Data-driven Intelligent Systems

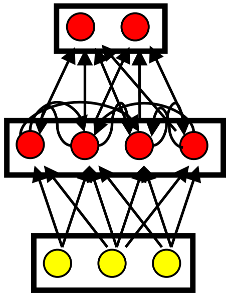
Lecture 13
Recurrent Neural Network Classification

KNOWLEDGE
TECHNOLOGY

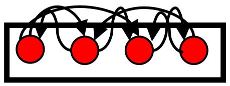
http://www.informatik.uni-hamburg.de/WTM/

Outline

- Simple Recurrent Neural Networks (Elman)
  - Time-Delay Neural Network
  - Simple Experiments with the RNN
  - Reservoir Network & Extreme Learning Machine

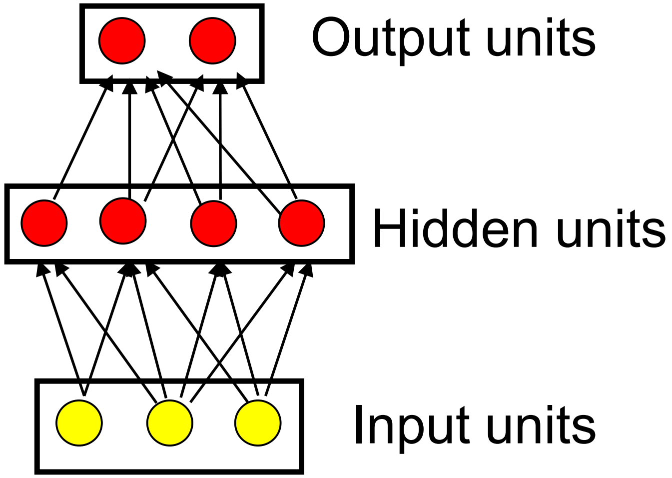


- Theory of Associative Memory
  - Hopfield Network
  - Energy Function

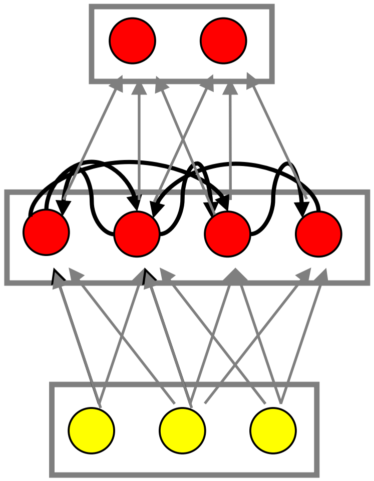


Introduction to Recurrent Neural Networks: Types of Connectivity

- Feedforward networks
  - These compute a series of transformations.
  - Non-linear functions, but elsewise simple input-output systems.



- Recurrent networks
  - These have directed cycles in their connection graph. Typically: $w_{ij} \neq w_{ji}$
    They can have complicated dynamics.
  - More biologically realistic.



Simple Recurrent Network (SRN)

- Tackle the problem with time

[0 1 1 1 0 0 0 0 0]

[0 0 0 1 1 1 0 0 0]

$\longrightarrow$
t

- Two vectors appear to be instances of the ***same*** basic ***pattern***, but displaced in space
- Relative temporal structure should be ***preserved*** in the face of absolute temporal displacements

Examples of Time Series

- Dow-Jones Industrial Average
- Products bought in a supermarket
- Electricity demand for a city
- Air temperature in a building
- Sunspot activity
- Speech, text, video

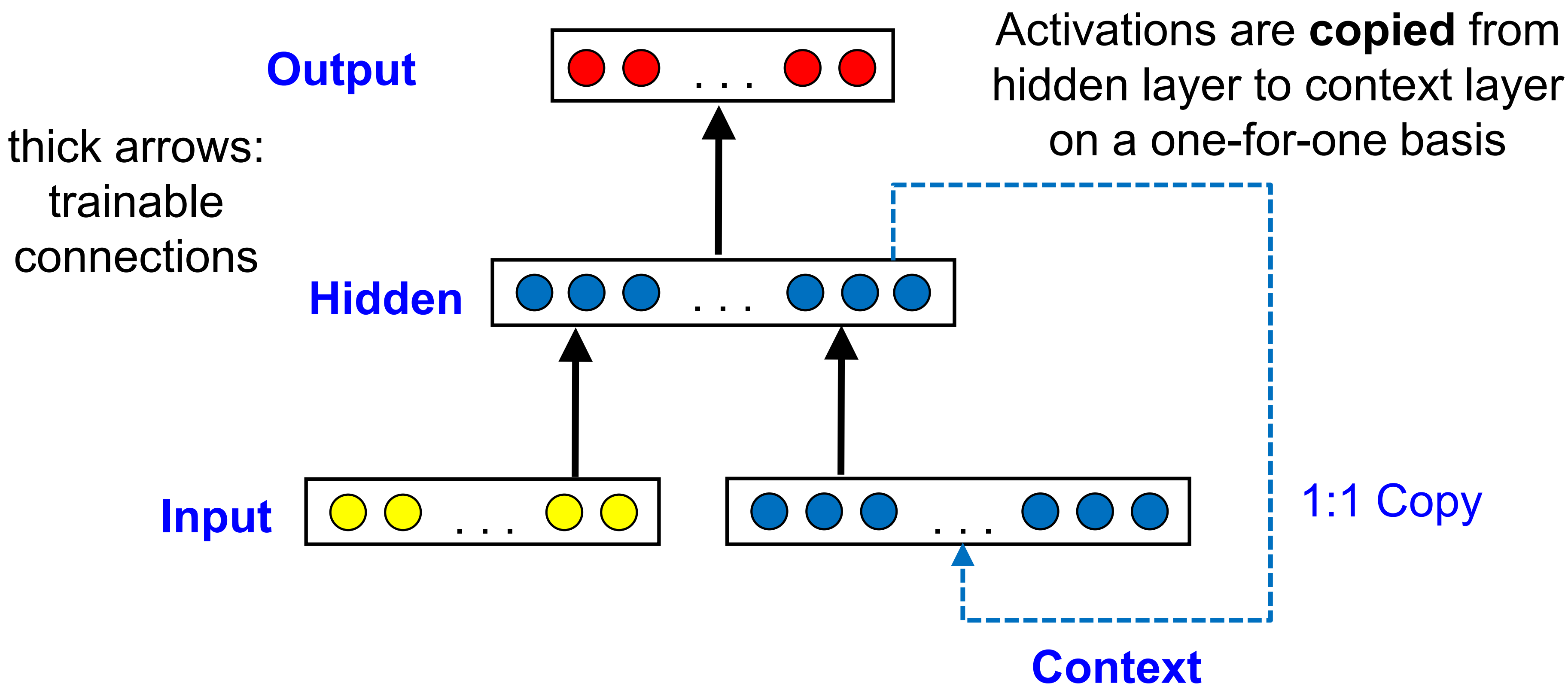
Predicting a time series depends not only on current available system variables, but also on past variables.

→ A prediction model should have a memory.

These phenomena may be

- discrete or continuous,
- uni- or multi-valued.

Simple Recurrent Network (SRN)

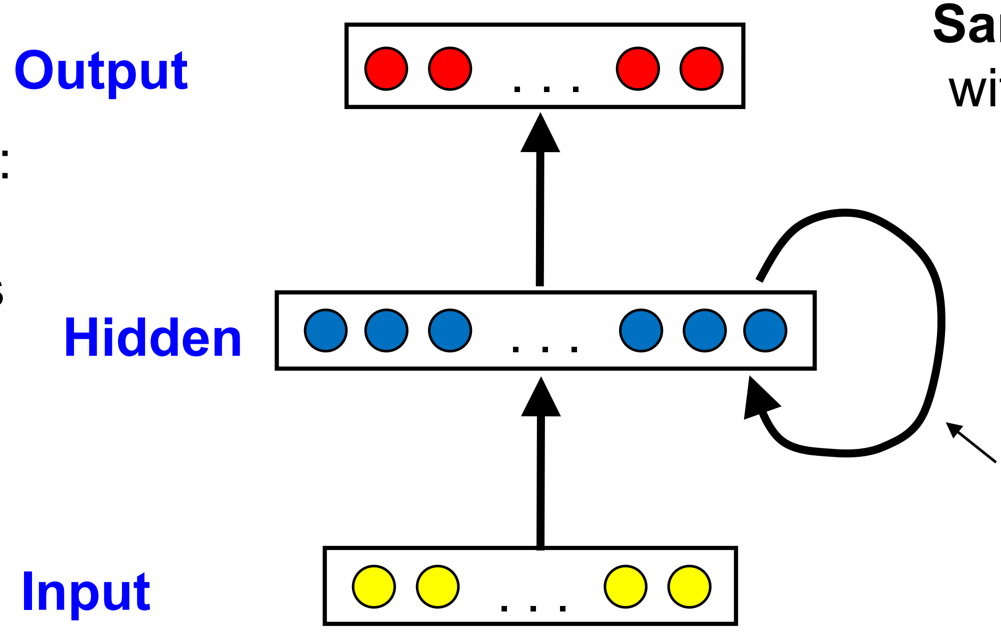


Example Prediction of words

Input: $w_1\ w_2\ w_3\ \ldots.\ w_n$

Output: $w_2\ w_3\ w_4\ \ldots.\ w_{n+1}$

Simple Recurrent Network (SRN)



thick arrows: trainable connections

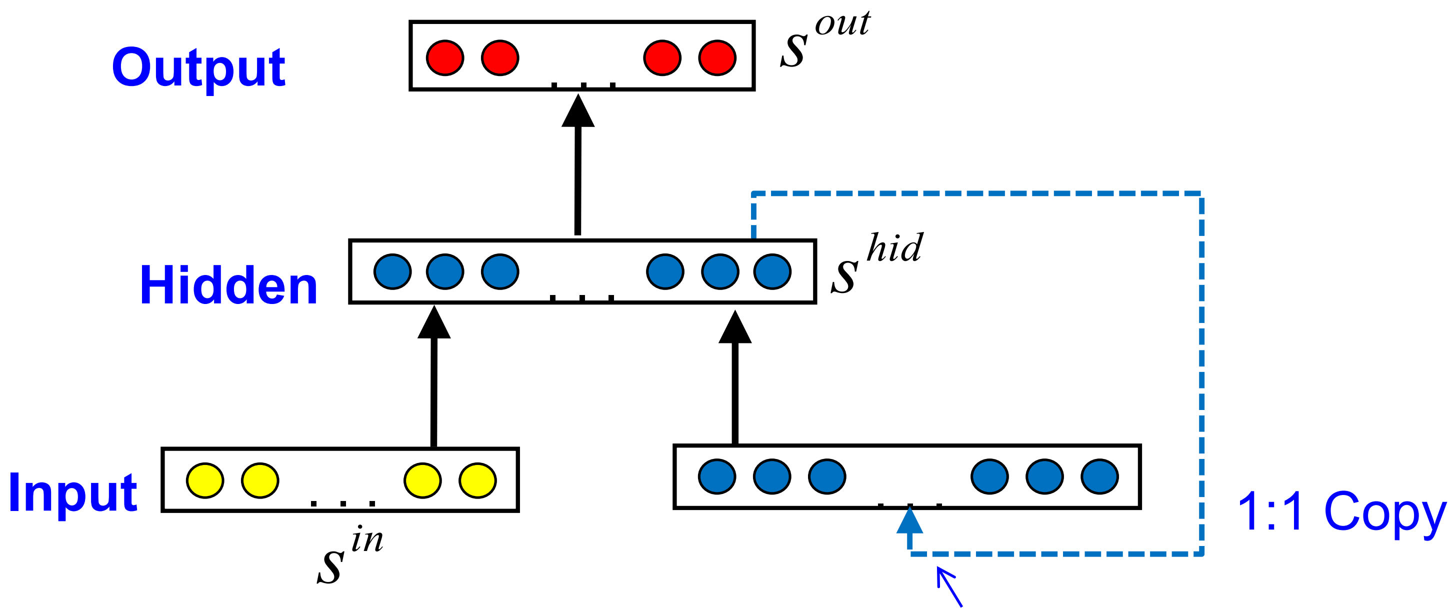
Same network, drawn without context layer

Recurrent weights: a square weight matrix

Simple Recurrent Network (SRN)

- The *hidden layer* contains information from the past, which is not contained in the current input
  - *internal state* information
- This can be implemented by a *context layer*
  - Copy the hidden layer activations for next input
- Paper: Elman J., Finding Structure in Time, Cognitive Science 14, 1990 → **Elman network**

Simple Recurrent Network (SRN)

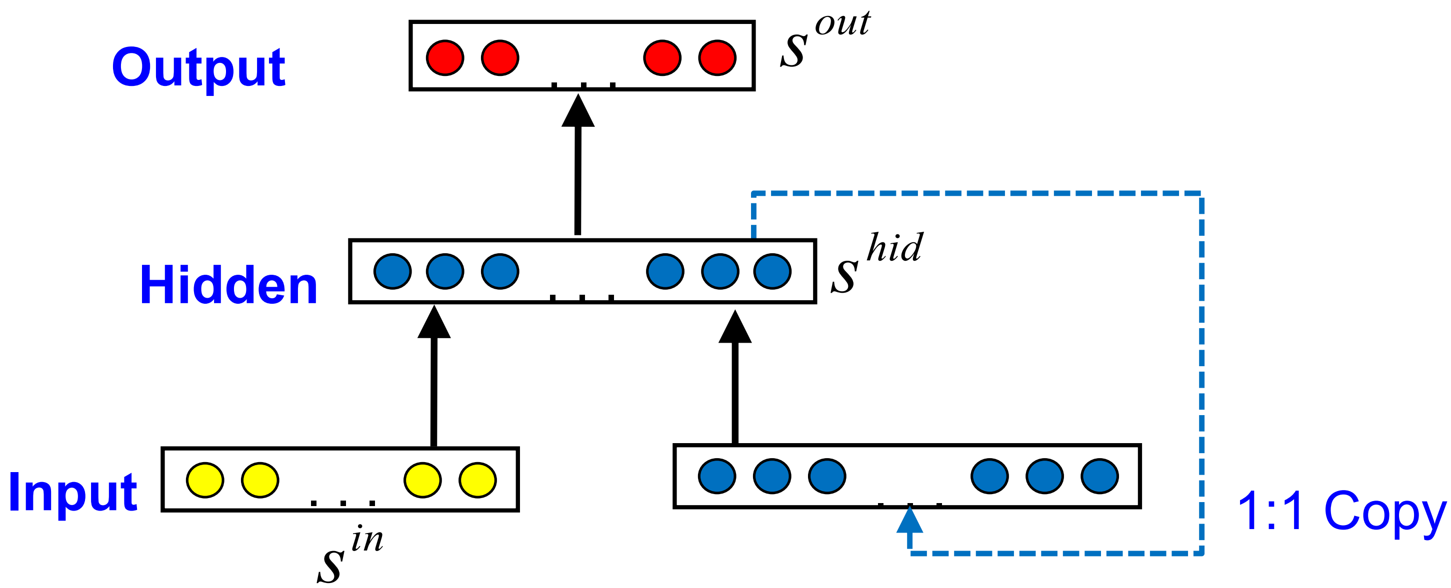


input Context

$$s_i^{hid}(t) = \text{sigmoid}\left(\sum_{k=1}^{Input} w_k^{in} s_k^{in}(t) + \sum_{j=1}^{N} w_{ij}^{rec} s_j^{hid}(t-1) + \underset{\text{bias}}{b_i}\right)$$

output layer $\longrightarrow$ $$s_l^{out}(t) = \sum_{l=1}^{N} w_{li}^{out} s_l^{hid}(t) + \underset{\text{bias}}{b_l}$$

Simple Recurrent Network (SRN)



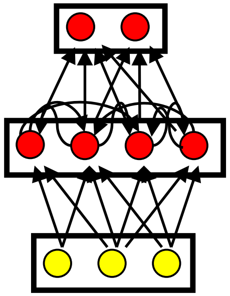
- Model of a general **dynamical system** of the form:

$$s^{hid}(t) = g(s^{in}(t), s^{hid}(t-1))$$

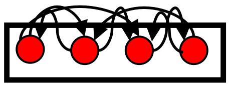
$$s^{out}(t) = h(s^{hid}(t))$$

Outline

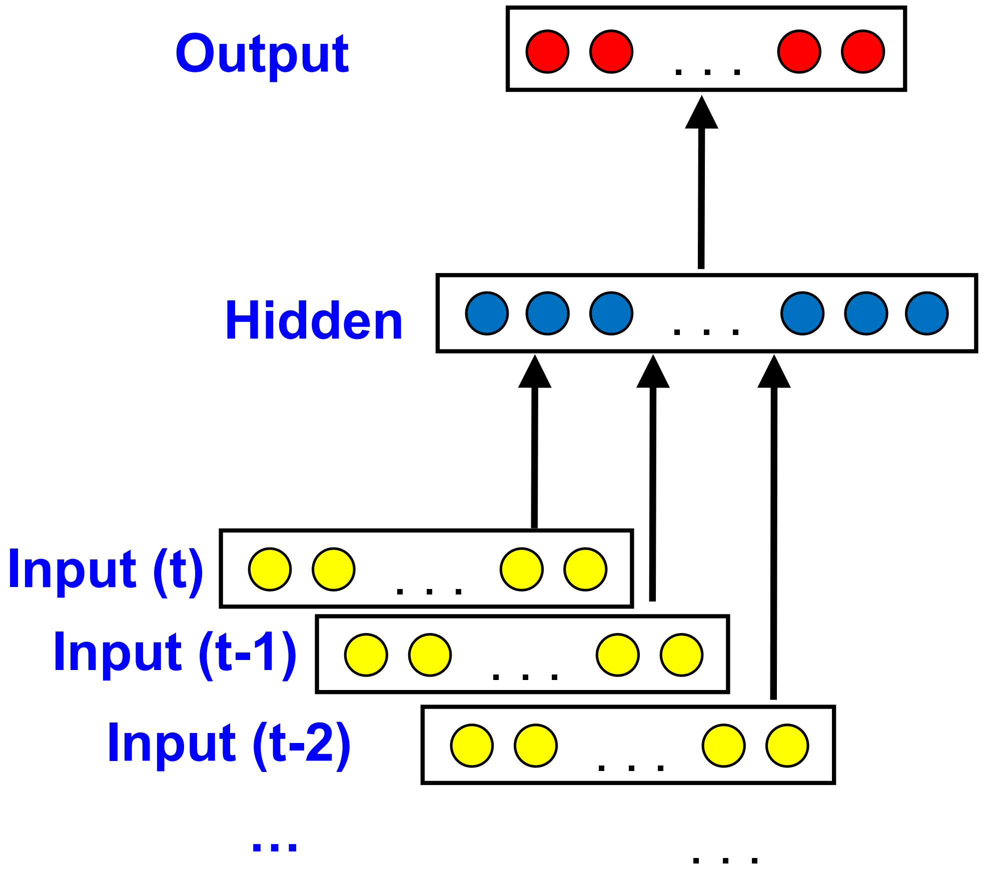
- Simple Recurrent Neural Networks (Elman)
  - ▶ Time-Delay Neural Network
  - Simple Experiments with the RNN
  - Reservoir Network & Extreme Learning Machine



- Theory of Associative Memory
  - Hopfield Network
  - Energy Function



SRN Alternative: Time Delay Neural Network



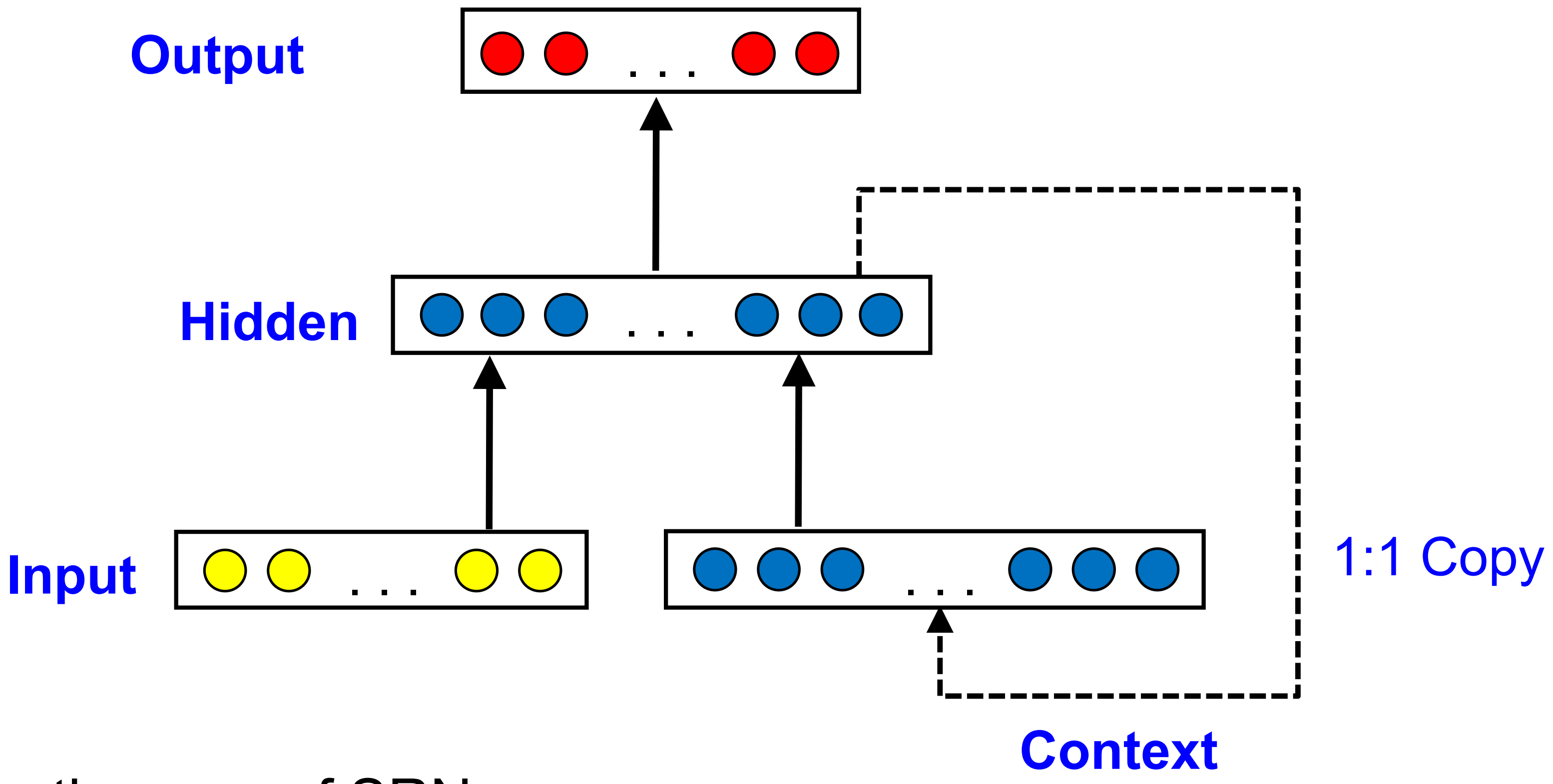
TDNN:
feedforward
architecture

Drawbacks of TDNNs:

- Increase of input dimensionality
- Many weights/parameters for large contexts
- Length of context is fixed

Temporal context realized by including ***time-delayed inputs***.

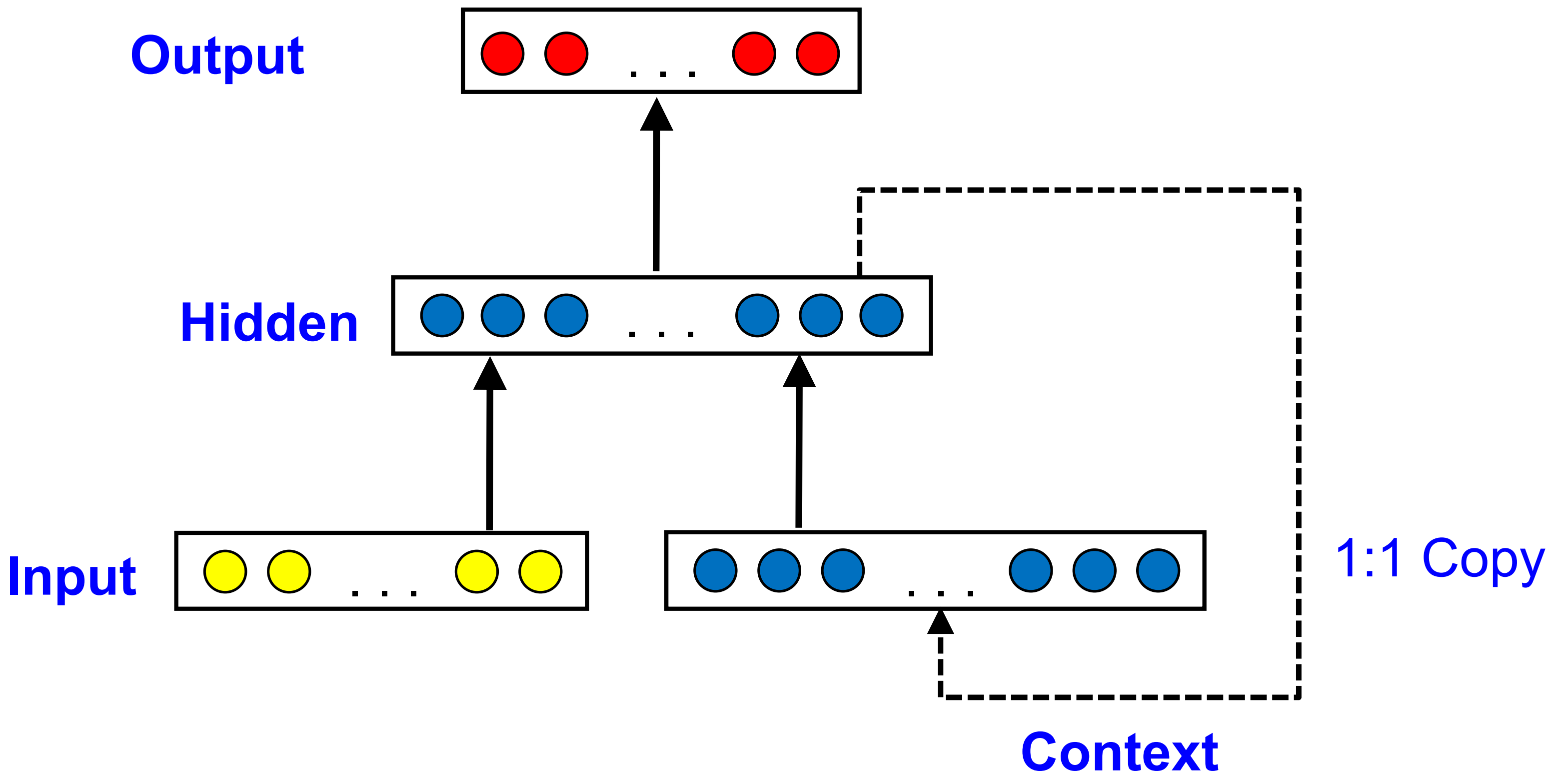
Simple Recurrent Network (SRN)



Attractiveness of SRNs:

- Old temporal context (memory) fades, but is not completely lost

Simple Recurrent Network (SRN)

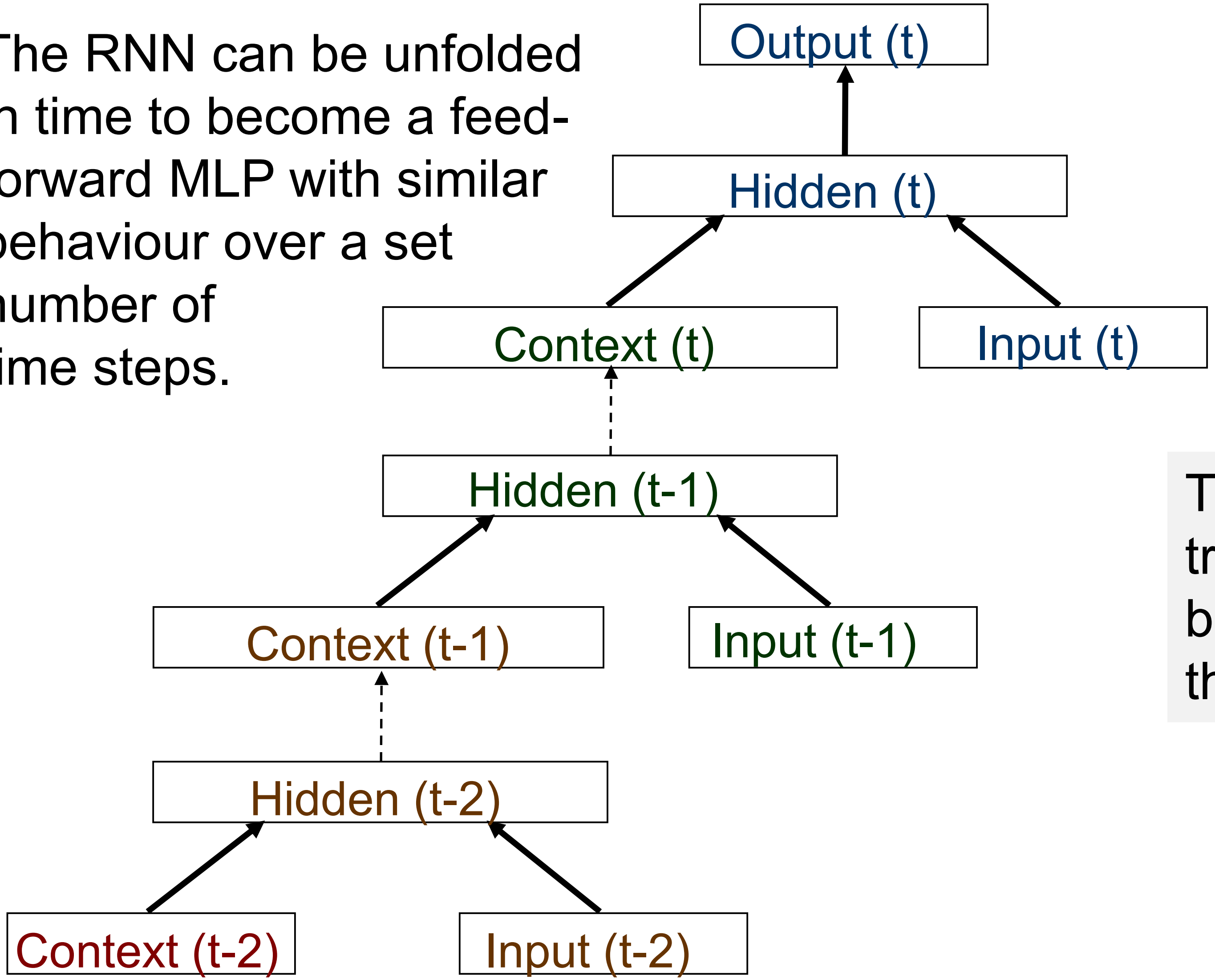


A larger hidden layer

- increases memory capacity
- yields a more complex (direct) input-output mapping

Backpropagation Through Time (e.g. 3 steps)

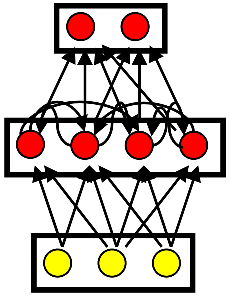
The RNN can be unfolded in time to become a feed-forward MLP with similar behaviour over a set number of time steps.



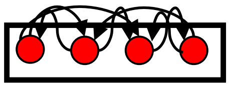
This is used for training: error backpropagation through time

Outline

- Simple Recurrent Neural Networks (Elman)
  - Time-Delay Neural Network
  - Simple Experiments with the RNN
  - Reservoir Network & Extreme Learning Machine



- Theory of Associative Memory
  - Hopfield Network
  - Energy Function



Some Controlled Data Experiments: Learning Structure from Letter Sequences

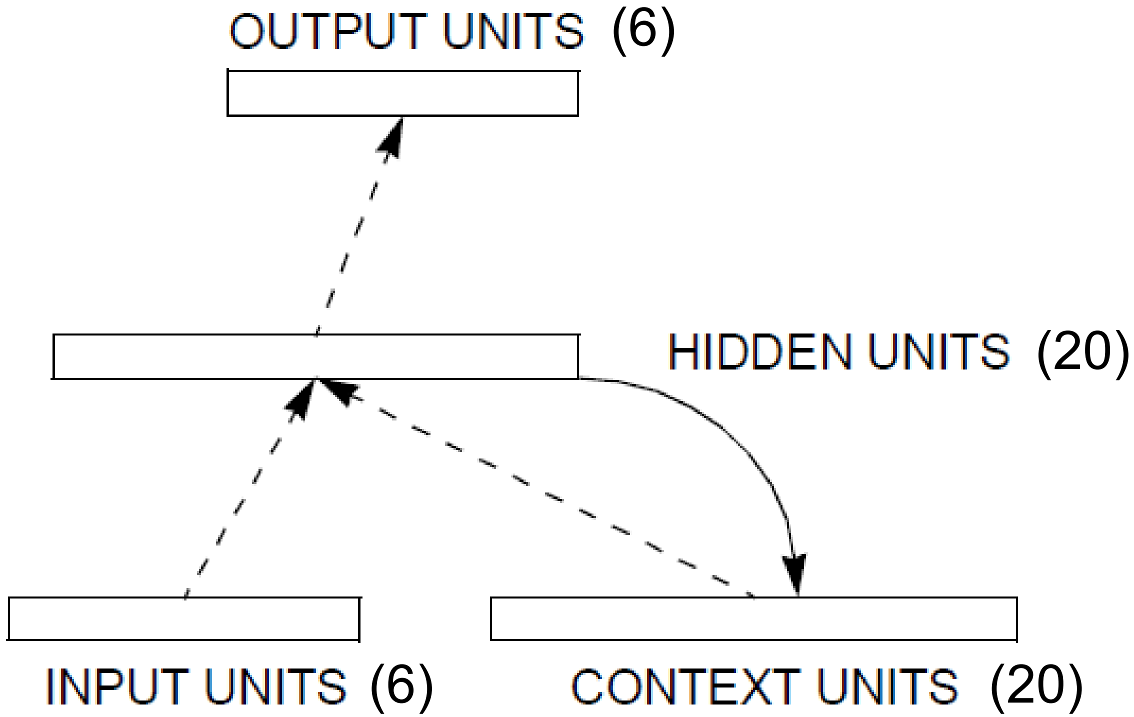
- Multi-bit inputs with temporal extent and longer sequences
- 3 consonants (**b**, **d**, **g**) combined in random order to obtain 1000-letter sequence. Then each consonant complemented with vowels using rules
  - b $\rightarrow$ ba
  - d $\rightarrow$ dii
  - g $\rightarrow$ guuu
- **Example**: dbgbddg $\rightarrow$ diibaguuubadiidiiguuu
- Task: predict next letter.
  There is uncertainty: sometimes the prediction is clear, sometimes not

Vector Definitions of Alphabet

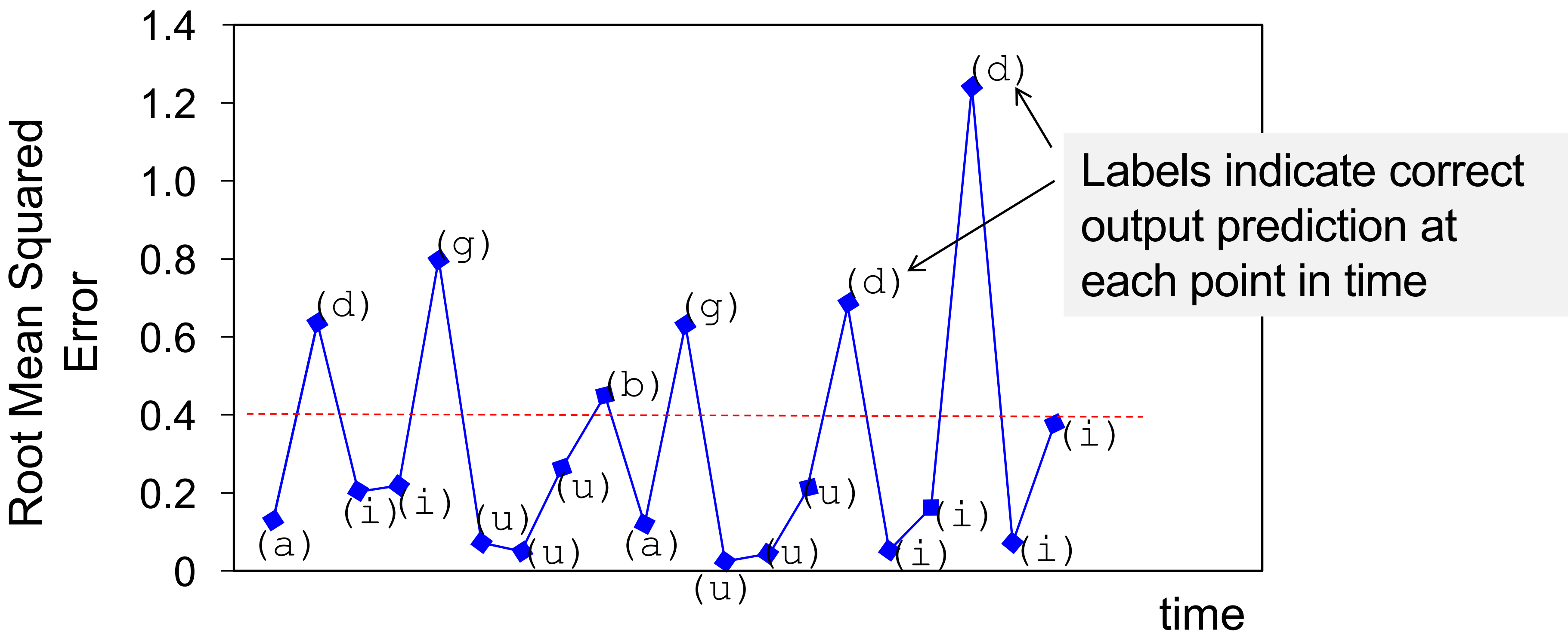
	Consonant	Vowel	Interrupted	High	Back	Voiced
b	[1	0	1	0	0	1]
d	[1	0	1	1	0	1]
g	[1	0	1	0	1	1]
a	[0	1	0	0	1	1]
i	[0	1	0	1	0	1]
u	[0	1	0	1	1	1]

- Each letter encoded by 6 binary features

SRN for Letter Sequences



Error in Letter Prediction Task



The network learnt the most important regularities:

- it predicts the correct vowels, which follow a consonant
- it cannot predict the next consonant

Can we Learn / Mine Lexical Classes from Word Order

- Order of words is constraint
- Can a network learn ***structure*** from order?

Scenario:

- Sentence generator based on categories of lexical items
- Each word represented by random 31 bit vector:
  - one bit is ON if word present, others OFF (one-hot encoding)
- 27,354 word vectors were concatenated
  into 10,000 sentences

Categories of Lexical Items

Category	Examples
NOUN-HUM	man, woman
NOUN-ANIM	cat, mouse
NOUN-INANIM	book, rock
NOUN-AGRESS	dragon, monster
NOUN-FRAG	glass, plate
NOUN-FOOD	cookie, sandwich
VERB-INTRAN	think, sleep
VERB-TRAN	see, chase
VERB-AGPA	move, break
VERB-PERCEPT	smell, see
VERB-DESTROY	break, smash
VERB-EA	eat

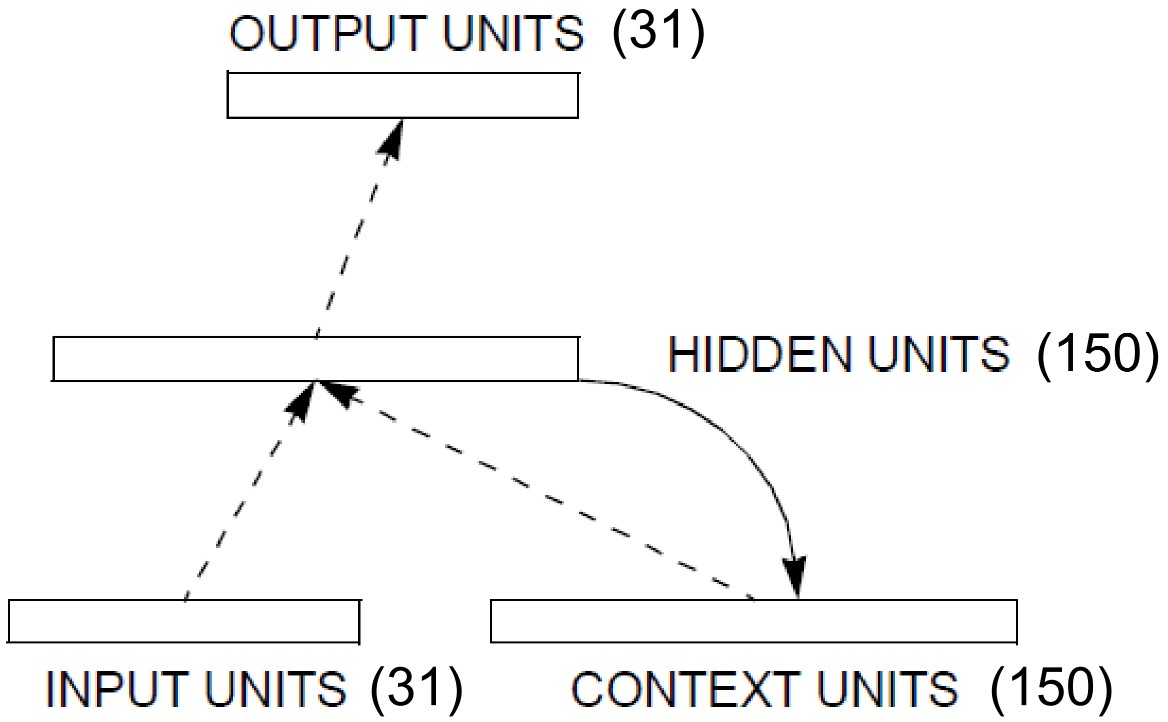
Templates for Sentence Generator

WORD 1	WORDS	WORD 3
NOUN-HUM	VERB-EAT	NOUN-FOOD
NOUN-HUM	VERB-PERCEPT	NOUN-INANIM
NOUN-HUM	VERB-DESTROY	NOUN-FRAG
NOUN-HUM	VERB-INTRAN	
NOUN-HUM	VERB-TRAN	NOUN-HUM
NOUN-HUM	VERB-AGPAT	NOUN-INANIM
NOUN-HUM	VERB-AGPAT	
NOUN-ANIM	VERB-EAT	NOUN-FOOD
NOUN-ANIM	VERB-TRAN	NOUN-ANIM
NOUN-ANIM	VERB-AGPAT	NOUN-INANIM
NOUN-ANIM	VERB-AGPAT	
NOUN-INANIM	VERB-AGPAT	
NOUN-AGRESS	VERB-DESTORY	NOUN-FRAG
NOUN-AGRESS	VERB-EAT	NOUN-HUM
NOUN-AGRESS	VERB-EAT	NOUN-ANIM
NOUN-AGRESS	VERB-EAT	NOUN-FOOD

Learning Successive Words

INPUT		OUTPUT	
0000000000000000000000000000010	(woman)	0000000000000000000000000010000	(smash)
0000000000000000000000000010000	(smash)	0000000000000000000001000000000	(plate)
0000000000000000000001000000000	(plate)	0000010000000000000000000000000	(cat)
0000010000000000000000000000000	(cat)	0000000000000000000100000000000	(move)
0000000000000000000100000000000	(move)	0000000000000000100000000000000	(man)
0000000000000000100000000000000	(man)	0001000000000000000000000000000	(break)
0001000000000000000000000000000	(break)	0000100000000000000000000000000	(car)
0000100000000000000000000000000	(car)	0100000000000000000000000000000	(boy)
0100000000000000000000000000000	(boy)	0000000000000000000100000000000	(move)
0000000000000000000100000000000	(move)	0000000000001000000000000000000	(girl)
0000000000001000000000000000000	(girl)	0000000000100000000000000000000	(eat)
0000000000100000000000000000000	(eat)	0010000000000000000000000000000	(bread)
0010000000000000000000000000000	(bread)	0000000010000000000000000000000	(dog)
0000000010000000000000000000000	(dog)	0000000000000000000100000000000	(move)
0000000000000000000100000000000	(move)	0000000000000000001000000000000	(mouse)
0000000000000000001000000000000	(mouse)	0000000000000000001000000000000	(mouse)
0000000000000000001000000000000	(mouse)	0000000000000000000100000000000	(move)
0000000000000000000100000000000	(move)	1000000000000000000000000000000	(book)
1000000000000000000000000000000	(book)	0000000000000001000000000000000	(lion

SRN for Word Sequences

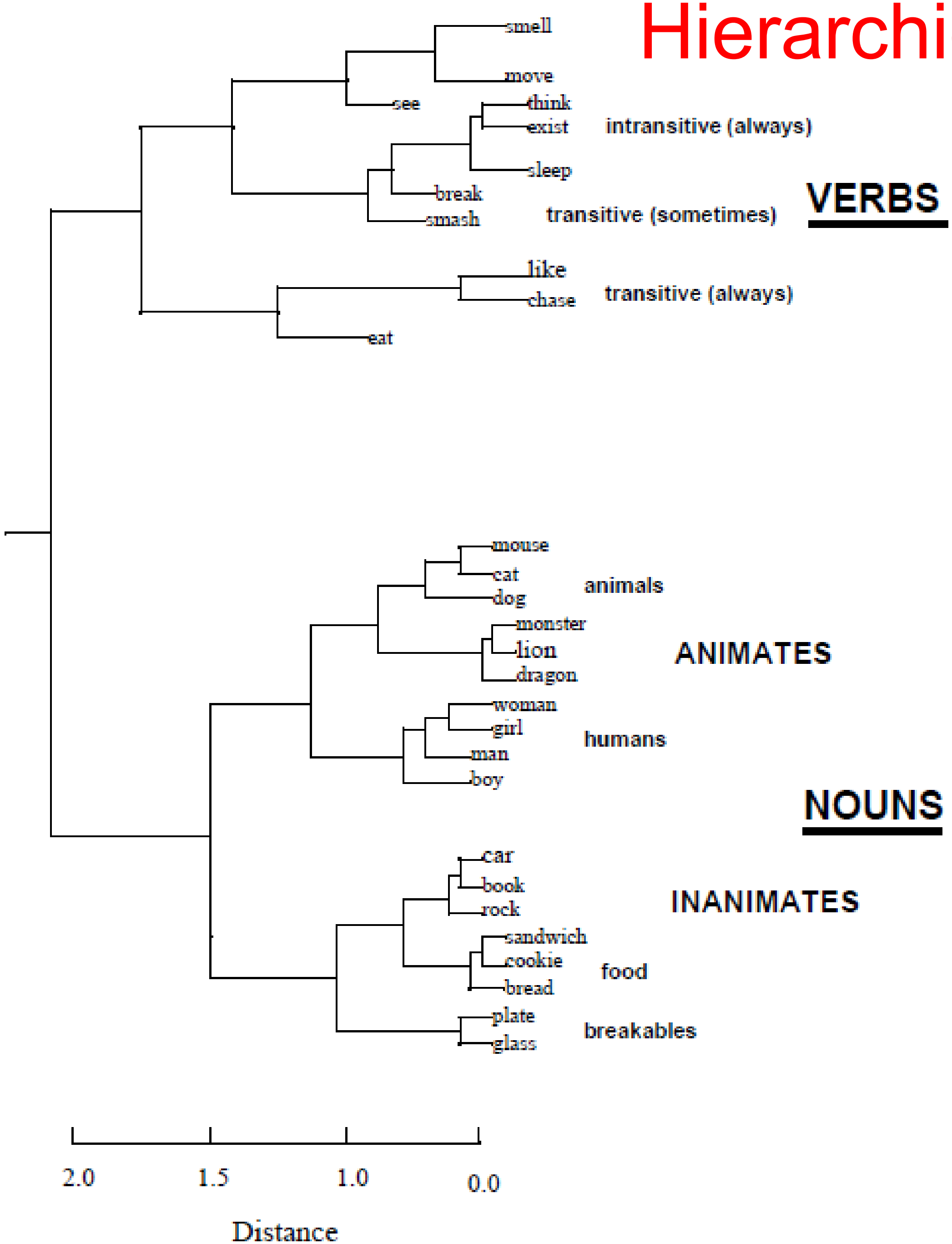


SRN for Word Sequences: Hidden Layer Representations

- Representation of the word “man”:
  - In the input layer, a 31-dimensional one-hot vector
  - In the hidden layer, a 150-dimensional continuous vector
- Hidden representation depends
  - On current input (always same for “man”)
  - On context (varies a lot)

  ← It’s not just by linear superposition of these two!
- Obtain representative hidden representation (for display): average over many occurrences of “man” in many contexts

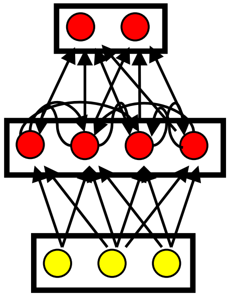
Hierarchical Cluster Analysis of Hidden Layer Representations



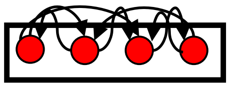
Words that are used similarly appear in similar areas of the dendrogram

Outline

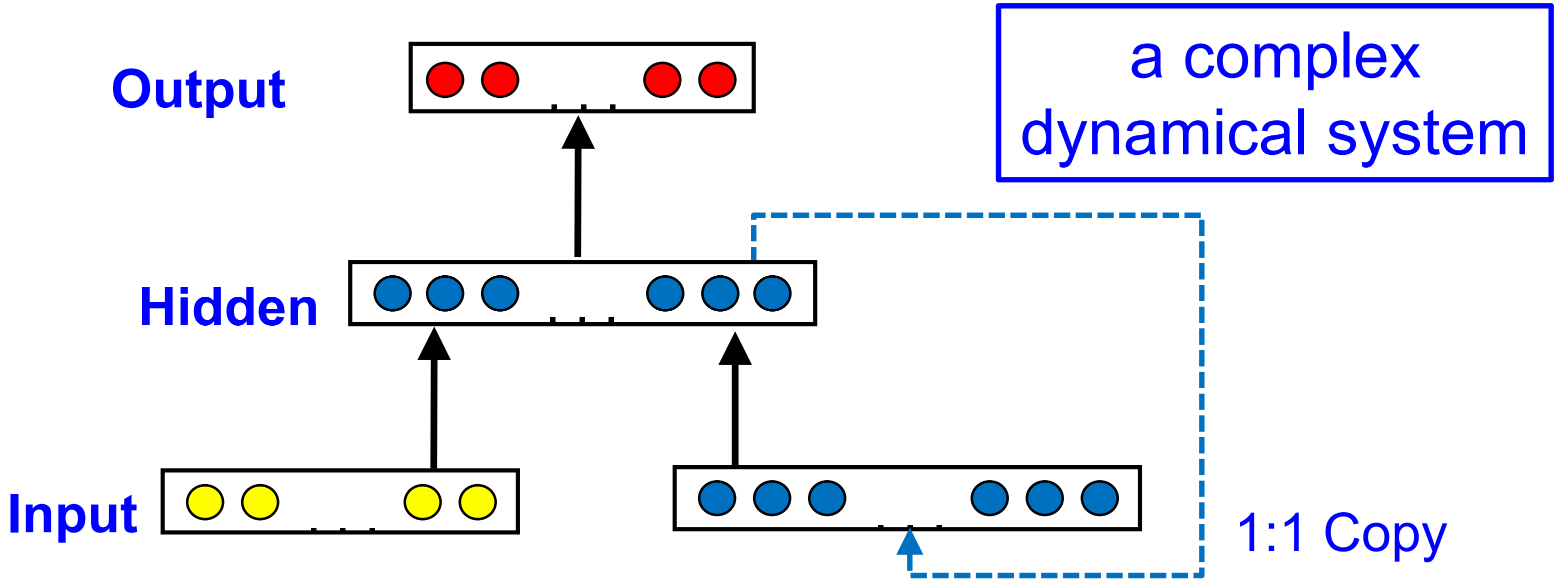
- Simple Recurrent Neural Networks (Elman)
  - Time-Delay Neural Network
  - Simple Experiments with the RNN
  - Reservoir Network & Extreme Learning Machine



- Theory of Associative Memory
  - Hopfield Network
  - Energy Function



SRN Variation: Echo State Network (1/4)

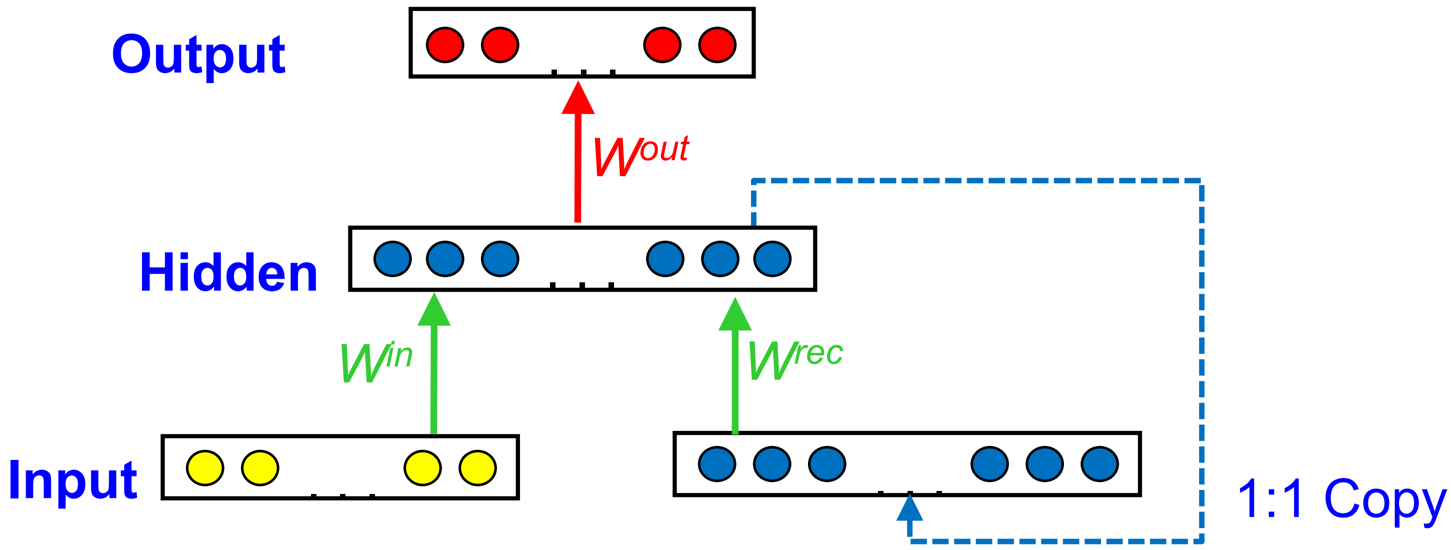


For a very large hidden layer:

- Different histories of inputs will lead to different activations, if recurrent/input weights are random (even if not trained!)
- **ESN**: Train only the output weights (linear perceptron)!

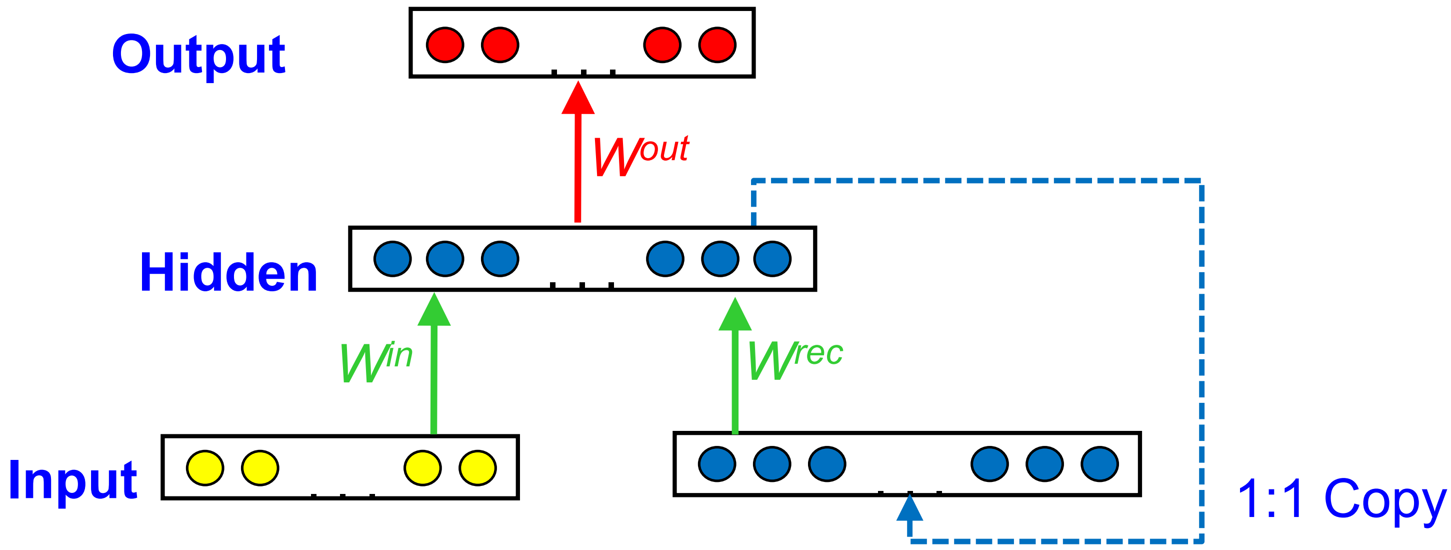
→ perceptron's high-dimensional input allows all classifications

SRN Variation: Echo State Network (2/4)



- W^{rec} *fixed*, random, sparse, non-symmetric
- W^{in} *fixed*, random, sparse
- Train only W^{out} $\leftarrow$ supervised perceptron learning
- N^{hid} large, hidden layer acts as a **reservoir**

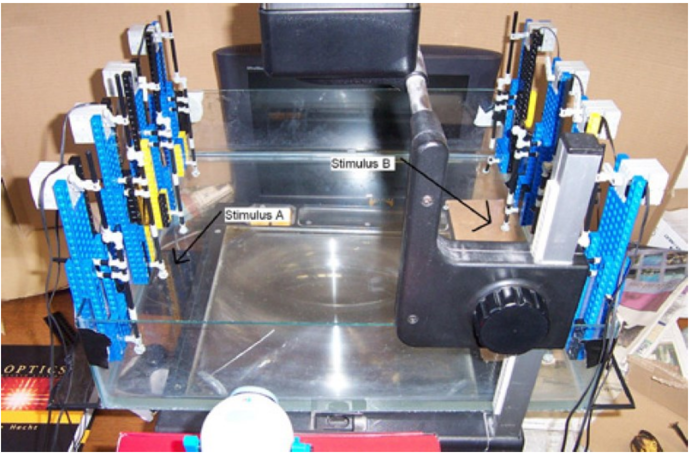
SRN Variation: Echo State Network (3/4)



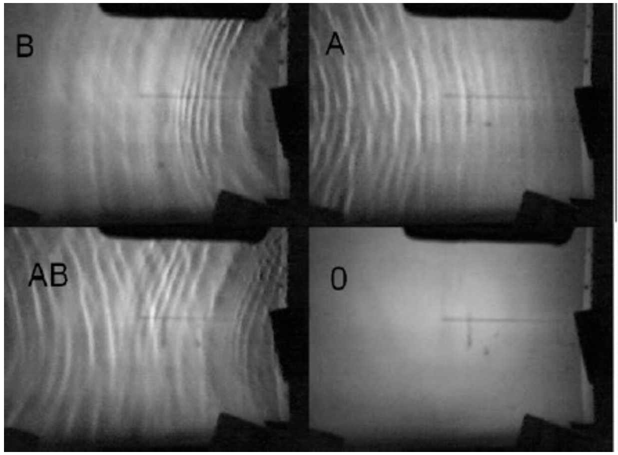
- ESNs must fulfill the echo state property: the largest eigenvalue of W^{rec} must not be much larger than 1
- In practice this means that weights must not be too large
  - → activations will slowly decay in time, but not blow up
  - → similar input histories will be similarly represented

Water Reservoir Networks (4/4)

LEGO bumpers supply inputs

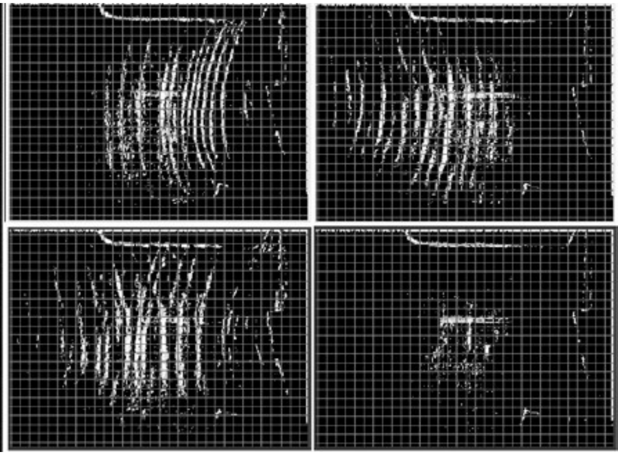


01 10



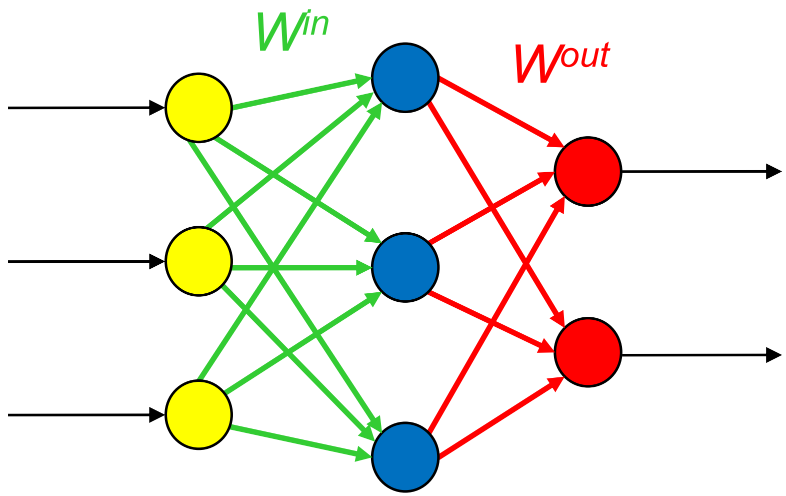
11 00

Sobel filtered and thresholded images of the reservoir states



- Perceptron solves (static) XOR with input from reservoir
- Activity on water surface contains all relevant information
- High complexity
- Inherent stability to wide range of inputs

MLP with Partially Fixed Weights

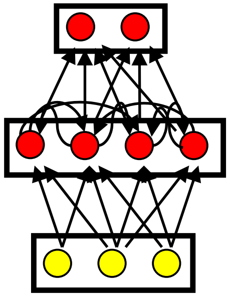


- W^{in} ***fixed***, random
- W^{out} trained
- N^{hid} large

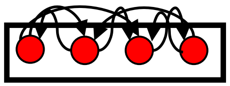
- "*Extreme Learning Machine*" maybe inspired by ESN or RBF
- Like ESN, requires large number of hidden neurons
- Fast to train
- Difficult / data-dependent setting of parameters, e.g. scaling of weights and their sparseness

Outline

- Simple Recurrent Neural Networks (Elman)
  - Time-Delay Neural Network
  - Simple Experiments with the RNN
  - Reservoir Network & Extreme Learning Machine



- Theory of Associative Memory
  - Hopfield Network
  - Energy Function



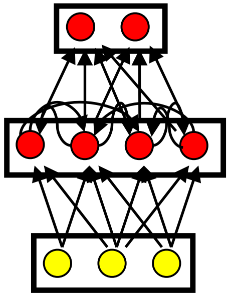
Associative Memory

What is “behind” recurrent neural networks?

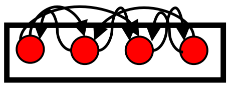
- We associate similar events/items/patterns in our brains
- We generalise from patterns we have seen
  - **Example**: recognize letters in different fonts/sizes/colors
- We complete incomplete information
- We recall memories
- This is ***Context-Addressable*** or ***Associative*** Memory

Outline

- Simple Recurrent Neural Networks (Elman)
  - Time-Delay Neural Network
  - Simple Experiments with the RNN
  - Reservoir Network & Extreme Learning Machine

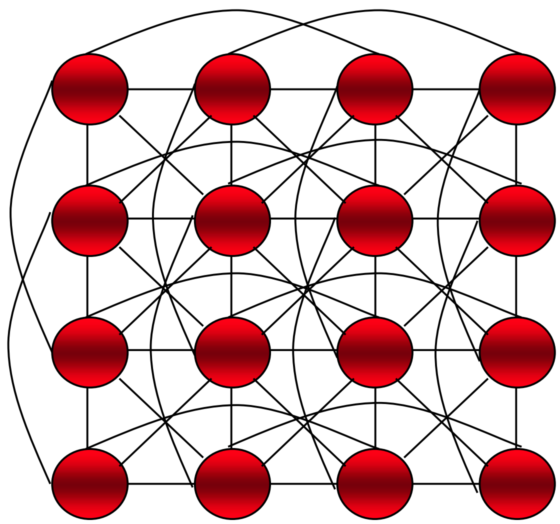


- Theory of Associative Memory
  - Hopfield Network
  - Energy Function



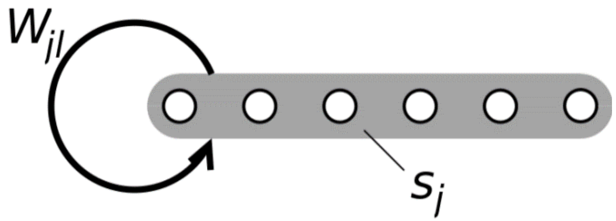
The Hopfield Network as Associative Memory

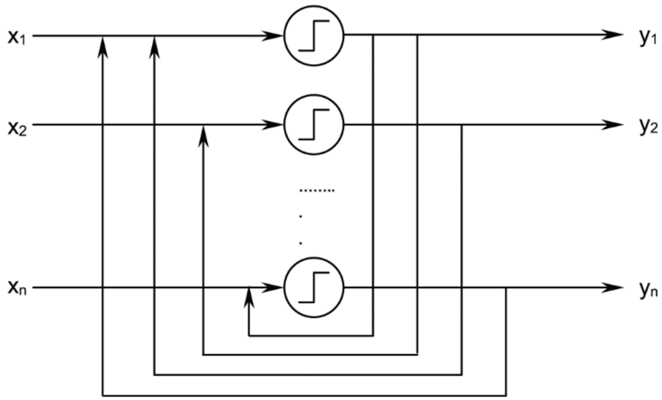
- An example of a network of simple neurons
  - Perceptrons
  - Binary Threshold Units

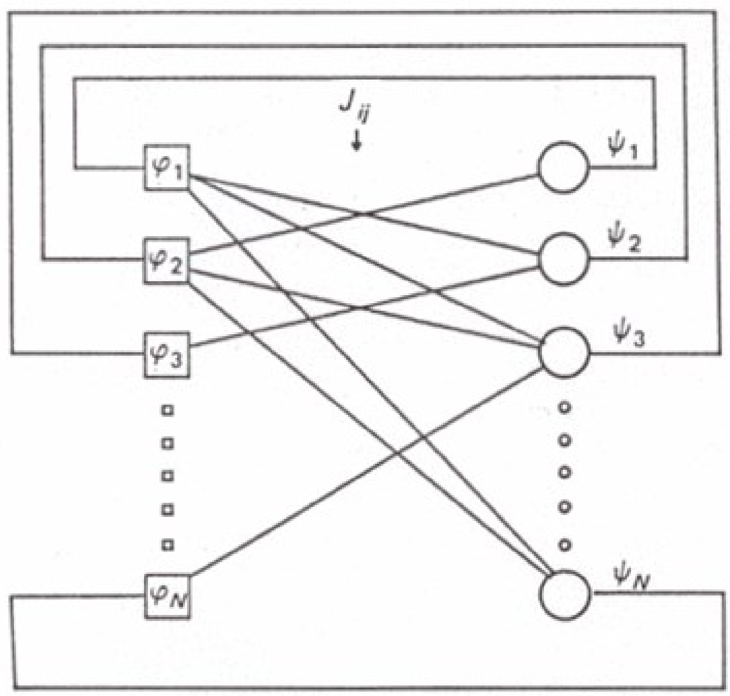


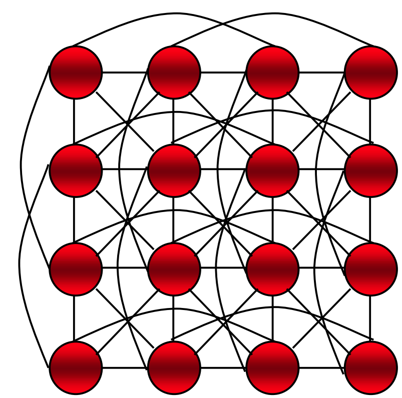
- Symmetric weights
- No self-connection
- Invented by J. Hopfield

The Hopfield Network – Differently Displayed









The Hopfield Network – An Attractor Network

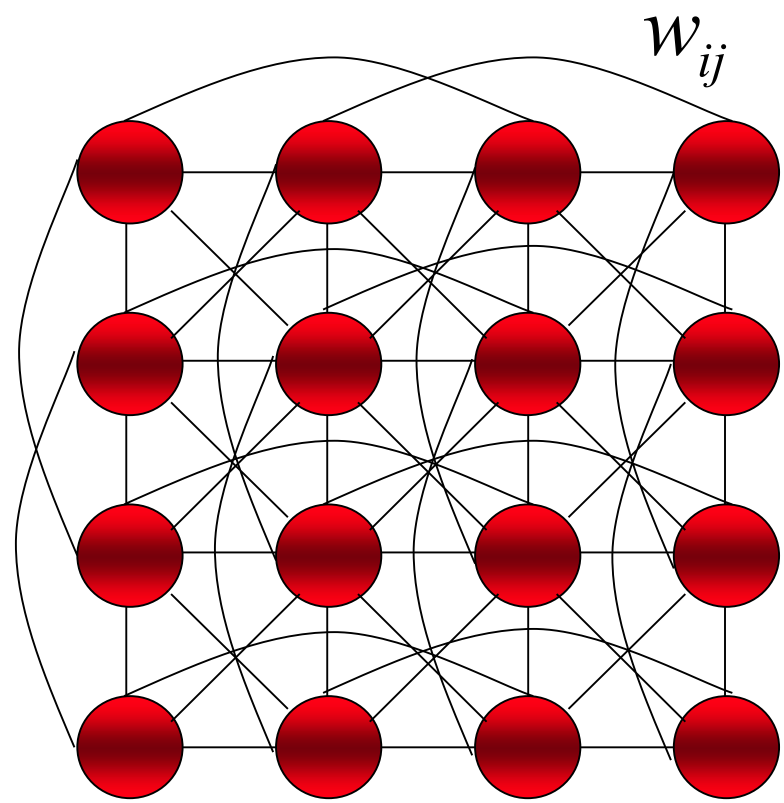
- Every neuron connected to every other neuron
- Activations are ±1

$$s_i = \mathrm{sign}\left(\sum_{j=1}^{N} w_{ij} s_j\right)$$

N = # neurons

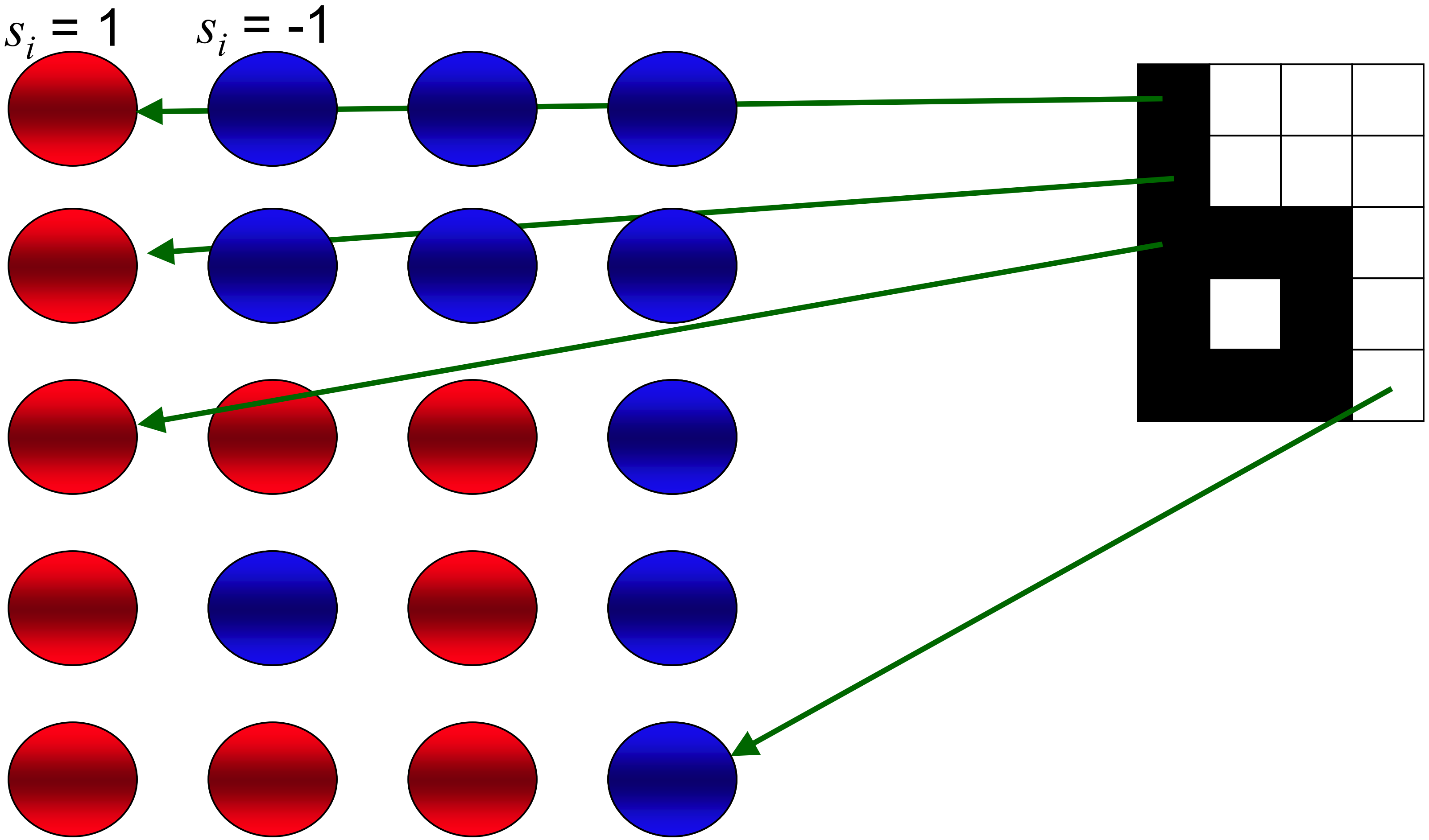
square weight matrix

- Synchronous or random update



Here w_{ij} is the weight from j to i, as clear from context. Different conventions can be found in different books.

Representing Images



Learning One Pattern

- Set the weights as: $w_{ij} = s_i^p s_j^p$

Hebbian rule
for pattern s^p

If # patterns = 1:

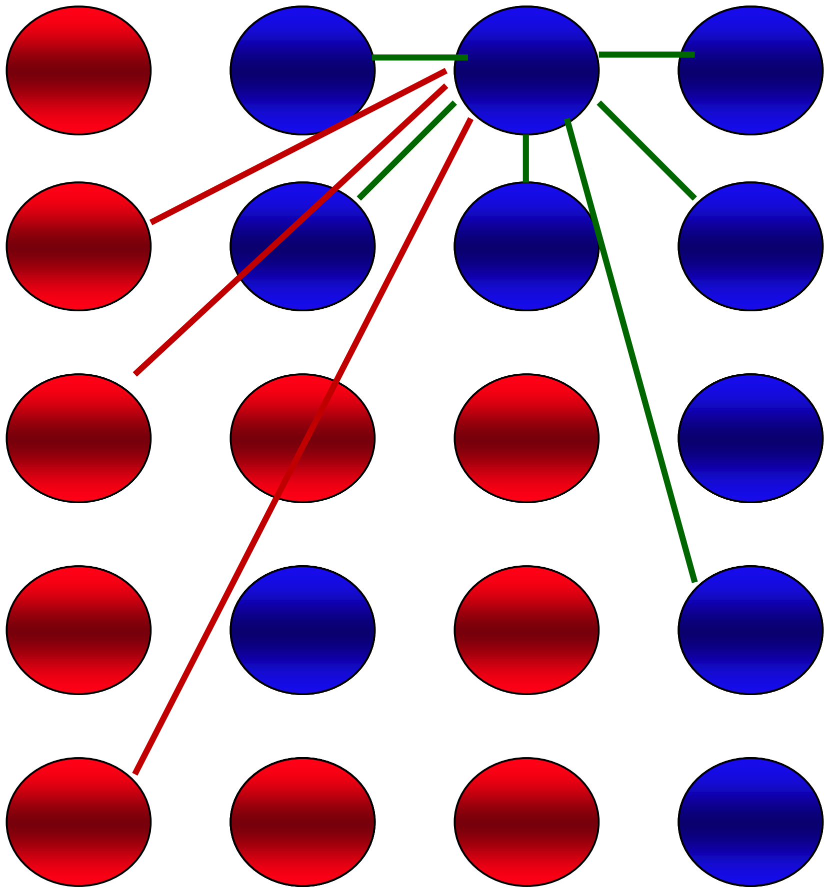
- w_{ij} = 1 if both units *i* and *j* have activity 1, or both -1
- w_{ij} = -1 if both units' activities have opposing sign

- The stored pattern s^p will be stable under activity update:

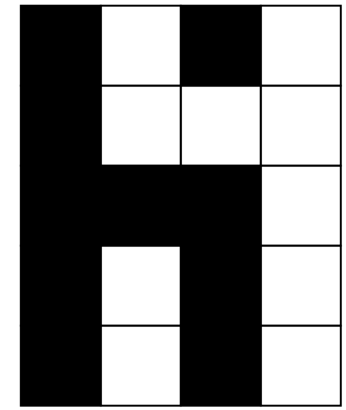
$$s_i = \text{sign}\left(\sum_{j=1}^{N} w_{ij} s_j \right)$$

$$\overset{s_j = s_j^p,\ w_{ij} = s_i^p s_j^p}{=} \quad \text{sign}\left(\sum_{j=1}^{N} s_i^p \underbrace{s_j^p s_j^p}_{1} \right) = s_i^p$$

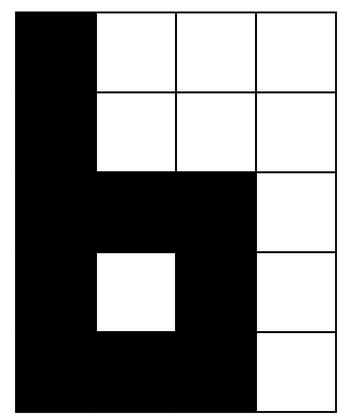
Using the Memory



initial
pattern
…



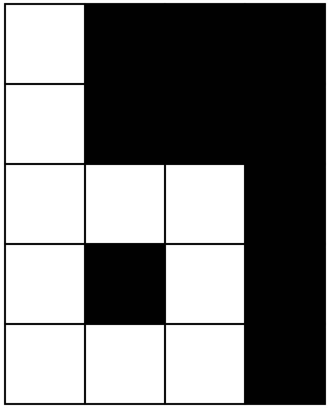
… iterative update of
neuron activations …



… final
pattern

Attractors

- Applying the activity update iteratively will make random initial patterns converge to the stored pattern
- The final pattern is an ***attractor***
- Useful for
  - constraint satisfaction
  - noisy preclassification
  - known attractors



More Patterns in the Memory

- Need to learn many patterns s^p, not just one
- Set the weights:

$$w_{ij} = \frac{1}{P} \sum_{p}^{P} s_i^p s_j^p$$

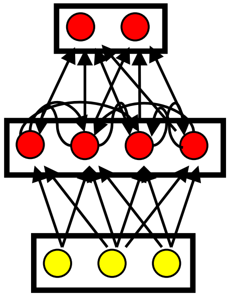
P = # patterns

Idea: For any input, network activations shall ***converge*** to the closest trained pattern

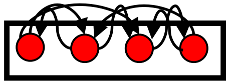
- Problem: multiple patterns may interfere
- The Hebbian rule leads to a memory capacity of ~0.13 N, where N = number of neurons. (under ideal conditions, i.e. little overlap between the patterns)

Outline

- Simple Recurrent Neural Networks (Elman)
  - Time-Delay Neural Network
  - Simple Experiments with the RNN
  - Reservoir Network & Extreme Learning Machine



- Theory of Associative Memory
  - Hopfield Network
  - Energy Function

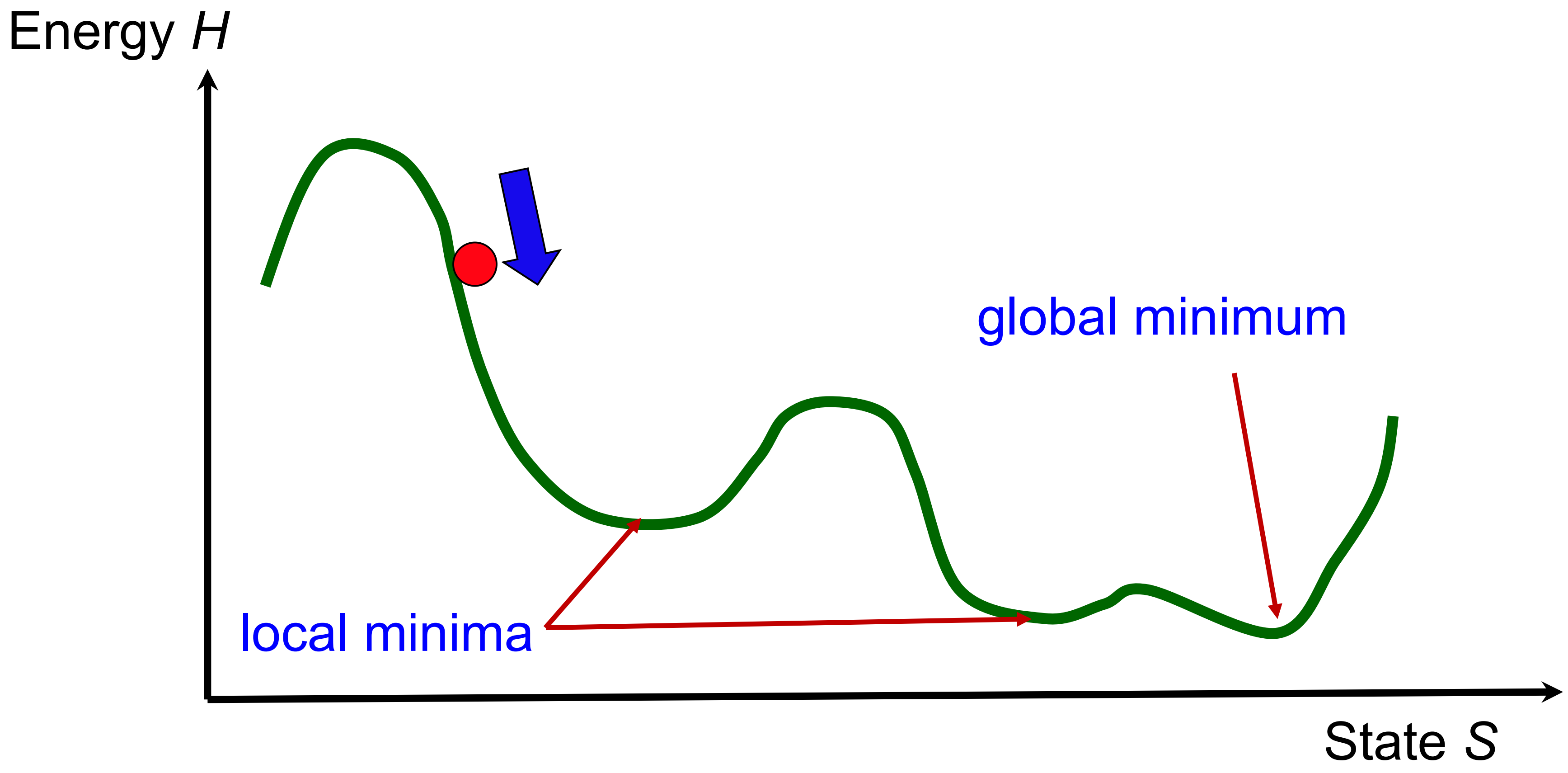


"Energy" in the Network

$$H = -\frac{1}{2}\sum_i \sum_j w_{ij} s_i s_j$$

- The energy contained in the network activations
  - Low energy if $s_i \cdot s_j$ tend to match the sign of w_{ij}
  - High energy if $s_i \cdot s_j$ tend not to match the sign of w_{ij}
- Asynchronous activity update: *H* never increases
  - The proof requires symmetric weights: $w_{ij} = w_{ji}$
- The energy decreases as the network activities stabilize
- The attractors are the local minima of the energy function

Energy Landscapes



Similar as in the context of Error function minimization, but now, ***states***, i.e. activations (**not**: weights) are on the x-axis

"Energy" in the Network

$$H = -\frac{1}{2}\sum_i \sum_j w_{ij} s_i s_j$$

- Positive w_{ij} $\rightarrow$ H becomes small if $s_i = s_j$
- Negative w_{ij} $\rightarrow$ H becomes small if $s_i = -s_j$
- Large $|w_{ij}|$ $\rightarrow$ more importance

$\rightarrow$ a minimum of H is found by multiple constraint satisfaction

- **Symmetry**: $H(S) = H(-S)$
  (all activations sign-reversed)

"Energy" in the Network

$$H = -\frac{1}{2}\sum_i \sum_j w_{ij} s_i s_j$$

- Why is the Hebb rule good? $w_{ij} = s_i^p s_j^p$ (***for one pattern***)

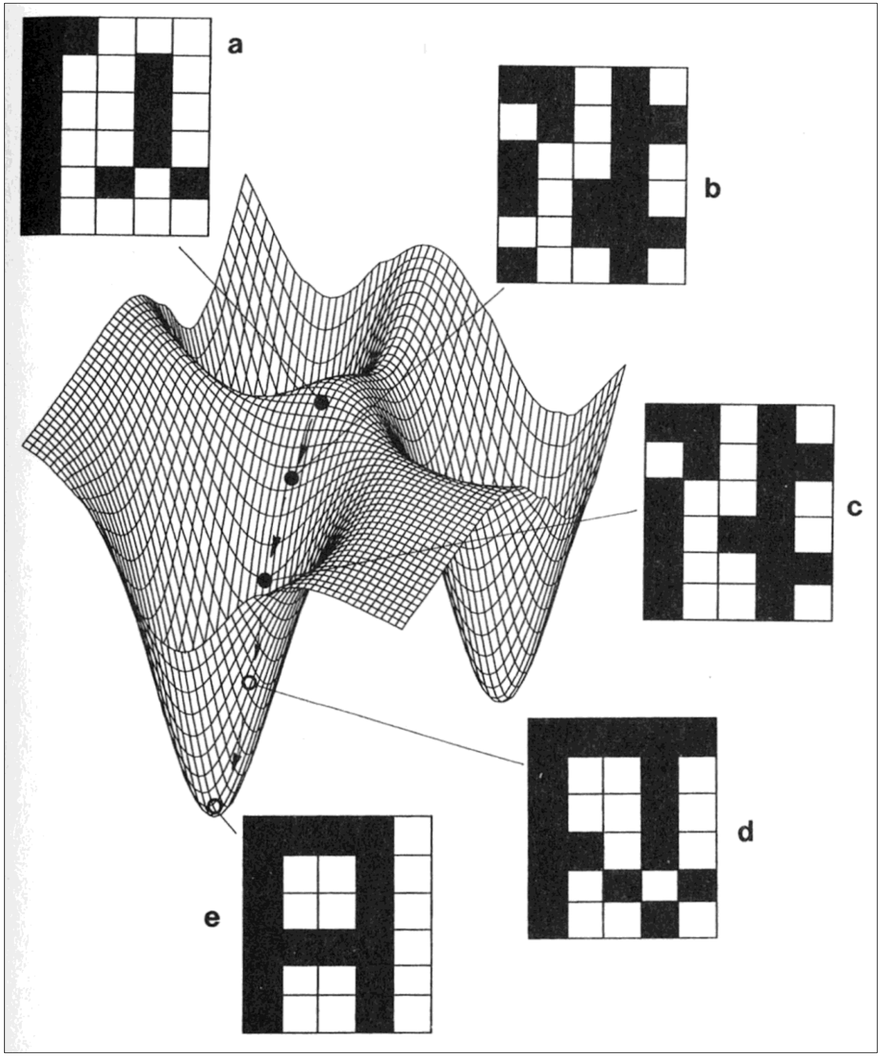
  Insert it:

$$H = -\frac{1}{2}\sum_i \sum_j s_i^p s_j^p s_i s_j = -\frac{1}{2}\sum_i s_i^p s_i \sum_j s_j^p s_j$$

  $\rightarrow$ *H* is minimised if $s_i = s_i^p$ and $s_j = s_j^p$ for all *i*, *j*.

- A single pattern s^p stored by Hebb rule has minimal energy
  (not necessarily true for multiple patterns).

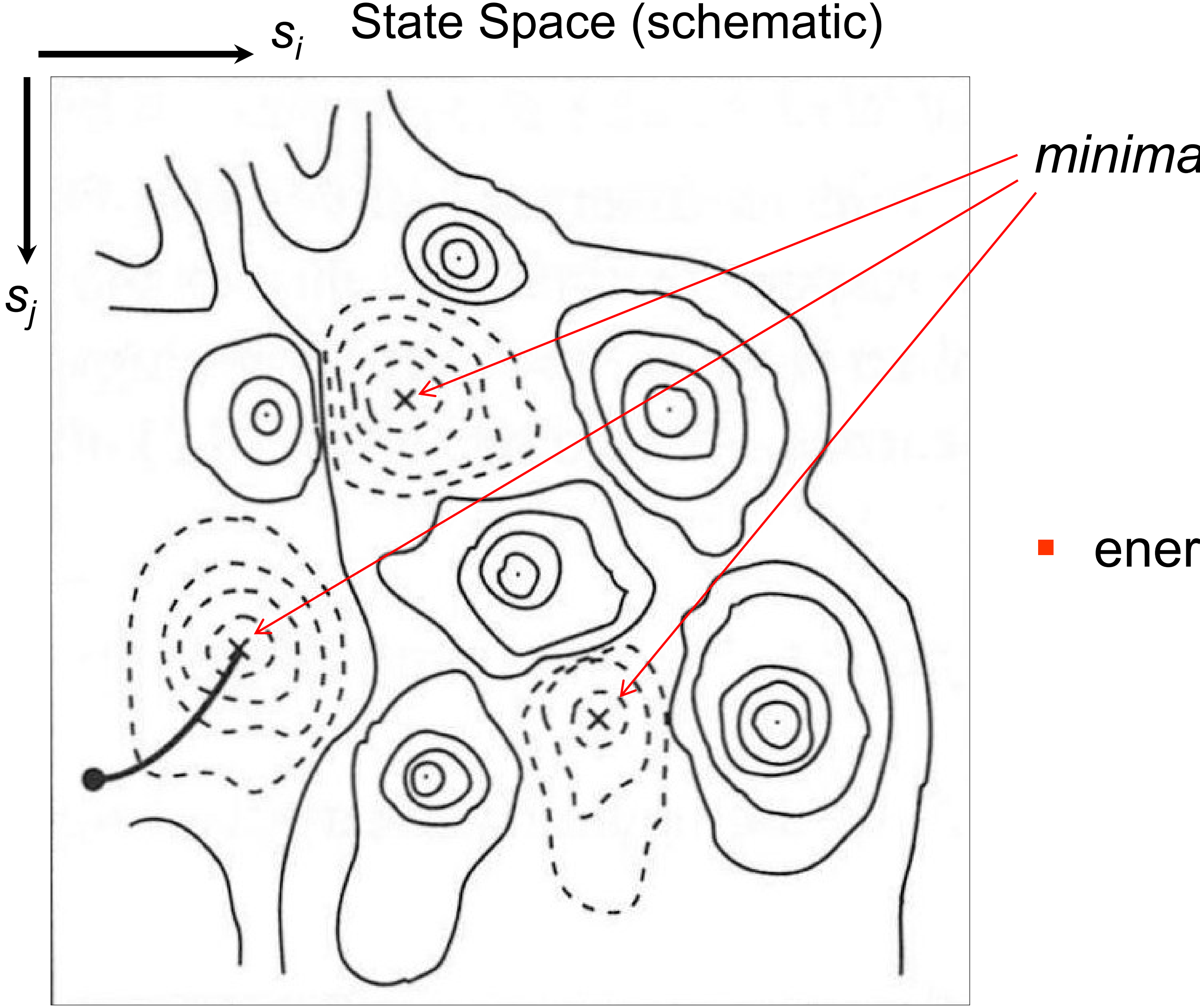
Energy Landscapes



Descent on energy surface while updating activities

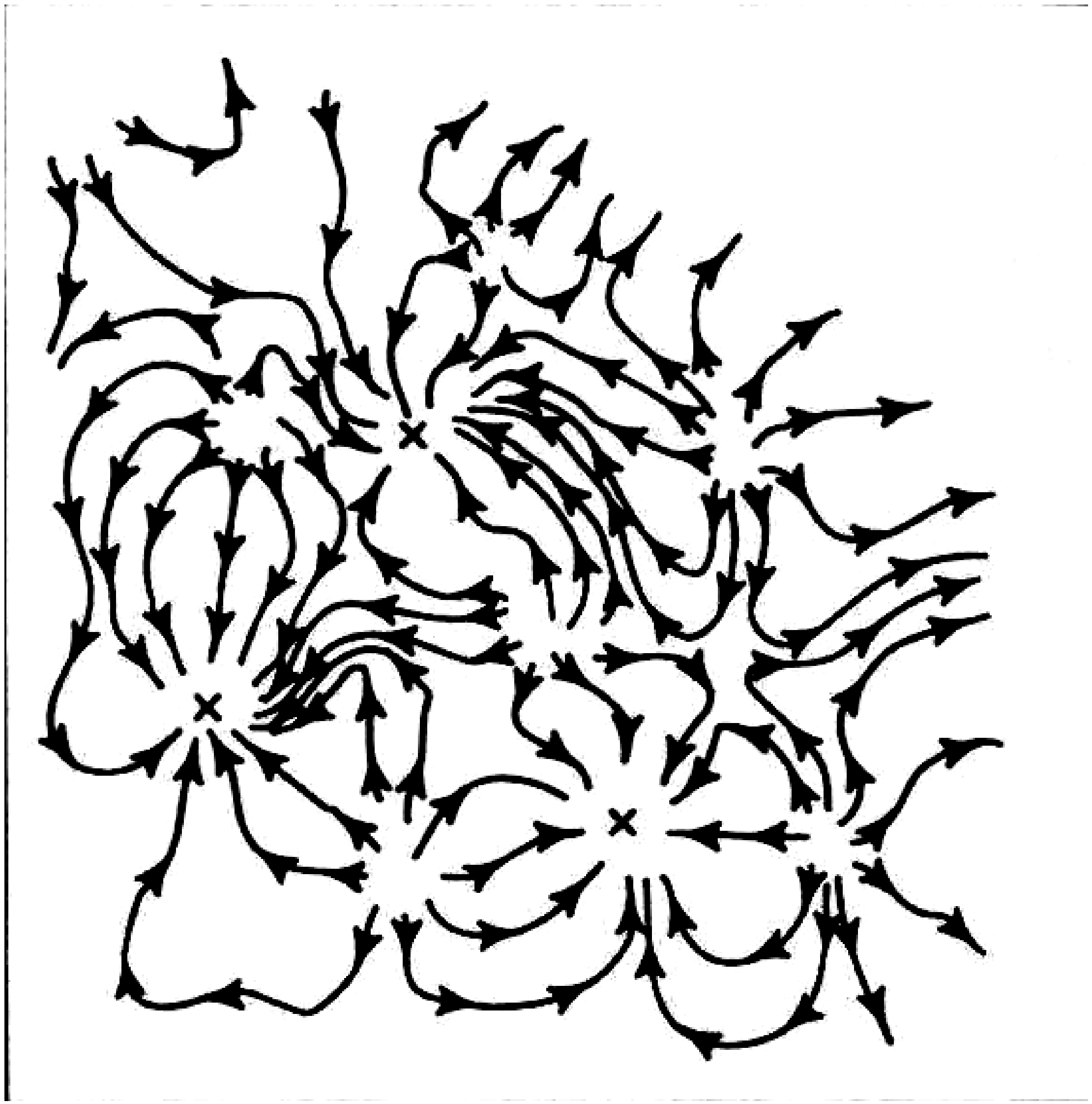
(fig. from Solé & Goodwin)

State Space (schematic)



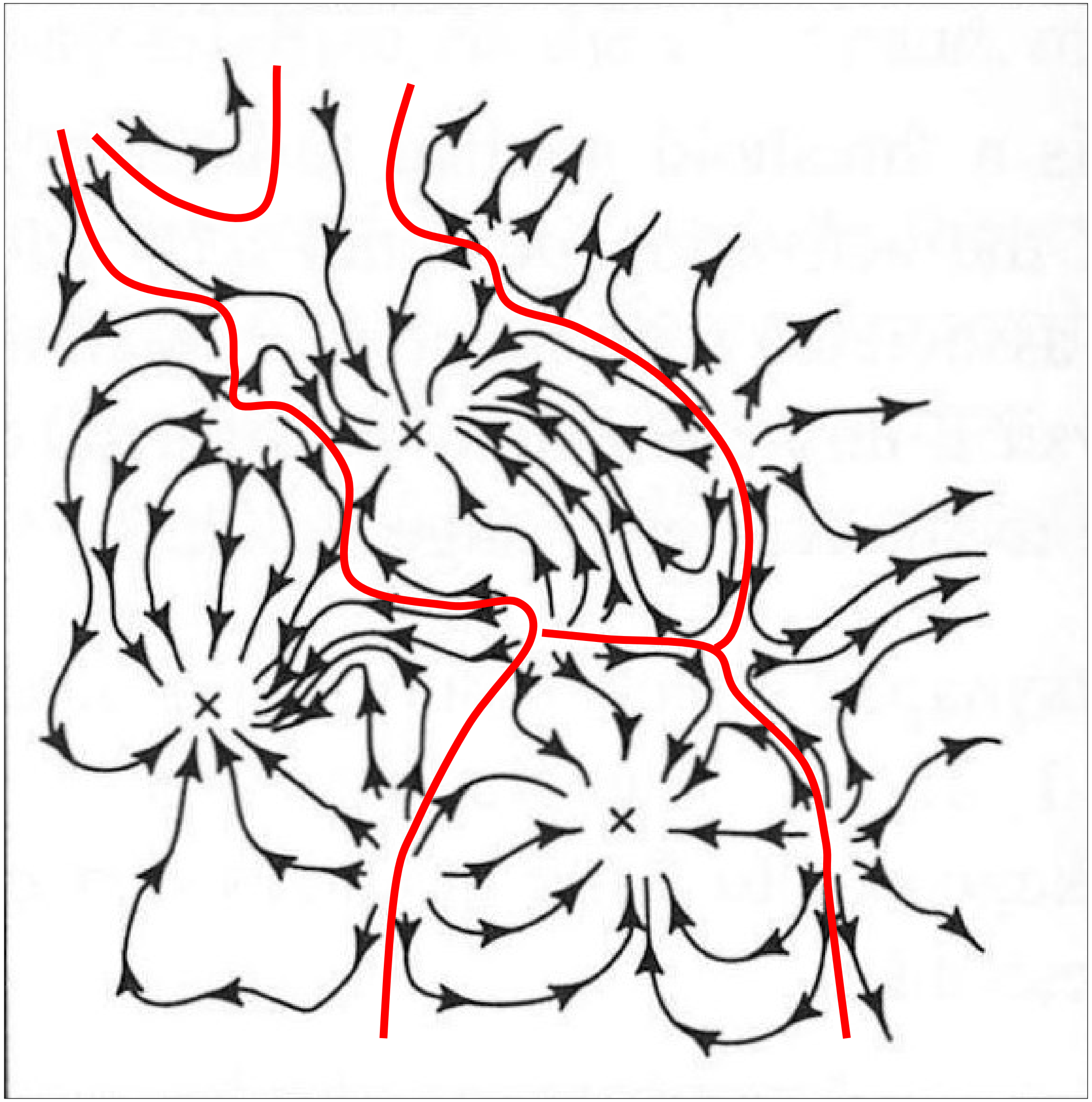
- energy surface

(fig. from Haykin *Neur. Netw.*)



- energy surface
- flow lines

(fig. from Haykin *Neur. Netw.*)



- energy surface
- flow lines
- attractor basins

(fig. from Haykin *Neur. Netw.*)

Extending Hebb Rule

Hebb

$$w_{ij} = \frac{1}{P} \sum_{p}^{P} s_i^p s_j^p$$

- Hebb rule led to small memory capacity, #patterns P ≤ ~0.13 #neurons N.
- Perceptron rule: network can store up to 2N patterns
- Idea: train only units i with patterns d where errors occur

init weights (e.g. random or using Hebb rule)

for all patterns p:

for all units i:

if $s_i^p \neq \text{sign} \sum_j w_{ij} s_j^p$ then:

$\Delta w_{ij} = s_i^p s_j^p \quad \forall j$

if unit *i* is wrong for pattern *p*

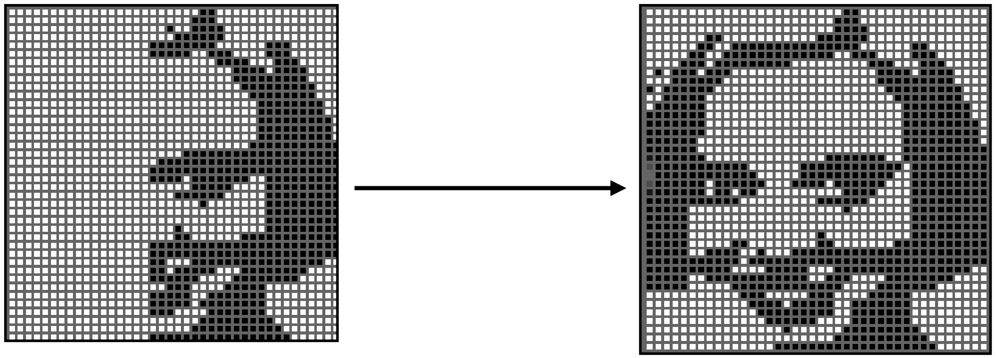
train incoming weights of unit *i* with pattern *p* using Hebb rule

- Note: $\rightarrow w_{ij} \neq w_{ij}$ in general, so no energy function

Hopfield Networks for Face Detection(?)

Associative memory

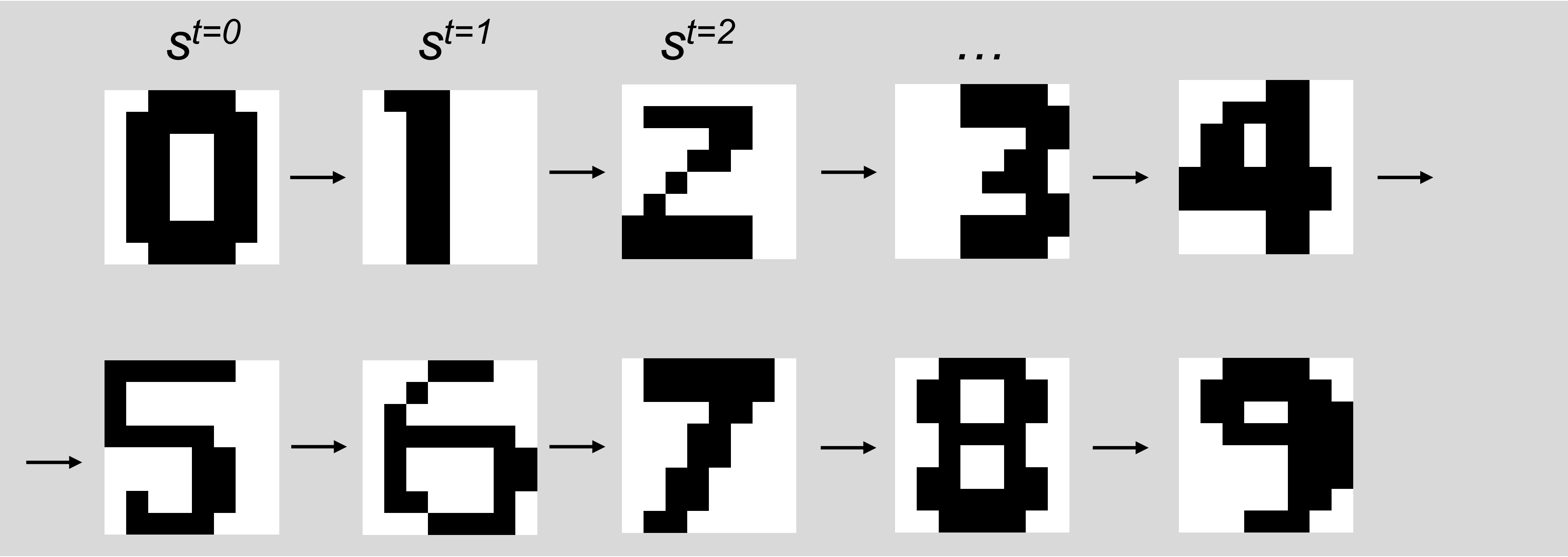
- Parts of a pattern can be used to recover the whole pattern



- Simple dynamical system
- Limited to binary variables

Sequence Generation (not Hopfield)

- Training patterns to be learnt in order:



pattern s^t will strengthen s^{t+1}

- Hebb-like rule: $w_{ij}^{seq} = \frac{1}{P} \sum_t^P s_i^{t+1} s_j^t$

$\longrightarrow$ **asymmetric weights**

Sequence Generation (not Hopfield)

- Actually learnt & generated sequence:

t=0 *t=1* *t=2* …

- A dynamic pattern emerges, *almost* as wanted …
- But: cannot extend network size/complexity!

Sequence Generation (not Hopfield)

- Asymmetric weight matrix can be written as a sum of a symmetric plus an unsymmetric component:

$$W^{sequence} = \underbrace{W^{sym}}_{w_{ij} = w_{ji}} + \underbrace{W^{asym}}_{w_{ij} \neq w_{ji}}$$

- The symmetric component stabilizes patterns, favouring static attractors (auto-association)
- The unsymmetic component favors the transition from one pattern to another pattern (hetero-association)

Summary of Hopfield Networks

- Associative network with dynamics
- Simple binary neuron model
- A single layer: input units = output units
  - Serves as a model of the hidden layer of a SRN, since (after initialization) units become activated by the network
- Symmetric weights ensure convergence of activations to a state with minimum energy
- Non-symmetric weight components may cause ever-lasting activation dynamics
  - but **not** a Hopfield network, according to definition

Summary of Simple Recurrent Networks

- *Efficiently applicable* to ***sequence prediction***, e.g.
  - Letters and words
  - Stock markets
  - ... *and much more!*
- *Efficiently applicable* to ***sequence classification***, e.g.
  - Handwriting and speech recognition
  - Gesture recognition
  - ... *and much more!*
- Method to problem solving ***using*** some ***context***

  ... that is still ***deterministic*** and can be analyzed