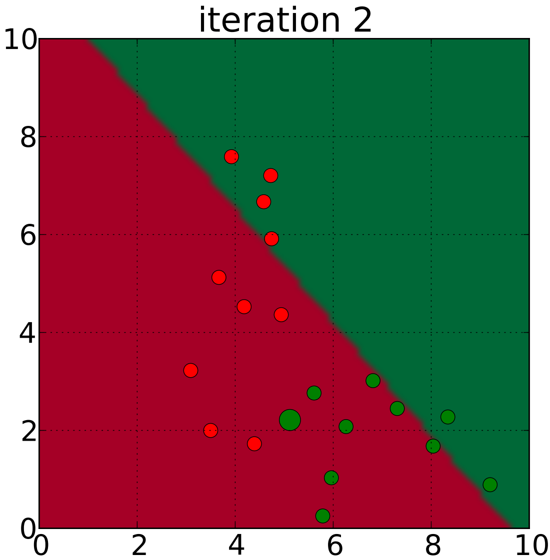
The Delta Rule: Code Example



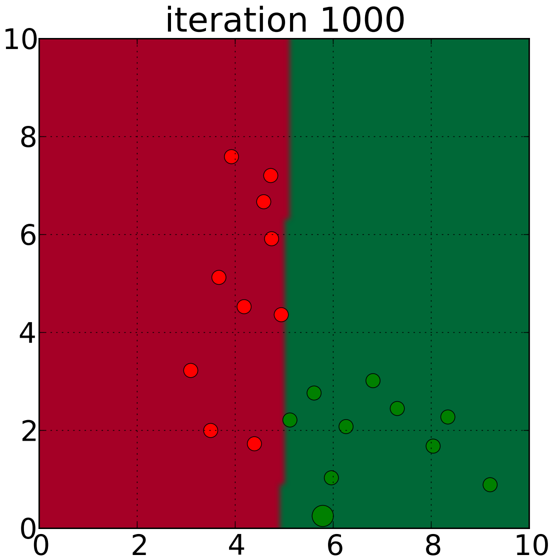
The Delta Rule: Code Example



The Delta Rule: Code Example



The Delta Rule: Code Example



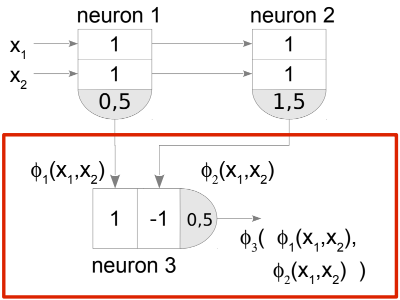
Outline

An Important Question...

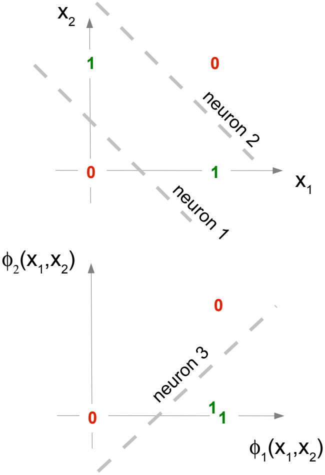
Can a single perceptron learn any Function $f : \mathbb{R}^d \to \{-1, 1\}$?

Achieving Non-Linearity by connecting Neurons

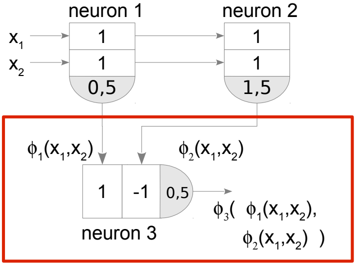
We can realize XOR by connecting multiple neurons!



x_1	x_2	$\phi_1(x_1,x_2)$	$\phi_2(x_1,x_2)$	$\phi_3(\phi_1(x_1,x_2), \phi_2(x_1,x_2))$	$x_1 \otimes x_2$
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	1	1	0	0



Achieving Non-Linearity



Remarks

- This is only possible because of the neurons' non-linear activation functions!

- with activation functions

$$y = f(w_1'' \cdot f(w_1 x_1 + w_2 x_2) + w_2'' \cdot f(w_1' x_1 + w_2' x_2))$$

- without activation functions

$$\begin{aligned} y &= w_1'' \cdot (w_1 x_1 + w_2 x_2) + w_2'' \cdot (w_1' x_1 + w_2' x_2) \\ &= w_1''' x_1 + w_2''' x_2 \end{aligned}$$

This is just a linear function (and does not solve XOR)

Achieving Non-Linearity by connecting Neurons

Theorem (Learning Capacity of Networks of McP-Neurons)

We can realize any boolean function $f : \{0,1\}^d \rightarrow \{0,1\}$ *by connecting not more than* $2^d + 1$ *McP-neurons.*

Proof(by construction)

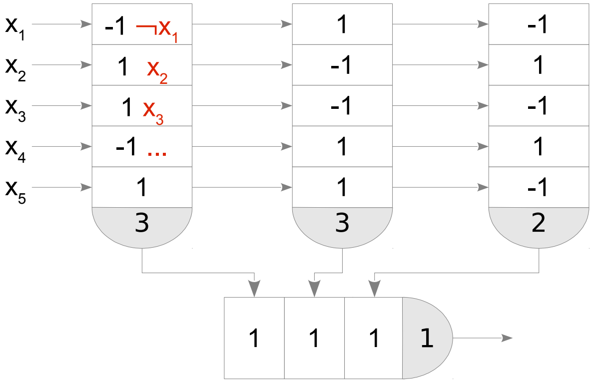
Proof by Construction

Proof by Construction: Example

$\neg x_1 \wedge x_2 \wedge x_3 \wedge \neg x_4 \wedge x_5 \quad \vee$

$x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4 \wedge x_5 \quad \vee$

$\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4 \wedge \neg x_5$



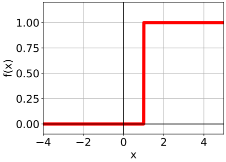
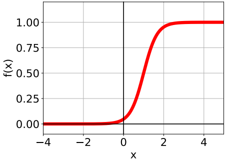
The McP-Neuron: Do-it-Yourself

1. Sketch an McP-Neuron that models the (even) **parity bit** for a given sequence of 3 input bits, x_1, x_2, x_3.
2. Given n input bits, the above model seems to require $2^n + 1$ neurons. Can you do 'better' by using more than 2 layers?

The McP-Neuron: Do-it-Yourself

Perceptron: Activation Functions

We can use **other activation functions**:

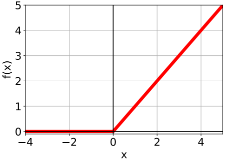
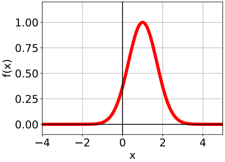
plot	function	parameters
	$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{else.} \end{cases}$	θ
	$f(x) = \frac{1}{1+e^{-(x-\theta)}}$ *(sigmoidal)*	θ

Remarks

- The sigmoid approximates a step function, but is also **differentiable** *(with $f'(x) = f(x) \cdot (1 - f(x))$)*

Perceptron: Activation Functions

We can use **other activation functions**:

plot	function	parameters
	$f(x) = max(0, x)$ *rectified linear unit (RELU)*	-
	$f(x) \propto e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ *(Gaussian)*	μ, σ

Can connected Perceptrons learn Any Function?[1]

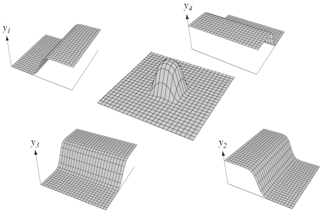
Theorem (Learning Capacity of Networks of Perceptrons)

By connecting multiple perceptrons with sigmoidal activation function, we can approximate any continuous function $f : [0,1]^d \rightarrow [0,1]$.

Remarks

- By connecting perceptrons, we can learn <u>any</u> decision boundary!
- This works with just **one hidden layer** (see below).
- Open Question: How many perceptrons do we need?
- Open Question: How do we *find* a solution that *generalizes properly*?

"*In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube. Only mild conditions are imposed on the univariate function. Our results settle an open question about representability in the class of single bidden layer neural networks. In particular, we show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity.* "



(Cybenko., G. [3])

[1] Try Nielsen's online demo: http://neuralnetworksanddeeplearning.com/chap4.html

Outline

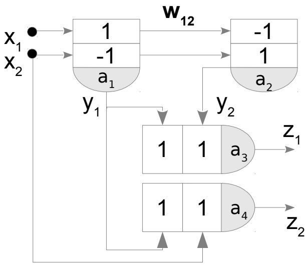
Definition: Neural Network

Neural Network Definition

- A neural network is a set of **(partially) connected** neurons
- The output signal of a neuron can be used as input signals to (multiple) other neurons
- The **networks' input** consists of all input signals that are not derived from other neurons
- The **network's output** consists of all output signals that are not used as input for other neurons
- Each link between two neurons has a **weight**. We denote the weight of the connection from neuron i to neuron j with w_{ij}.

Example

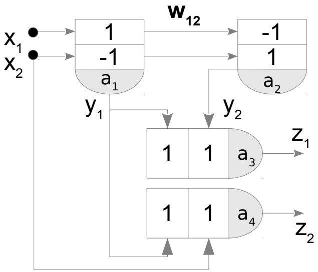
- 4 neurons $n_1, \ldots, n_4$
- activation functions $a_1, \ldots, a_4$
- network input: x_1, x_2
- network output: z_1, z_2
- "hidden signals": y_1, y_2

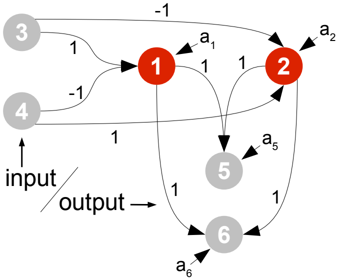


Neural Network: Graphical Representation

Neural networks are **weighted, directed graphs**

- Neurons are **nodes**, connected by weighted **edges**
- **Inputs** and **outputs** are modeled as separate nodes





Network Topologies

We can distinguish two general **network topologies**

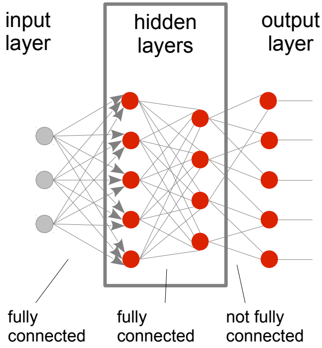
- feedforward networks
- recurrent networks

Feedforward Networks: Layers

- **Feedforward networks** are DAGs ("directed acyclic graphs"), i.e. they do not contain any cycles.
- The signal is never propagated **backwards** through the network (hence "feedforward")
- **Convention**: We organize feedforward networks in **layers**

Layer Architecture

- Every neuron is **only** connected with neurons from the previous and next layers
- We call two layers **fully connected** in case all of their neurons are connected



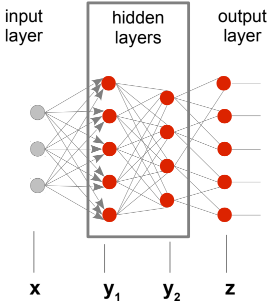
Feedforward Networks: Computation

Computational Model

- the signal is propagated through the network instantly
- We collect each layer's weights+biases in a **matrix**/**vector** *(for non-existing edges, entries are zero)*.

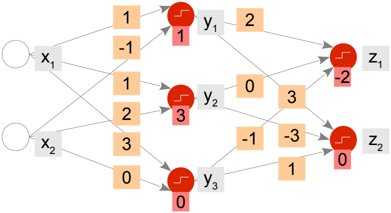
Example (two hidden layers)

- We collect all the signals leaving **each layer** in a **vector**
  - input $\mathbf{x} = (x_1, ..., x_n)$
  - hidden layer 1: $\mathbf{y_1} = (y_1^1, ..., y_m^1)$
  - hidden layer 2: $\mathbf{y_2} = (y_1^2, ..., y_p^2)$
  - output $\mathbf{z} = (z_1, ..., z_q)$
- Each layer applies (1) a weighted sum (a linear operation!), and (2) a non-linear activation f (for each neuron)
  - $\mathbf{y_1} = f(W \cdot \mathbf{x} + b)$
  - $\mathbf{y_2} = g(W' \cdot \mathbf{y_1} + b')$
  - $\mathbf{z} \ = h(W'' \cdot \mathbf{y_2} + b'')$



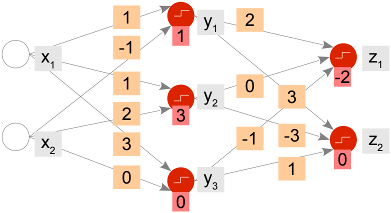
Do-Forward-Computation Yourself

Given is the following network with threshold activation functions and input $\mathbf{x} = (1, 0)$. Compute the output z_1.

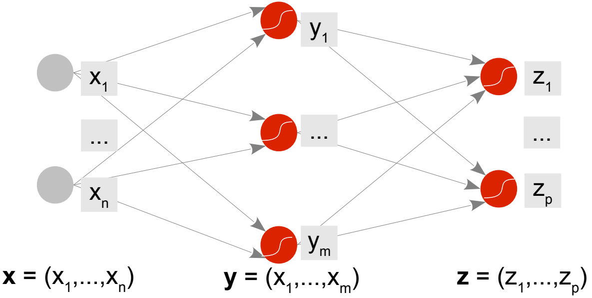


Do-Forward-Computation Yourself

Let's do it again, this time in matrix notation:



The 3-layer Perceptron (MLP)



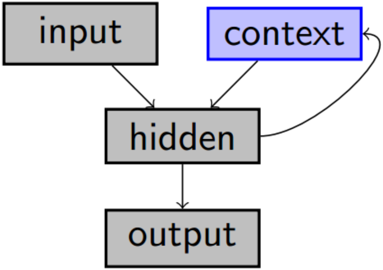
In the following, we focus on the most basic type of neural network, the **multi-layer perceptron** (MLP).

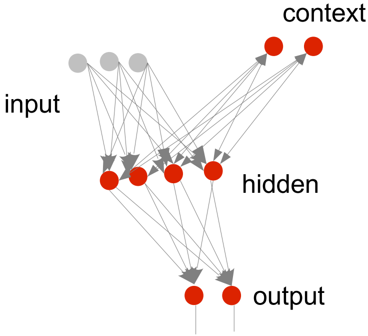
- The network is **feed-forward**
- There is only **one hidden** layer
- All layers are **fully connected**
- The network uses a **sigmoidal** activation function
- We know: such a network can learn **any function**!

Recurrent Neural Networks

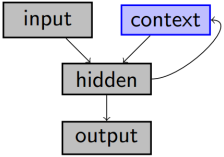
In contrast to feedforward networks, **recurrent networks** may contain **cycles** $\rightarrow$ the signal is propagated backwards through the network!

Example Architecture: Elman Networks





Recurrent Neural Networks



- The signals in the network must be **clocked**
- We extend the computational model with a **time component** $t = 1, 2, 3, \ldots$
- Example: An OCR system recognizing a sequence of characters $\mathbf{x}_1, \mathbf{x}_2, \ldots$ (here, t is the character number)
- At each time t, the network is fed an input $\mathbf{x}(t)$

```
function update_elman_network(x(t))
        hidden(t) := f(x(t), context(t-1))
        context(t) := g(hidden(t))
        output(t) := h(hidden(t))

```

- This way, the network achieves a **memory effect**!
- Example "... in Europe. **I**taly ..."

References

[1] fdecomite: Minsky & Papert Model.
https://flic.kr/p/5VsZ1M (retrieved: Nov 2016).

[2] Google DeepDream robot: 10 weirdest images produced by AI 'inceptionism' and users online (Photo: Reuters).
http://www.straitstimes.com/asia/east-asia/alphago-wins-4th-victory-over-lee-se-dol-in-final-go-match (retrieved: Nov 2016).

[3] G. Cybenko.
Approximation by Superpositions of a Sigmoidal Function.
Mathematics of Control, Signals, and Systems (MCSS), 2(4):303–314, Dec. 1989.

[4] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean.
Distributed Representations of Words and Phrases and their Compositionality.
In Advances in Neural Information Processing Systems 26, pages 3111–3119. Curran Associates, Inc., 2013.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis.
Human-level control through deep reinforcement learning.
Nature, 518(7540):529–533, 02 2015.

[6] M.-A. Russon.
Google DeepDream robot: 10 weirdest images produced by AI 'inceptionism' and users online.
http://www.ibtimes.co.uk/google-deepdream-robot-10-weirdest-images-produced-by-ai-inceptionism-users-online-1509518 (retrieved: Nov 2016).

[7] C. Szegedy.
Building a deeper understanding of images (Google Research Blog).
https://research.googleblog.com/2014/09/building-deeper-understanding-of-images.html (retrieved: Nov 2016).