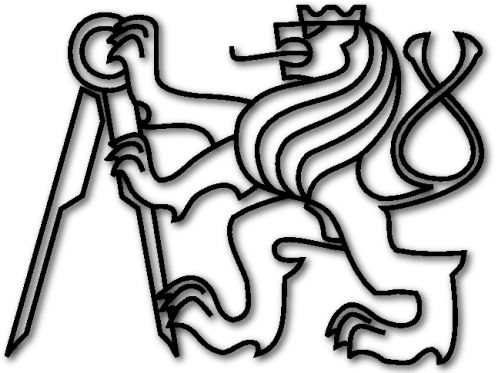
Artificial Neural Networks
Recurrent Neural Networks

Jan Drchal
drchajan@fel.cvut.cz



Computational Intelligence Group
Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague

Outline

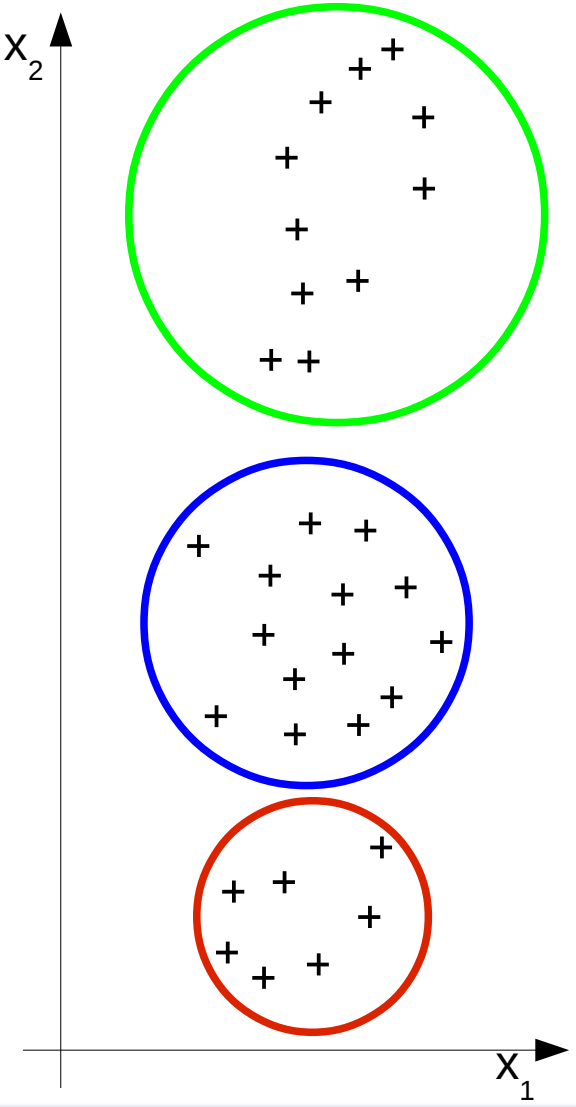
- Time & dynamics – motivation.
- Time-series with feed-forward ANNs.
- Recurrent ANNs:
  - architectures,
  - recall,
  - training.
- Hopfield network.

Motivation

- Real world can be understood as a collection of different signals:
  Let's focus on **time & dynamics.**
- Tasks:
  - prediction (economy, weather, ...),
  - recognition (speech, video, ...),
  - modelling (text – grammar, automata description, ...),
  - filtration.
- Working with **time-series**.

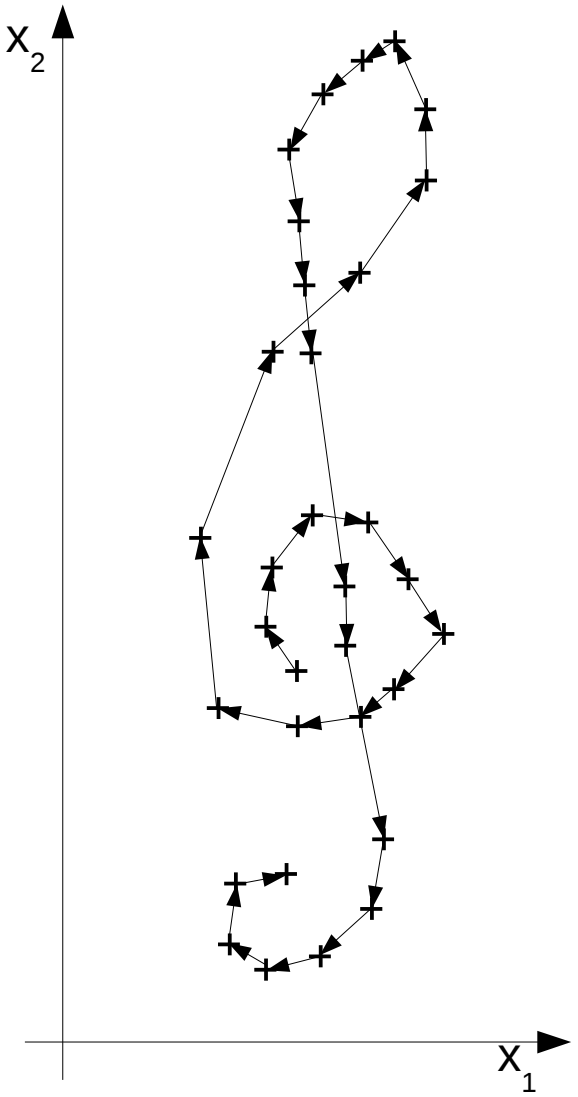
Motivation, Time & Signals

- **Static data**, independent, isolated data vectors.
- What do you see in the figure?
  - 3 clusters in Euclidean space.



Motivation, Time & Signals II

- **Dynamic (sequential) data.**
- Temporal order of vectors utilized.
- Proper meaning of the data can be now recognized - **G-clef**.



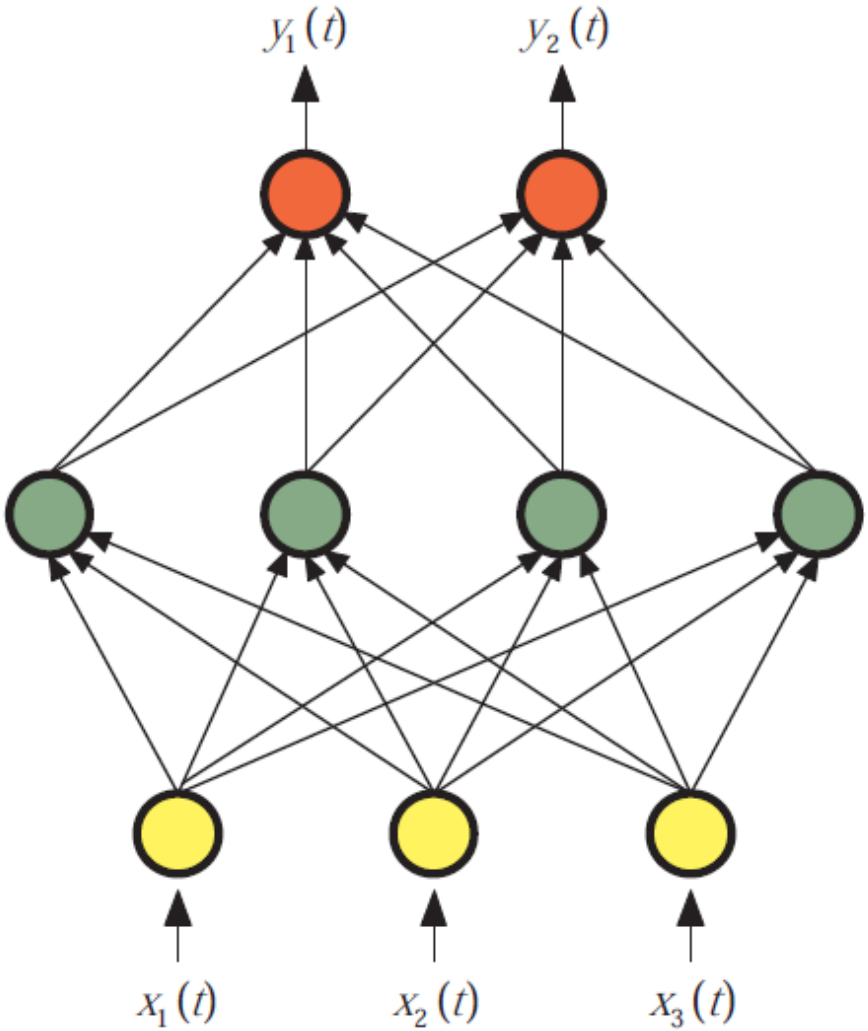
Processing Dynamic Data

- Architectures:
  - feed-forward networks,
  - **recurrent networks** (RNNs):
    - partially/fully recurrent networks.
- Dynamics of recurrent networks:
  - **discrete time dynamics**,
  - continuous time dynamics.

Dynamics

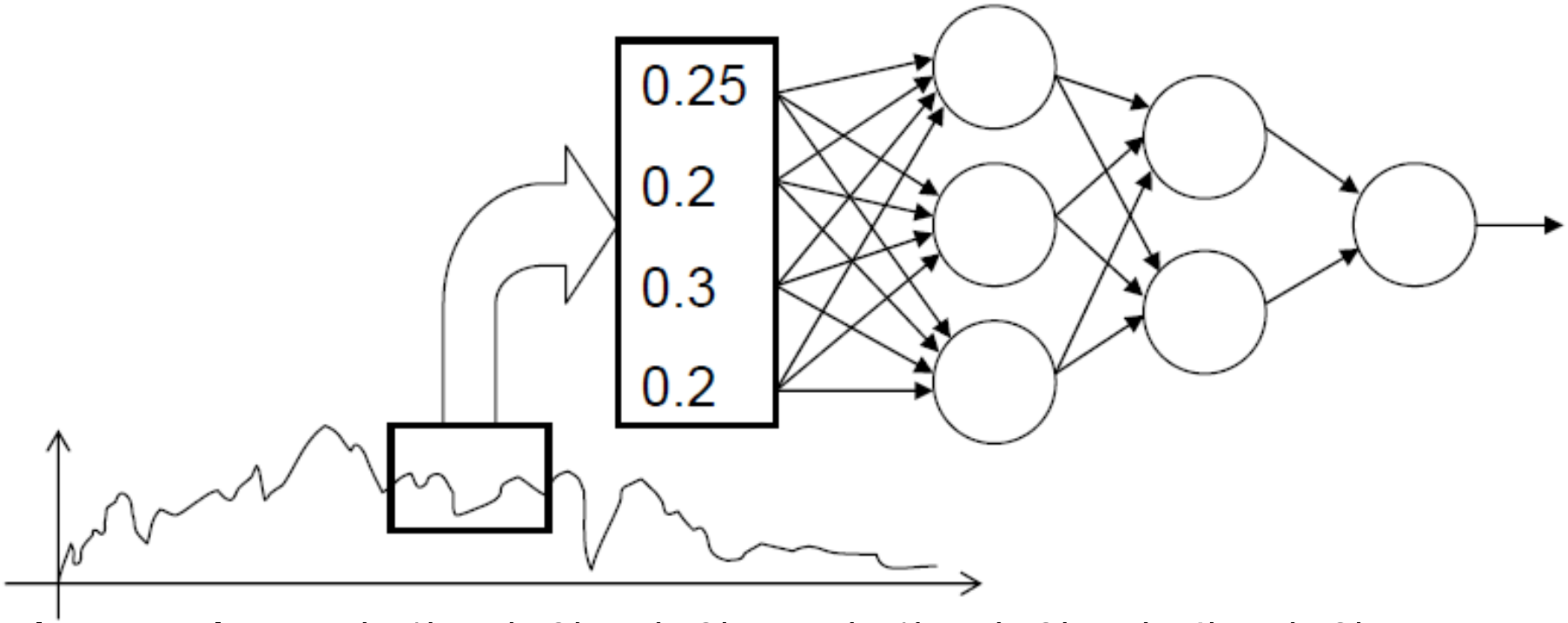
- Discrete time:
  - the network state in time *t+1* depends on the network state in time *t.*
- Continuous time:
  - special types of neurons: i.e. leaky-integrator neurons, spiking neurons.

Feed-forward Architecture



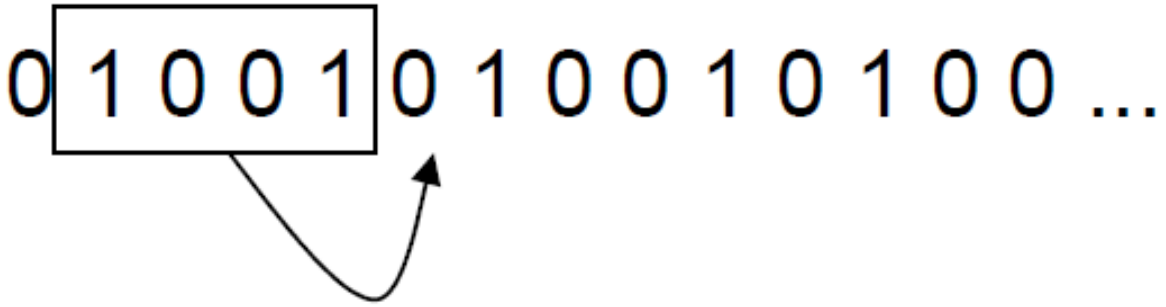
Processing Sequence by Feedforward Neural Network

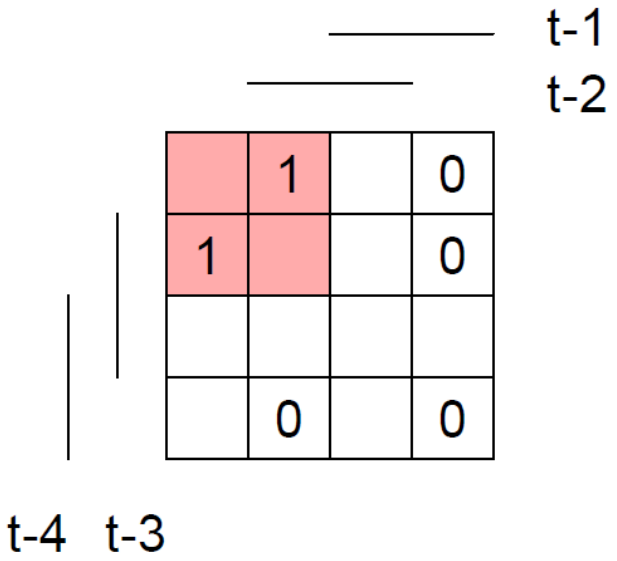
- Simplest way: **sliding window → Time Delay Neural Networks (TDNN)**
- Feedforward ANN → *combination circuit*.



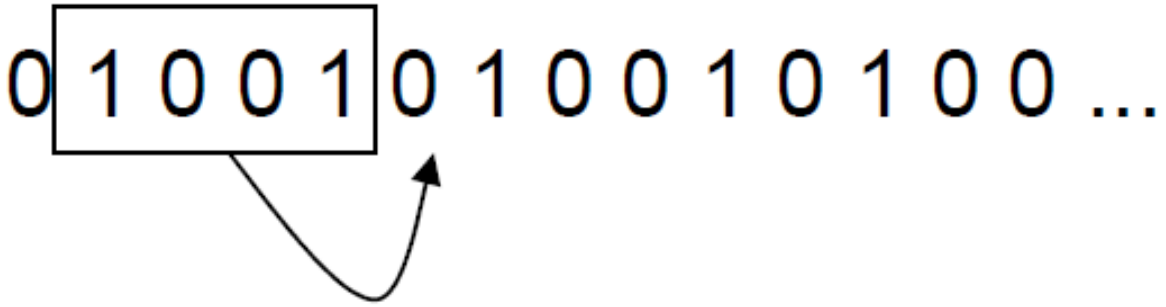
Input values *x(t-1), x(t-2), x(t-3)* or *x(t-1), x(t-2), x(t-4), x(t-8)*...

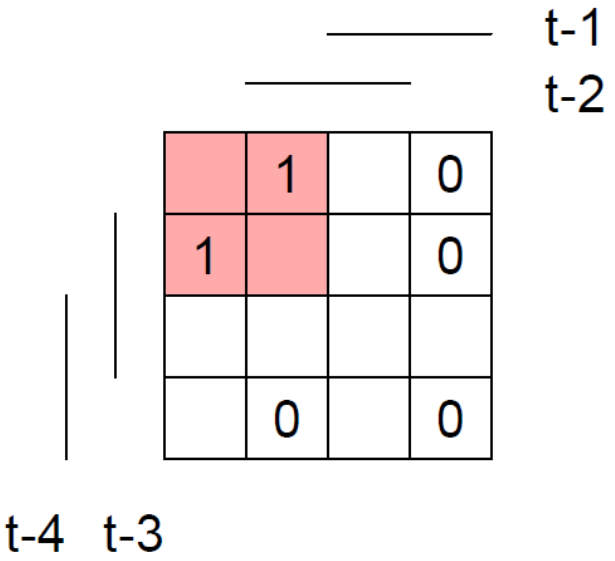
Time Delay Neural Network (TDNN) Example

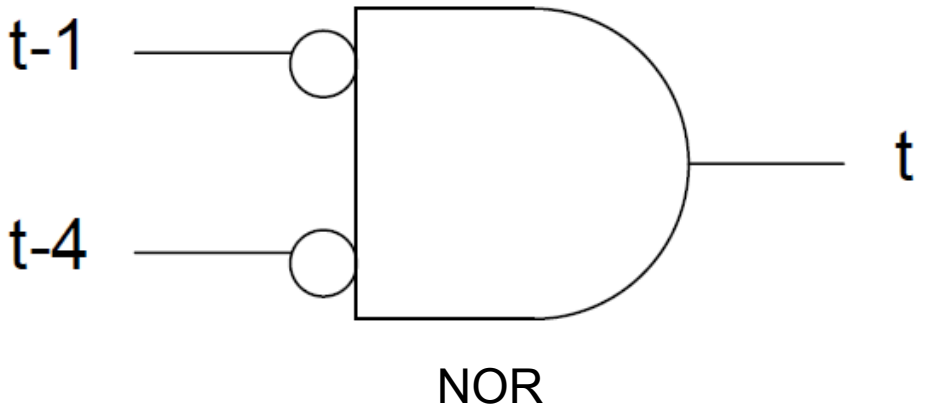




Time Delay Neural Network (TDNN) Example





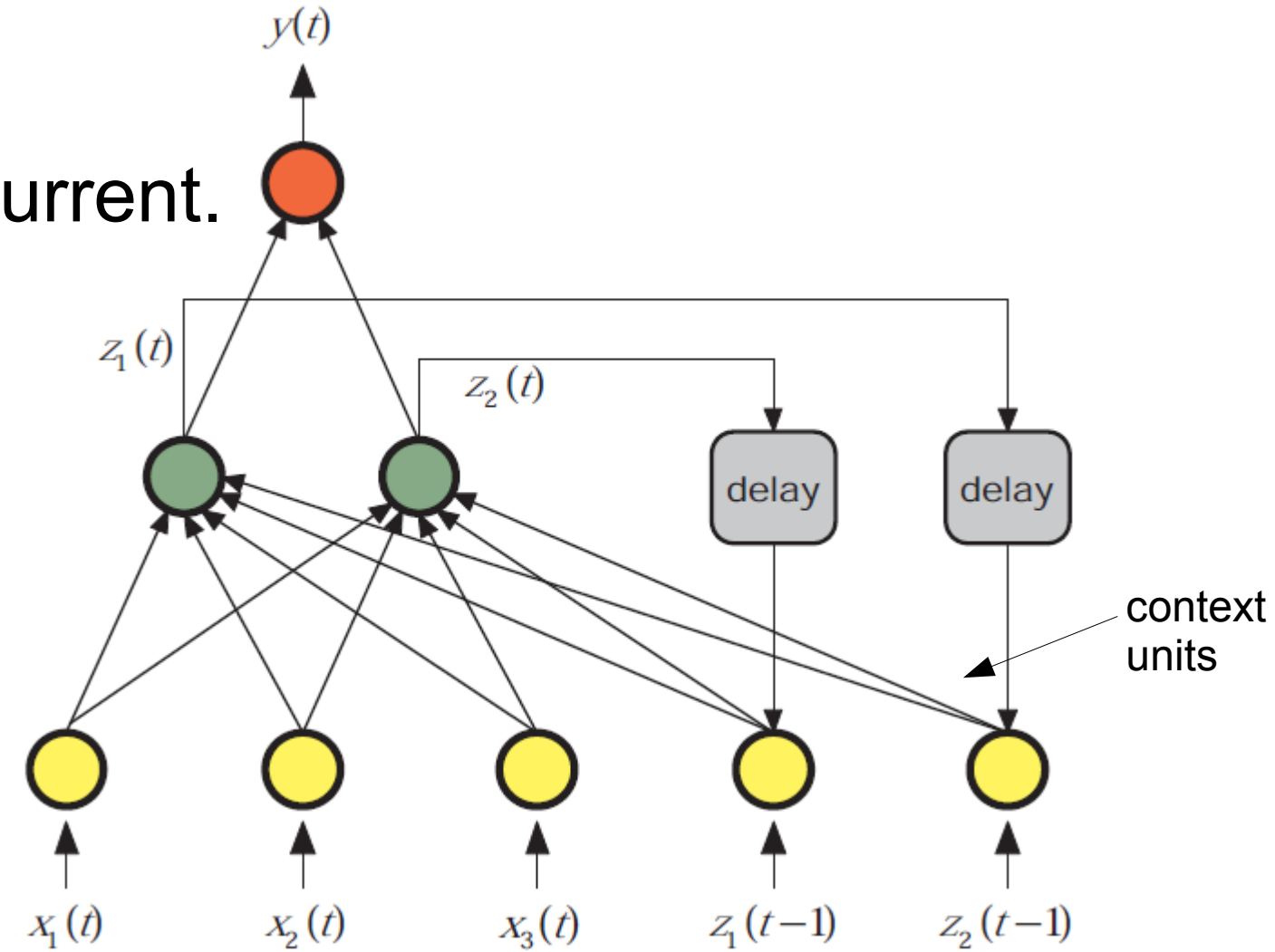


TDNN Remarks

- We need to know the period (memory depth).
- Disadvantageous for longer periods.
- Large growth of number of neurons.
- Better use sequential circuitry → native approach → states/memory.
- Examples:
  - automatons,
  - grammars,
  - regular expressions,
- **Recurrent connections.**

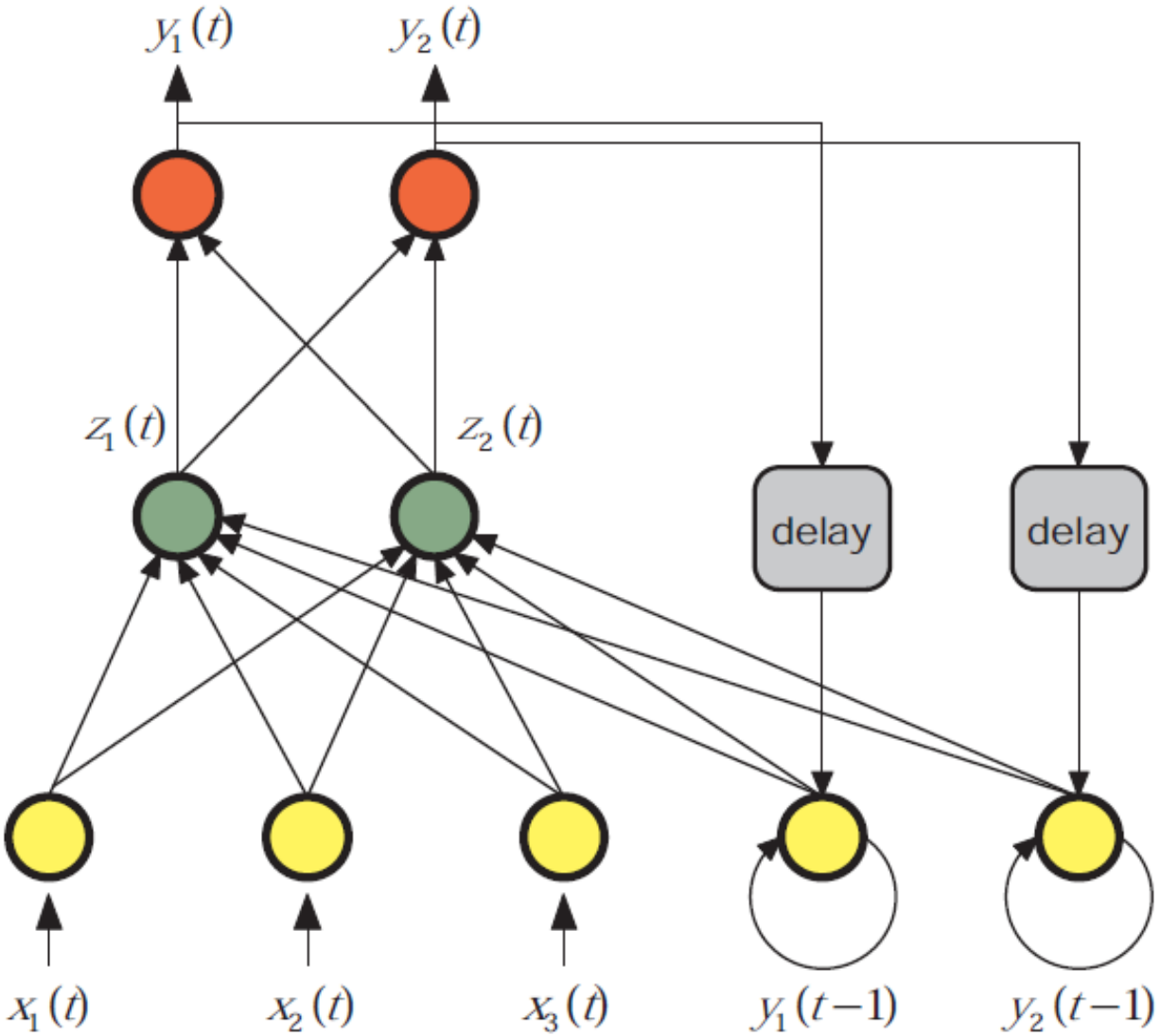
Elman's Network

- 1990.
- Partially recurrent.



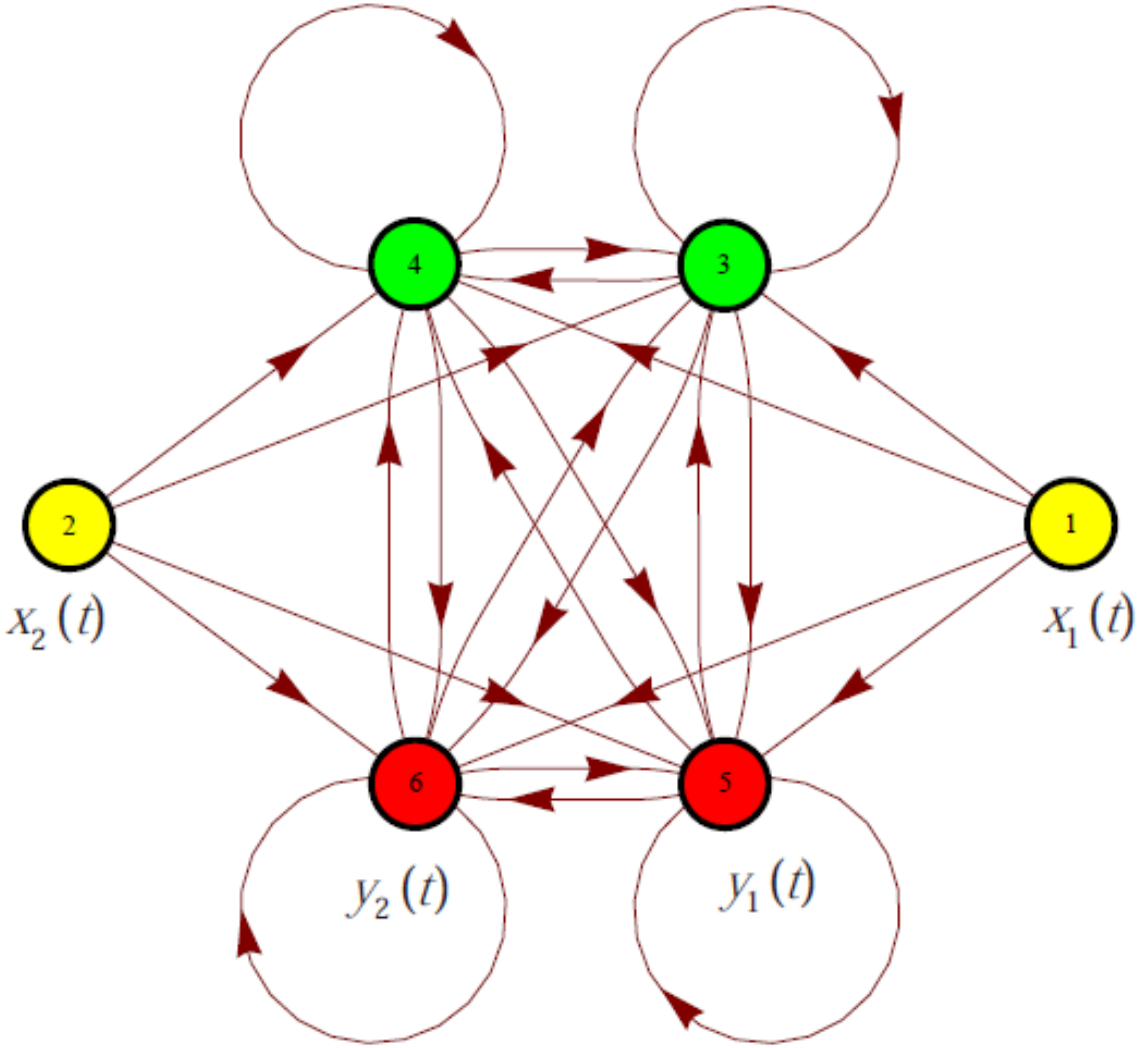
Jordan's Network

- 1989.



Fully Connected Architecture

Note: every network is a sub-network of a fully-recurrent network having the same # of neurons.



RNN Recall

- **Synchronous**:
  - change all neuron states simultaneously,
  - precompute neuron activities for time *t* based on state in *t-1* → change all activities at once.
- **Asynchronous**:
  - compute and immediately set activities for individual neurons,
  - proceed in predefined order (corresponding to signal flow, random).

RNN Recall 2

- When to read RNN output?
  - Perform predefined number of simulation steps.
  - **Wait until relaxed.**
  - **Continuously push new input data to RNN**
    inputs and read responses at outputs →
    typical approach for controlling tasks.
    - Note the delay equal to the depth of the network (shortest path from inputs to outputs).

Response of RNNs

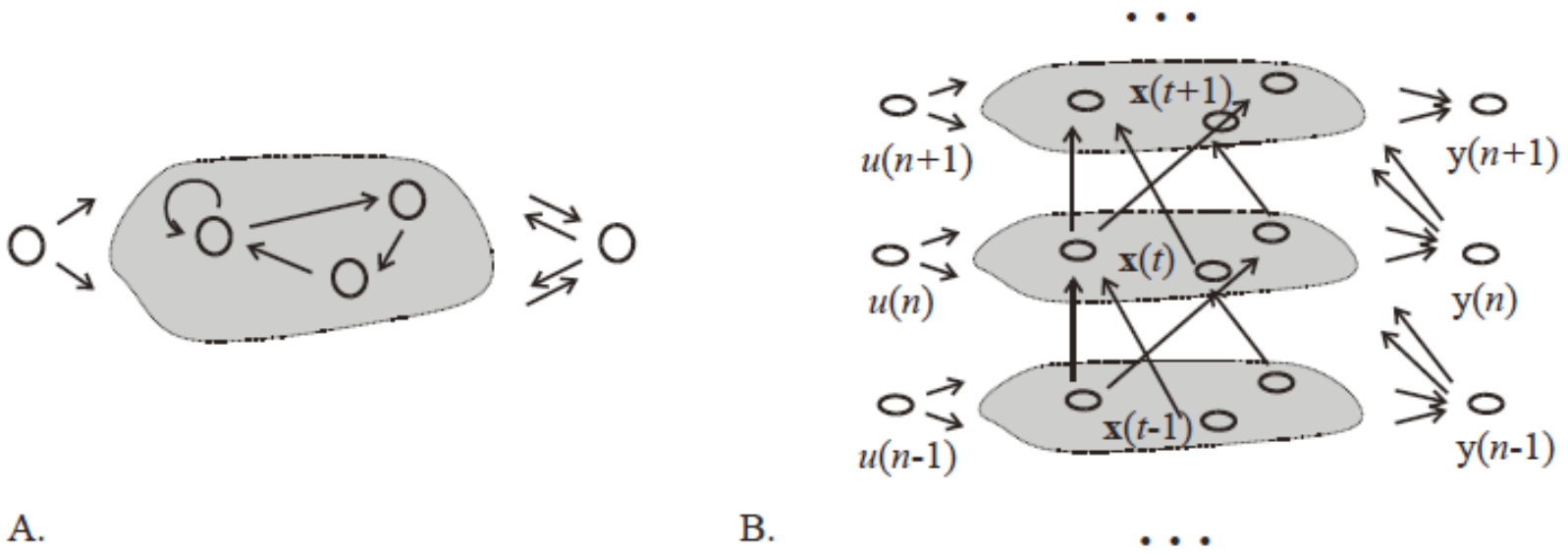
- Possibilities of output behaviour:
  - **convergence to a stable value**,
  - oscillation,
  - chaotic behaviour.
- Let's see demonstration with polynomial neuron....

Relaxation (Settling)

- Relaxation – wait until system (RNN) settles to a locally optimal state.
- Typical implementation:
  - Repeat evaluation steps of RNN until the difference between outputs of successive steps drops below a certain threshold.

Backpropagation Through-Time (BPTT)

- 1986 - Rumelhart, Hinton, Williams
- Unfold ANN in time & use BP.



Jaeger: A tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach, 2003

Backpropagation Through-Time 2

- Unfolding *p* times: *p*-BPTT – user must choose *p* :(
- Low time complexity, single epoch $O(TN^2)$:
  - *T ... # of training pairs,*
  - *N … # of neurons.*
- Slow convergence - same reasons as BP.
- Hard to achieve memory longer than 10-20 steps.
- Unlike BP, not guaranteed to converge to a local error minimum!
- Modifications:
  - BBPTT: Batch Backpropagation Through Time – averages weight changes over epoch,
  - QPTT: Quickprop Through Time.

Real-Time Recurrent Learning (RTRL)

- 1989 - Williams & Zipser
- Different approach to compute gradient.
- Often called “forward-propagation”.
- No *p* to choose :)
- Complexity of a single epoch: $O(N^4)$:(
- Better convergence, but slower epoch than BPTT.

Extended Kalman Filters

- Second order method suitable for recurrent neural networks.
- *"The Kalman filter uses a system's dynamics model, known control inputs to that system, and measurements (such as from sensors) to form an estimate of the system's varying quantities (its state) that is better than the estimate obtained by using any one measurement alone. "* (Wikipedia)

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \boldsymbol{\omega}_k$$

ANN: stationary process, corrupted by noise

$$\mathbf{y}_k = \mathbf{h}_k(\mathbf{w}_k, \mathbf{u}_k, \mathbf{v}_{k-1}) + \boldsymbol{\nu}_k.$$

measurement noise

desired output

nonlinear function

input vector

neuron activations

Simon Haykin: KALMAN FILTERING AND NEURAL NETWORKS

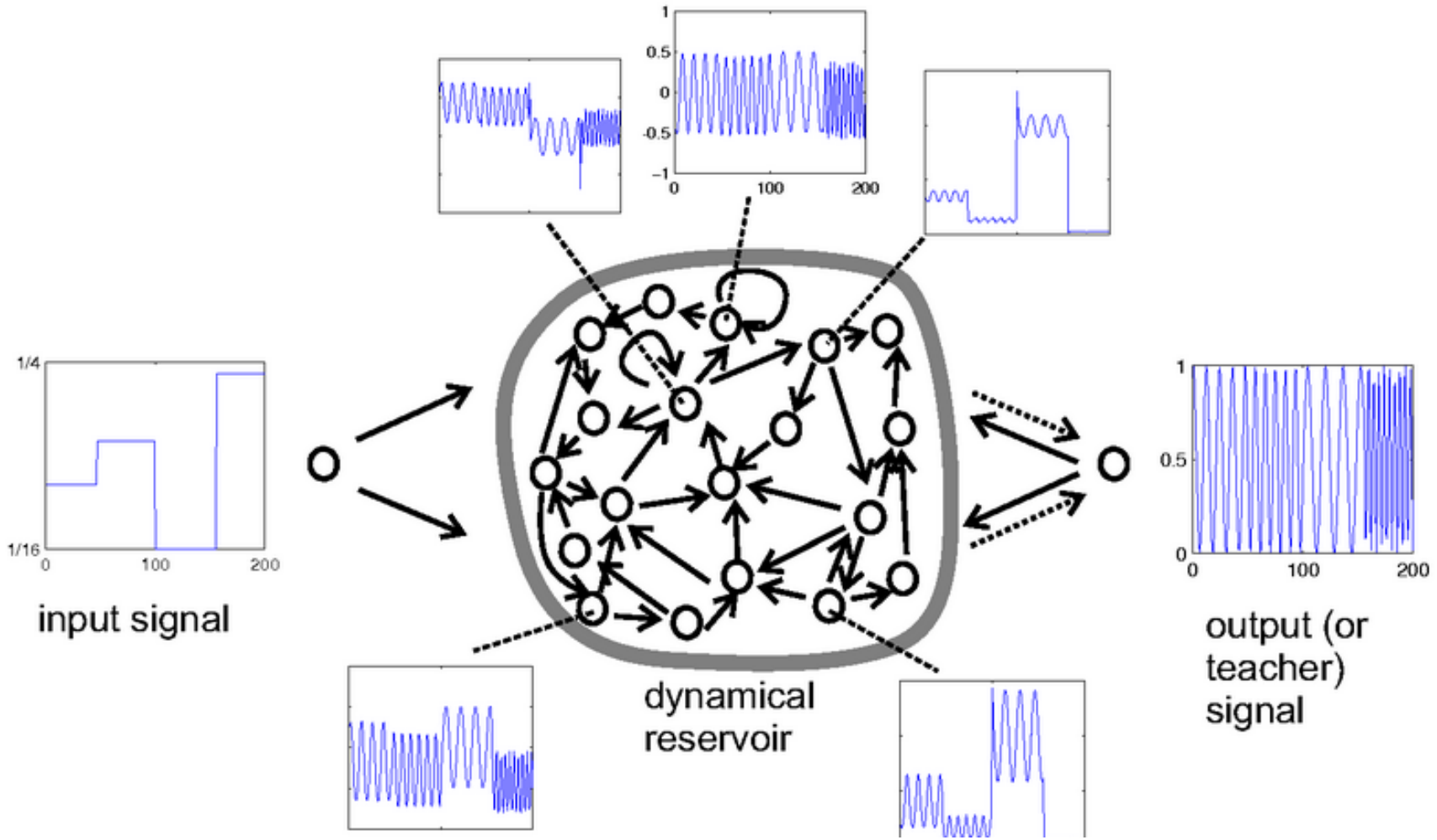
Echo State Networks (ESN)

- 2001, Jaeger
- Observation: **most often, dominant changes happen to output weights while training.**
- Pool of randomly connected neurons with **random** weights → **dynamical reservoir**.
- Connect inputs to reservoir: **random**, fully-connected.
- Connect reservoir to outputs: fully-connected, **these weights will undergo training**.

Echo State Networks (ESN) 2

- Training uses Linear Regression → general methods to estimate parameter of linear model (see http://en.wikipedia.org/wiki/Linear_regression)
- Very fast.
- Unlike previous algorithms: **prone to bifurcations** (change in dynamics caused by changes of system's parameters).
- For more info see: http://www.faculty.jacobs-university.de/hjaeger/esn_research.html

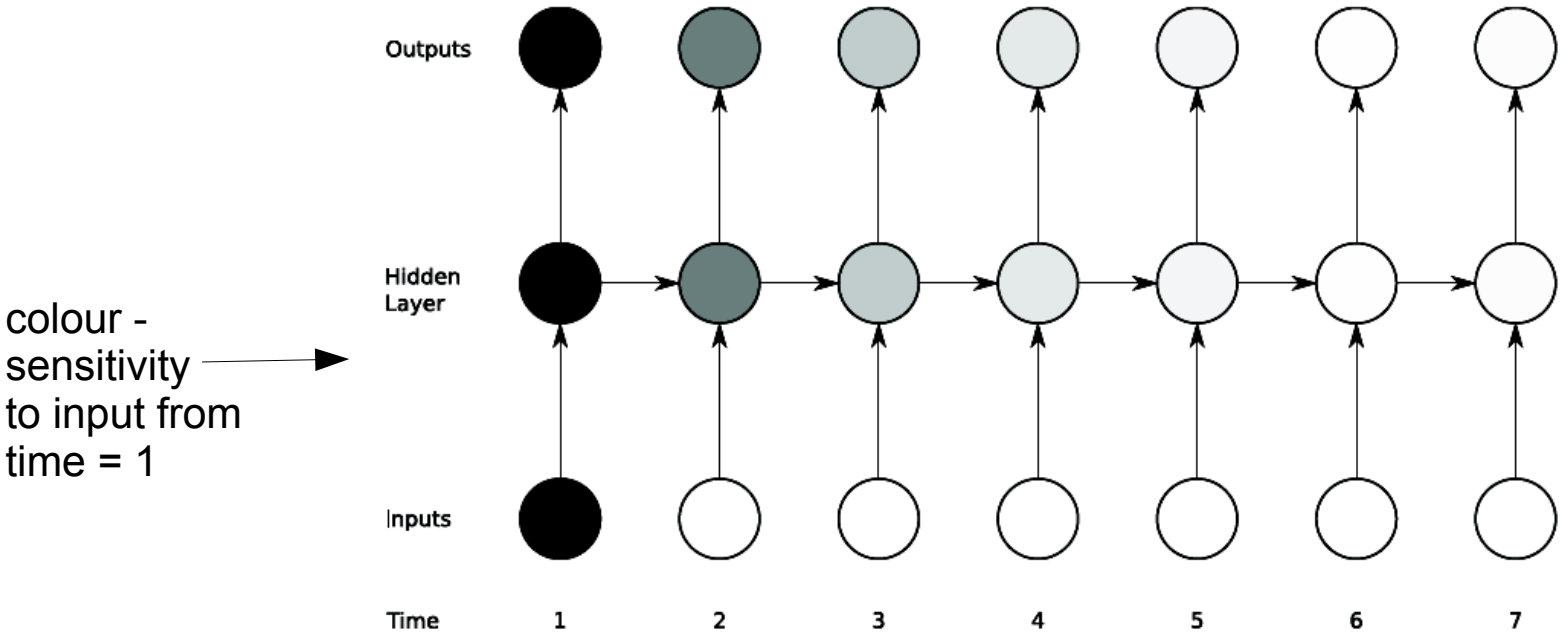
Echo State Networks Example: Tunable Frequency Generator



http://www.scholarpedia.org/article/Echo_state_network

Vanishing Gradient

It is hard to train RNNs with delays > 10 timesteps.

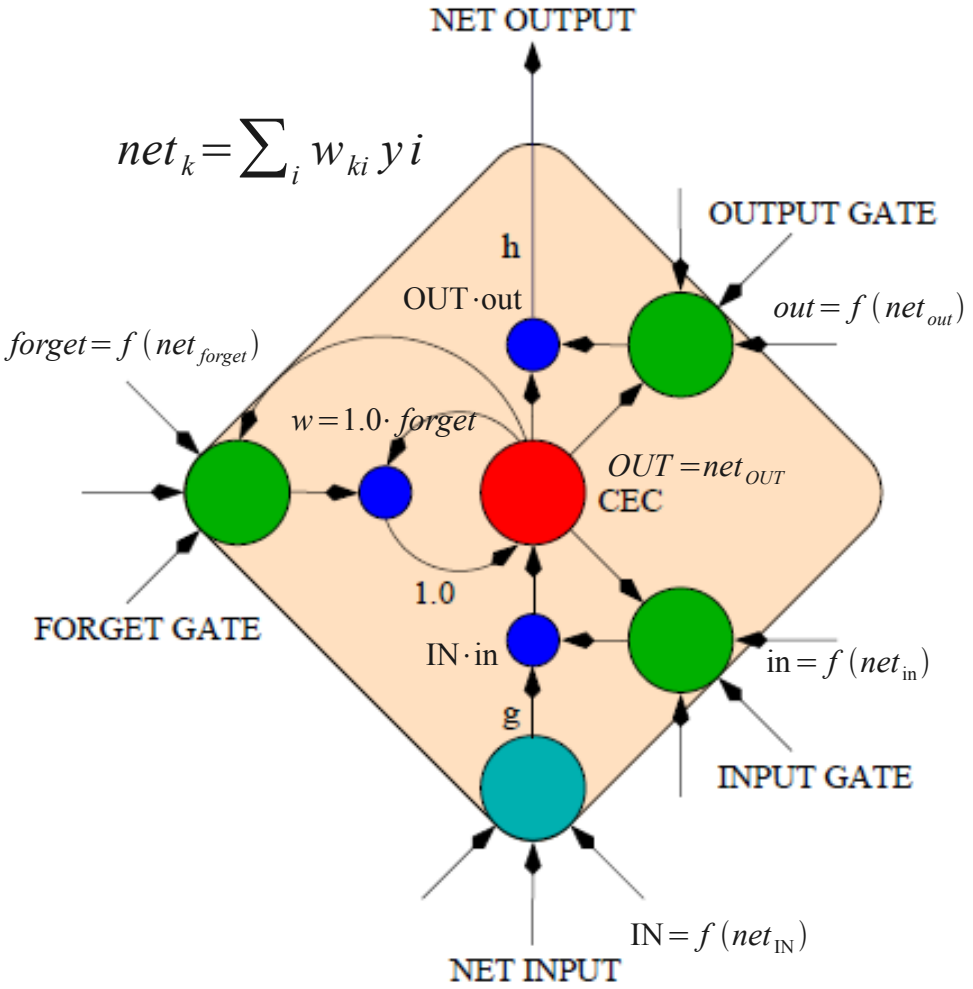


Exponential decay of sensitivity as new inputs overwrite the activation of hidden unit and the network “forgets” the first input.

Alexander Graves, Supervised Sequence Labelling with Recurrent Neural Networks, 2008

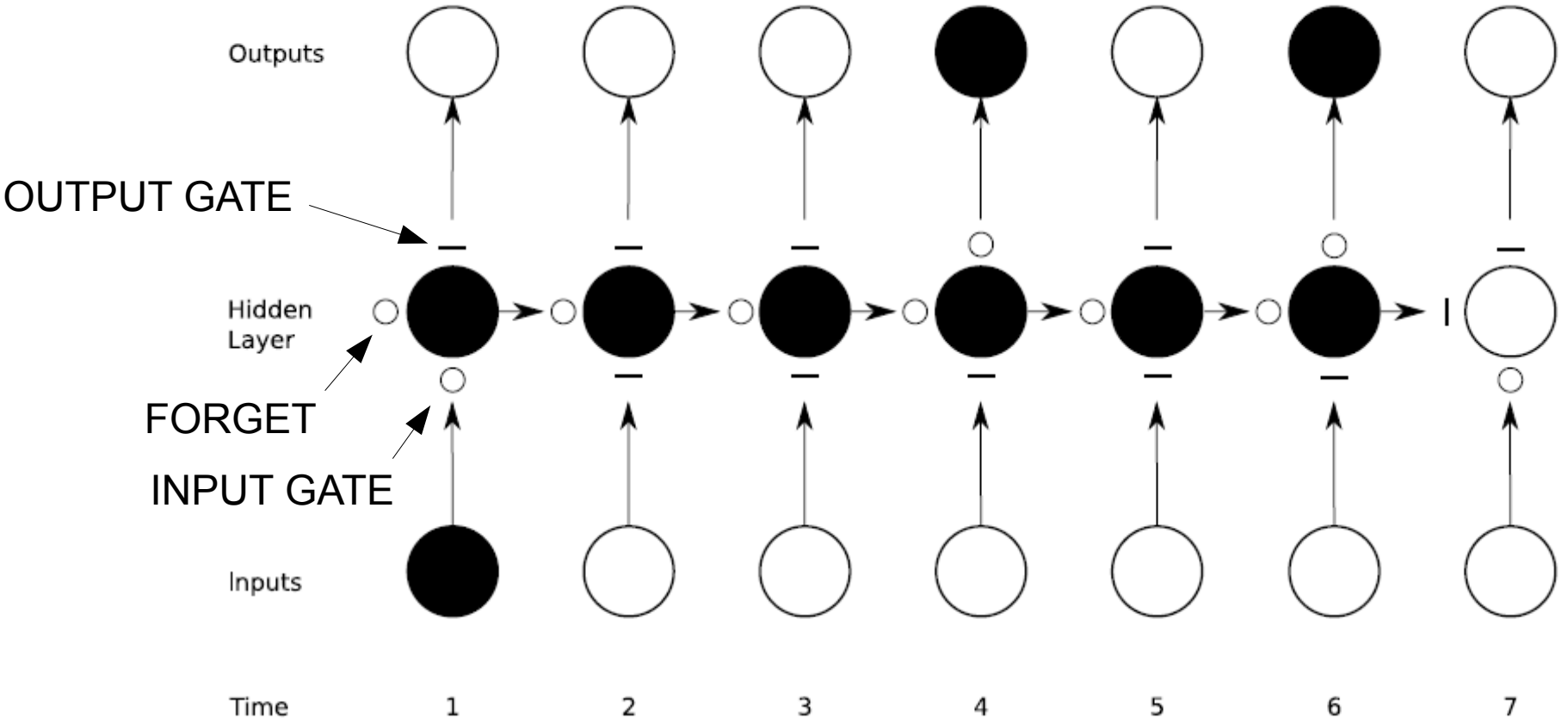
Long Short-Term Memory (LSTM)

- 1997, Juergen Schmidhuber
- LSTM cell: subnetwork,
- Supports both long and short-term memories – **thousands timesteps**!
- Combined with classic networks.
- **Constant Error Carrousel** (CEC): with linear transfer function stores the information.
- **Gates** control access to the memory: reminds Write Enable (WE) and Output Enable (OE) on memory ICs.
- Originally gradient learning.



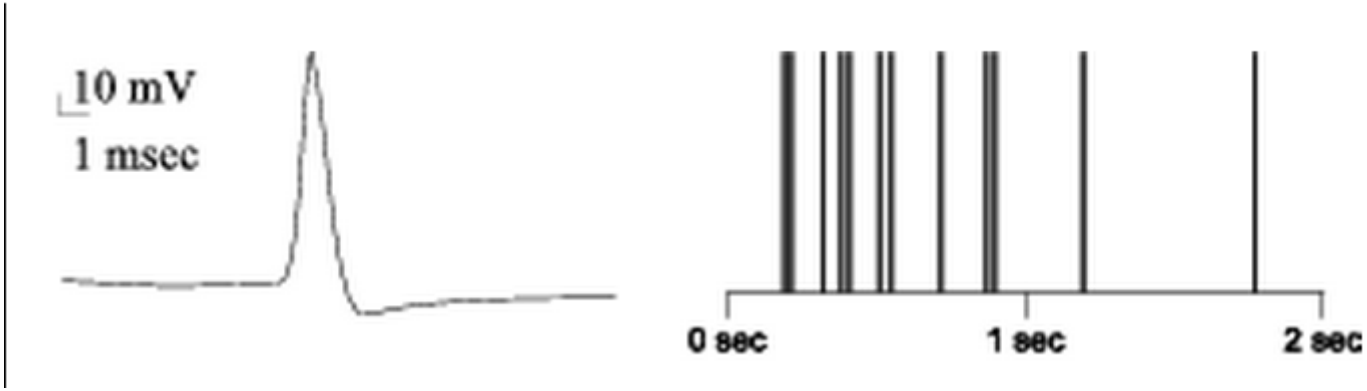
Long Short-Term Memory 2

Gate states: O (opened), – (closed).



Dynamic Neuron Model

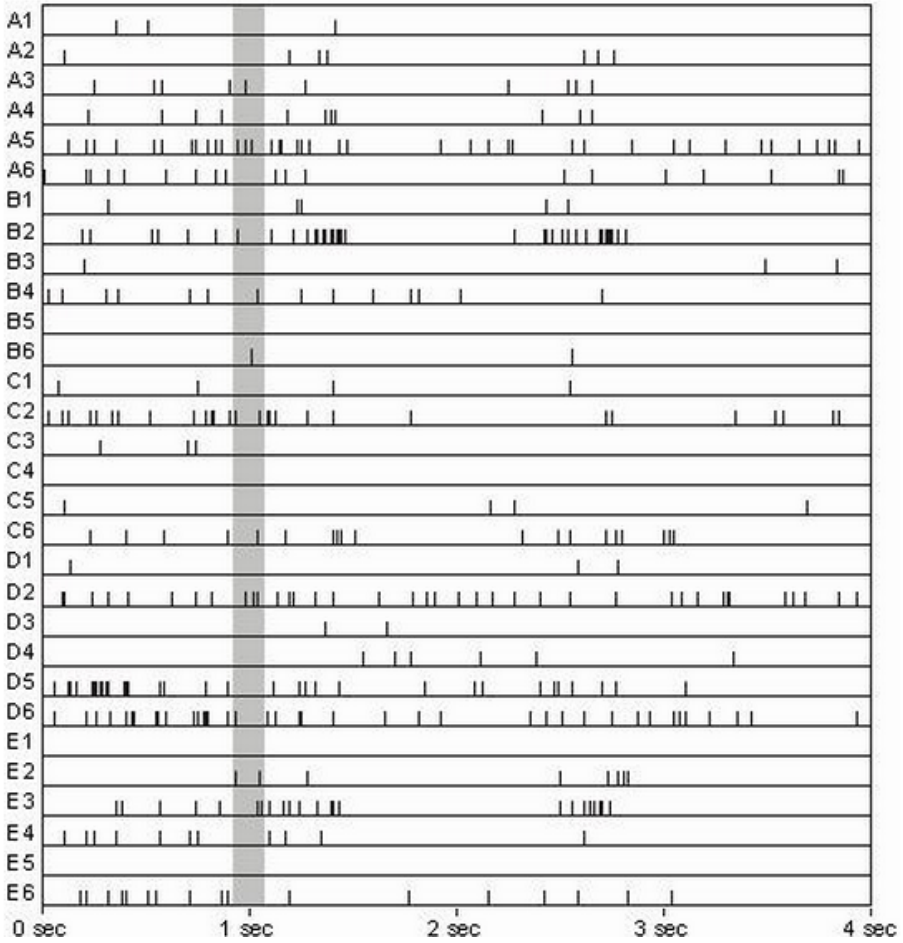
- Real neurons don't work with activation levels → they fire **spike trains**.



We talk about rate: spikes/time unit → **firing rate**.

http://www.igi.tugraz.at/maass/123/node2.html
and K. Stanley's presentation CAP6938: Leaky Integrator Neurons and CTRNNs

Spike Trains



- 30 neurons firing from monkey striate cortex (Krüger and Aiple, 1988).
- Selected 150ms range shows time needed for complex computations i.e. face recognition.
- **Spiking neural networks**:
  - different models of neurons: integrator accumulates inner potential, then fires...
  - different simulation approach.

http://www.igi.tugraz.at/maass/123/node2.html
and K. Stanley's presentation CAP6938: Leaky Integrator Neurons and CTRNNs

Hopfield Network

- John Hopfield, 1982
- Keynote:
  - The patterns are represented as deepest possible valleys of the error landscape.

Characteristics

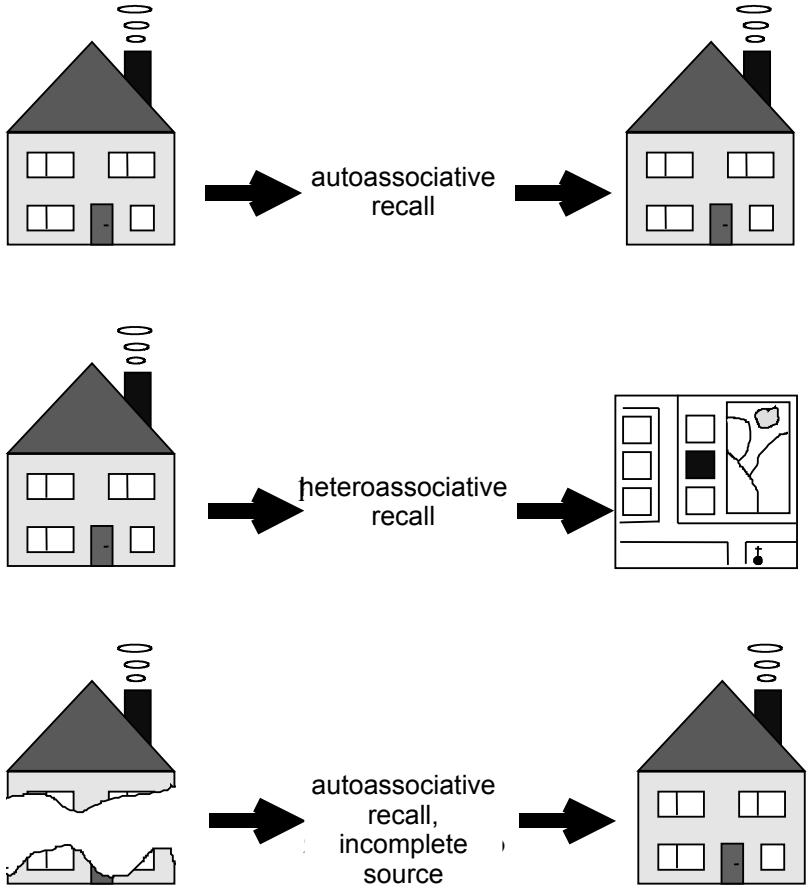
- Recurrent Network.
- Recalls the closest neighbour
  - **autoassociative behaviour.**
- Hebbian learning.

Reminder: Hebbian Learning

- *"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."*

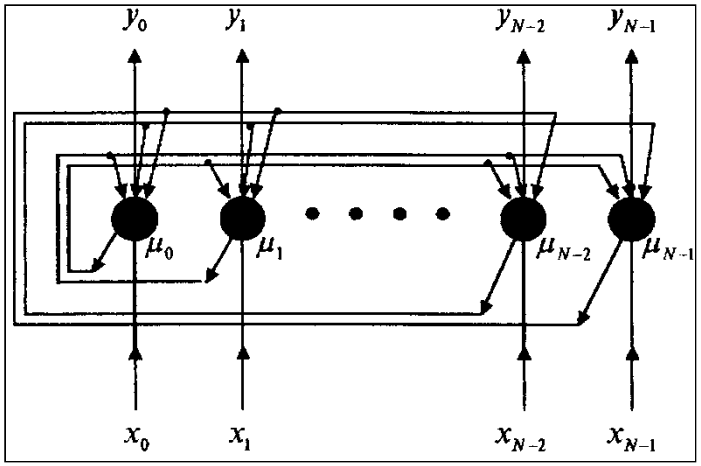
  → ***"cells that fire together, wire together"***

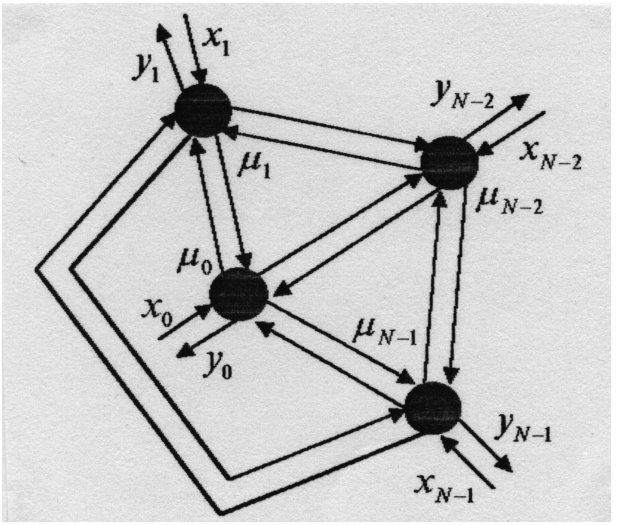
Autoassociative vs. Heteroassociative Recall



Hopfield Network Architecture

or





x_i – inputs,

y_i – network outputs,

μ_i – actual state of the neuron

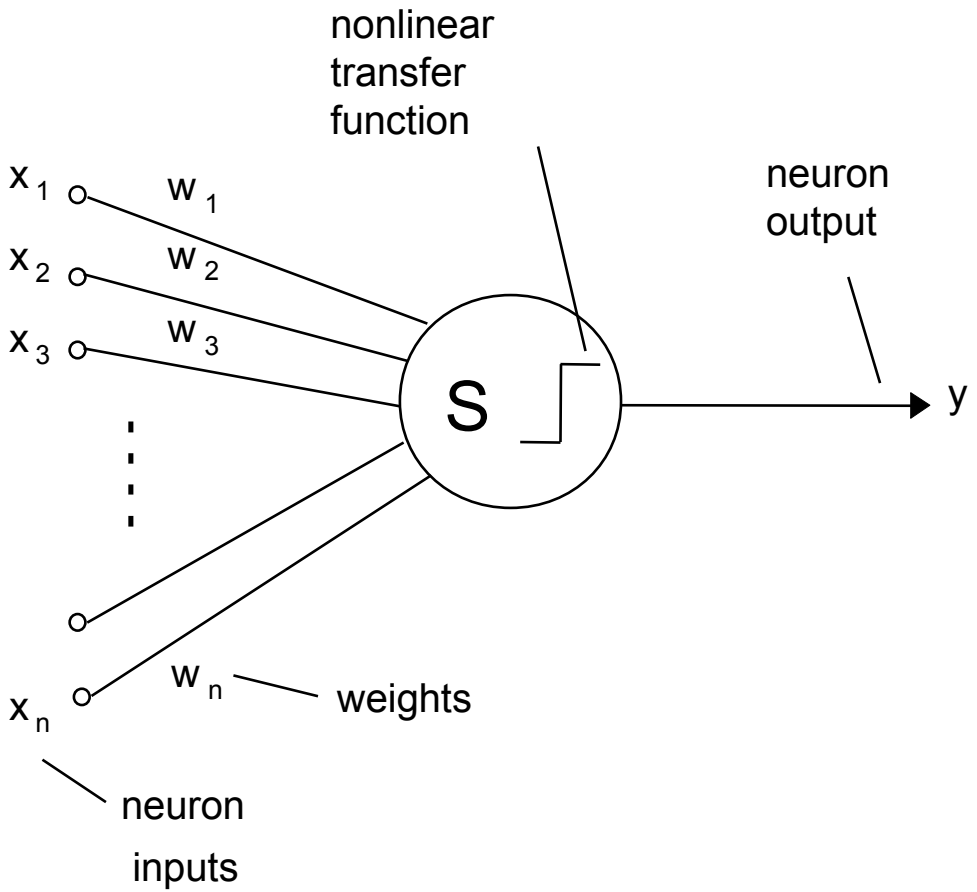
Architecture Remarks

- Output of a neuron is connected to inputs of all other neurons – there is no loop to self!
- Weights between neurons are symmetric

$$w_{ij} = w_{ji}$$

- The number of neurons is determined by the number of network inputs.
- Important note: neuron outputs *y* are inactive until the last step of network recall.

Neuron in Hopfield Network



Classic McCulloch-Pitts neuron **mostly without threshold**.

Concrete Implementation Details

- There exist other modifications, but:
  - binary input/outputs (more precisely +1/-1).
- Activation function:
  - bipolar step function (outputs -1 or +1).

Weight Matrix

- Square matrix with *N* rows and columns.
- *N* denotes the number of neurons in the network.
- Elements: w_{ij}.
- Properties?

Weight Matrix

- Square matrix with *N* rows and columns.
- *N* denotes the number of neurons in the network.
- Elements: w_{ij}.
- Properties:
  - symmetric: $w_{ij} = w_{ji}$,
  - zero diagonal: $w_{ii} = 0$

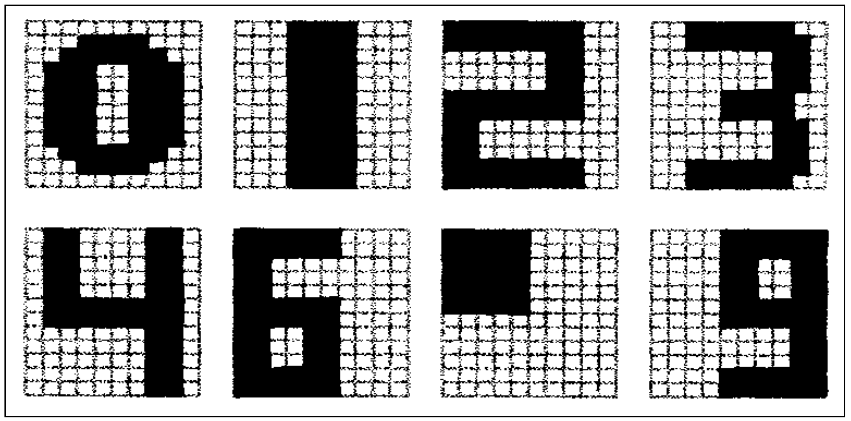
Learning Hopfield Network

- The weight matrix is acquired in a single step:

$$w_{ij} = \begin{cases} \sum_{s=1}^{M} x_i^s x_j^s, \; for\, i \neq j \\ 0, \; for\, i = j \;,\; 1 \leq i \;,\; j \leq N \end{cases}$$

M patterns

Example: image associative memory



Input vector *x* encoding*:*

+1 for white pixel,

-1 for black.

Notice:

- Two points in the image have:
  - same colour → synaptic weight set to +1,
  - different → -1
    → **Hebbian learning!**
- One-shot learning of a single input pattern!
  → The weight matrix is computed in a single step.
- **Q**: what are possible values of the weight matrix?

Don't be too Enthusiastic :(

- Not many patterns can be stored (we will see later).
- The patterns must be mutually diverse (having the greatest possible Hamming distance).
- But, at least, the network knows inverse patterns for free (really an advantage?).

Hopfield Network Recall

- Iterative, until relaxation.

Step 1. Initialization

Set initial neuron states to an input pattern.

$$\mu_i(0) = x_i, 1 \leq i \leq N$$

x_i is an input pattern element (+1 or -1).

Hopfield Network Recall 2

Step 2. Iterate until relaxation

$$\mu_i(t+1) = f\left[\sum_{j=1}^{N} w_{ij}\mu_i(t)\right], 1 \le i \le N .$$

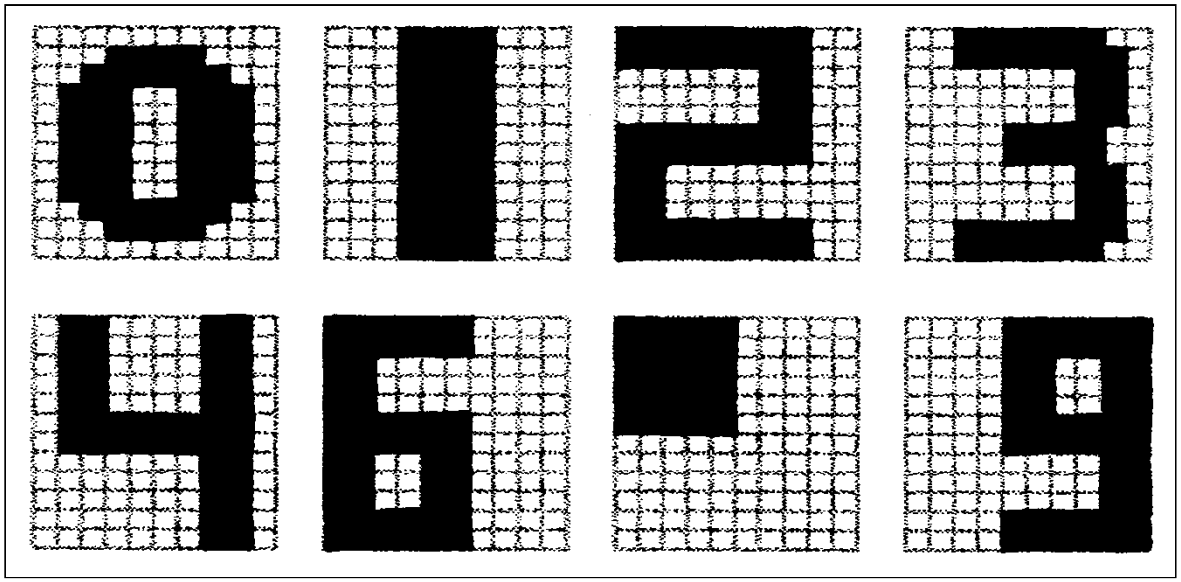
This step is repeated, as long as there is any difference between current and previous states.

$y_i = \mu_i(t_{last})$ → when relaxed, set network outputs

Note, both synchronous and asynchronous versions exist.

Example

- Let's teach the Hopfield network having 120 neurons to remember this set of characters:

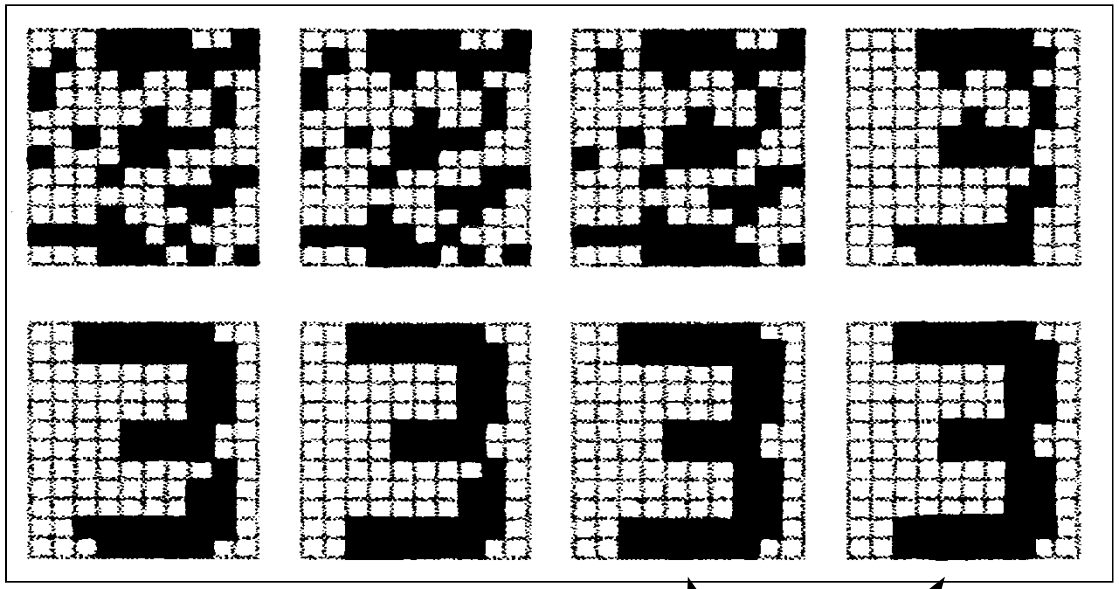


Matrices

10 x 12

Recall Result

- Apply a 25% distorted image of “3”:



Recall Iterations

No difference → the network is relaxed.

Hopfield Network Properties

- The capacity of network is *M* patterns, unfortunately it is quite small:
  - it can be inferred that: $M < 0,15 \times N$, where *N* is the number of pixels (neurons)
  - to learn *0,…,9*, we need at least 67 pixels per image.
- Patterns should be diverse.
- The weight matrix grows with a square of *N*.
- Problem: not tolerant to image rotation/shift/resize!

Common mistake!

- Hopfield network is not a classifier! It ouputs the nearest neighbour pattern.
- How to make a classifier out of Hopfield network?

FAQ: Why Such Learning Algorithm?

$$w_{ij} = \begin{cases} \sum_{s=1}^{M} x_i^s \, x_j^s, \; for \; i \neq j \\ 0, for \; i = j \; , \; 1 \leq i \; , \; j \leq N \end{cases}$$

Why do we compute the weight this way?

Answer: it is based on a notion of energy landcape defined by an energy function.

Energy Function by Hopfield

The energy function should be constructed in this way:

- high values for large errors (hills),
- low values for small errors (valleys).

One possible solution:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} x_i x_j + \sum_i x_i q_i$$

Task:

- Find weight matrix *W* in order to minimize energy function for a given pattern.
- Do not damage already learned patterns (dug valleys)!

Solution

- Let's start with our energy function:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} x_i x_j + \sum_i x_i q_i$$

- We want it to be minimal (most negative).
- For simplicity the thresholds are omitted:

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij} x_i x_j$$

Energy Function Decomposed

- For pattern *s* let's split the weight matrix *W* into two parts:
  - influence of all patterns except of the s^{th} (denote w^*)
  - influence of the s^{th} pattern only (denote w^s).

$$E = -\frac{1}{2}\sum_i \sum_j w^*_{ij} x_i x_j - \frac{1}{2}\sum_i \sum_j w^s_{ij} x_i x_j = E_{others} + E_s$$

noise **signal**

Energy Function Decomposed 2

- **Signal** is an energy given by the s^{th} pattern.
- **Noise** is caused by contributions of all other patterns:
  - we can't do much with the noise!
- On the other hand, we can lower the energy value of the signal (minimize the contribution of the s^{th} pattern)
  → learning is just a minimization of energy after all!

How to Minimize E_s?

- Corresponds to **maximization** of:

$$\sum_i \sum_j w_{ij}^s x_i^s x_j^s$$

- The values of x_i^s and x_j^s are either +1 or -1.
- The value of $(x_i^s x_j^s)^2$ is always positive, so we choose:

$$w_{ij}^s = x_i^s x_j^s$$

Problems: Local Minima

- The energy function

$$E = -\frac{1}{2}\sum_i \sum_j w_{ij}^{*} x_i x_j - \frac{1}{2}\sum_i \sum_j w_{ij}^{s} x_i x_j = E_{others} + E_s$$

respects the placement of all patterns→ local extremes might appear →

Phantoms – false patterns not corresponding to any pattern of a training set.

How to Fix it?

- Simulated annealing based methods.

Solving Optimization Problems Using Hopfield Network

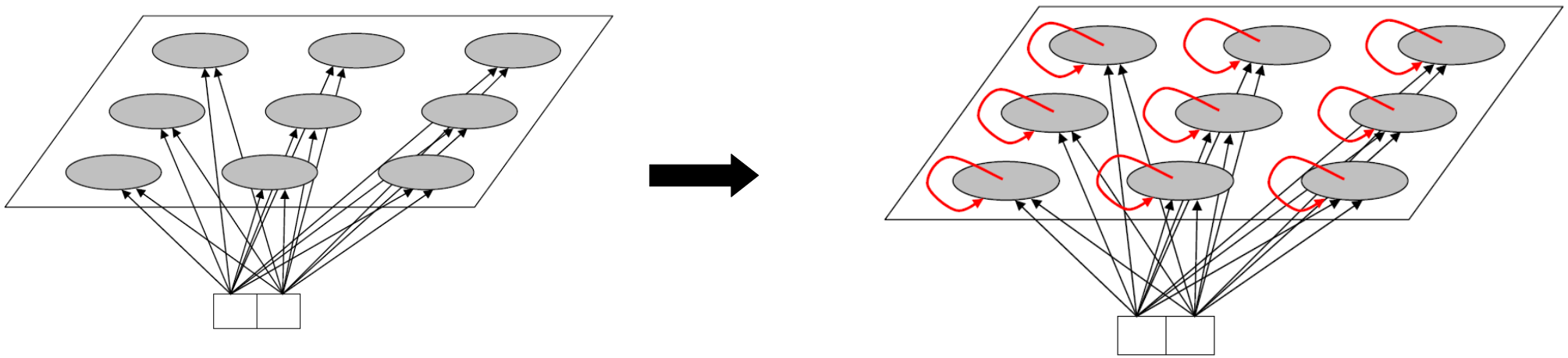
- Hopfield & Tank (1985)
- They use Hopfield model to solve the TSP.
- How to do it?
  Design an energy function which is similar to the cost function of the problem (the tour length) → find minimum energy.

Next Lecture

- Neuro Evolution = ANN + EA
- Mostly TWEANNs - Topology and Weight Evolving ANNs).
- Direct/Indirect encodings → possibilities to evolve really HUGE ANNs (HyperNEAT).

Temporal Kohonen Map (TKM)

- 2001, Varsta, Based on SOM



Temporal Kohonen Map (TKM) 2

Neuron output

$$y_i(t) = \left\| (\mathbf{x}(t) - \mathbf{w}_i(t)) \right\| \qquad y_i(t) = \alpha y_i(t-1) - \frac{1}{2} \left\| (\mathbf{x}(t) - \mathbf{w}_i(t)) \right\|$$

$$y_b(t) = \min_{i \in V} \left[y_i(t) \right] \qquad y_b(t) = \max_{i \in V} \left[y_i(t) \right]$$

BMU selection

Weight update:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \gamma(t) h_{i,b}(t) (\mathbf{x}(t) - \mathbf{w}_i(t))$$

Temporal Kohonen Map (TKM) 3

- How to evaluate results?