DD2421
Introduction to Artificial Neuronal Networks

Jörg Conradt

NeuroComputing Systems
Computational Science and Technology
School of Electrical Engineering and Computer Science
KTH Stockholm, Sweden

http://neurocomputing.systems

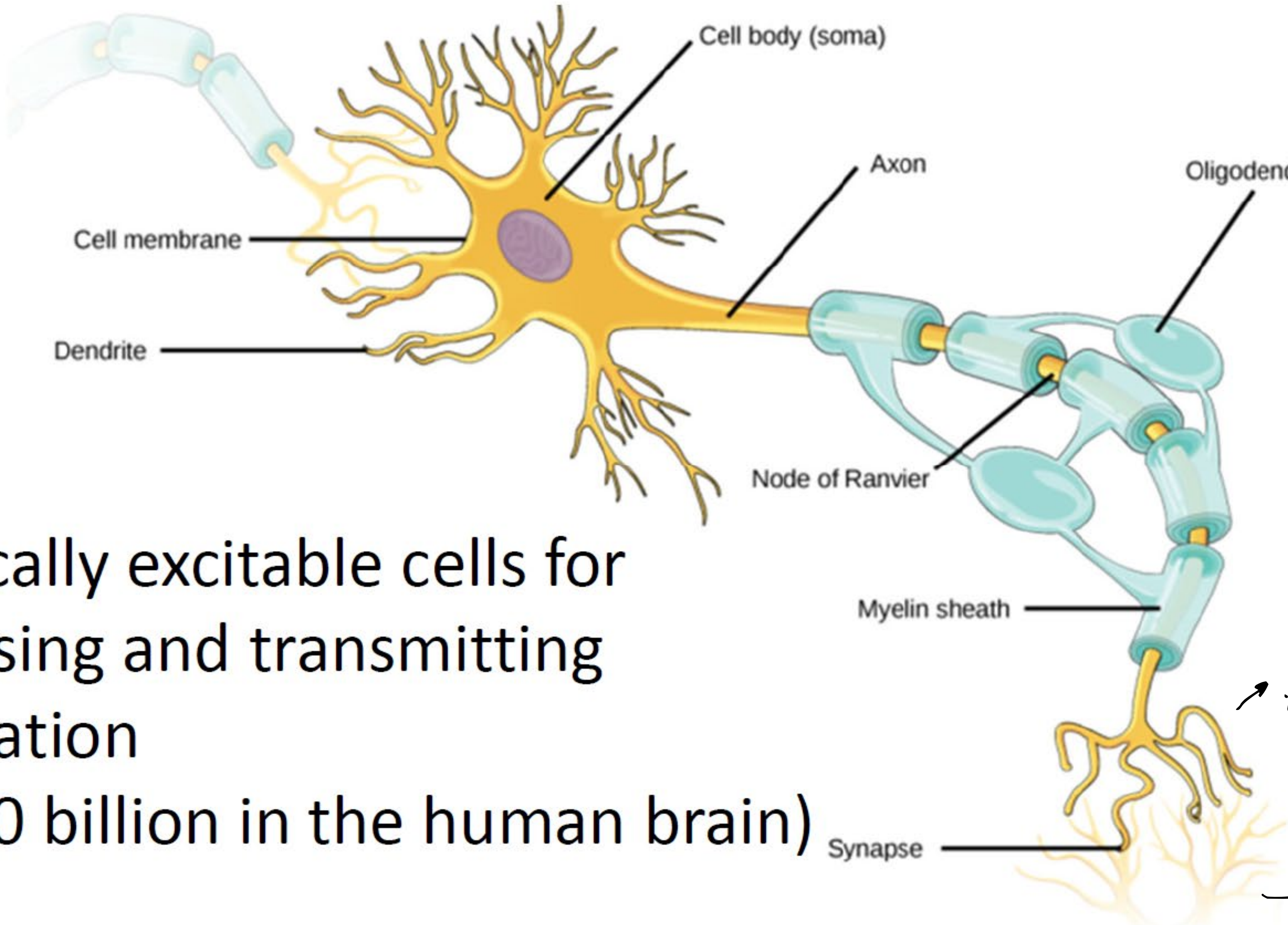
Human Brain Project
ine Institute of Neuromorphic Engineering
BOSCH Invented for life
SpiNNaker
bccn munich
iniVation
iniLabs
BMW GROUP Research and Technology

NCS
HTTP://NEUROCOMPUTING.SYSTEMS
CST - EECS - KTH STOCKHOLM, SWEDEN

Overview

- Biological Neurons
- Artificial Neurons
- What can a single Neuron do?
- From Neurons to Networks
- How to Train a Neuronal Network
- Tricks of the Trade … how to get your Network working
- Software Example
- Other classes of Neuronal Network

Biological Neurons



bifurcates into different outputs

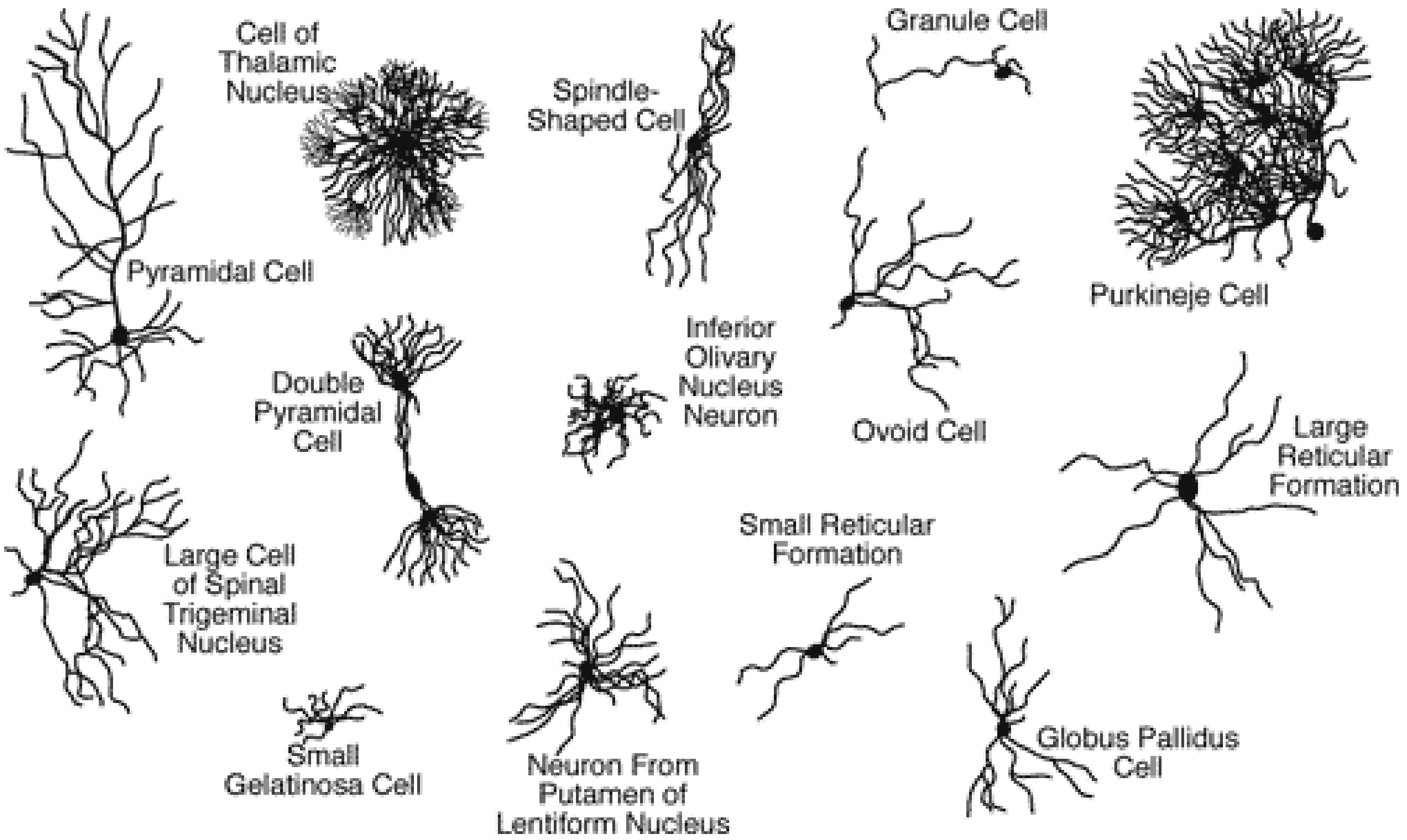
new neuron's inputs.

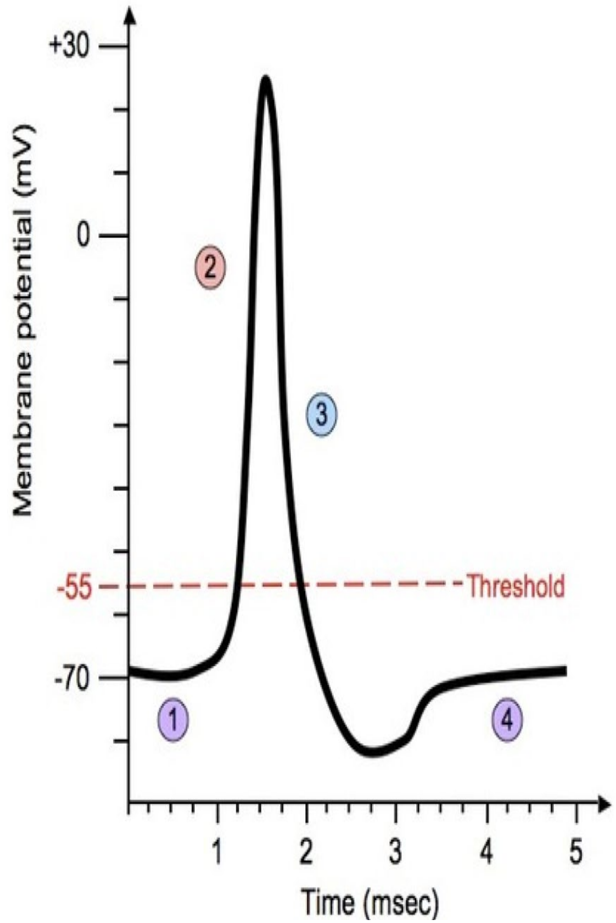
electrically excitable cells for processing and transmitting information
(ca. 100 billion in the human brain)

Biological Neurons

A large variety of cell types, but similar function:

(leaky) Integrate and Fire

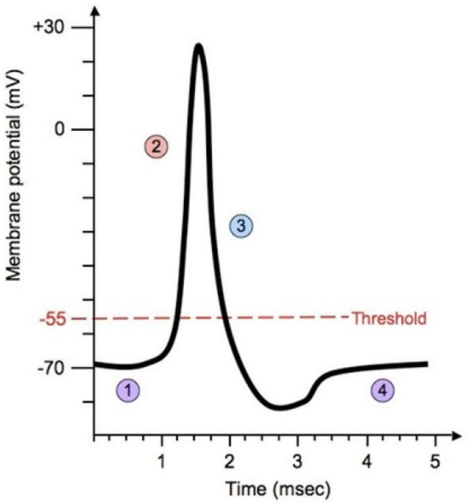


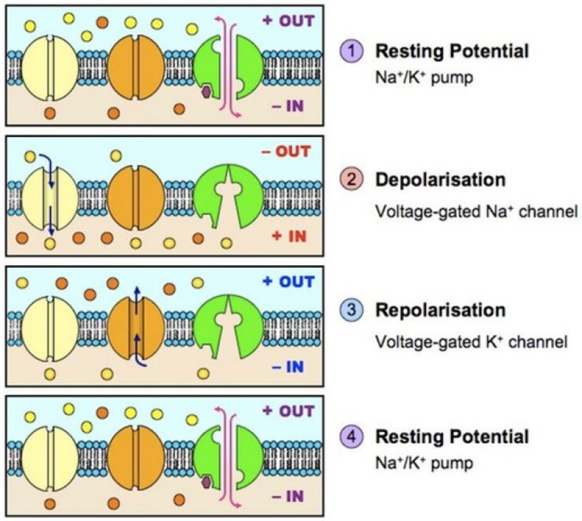


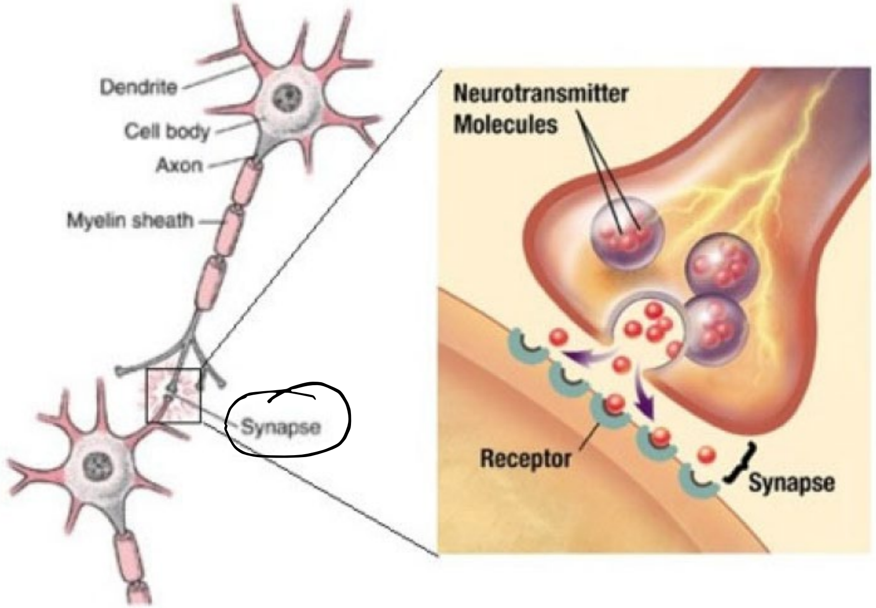
they all have a cell body.
different on how they capture or project info.

NCS

Biological Details







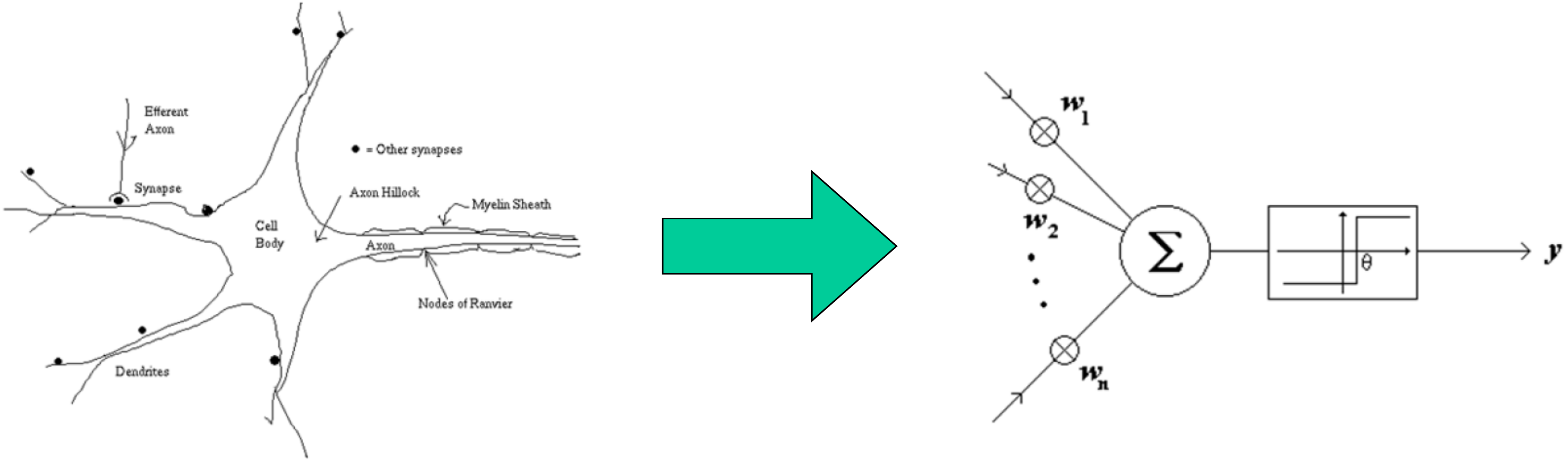
More details on neuroscience? KTH CST course in P4

https://www.kth.se/social/course/DD2401/

NCS

Artificial Neuron

From Biology to Technology

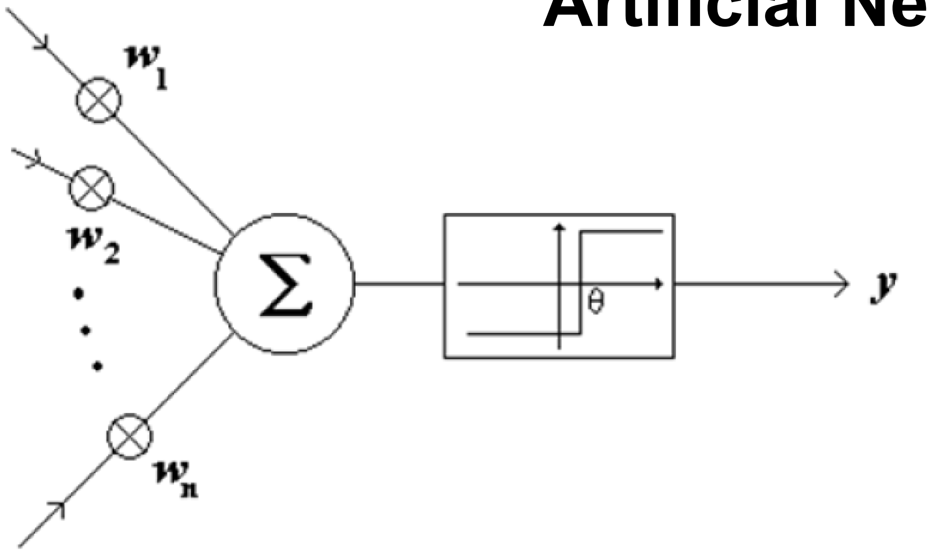


- Analog Leaky Integration → will forget if not constantly "remembered".
- Asynchronous Operation → work on their own time (when "spiked")
- Spiking Communication → no output except for "spikes".
- Energy Efficient Implementation ↳ no energy wasting

?

NCS

Artificial Neuron



- Discrete Summation of Weighted Input
- Centrally Clocked System
- Continuous Values Communication (events → rates)
- Von Neumann Computing Architectures

➡ We transition to a technical abstraction, the *Perceptron*, which can get implemented on computers (time: 1970s)

A one-slide history of Neurons and Neuronal Networks

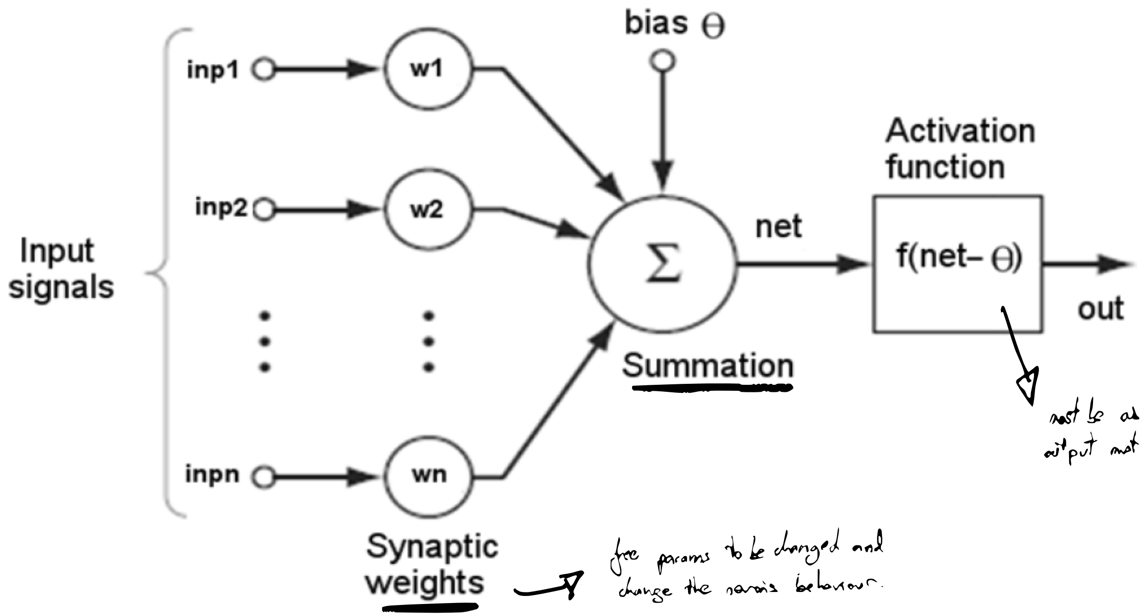
- 1940's McCulloch and Pitts Neuronal Description
  Donald Hebb "The Organization of Behaviour"
- late 1960's Rosenblatt's Perceptron
- 1970's Minsky and Papert's criticism (1969), first *AI winter.*
  Concerns and unsolved obstacles led to lower interest and poor funding

  → non linear settings don't allow computing correct weights
  ↳ what if they can self-learn?
- 1980's Hopfield's impact (self-learning)
- 1990's training through error-backpropagation (1986) → self-adjust params.
- 2000's more mathematically rigorous statistical learning theory
- 2000's more computing power → deep neural networks

Applications today: Pattern recognition: classification and clustering; General interpolation problems; Data representations, coding and compression; Signal processing; Time series prediction; System identification; Decision support (e.g. medical or industrial diagnostics); Memory storage, modelling; Optimization, combinatorial problems

NCS

What can a Single Neuron do?



must be as simple as possible
output must depend on weights.

free params to be changed and
change the neuron's behaviour.

The **net input** of the neuron is given by

$$net = \sum_{i=1}^{n} p_i w_i = p_1 w_1 + p_2 w_2 + \cdots + p_n w_n$$

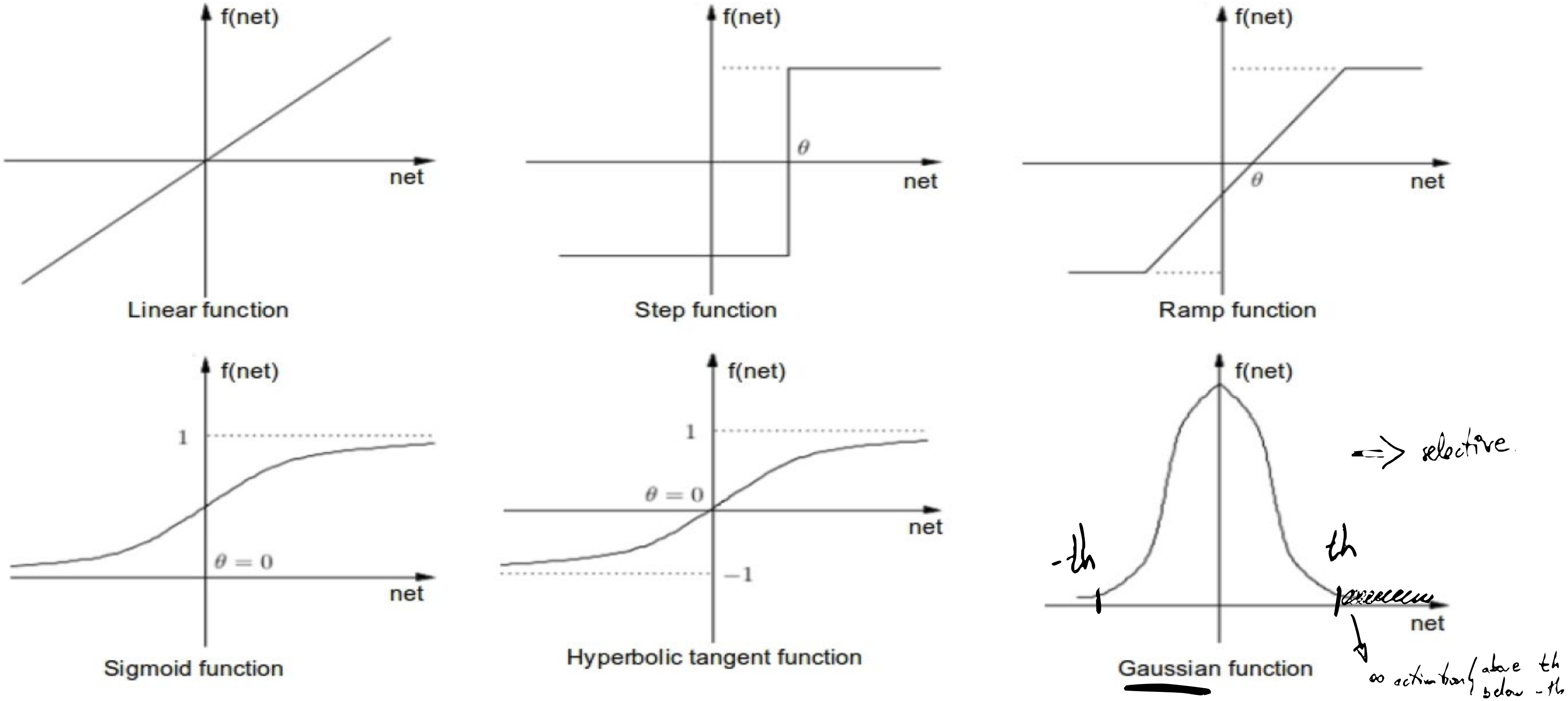
whereas the output of the neuron is computed as

$$out = f(net - \theta)$$

… anything … depending on the activation function $f(\cdot)$

NCS

Typical Activation Functions $f(\cdot)$



By hand-selecting the activation function, we can allow arbitrary functionality.

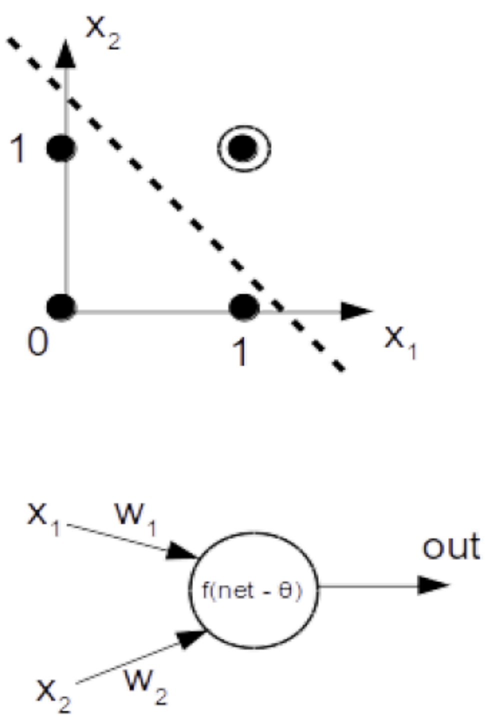
But we can't know a-priory!

Hence, we require a generic system that works independent of activation function.

What can a Single Neuron do?

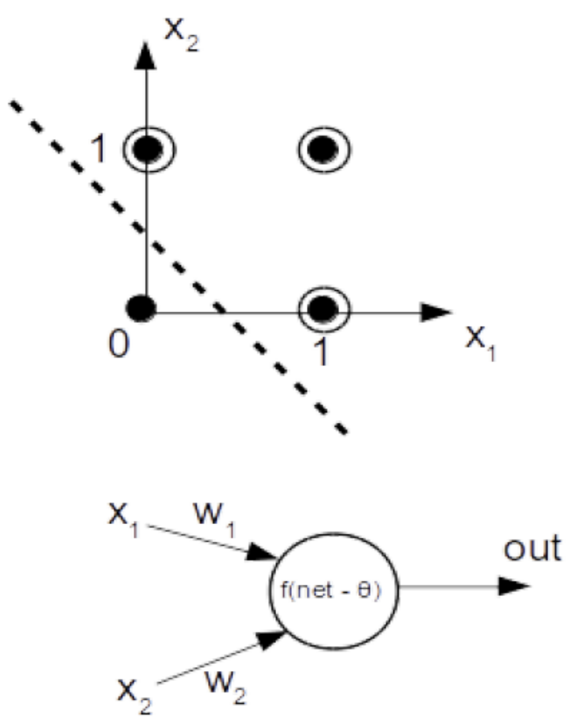
Assume a **LINEAR activation function**. “Boolean Logic” Classification on 2-dimensional input, decision based on out ≥ 0 or <0

AND problem



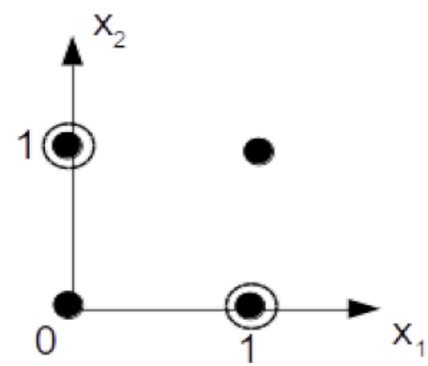
Sample solution:
$w_1 = w_2 = 0.8$
$\theta = 1$

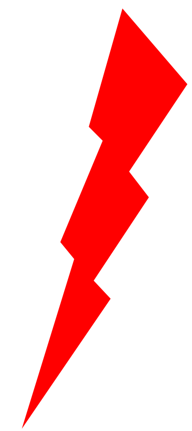
OR problem



Sample solution:
$w_1 = w_2 = 0.8$
$\theta = 0.5$

XOR problem





Sample solution:
The problem is not linearly separable → we need a **multi-layer perceptron**

use different neurons that solve small tasks, and solve the whole problem when combined.

Training Neurons Networks

SUPERVISED LEARNING (labeled training data exists)

Idea: use the error produced by a neuron to adjust its weights

Error metric of a neuron

$$E(t) = \sum_{p=1}^{P_T} \left(t_p(t) - out_p(t)\right)^2$$

t_p target output
$out_p(t)$ current output

Weight update for a neuron *i*

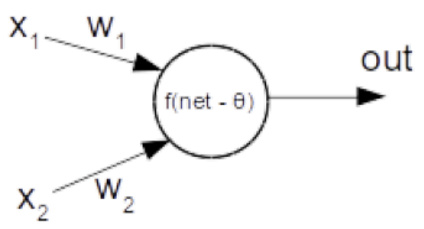
$$w_i(t) = w_i(t-1) + \Delta w_i(t)$$

$$\Delta w_i(t) = \eta \left(\frac{-\partial E(t)}{\partial w_i(t)} \right)$$

This updates all neuronal input weights w_i for a all data point *p*, subject to the current weights.

Training a single neuron, real-world example

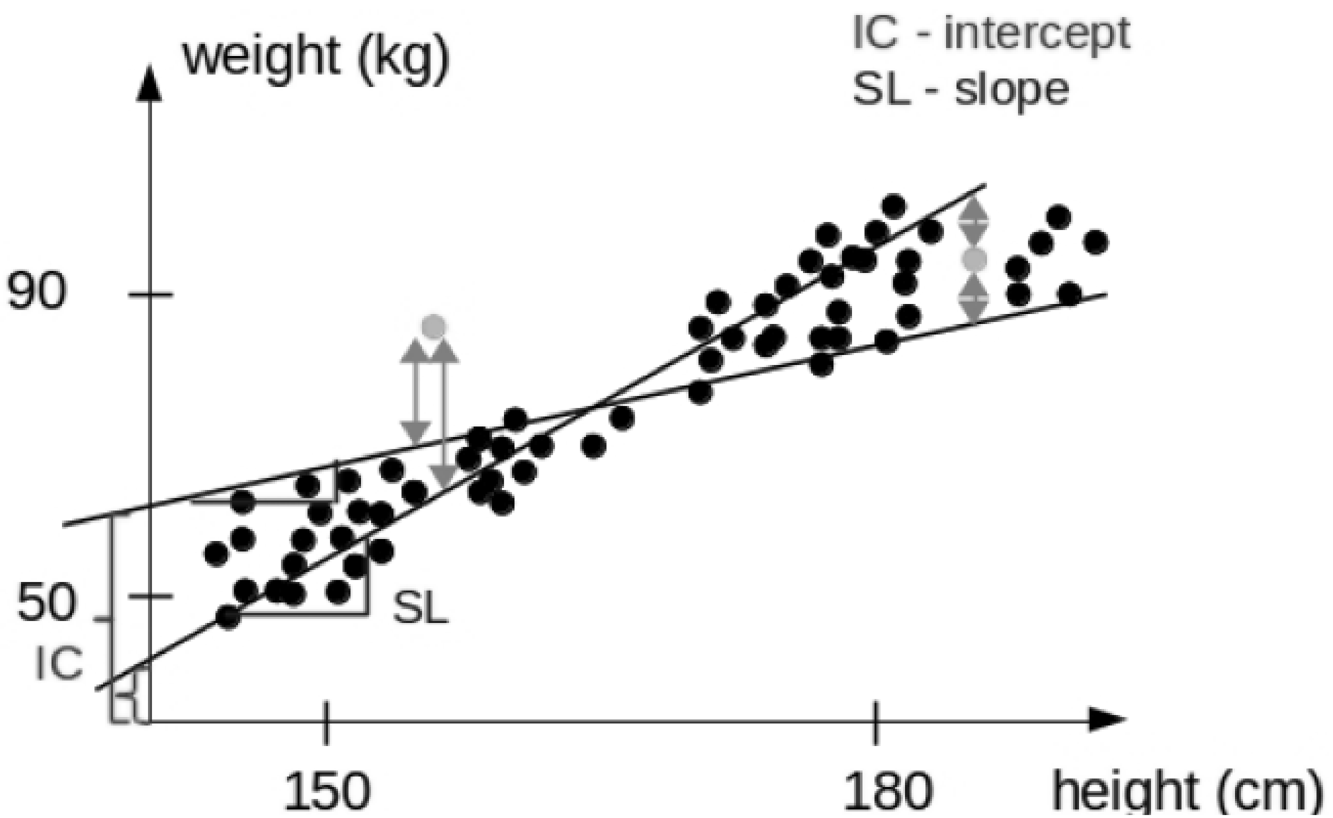
height



weight

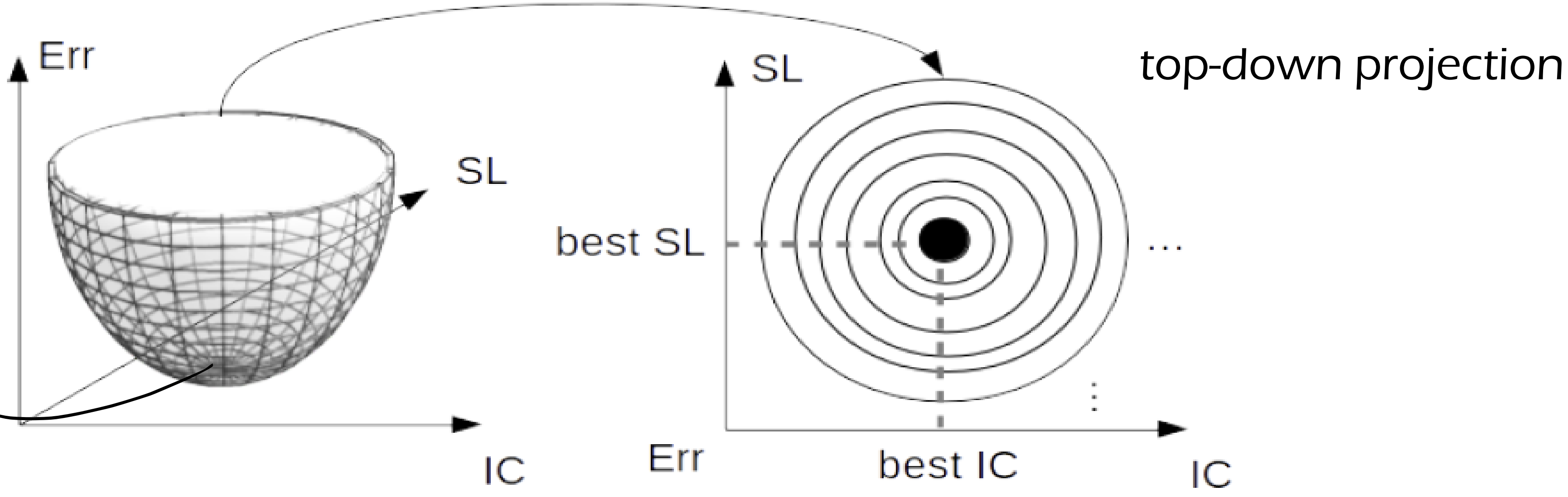
Bias offset
(= const 1)

$$\text{weight} = w1 * \text{height} + w2 * \text{bias}$$
$$= SL * \text{height} + IC$$



Every new data point will adjust SL and IC such that ultimately the best matching linear approximation is encoded in w_1, w_2

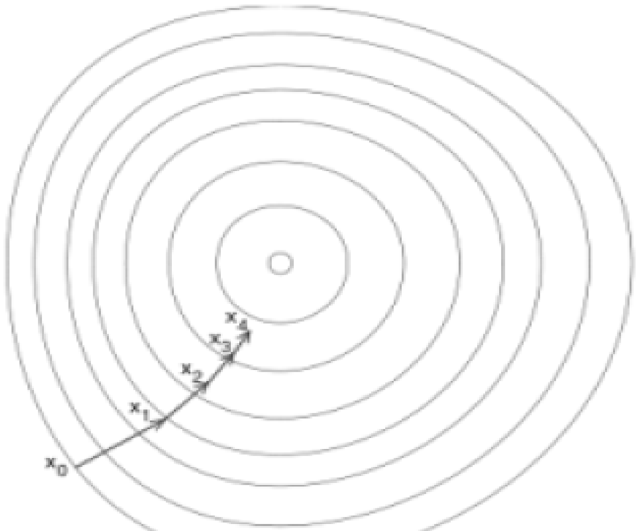
Training a single neuron, real-world example



SL and IC minimizing error.

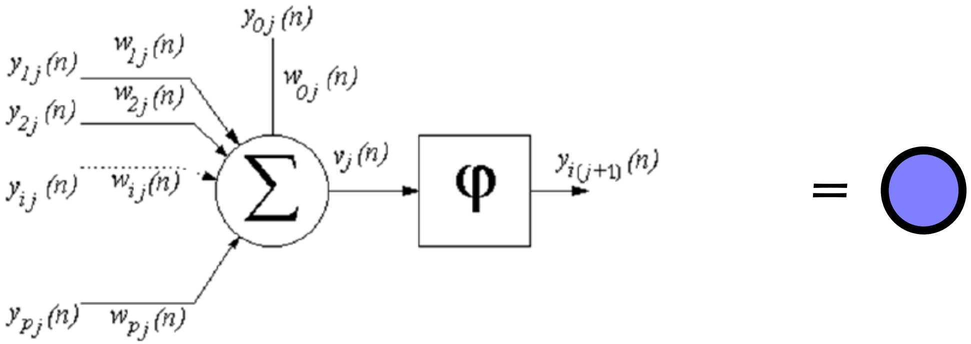
How do we achieve this? In order to guide the search for the suitable parameters Gradient Descent is used:

1. Pick **random initial values** for IC/SL (e.g. x_0)
2. **Calculate the gradient** with respect to each model parameter (i.e. IC,SL)
3. **Update the parameters** in the **direction** of the negative **gradient**
4. **Repeat** 2 and 3 until **convergence** (e.g. $x_{1...4}$)

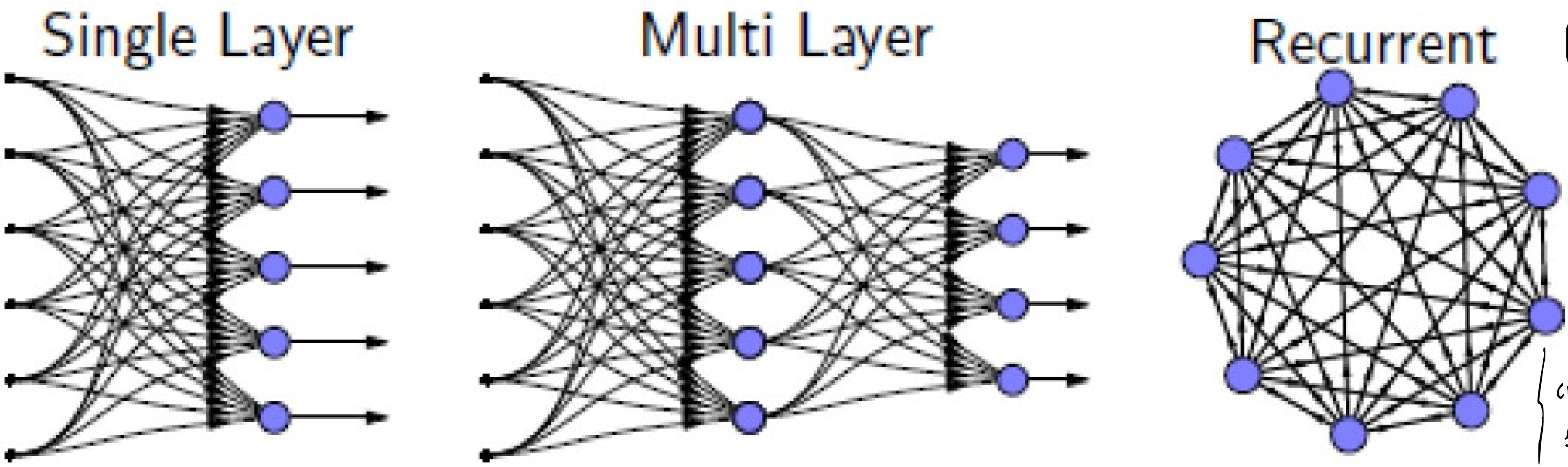


From Neurons to Networks

Single neuron "node"



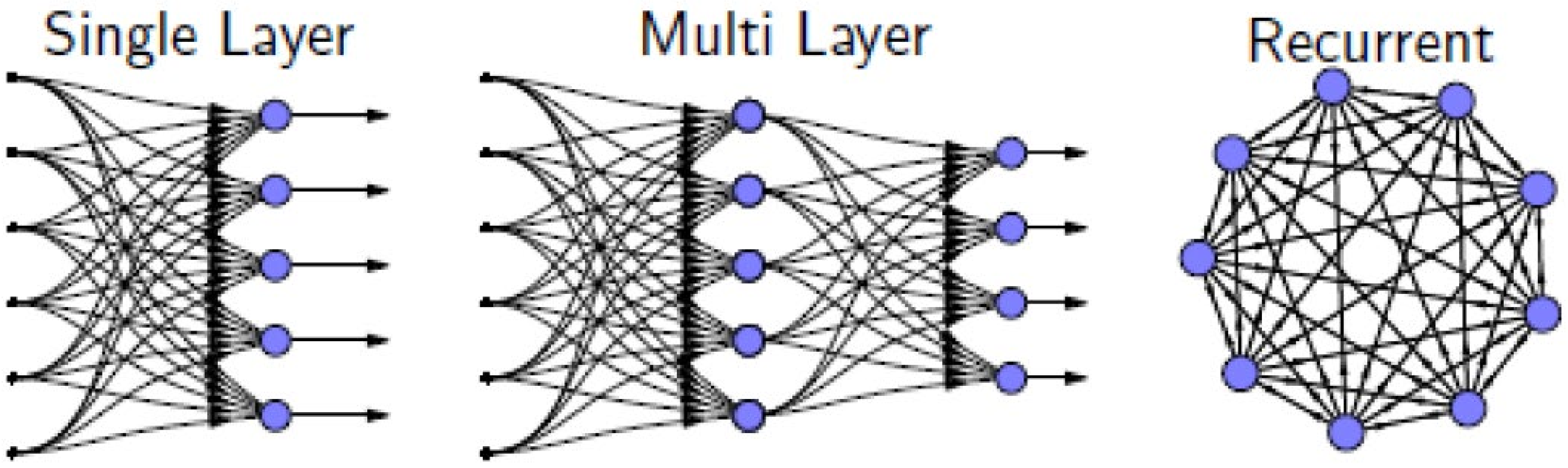
Neuronal Networks



(loop).

critical set-up
store time info !
(remembers past!)

Neuronal Networks



Main Challenges

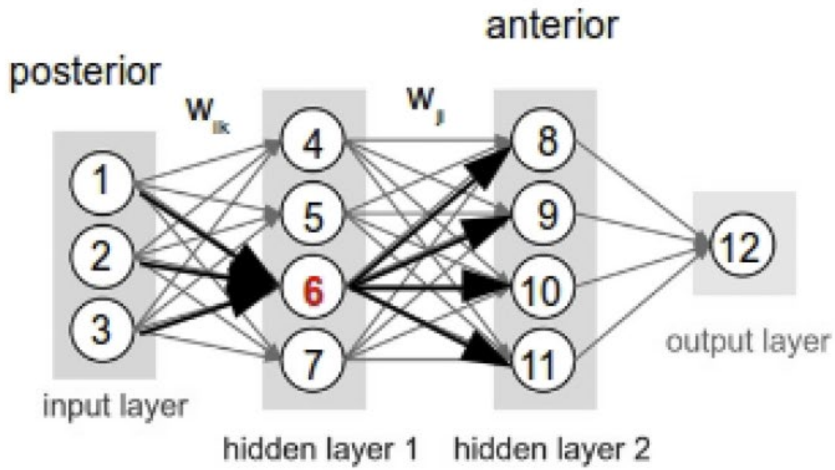
- Which structure to use? → which network? How many layers and neurons/layer?
- How to train (adjust weights) given a particular problem?
  We learned how to do for one neuron, but for networks?
- (How to provide sufficient computing power?)

How to train Neuronal Networks

- We can not simply train neurons in a network, as the "desired output" (t_p) is unknown for neurons in middle layers
- Instead, **ERROR BACKPROGATION** through the network

- last neuron *only* is expected to behave someway (error cannot be computed elsewhere).

- if last neuron produces error, update those neuron's weights that lead to the last neuron output (whole neural path).



Notation:
o - output
j - posterior
i - current
k - anterior

W_{ji} = $W_{\text{<TO><FROM>}}$

$$\Delta w_{ik}(t) = f'\big(net_i(t)\big) \cdot \sum_{j \in P} \delta_j(t) \cdot w_{ji}(t) \cdot f\big(net_k(t)\big)$$

$$\delta_i(t) = f'\big(net_i(t)\big) \cdot \sum\nolimits_{j \in P} \delta_j(t) \cdot w_{ji}(t)$$

Tricks of the Trade – how to get your network working

1. Weight Initialization

   initialize all weights to **small random** numbers

2. Input / Output Normalization

   output: use linear output neuron → (update linearly)

   input: try to normalize to [-1 .. +1] or close to that → all inputs should have the same range!

3. Use Small Weight Update η

   Constant or time dependent $\eta(t)$. → $\uparrow \eta$: risk jumping across right settings.

4. Avoid local minima in weight space

   Simulated Annealing, multiple independent runs

Tricks of the Trade – how to get your network working

5. Network size

   Complete mystery; rules of thumb, fan-in / fan-out (projections)

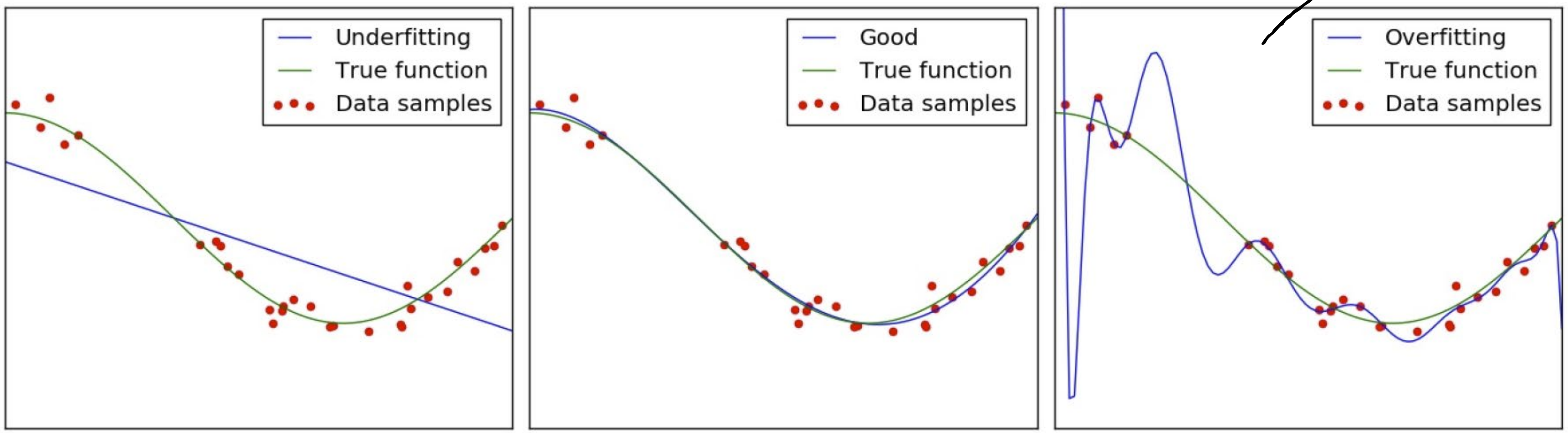
   → no standard way to know how complex our networks should be.

6. Vanishing Gradient

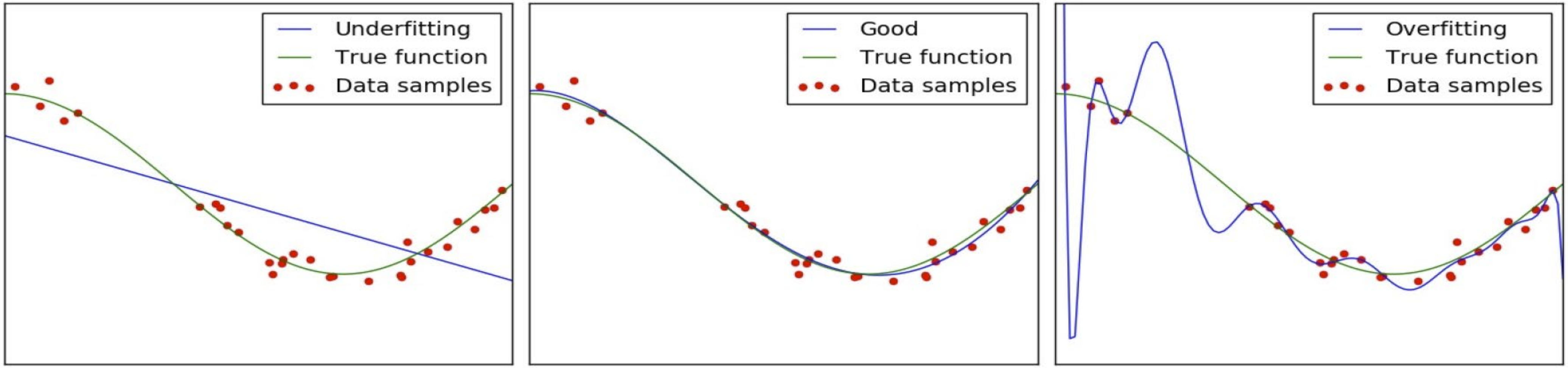
   multiple layers significantly reduce the available gradient

7. Overtraining / Overfitting

more and more neurons will risk generalization!

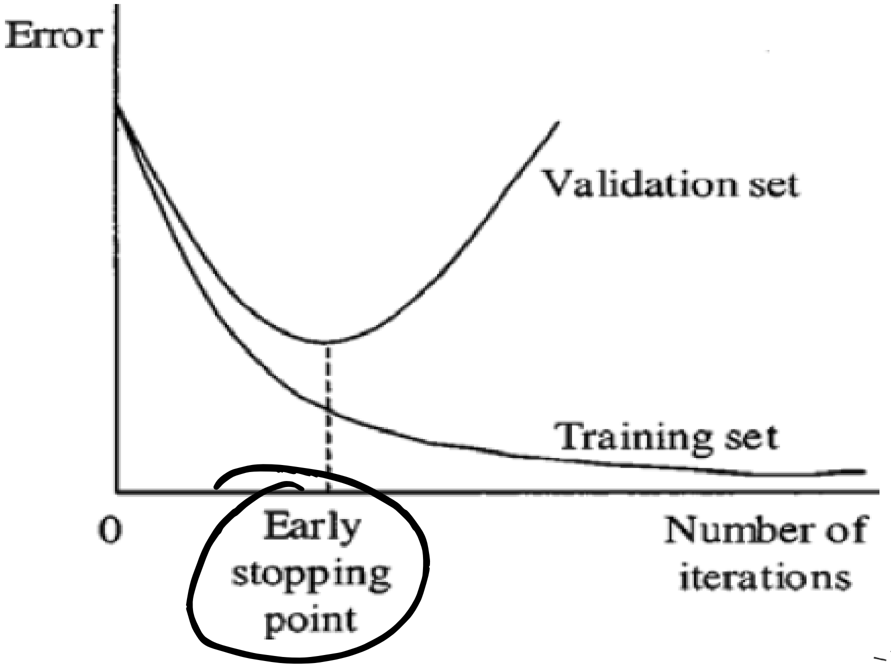


Overtraining / Overfitting



Split available data in two sets
-) 70% Training Data
-) 30% Validation

Train on training data only,
evaluate on Validation data,
STOP training as performance drops

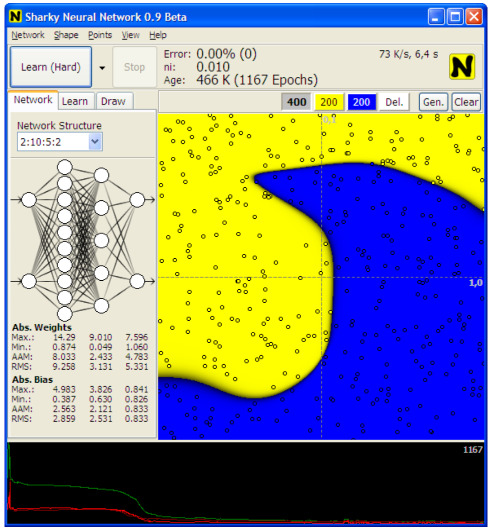


 NCS

Software Example Neuronal Nets

- Google for “Neuronal Network Online Software Simulation” or similar

- http://www.sharktime.com/en_SharkyNeuralNetwork.html



- https://grey.colorado.edu/emergent/index.php/Comparison_of_Neural_Network_Simulators

Other Forms of Neuronal Networks

- Recurrent Neuronal Networks
  - Liquid State Machine → with memory!
- Unsupervised Learning
  - Hebbian Networks
  - Kohonan Self-Organizing Maps
- Deep Neuronal Networks
  - Deeeeeeep

more interest in this topic?

KTH DD2437 – Artificial Neural Networks and Deep Architectures

A Brief Introduction to Our Own Research

NCS

Computation in Brains ... and in Machines

Getting to know your Brain

- 1.3 Kg, about 2% of body weight
- 10^{11} neurons
- Neuron growth
  250.000 / min (early pregnancy)
  … but also loss, 1 neuron/second

"Operating Mode" of Neurons

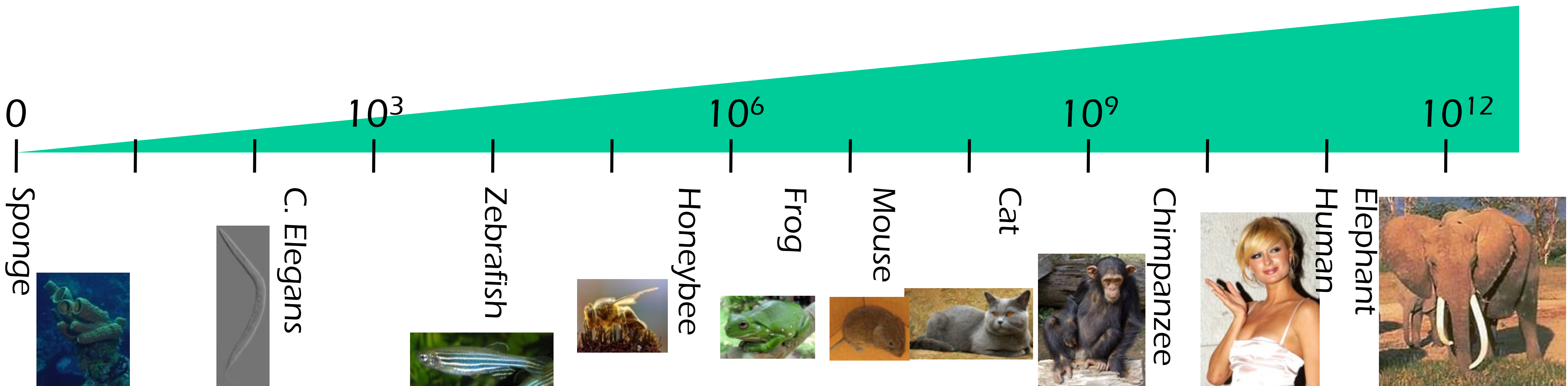
- Analog leaky integration in soma
- Digital pulses (spikes) along neurites
- 10^{14} stochastic synapses
- Typical operating "frequency"
  ≤100Hz, typically ~10Hz, asynchronous

Getting to know your Computer's CPU

- 50g, largely irrelevant
- 10^{10} transistors (SPARC M7, 2015)
- ideally no modification over lifetime

"Operating Mode" of CPUs

- Digital Boolean logic processing
- Digital signal propagation
- Reliable remote storage of data
- Typical operating frequency
  Several GHz, synchronous

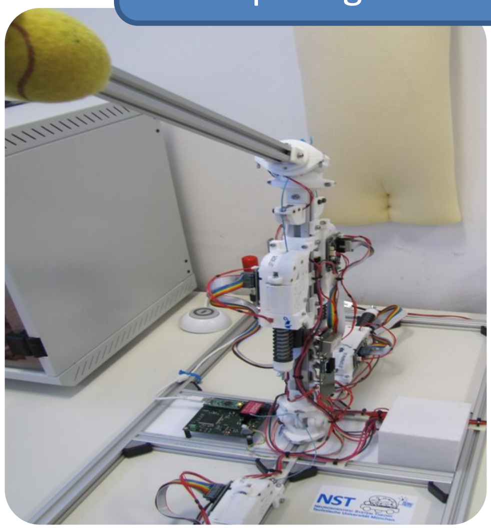


Real-Time Neuronal Information Processing in Closed-Loop Systems

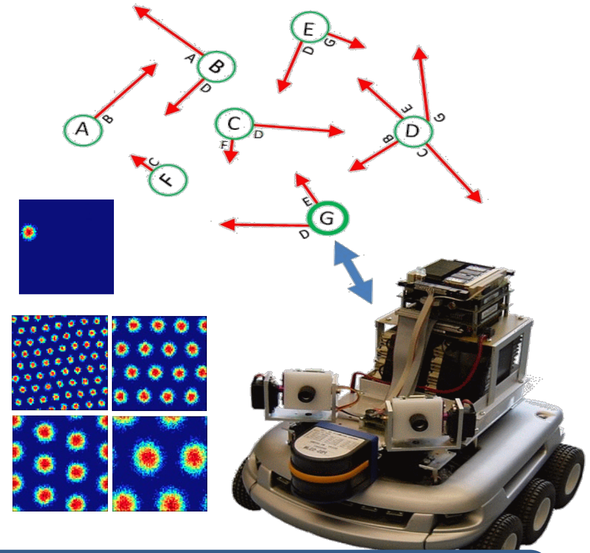
- Event-Based Neuromorphic Vision
- Information Processing in Distributed Neuronal Circuits
- Neuromorphic Real-Time Computing and Control
- Self-Construction and Organization of Neuronal Circuits

Event-Based Neuromorphic Vision

Neuromorphic Real-Time Computing and Control



Information Processing in Distributed Neuronal Circuits



Self-Construction and Organization of Neuronal Circuits

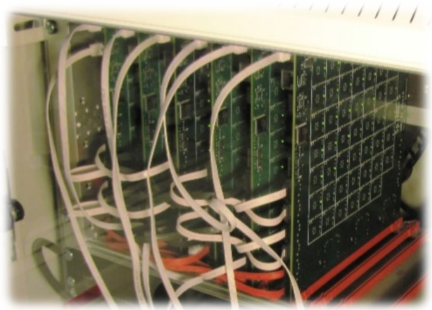
http://neurocomputing.systems

NCS

Neuro Computing Hardware Systems

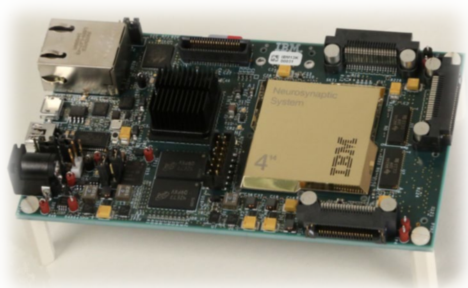
CPU/GPU

- Software Neuronal Net Simulation
- High Energy
- Central Code and Data Memory
- Limited Communication Bandwidth
- Limited Parallelism
- High System Clock



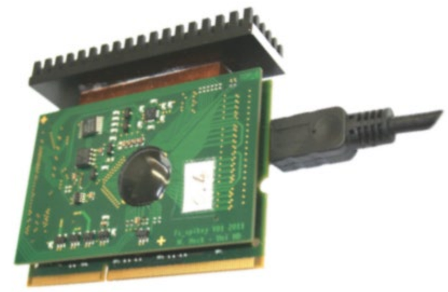
SpiNNaker

- Software Neuronal Net Simulation
- Reduced Energy
- Distributed Code and Data Memory
- Optimized for Spiking Communication
- Massive Parallelism
- Reduced System Clock



TrueNorth

- Neuronal Circuits in Digital Hardware
- Low Energy
- Function given by Synaptic Connectivity
- Limited Local Spiking Communication
- Expandable Parallelism
- Low System Clock



Spikey

- Neuronal Circuits in Analog Hardware
- Minimal Energy
- *Function given by Synaptic Connectivity*
- *(Limited Local Spiking Communication)*
- (Limited Parallelism)
- Real-time Accelerated System Clock