Learning in Neural Networks: Introduction

Wulfram Gerstner
EPFL, Lausanne, Switzerland

No BackProp, please!!!

Objectives for today:

- understand why backprop is problematic
  - → for biology
  - → for hardware
- PCA as a biologically plausible algorithm

- course overview

Reading:

F. Crick, The recent excitement about Neural Networks, Nature 337:129-132 (1989)
T.P. Lillicrap et al., Backpropagation and the brain, Nature Reviews Neurosci. 21: 335-346 (2020)
E. Oja, Simplified Neuron Model as Principal Component Analyser, J. Math. Biol. 15:267-263 (1989)

Previous slide.
After most slides you will find a hidden slide with some annotations and text.
Slides themselves have often graphical elements and keywords, but never text in the form of full sentences.
I optimize my slides for explanations during the lecture; they are not meant as a self-contained ‘textbook’.
The first slide indicates objectives as well as important literature.

Topics today (corresponding to Sections of Lecture)

Intro: No Backprop please !!!!
Intro: From Biological to Artificial Neurons
Intro: From Artificial Neurons to Neuromorphic hardware
Intro: Coarse Brain Anatomy
Lecture 1.1: Synaptic Plasticity: Learning rules of the brain
Lecture 1.2 Hebbian Learning rules
Lecture 1.3 PCA as Hebbian learning

Semester plan

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Content

- Why BackProp is biologically not plausible. Biological two-factor rules and neuromorphic hardware
- Hebbian Learning (two-factor rules) for PCA and ICA
- Two-factor rules for dictionary learning (k-means/competitive learning/winner-takes-all)
- Three-factor rules and neuromodulators (theory and neuroscience)
- Three-factor rules for reward-based learning (theory)
- Three-factor rules for TD reinforcement-learning (algorithmic formulations)
- Actor-critic networks
- Reinforcement learning in the brain
- Learning by surprise and novelty: exploration and changing environments (algorithmic)
- Surprise and novelty in the brain
- Learning representations in multi-layer networks (algorithms without backprop)
- Learning to find a goal: a bio-plausible model with place cells and rewards
- Neuromorphic hardware and in-memory computing

Previous slide.
I discuss the semester plan a bit later.

Learning in Neural Networks: Introduction

Wulfram Gerstner
EPFL, Lausanne, Switzerland

No Backprop, please!!!

Memories

are available because of

Learning

Learning actions:

→ riding a bicycle

Remembering facts

→ previous president of the US

→ first name of your mother

Remembering episodes

→ first day at EPFL

Where is your bicycle today?

Previous slide.

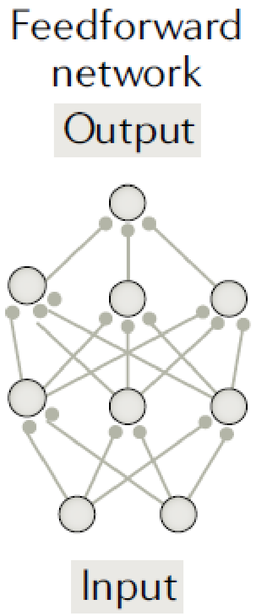
This class is about Learning in Neural Networks, both artificial ones and biological ones.

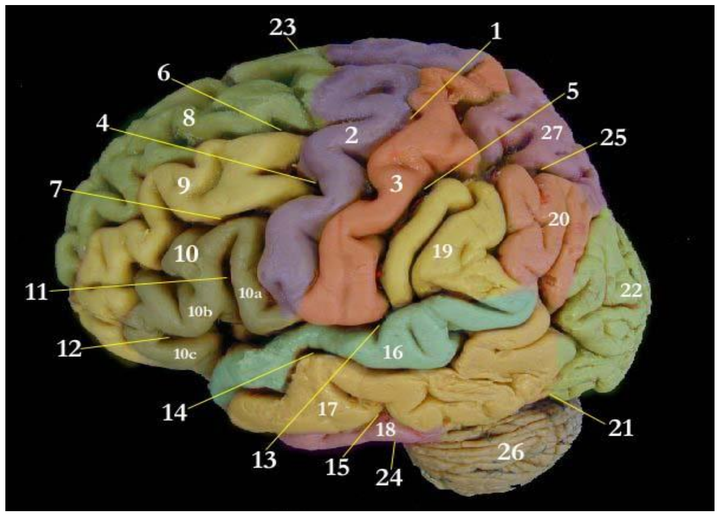
Learning is the basis of acquiring knowledge and building memories.

Learning and memory come in different shades.

All of us have learnt actions.
We all remember some facts.
And we also remember short episodes of our lives.
Some things you remember for a long time. Others not. While you may remember where you have parked your bike today, you will not necessarily remember where you parked in on the Tuesday 11 weeks ago.

Artificial Neural Networks, Inspired by the Brain





History: 3 waves of Neural Networks

- starting 1950
- starting 1981
- starting 2012

Previous slide.
Artificial Neural Networks have always been inspired by the brain.

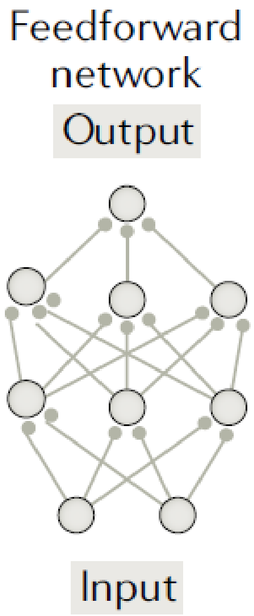
Artificial Neural Networks appeared first in the 1950ies and 60ies, with work of McCullogh&Pitts (binary neurons, 1943), Rosenblatt (Perceptron, 1958), Steinbuch (Lernmatrix, precursor of the Hopfield model, 1961), and then activity diminished (even though it never disappeared).

They appeared again in the 1980ies with the work of Rumelhard and McCleland (book on Parallel Distributed Processing, 1986), Hopfield (1982), Sutton and Barto (reward-based learning, 1981), Ackley-Hinton-Sejnowski (Boltzman machines, 1985), the first NeurIPS conference (called NIPS at the time), and the re-discovery of BackProp (D.E. Rumelhard et al., Nature, 1996)
Then the wave stopped when researchers pursued Support Vector Machines or Classic AI.

And finally, as you know, a third wave started under the name of Deep Learning around 2014. Again, attention modules and many other aspects are brain-inspired.

Learning Artificial Neural Networks

Learning = change of parameters
Parameters =connections (mainly)



Deep Networks for Vision
(e.g. AlexNet …)

Sutskever and Hinton, 2012

→ trained with BackProp

Deep Networks for Chess and Go
(alpha-go, alpha-zero)

Silver et al. (2017) , Deep Mind

→ trained with BackProp

Foundation Models
(LLMs, ChatGPT, Bert)

→ trained with BackProp

Previous slide.

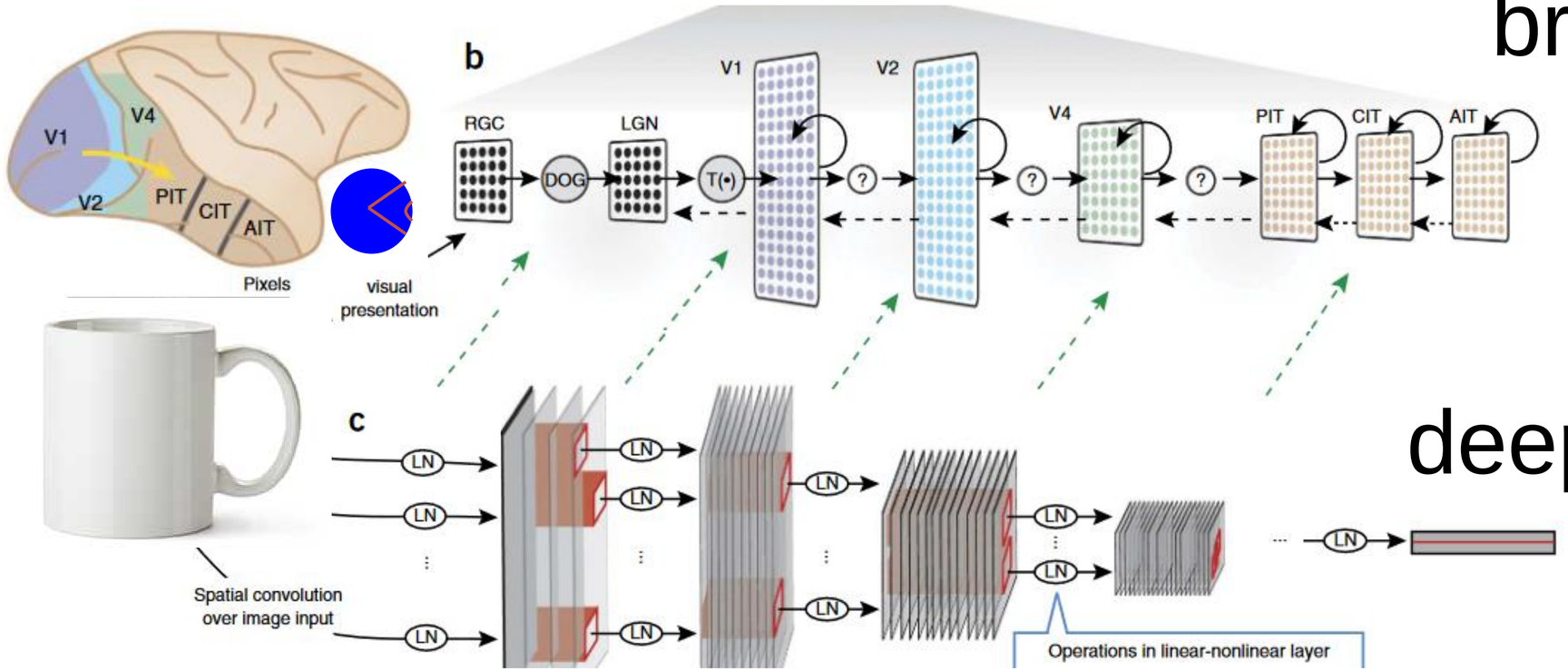
All impactful models have been trained with BackProp. BackProp (combined with well-known optimization tricks) is a powerful algorithm and can be adapted to all sorts of network architecture.
(If you do not recall BackProp details, take any textbook. I also added slides as an appendix to the stack of slides of this first lecture.)

In this class, I will argue that the human brain does not use backprop – but is has powerful learning capacities as well. And it uses much less energy than modern AI.

Let us therefore try to explore alternative learning algorithms that could eventually (and we are not yet there!) to low-power implementations of learning in neural networks.

BackProp useful to optimize brain models

brain model



deep network

- Train on classification of Mio of images with BackProp
- 1 Cortical Area → 1 layer of a Deep Convolutional Neural Network

BUT: - Real networks do not use BackProp

- Real networks are not trained with labeled images!

Using goal-driven deep learning to understand sensory cortex
D Yamins, JH DiCarlo Nat. Neurosci.19: 356-365 (2016)

Previous slide.

And even some of the best models to explain data in the neurosciences have been trained with BackProp!

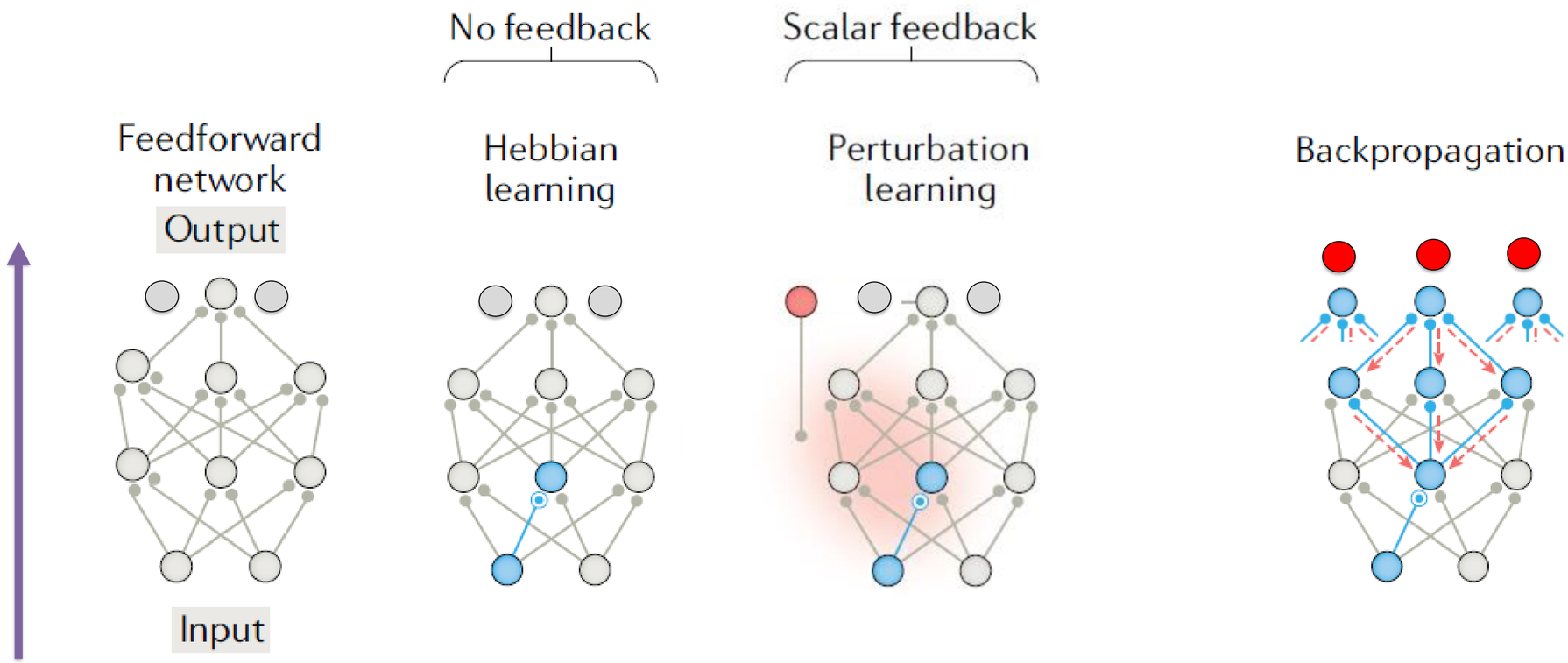
The slide shows deep convolutational neural network model and its equivalence of processing steps in the brain, following the visual pathway.

The model is trained with BackProp on an image classification task with Millions of images!

But the training with BackProp does not describe biological learning.

A Spectrum of Learning Algorithms: connections change based on …

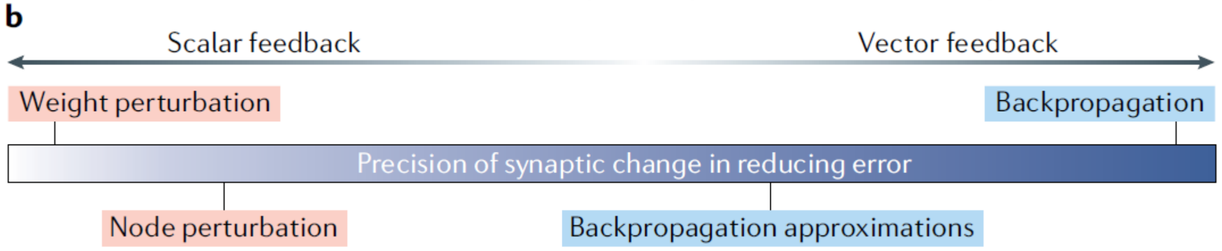
Adapted from Lillicrap et al. 2020 Nat. Rev. Neurosci.



2-factor rules: unsupervised, no feedback

3-factor rules: reward-based, feedback

BackProp rules: vector feedback (many local ‘errors’)



Previous slide.

We study various types of neural networks in this class. Many of these are feedforward networks. I mostly use a convention where input comes in from the bottom and moves towards the top layer.
(rarely also from left to right).

Learning in Artificial Neural Networks (ANNs) is related to changes in the connection strength. In a biological context, the connection point is called a synapse. Changes in connection strength are called synaptic plasticity.

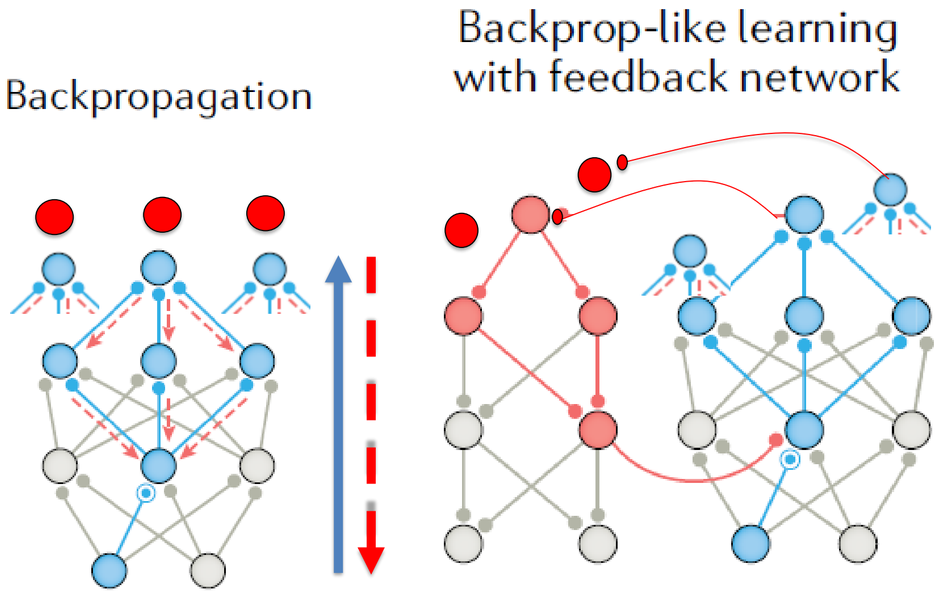
The question then is whether changes induced by a learning algorithm depend only on information of the two connected neurons (left: no feedback), or also on additional information.

Additional information could be a global, scalar feedback (the performance was good = reward or bad = no reward); or very detailed information in the form of vectorized feedback (right).

Backprop needs precise error feedback

Vector feedback:
- multiple outputs,
- one ‘signed error per output’
- error vector transmitted back
- precise neuron-specific errors

BackProp Algo has 4 phases:
1) Forward pass and freeze
2) Calculate local output errors
3) Backprop pass, using 2)
4) Update the weights, using 1) +3)



BackProp rules: vector feedback

Adapted from Lillicrap et al. 2020 Nat. Rev. Neurosci.

Previous slide.
In a network with multiple outputs, standard backprop calculates local signed mismatches for each output unit, and then transfers this information back.

For example consider self-supervised learning and auto-encoders applied to images with D pixels. Each pixel value is one input=one output unit. ‘error feedback vector δ’ has therefore D components.

Vector feedback is very different from scalar feedback!

Let us recall the BackProp algorithm:

1) in the forward phase, and input vector is applied and passed through the network. Activities of each neuron need to be stored (or ‘frozen’)
2) the outputs are compared with the target values, which gives the vector of mismatch values δ also called signed local errors.
3) the mismatch values are propagated backward. This feedback uses the same weights as the feedforward network, but different nonlinearities.
4) Weights are updated using the frozen activities and the local δ values

No BackProp please (in this class)

- BackProp needs four separate phases:

  forward pass, output mismatch, backward pass, weight update.

- Backward pass needs specific feedback architecture

  (e.g., all linear; feedback weights = feedforward weights;
  backward multipliers proportional to activity of feedforward network).

- **The feedback architecture must enable vector feedback**

→ Not implementable in biology
→ Difficult to implement in low-energy neuromorphic hardware

F. Crick, The recent excitement about Neural Networks, Nature 337:129-132 (1989)
T.P. Lillicrap et al., Backpropagation and the brain, Nature Reviews Neurosci. 21: 335-346 (2020)

Previous slide.

In a network with multiple outputs, standard backprop calculates local signed mismatches for each output unit, and then transfers this information back.

If you think of a physical network (be it in artificial or biological hardware), this would require a separate feedback network.

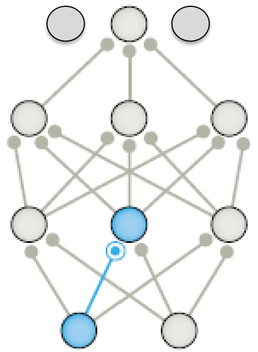
This is the main reason that BackProp is biologically not plausible; for the same reason it is also not attractive for energy-efficient alternative computing paradigms, sometimes called ‘in-memory computing’, ‘neuromorphic hardware’, or ‘non-von-Neumann architecture’.

Essentially, we aim for algorithms that where each unit only uses locally available information to change parameters. You can think of such algorithms as asynchronous fully distributed algorithms.
And that type of algo is what we will study in the class. We will use mathematical language, but take inspiration from the brain.

2-factor rules use information locally available at the synapse

No feedback

Hebbian learning



2-factor rules:
unsupervised,
no feedback

Big question:
Can we learn anything at all without feedback?

→ first part of class:
2-factor rules

“Each connection is a (conditionally) independent actor and uses only locally available information for changes.”

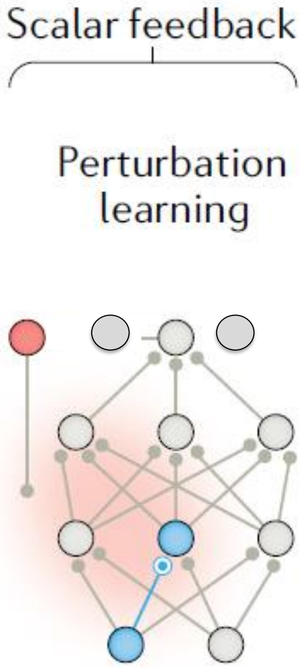
Previous slide.
Most scientists believe that biological neural networks (your brains!) do not have such a specific detailed error feedback network.

In the absence of such a detailed error feedback network, can we learn anything at all?

Learning in neural networks means changing the connection. Think of each connection as an independent actor that only has access to locally available information. What can we achieve?

In the first part of the class we focus on this question and study a few example algorithms.

3-factor rules use information locally available at the synapse combined with one global feedback signal



3-factor rules:
reward-based,
feedback

Big question:
What is the relation to **reinforcement learning**?
What can be learned with these rules?

→ second part of class:
3-factor rules

Previous slide.

But maybe the assumption that there is no feedback at all is too strict. Indeed, you know the feeling of happiness if you succeeded to do something you want to do. Success causes in the brain an internal reward signal.

Such a reward signal conveys a SCALAR information on how good or bad your performance was, but not what you should do to improve the performance.

Sounds a bit like Reinforcement Learning, but is there a precise connection?

This is what we will study in the second part of the class.

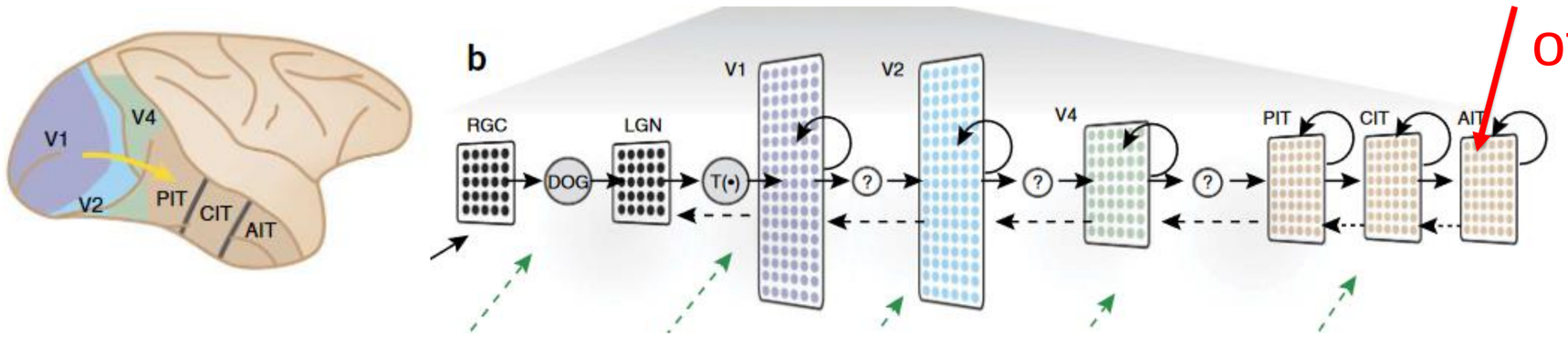
multi-factor rules that use locally available information combined with several global feedback signals

Learning rules???

Network architecture???

What kind of feedback???

Big question:
Can we have local learning rules (with several global signals) that yield **good representations in multi-layer networks**?



Previous slide.

Even reward-based learning methods (using scalar feedback) are somewhat limited.

Therefore we ask: can we generalize the idea of ‘local learning rule’ with a few global (or near-global) signal?

A critical test is to learn representations in multi-layer networks.

Summary: Learning in Neural Networks (Introduction)

Backprop has several problems as model for neuroscience

- 4 phases for update → online, continuous time
- precise feedback architecture → robust, plausible feedback
- forward=backward weights → Learning rules for all weights

No BackProp, please!!!

Active research area in computational neuroscience

Also relevant for low-energy neuromorphic computing

Reading:

F. Crick, The recent excitement about Neural Networks, Nature 337:129-132 (1989)

T.P. Lillicrap et al., Backpropagation and the brain, Nature Reviews Neurosci. 21: 335-346 (2020)

Semester plan

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Content

3 or 4 weeks

- Why BackProp is biologically not plausible. Biological two-factor rules and neuromorphic hardware
- Hebbian Learning (two-factor rules) for PCA and ICA
- Two-factor rules for dictionary learning (k-means/competitive learning/winner-takes-all)

3 or 4 weeks

- Three-factor rules and neuromodulators (theory and neuroscience)
- Three-factor rules for reward-based learning (theory)
- Three-factor rules for TD reinforcement-learning (algorithmic formulations)

miniproject handout

flexible topics

- Actor-critic networks
- Reinforcement learning in the brain
- Learning by surprise and novelty: exploration and changing environments (algorithmic)
- Surprise and novelty in the brain
- Learning representations in multi-layer networks (algorithms without backprop)
- Learning to find a goal: a bio-plausible model with place cells and rewards
- Neuromorphic hardware and in-memory computing

Assessment methods

Oral exam (70 percent) plus miniproject (30 percent).
If more than 45 students participate, the oral exam is replaced by a written exam.

Oral exam (27 min):

- Presentation of an (important) research paper related to class.
- Followed by questions to paper and to lectures.
- Sample session during last 2 weeks (TA=role of student)

Questions?

For those who are not available on Tuesday 2pm-4pm,
we offer an alternative exercise sessions Wednesday, 4pm or 5pm

Previous slide.

The first 7 weeks are a very systematic introduction to
- Representation learning with two-factor rules
- Reinforcement learning with three-factor rule

Then we hand out the miniprojects. You can choose one of two projects:
(i) receptive field learning with two-factor rules
(ii) learning to navigate in a maze with three-factor rules

I will handle the last weeks a bit more flexibly in terms of topics.

The course finishes with an oral exam (unless the number of students is above 45).

Learning in Neural Networks: Introduction

From Biological to Artificial Neurons

Wulfram Gerstner

EPFL, Lausanne, Switzerland

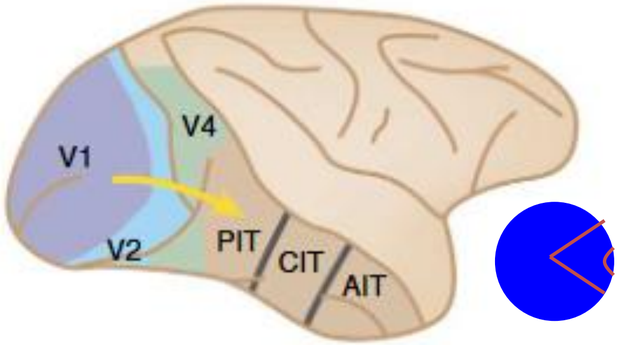
From Biological to Artificial Neurons

Previous slide.

In this first introduction lecture the focus is on a general introduction into the field (with its subparts Reinforcement learning and Supervised Learning for Classification).

We start with a glimpse of the biological inspirations of the field.

The brain: Cortical Areas

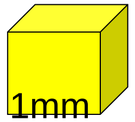


Previous slide.
Cortex is divided into different areas:
Information from the eye will first arrive at visual cortex (at the back of the head), and from there it goes on to other areas. Comparison of the input with memory is thought to happen in the frontal area (above the eyes). Movements of the arms a re controlled by motor cortex somewhere above your ears.

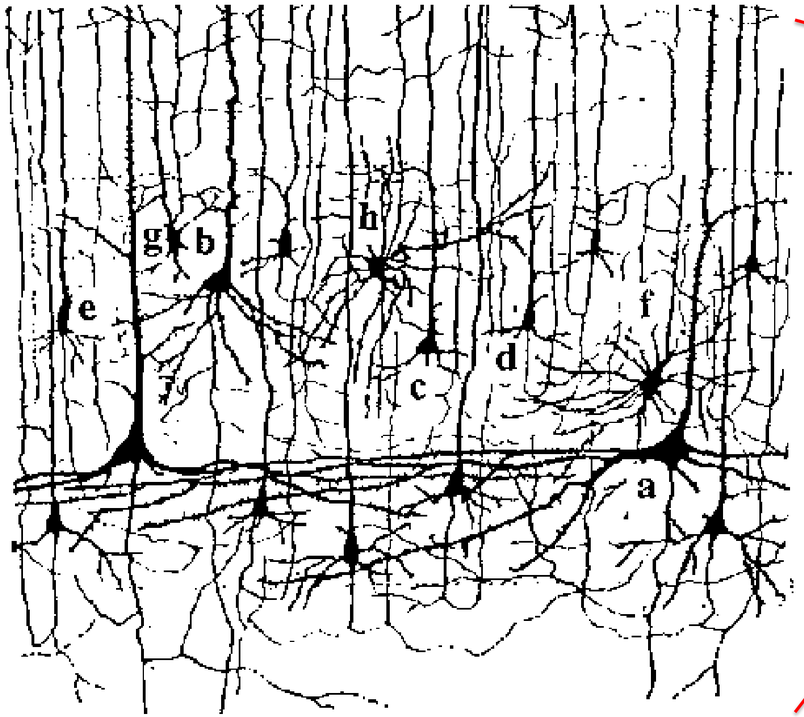
Talking about cortical areas provides a **macroscopic** view of the brain.

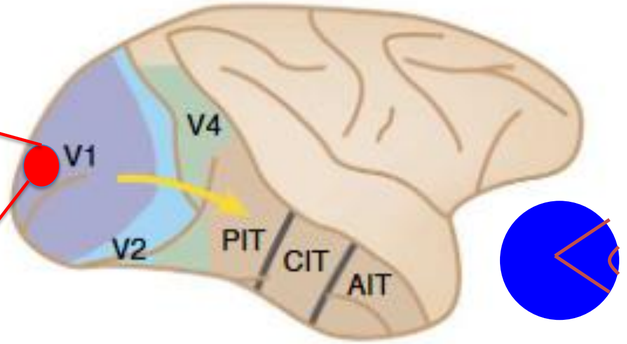
The Brain: zooming in

1mm



10 000 neurons
3 km of wire





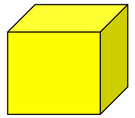
Ramon y Cajal

Previous slide.
If we zoom in and look at one cubic millimeter of cortical material under the microscope, we see a network of cells.
Each cell has long wire-like extensions.
If we counted all the cells in one cubic millimeter, we would get numbers in the range of ten thousand.
Researchers have estimated that, if you put all the wires you find in one cubic millimeter together you would find several kilometers of wire.

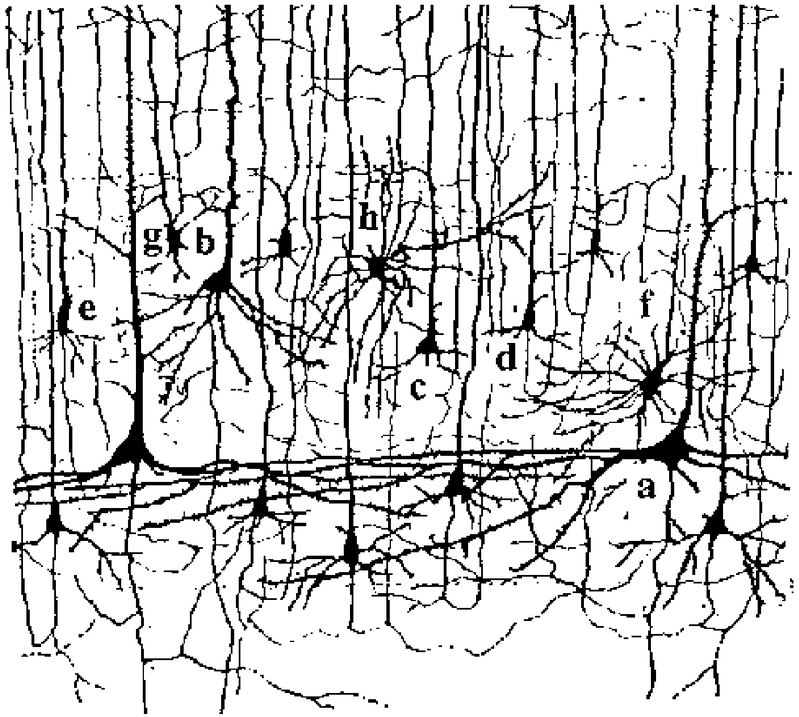
Thus, the neural network of the brain is a densely connected and densely packed network of cells.

The brain: a network of neurons

1mm

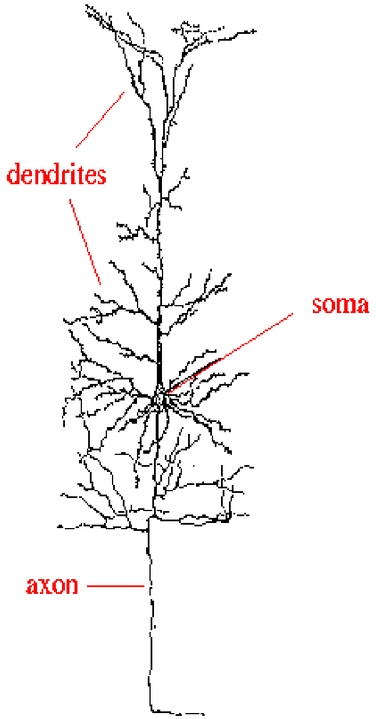


10 000 neurons
3km of wire



Ramon y Cajal

Signal:
Action potential (short pulse)

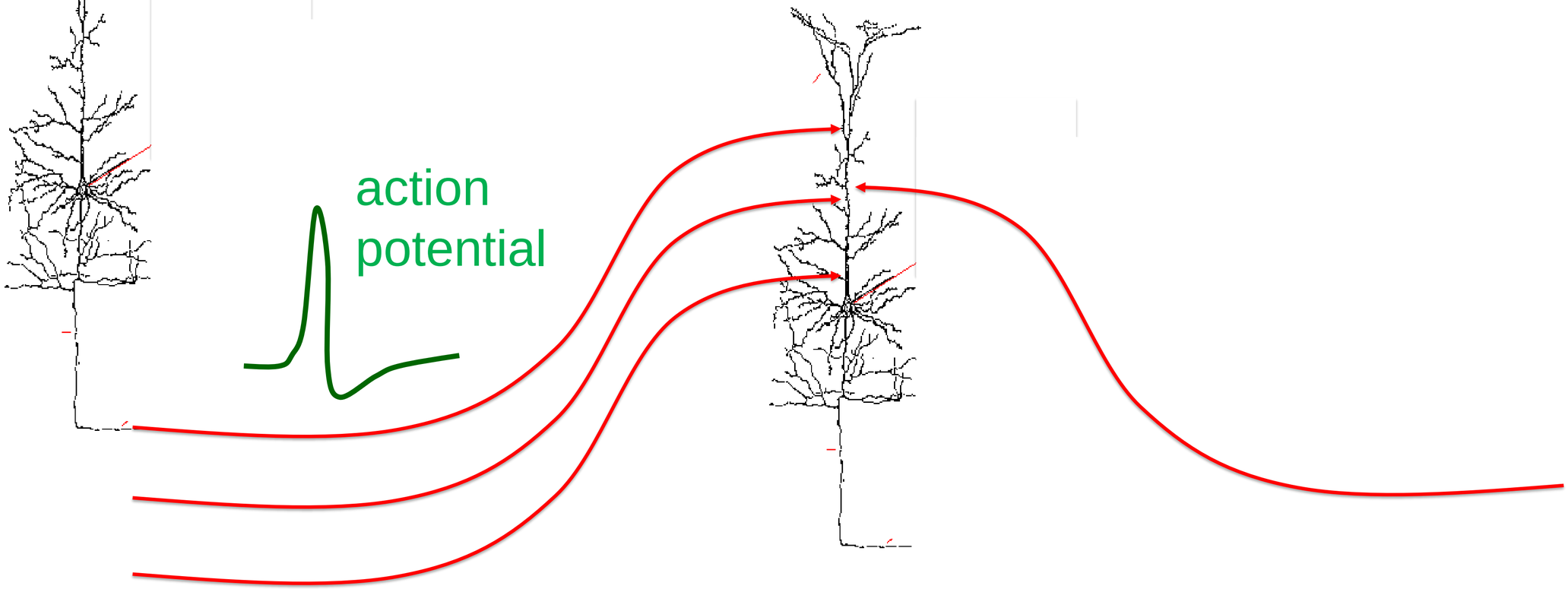


Previous slide.
These cells are called neurons.
The part of the neuron where signals arrive is called the dendrite, or the dendritic tree. The cables that transmit the signal to other neurons is called the axon. The central part of the cell is called the soma.

What is the signal? Neurons communicate with each other by short electrical pulses, called action potentials, or ‘spikes’.

The brain: signal transmission

Signal:
action potential (short pulse)

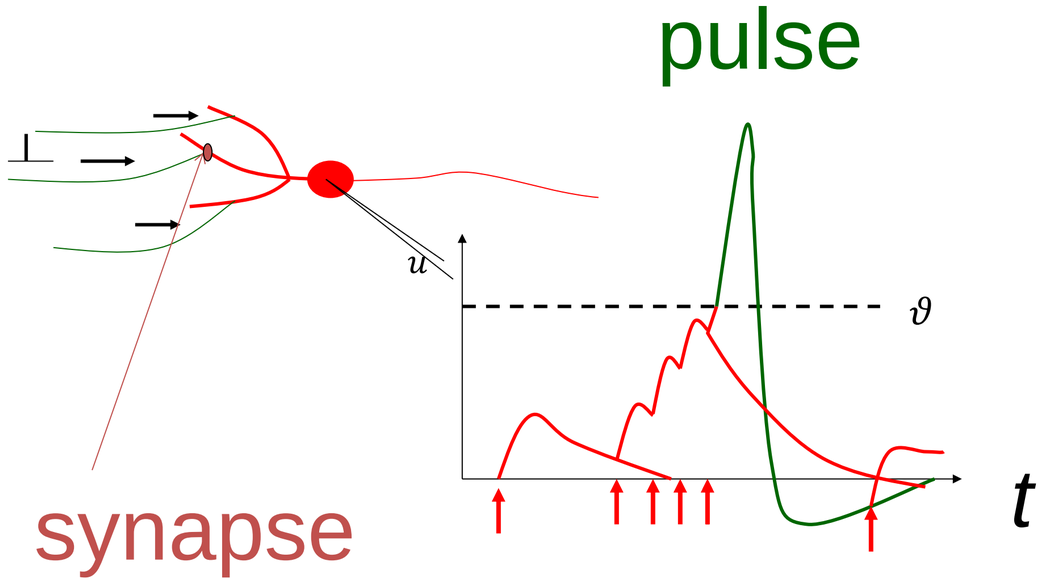


More than 1000 inputs

Previous slide.
Signals are transmitted along the wires (axons). These wires branch out to make contacts with many other neurons.

Each neuron in cortex receives several thousands of wires from other neurons that end in ‘synapses’ (contact points) on the dendritic tree.

The brain: neurons sum their inputs

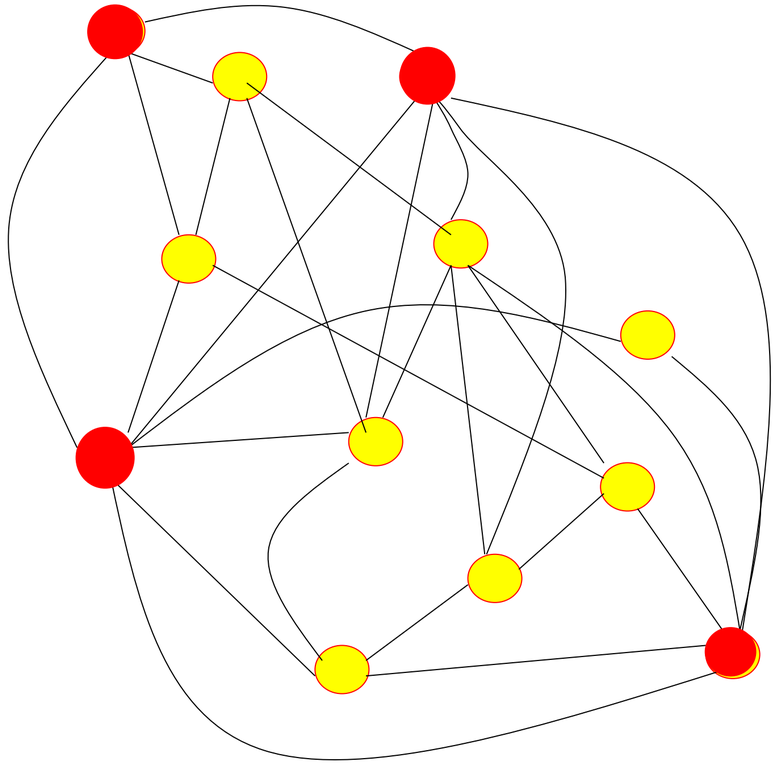


Previous slide.
If a spike arrives at one of the synapses, it causes a measurable response in the receiving neuron.

If several spikes arrive shortly after each other onto the same receiving neuron, the responses add up.

If the summed response reaches a threshold value, this neuron in turn sends out a spike to yet other neurons (and sometimes back to the neurons from which it received a spike).

Summary: the brain is a large recurrent network of neurons



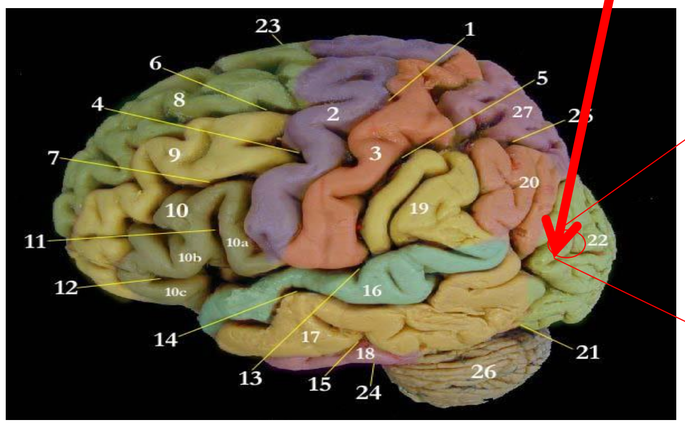
- Active neuron

Previous slide.

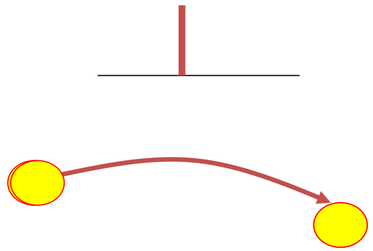
Thus, signals travel along the connections in a densely connected network of neurons.

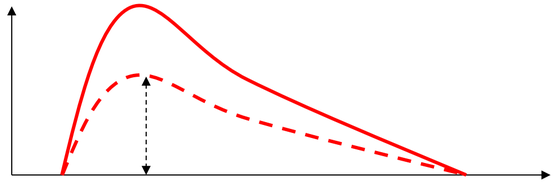
Sometimes I draw an active neuron (that is a neuron that currently sends out a spike) with a filled red circle, and an inactive one with a filled yellow circle.

Learning in the brain: changes between connections

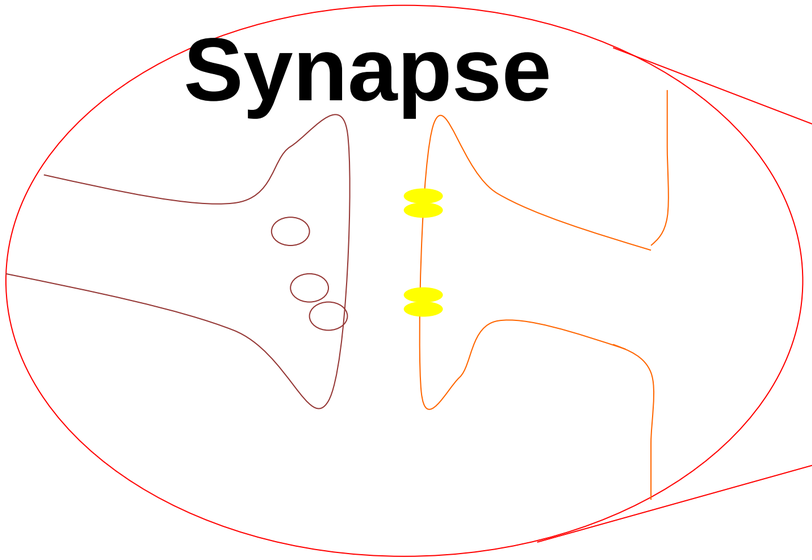


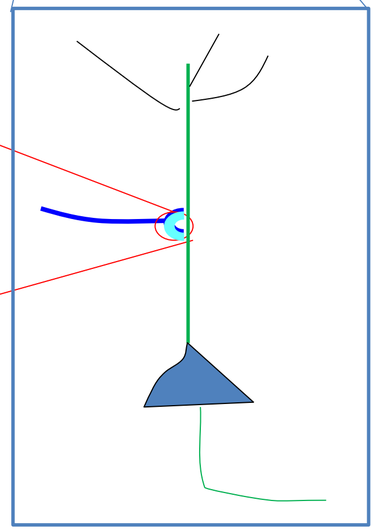
Neurons





Synapse





learning = change of connection

Previous slide.

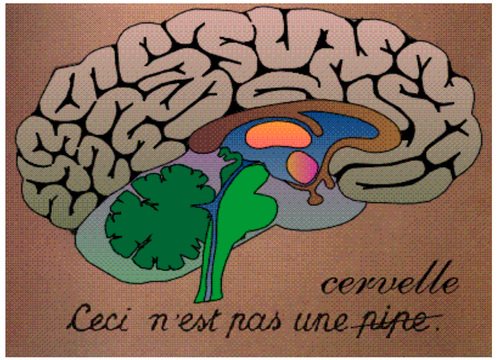
Synapses are not jut simple contact points between neurons, but they are crucial for learning.

Any change in the behavior of an animal (or a human, or an artificial neural network) is thought to be linked to a change in one or several synapses.

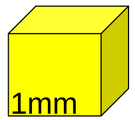
Synapses have a ‘weight’. Spike arrival at a synapse with a large weight causes a strong response; while the same spike arriving at a synapses with a small weight would cause a low-amplitude response.

All Learning corresponds to a change of synaptic weights. For example, forming new memories corresponds to a change of weights. Learning new skills such as table tennis corresponds to a change of weights.

Neurons and Synapses form a big network



Brain

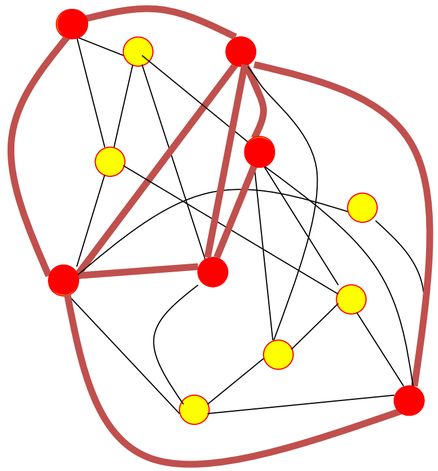


1mm

10 000 neurons
3 km of wire

10 billions neurons

10 000 connexions/neurons



Distributed Architecture

memory in the connections

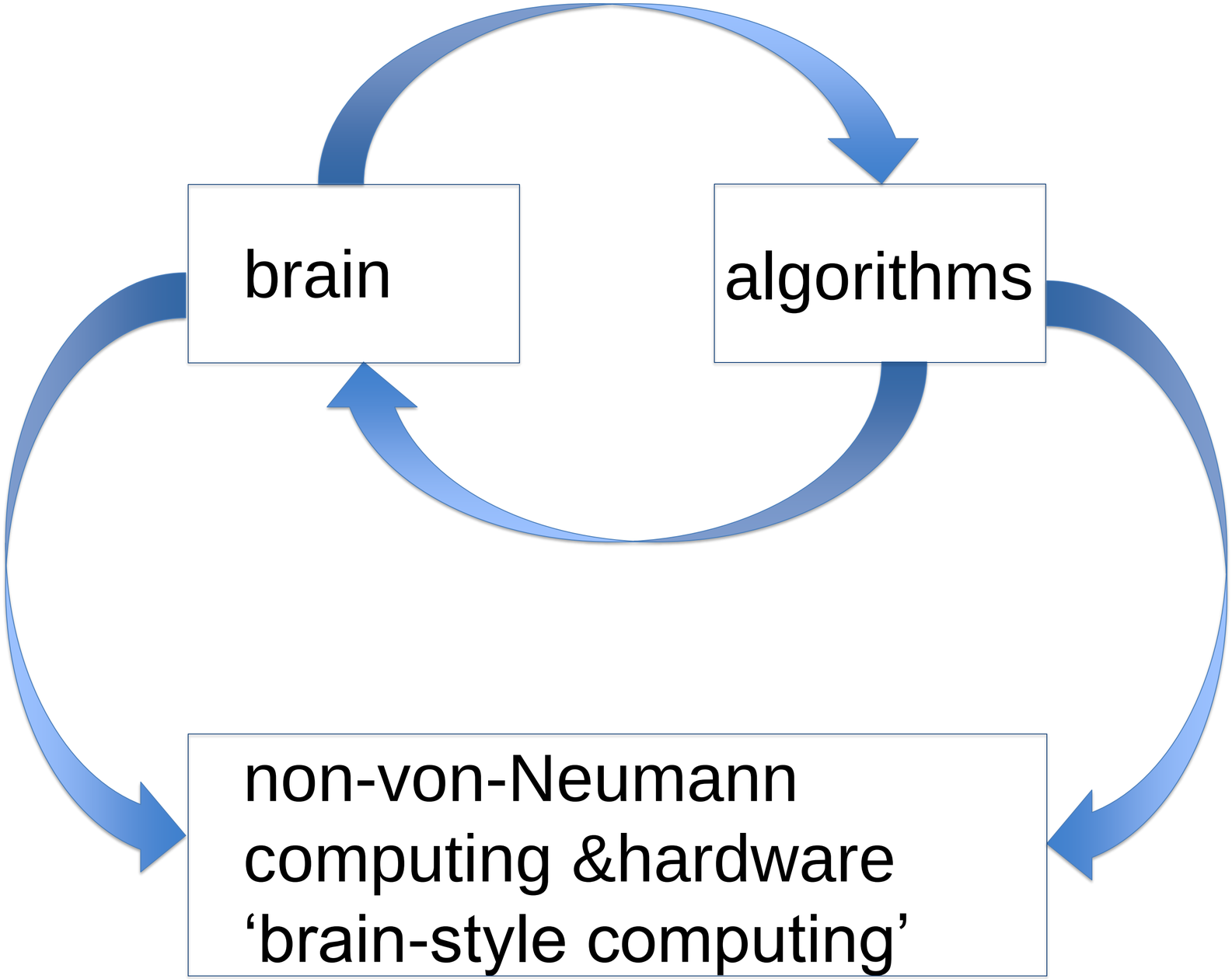
No separation of processing and memory

Previous slide.
Even though we are not going to work with the Hebb rule during this class, the above example still shows that

- Memory is located in the connections
- Memory is largely distributed
- Memory is not separated from processing

(as opposed to classical computing architectures such as the van Neumann architecture or the Turing machine)

Learning Rules in Neural Networks

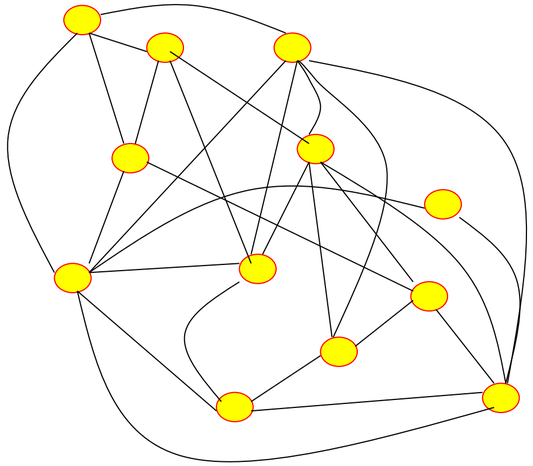


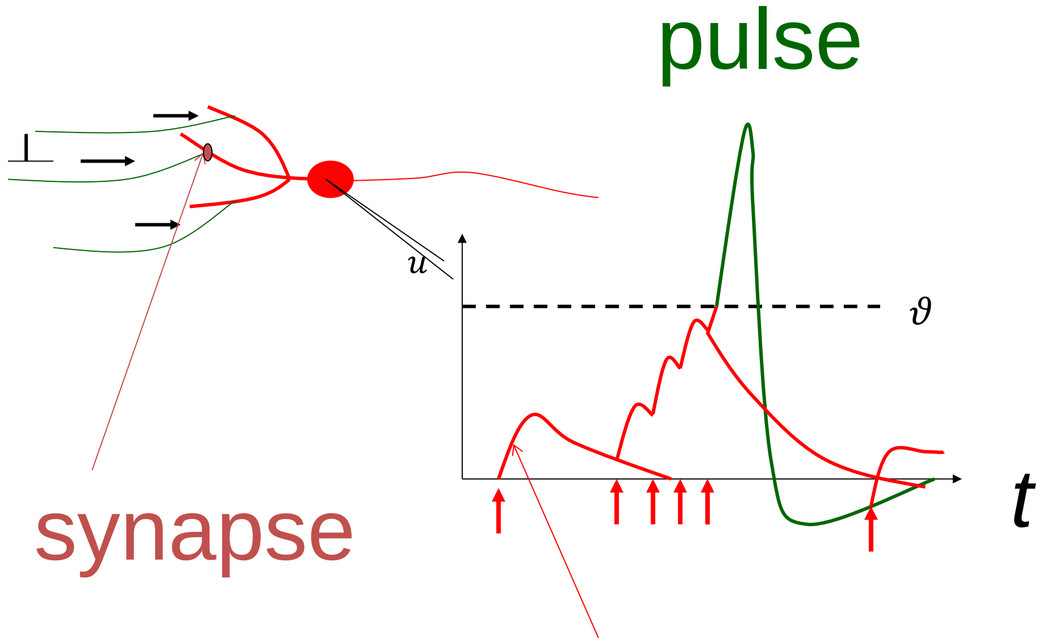
Previous slide.

In this class we will focus over 14 weeks on learning rules.

These learning rules can be interpreted
- either as solving an optimization problem (computer science/machine learning)
- or as implementing changes in connections (biology/neuromorphic hardware)

Modeling: artificial neurons





-responses are added
-pulses created at threshold
-transmitted to other

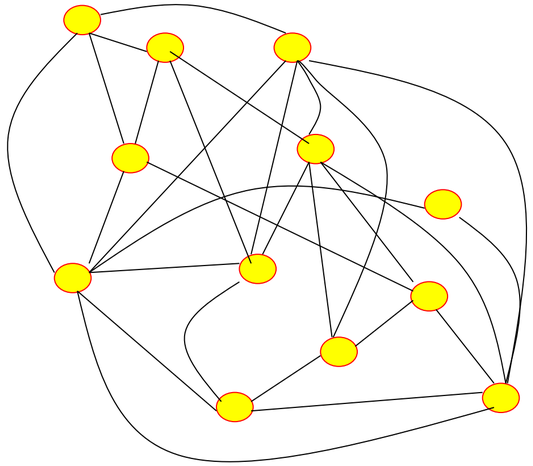
→ Mathematical description

Previous slide.
In the previous part we have seen that response are added and compared with a threshold.

This is the essential ideal that we keep for the abstract mathematical model in the following.

We drop the notion of pulses or spikes and just talk of neurons as active or inactive.

Modeling: artificial neurons

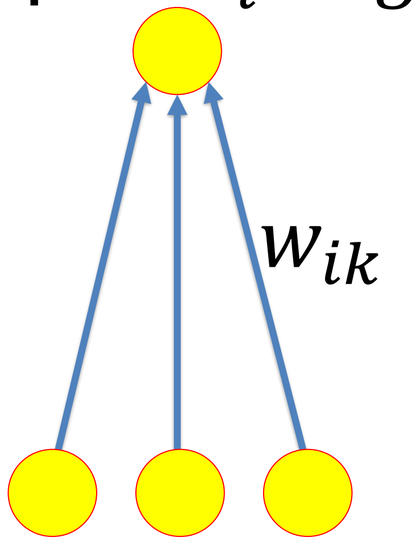


forget spikes: continuous activity x

forget time: discrete updates

nonlinearity/threshold

activity of output $x_i = g\left(\sum_k w_{ik}\, x_k\right)$



activity of inputs x_k

weights = adaptive parameters

Previous slide.

The activity of inputs (or input neurons) is denoted by x_k

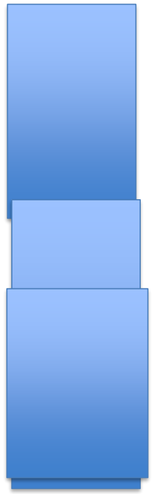
The weight of a synapse is denoted by w_{ik}

The nonlinearity (or threshold function) is denoted by g

The output of the receiving neuron is given by

$$x_i = g\left(\sum_k w_{ik}\, x_k\right)$$

Quiz: biological neural networks



[ ] Neurons in the brain have a threshold.
[ ] Learning means a change of the connection weights
[ ] The total input to a neuron is the weighted sum of individual inputs
[ ] The neuronal network in the brain is feedforward: it has no recurrent connections

[ ] Most biological neurons (brain) communicate by short pulses (spikes)
[ ] Most artificial neurons (ANNs) communicate by short pulses (spikes)
[ ] Classic implementations of ANNs use a von-Neumann architecture
[ ] GPU implementations of ANNs use a von-Neumann architecture (i.e., separation of processing and memory)

Previous slide. Your notes

Learning in Neural Networks: Introduction

Towards Neuromorphic Hardware

Wulfram Gerstner

EPFL, Lausanne, Switzerland

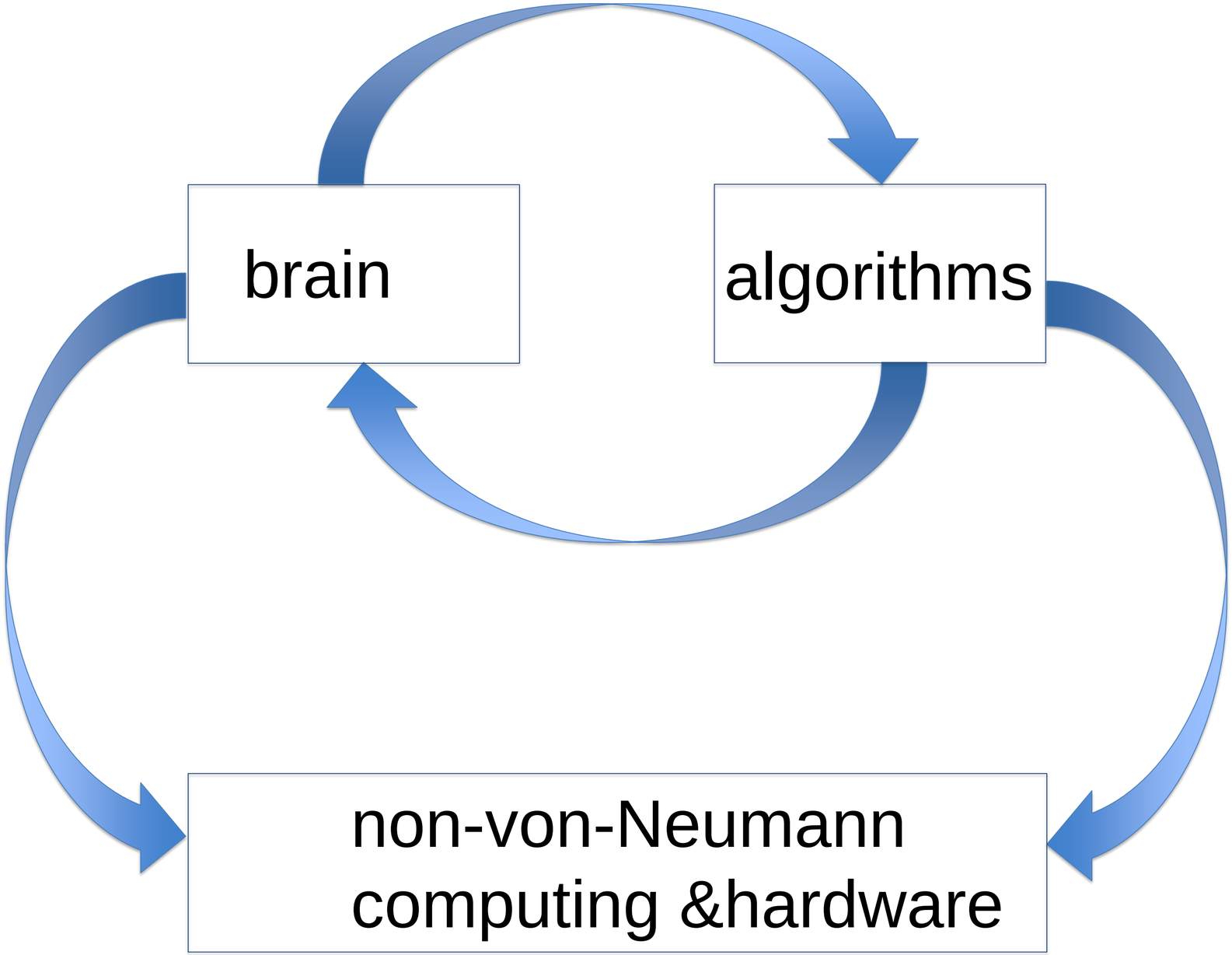
From Artificial Neurons to
Neuromorphic hardware

Previous slide:
Neuromorphic hardware in the narrow sense is a hardware that mimics ‘on the substrate’ aspects of biological processing or brain-like neural networks. In the broad sense it refers to a class of different computing materials or unconventional computing paradigms. Sometimes the latter are also called non-von-Neumann computing or non-von-Neumann hardware.

One of the major drivers for developments in this direction is the high energy consumption of standard computers, in particular in comparison to the energy consumption of the human brain.

This class will have only one week on neuromorphic hardware, but neuromorphic applications are always in the background when we think about the learning algorithms that we explore in this class.

Learning Rules in Neural Networks



Previous slide.
A quick look at the field of neuromorphic hardware.

Energy consumption of the brain

- Sedentary humans eat and use 2500 kCal per day
- Translate to Joule → 10 000 kJ
- Brain facts: 20 percent of energy consumption of human at rest goes into the brain
- Most of it goes into synaptic signaling (spike transmission)
- Brain uses 24 – 30 Watt (5 modern light bulbs)

The power consumption of the brain is relatively low!
→ **10h of hard thinking = 0.3kWh**

https://www.brainfacts.org/Brain-Anatomy-and-Function/Anatomy/2019/How-Much-Energy-Does-the-Brain-Use-020119
https://biology.stackexchange.com/questions/16316/what-is-the-energy-consumption-of-the-brain

Previous slide.
The claim is that the power consumption of the brain (30W) is relatively low.

Low compared to what?
- Compare with GPU
- Compare with household power consumption.

Energy consumption of one GPU

- 300 W (RX 6800/6900 XT)
- 350 W (RTX 3080/3090)

→ 10h of training an ANN on 1 GPU = 3.5 kWh

1 day of training an ANN on 1 GPU = 8000Wh = 8 kWh

4 months GPU usage → 1000 kWh

12 months GPU usage → 3000 kWh

https://www.tomshardware.com/features/graphics-card-power-consumption-tested

Previous slide:

A day has 24 hours. So we multiply the power (350W) with the number of hours.

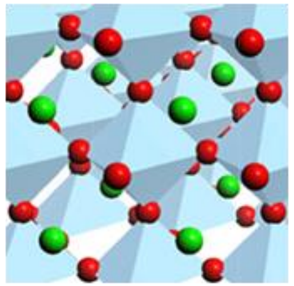
4 months have 120 days. Again a simple multiplication

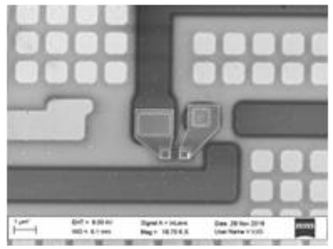
The question then is: are 3000kWh per year a lot?

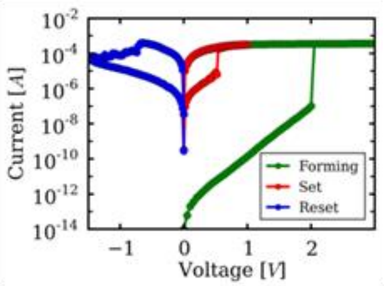
We need to compare with ‘normal’ energy consumption.

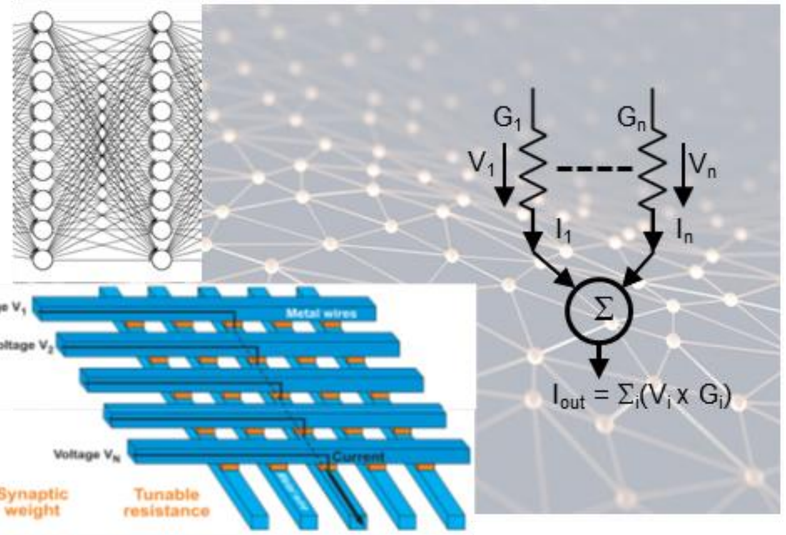
A modern apartment (with heat pump heating) needs, less than 1000kWh per year. A family of four with normal electricity usage uses less than 2000 kWh per year.
In other words, a single GPU burns as much energy as a whole family!

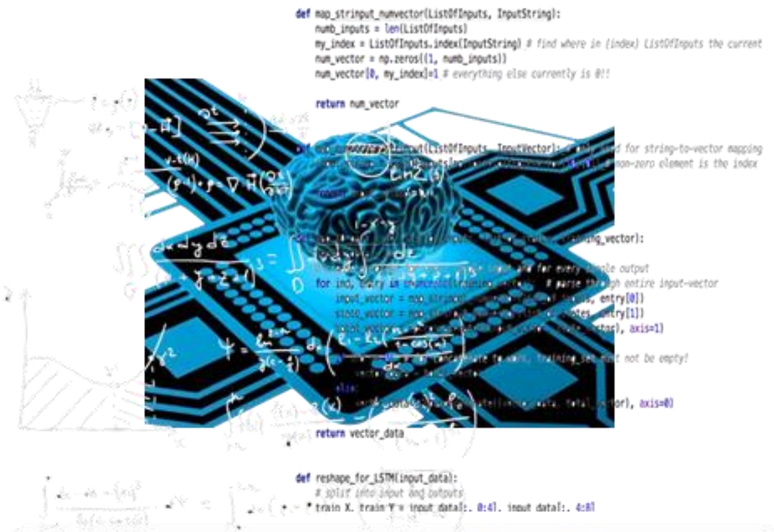
Accelerating Neuromorphic Workloads – Innovation required at all levels











New Materials and Devices

Non *von Neumann* Architecture

Hardware – Algorithm Interplay

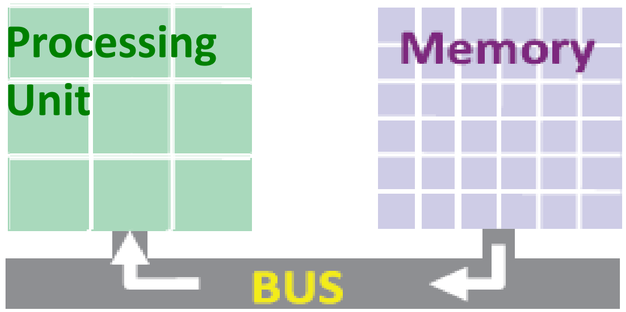
IBM slide

Previous slide:

The project of IBM research focuses mostly on Matrix multiplication (middle) and update of the matrix elements as a result of a learning rule (‘algorithm’, right).

Analog signal processing for scalability

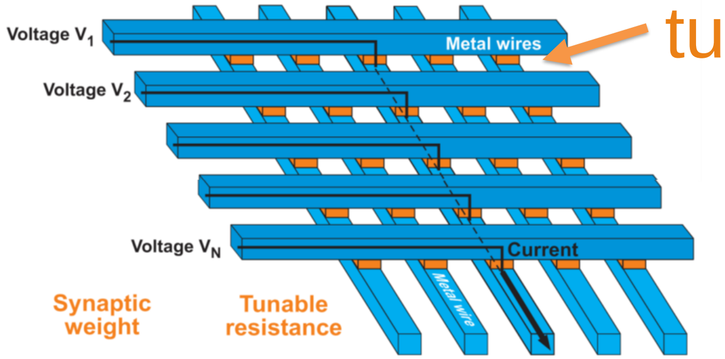
- **Limiting factors of von Neumann architecture**
  - Memory access
  - Sequential operations
  - Digital signal processing



Compute effort ~O(#Neurons^2)

- **Overcome by**
  - In-memory computing
  - Parallel operations
  - Analog signal processing

memory in resistors: tunable



Compute effort ~O(N)

Electrical (and optical solutions) are viable candidates

IBM slide

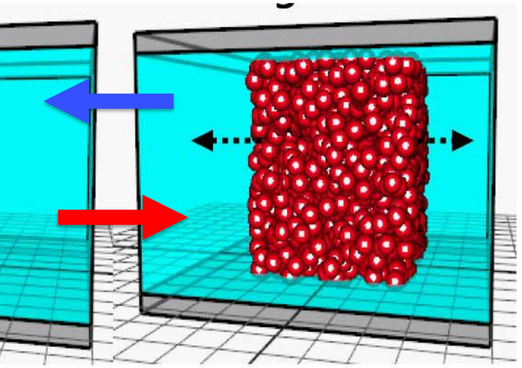
Previous slide:

For these kind of matrix operations we should exploit new computing concepts.
The traditional von-Neumann paradigm is limited by signal flow and bad scaling as a the number of neurons per layer increases.

Tunable weights via Memristive Devices

‘memory of resistance’ = ‘memristor’

Understanding the mechanism



IBM MO_3+HfO_2

Continuous & symmetric change of R

Woo et al. IEEE Electr. Dev. Lett. 38, 9 (2017)

- Resistance depends on molecular configuration
- Resistance increase or decreases with voltage pulses above threshold value
- Resistance keeps memory

Images: IBM

Previous slide:

Memristive material studied by IBM.

The basic function arises from the following principle.

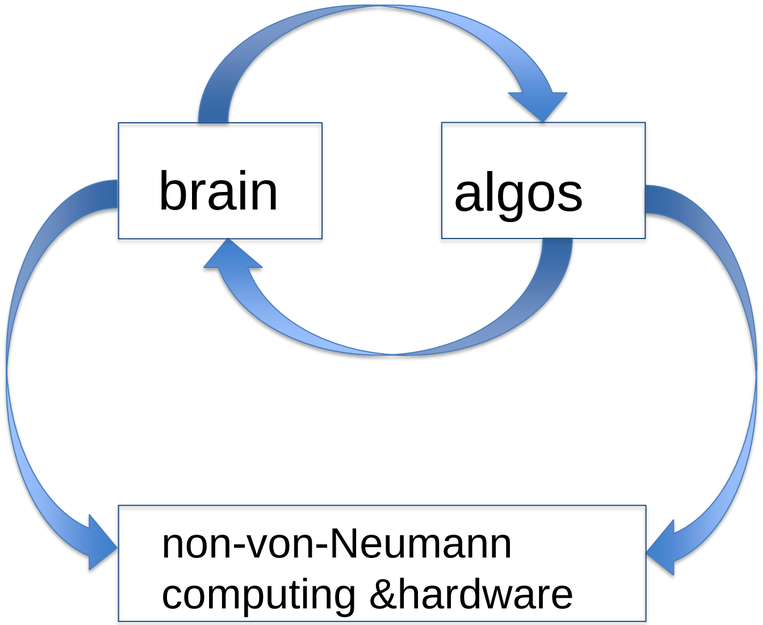
The material in light blue is an electrical insulator (dielectric material). However, with a first strong voltage pulse one can create an initial breakdown in the material. This leads to a short-cut illustrated by a thin red column of molecules in a conducting state (lower left). Now the material is now longer insulating, but has a finite resistance.

With an additional medium-sized **positive voltage pulse** (red), the column of conducting molecules can be made thicker so that the resistance decreases (lower right).

With a later medium-sized **negative voltage pulse** (blue), one can return to the initial configuration (lower left).

Weak currents and weak voltage pulses have no effect. Hence the material keeps its configuration and resistance for a long time. It has a **‘Memory of Resistance’ → Memristor.**

Summary: Learning in Neuromorhoic Hardware



- In-memory computing: non-von Neumann architecture
- Connections between units can be physically implemented (cross-bar array with resistors at crossing points)
- Learning implies changes of connections
- Memristors are changeable resistors that keep their ‘memory’.
- Changes of memristors can be induced by voltage pulses

→ **Candidate for implementing distributed learning algorithms**

Previous slide:

Summary

Learning in Neural Networks: Introduction

Coarse Brain Anatomy

Wulfram Gerstner

EPFL, Lausanne, Switzerland

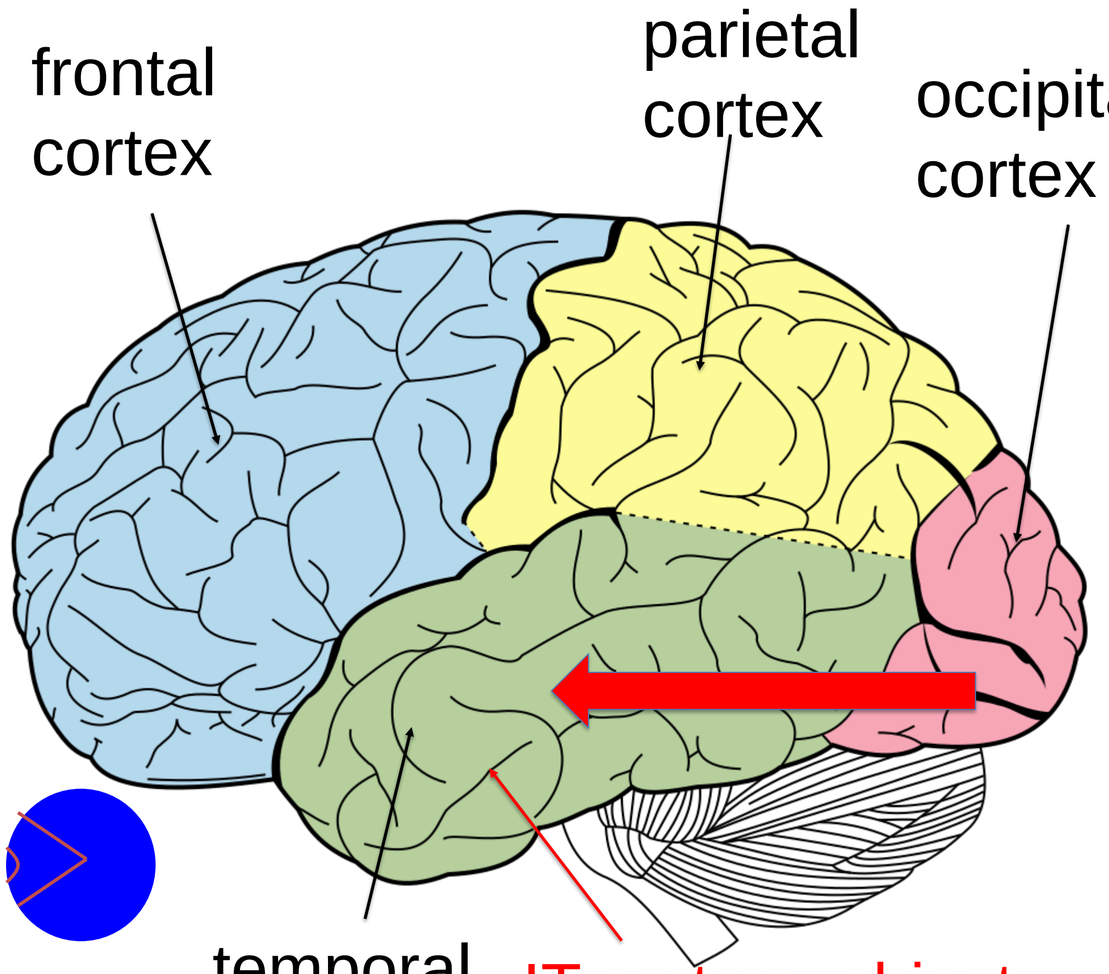
Coarse Brain Anatomy

Previous slide.

Let us have a coarse look at the brain.

Coarse Brain Anatomy: Cortex

Sensory representation in visual/somatosensory/auditory cortex



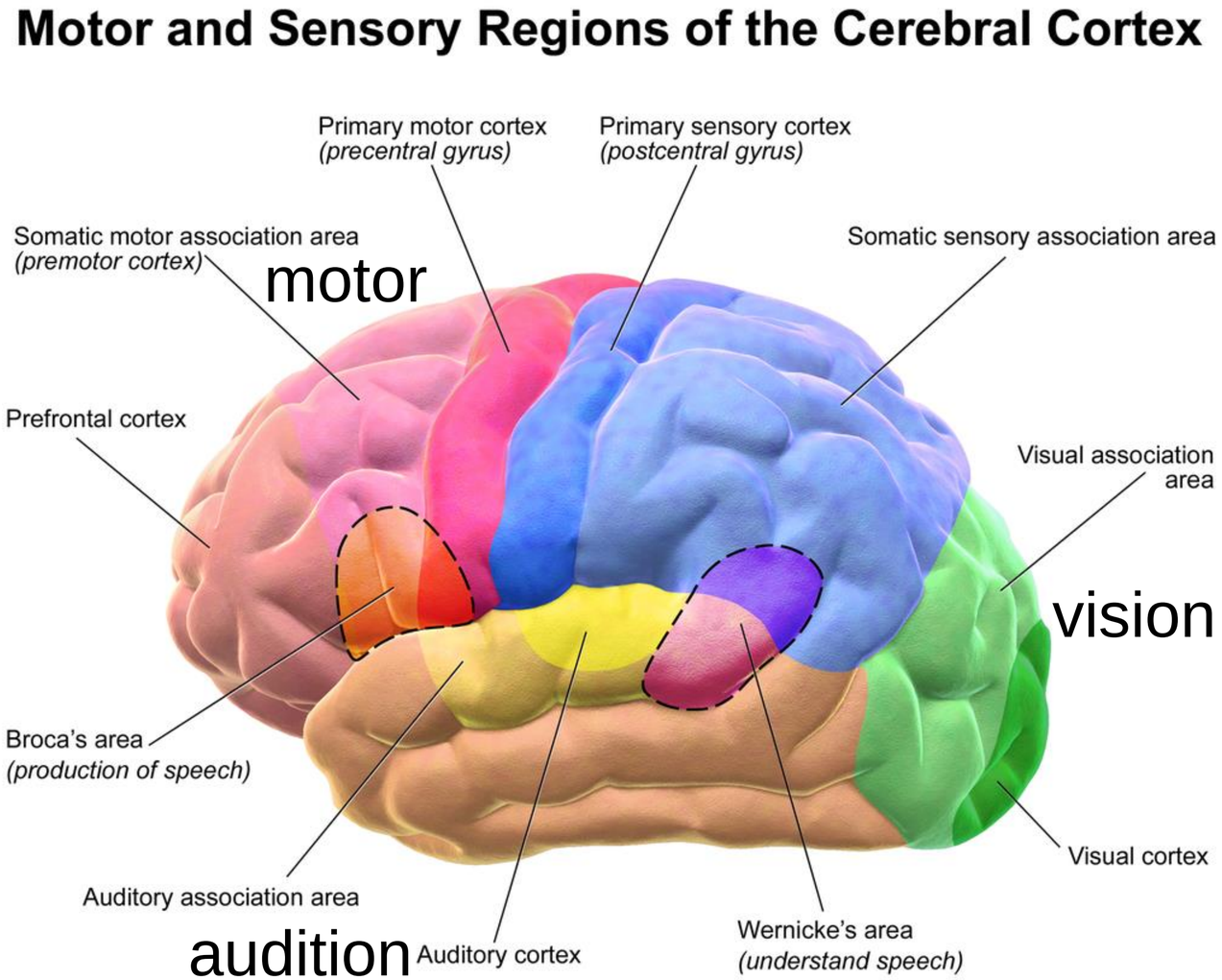


fig: Wikipedia

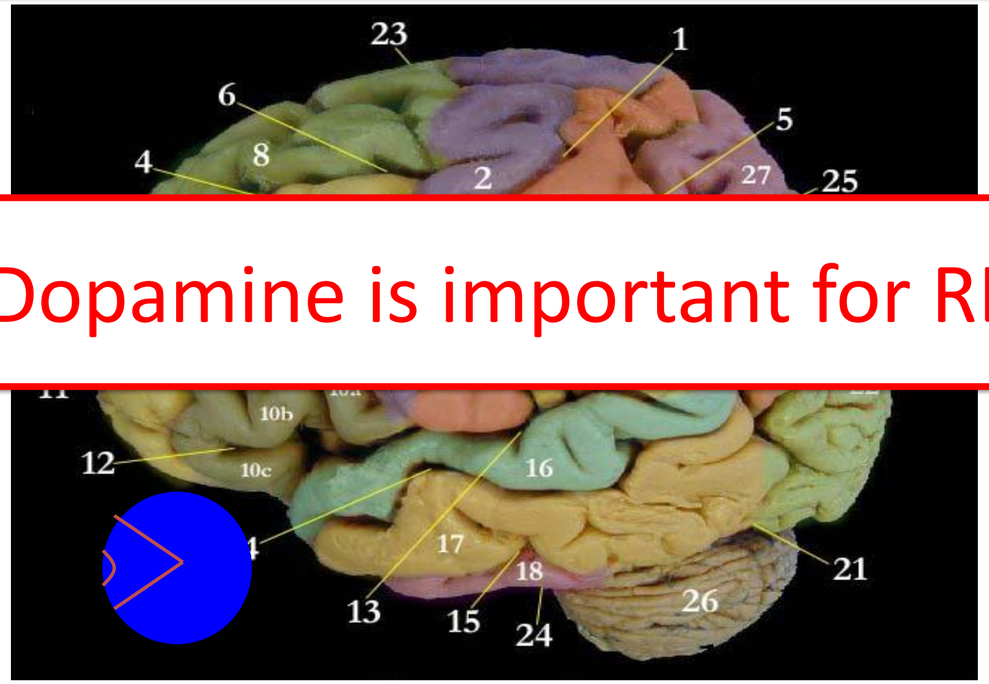
Previous slide.
Left: Anatomy. The Cortex is the part of the brain directly below the skull. It is a folded sheet of densely packed neurons. The biggest folds separate the four main parts of cortex (frontal, Parietal, occipital, and temporal cortex)

Right: Functional assignments. Different parts of the brain are involved in different tasks. For example there several areas involved in processing visual stimuli (called primary and secondary visual cortex). Other areas are involved in audition (auditory cortex) or the presentation of the body surface (somatosensory cortex). Yet other areas are prepared in the preparation of motor commands for e.g., arm movement (primary motor cortex)

Coarse Brain Anatomy

- many different cortical areas
- but also several brain nuclei sitting below the cortex
- Some of these nuclei send dopamine signals
- Dopamine sent from: VTA and substantia nigra
- **Dopamine is related to reward, surprise, and pleasure**

Dopamine is important for RL →



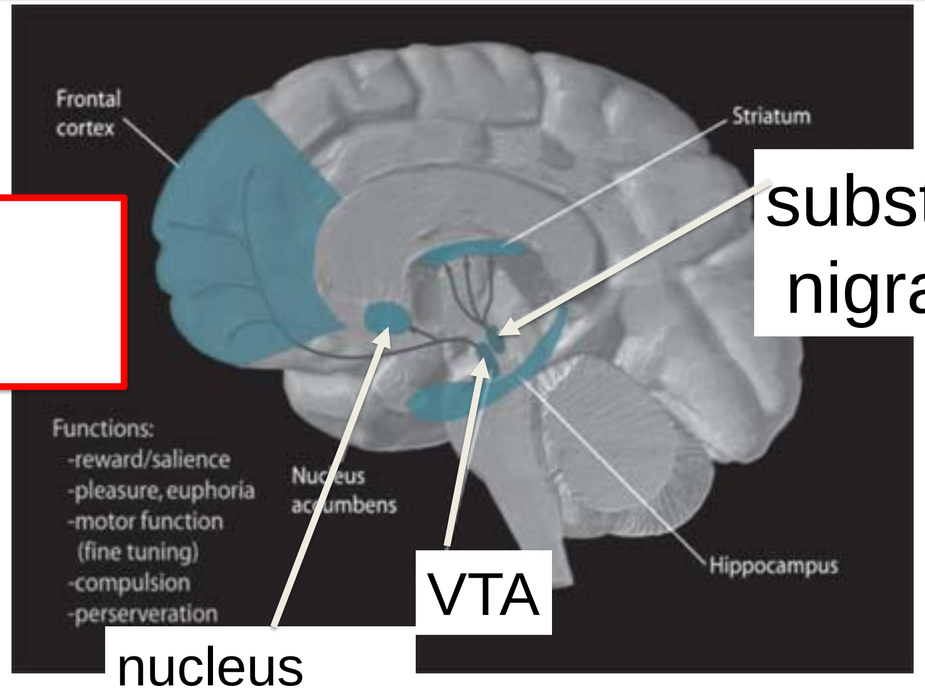
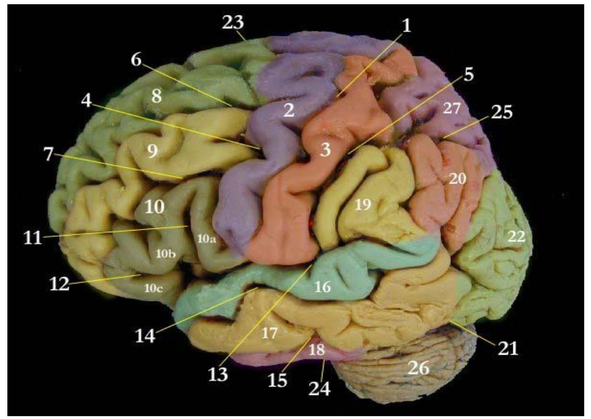


fig: Wikipedia commons

Previous slide.
Left: Anatomy. View on the folds of the cortex, and main cortical areas in different color.

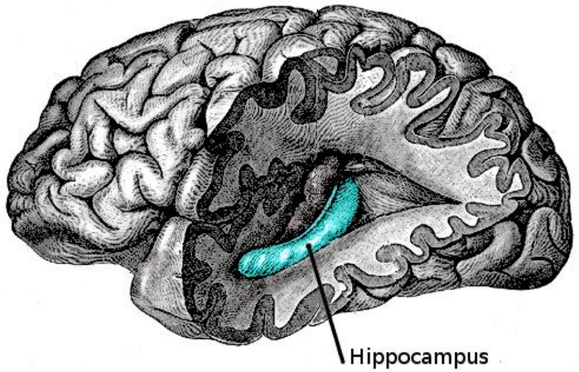
Right: Below the cortex sit different nuclei. Some of these nuclei use dopamine as their signaling molecule. Important nuclei for dopamine are the Ventral Tegmental Area (VTA) and the Substantia Nigra pars compacte (SNc). These dopamine neurons send their signals to large areas of the cortex as well as to the striatum (and nucleus accumbens).
Since dopamine is involved in reward, these dopamine neurons will play a role in this class in the parts that links reinforcement learning (RL) and the brain.

Coarse brain anatomy: the brain adapts during use



More space for fingers allocated in **somato-sensory cortex**
(=body representation; number 3 on image)
- musicians vs. non-musicians

Amunts et al. Human Brain Map. 1997
Gaser and Schlaug, J. Neuosci. 2003



More space allocated in **hippocampus**
(= representation of space; blue on image)
- London taxi driver vs bus driver

Macquire et al. Hippocampus 2006

DOI 10.1002/hipo.2023

→ ‘state representation’ is ‘learned’

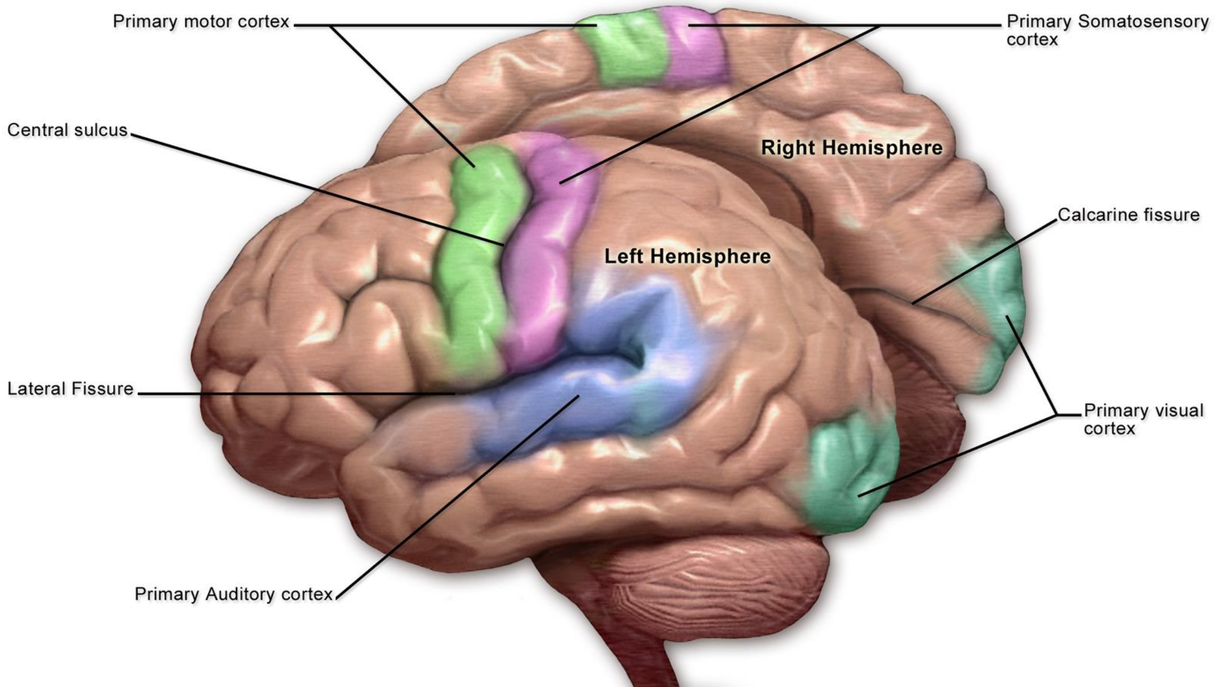
Previous slide. (not shown in class)

https://en.wikipedia.org/wiki/Primary_somatosensory_cortex#/media/File:Blausen_0103_Brain_Sensory&Motor.png

We said that different areas of the brain are involved in different tasks. For example, the somatosensory cortex represents the body surface. Nowadays one can measure that the size of the cortical area devoted to fingers is larger for musicians than for non-musicians. Since musicians are not born with a larger area, this result implies that experience can influence the function of the neurons in the brain. Somatosensory cortex is labeled 3 (previous page). The actual movements of fingers and other body parts are controlled by motor cortex (label 2).

Similarly, hippocampus is involved in spatial navigation. Not surprisingly, London taxi drivers have a bigger hippocampus than London bus drivers.

Image from wikipedia



1. Quiz: Coarse Functional Brain anatomy

[ ] the brain = the cortex (synonyms)
[ ] the cortex consists of several areas
[ ] some areas are more involved in vision, others more in the representation of the body surface
[ ] below the cortex there are groups (clusters) of neurons
[ ] dopamine is linked to reward, pleasure, surprise

End of Introduction
Questions?

Motivation: No Backprop, please!!!
Overview: Neuronal Networks in the brain
Overview: Neuromorphic hardware
Overview: Coarse Brain Anatomy

BREAK now.
Next Lecture at 12h15

Learning in Neural Networks: Lecture 1

Synaptic Plasticity and Learning rules

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Synaptic Plasticity: Learning rules of the brain

Previous slide.

Learning is related to synaptic plasticity. Therefore this is our second topic.

The claim is that the biological observation of ‘synaptic plasticity’ is the basis of ‘learning rules’ implemented in the brain.

Two important manifestations of synaptic plasticity are Hebbian Learning and Long-Term Potentation (LTP) that will be explained in this part.

Behavioral Learning

Learning actions (reward-based):

→ riding a bicycle

→ play tennis

→ play the violin

‘models of action choice’

Remembering episodes

→ first day at EPFL

→ first visit in a new city

→ reward-free

‘models of the world’

Previous slide.

When we walk around a city for the first time we develop a model of the environment – even in the absence of any specific rewards (except, may be, that it is good to know how to find the way home).

You may remember your first day at EPFL.

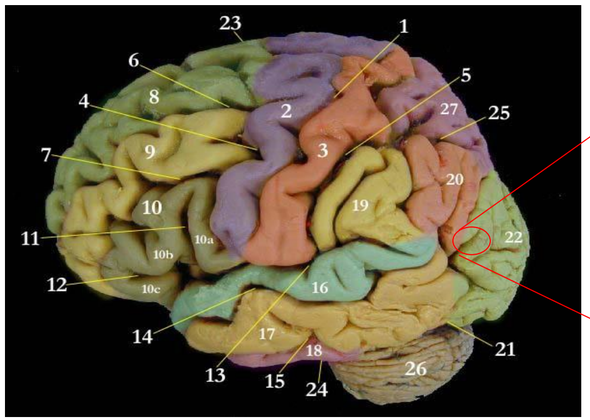
Both are examples of learning. Remembering episodes (and environments) is mainly **unsupervised learning** (or in some context: self-supervised learning).

When we learn to ride a bike we learn with Reinforcement-like feedback, e.g., we don’t want to fall because falling hurts.

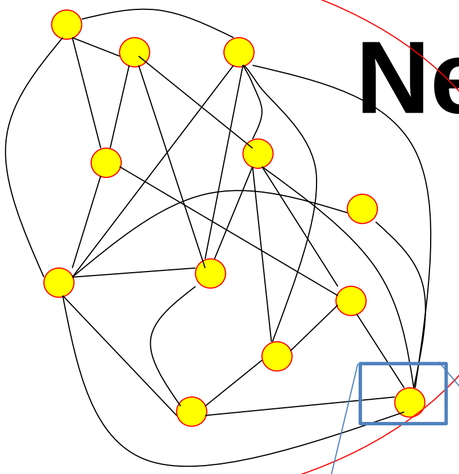
When we learn play the tennis we also get feedback via the observed outcome – which can be good or bad.

Both are example of **reinforcement learning** (also called reward-based learning).

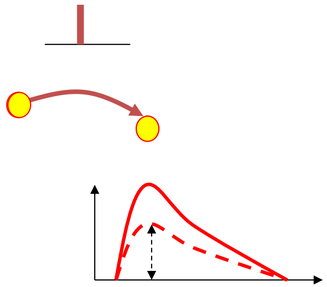
Behavioral Learning – and synaptic plasticity



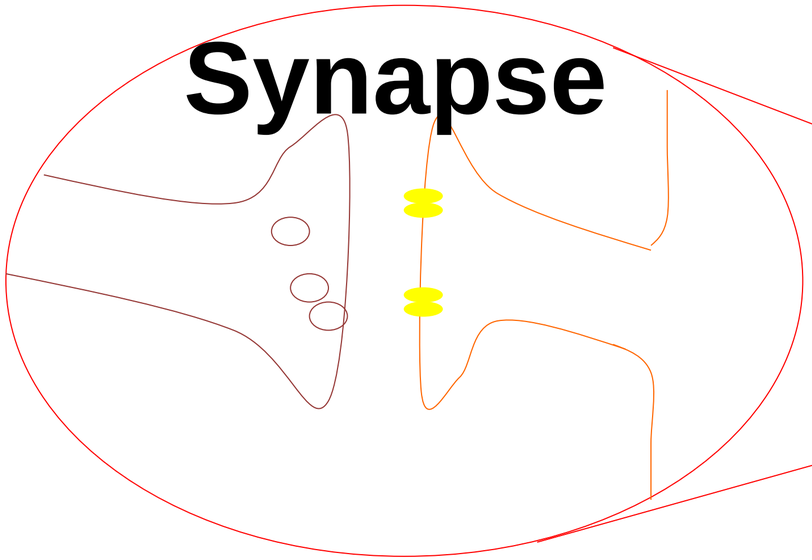
Neurons

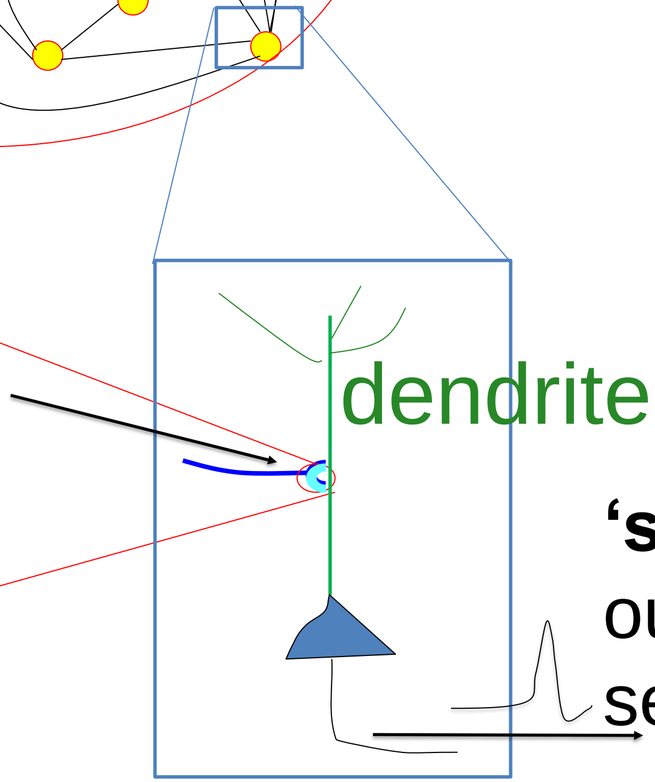


Amplitude of Postsynaptic Potential (PSP)



Synapse





‘spike’:
output signal (pulse)
sent to other neurons

Synaptic Plasticity = Change in Connection Strength

Previous slide.

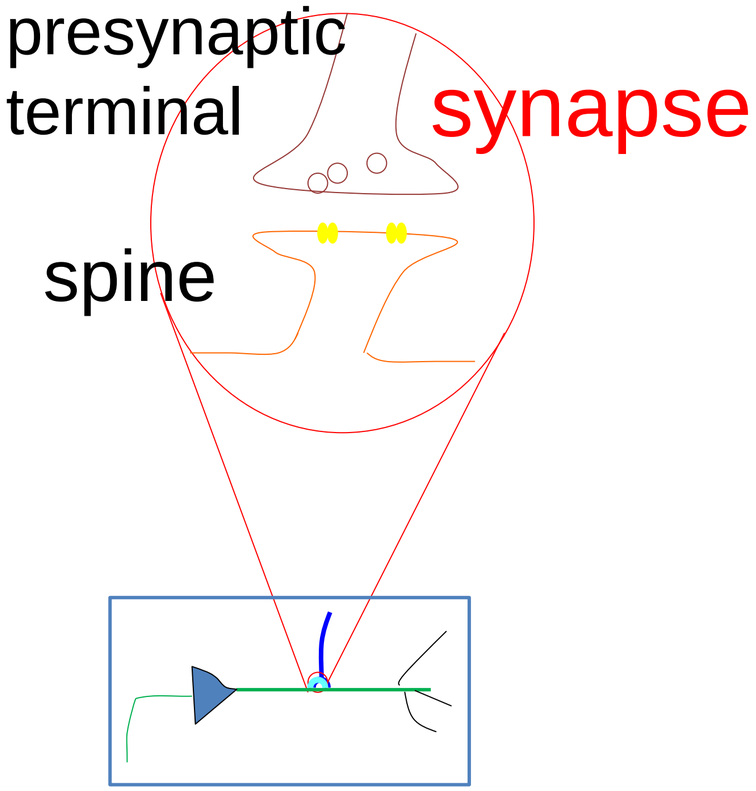
When we observe learning on the level of behavior (we get better at tennis), then this implies that something has changed in our brain:
The contact points between neurons (called synapses) have changed. Synaptic changes manifest themselves as a change in connections strength.

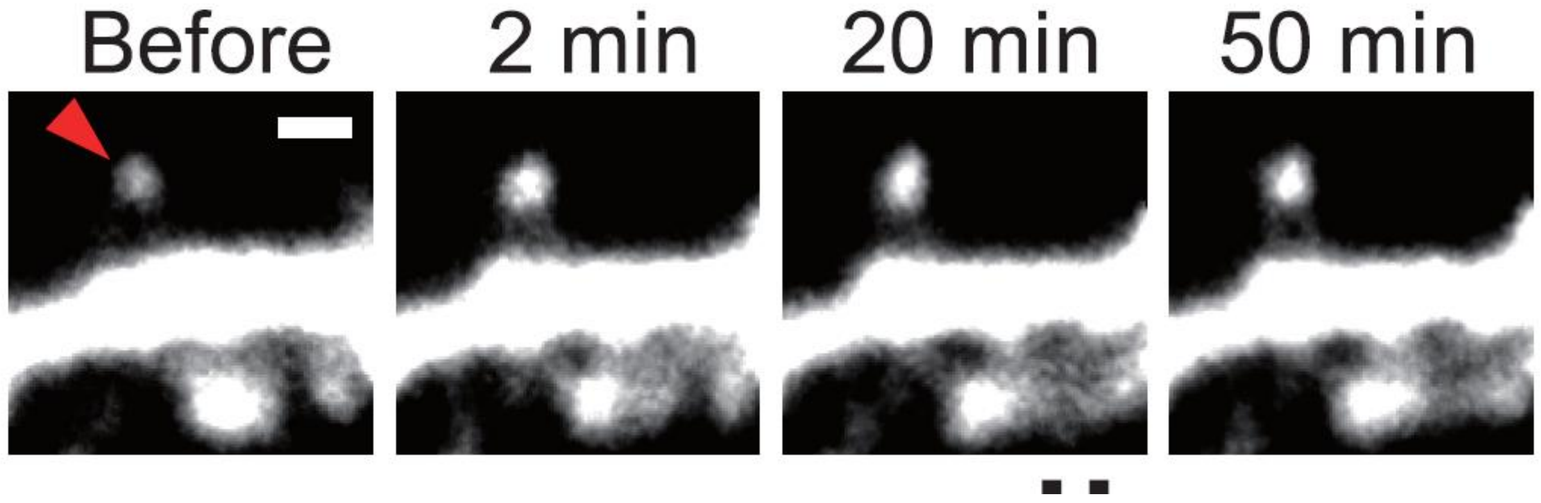
Synaptic plasticity describes the phenomena and rules of synaptic changes.

The connection strength can be measured by the
- amplitude of the postsynaptic potential (PSP)
- by physical size of the synapse (in particular the spine, see next slide)

Important:
Neurons communicate with each other by short electrical pulses, often called ‘spikes’.

Synaptic plasticity – structural changes





Yagishita et al.
Science, 2014

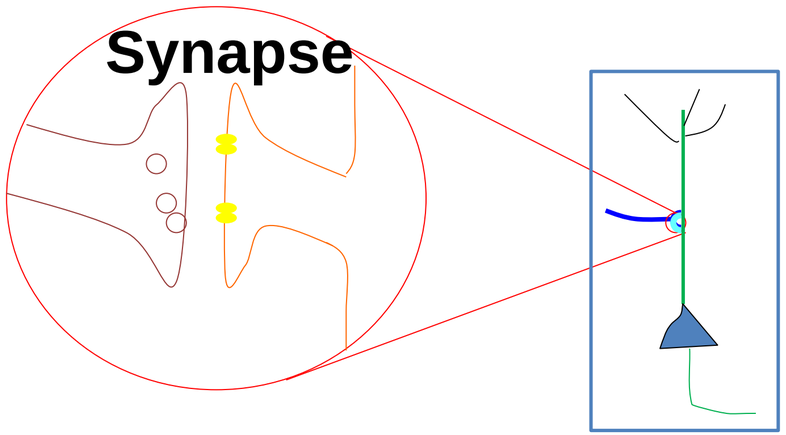
Previous slide.

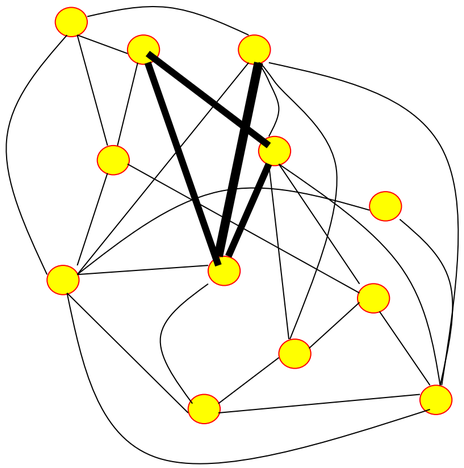
The synaptic connection consists of two parts. The end of an axonal branch coming from the sending neuron; and the counterpart, a protrusion on the dendrite of the receiving neuron, called spine.

We refer to the sending neuron as presynaptic and to the receiving one as postsynaptic.

A change in the connection strength is observable with imaging methods as an increase in the size of the spine. The bigger spine remains big for a long time (here observed for nearly one hour).

Synaptic plasticity: summary





- Connections can be strong or weak
- Strong connections have thick spines
- **Synaptic plasticity**
  = change of connection

Syn. Plasticity should enable Learning

- memorize facts and episodes
- learn to recognize WHERE we are
  → current state/representation of current input
- learn models of the world
  → predict the near future
- learn appropriate actions

Previous slide.

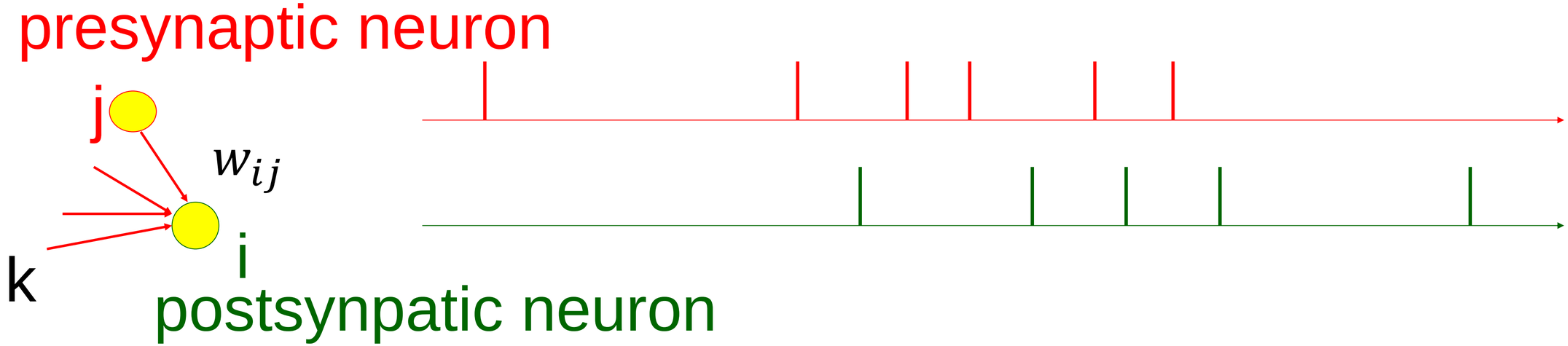
Thus connections can be strong or weak – and synaptic plasticity describes the changes of one synapse from weak to strong or back.

The synaptic changes are thought to be the basis of learning – whatever the learning task at hand. And RL has several aspects of learning:

- learn to recognize states = where we are;
- learn to choose good actions = action selection;
- learn to predict possible next states = model-based reinforcement learning.

The question now is: Are the any ‘rules’ for connection changes that would predict whether and when a synapse gets stronger?

Hebb rule / Hebbian Learning



When an axon of cell **j** repeatedly or persistently takes part in firing cell **i**, then j's efficiency as one of the cells firing i is increased

Hebb, 1949

- local rule
- simultaneously active (correlations)

Previous slide.

The Hebb rule is the classic rule of synaptic plasticity.

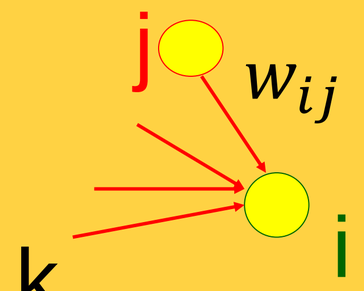
It is often summarized by saying: if two neurons are active together, the connection between those two neurons gets stronger.
Note that the original formulation of Hebb also has a ‘causal’ notion: ‘takes part in firing’ – which is more than just firing together.

Local rule means: changes only depend on information that is available at the synapse.
The changes for the weight from j to i can depend on the activity of neuron j and the state (or activity) of neuron i, and the value of the weight itself, but for example not explicitly on the activity of another neuron k. Note that if k connects to i, the activity of i is a good summary of the influence of k. In other words, i may depend IMPLICITLY on k, but the weight changes do not depend EXPLICITLY on k.

Quiz. Terms used Synaptic Plasticity and Learni

We look at the specific synapse w_{ij}

[ ] k is called the presynaptic neuron of the synapse w_{ij}
[ ] k is called a presynaptic neuron of i
[ ] j is called the presynaptic neuron of this synapse
[ ] i is called the postsynaptic neuron of this synapse



[ ] the strength of a synapse can be measured by the PSP amplitude.
[ ] PSP means presynaptic potential

Learning rules in the brain

[ ] Hebbian learning depends on presynaptic activity AND on state of postsynaptic neuron
[ ] A learning rule is called local, if it uses only information available at the location of the synapse.

Previous slide.

1. The neuron BEFORE the synapse is called the **presynaptic neurons:**
   it sends spike to the synapse.

2. The neuron AFTER the synapse is called the **postsynaptic neurons:**
   it receives a signal via thesynapse.

3. Hebbian learning: the joint activation of pre- and postsynaptic neuron induces a strengthening of the synapses.

4. A learning rule is called local, if it uses only information available at the location of the synapse.

Hebbian Learning (LTP)

Hebbian coactivation:
pre-post-post-post

"if two neurons are active together, the connection between those two neurons gets stronger."

"another synapse (red) which does not receive presynaptic spikes, does NOT increase"

Previous slide.

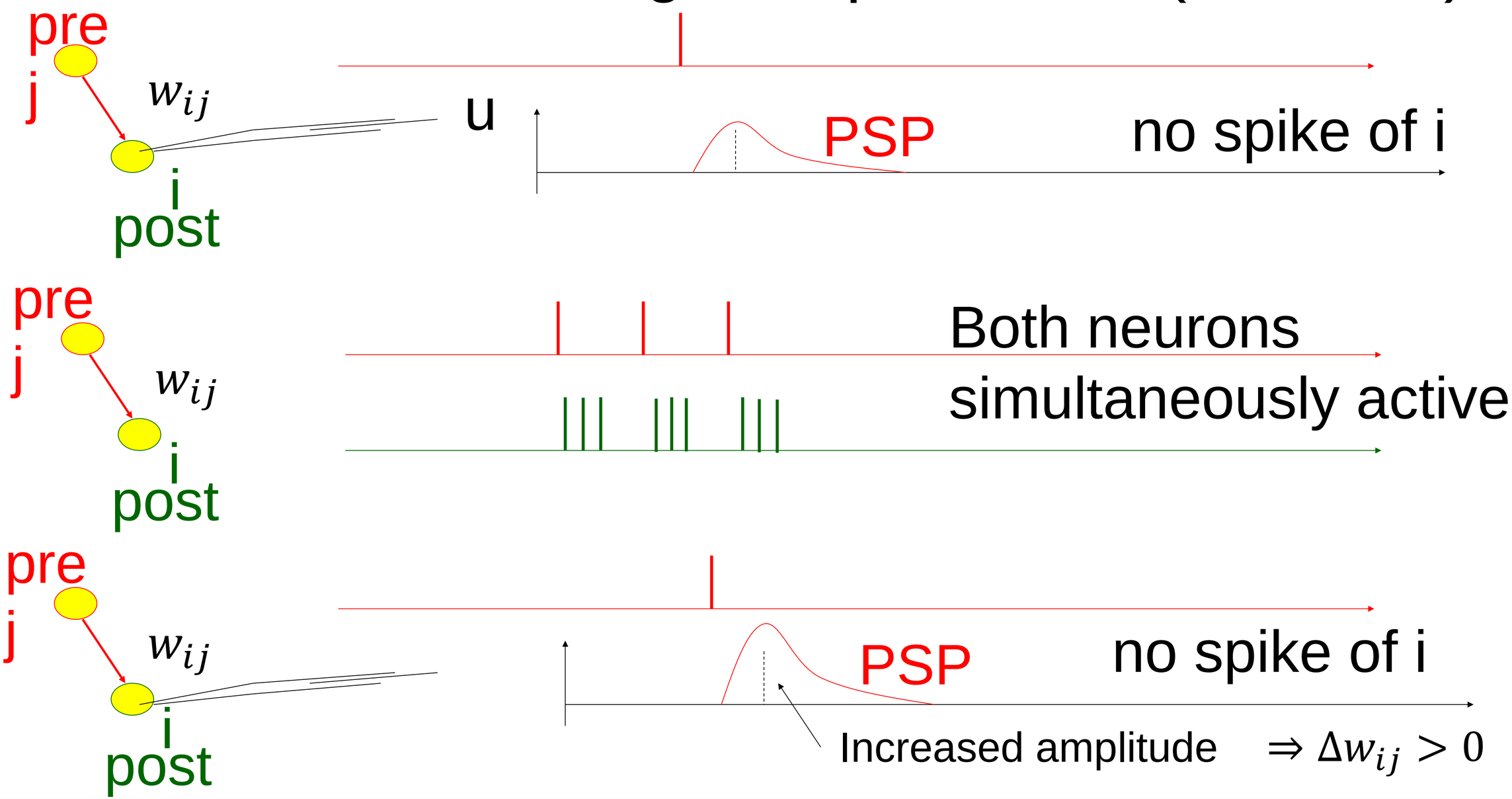
The joint activation of pre- and postsynaptic neuron induces a strengthening of the synapses. A strong stimulus is several repetitions of a pulse of the presynaptic neuron, followed by three or four spikes of the postsynaptic neuron.

Hundreds of experiments are consistent with Hebbian learning.

Note that by definition of Hebbian learning, only the stimulated synapses (green) is strengthened, but not another synapses (red) onto the same neuron.

Synaptic plasticity: Long-Term Potentiation (LTP)

Hebbian Learning in experiments (schematic)



Previous slide.

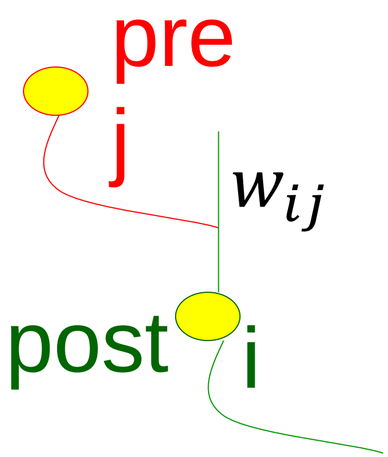
In a schematic experiment,

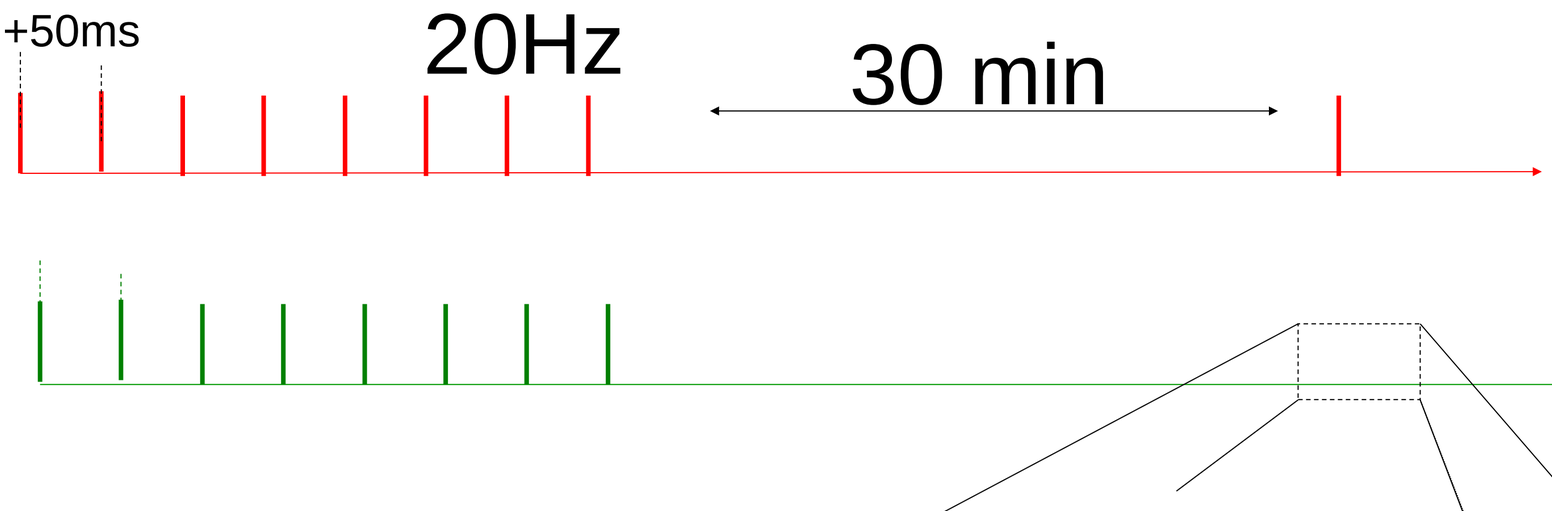
1) You first test the size of the synapse by sending a pulse from the presynaptic neurons across the synapses. The amplitude of the excitatory postsynaptic potential (EPSP) is a convenient measure of the synaptic strength. It has been shown that it is correlated with the size of the spine.

2) Then you do the Hebbian protocol: you make both neurons fire together

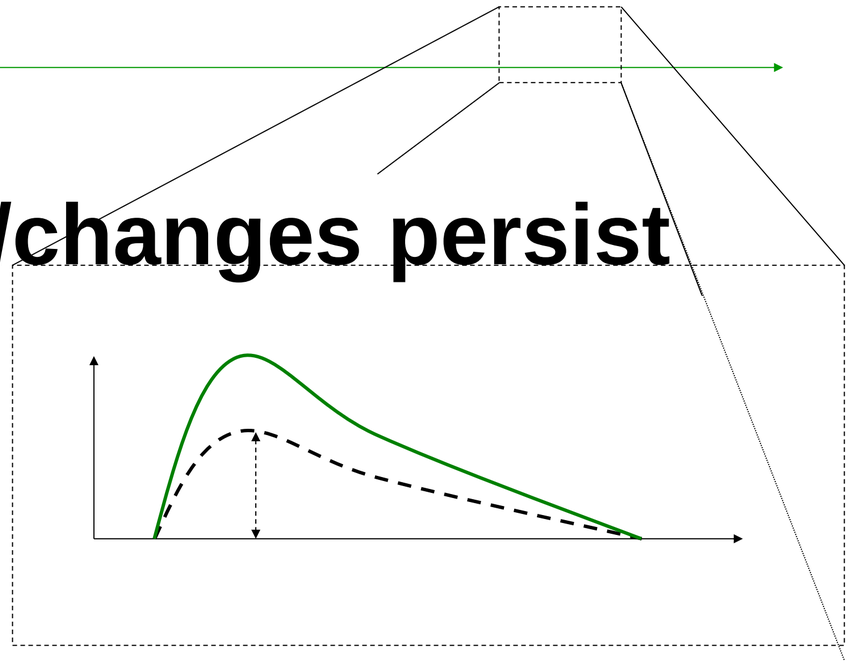
3) Finally you test again the size of the synapse. If the amplitude is bigger you conclude that the synaptic weight has increased.

Why the name ‘ Long-term plasticity ‘ (LTP)?





Long-term plasticity/changes persist



Changes

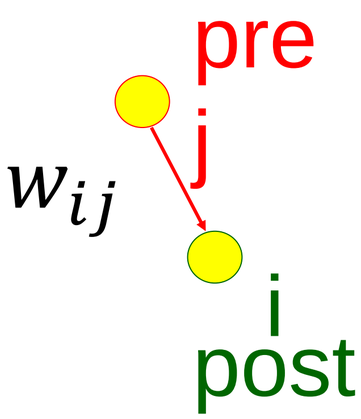
- induced over 3 sec
- persist over 1 – 10 hours (or longer?)

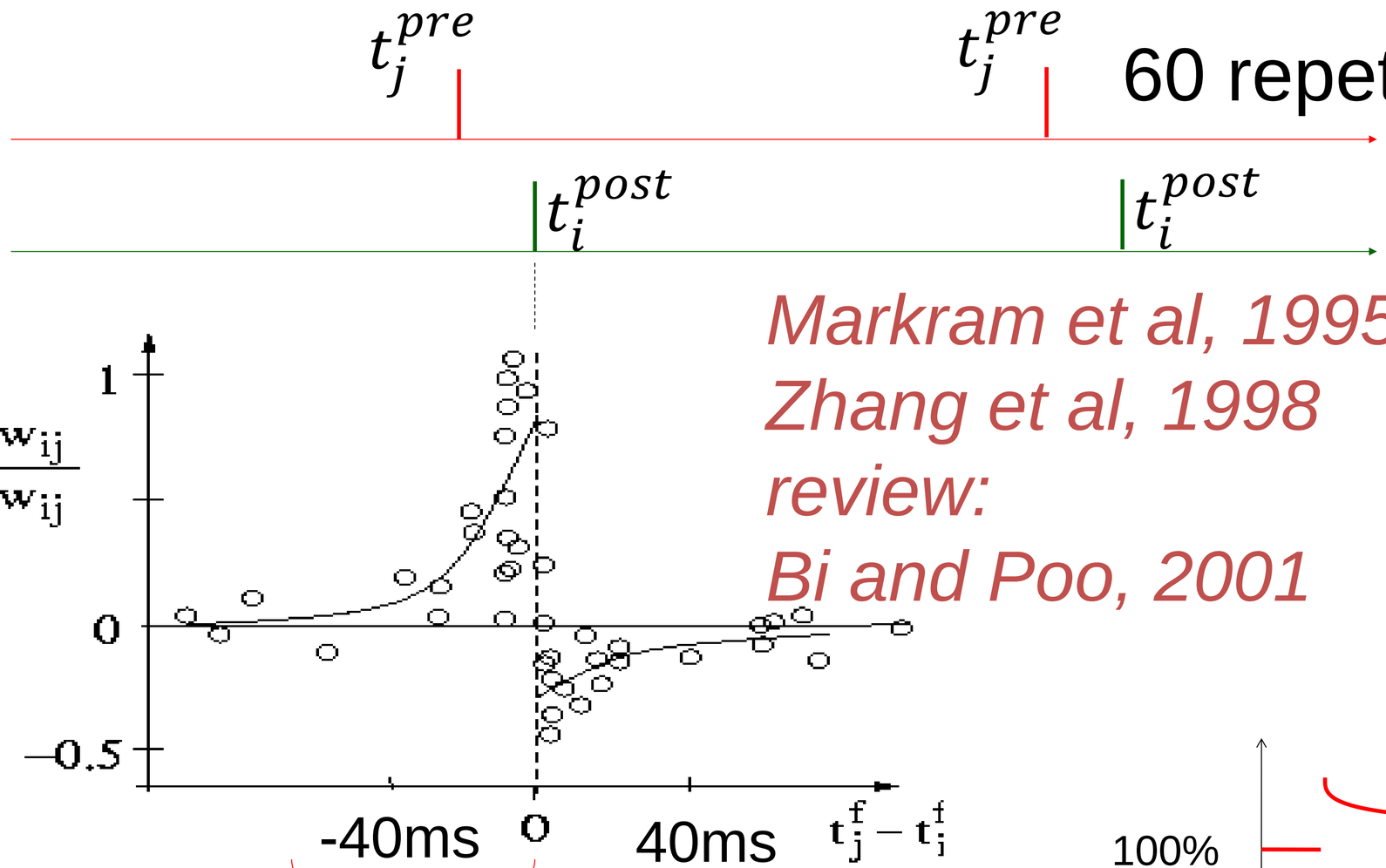
Previous slide.

Experimentalists talk about Long-Term Potentiation (LTP), because once the change is induced it persists for a long time. Interestingly, it is sufficient to make the two neurons fire together for just a few seconds.

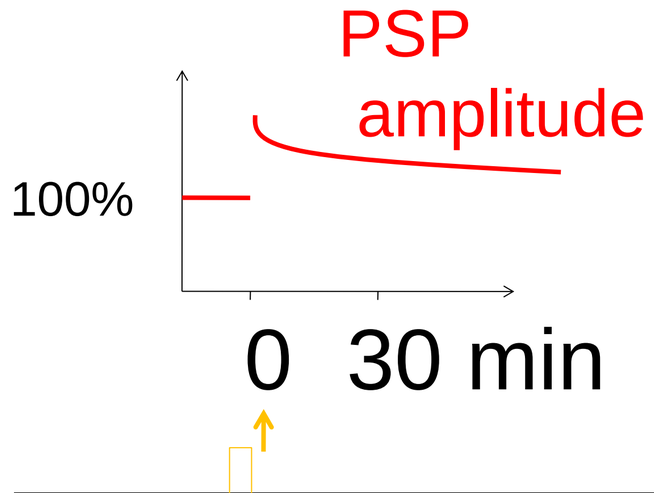
Thus induction of plasticity is rapid, but the changes persist for an hour or more.

Spike-timing dependent plasticity (STDP)





Markram et al, 1995,1997
Zhang et al, 1998
review:
Bi and Poo, 2001



Previous slide (not shown in class).
In the STDP paradigm of LTP induction, the presynaptic neuron is stimulated so that it emits a single spike, and the postsynaptic neuron is also stimulated so that emits a single spike – either a few milliseconds before or after the presynaptic spike. This stimulation protocol (for example pre-before-post) is then repeated several times.

The increase of the synaptic weight (induced by repeated pre-before-post) persists for a long time.
How much it increases (or decreases) depends on the exact timing of conicidences of pre- and post-spikes on the time scale of 10ms

Since the size of the increase depends on the relative timing of the two spikes, this induction protocol is called Spike-Timing-Dependent Plasticity (STDP).

Summary: Synaptic plasticity

Synaptic plasticity

- makes connections stronger (LTP) or weaker (LTD)
- can be experimentally induced
- needs ‘joint activation’ of the two connected neurons
- is induced rapidly, but can last for a long time
- There are many protocols (combinations of pre and post) to induce changes

Hebb rule:

- ‘neurons that fire together, wire together’

S. Loewl and W. Singer, Science 1992

‘Local rule’:

- only the activity of sending and receiving neurons matters

Previous slide.

There are several experimental paradigms to induce synaptic changes.
Most of these paradigms are consistent with the Hebb rule of LTP:
Neurons that fire together, wire together, a slogan that was introduced by Loewl and Singer in 1992. Other paradigms induce a DEPRESSION of the synapse, called LTD (long-term depression).

However, in all these Hebbian learning rules and their corresponding experimental paradigms, the role of reward is unclear and not considered.

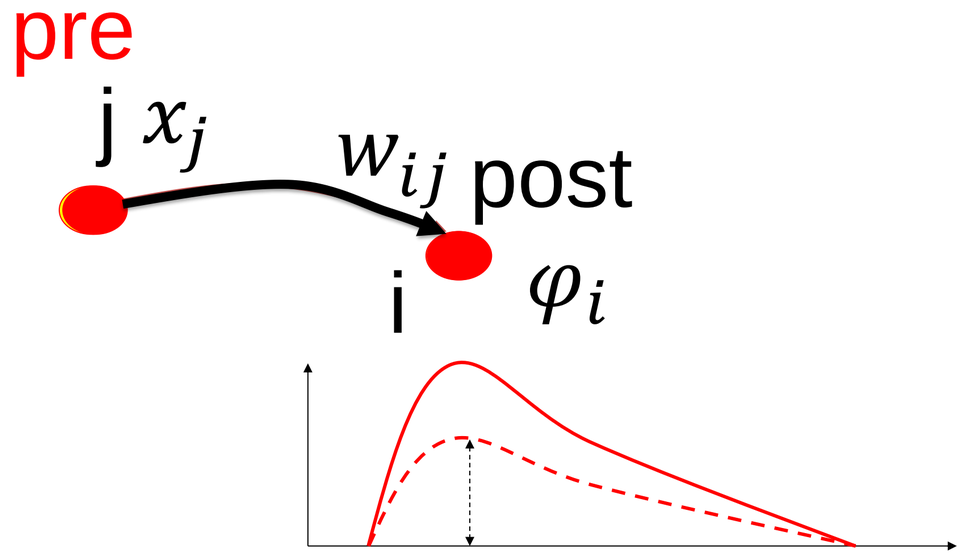
Hebbian rules are examples of ‘LOCAL’ learning rules.

- For the change of a connection from neuron j to neuron i, only the activity of these two neurons i and j matters, but not the activity of some other neuron k further away.
- Local means that only information that is locally available at the site of the synapse can be used to drive a weight change. What is available is the value of the weight itself, as well as the state of the postsynaptic neuron and the incoming spikes sent by the presynaptic neuron.

Hebbian Learning depends on two factors

1. ‘local’ learning rule: only local information is used

$$\Delta w_{ij} = F\left(pre, post, w_{ij}\right)$$



2. Changes depend on two factors:

- pre (spike arrival from neuron j)
  → variable x_j
- post (activation or output spike of postsynaptic neuron i)
  → variable φ_i

$$\Delta w_{ij} = F\left(x_j, \varphi_i, w_{ij}\right)$$

3. Sensitive to coincidences ‘pre’ and ‘post’

example

$$\Delta w_{ij} = c\, x_j \left[\varphi_i - b\right]$$

Hebbian rules = 2-factor rules

Previous slide.

In standard Hebbian learning, the change of the synaptic weight depends on presynaptic activity x_j (the presynaptic factor, pre) and the state of the postsynaptic neuron (a specific example of a postsynaptic factor is $\varphi_i - b$, where b is an arbitrary constant).

1. The rule is local: it depends only on information that is available at the synapse.
2. It is built from two factors: the multiplication of a presynaptic and a postsynaptic factor.
3. Note that it does not contain the notion of reward or success.

Now we want to see whether such rules can be mapped to the math we did in this class!

I use the term Hebbian rules and 2-factor rules interchangeably.

Quiz. Synaptic Plasticity and Learning Rules

Standard Long-term potentiation

[ ] has an acronym LTP

[ ] takes more than 10 minutes to induce

[ ] lasts more than 30 minutes

[ ] depends on presynaptic activity
AND on state of postsynaptic neuron

Hebbian Learning:

[ ] Hebbian learning depends on presynaptic activity (presynaptic factor)
AND on state of postsynaptic neuron (postsynaptic factor)

Feedback on Brain Anatomy and Hebbian Learning rules

[ ] Up to here at least 60 percent of the material was new to me

For 80 percent of the material that we have seen so far

[ ] I understood the concepts and got a rough or reasonably precise idea of the biological phenomena

Learning in Neural Networks: Lecture 1

Hebbian Learning rules

Wulfram Gerstner

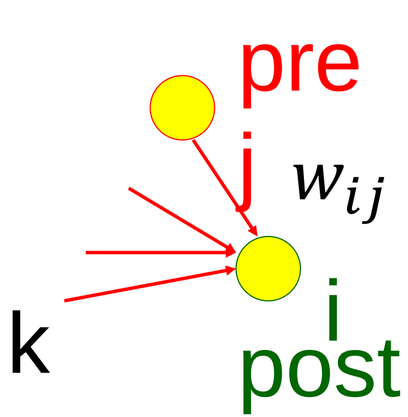
EPFL, Lausanne, Switzerland

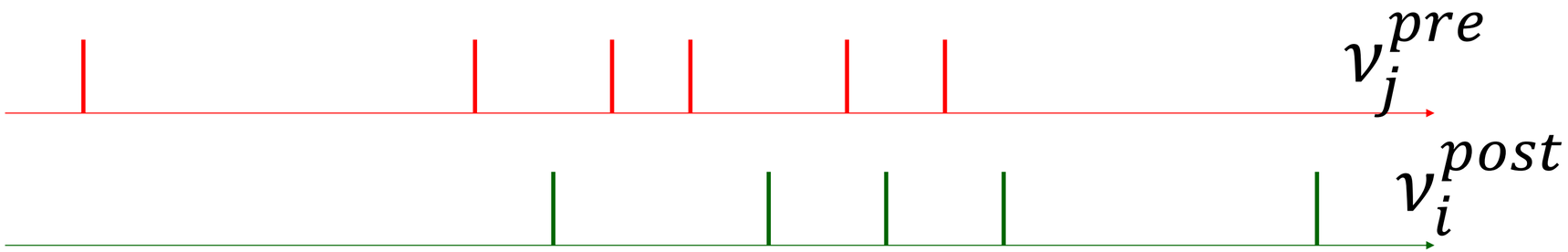
1. Synaptic Plasticity: Learning rules of the brain

2. Hebbian Learning Rules

Previous slide.

Hebbian Learning (rate models)





When an axon of cell **j** repeatedly or persistently takes part in firing cell **i**, then j's efficiency as one of the cells firing i is increased

Hebb, 1949

- local rule
- simultaneously active (correlations)

Rate model:

active = high rate = many spikes per second

→ Continuous real-valued variables $\nu_j^{pre}, \nu_i^{post}$

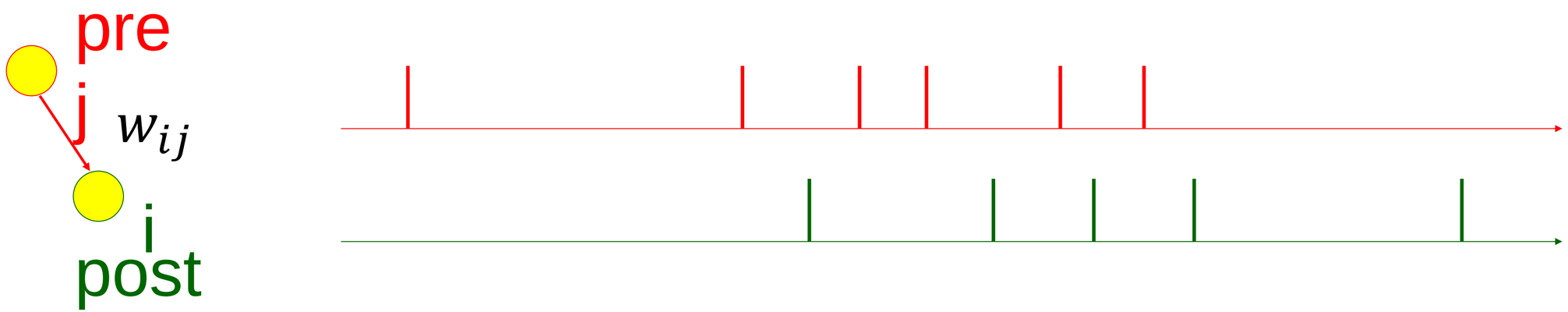
Previous slide.
In this section we consider synaptic plasticity in a rate model. In a rate model we do not describe single spikes but only the ‘rate’ of spike arrival. The rate ν is a continuous variable.

We focus on the directed connection from a neuron j to a neuron i. The connection point is called the synapse. It’s strength is described by the weight Δw_{ij} .
Neuron j and i are called the presynaptic and postsynaptic neuron, respectively.

Donald Hebb proposed a rule of synaptic changes in form of a written statement. The essence is:

(i) only the activity of the presynaptic neuron j and the state of the postsynaptic neuron i should matter for the change of the connection from j to i . (but not that of another neuron k. Hence the rule only uses locally available information.
(ii) Both neurons should be active to generate an increase in the synapse.

Rate-based Hebbian Learning



Local rule:

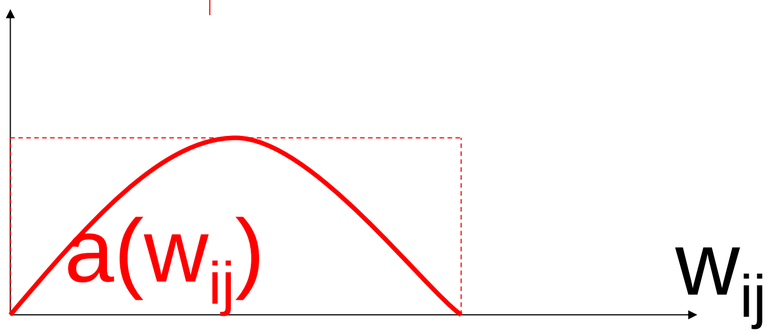
$$\Delta w_{ij} = F(w_{ij}, MOD; \nu_j^{pre}, \nu_i^{post})$$

Modulator MOD=const

Taylor expansion:

$$\Delta w_{ij} = a_0 + a_1^{pre}\nu_j^{pre} + a_1^{post}\nu_i^{post} + a_2^{corr}\nu_j^{pre}\nu_i^{post} + a_2^{post}\left(\nu_i^{post}\right)^2 + a_2^{pre}\left(\nu_i^{pre}\right)^2 \ldots$$

$a = a(w_{ij})$

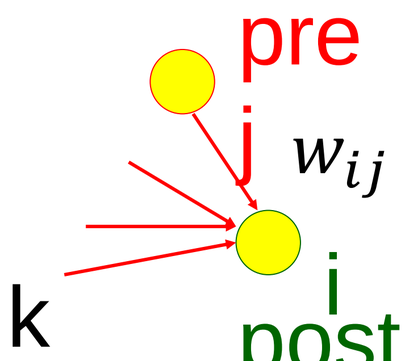


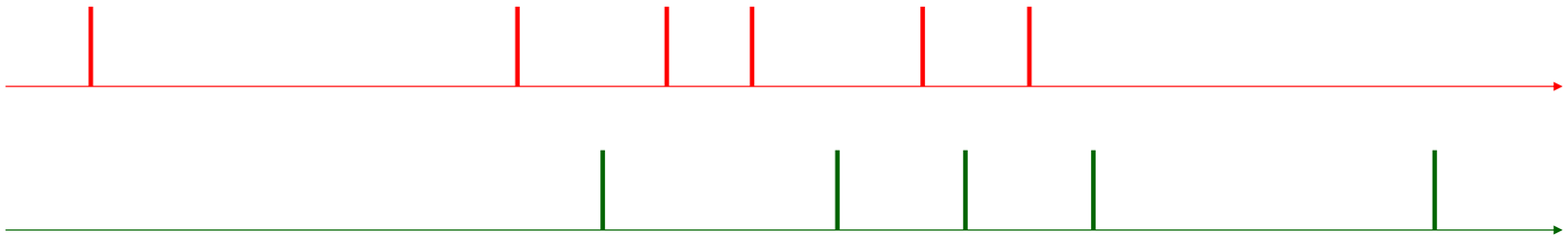
Previous slide.
Let us formulate these insights mathematically.

(i) Local rule implies that the weight change Δw_{ij} depends explicitly only on th firing rate of the pre- and postsynaptic neuron. It can also depend on the momentary value of the synaptic weight w_{ij} itself. Finally, it could also depend on other factors, for example on the presence or absence of a neuromodulator such as dopamine, called *MOD*. At the moment we assume that the value of *MOD* does not change so that we can disregard it.

(ii) The Hebbian rule says little about the function F. We assume that F allows a Taylor expansion. We expand F with respect to the two firing rates, but not with respect to the weight value itself. As a result we have expansion coefficients that still carry the weight-dependence as an argument.

Rate-based Hebbian Learning





$$\Delta w_{ij} = a_2^{corr} \nu_j^{pre} \nu_i^{post}$$

on	off	on	off
on	on	off	off
+	0	0	0

Previous slide.
Let us now focus on some terms in the expansion.
We first focus on the ‘correlation’ between presynaptic and postsynaptic rate and set all other coefficients in the expansion to zero.

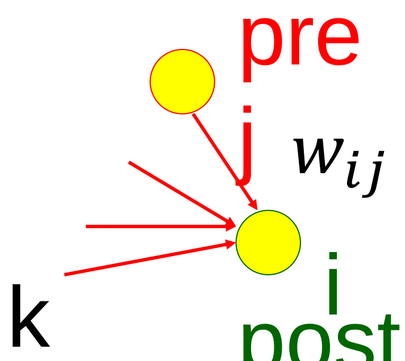
To keep things simple, we assume that the firing rates are either high (say 20Hz) or zero. In the first case, we say that the neuron is ‘on’; in the second case that it is ‘off’.

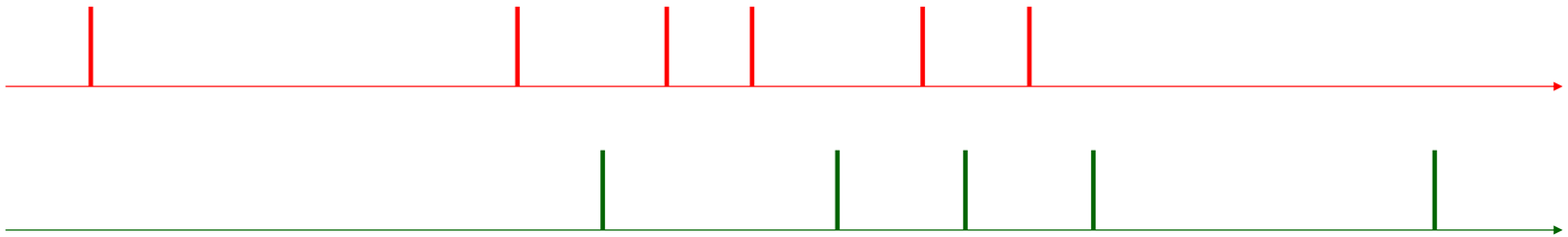
We can then construct a table of four different combinations of on and off.

For this first correlation-based rule we find that a synapse either increases or does not change.

This looks potentially problematic bit then a synapse could never increase!

2. Rate-based Hebbian Learning





$$\Delta w_{ij} = a_2^{corr} \nu_j^{pre} \nu_i^{post}$$

$$\Delta w_{ij} = a_2^{corr} \nu_j^{pre} \nu_i^{post} - c$$

$$\Delta w_{ij} = a_2^{corr} \nu_j^{pre} (\nu_i^{post} - \vartheta)$$

$$\Delta w_{ij} = a_2^{corr} (\nu_j^{pre} - \vartheta)(\nu_i^{post} - \vartheta)$$

on	off	on	off
on	on	off	off
+	0	0	0
+	-	-	-
+	0	-	0
+	-	-	+

Previous slide.
In the second line of the table, we now use a negative expansion coefficient $a_0 = -c < 0$.

In the third line of the table we use instead $a_1^{pre} = -\vartheta a_2^{corr} < 0$.

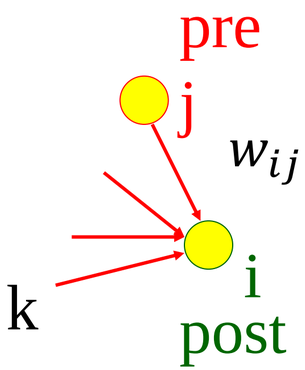
In the fourth line we combine terms a_0 , a_1^{pre}, a_1^{post} , a_2^{corr} in a specific form

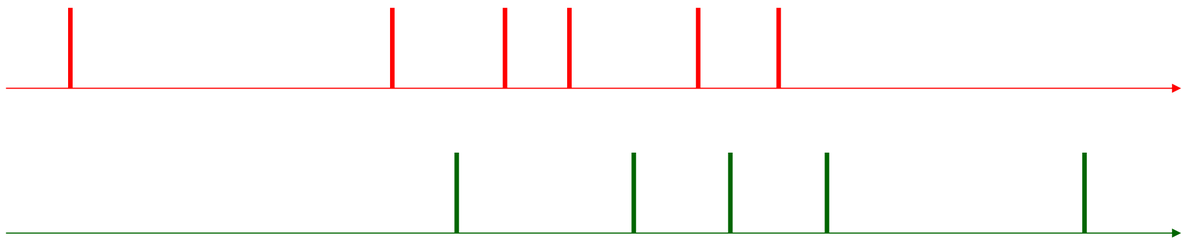
All four examples are Hebbian rules. Indeed, in all four examples the joint activity of pre- and postsynaptic neuron leads to a positive weight change.

Therefore, there are many different Hebbian rules! Experimental data should be used to decide which of these rules (if any) is implement in the brain.

As computer scientists or theoreticians we can also use simulations and mathematical arguments to find out which one of these four Hebbian rules will ‘work’ or is the ‘best’ in achieving good learning results – in ways that we need to define.

. Presynaptically gated plasticity rule

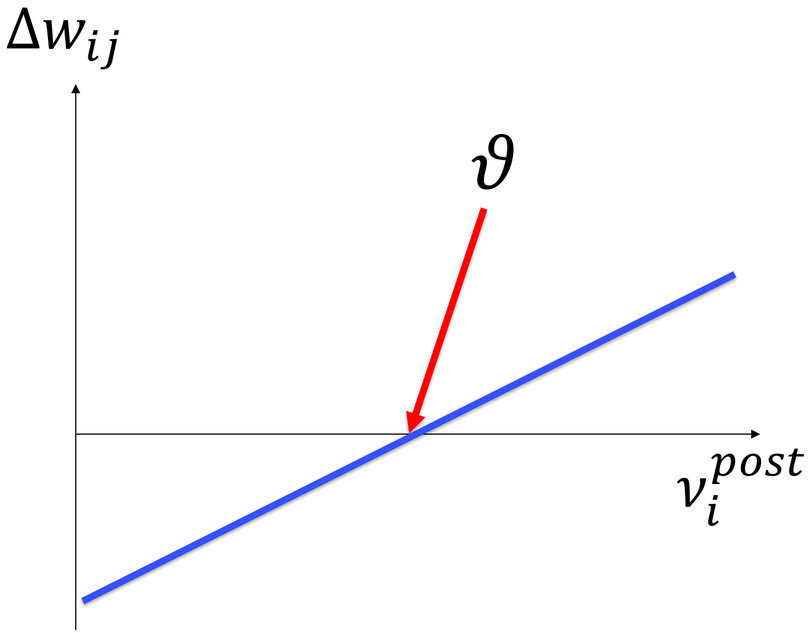




presynaptically gated

$$\Delta w_{ij} = a_2^{corr}(\nu_i^{post} - \vartheta)\nu_j^{pre}$$

Assume activity $\nu_j^{pre} > 0$



Previous slide (not shown in class).
Let us focus for a moment on the third line of the table and plot the weight change as a function of the postsynaptic rate ν_i^{post}.
Let us assume that the presynaptic activity is nonzero.
The threshold parameter ϑ indicates the critical postsynaptic rate where the weight change switches from negative to positive.

In the absence of presynaptic activity there is no weight change. Therefore we call this the presynaptically gated plasticity rule.

Bienenstock-Cooper-Munro rule and Oja rule

BCM: 3rd order (‘triplet’)

$$\Delta w_{ij} = b(v_i^{post})^2 v_i^{pre} - b\vartheta v_i^{post} v_j^{pre}$$

triplet

pair

Bienenstock, Cooper Munro, 1982

Oja: self-normalizing

$$\Delta w_{ij} = \eta\, v_i^{post} v_j^{pre} - c(w_{ij})(v_i^{post})^2$$

Oja, 1982

Hebbian term drives weight increase if both neurons are active

term with minus sign drives weight decrease

Previous slide.
A variant of the presynaptically gated plasticity rule is the BCM rule, named after Bienenstock, Cooper, and Munro, the authors of a paper in 1982.
The two main differences to the presynaptically gated rule are that

(i) It contains an extra rate factor ν_i^{post}.

(ii) The threshold value ϑ is taken to be a function of the low-pass-filtered rate ν_i^{post}. This leads a so-called ‘sliding threshold’.

If we neglect the sliding threshold, the BCM rule can be mapped directly to the Taylor expansion.

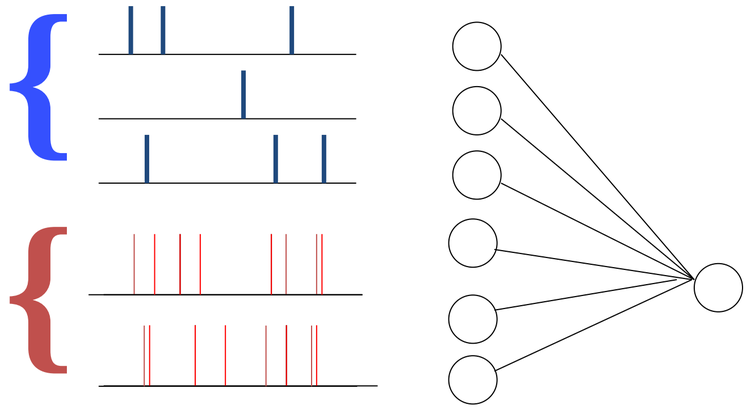
The BCM rule is another example of a Hebbian rule!

And the Oja rule as well. We will come back to the Oja rule later in this lecture.

Functional Consequence of Hebbian Learning

Fixed rate

Jointly variing rate



Hebbian Learning detects correlations in the input

→ Development of Receptive Fields ('filters')

Example: input natural images →

Image: Brito&Gerstner 2016

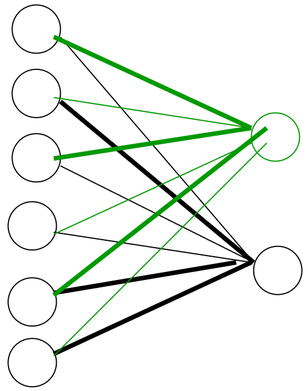
Previous slide.
How can a Hebbian rule be useful?
Suppose a single postsynaptic neuron receives input from several presynaptic neurons.
Let us assume that the top half of the presynaptic neurons just send random spikes whereas the lower half of the presynaptic neuron send highly correlated spikes, because the spike rates of this second group increase and decreases together.

In this case a useful Hebbian learning rule will strengthen the synapses that see the jointly varying firing rate and set the other synapses to very small values or even zero.
Loosely speaking, a good Hebbian learning rule will focus on the parts of input with ‘interestingly correlated’ signals.

For example, if the input consists of small patches of natural images, then the learning rule will develop specific two-dimensional filters that are adapted to the image statistics. In neuroscience, these filters are called ‘receptive fields’

Synaptic Changes for Development of Cortex

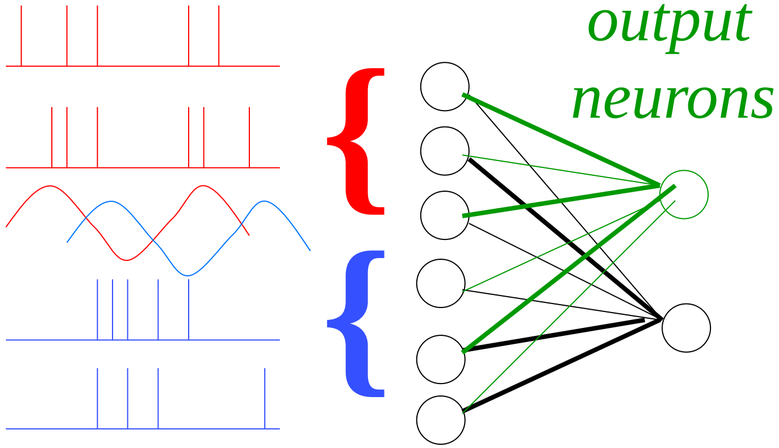
Initial:
random
connections



unselective
neurons

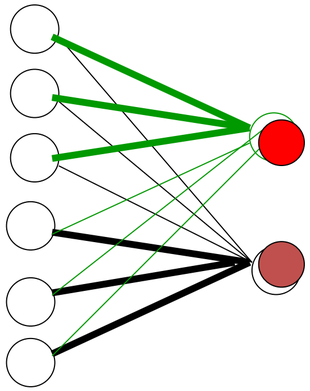
BCM leads to specialized
Neurons (developmental learning);
Bienenstock et al. 1982

Development and learning rules:
Willshaw&Malsburg, 1976
Linsker, 1986
K.D. Miller et al., 1989



output
neurons

Correlated input



output neurons specialize:
→Receptive fields

Previous slide.
The BCM rule from 1982 is an example of such a ‘good’ Hebbian rule.
Willshaw and von der Malsburg formulated in 1976 a slightly different mathematical learning rule; since 1986 dozens of scientific papers have explored Hebbian learning rules that lead to nice receptive fields.

Summary: Models for Hebbian Learning

- A family of ‘Hebbian’ rules
- Considered as models for LTP and LTD
- BCM and Oja rule are well-known examples
- Competition: some synapses grow (i.e., show LTP) at the expense of others (which show LTD
- Detect correlated inputs → relation to PCA?

Previous slide.

- Hebbian learning refers to a **family of learning rules**, rather than one specific rule.

- Rules can be classified by mapping them to a **Taylor explansion**.

- Terms with a negative coefficient induce **Long-Term Depression (LTD).**

- A clever combination of LTP and LTD can explain the **development of receptive fields (RF) in Visual Cortex =** filters that are typical for the first layer in a deep convolutional network trained on images.

- A clever combination of LTP and LTD leads to synaptic **competition: some synapses grow at the expense of others.** A well-known example of a Hebbian rule is the Bienenstock-Cooper-Munro (BCM) rule.

Learning in Neural Networks: Lecture 1

Hebbian Learning rules

Wulfram Gerstner

EPFL, Lausanne, Switzerland

1. Synaptic Plasticity: Learning rules of the brain

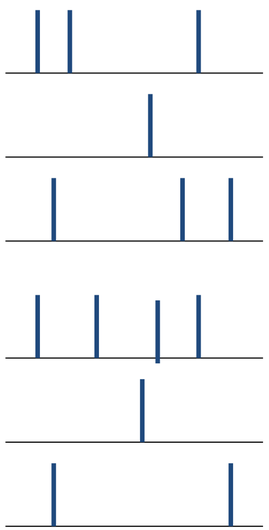
2. Hebbian Learning Rules

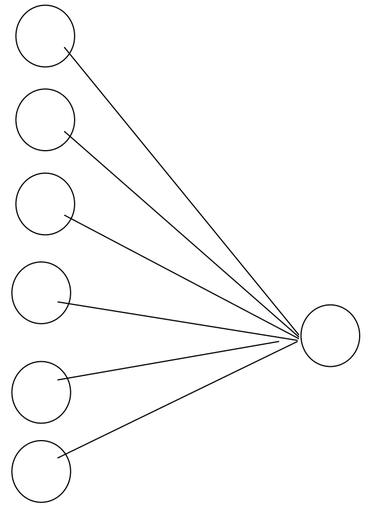
3. PCA as Hebbian Learning

Previous slide.
In this section we want to mathematically formalize the consequences of Hebbian Learning in a first oversimplified model.

Hebbian Learning for PCA: notation

$$\vec{x} = \begin{pmatrix} \nu_1^{pre} \\ \nu_2^{pre} \\ \nu_3^{pre} \\ \nu_j^{pre} \\ \dots \\ \nu_N^{pre} \end{pmatrix}$$





Vector of
input rates

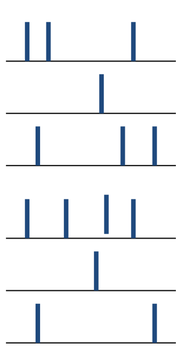
- discrete time
- sequential presentation
of input patterns

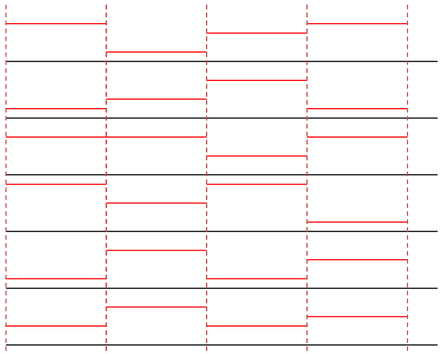
$$\vec{x}^1 = \begin{pmatrix} \nu_1^1 \\ \nu_2^1 \\ \nu_3^1 \\ \nu_j^1 \\ \dots \\ \nu_N^1 \end{pmatrix} \qquad \vec{x}^\mu = \begin{pmatrix} \nu_1^\mu \\ \nu_2^\mu \\ \nu_3^\mu \\ \nu_j^\mu \\ \dots \\ \nu_N^\mu \end{pmatrix}$$

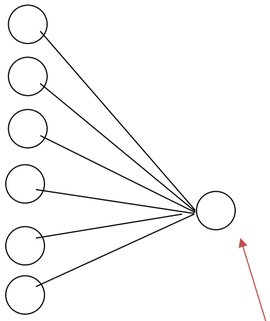
Previous slide.
We consider a single postsynaptic neuron driven by input from N presynaptic neurons.
Presynaptic inputs arise from input patterns $\vec{x}^{\mu}$.

- We work in discrete time. In each time step of duration Δt we apply one of the patterns. Patterns are selected either randomly or one of the other for several cycles.

Hebbian Learning for PCA







Linear neuron model

$$\nu_i^{post} = \sum_k w_{ik}\, \nu_k^{pre}$$

P input patterns

$$\{\vec{x}^{\mu}; 1 \le \mu \le P\}$$

$$\vec{x}^{\mu} = \begin{pmatrix} \nu_1^{\mu} \\ \nu_2^{\mu} \\ \nu_3^{\mu} \\ \nu_j^{\mu} \\ \dots \\ \nu_N^{\mu} \end{pmatrix}$$

Theorem: ASSUME

- linear neuron model
- small learning rate $\eta = a_2^{corr}$
- $\Delta w_{ij} = \eta\, \nu_j^{pre}\, \nu_i^{post}$
- $\frac{1}{P} \sum_{\mu=1}^{P} \vec{x}^{\mu} = 0$
- patterns presented sequentially several times

THEN

the weight vector aligns with principal eigenvector of correlation matrix.

Blackboard

Previous slide.
The specific assumptions are

- We assume that the patterns have mean zero (normally firing rates should be positive but we ignore this for the moment.
- The simplest Hebbian learning rule
- A linear neuron model.
- A very small learning rate (or time steps going to zero)

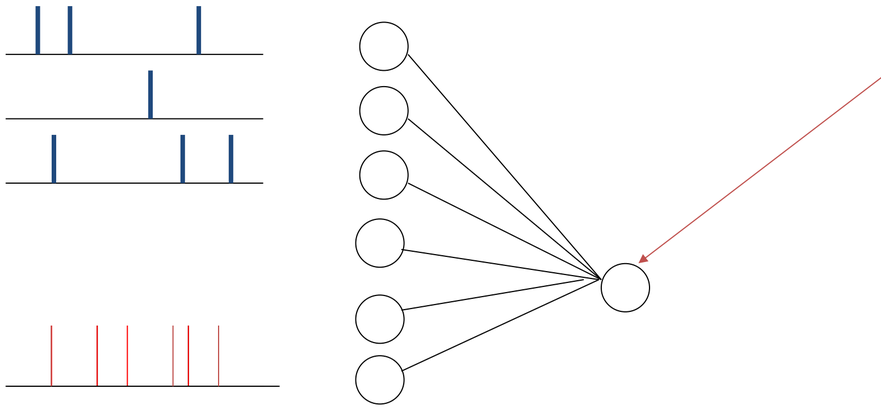
A blackboard calculation then yields that the weight vector converges towards the eigenvector of the correlation matrix.

In theory this result is valid only for the expected weight change. However, for small learning rate, weights hardly change during one time step. Therefore, before a significant change is implemented all patterns have been applied several times as inputs. In this limit the actual weight change becomes very close to the expected weight change.

Analysis: Hebbian Learning for PCA

Fixed rate

Jointly variing rate



$$\nu_i^{post} = \sum_k w_{ik}\, \nu_k^{pre}$$

linear neuron model

Detects correlations in the input

learning rule (Hebb)

$$\Delta w_{ij} = a_2^{corr} \nu_i^{post} \nu_j^{pre}$$

linear neuron

$$\Delta w_{ij} = a_2^{corr} \left[\sum_k w_{ik}\, \nu_k^{pre} \right] \nu_j^{pre}$$

expected weight change

$$E[\Delta w_{ij}] = a_2^{corr} \sum_k w_{ik}\, E[\nu_k^{pre} \nu_j^{pre}]$$

correlation matrix

$$C_{kj}$$

Previous slide.
After inserting the equation of the linear neuron into the learning rule, the correlation matrix C appears naturally on the right-hand side.

We then write the expansion coefficient $a_2^{corr} = \gamma \Delta t$.
This expresses that during longer presentation times weight changes are longer.

Then we take the limit $\Delta t \to 0$ and find a differential equation for the weight vector $\vec{w}_i = (w_{i1}, w_{i2}, \ldots, w_{iN})$ which we express in the coordinate system of the eigenvectors of C.

The largest eigenvalue dominates the dynamics, and therefore we conclude that $\vec{w}_i$ aligns with $\vec{e}_1$.

However, we also not that the length of $\vec{w}_i$ explodes.

Hebbian Learning for PCA

$$C_{kj} = E[x_k x_j] = \frac{1}{P}\sum_{\mu=1}^{P} x_k^\mu x_j^\mu$$

correlation

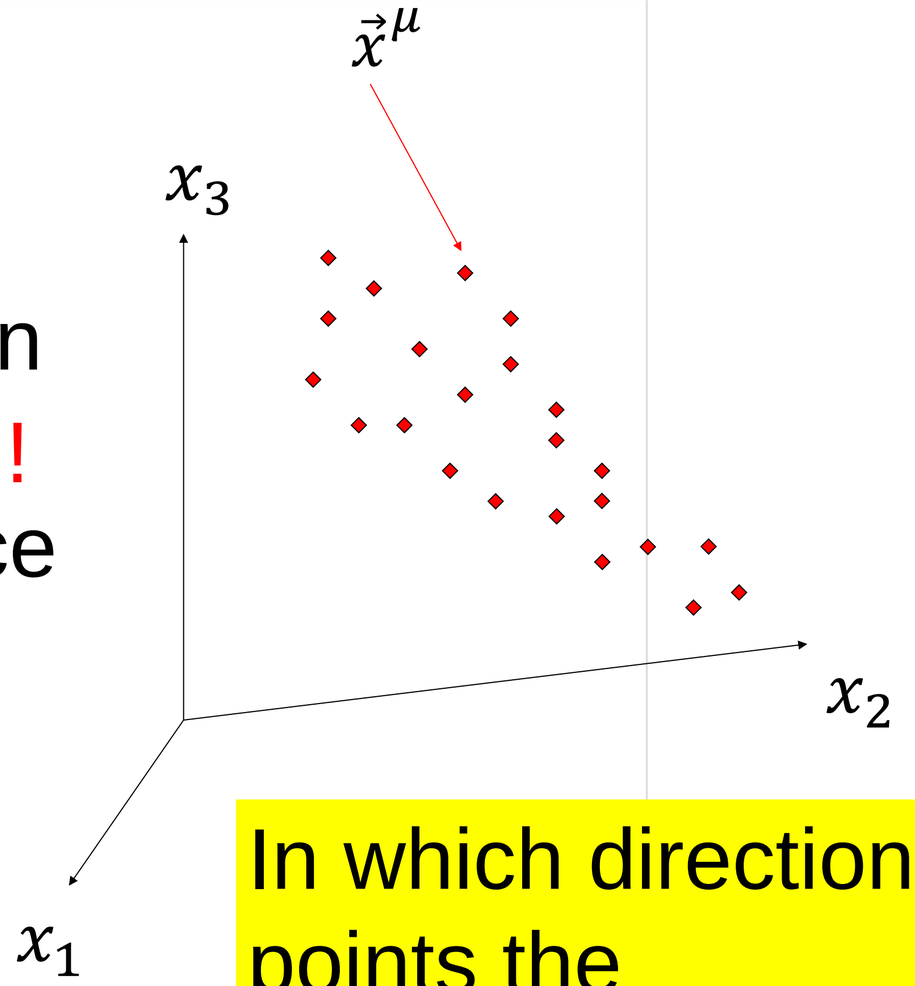
$C \neq C^0$!

$$C_{kj}^0 = E[(x_k - \langle x_k \rangle)(x_j - \langle x_j \rangle)]$$

covariance

$$C^0 \vec{e}^n = \lambda^n \vec{e}^n$$

$$\lambda^1 > \lambda^2 > \ldots > \lambda^N$$



In which direction points the eigenvector of C ?

$\vec{e}^1$ (eigenvector of C^0 with maximal eigenvalue) is called the Principal Component

Hebbian learning → weight vector $\vec{w}$ aligns with $\vec{e}^1$

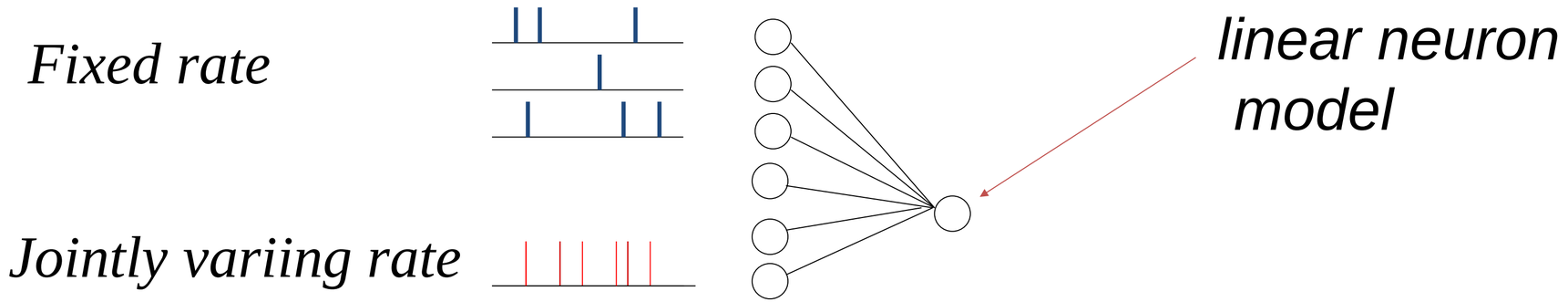
Previous slide.

If we drop the assumption that the input patterns have zero mean, the correlation matrix is different from the covariance matrix.

Standard PCA assumes that the mean is zero!

Zero-mean input is therefore an important assumption of the theorem.

Summary: Hebbian Learning for PCA



- **Hebbian learning detects correlations in the input**
- **with linear neuron model and simple plasticity**

$$\frac{d}{dt} w_{ij} = a_2^{corr} \nu_i^{post} \nu_j^{pre}$$

Hebbian learning aligns weight vector with first PC of correlation matrix

- **BUT: length of weight vector explodes**

Previous slide.

Oja rule: detects first principal comp.

$$\frac{d}{dt}w_{ij} = a_2^{corr} v_j^{pre} v_i^{post} - a_2^{corr} w_{ij} (v_i^{post})^2$$

$$v_i^{post} = \sum_k w_{ik} v_k^{pre} \quad (1)$$

$$E[\frac{d}{dt}w_{ij}] = a_2^{corr} \sum_k w_{ik} \underbrace{E[v_k^{pre} v_j^{pre}]}_{C_{kj}} - w_{ij} \sum_{k,n} w_{ik} \underbrace{E[v_k^{pre} v_j^{pre}]}_{C_{kn}} w_{in}$$

$$E[\frac{d}{dt}\vec{w}] = a_2^{corr} [C\vec{w} - (\vec{w}^T C \vec{w})\vec{w}]$$

scalar!

Converges to fixed point: $0 = C\vec{w} - \lambda\vec{w}$

Only $\vec{w} = \vec{e}_1$ is stable

Exercise: The weight vector stays normalized and parallel to first PC!

E. Oja, Simplified Neuron Model as Principal Component Analyser, J. Math. Biol. 15:267-263 (1982)

Previous slide.
The analysis from the Blackboard can be directly transferred to that of the Oja rule.
The Oja rule controls weight growth and avoids the explosion of weights.

1) We express the postsynaptic rate by Equation (1) and insert it several times.
2) We study the expected weight change.
3) We switch to vector notation.
4) We analyze the fixed point.

5) The term in the red oval is a scalar number (called λ), since we apply from both sides a vector to the matrix C. This yields an Eigenvalue equation for the fixed point.

6) The result of the analysis (Exercises) is that the weight vector asymptotically approaches the first principal component. With the choice of parameters given here, the weight vector at the stable fixed point is normalized to a length of one.

Summary Lecture 1

T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton. Backpropagation and the brain. *Nat. Rev. Neurosci.*, 21(6):335–346, 2020. ISSN 14710048. doi: 10.1038/s41583-020-0277-3.

No BackProp, please!

- BackProp not bio-plausible because it needs four separate phases and feedback architecture for vector feedback
- Brain consists of neurons organized in several brain areas
- Learning corresponds to changes in the synapses.
- Synaptic changes follow rules, called learning rules.

Hebbian Learning

- not a single learning rule but a family of rules!
- detects correlations in the input
- is useful to drive the development of receptive fields
- bilinear Hebbian learning rule with linear neuron model detects the first principal component
- The Oja rule is a stable example of Hebbian learning.

E. Oja, Simplified Neuron Model as Principal Component Analyser, J. Math. Biol. 15:267-263 (1982)

Previous slide.
Overall summary of the lecture.

The End

Previous slide.

End of Lecture.

The following slides are an appendix and not covered in class.

1) Review of PCA as a standard method covered in introduction lectures to signal processing or data analysis. First you subtract the mean. Then you calculate the eigenvectors. The eigenvector with the largest eigenvalue is called the principal component.

2) A table of further well-known Hebbian rules.

3) A review of standard BackProp

All three topics are covered in standard textbooks (e.g., Haykin, Neural Networks and Learning machines)

1. Appendix: Review of Standard PCA algorithm

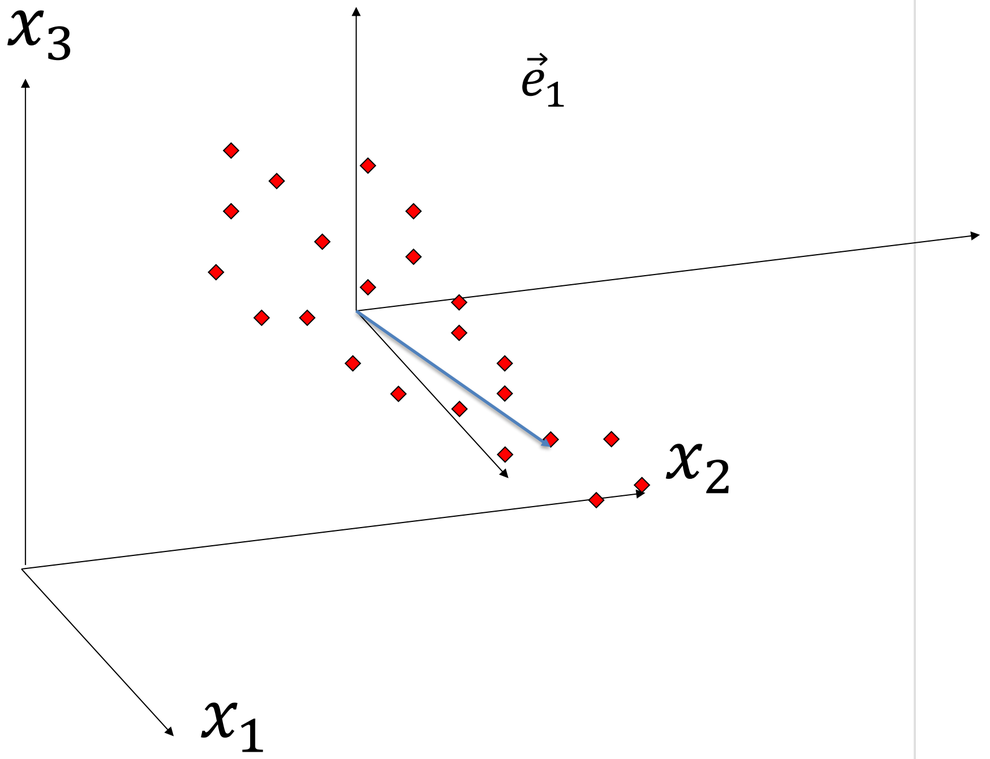
1) Subtract mean $\vec{x} \longrightarrow \vec{x} \quad - \langle \vec{x} \quad \rangle$

2) Calculate covariance matrix

$$C_{kj}^{0} = \left\langle (x_k - \langle x_k \rangle)(x_j - \langle x_j \rangle) \right\rangle$$

3) Calculate eigenvectors

$\vec{e}^{1}$ (eigenvector with maximal eigenvalue) is called the Principal Component

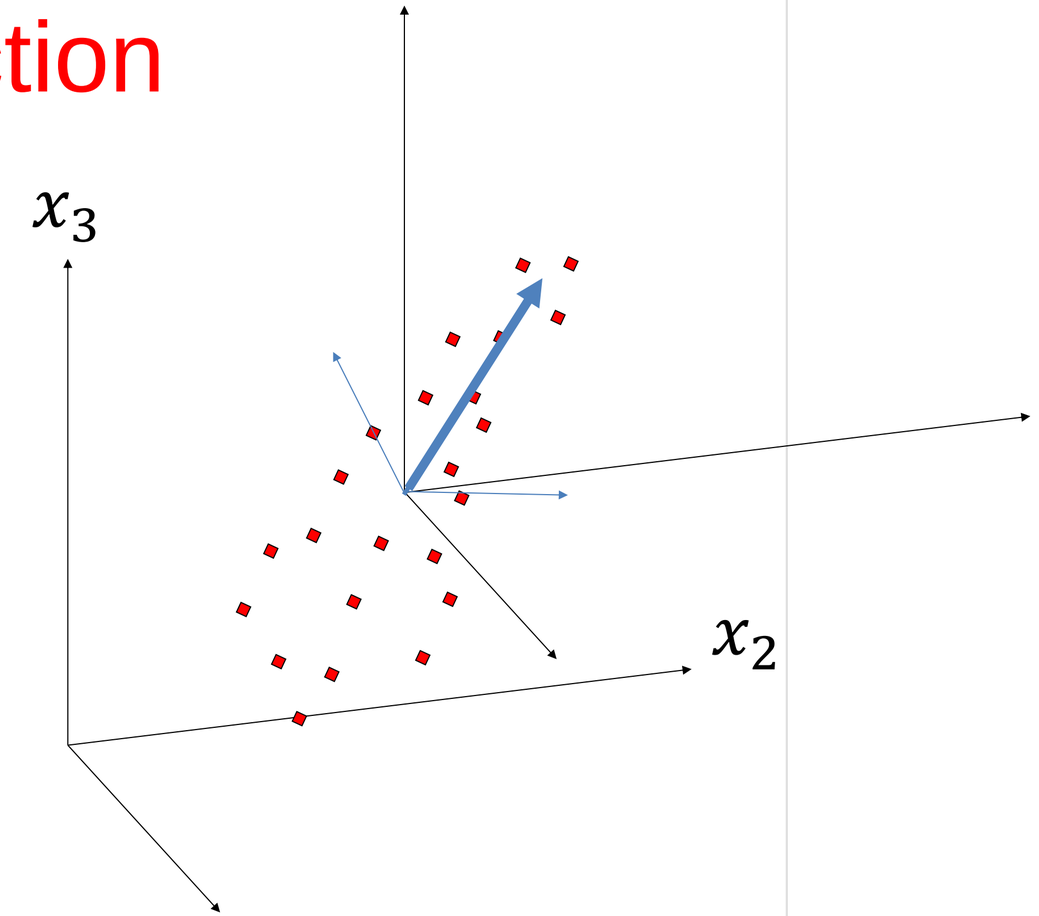


Previous slide.

PCA is a standard method covered in introduction lectures to signal processing or data analysis.

First you subtract the mean. Then you calculate the eigenvectors. The eigenvector with the largest eigenvalue is called the principal component.

PCA for dimension reduction



$$\vec{x}^{\mu} = \begin{pmatrix} x_1^{\mu} \\ x_2^{\mu} \\ \dots \\ x_N^{\mu} \end{pmatrix}$$

Subtract mean

Rotate via PCA

$$\tilde{\vec{x}}^{\mu} = \begin{pmatrix} \tilde{x}_1^{\mu} \\ \tilde{x}_2^{\mu} \\ \dots \\ \tilde{x}_N^{\mu} \end{pmatrix}$$

Keep only

First components

$$\begin{pmatrix} \tilde{x}_1^{\mu} \\ \tilde{x}_2^{\mu} \end{pmatrix}$$

Previous slide.

Preprocessing of data (to zero mean) is equivalent to a shift of the coordinate system.

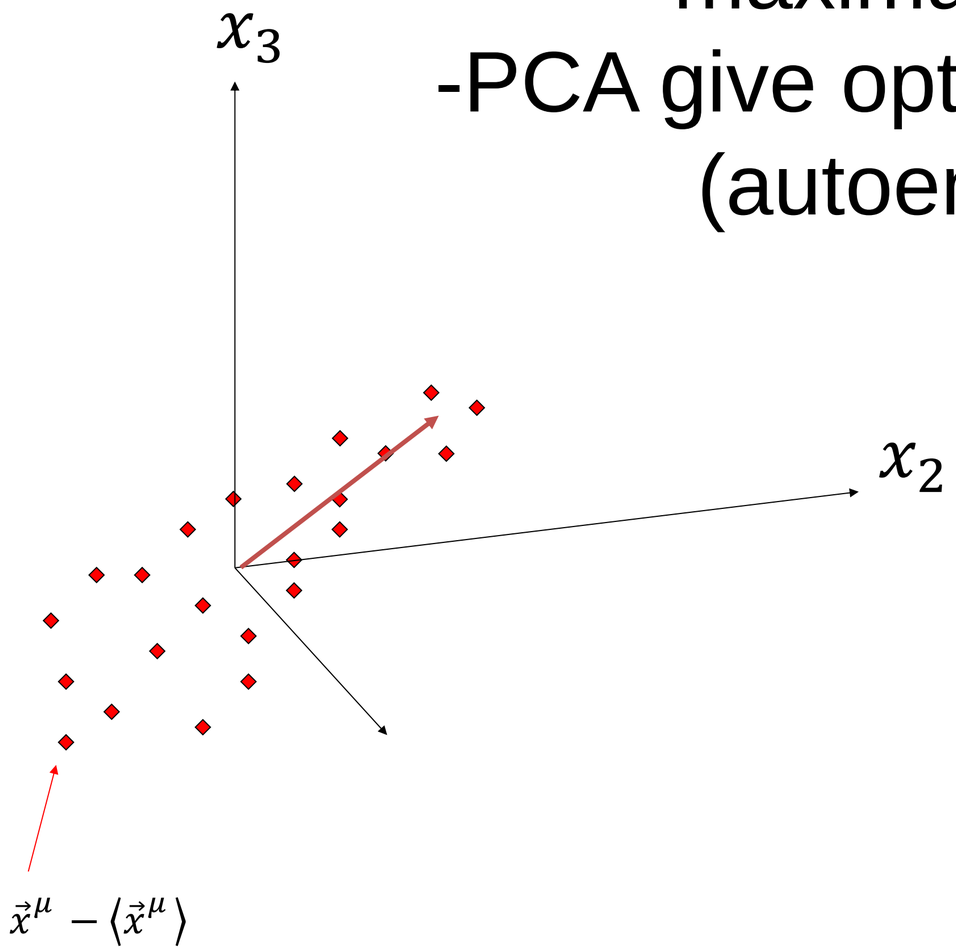
PCA is equivalent to a rotation of the coordinate system (blue axis).

For dimension reduction, one often keeps only the first few components

Review of for PCA

-PCA detects the direction of maximal variance

-PCA give optimal reconstruction (autoencoder)



Maximization principle in Lecture 2: comparison of PCA and ICA

Previous slide.

2. Appendix well-known Hebbian Learning rules:

Linear neuron model, $E[\vec{x}_i] = 0$, and:

simple, bilinear
$$\Delta w_{ij} = \eta\, \nu_j^{pre} \nu_i^{post}$$

Oja 1989
$$\Delta w_{ij} = \eta\, \nu_j^{pre} \nu_i^{post} - c(w_{ij})(\nu_i^{post})^2$$

Covariance, Sejnowski 1977
$$\Delta w_{ij} = \eta(\nu_j^{pre} - \vartheta)(\nu_i^{post} - \vartheta)$$

BCM 1982
$$\Delta w_{ij} = \eta\, \nu_j^{pre} \nu_i^{post}(\nu_i^{post} - \vartheta)$$

pre	on	off	on	off	
post	on	on	off	off	
	+	**0**	**0**	**0**	$\lvert\vec{w}_i\rvert$ *explodes*, PCA
	+	-	**0**	**0**	$\lvert\vec{w}_i\rvert$ *stable* PCA
	+	-	-	+	$\lvert\vec{w}_i\rvert$ *explodes*, PCA
	+	**0**	-	**0**	$\lvert\vec{w}_i\rvert$ *stable* ICA or PCA,

Previous slide.
All these examples are Hebbian learning rules!

The Oja rule provides a simple modification which ensures that the norm of the weight vector is stable.

The covariance rule does not need the condition that the input is zero mean. Otherwise same as simple rule

The BCM rule is nonlinear in the plasticity terms. We will see in the lecture on ICA that nonlinearities generically focus on ICA, but only if the mean input is zero and if all eigenvalues of the correlation matrix are identical.

3. APPENDIX:
Review of Backprop

Wulfram Gerstner
EPFL, Lausanne, Switzerland

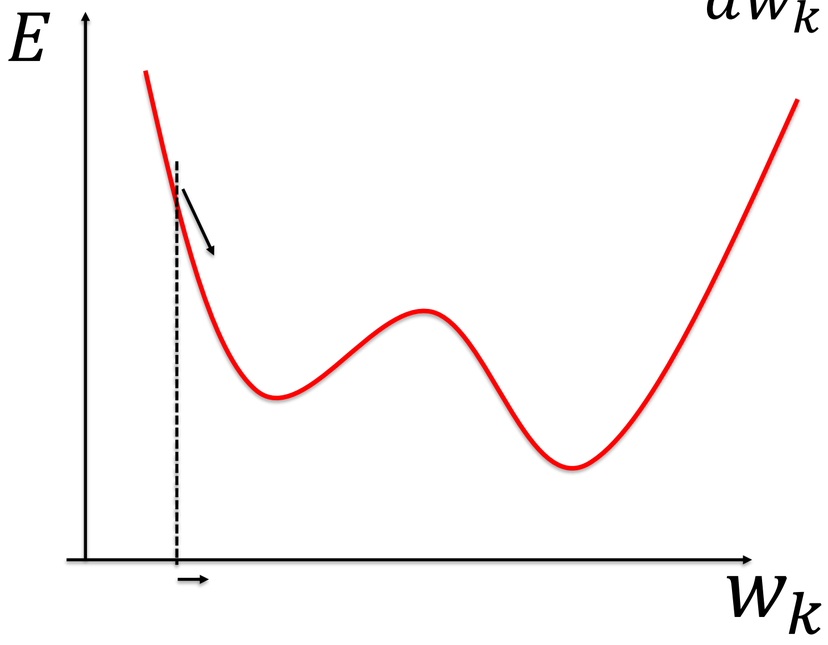
BackProp as Gradient Descent

Modern gradient descent

Some **error function,**
also called **loss function**

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{P}\sum_{i}\left[t_i^{\mu} - \hat{y}_i^{\mu}\right]^2$$

gradient descent $\Delta w_k = -\gamma \frac{dE}{dw_k}$



Batch rule:
one update after all ***P*** **patterns**

(normal gradient descent)

Online rule:
one update after **one pattern**

(stochastic gradient descent)

Mini Batch rule:
one update after ***P'=P/K*** **patterns**

(minibatch update)

1 epoch = all patterns applied once.
Training over many epochs

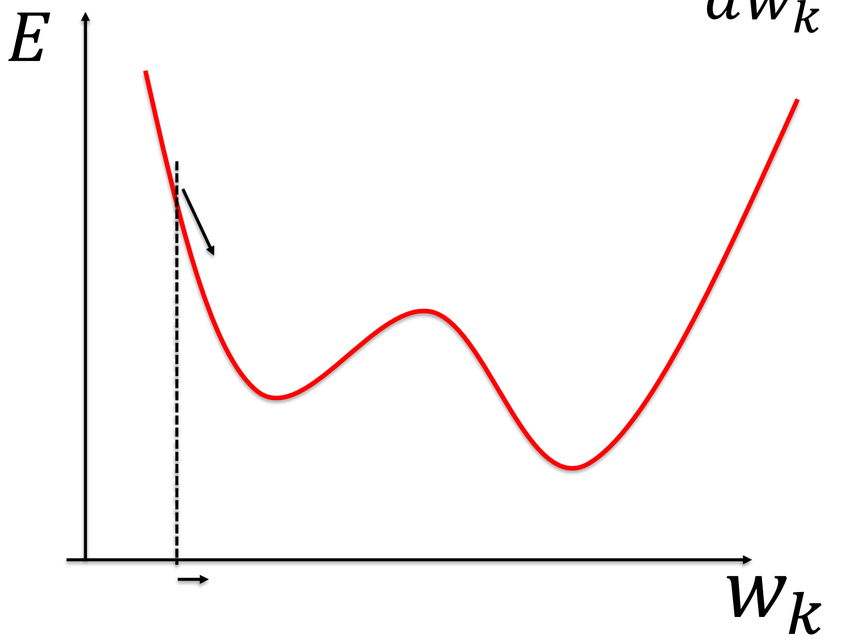
Modern gradient descent

Some **error function,** also called **loss function**

$E(\boldsymbol{w})$

gradient descent

$$\Delta w_k = -\gamma \frac{dE}{dw_k}$$

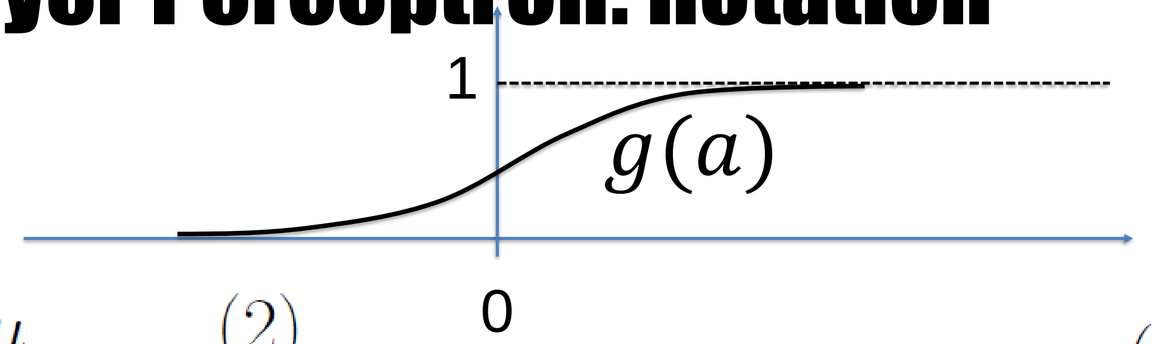


Convergence

- To local minimum
- No guarantee to find global minimum
- Learning rate needs to be sufficiently small
- Learning rate can be further decreased once you are close to convergence

→ See course: *Machine Learning* (Jaggi-Urbanke)

Multilayer Perceptron: notation



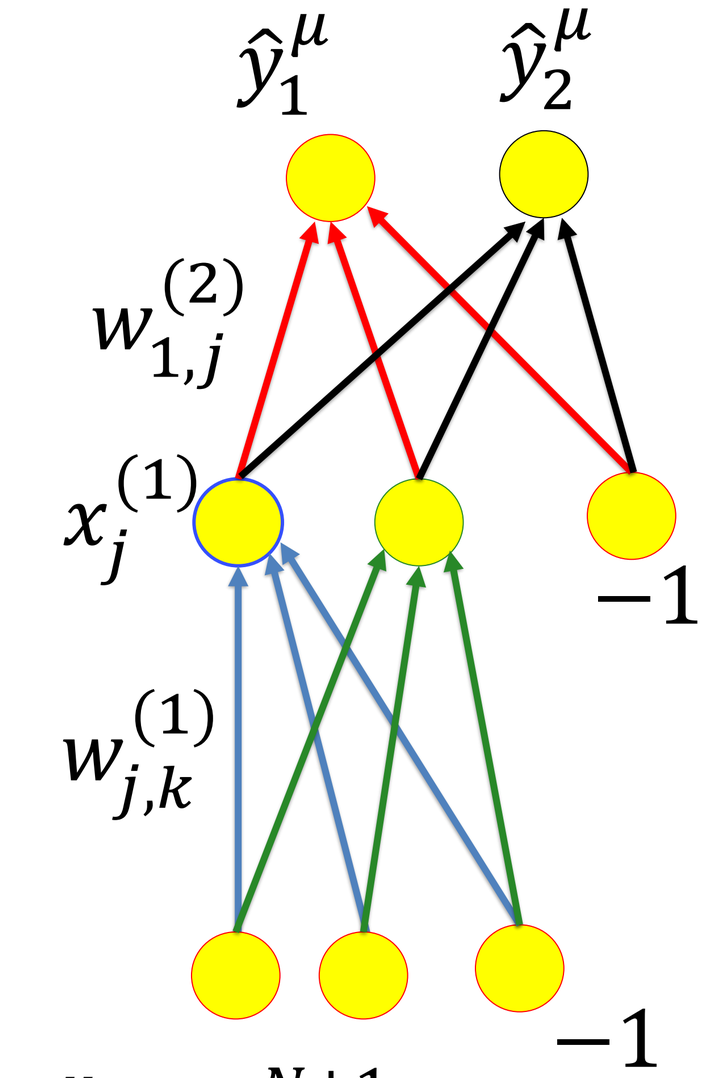
$$\hat{y}_i^{\mu} = x_i^{(2)} \quad (1)$$

$$= g^{(2)}[a_i^{(2)}] \quad (2)$$

$$= g^{(2)}[\sum_j w_{ij}^{(2)} x_j^{(1)}] \quad (3)$$

$$= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(a_j^{(1)})] \quad (4)$$

$$= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(\sum_k w_{jk}^{(1)} x_k^{\mu})] \quad (5)$$



$\boldsymbol{x^{\mu}} \in R^{N+1}$

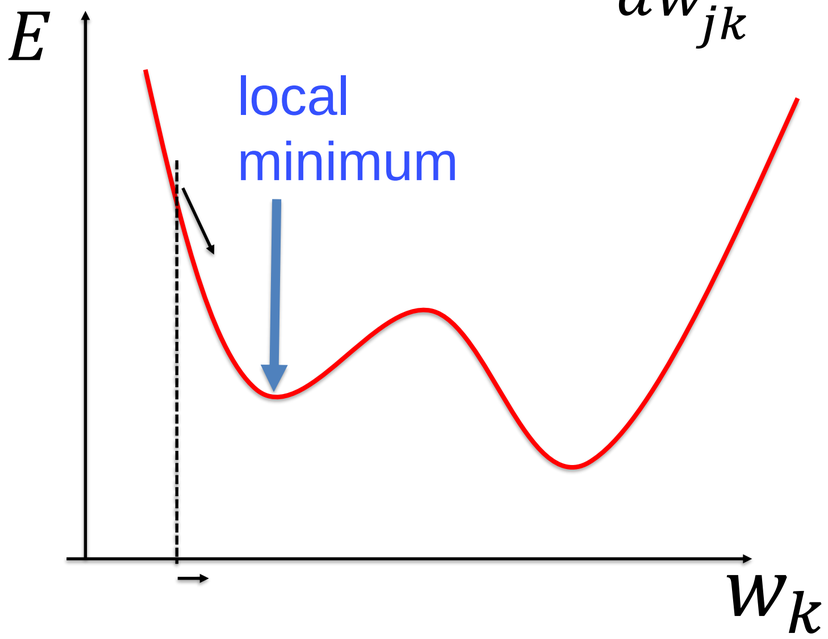
Multilayer Perceptron: gradient descent

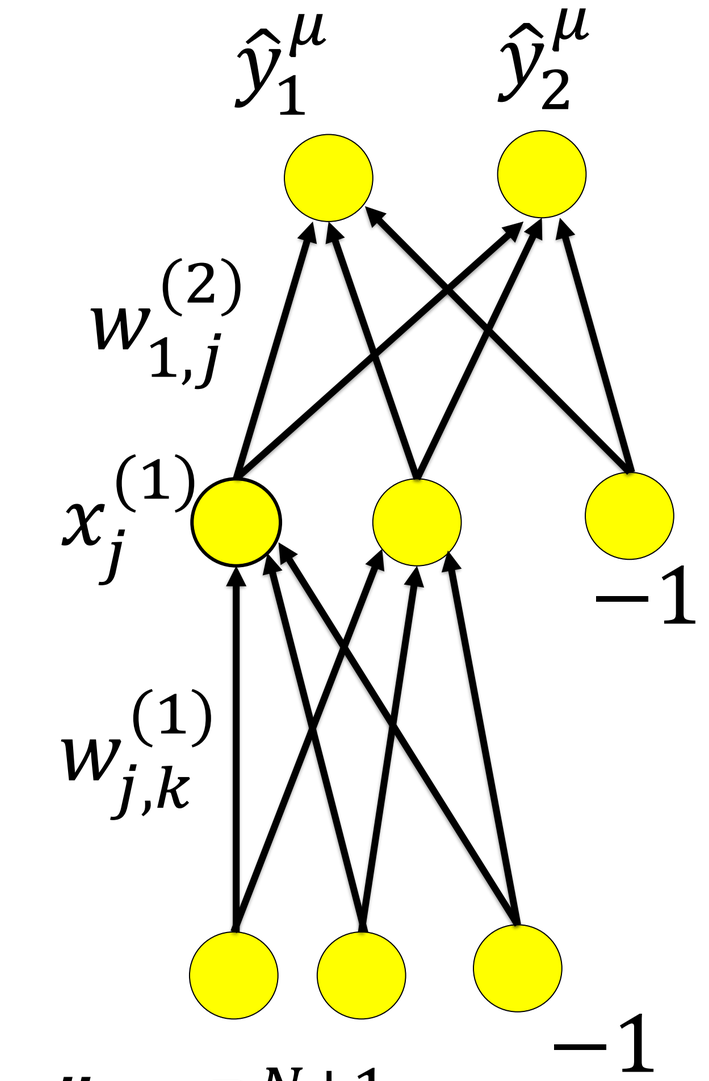
Quadratic **error**

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{P}\sum_{i}\left[t_i^{\mu} - \hat{y}_i^{\mu}\right]^2$$

gradient descent

$$\Delta w_{jk}^{(1)} = -\gamma \frac{dE}{dw_{jk}^{(1)}}$$





$\boldsymbol{x}^{\boldsymbol{\mu}} \in R^{N+1}$

Exercise 1 now: Calculate gradient!
Use Chain rule, be smart!

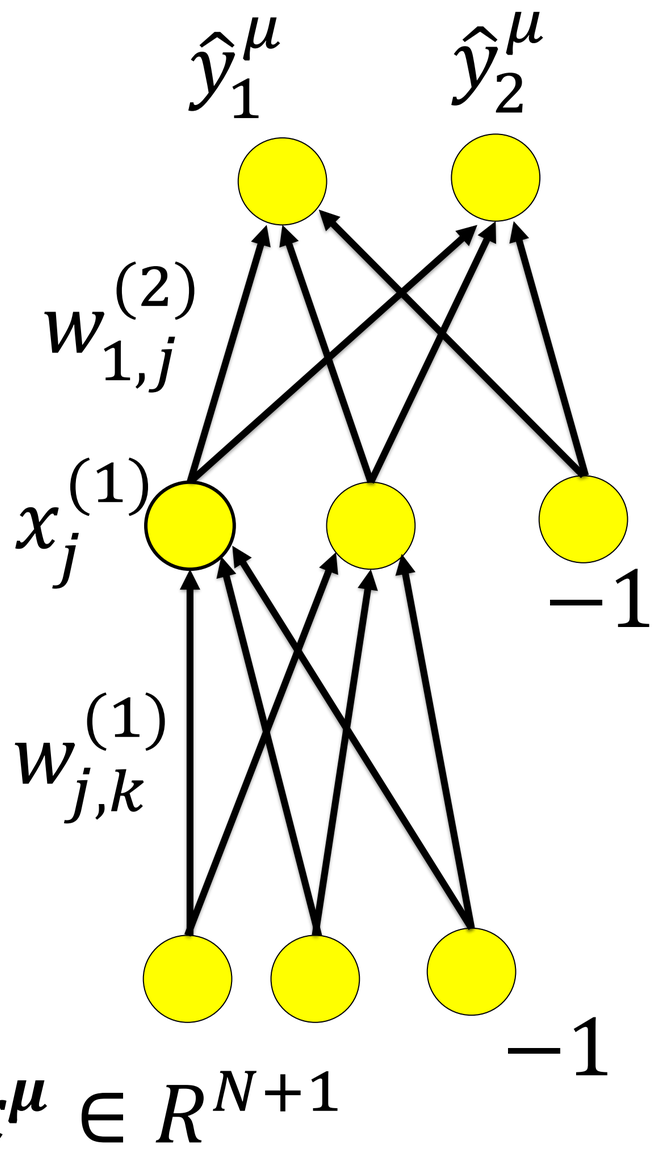
We continue in **8 minutes!**

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{P}\sum_{i}\left[t_i^{\mu} - \hat{y}_i^{\mu}\right]^2$$

$$\Delta w_{jk}^{(1)} = -\gamma \frac{dE}{dw_{jk}^{(1)}}$$

with

$$\hat{y}_i^{\mu} = g^{(2)}\left[\sum_j w_{ij}^{(2)} g^{(1)}\left(\sum_k w_{jk}^{(1)} x_k^{\mu}\right)\right]$$



gradient descent

Calculating a gradient in multi-layer networks:

- write down chain rule
- analyze dependency graph
- store intermediate results
- update intermediate results while proceeding through graph

compare with dynamic programming

- update all weights together at the end

BackProp

0. **Initialization of weights**

1. **Choose pattern** $\mathbf{x}^\mu$

   input $x_k^{(0)} = x_k^\mu$

2. **Forward propagation of signals** $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \qquad (1)$$

   output $\hat{y}_i^\mu = x_i^{(n_{\max})}$

3. **Computation of errors in output**

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^\mu - \hat{y}^\mu] \qquad (2)$$

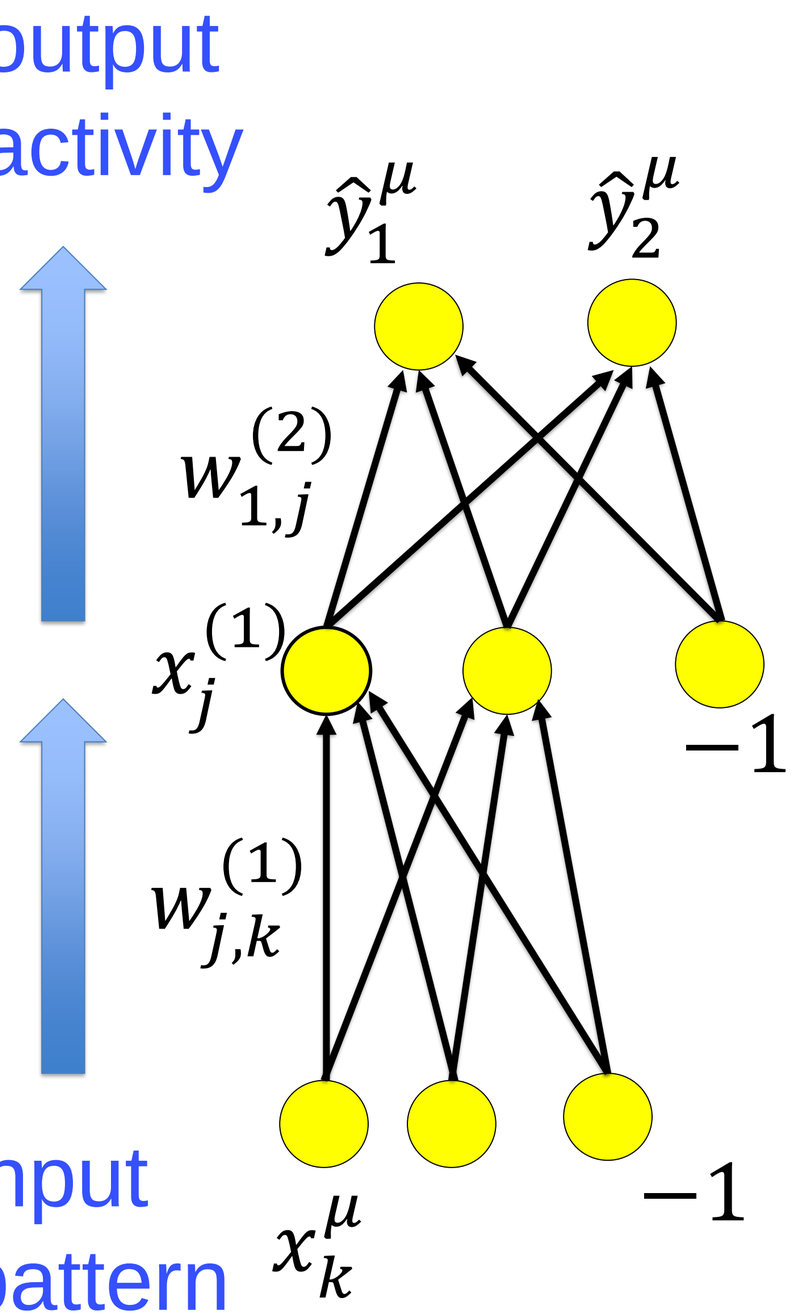
4. **Backward propagation of errors** $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \qquad (3)$$

5. **Update weights (for each** (i,j) **and all layers** (n)**)**

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \qquad (4)$$

6. **Return to step 1.**



BackProp

0. **Initialization of weights**

1. **Choose pattern** $\mathbf{x}^\mu$

   input $x_k^{(0)} = x_k^\mu$

2. **Forward propagation of signals** $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \qquad (1)$$

   output $\hat{y}_i^\mu = x_i^{(n_{\max})}$

3. **Computation of errors in output**

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^\mu - \hat{y}^\mu] \qquad (2)$$

4. **Backward propagation of errors** $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

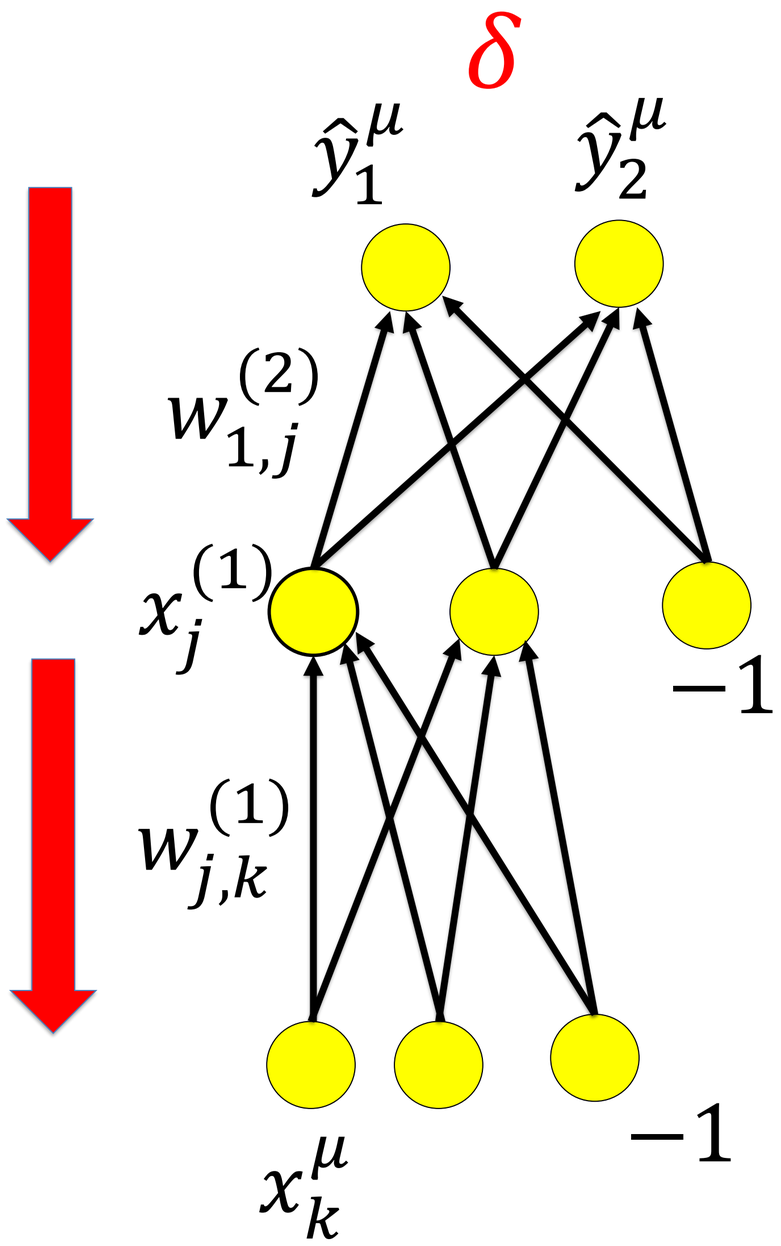
$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \qquad (3)$$

5. **Update weights (for each** (i,j) **and all layers** (n)**)**

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \qquad (4)$$

6. **Return to step 1.**

Calculate output error



BackProp

0. **Initialization of weights**

1. **Choose pattern** $\mathbf{x}^\mu$

   input $x_k^{(0)} = x_k^\mu$

2. **Forward propagation of signals** $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \qquad (1)$$

   output $\hat{y}_i^\mu = x_i^{(n_{\max})}$

3. **Computation of errors in output**

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^\mu - \hat{y}^\mu] \qquad (2)$$

4. **Backward propagation of errors** $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \qquad (3)$$

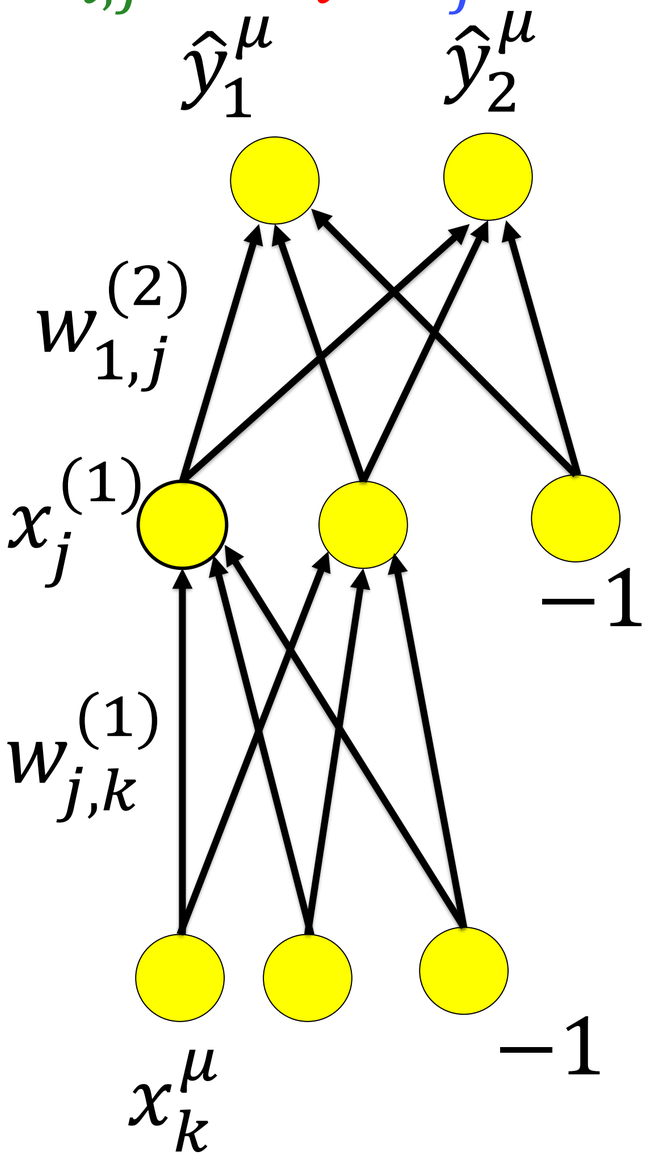
5. **Update weights (for each** (i,j) **and all layers** (n)**)**

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \qquad (4)$$

6. **Return to step 1.**

update **all** weights

$$\Delta w_{i,j}^{(n)} = \delta_i^{(n)} x_j^{(n-1)}$$



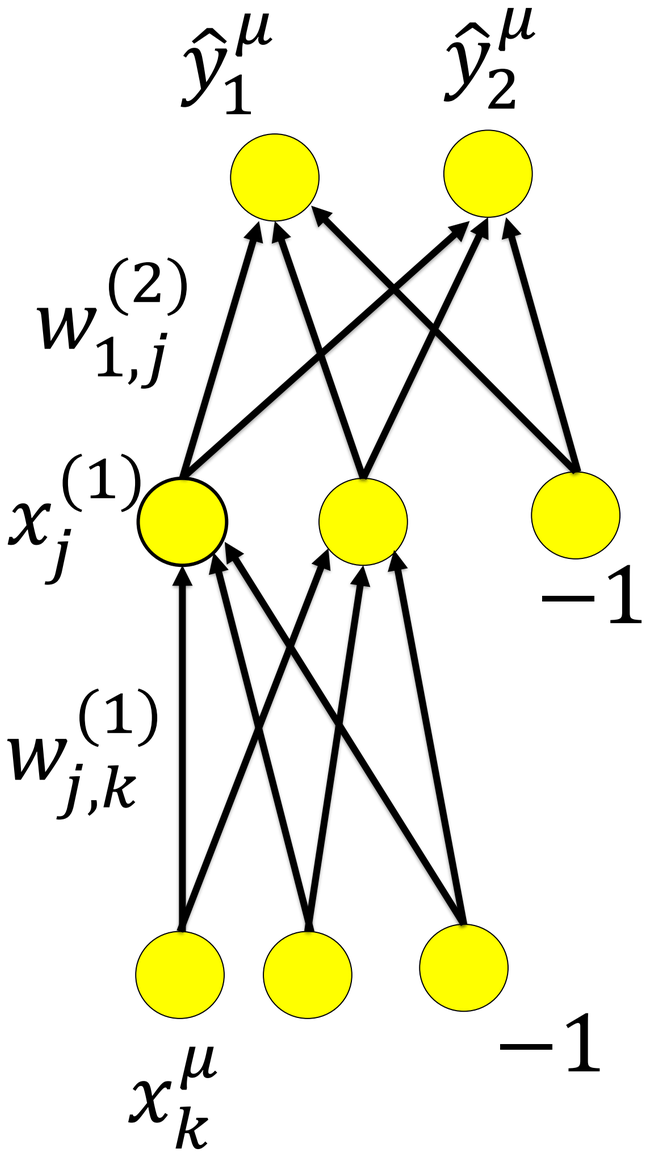
Backprop versus direct numerical evaluation

$$\Delta w_{jk}^{(1)} = -\gamma \frac{dE}{dw_{jk}^{(1)}}$$

$$= -\gamma \frac{E\left(w_{jk}^{(1)} + \varepsilon\right) - E\left(w_{jk}^{(1)} - \varepsilon\right)}{2\varepsilon}$$

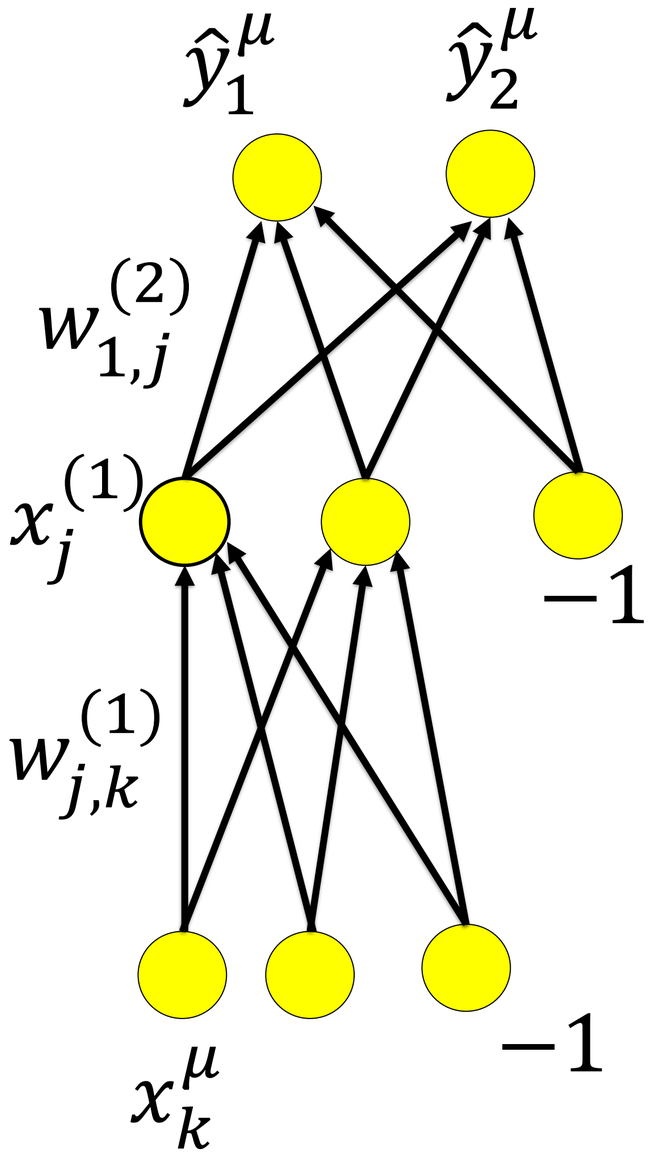
calculate $E(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{P}\sum_{i}\left[t_i^{\mu} - \hat{y}_i^{\mu}\right]^2$

→ calculate $\hat{y}_i^{\mu}$ for one pattern



Blackboard 3:
algorithmic complexity

Blackboard 3:
algorithmic complexity



$$\frac{E\left(w_{jk}^{(1)}+\varepsilon\right)-E\left(w_{jk}^{(1)}-\varepsilon\right)}{2\varepsilon}$$

$$E(\boldsymbol{w})=\frac{1}{2}\sum_{\mu=1}^{P}\sum_{i}\left[t_i^\mu-\hat{y}_i^\mu\right]^2$$

$$\begin{aligned}
\hat{y}_i^\mu &= x_i^{(2)} && (1)\\
&= g^{(2)}[a_i^{(2)}] && (2)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} x_j^{(1)}] && (3)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(a_j^{(1)})] && (4)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(\sum_k w_{jk}^{(1)} x_k^\mu)] && (5)
\end{aligned}$$

Direct numerical evaluation: complexity

calculate $E(\boldsymbol{w}) = \frac{1}{2}\sum_{\mu=1}^{P}\sum_{i}\left[t_i^{\mu} - \hat{y}_i^{\mu}\right]^2$

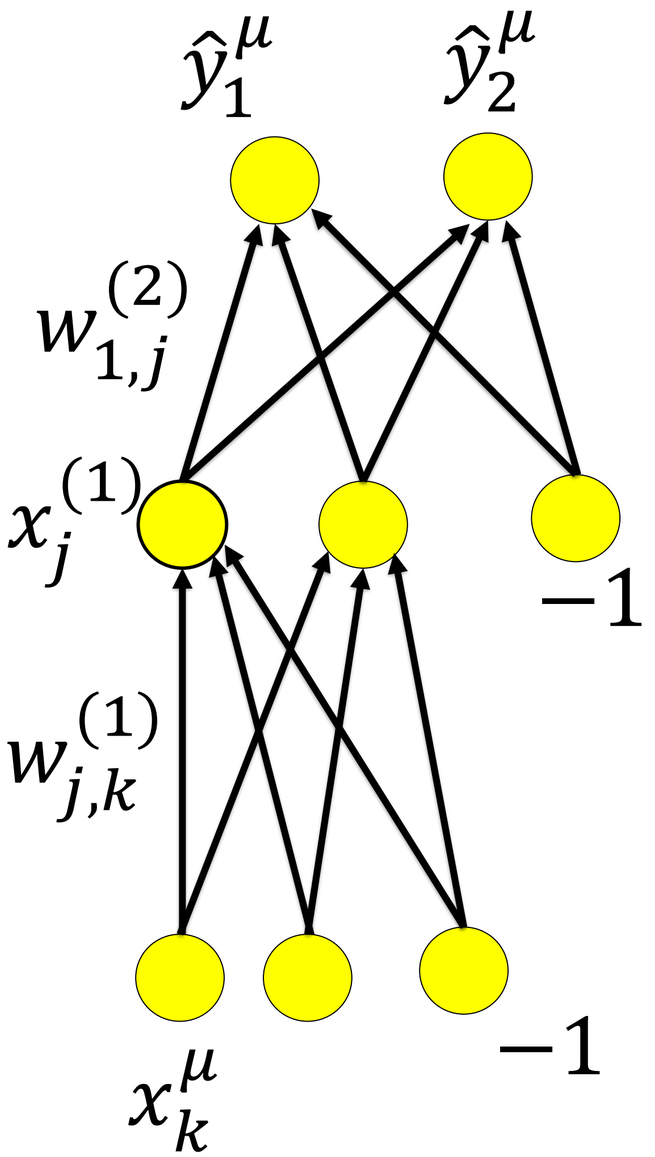
1) calculate $\hat{y}_i^{\mu}$ for one pattern

→ each weight is touched once

2) for each change of weight, evaluate E twice

$$\Delta w_{jk}^{(1)} = -\gamma \frac{dE\left(w_{jk}^{(1)} + \varepsilon\right) - dE\left(w_{jk}^{(1)} - \varepsilon\right)}{2\varepsilon}$$

3) For $\boldsymbol{n}$ weights, order $\boldsymbol{n}$-square!!!



Backprop: complexity

Exercise 2 at home: show algo is of order ***n***

0. **Initialization of weights**

1. **Choose pattern** $\mathbf{x}^{\mu}$

   input $x_k^{(0)} = x_k^{\mu}$

2. **Forward propagation of signals** $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \qquad (1)$$

   output $\hat{y}_i^{\mu} = x_i^{(n_{\max})}$

3. **Computation of errors in output**

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^{\mu} - \hat{y}^{\mu}] \qquad (2)$$

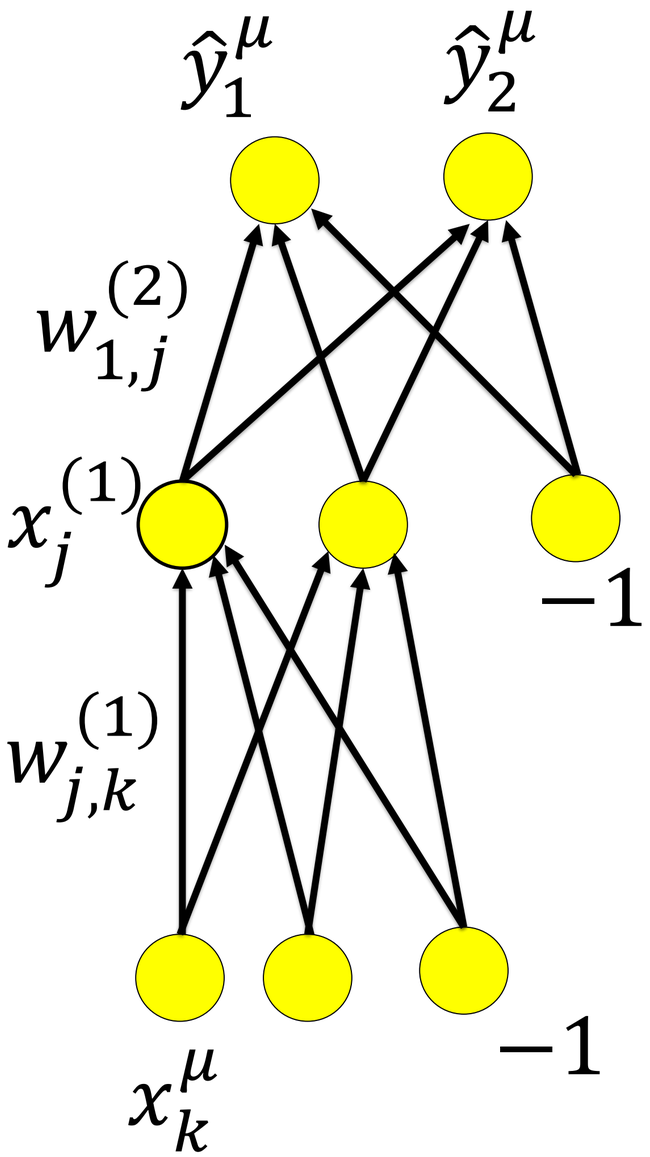
4. **Backward propagation of errors** $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \qquad (3)$$

5. **Update weights (for each** (i,j) **and all layers** (n)**)**

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \qquad (4)$$

6. **Return to step 1.**



Backprop: Quiz

Your friend claims the following; do you agree?

[ ] BackProp is nothing else than the chain rule, handled well.

[ ] BackProp is just numerical differentiation

[ ] BackProp is a special case of automatic algorithmic differentiation

[ ] BackProp is an order of magnitude faster than numerical differentiation

Conclusions: Multilayer Perceptron and Backprop

- Weights can be adapted by gradient descent
- Backprop is an implementation of gradient descent
- Gradient descent converges to a local minimum

→ **Big Multilayer perceptrons (and infinitely many Network architectures) can be trained by BackProp to minimize a loss function**

BUT no BackProp please (in this class)

- BackProp needs four separate phases:

  forward pass, mismatch calculation, backward pass, weight update.

- Backward pass needs specific feedback architecture

  (e.g., all linear, and feedback weights = feedforward weights).

- The feedback architecture must enable vector feedback

T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton. Backpropagation and the brain. *Nat. Rev. Neurosci.*, 21(6):335–346, 2020. ISSN 14710048. doi: 10.1038/s41583-020-0277-3.