Fakulta matematiky, fyziky a informatiky

Univerzity Komenského v Bratislave

Neural Networks

Igor Farkaš

2016 (upravené)

Autor: Igor Farkaš
Názov: Neural networks
Podnázov: Učebný text k prednáške „Neurónové siete“ založenej hlavne na knihe S. Haykin: *Neural Networks and Learning Machines*, Prentice Hall, 2009.
Recenzent: prof. Ing. Vladimír Kvasnička, DrSc.
Vydavateľ: Knižničné a edičné centrum FMFI UK
Grafická úprava: Igor Farkaš
Rok vydania: 2011
Miesto vydania: Bratislava
Vydanie: prvé
Počet strán (číslované): 192
ISBN: 978-80-89186-93-8
Internetová adresa: http://www.fmph.uniba.sk/fileadmin/user_upload/editors/sluzby/kniznica/el_materialy/svet/neural-networks.pdf

Publikácia bola podporená grantom KEGA č. 3/7300/09.

Contents

1

Introduction

Introduction to connectionism

Connectionism – theory of information processing, inspired by biology (brains). It's based on Artificial Neural Networks (ANNs).

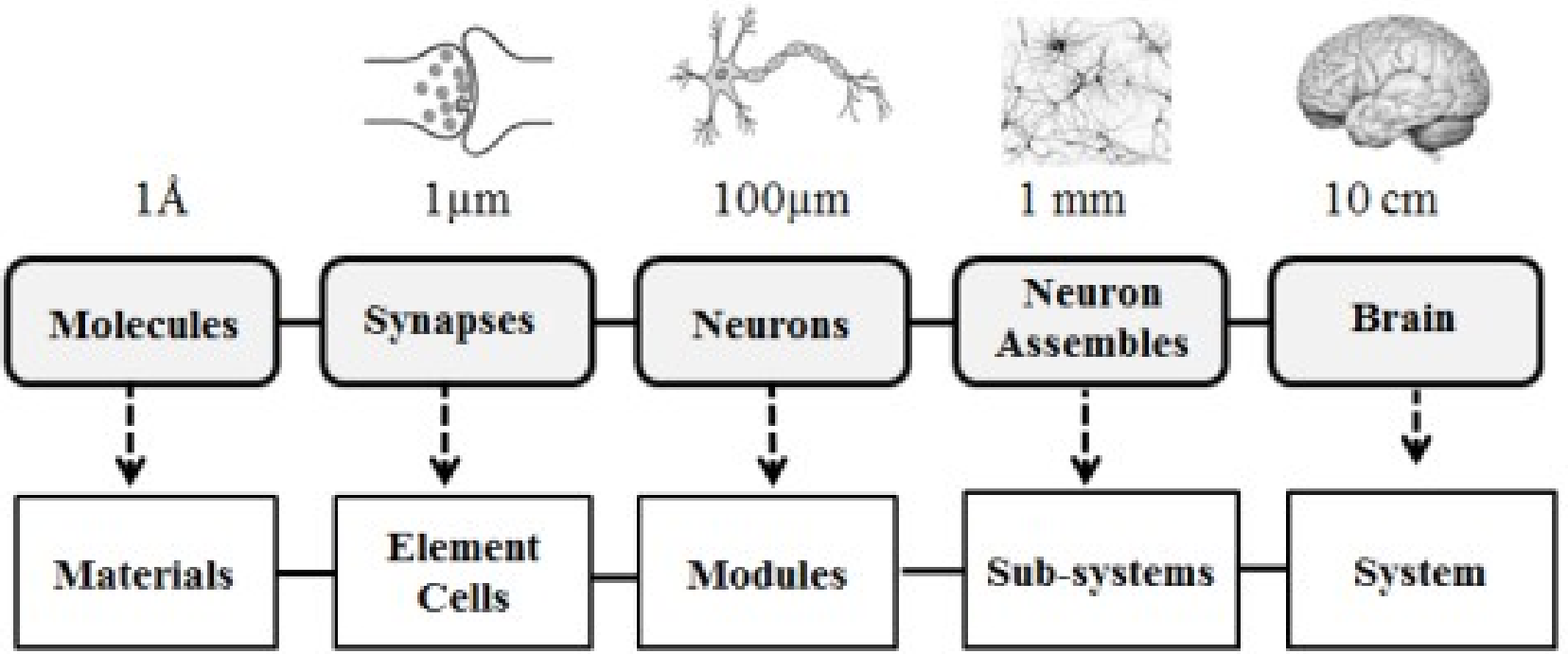
It has two goals:

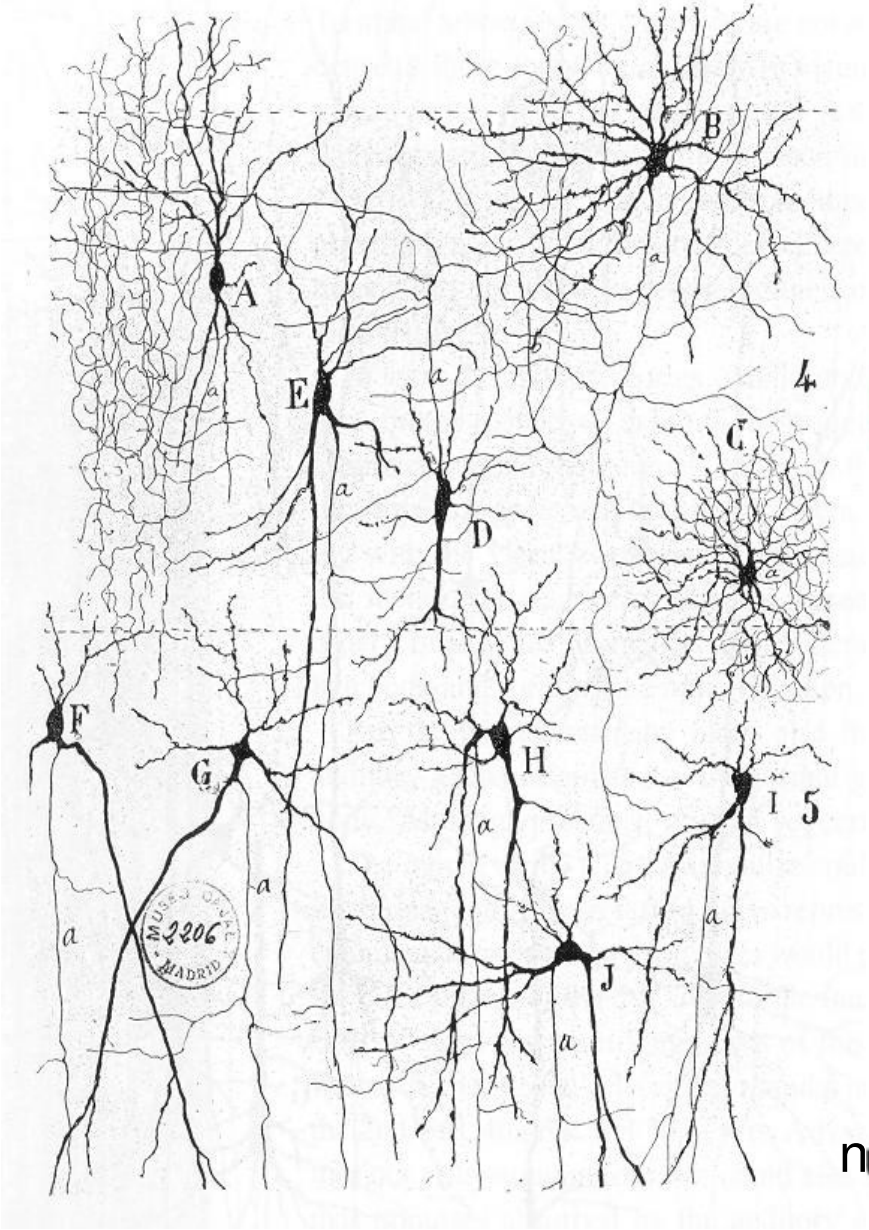
- theoretical foundations of cognitive science (modeling mental processes)
  - contrasting with symbolic approaches
  - features: parallelism, robustness, learning from experience,...
- applications in practical problems
  - tasks: pattern recognition, classification, associative memory, time series prediction, dimensionality reduction, data visualization, ...

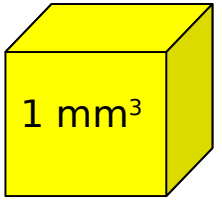
Few facts about human brain

- Brain = highly complex, non-linear and parallel information processing system (“computer”)
- composed of ~ 10^{11} neurons (basic information-processing elements), connected via ~ 10^{15} synapses
- Glial cells probably more important than previously thought
- on certain tasks, brain is much faster than supercomputers of today, even though neurons are very slow (~ ms)
- mostly prewired at birth, but very plastic throughout life
- importance of learning: involves 3 mechanisms
  - modification of existing synapses,
  - generation of new synapses,
  - neneration of new neural cells

Structural organization of levels in the brain

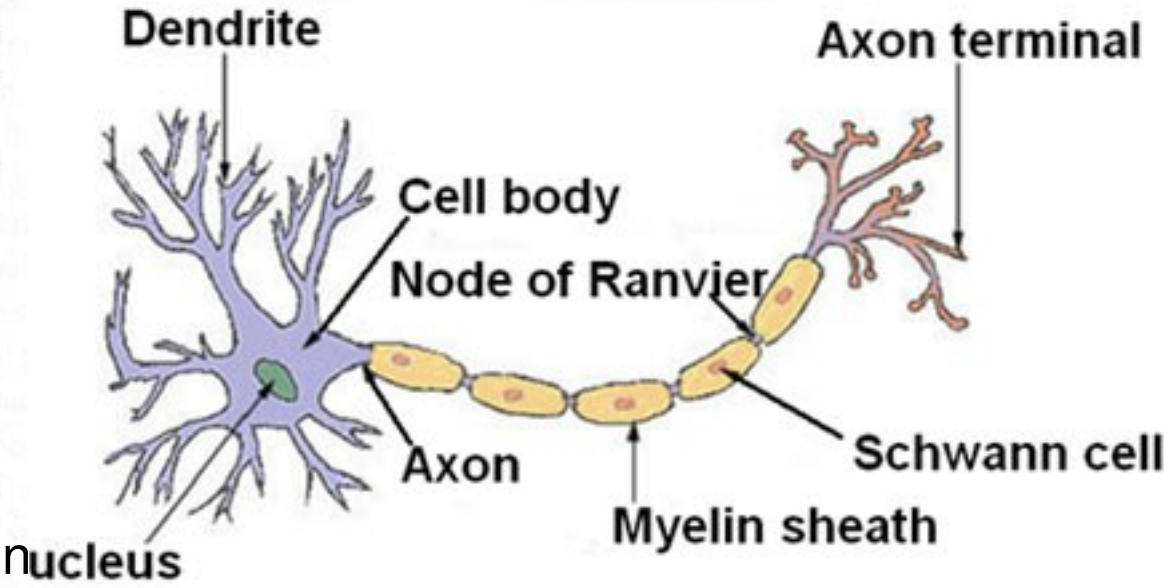






10^5 neurons
3 km axons

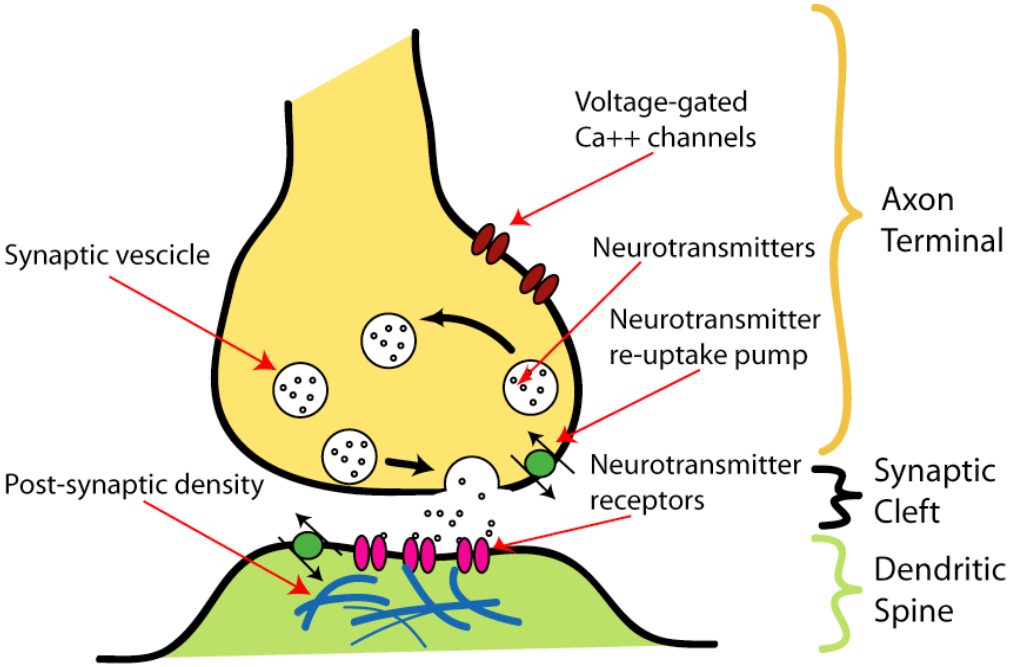
Structure of a Typical Neuron

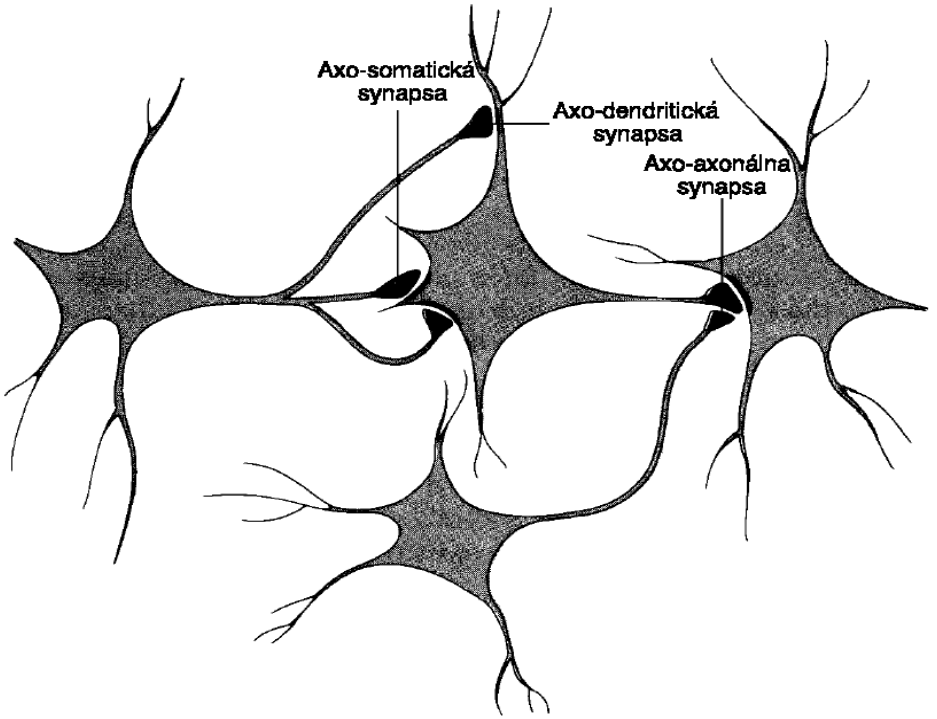


From R. y Cajal: Texture of the Nervous System of Man and the Vertebrates (illustrates the diversity of neuronal morphologies in the auditory cortex).

Synapse

- Synapse maintains the interaction between neurons.
- Presynaptic process releases a neurotransmitter, which diffuses across the synaptic junction between neurons and then acts on a postsynaptic process.
- Synapse mediates electrical-chemical-electrical signal conversion.
- Effect on a postsynaptic neuron can be either excitatory or inhibitory.





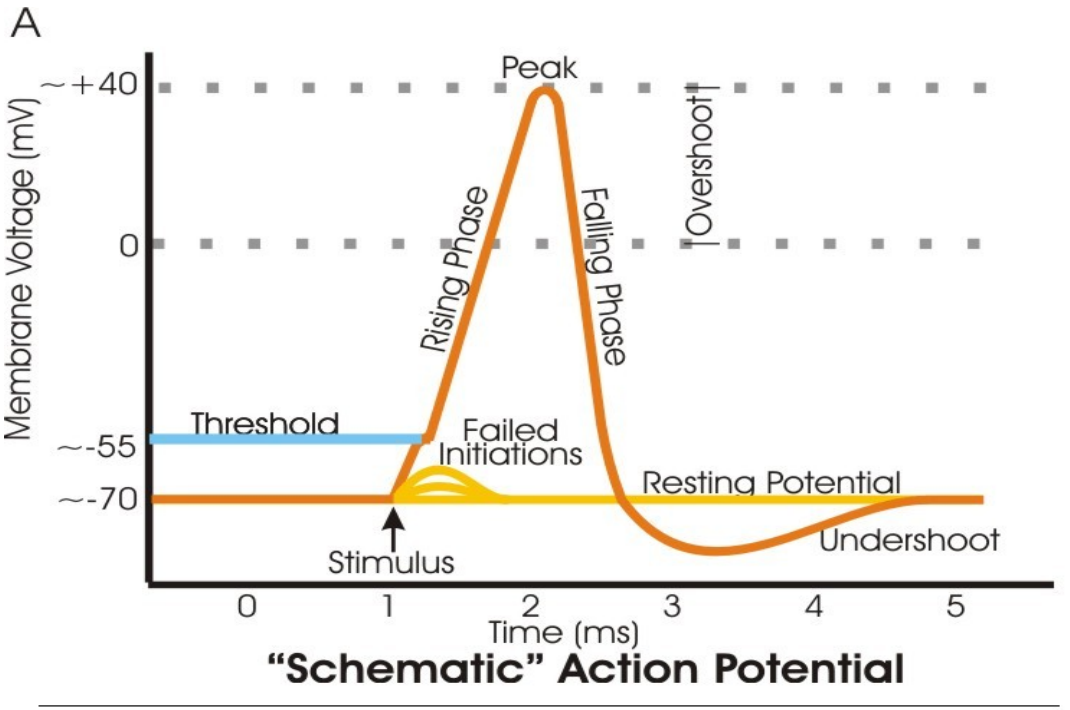
Action potential

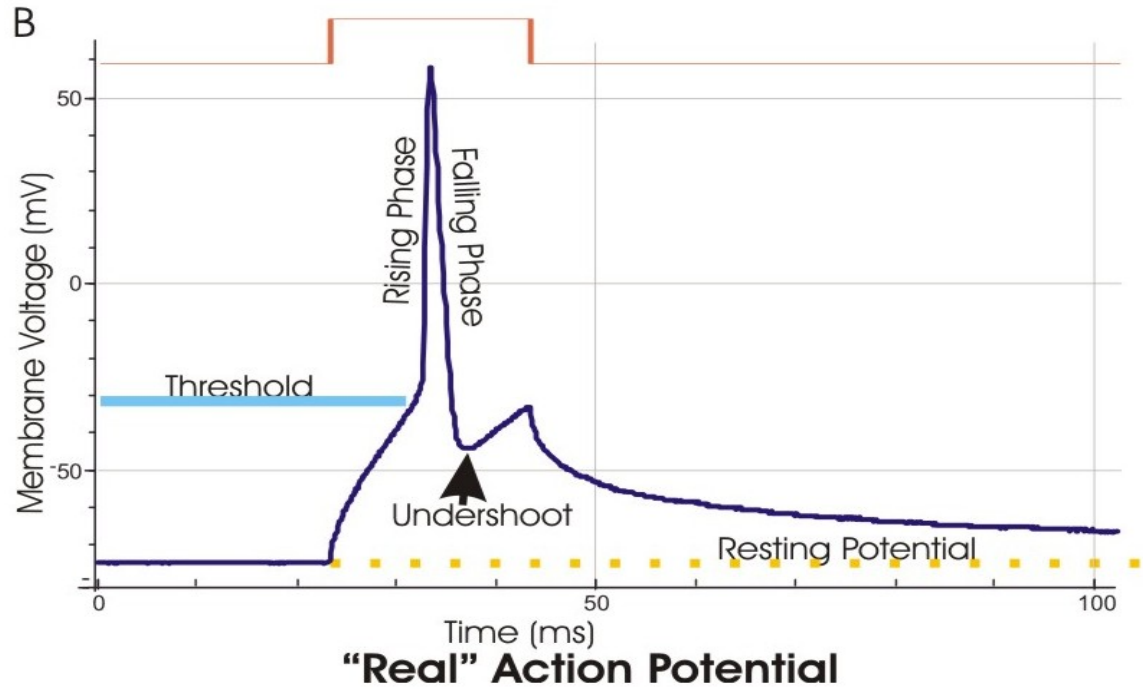
If a neuron is made to “fire”, generated action potential (AP) traverses along the axon, uninhibited.

Generation of AP (neuron “firing”) requires that membrane potential exceed the excitation threshold.

This usually requires one of the two conditions:

1. temporal summation - arrival of multiple impulses in time at the same synapse, or
2. spatial summation - impulses arrive simultaneously at sufficient number of different synapses.



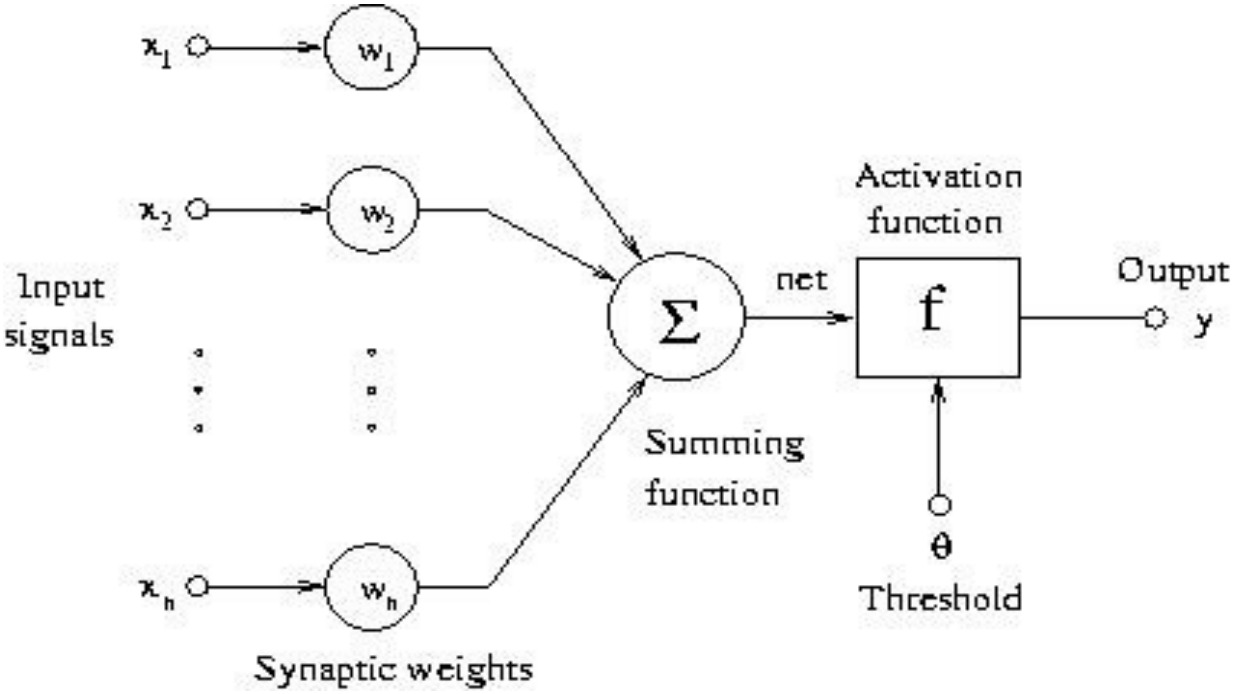


Typical artificial neuron model

1. receives signals from other neurons (or sensors)
2. processes (integrates) incoming signals
3. sends the processed signal to other neurons (or muscles)

Deterministic model

$$y = f\left(\sum_i w_i x_i - \theta\right)$$



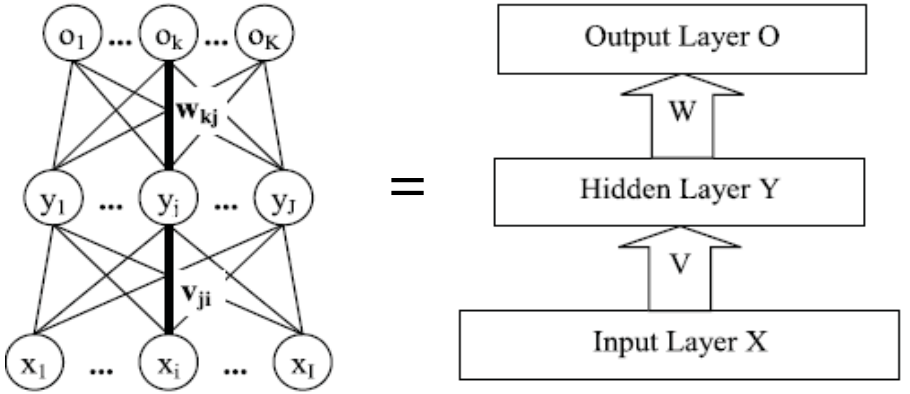
Stochastic model $P(y=1) = 1/(1+\exp(-\sum_i w_i x_i + \theta))$

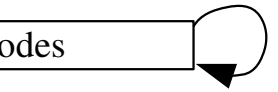
Useful features of artificial neural networks

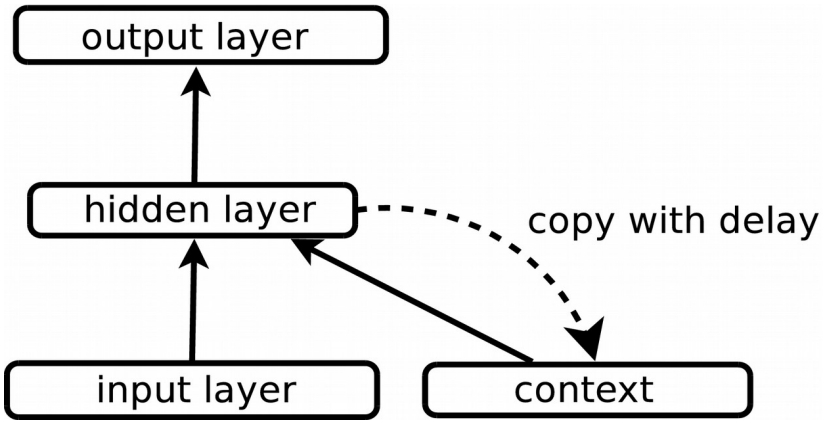
- Nonlinearity (of processing units)
- Input-output mapping (nonparametric statistical inference)
- Adaptivity (parameter tuning)
- Evidential response (degree of 'confidence')
- Contextual information (← thank to connectivity)
- Fault tolerance (graceful degradation)
- VLSI implementability
- Neurobiological analogy
- Uniformity of analysis and design
- Importance of environment (for design)

Neural network architectures

- Single-layer feedforward NNs
- Multi-layer feedforward NNs
  - Have hidden layers
  - Partial connectivity
- Recurrent networks
  - Associative memories
  - Input-output mapping networks





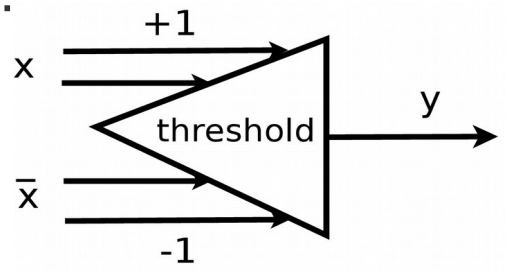


Classical connectionism

- Aristoteles (400 BC) – introduced concepts of memory, and connectionism
- Spencer (1855) – separated psychology from philosophy, postulated that “neural states affect psychological states”, knowledge is in connections.
- James (1890) – model of associative memory, “law of neural habit”
- Thorndike (1932) – distinguishes sub-symbolic view on neural associations, formulated two laws of adaptation: “the law of effect” and “the law of exercise”.
- McCulloch & Pitts (1943) – neural networks with threshold units
- Minsky (1967) extended their results to comprehensible form, and put them in context of automata theory and the theory of computation

First neural network

- Birth of computer era
- How could information be represented in a nervous system?
- McCullogh & Pitts (1943) – neural networks with threshold units
  - Logical neuron:



$y = 1$ if $\text{sum}(x) - \text{sum}(\text{not } x) \geq \text{thr}$

    - can simulate any linearly separable Boolean function:
    - Fixed weights, positive and negated inputs
  - Any Boolean function $\{0,1\}^n \rightarrow \{0,1\}$ can be simulated by a 2-layer NN with logical units.
- Birth of neural networks and artificial intelligence disciplines

Brief history of connectionism

- classical connectionism (until 1940s)
  - within philosophy, psychology
- old connectionism (1950s-1970s) – birth of computer era
  - beginning of theory of ANN, linked to cognitive science revolution
- new connectionism (from 1986)
  - parallel distributed processing ⇡ subsymbolic processing
  - multi-layer NN models (incl. recurrent)
- even newer connectionism (late 1990s)
  - multilayer generative models (probabilistic approach)
- 2nd renaissance of ANNs (Schmidhuber, 2000-)
  - deep networks, convolution, recurrent nets (reservoir computing)

Knowledge representation in AI

- Knowledge refers to stored information or models used by a person or machine to interpret, predict and appropriately respond to the outside world. (Fischler & Firschein, 1987)
- Primary characteristics of KR:
  - What information is made explicit
  - How the information is physically encoded for use
- KR is goal oriented: In “intelligent” machines, a good solution depends on a good KR.
- ANNs are a special class of intelligent machines.
- Knowledge in ANN is distributed in free parameters (synaptic weights).

Rules for knowledge representation in ANN

1. Similar inputs from similar classes should usually produce similar representations inside the NN, and should hence be classified as belonging to the same category (measures of similarity: e.g. Euclidean dist., Mahalanobis dist).
2. Items to be categorized as separate classes should be given widely different representations in the NN.
3. If a particular feature is important, then there should be a large number of neurons involved in the representation of that item in the NN.
4. Prior information and invariance (e.g. using constraints) should be built into the design of an ANN.

Artificial Intelligence and ANNs

AI system must be able to:

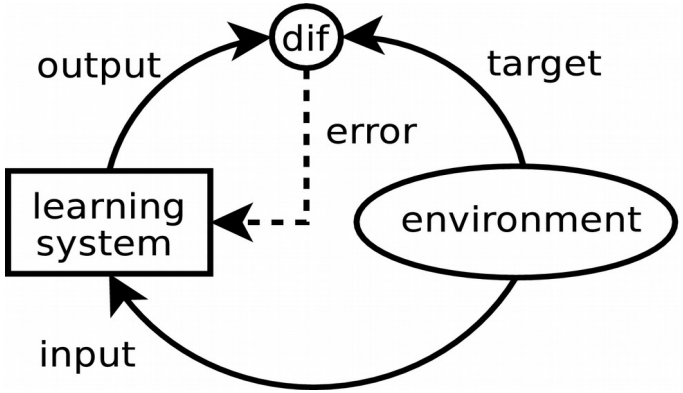
1. store knowledge,
2. apply the knowledge
3. acquire new knowledge through experience.

Key components of AI system:

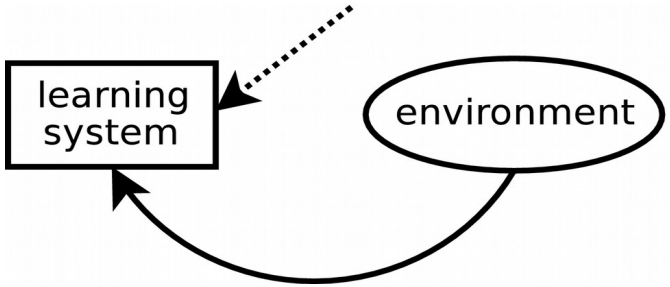
- representation – symbolic (declarative, procedural)
  - in NN: mostly subsymbolic, no separation b/w processor and memory
- reasoning – ability to solve problems (searching)
  - in NN: straightforward (propagation of activation)
- learning – inductive, deductive
  - in NN usually: training – inductive, testing - deductive

Learning paradigms in NN

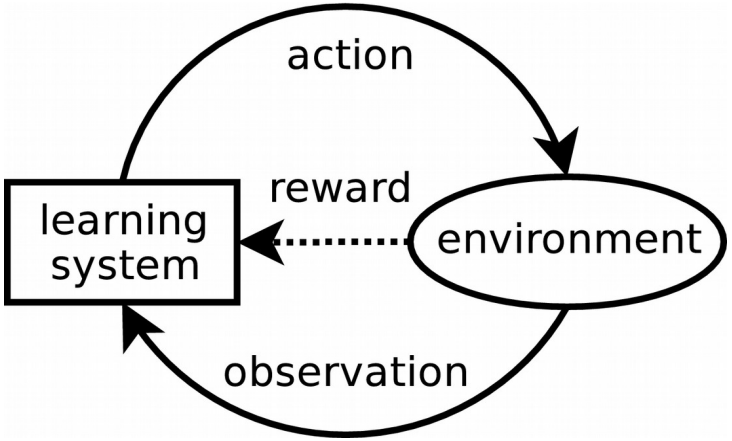
supervised (with teacher)



unsupervised (self-organized)



reinforcement learning (partial feedback)



Learning rule types in ANN

- Error-correction – supervised
  - closed-loop feedback system
- Memory-based (e.g. k-nearest neighbors classifier)
  - knowledge stored in examples
- Hebbian – unsupervised
  - correlational synaptic potentiation/depression
- Competitive
  - competition for inputs, feature detectors
- Boltzmann – stochastic
  - inspired by from stat. mechanics, good for high-dim. problems

Learning tasks

- Pattern association (auto-, hetero-)
- Pattern classification (within pattern recognition)
- Feature extraction (within PR or independently)
- Data compression
- Function approximation
- Control
- Filtering
- Prediction
- Signal generation (with recurrent networks)
- ...

ANNs as models of cognition

- Connectionism as a paradigm in cognitive science
  - together with symbolic, dynamic and probabilistic paradigms
- Level of explanation:
  - subsymbolic (less transparent than symbolic)
- Information processing:
  - parallel, distributed
- Representational power: (compositionality)
  - representing structured data is a challenge for ANN
- Requires looser definition of computation
  - ANN can work in continuous time/space
- Major strength: learning algorithms

Significant recent developments

- Deep learning in feedforward multilayer networks – very successful in various domains (image recognition, speech processing, language modeling)
- Reservoir computing – efficient approach to processing spatio-temporal signals
- NNs as building blocks of various cognitive architectures
- Computational neuroscience (spiking neural networks) – quest for neural code
  - Blue Brain Project (since 2005)
  - (huge EU) Human Brain Project (since 2013)

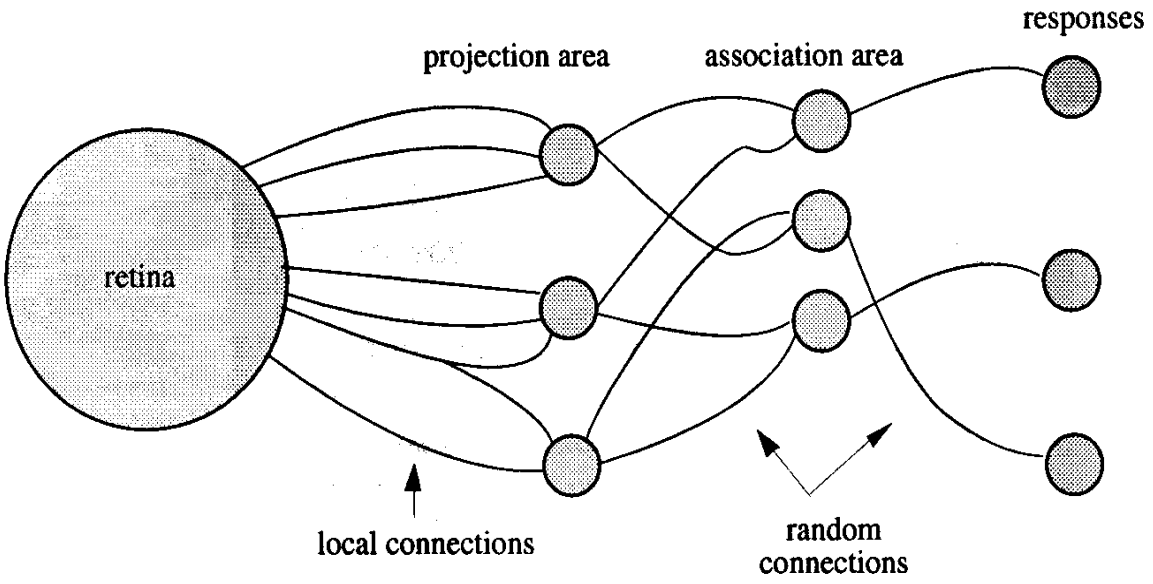
2

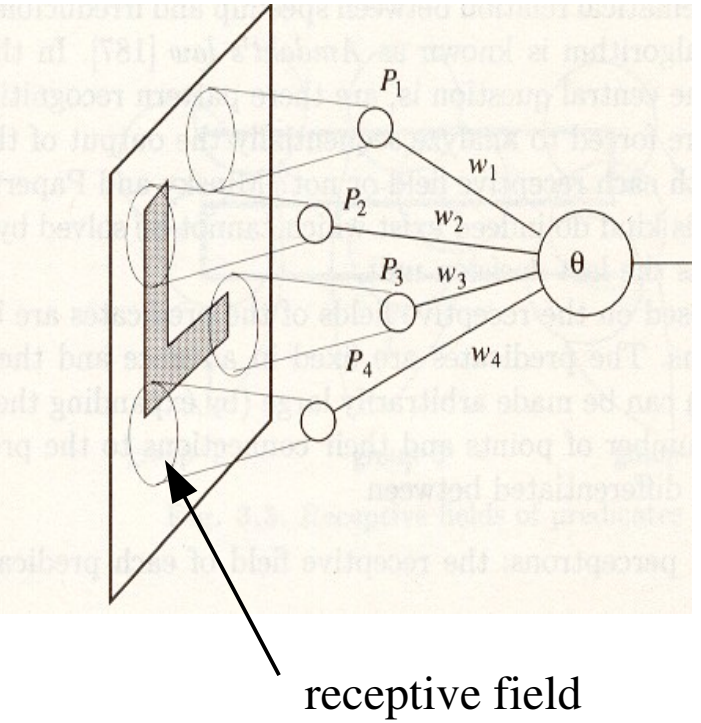
Simple perceptrons

Classical perceptron

In 1958, F. Rosenblatt (American psychologist) proposed perceptron, a more general computational model (than McCulloch-Pitts units) with free parameters, stochastic connectivity and threshold elements.

In 1950, Hubel & Wiesel “decoded” the structure of retina and receptive fields.





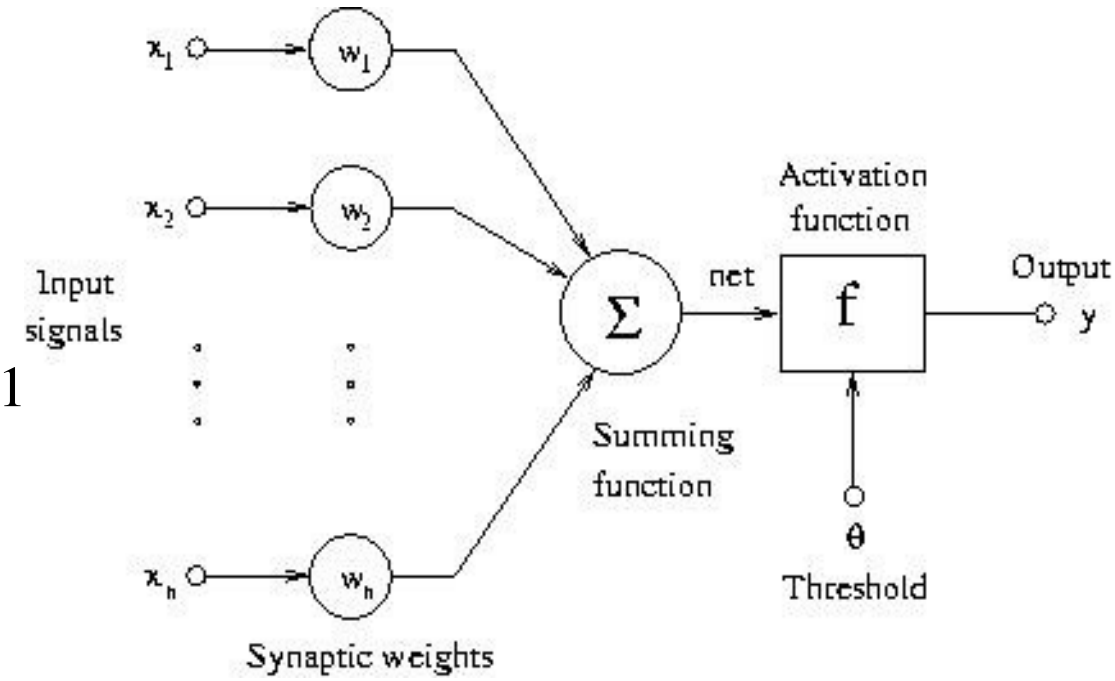
Discrete perceptron

(Rosenblatt, 1962)

- Inputs ***x*** , weights ***w***, output *y*
- Activation:

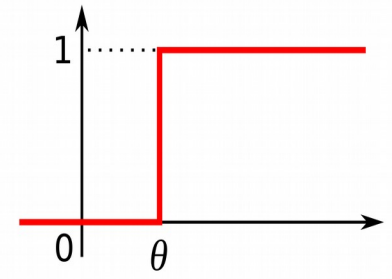
$$y = f(\sum_{j=1}^{n} w_j x_j - \theta)$$

$$y = f\left(\sum_{j=1}^{n+1} w_j x_j\right) \quad x_{n+1} = -1$$



- *f* = threshold function: unipolar {0,1} or bipolar {-1,+1}
- Supervised learning – uses teacher signal *d*
- Learning rule:

$$w_j(t+1) = w_j(t) + \alpha \ (d - y)\, x_j$$



F. Rosenblatt (1962). *Principles of Neurodynamics*, Spartan, New York, NY.

Summary of perceptron algorithm

Given: training data: input-target $\{\boldsymbol{x}, d\}$ pairs, unipolar perceptron
Initialization: randomize weights, set learning rate, $E \leftarrow 0$.

Training:

1. choose input $\boldsymbol{x}$, compute output y
2. evaluate error function $e(t) = ½ (d - y)^2$, $E \leftarrow E + e(t)$
3. adjust weights using delta rule (if $e(t) > 0$)
4. if all patterns used, then goto 5, else go to 1
5. if $E = 0$ (all patterns in the set classified correctly), then end

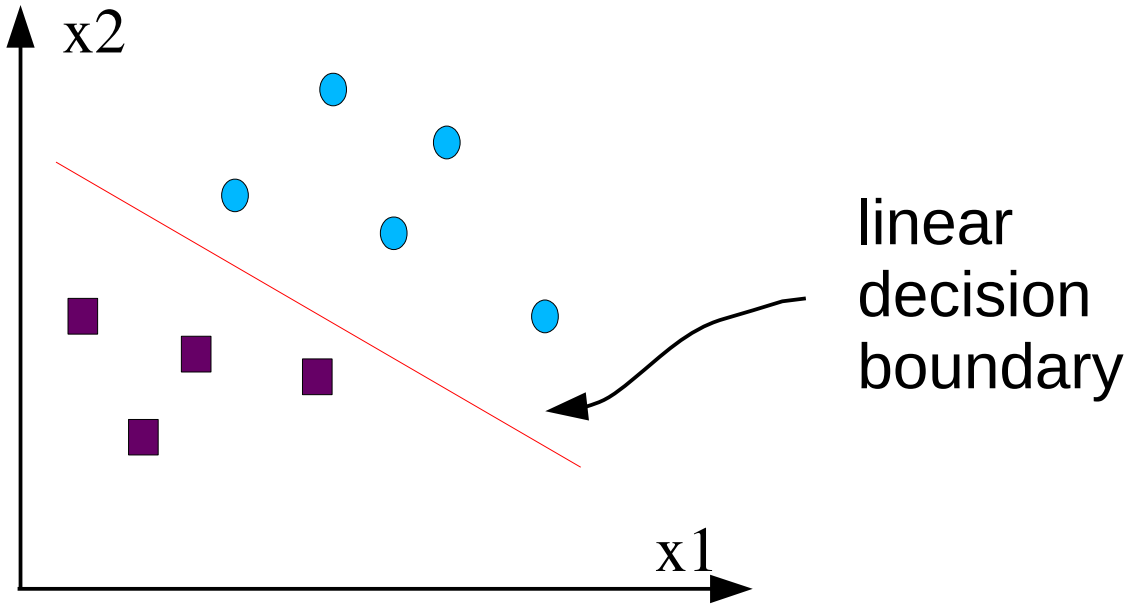
   else reorder inputs, $E \leftarrow 0$, go to 1

Perceptron classification capacity

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = \theta$$

linear separability of two classes

2D example



Fixed-increment convergence theorem (Rosenblatt, 1962): *"Let the classes A and B are finite and linearly separable, then perceptron learning algorithm converges (updates its weight vector) in a finite number of steps."*

Finding a solution

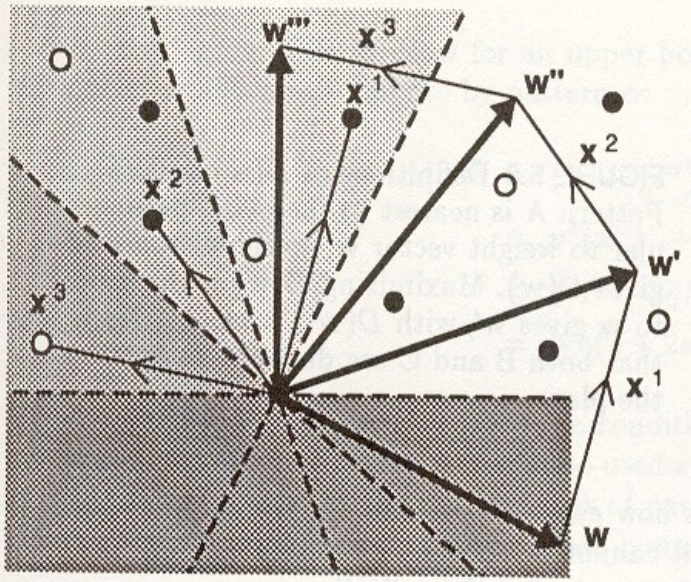
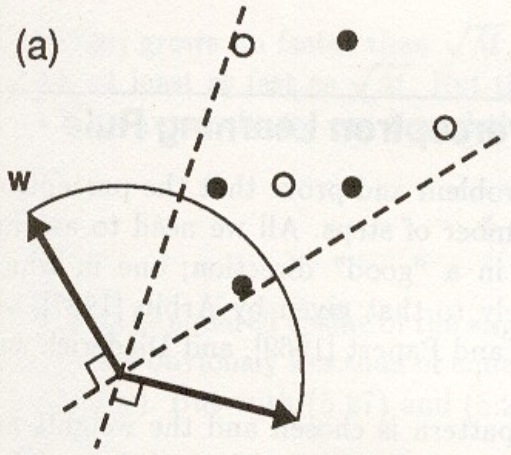
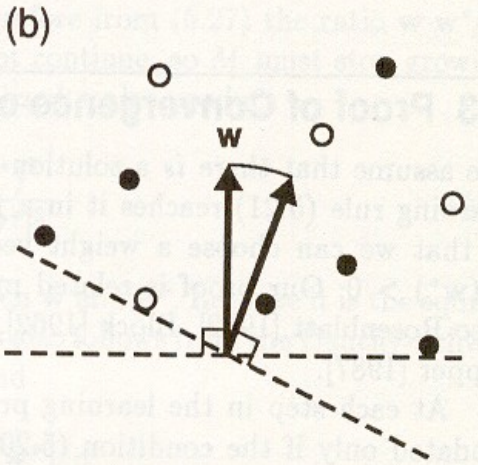


FIGURE 5.6 How the weight vector evolves during training for $\kappa = 0$, $\eta = 1$. Successive values of the weight vector are shown by $\mathbf{w}$, $\mathbf{w}'$, $\mathbf{w}''$, and $\mathbf{w}'''$. The darker and darker shading shows the "bad" region where $\mathbf{w} \cdot \mathbf{x} < 0$ for the successive $\mathbf{w}$ vectors. Each $\mathbf{w}$ is found from the previous one (e.g., $\mathbf{w}'$ from $\mathbf{w}$) by adding an $\mathbf{x}^{\mu}$ from the current bad region.

easy





more difficult

Perceptron convergence proof

- We prove convergence for $\alpha=1$ [magnitude of α only affects scaling]
- Initial cond.: $\boldsymbol{w}(0) = \mathbf{0}$. Assume $\boldsymbol{w}^{\mathrm{T}}(n)\,\boldsymbol{x}(n) < 0$ for $n=1,2,\ldots$ but should be > 0 (#)
- Iteratively apply learning rule: $\boldsymbol{w}(n+1) = \boldsymbol{x}(1) + \boldsymbol{x}(2) + \ldots + \boldsymbol{x}(n)$
- Since C1 and C2 are lin. separ., so $\exists$ solution $\boldsymbol{w}_0$: $\boldsymbol{w}_0^{\mathrm{T}}\boldsymbol{x}(i) > 0$ for $i = 1,\ldots,n$
- Let's define $a = \min_{\mathbf{x}(n)\in C1}\{\boldsymbol{w}_0^{\mathrm{T}}\boldsymbol{x}(n)\}$. So, $\boldsymbol{w}_0^{\mathrm{T}}\boldsymbol{w}(n+1) = \boldsymbol{w}_0^{\mathrm{T}}\boldsymbol{x}(1) + \ldots + \boldsymbol{w}_0^{\mathrm{T}}\boldsymbol{x}(n)$ => $\boldsymbol{w}_0^{\mathrm{T}}\boldsymbol{w}(n+1) \geq n.a$
- Cauchy-Schwarz inequality: $\|\boldsymbol{w}_0\|^2.\|\boldsymbol{w}(n+1)\|^2 \geq \|\boldsymbol{w}_0^{\mathrm{T}}.\boldsymbol{w}(n+1)\|^2$
- Hence, $\|\boldsymbol{w}_0\|^2.\|\boldsymbol{w}(n+1)\|^2 \geq n^2.a^2$ => $\|\boldsymbol{w}(n+1)\|^2 \geq n^2.a^2 / \|\boldsymbol{w}_0\|^2$ (*)
- Now, $\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \boldsymbol{x}(k)$, for $k = 1,2,\ldots,n$, $\boldsymbol{x}(k) \in C1$
- $\|\boldsymbol{w}(k+1)\|^2 \leq \|\boldsymbol{w}(k)\|^2 + \|\boldsymbol{x}(k)\|^2$ (b/c #) i.e. $\|\boldsymbol{w}(k+1)\|^2 - \|\boldsymbol{w}(k)\|^2 \leq \|\boldsymbol{x}(k)\|^2$ (&)
- Adding (&), $k \equiv n$, with $\boldsymbol{w}(0) = \mathbf{0}$, we get: $\|\boldsymbol{w}(n+1)\|^2 \leq \sum_{k=1}^{n}\|\boldsymbol{x}(k)\|^2 \leq n.b$ with $b = \max_{\mathbf{x}(k)\in C1}\{\|\boldsymbol{x}(k)\|^2\}$, which is in conflict with (*). So $n_{\max} = b\|\boldsymbol{w}_0\|^2 / a^2$

Continuous perceptron

- Nonlinear unit with sigmoid activ. function: $y = f(net) = 1 / (1+e^{-net})$

  - has nice properties (boundedness, monotonicity, differentiability)

- Quadratic error function: $E(\boldsymbol{w}) = \frac{1}{2} \sum_p (d^{(p)} - y^{(p)})^2$ [p ~ patterns]

- (unconstrained) minimization of the error function: necessary conditions $e(\mathbf{w}^*) \leq e(\mathbf{w})$ and $\nabla e(\boldsymbol{w}^*) = 0$, gradient operator $\nabla = [\partial/\partial w_1, \partial/\partial w_2, \ldots]^T$

- (stochastic, online) gradient descent learning:

$$w_j(t+1) = w_j(t) + \alpha \ (d^{(p)} - y^{(p)}) f' x_j = w_j(t) + \alpha \delta^{(p)} x_j$$

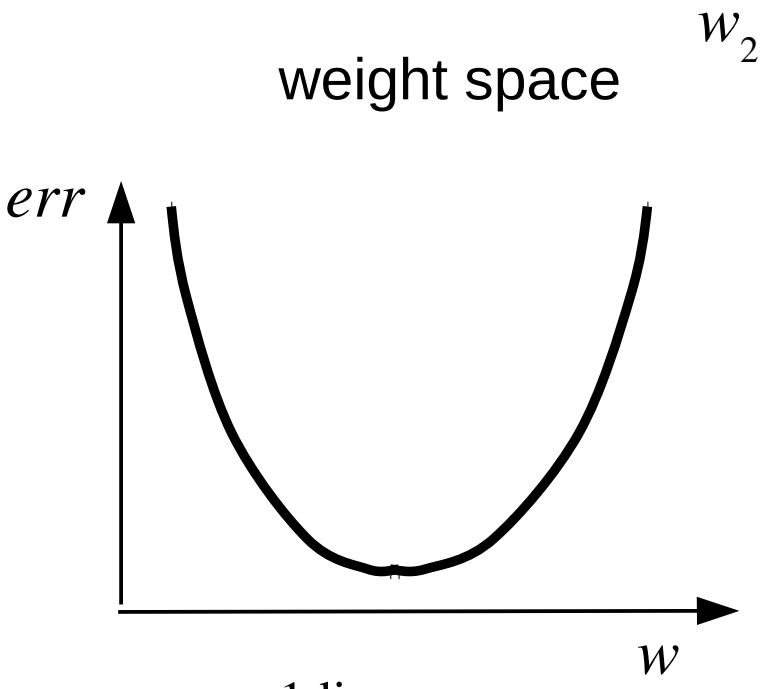
- (alternative) batch learning:

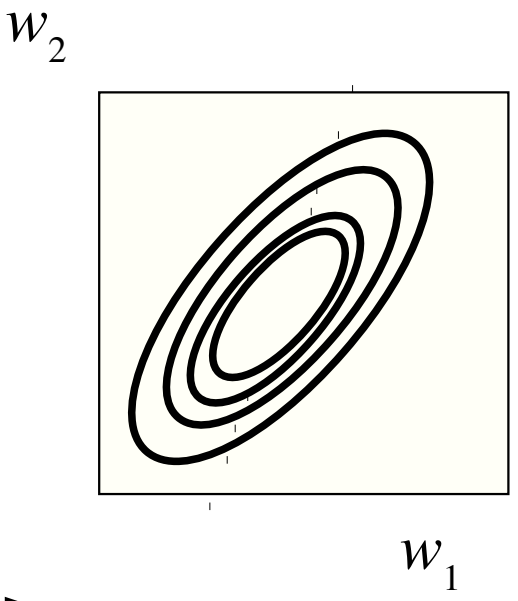
$$w_j(t+1) = w_j(t) + \alpha \sum_p \delta^{(p)} x_j^{(p)}$$

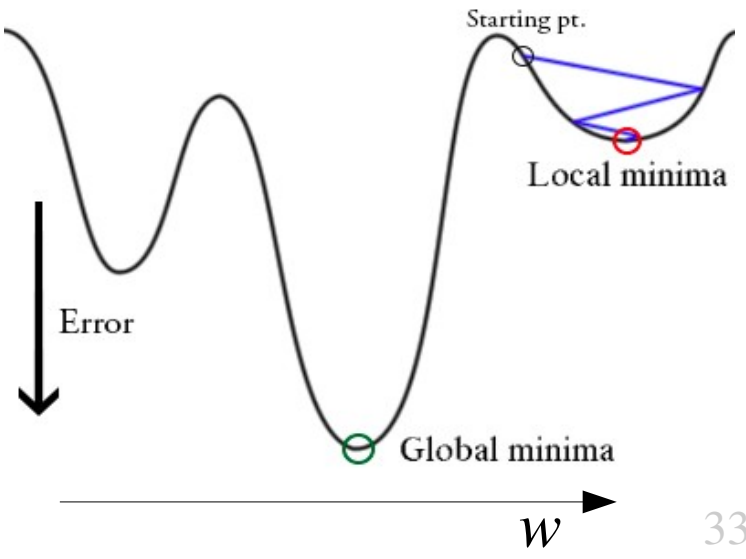
Perceptron as a binary classifier

Error surface

- The output error $e = f(w_1, w_2, \ldots, w_{|w|})$
- For a linear neuron with 2 inputs it is a quadratic bowl; vertical cross-sections are parabolas; horizontal cross-sections are ellipses.



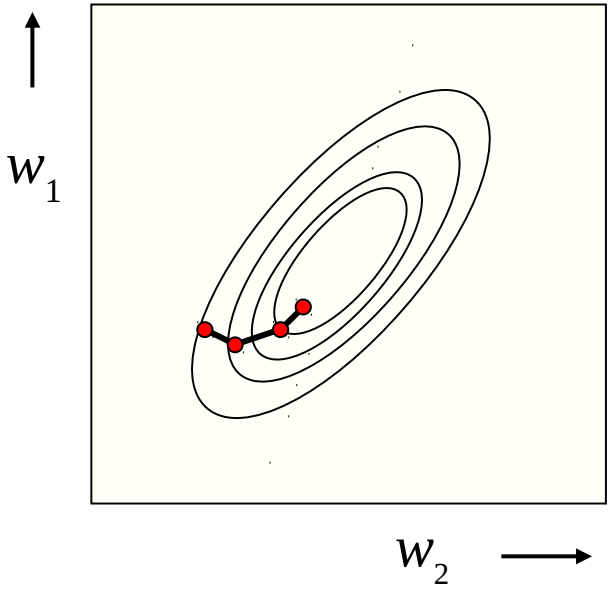


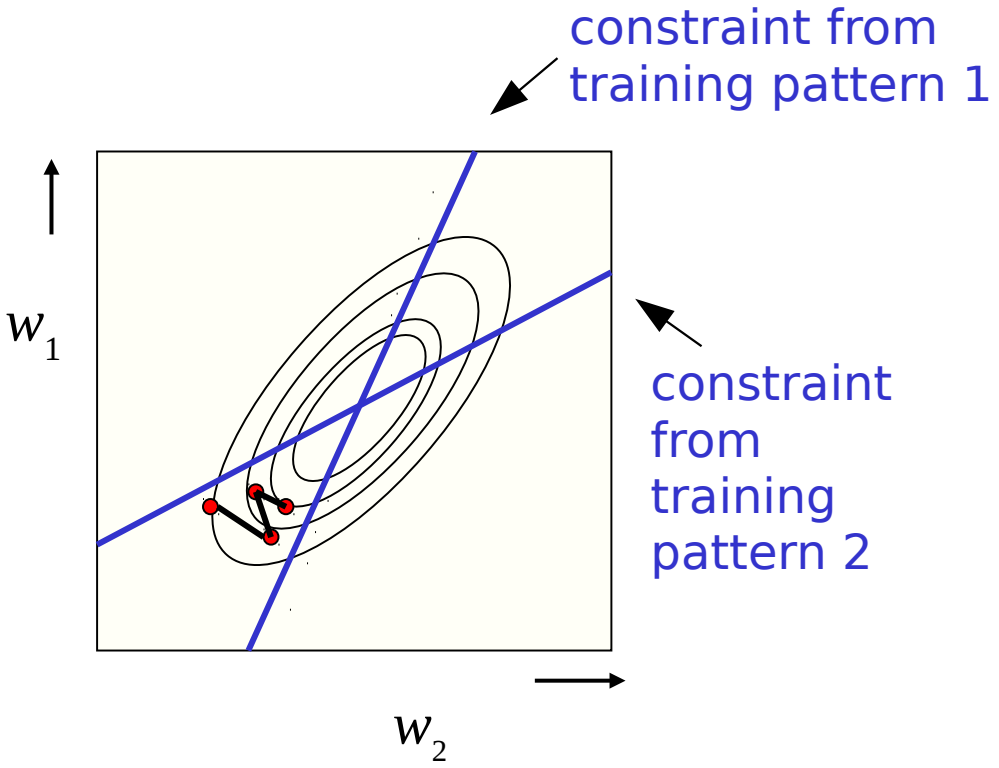


Online versus batch learning

Batch learning - does steepest descent on the error surface

Online learning - zig-zags around the direction of steepest descent





Effect of learning rate

$$E = x^2 + 20y^2$$

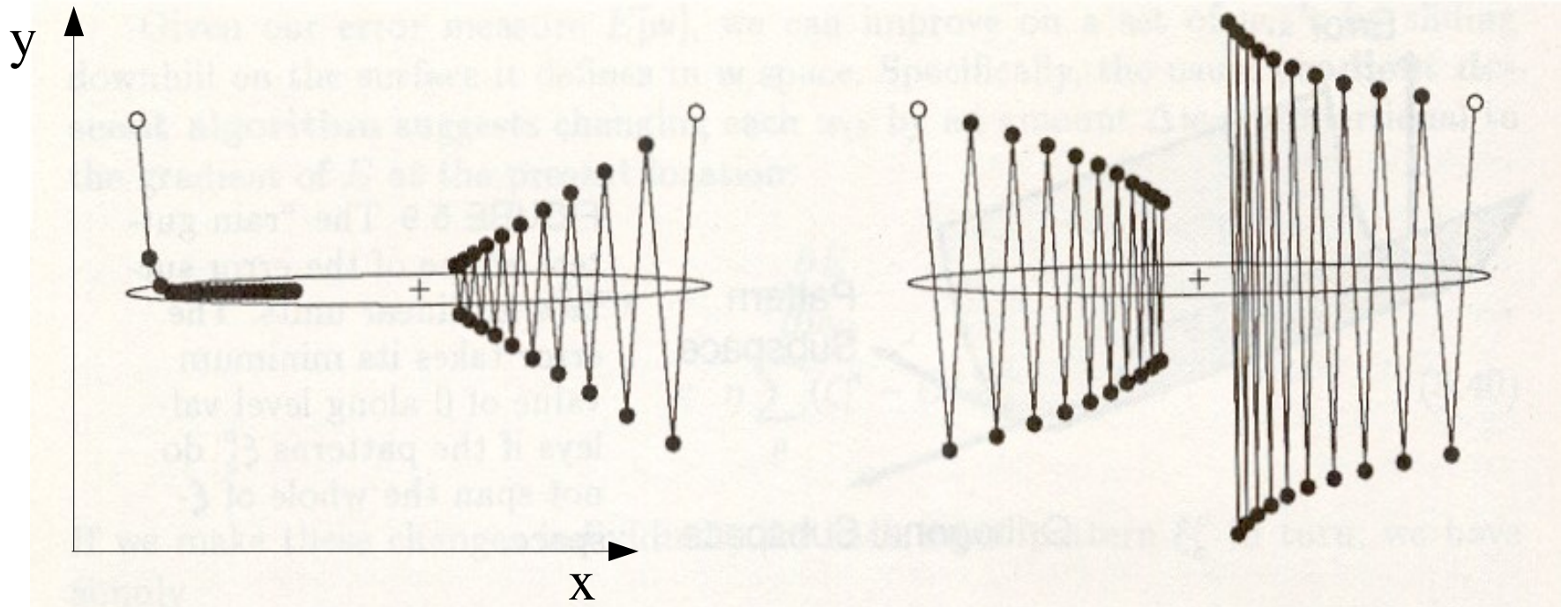


FIGURE 5.10 Gradient descent on a simple quadratic surface (the left and right parts are copies of the same surface). Four trajectories are shown, each for 20 steps from the open circle. The minimum is at the + and the ellipse shows a constant error contour. The only significant difference between the trajectories is the value of η, which was 0.02, 0.0476, 0.049, and 0.0505 from left to right.

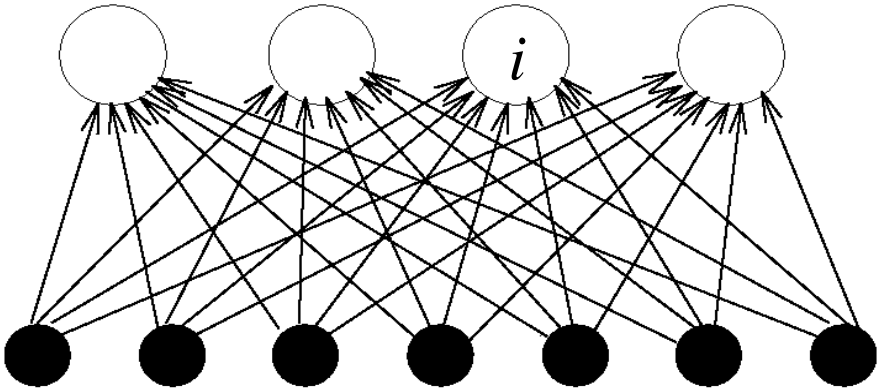
Linear neuron as a least-squares filter

- Consider: $y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}$, inputs-target pairs $\{\boldsymbol{x}(p), d(p)\}$, $p = 1, ..., N$
- Collect inputs $\mathbf{X} = [\boldsymbol{x}(1)\ \boldsymbol{x}(2)\ ....\ \boldsymbol{x}(N)]^{\mathrm{T}}$ ($N \times n$ matrix)
- Let $\boldsymbol{e} = [e(1)\ e(2)\ ...\ e(N)]^{\mathrm{T}}$ then output error $\boldsymbol{e} = \boldsymbol{d} - \mathbf{X}.\boldsymbol{w}$
- Gauss-Newton method: $E(\boldsymbol{w}) = \tfrac{1}{2} \sum_{p} (d^{(p)} - y^{(p)})^2$, compute $\nabla \boldsymbol{e}$ ($n \times N$)
- $j_{pk} = \partial e(p)/\partial w_k \Rightarrow$ Jacobian $\mathbf{J}(t) = [j_{pk}]$ is $(N \times n)$ $\mathbf{J}(t) = -\mathbf{X}(t) = [\nabla \boldsymbol{e}^{\mathrm{T}}]$
- $\boldsymbol{e}'(N,\boldsymbol{w}) = \boldsymbol{e}(\boldsymbol{w}) + \mathbf{J}(N).(\boldsymbol{w} - \boldsymbol{w}(N))$.
- Substitute $[N \equiv t]$ $\boldsymbol{w}(t+1) = \arg\min_{\boldsymbol{w}} \{1/2\ \|\boldsymbol{e}'(t,\boldsymbol{w})\|^2\}$
- Update $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - (\mathbf{J}^{\mathrm{T}}(t)\,\mathbf{J}(t))^{-1}\mathbf{J}(t)\,\boldsymbol{e}(t) = \boldsymbol{w}(t) + (\mathbf{X}^{\mathrm{T}}(t)\,\mathbf{X}(t))^{-1}\mathbf{X}(t)$
  $[\boldsymbol{d}(t) - \mathbf{X}(t)\,\boldsymbol{w}(t)] = [\mathbf{X}^{\mathrm{T}}(t)\,\mathbf{X}(t))^{-1}\mathbf{X}(t)]\,\boldsymbol{d}(t) \Rightarrow \boldsymbol{w}(t+1) = \mathbf{X}^{+}(t)\,\boldsymbol{d}(t)$
- *"The weight vector $\boldsymbol{w}(t+1)$ solves the least-squares problem in an observation interval until time t."*

Single-layer perceptrons

$$w_{ij}(t+1) = w_{ij}(t) + \alpha\ (d_i - y_i) f_i{}' x_j$$

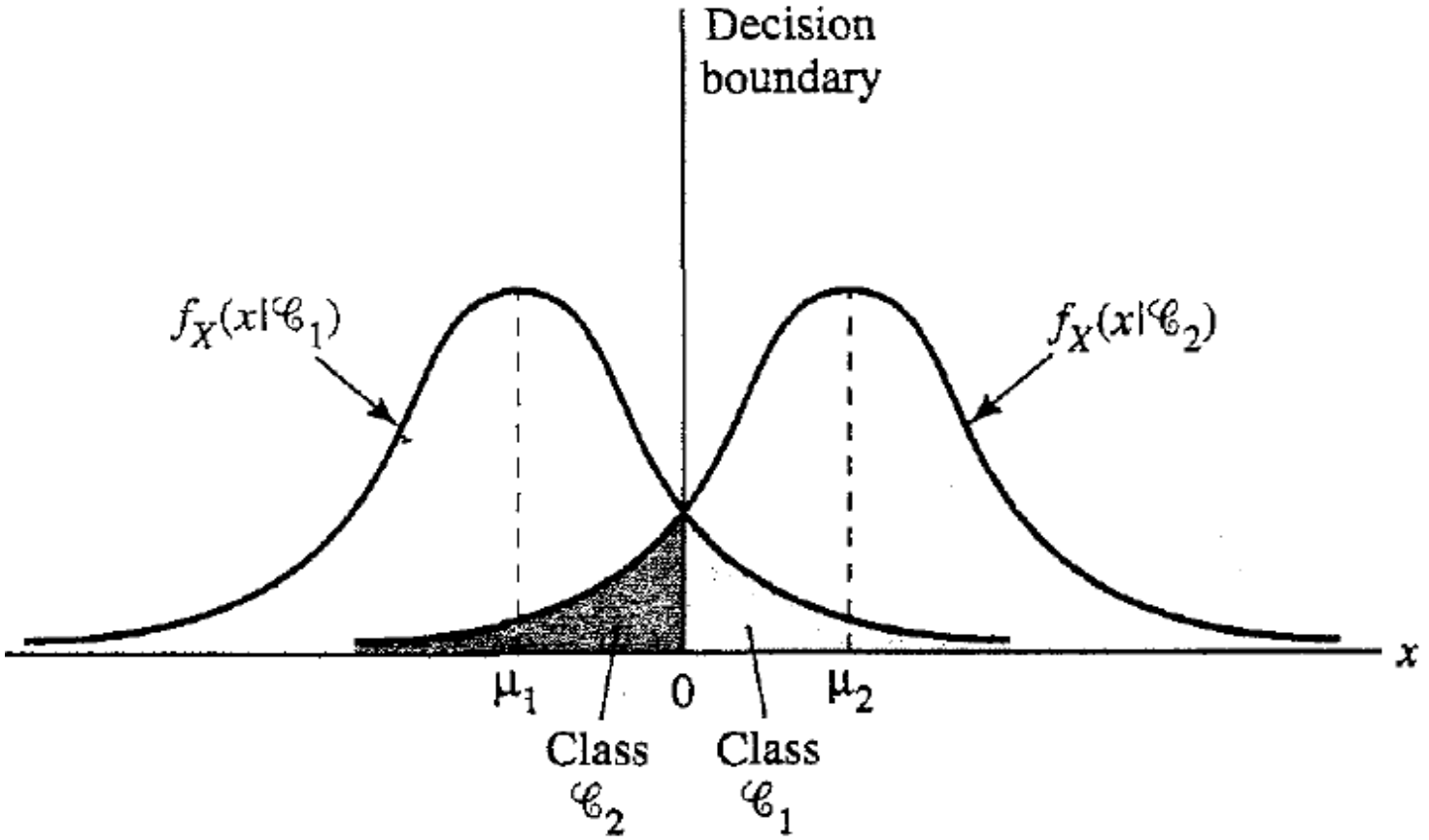
- $i = 1,2,..., N$ perceptrons work independently
- can classify linearly separable classes $(\max = 2^N)$



- for linear neurons we get the least-means-square (LMS) learning rule $w_{ij}(t+1) = w_{ij}(t) + \alpha\ (d_i - y_i)\ x_j$
- perceptron learning ↔ adaptive filtering (in signal processing)

Bayes classifier for two classes

- (linear) Bayes classifier for a Gaussian environment



Perceptron link to Bayes classifier

Assumptions:

- random vector $\boldsymbol{X}$, two classes C_1: $E[\boldsymbol{X}] = \boldsymbol{m}_1$, C_2: $E[\boldsymbol{X}] = \boldsymbol{m}_2$
- covariance matrix $\Omega = E[(\boldsymbol{X} - \boldsymbol{m}_1)(\boldsymbol{X} - \boldsymbol{m}_1)^{\mathrm{T}}] = E[(\boldsymbol{X} - \boldsymbol{m}_2)(\boldsymbol{X} - \boldsymbol{m}_2)^{\mathrm{T}}]$

  We can express conditional probability density function:

  $$f(\boldsymbol{x}|\mathrm{C}_i) = [(2\pi)^{m/2}\det(\Omega)^{1/2}]^{-1} \exp[-\tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{m}_i)^{\mathrm{T}}\Omega^{-1}(\boldsymbol{x} - \boldsymbol{m}_i)]$$

  $\boldsymbol{x}$ – observation vector, $i = \{1,2\}$
- the 2 classes are equiprobable, i.e. $p_1 = p_2$ (a priori probs)
- misclassifications carry the same cost, i.e. $\omega_{12} = \omega_{21}$
- Bayes classifier: "If $p_1(\omega_{21} - \omega_{11}) f(\boldsymbol{x}|\mathrm{C}_1) > p_2(\omega_{12} - \omega_{22}) f(\boldsymbol{x}|\mathrm{C}_2)$ *holds, assign the observation vector* $\boldsymbol{x}$ *to* C_1. *Otherwise, assign it to* C_2."

Bayes classifier (ctd)

- Define likelihood ratio $\Lambda(\boldsymbol{x}) = f(\boldsymbol{x}|\mathrm{C}_1) / f(\boldsymbol{x}|\mathrm{C}_2)$ and threshold $\xi = [p_2(\omega_{12} - \omega_{22})] / [p_1(\omega_{21} - \omega_{11})]$. Then:
- $\log \Lambda(\boldsymbol{x}) = -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{m}_1)^{\mathrm{T}}\Omega^{-1}(\boldsymbol{x} - \boldsymbol{m}_1) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{m}_2)^{\mathrm{T}}\Omega^{-1}(\boldsymbol{x} - \boldsymbol{m}_2)$
- $\log \xi = 0$
- *Then*: we get a linear Bayes classifier $y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$ where

  $y = \log \Lambda(\boldsymbol{x}), \quad \boldsymbol{w} = \Omega^{-1}(\boldsymbol{m}_1 - \boldsymbol{m}_2), \quad b = \frac{1}{2} (\boldsymbol{m}_2^{\mathrm{T}}\Omega^{-1}\boldsymbol{m}_2 - \boldsymbol{m}_1^{\mathrm{T}}\Omega^{-1}\boldsymbol{m}_1) \rightarrow$

  log-likelihood test: $If \ y > 0, then \ \boldsymbol{x} \in \mathrm{C}_1, else \ \mathrm{C}_2.$
- Differences b/w Perceptron (P) and Bayes classifier (BC):
  - P assumes linear separability, BC does not
  - P convergence algorithm is nonparametric, unlike BC
  - P convergence algorithm is adaptive and simple, unlike BC.

Alternative activation function and error function

- Error function – cross-entropy (for one output):

  [relative entropy b/w empirical probability distribution $(d^{(p)}, 1- d^{(p)})$ and output distribution $(y, 1-y)$]

$$E_{CE}(\boldsymbol{w}) = \Sigma_p E(p) = -\Sigma_p [d^{(p)} \ln y^{(p)} + (1 - d^{(p)}) \ln (1 - y^{(p)})]$$

- minimization of $E(p)$ results in learning rule:

$$w_j(t+1) = w_j(t) + \alpha \ (d^{(p)} - y^{(p)}) \ x_j$$

- With more outputs one can use softmax activation function and compute cross-entropy error

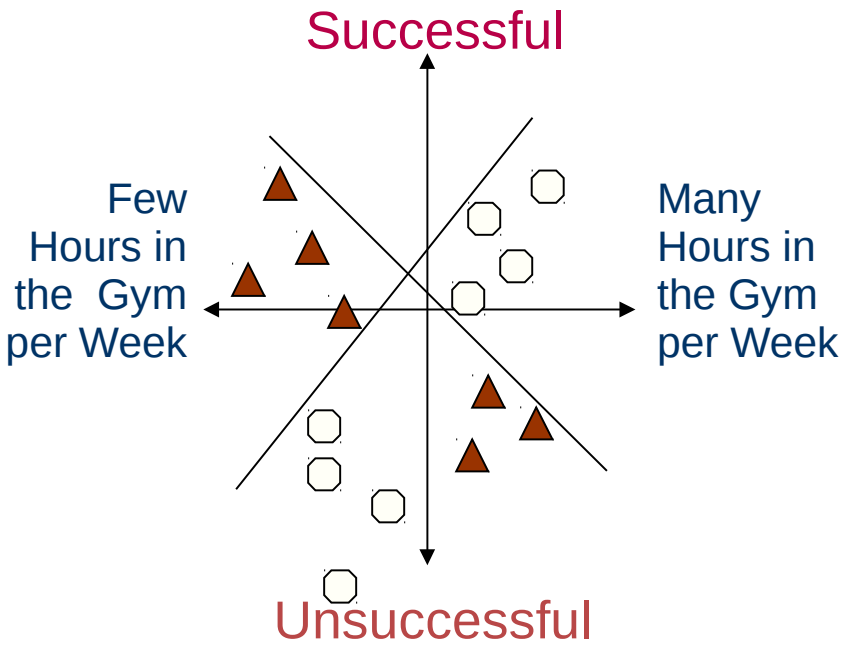
$$E_{CE} = \Sigma_p \Sigma_i d_i^{(p)} \ln y_i^{(p)}$$

$$y_i = \frac{\exp(net_i)}{\sum_j \exp(net_j)}$$

- *Note*: In case of 2 classes one can use either two softmax units, or one logistic unit. In the latter case (C_1: $d = 0$, C_2: $d = 1$), output y can be interpreted as $P(\boldsymbol{x} \in C_2)$.

Perceptron limits

- cannot separate linearly non-separable classes



Footballers

Academics

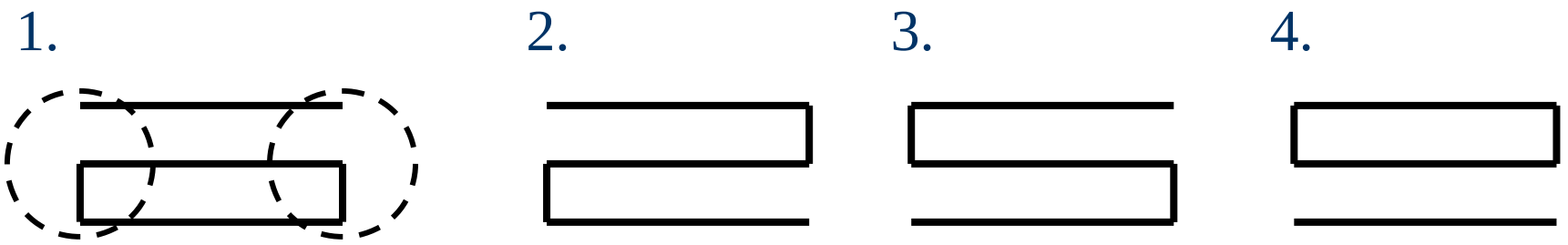
…despite the simplicity of their relationship:

Academics = Successful XOR Gym

This failure caused the loss of interest in connectionism by many (in 1970s). Many categories in real problems turned out to be linearly non-separable.

M. Minsky & S. Papert (1969). *Perceptrons*, MIT Press, Cambridge, MA.

A deeper problem behind XOR



- Consider a perceptron classifying shapes as connected or disconnected and taking inputs from shape ends (shown as dashed circles for pattern 1)
- The problem arises because single layer of processing local knowledge cannot be combined into global knowledge
- No feature-weighing machine (such as a simple perceptron) can do this type of separation, because information about the relation between the bits of evidence is lost (mathematically proven by Minsky & Papert, 1969)

Summary

- single perceptron can separate two linearly separable classes
- binary (McCulloch & Pitts) and continuous (Rosenblatt) perceptron
- perceptron as a detector
- gradient descent learning
- link to adaptive filtering – error correction learning
- two types of error functions
- link to statistics: probabilistic Bayes classifier
- perceptron limitations

3

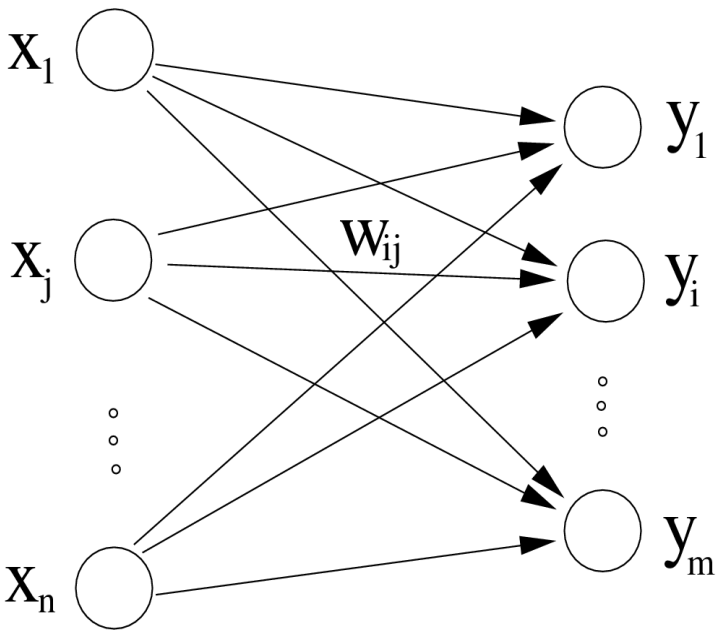
Linear networks

Linear NN models

Input vector: $\boldsymbol{x} = [x_1, x_{2,..}, x_n]^T$

Output vector: $\boldsymbol{y} = [y_1, y_{2,..}, y_m]^T$

Weight matrix: $\mathbf{W}$ ~ type $[m \times n]$



Linear transformation $\varphi : \Re^n \rightarrow \Re^m$, $\boldsymbol{y} = \mathbf{W}\boldsymbol{x}$

☹ ignores saturation property of neurons

☺ allows to find analytic solutions using linear algebra.

(Kohonen, 1970; Anderson, 1972; Cooper, 1973)

- Adding layers in a linear NN does not appear reasonable (since no complexity is added).
- Opposite argument: it allows nonlinear learning dynamics in linear deep networks (Saxe, 2015).

Linear NN – analytic solution

Let's consider the train set: $A_{train} = \{(\boldsymbol{x}^{(p)}, \boldsymbol{y}^{(p)}), p = 1,2,...,N\}$.

We look for matrix $\mathbf{W}$ that satisfies $\boldsymbol{y}^{(p)} = \mathbf{W}\boldsymbol{x}^{(p)}$, for each p.

In matrix notation: $\mathbf{Y} = \mathbf{W}\,\mathbf{X}$

$$\underset{(m\times N)}{[\boldsymbol{y}^{(1)}\ \boldsymbol{y}^{(2)} \ldots \boldsymbol{y}^{(N)}]} = \underset{(m\times n)}{\mathbf{W}} \times \underset{(n\times N)}{[\boldsymbol{x}^{(1)}\ \boldsymbol{x}^{(2)} \ldots \boldsymbol{x}^{(N)}]}$$

If $\mathbf{X}$ was regular (i.e., square matrix with $N = n$, linearly independent rows) then X^{-1} would exist and $\mathbf{W} = \mathbf{Y}\mathbf{X}^{-1}$.

However, in general we cannot assume $N = n$, nor linear independence of input vectors ($\Rightarrow \mathbf{X}^{-1}$ does not exist). Only degenerate solutions exist:

$$\mathbf{W} = \mathbf{Y}\mathbf{X}^{+}$$

$\mathbf{X}^{+}$ is called (Moore-Penrose) pseudoinverse matrix of $\mathbf{X}$. (Theorem: $\forall\ \mathbf{X},\ \exists\ \mathbf{X}^{+}$) (Properties: 1) $\mathbf{X}\mathbf{X}^{+}\mathbf{X} = \mathbf{X}$, 2) $\mathbf{X}^{+}\mathbf{X}\mathbf{X}^{+} = \mathbf{X}^{+}$, 3) symmetric $\mathbf{X}\mathbf{X}^{+}$ and $\mathbf{X}^{+}\mathbf{X}$).

a) $\mathbf{X}^{+} = \mathbf{X}^{T}(\mathbf{X}\mathbf{X}^{T})^{-1}$, if $n < N$ and $rank(\mathbf{X}) = n$.

b) $\mathbf{X}^{+} = (\mathbf{X}^{T}\mathbf{X})^{-1}\mathbf{X}^{T}$, if $n > N$ and $rank(\mathbf{X}) = N$.

Auto-associator case

Let's consider $N < n$ and the autoassociative case: $\boldsymbol{y}^{(p)} = \boldsymbol{x}^{(p)}$, $m = n$

Model is supposed to remember N prototypes $[\boldsymbol{x}^{(1)}\ \boldsymbol{x}^{(2)} \ldots \boldsymbol{x}^{(N)}]$.

Goal: train on prototypes and then submit a corrupted version of a prototype. Model should be able to reconstruct it.

Since $\mathbf{Y} = \mathbf{X}$, $\mathbf{W} = \mathbf{X}\mathbf{X}^{+}$. How to interpret $\mathbf{W}$?

In special case, which is too restrictive ($N = n$, linearly independent inputs), we would have a trivial solution $\mathbf{W} = \mathbf{I}$ (identity).

How about a general case?

Basics of linear vector spaces

Let's have a linear space $\mathfrak{R}^n$.

Linear manifold $\mathcal{L} = \{\boldsymbol{x} \in \mathfrak{R}^n \mid \boldsymbol{x} = a_1 \boldsymbol{x}^{(1)} + a_2 \boldsymbol{x}^{(2)} + \ldots + a_N \boldsymbol{x}^{(N)},\ a_p \neq 0\}$

$\mathcal{L} \subset \mathfrak{R}^n$

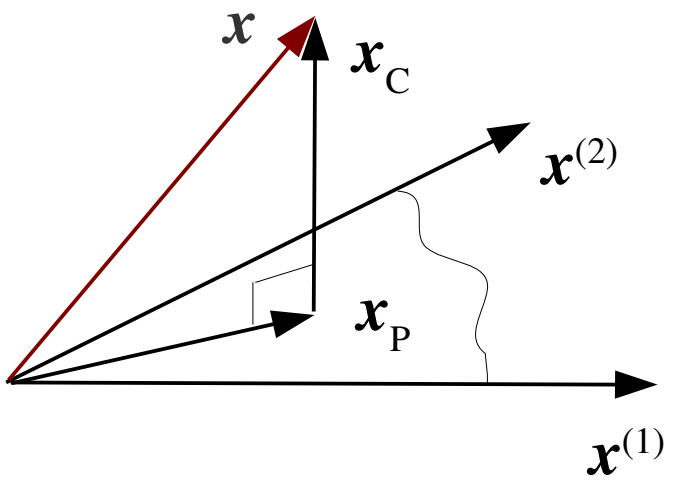
Orthogonal complement $\mathcal{L}^{\perp} = \{\boldsymbol{x} \in \mathfrak{R}^n \mid \boldsymbol{x} \perp \mathcal{L}\}$

Hence, $\mathcal{L} \cup \mathcal{L}^{\perp} = \mathfrak{R}^n$

Each vector $\boldsymbol{x} \in \mathfrak{R}^n$ can be uniquely decomposed:

$\boldsymbol{x} = \boldsymbol{x}_{\mathrm{P}} + \boldsymbol{x}_{\mathrm{C}}$

where $\boldsymbol{x}_{\mathrm{P}} \in \mathcal{L}$ and $\boldsymbol{x}_{\mathrm{C}} \in \mathcal{L}^{\perp}$.



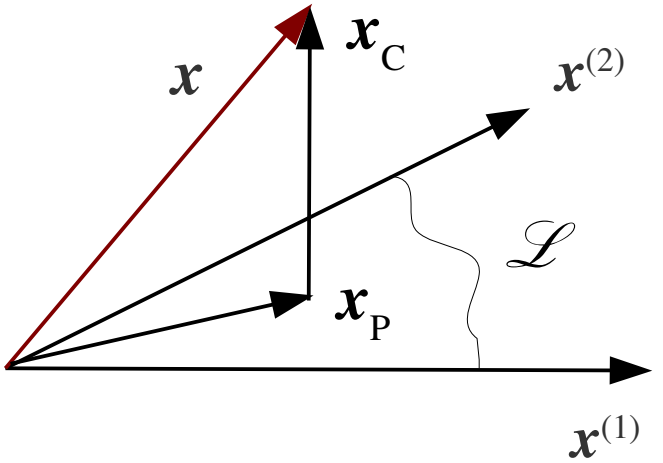
What does an autoassociative NN do?

Training set $A_{train} = \{\boldsymbol{x}^{(p)}, p = 1,2,...,N\}$ forms the linear manifold $\mathscr{L}$.

NN considers every departure $\boldsymbol{x}$ from $\mathscr{L}$ as added noise that needs to be filtered out by projecting $\boldsymbol{x}$ to $\mathscr{L}$:

We need to show that output $\mathbf{W}\boldsymbol{x} = \mathbf{X}\mathbf{X}^{+}\boldsymbol{x} = \boldsymbol{x}_P$ (filtered version of $\boldsymbol{x}$), i.e. that operator $\mathbf{W} = \mathbf{X}\mathbf{X}^{+}$ makes an orthogonal projection to $\mathscr{L}$.

Alternatively, the NN model with operator $\mathbf{W} = \mathbf{I} - \mathbf{X}\mathbf{X}^{+}$ will be called novelty detector, where $\mathbf{W}\boldsymbol{x} = \boldsymbol{x}_C \in \mathscr{L}^{\perp}$.



Now assume: you learned N patterns, and want to add $(N+1)$-st pattern.
How to change $\mathbf{W}$ efficiently?

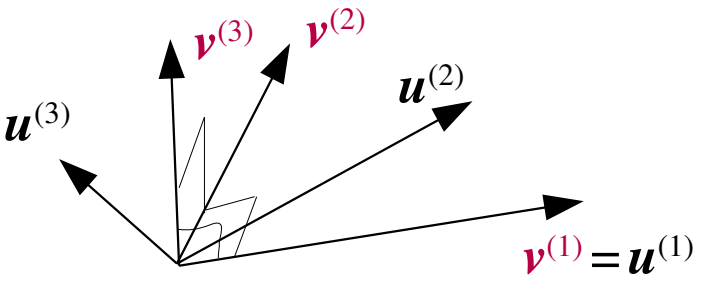
Gram-Schmidt orthogonalization process

Let's have a base $\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ldots, \boldsymbol{u}^{(k)} \in \mathscr{L}$, for which we want to create an orthogonal base $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(k)} \in \mathscr{L}$.

Procedure:

Set $\boldsymbol{v}^{(1)} = \boldsymbol{u}^{(1)}$. In space with base $\boldsymbol{v}^{(1)}$, $\boldsymbol{u}^{(2)}$ let's find vector $\boldsymbol{v}^{(2)}$ such that $\boldsymbol{v}^{(2)} \perp \boldsymbol{v}^{(1)}$. Hence, $\boldsymbol{v}^{(2)} = a_1 \boldsymbol{v}^{(1)} + a_2 \boldsymbol{u}^{(2)}$.

$$\boldsymbol{v}^{(2)} = \boldsymbol{u}^{(2)} - \frac{\boldsymbol{v}^{(1)T} \boldsymbol{u}^{(2)}}{\left|\boldsymbol{v}^{(1)}\right|^2} \boldsymbol{v}^{(1)}$$



Recursive formula: we have $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(k-1)}$ and compute $\boldsymbol{v}^{(k)}$ such that $\boldsymbol{v}^{(k)} \perp \boldsymbol{v}^{(i)}$, $i = 1,2, \ldots, k\text{-}1$

$$\boldsymbol{v}^{(k)} = \boldsymbol{u}^{(k)} - \sum_{i=1}^{k-1} \frac{\boldsymbol{v}^{(i)T} \boldsymbol{u}^{(k)}}{\left|\boldsymbol{v}^{(i)}\right|^2} \boldsymbol{v}^{(i)}$$

General Inverse model

We have patterns $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}$ and the associated orthogonal base (via Gram-Schmidt process) $\tilde{\boldsymbol{x}}^{(1)}, \tilde{\boldsymbol{x}}^{(2)}, \ldots, \tilde{\boldsymbol{x}}^{(N)}$

$\mathbf{W}$ is computed recursively, upon adding a new input $\boldsymbol{x}$.

1. Initialize $\mathbf{W}^{(0)} = \mathbf{0}$.

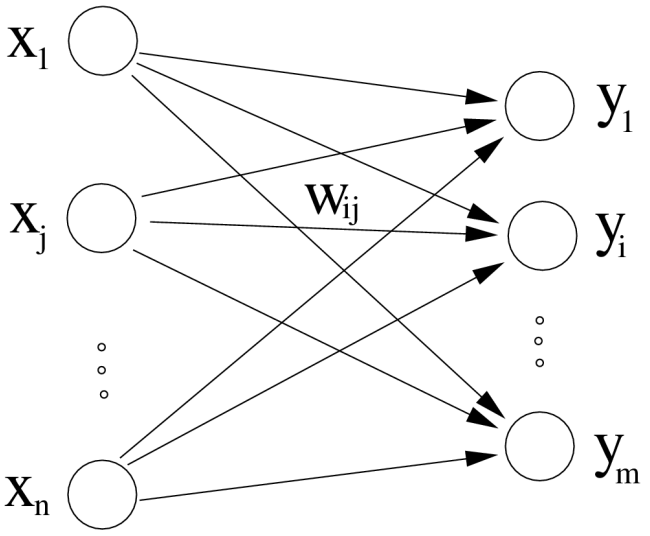
2. $$W^{(N+1)} = W^{(N)} + \frac{\tilde{\boldsymbol{x}}^{(N+1)} \tilde{\boldsymbol{x}}^{(N+1)T}}{\left|\tilde{\boldsymbol{x}}^{(N+1)}\right|^2}$$

$$\tilde{\boldsymbol{x}}^{(N+1)} = \boldsymbol{x}^{(N+1)} - \boldsymbol{W}^{(N)} \boldsymbol{x}^{(N+1)}$$

Correlation Matrix Memory

Let's have train set: $A_{train} = \{(\boldsymbol{x}^{(p)}, \boldsymbol{y}^{(p)}), p = 1,2,...,N\}$.

and let weights be set using Hebb's (1949) hypothesis:



$$w_{ij} = \sum_{k=1}^{N} x_j^{(k)} y_i^{(k)} \qquad \boldsymbol{W} = \sum_{k=1}^{N} \boldsymbol{y}^{(k)} \boldsymbol{x}^{(k)T} = \boldsymbol{Y}\boldsymbol{X}^{\boldsymbol{T}}$$

Let's focus again on auto-associative case: $\mathbf{W} = \mathbf{X}\mathbf{X}^T$.

$$x_i = \sum_{k=1}^{N} w_{ij} x_j^{(k)}$$

Recursive computation:

$$\boldsymbol{W}(N+1) = \boldsymbol{W}(N) + \boldsymbol{x}^{(N+1)} \boldsymbol{x}^{(N+1)T}$$

Cross-talk in CMM

Response to input $\boldsymbol{x}^{(p)}$, $p \in \{1,2,...,N\}$:

$$\boldsymbol{W}\,\boldsymbol{x}^{(p)} = \boldsymbol{X}\boldsymbol{X}^{T}\,\boldsymbol{x}^{(p)} = \sum_{k=1}^{N} \boldsymbol{x}^{(k)}\,\boldsymbol{x}^{(k)T}\,\boldsymbol{x}^{(p)} = \boldsymbol{x}^{(p)}\,\boldsymbol{x}^{(p)T}\,\boldsymbol{x}^{(p)} + \sum_{k=1;\,k\neq p}^{N} \boldsymbol{x}^{(k)}\,\boldsymbol{x}^{(k)T}\,\boldsymbol{x}^{(p)}$$

$$\boldsymbol{W}\,\boldsymbol{x}^{(p)} = \boldsymbol{x}^{(p)}\left|\boldsymbol{x}^{(p)}\right|^{2} + C(p)$$

noise vector due to "cross-talk" from other inputs

If the inputs are orthogonal, then $C(p) = 0$ and $\mathbf{X}^{\mathrm{T}} = \mathbf{X}^{-1} = \mathbf{X}^{+}$, i.e. CMM = GI.

Max. capacity of both GI and CMM equals n.

For sufficiently dissimilar inputs, CMM is a good faster alternative to GI.

Interpretation of CMM behavior

Let's consider inputs $\boldsymbol{x}^{(1)}$ $\boldsymbol{x}^{(1)}$ ... $\boldsymbol{x}^{(N)}$ as realizations of a random variable $X = (X_1, X_2, ..., X_n)^{\mathrm{T}}$, where $E(X_i) = 0$, $i = 1,2,...,n$. Then

$$w_{ij} = \sum_{k=1}^{N} x_j^{(k)} x_i^{(k)} \propto cov(X_i, X_j)$$ [unbiased estimate]

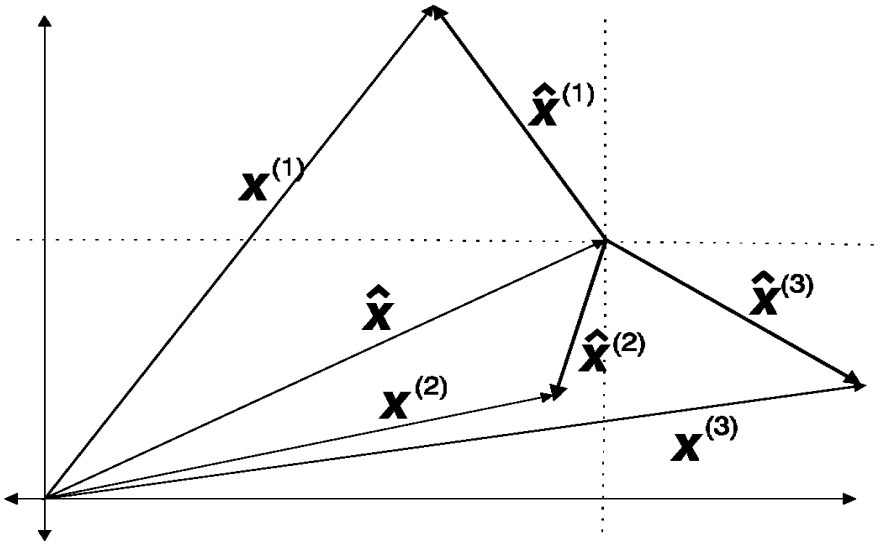
$$cov(X_i, X_j) = E[(X_i - E(X_i)).(X_j - E(X_j))] = E(X_i X_j)$$

w_{ij} carries information about linear correlations between nodes i and j.

Reduced (noisy) input component x_i can be recovered (bootstrapped) by other active components strongly correlated with i.

Shifting coordinates decreases “cross-talk”:

$$\hat{\boldsymbol{x}}^{(i)} = \boldsymbol{x}^{(i)} - \frac{1}{N} \sum_{k=1}^{N} \boldsymbol{x}^{(k)}$$



(Kvasnička et al, 1997)

Example: 8 faces from CMU image data repository

face 1

face 2

face 3

face 4

face 5

face 6

face 7

face 8

face 1 – corrupt

(Courtesy of P. Tiňo)

Recall by GI and CMM

GI

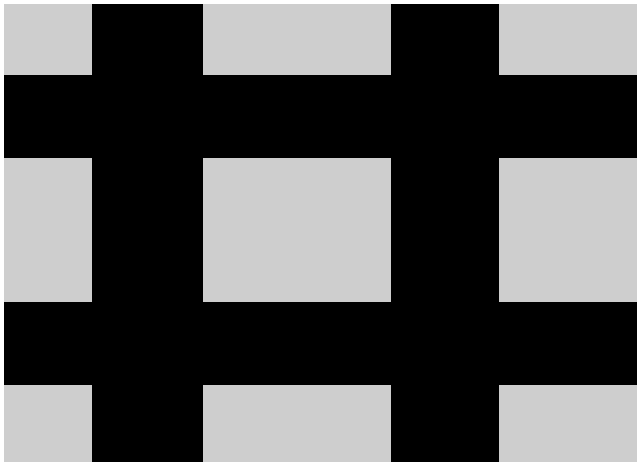
CMM

Novelty detection

↔ GI's novelty detection

Response to a new input

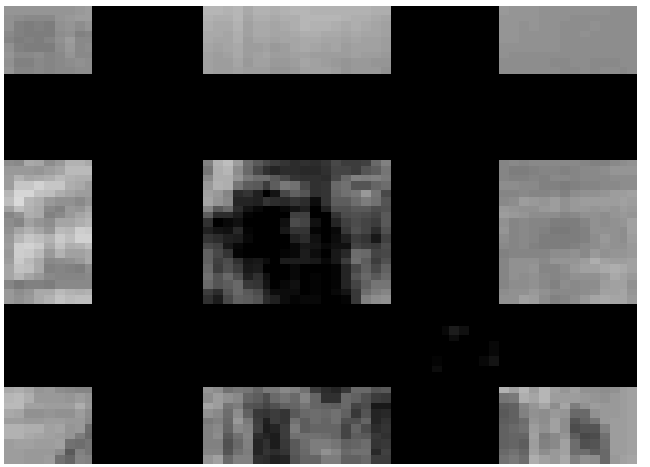
New input



Restoration – GI

Restoration – GMM

Novelty detection



Summary

- Linear models studied during connectionist depression in the 1970s
- Single layer models as auto-associative memories
- Analytic solutions possible
- General inverse model – noise filtering by projection to linear manifold (of the training data)
- GI – as novelty detector
- Correlation Matrix Memory – Hebbian-based learning, subject to cross-talk
- GI better in general, for sufficiently dissimilar inputs both models are comparable

4

Multi-layer perceptrons

Multi-layer perceptrons

- Generalization of simple perceptrons
- Features:
  - contains hidden-layer(s)
  - neurons have non-linear activation function (e.g. logistic)
  - full connectivity b/w layers
- (supervised) error "back-propagation" training algorithm introduced
- originated after 1985: Rumelhart & McClelland: *Parallel distributed processing* (algorithm described earlier by Werbos, 1974)
- response to earlier critique of perceptrons (Minsky & Papert, 1969)

Two-layer perceptron

- Inputs $\boldsymbol{x}$, weights $\boldsymbol{w}$, $\boldsymbol{v}$, outputs $\boldsymbol{y}$
- Nonlinear activation function f
- Unit activation:

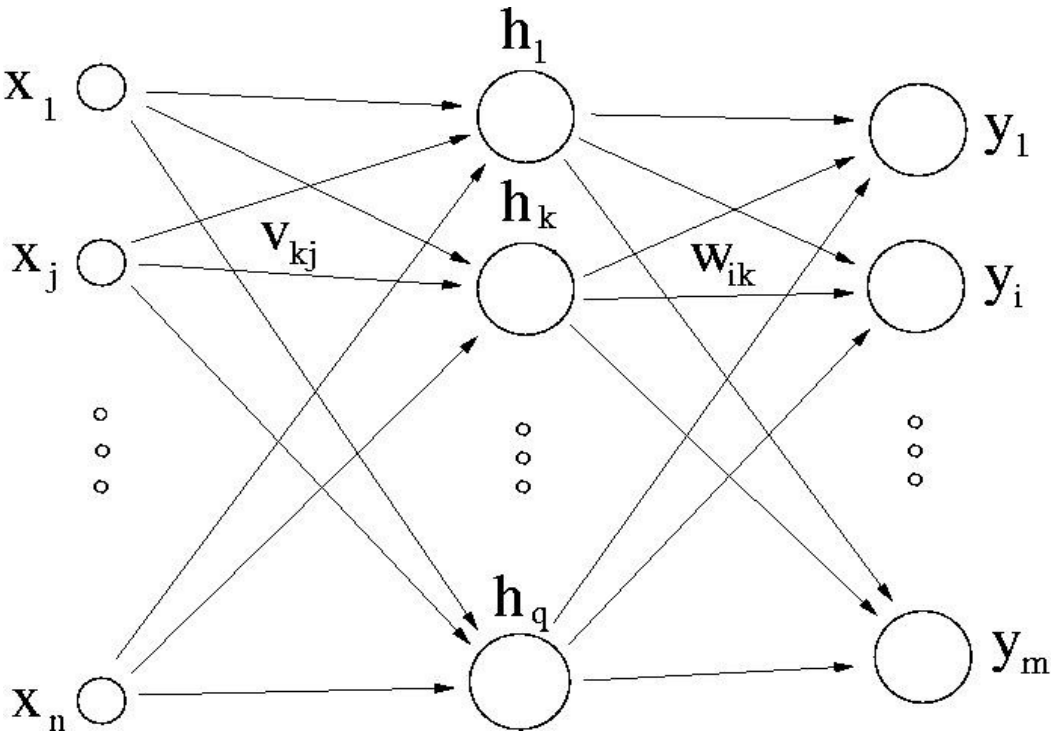
$$h_k = f\left(\sum_{j=1}^{n+1} v_{kj} x_j\right)$$

$$y_i = f\left(\sum_{k=1}^{q+1} w_{ik} h_k\right)$$

- Bias input: $x_{n+1} = h_{q+1} = -1$
- Activation function examples:

$$f(net) = 1 / (1 + \exp(-net))$$

$$f(net) = \tanh(net)$$



How to apply error?

- Output layer – application of delta rule
- How to compute error at hidden layer(s)?
- Instantaneous output error: $e^{(p)} = \frac{1}{2} \sum_i (d_i^{(p)} - y_i^{(p)})^2$
- We will show that error can be back-propagated across layers backwards
- At each layer the (local) weight correction has this form:

  (weight change) = (learning rate)*(unit error)*(input)

  - We will derive equations for BP algorithm.

Learning equations for original BP

Hidden-output weights:

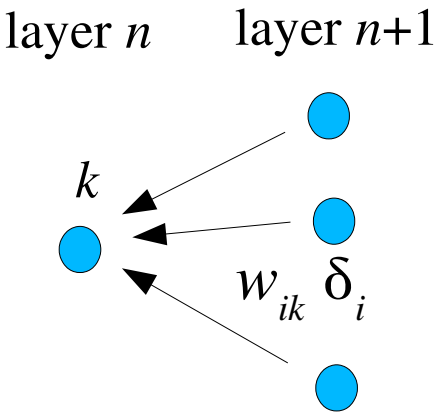
$$w_{ik}(t+1) = w_{ik}(t) + \alpha \delta_i h_k \quad \text{where} \quad \delta_i = (d_i - y_i) f_i'$$

Input-hidden weights:

$$v_{kj}(t+1) = v_{kj}(t) + \alpha \ \delta_k x_j \quad \text{where} \quad \delta_k = (\Sigma_i w_{ik} \delta_i) f_k'$$

BP provides an “approximation” to the trajectory in weight space computed by the method of steepest descent.

- smoothness of the trajectory depends on ·



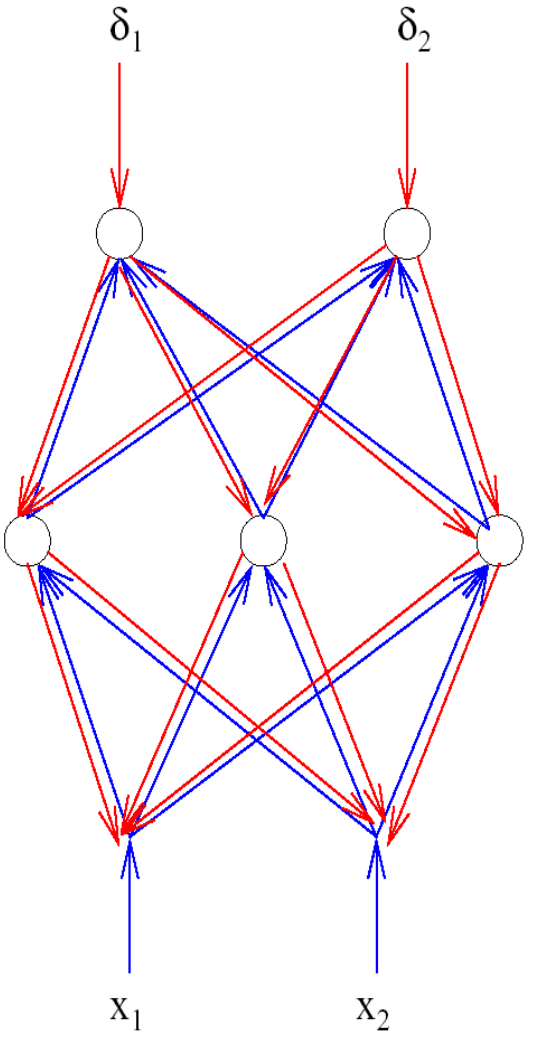
Summary of back-propagation algorithm

Given: training data: input-target $\{\boldsymbol{x}^{(p)}, \boldsymbol{d}^{(p)}\}$ pairs

Initialization: randomize weights, set learning parameters

Training:

1. choose input $\boldsymbol{x}^{(p)}$, compute outputs $\boldsymbol{y}^{(p)}$ (forward pass),
2. evaluate chosen error function $e(t)$, $E \leftarrow E + e(t)$
3. compute δ_i , δ_k (backward pass)
4. adjust weights Δw_{ik} and Δv_{kj}
5. if all patterns used, then goto 6, else go to 1
6. if stopping_criterion is met, then end
   else permute inputs and go to 1



No well-defined stopping criteria exist. Suggestions:

- when change in E_{epoch} is sufficiently small (<1%)
- when generalization performance is adequate

Adding a momentum

(Plaut et al., 1986)

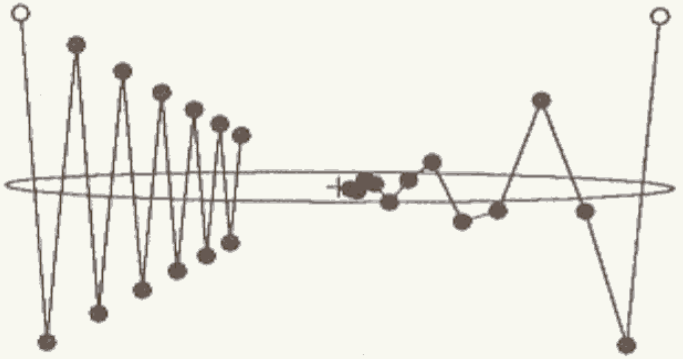


FIGURE 6.3 Gradient descent on the simple quadratic surface of Fig. 5.10. Both trajectories are for 12 steps with $\eta = 0.0476$, the best value in the absence of momentum. On the left there is no momentum ($\alpha = 0$), while $\alpha = 0.5$ on the right.

$E = x^2 + 20y^2$

$$\Delta w_{ik}(t) = \alpha \ \delta_i(t)\, h_k(t) + \mu \ \Delta w_{ik}(t\text{-}1)$$

$$\Delta w_{ik}(t) \ = \ \Sigma^{t}_{t'=0} \mu^{t-t'} \delta_i(t')\, h_k(t') = \Sigma^{t}_{t'=0} \mu^{t-t'} \partial E(t)/\partial w_{ik}$$

Observations:

- for convergence of time series: $0 \leq \ \mu \ < 1$
- acceleration/shrinking of $\Delta w_{ik}(t)$ depends on signs of $\partial E(t)/\partial w_{ik}$

Sequential and batch modes of training

Sequential mode

- on line, stochastic
- randomly permute training examples in each epoch
- requires less local storage
- appears to be superior in most cases
- difficult to establish theoretical conditions for convergence

Batch mode

- adaptation performed at the end of each epoch
- provides an accurate estimate of gradient vector

$$E_{av} = 1/(2N) \sum^{N}_{p=0} (\boldsymbol{d}^{(p)} - \boldsymbol{y}^{(p)})^2$$

$$\Delta w_{ik} \propto -\partial E_{av}(t)/\partial w_{ik}$$

$$\Delta v_{kj} \propto -\partial E_{av}(t)/\partial v_{kj}$$

Heuristics to improve BP

1. sequential mode is usually faster
2. shuffle the patterns before each epoch
3. consider “emphasizing scheme” with care
4. consider bipolar activation function (e.g. *tanh*)
5. use appropriate target values (ϵ-tolerance)
6. normalize input (mean removal, decorrelation, covariance equalization)
7. small initial weights
8. learn from hints if possible
9. proper learning rate

Output representation and decision rule

- for binary (0/1) targets, use logistic function
- for categorical targets, use 1-of-M coding and softmax activation
- for continuous-valued targets with a bounded range, logistic and tanh functions can be used (with proper scaling)
- if target values > 0, but have no known upper bound, you can use an exponential output activation function (beware of overflow)
- for continuous-valued targets with no known bounds, use the identity or linear activation function

M-class classification problem:

if softmax: outputs estimate a posteriori class probabilities $P(C_i \mid \boldsymbol{x})$

MLP as a universal approximator

Theorem: Let's have $A_{train} = \{\boldsymbol{x}^{(1)},..., \boldsymbol{x}^{(p)},..., \boldsymbol{x}^{(N)}\}$, $\boldsymbol{x}^{(p)} \in \mathbb{R}^n$. For $\epsilon > 0$ and arbitrary continuous function F: $\mathbb{R}^n \rightarrow (0,1)$ defined on discrete set A_{train} there exists such a function G:

$$G(\boldsymbol{x}^{(p)}) = f\left(\sum_{k=1}^{q+1} w_k f\left(\sum_{j=1}^{n+1} v_{kj} x_j^{(p)}\right)\right)$$

where parameters w_k, $v_{kj} \in \mathbb{R}$ and $f(z) = \mathbb{R} \rightarrow (0,1)$ is a continuous and monotone-increasing function satisfying $f(-\infty) = 0$ and $f(\infty) = 1$, such that:

$$\Sigma_p \mid F(\boldsymbol{x}^{(p)}) - G(\boldsymbol{x}^{(p)}) \mid < \epsilon.$$

We say that G approximates F on A_{train} with accuracy ϵ.

G can be interpreted as a 2-layer feedforward NN with 1 output neuron.

- it is an existence theorem
- curse of dimensionality – problem to get a dense sample for large n and complex F

Hecht-Nielsen (1987), Hornik, Stinchcombe & White (1989)

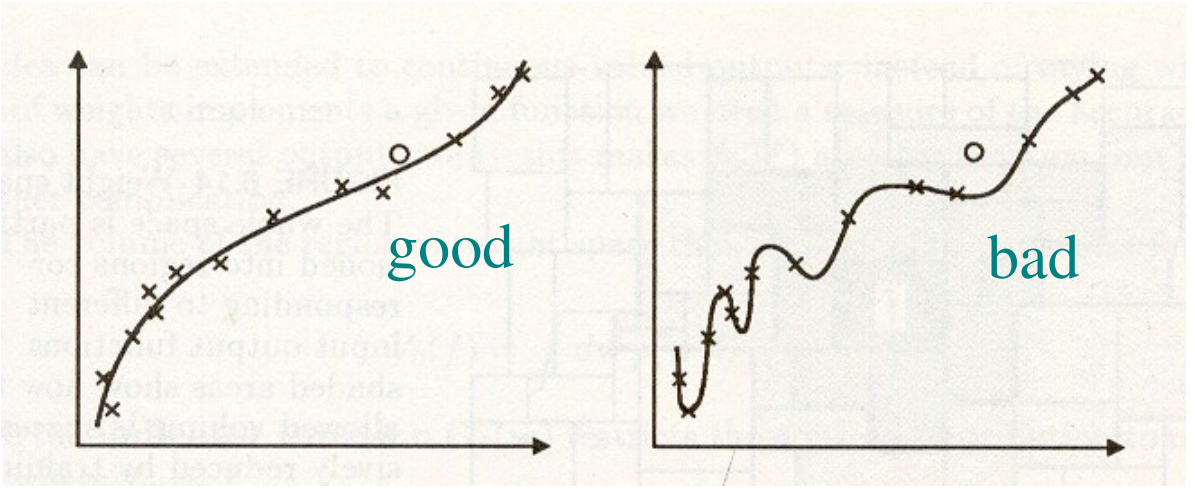
Generalization

= performance on test set

$$A = A_{train} \cup A_{test}$$

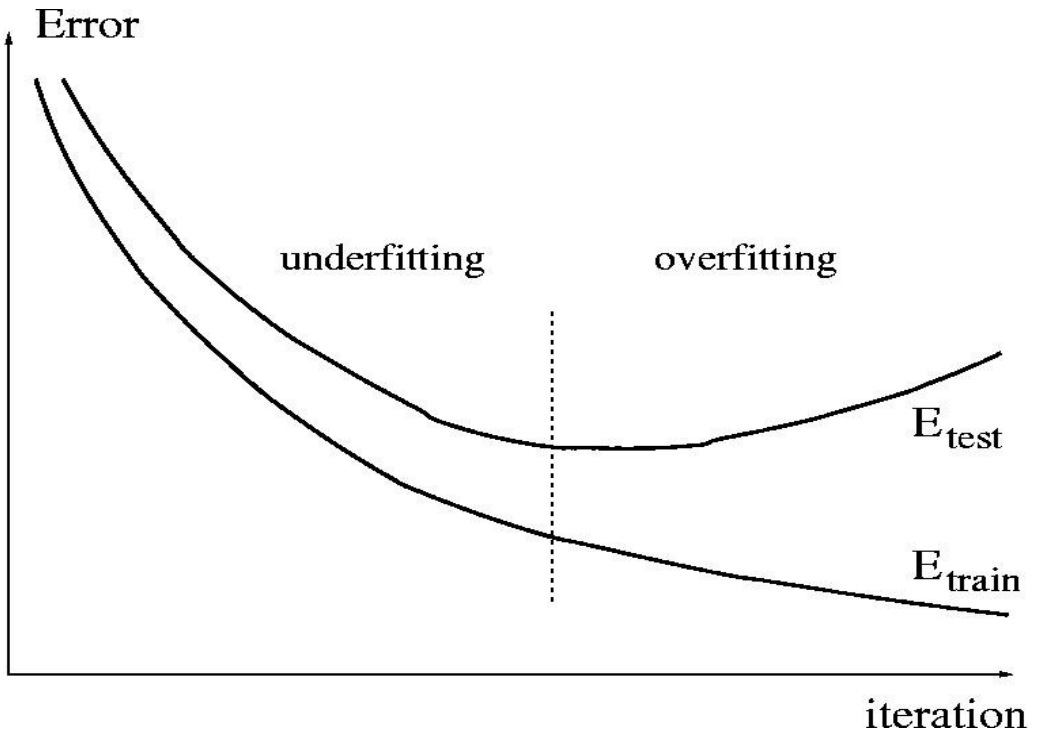
Influenced by:

- size of A_{train} and its representativeness
- architecture of NN
- complexity of the problem



Two perspectives:

1. A_{train} fixed, find optimal NN => network pruning, Occam's razor
2. NN fixed, find optimal size of A_{train} => theory based on VC dimension



Cross-validation

- in statistics (Stone, 1974): data partitioning = train + test
- training set (estimation subset + validation subset):
  - validate the model using a different data from training (i.e. parameter estimation)
  - find the best model (model selection) using early stopping
- test set – used for assessing the performance of the chosen model
- for large data sets
  - hold out (cross-validation) method is used, i.e. single split of the training set

k-fold cross-validation

- useful for smaller data sets: we need a statistically valid model
- Split $A = A^1_{val} \cup A^2_{val} \cup \ldots \cup A^k_{val}$ (A^i_{val} and A^j_{val} are non-overlapping)
- train each model *M* k-times, using $A^i_{train} = A \setminus A^i_{val}$, $i=1, 2,..., k$.

  Use early stopping in training.
- For each *M* compute the cross-validation coefficient:

$$CV(M) = 1/k \sum_{i=1}^{k} E^i_{val}(M)$$

- Choose *M* with smallest CV; computationally more demanding
- Extreme case: *leave-one-out* CV ($k = |A|$ trainings)

Automatic model selection

- Network growing and pruning techniques
- Example of a growing architecture: cascade-correlation learning (Fahlman & Lebiere, 1990)
  - start with a network without hidden units
  - assess model performance
  - add hidden units (one by one) by receiving connections from all preexisting hidden units and from all inputs
  - repeat until satisfactory performance is achieved

Network pruning techniques

Two ways: 1) regularization, 2) deletion of connections

1. Regularization: (risk minimization)

$$R(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda\, C(\boldsymbol{w})$$ [error + complexity penalty]

- weight decay: (e.g.) $C(\boldsymbol{w}) = \Sigma_l w_l^2$ λ- regularization factor

  leads to $w_l^{new}(t) = \epsilon\ w_l^{new}(t), \qquad 0 \ll \epsilon < 1$

- weight elimination: w_0 - preset parameter

  - prune unreliable weights: $|w_l| \ll w_0$

$$C(\boldsymbol{w}) = \Sigma_l \frac{(w_l/w_0)^2}{1+(w_l/w_0)^2}$$

2. Hessian-based pruning: ($\mathbf{H}$ – use 2nd order info)

- eliminate weights whose deletion will minimally increase E

Weight optimization techniques

$$E(\boldsymbol{w}) = E(\boldsymbol{w}_0) + \boldsymbol{g}^{\mathrm{T}}(\boldsymbol{w}_0)\Delta\boldsymbol{w} + 1/2\Delta\boldsymbol{w}^{\mathrm{T}}\mathbf{H}(\boldsymbol{w}_0)\Delta\boldsymbol{w} + O(\|\boldsymbol{w}\|^3) \qquad f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(0)}{i!} x^i$$

$$\Delta\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{w}_0 \qquad \boldsymbol{g}^T(\boldsymbol{w}_0) = \nabla E(\boldsymbol{w}_0) = \left[\frac{\partial E}{\partial w_1}/\boldsymbol{w}_0, \ldots, \frac{\partial E}{\partial w_n}/\boldsymbol{w}_0\right]^T \qquad \boldsymbol{H} = \left[\frac{\partial^2 E}{\partial w_i \partial w_j}/\boldsymbol{w}_0\right]_{ij}$$

- Error back-propagation is a linear approx. of E: $\Delta\boldsymbol{w}(t) = -\alpha\,\boldsymbol{g}(t)$
- Quadratic approximation: → Newton's method
- Differentiation of Taylor ex. → $\nabla E(\boldsymbol{w}) = \nabla E(\boldsymbol{w}_0) + H(\boldsymbol{w} - \boldsymbol{w}_0) + \ldots$
- Find minimum of $E(\boldsymbol{w})$ (i.e. deriv $= 0$) → $\Delta\boldsymbol{w}(t) = -\mathbf{H}^{-1}(t)\,\boldsymbol{g}(t)$
  - problem if $\mathbf{H}(t)$ is singular
- Newton method in 1D: $\boldsymbol{w} = \boldsymbol{w}_0 + \lambda\,\boldsymbol{d} = \boldsymbol{w}_0 + \Delta\boldsymbol{w}$

Weight optimization techniques (ctd)

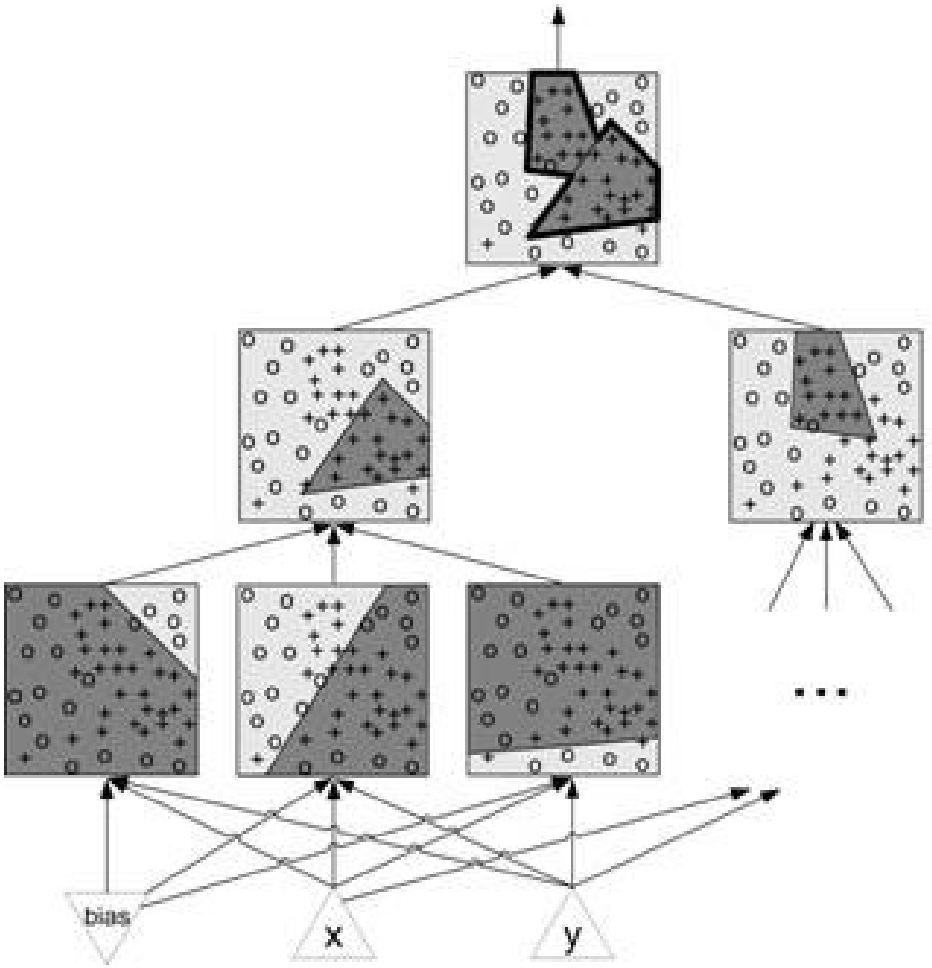
- Quasi-Newton's method: estimates $\mathbf{H}^{-1}$ without matrix inversion, but has complexity $O(W^2)$, suitable only for small-scale problems

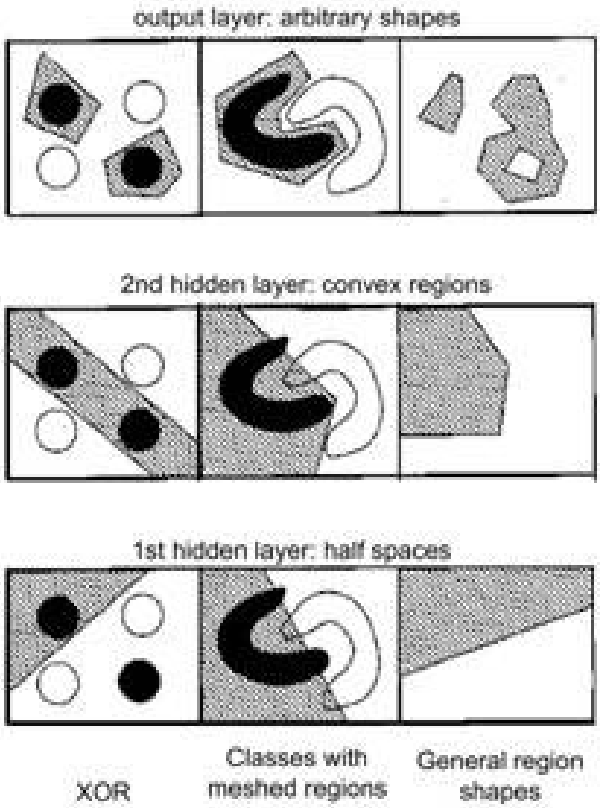
  $\boldsymbol{w}(t) = \boldsymbol{w}(t\text{-}1) - G(t\text{-}1).\nabla E(t\text{-}1), \qquad G \approx H^{-1}$
- Conjugate-gradient method:
  - $\boldsymbol{d}(t) = -\nabla E(t) + \beta.\boldsymbol{d}(t\text{-}1)$ find β such that conjugacy $\boldsymbol{d}(t\text{-}1).\mathrm{H}.\boldsymbol{d}(t) = 0$ holds
  - accelerates convergence of steepest descent
  - avoids computational requirements of Newton's method, which makes it suitable also for large-scale problems
  - Polak-Ribiere rule:

$$\beta = \frac{(\nabla E(t) - \nabla E(t-1)).\nabla E(t)}{(\nabla E(t-1))^2}$$

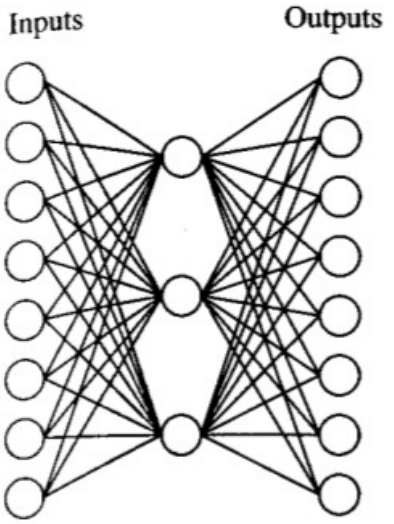
- Scaled conjugate-gradient method

Decision regions in MLP





Example applications

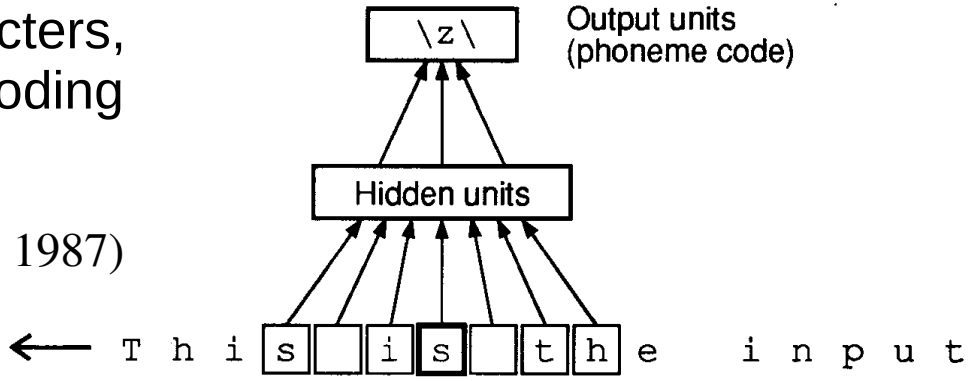


- XOR – nonlinearly separable 2-class problem
- parity problem – generalization of XOR
- encoder – auto-association: *n*-*Q*-*n*
- image compression: (Cottrell et al, 1987)
  - 64-16-64 units, inputs: 0-255 range, trained on 8×8 non-overlapping pixel region of the image, 150.000 iterations
  - self-supervised standard BP
  - non-linearity of hidden units is of no help (Bourland & Kamp, 1988)
  - linear NN performs PCA (Baldi & Hornik, 1989)

Application: Text reading

- NETtalk - NN learns to read English text
- Input 7×29 units encoding 7 characters, 80 hidden and 26 output units encoding phonemes.

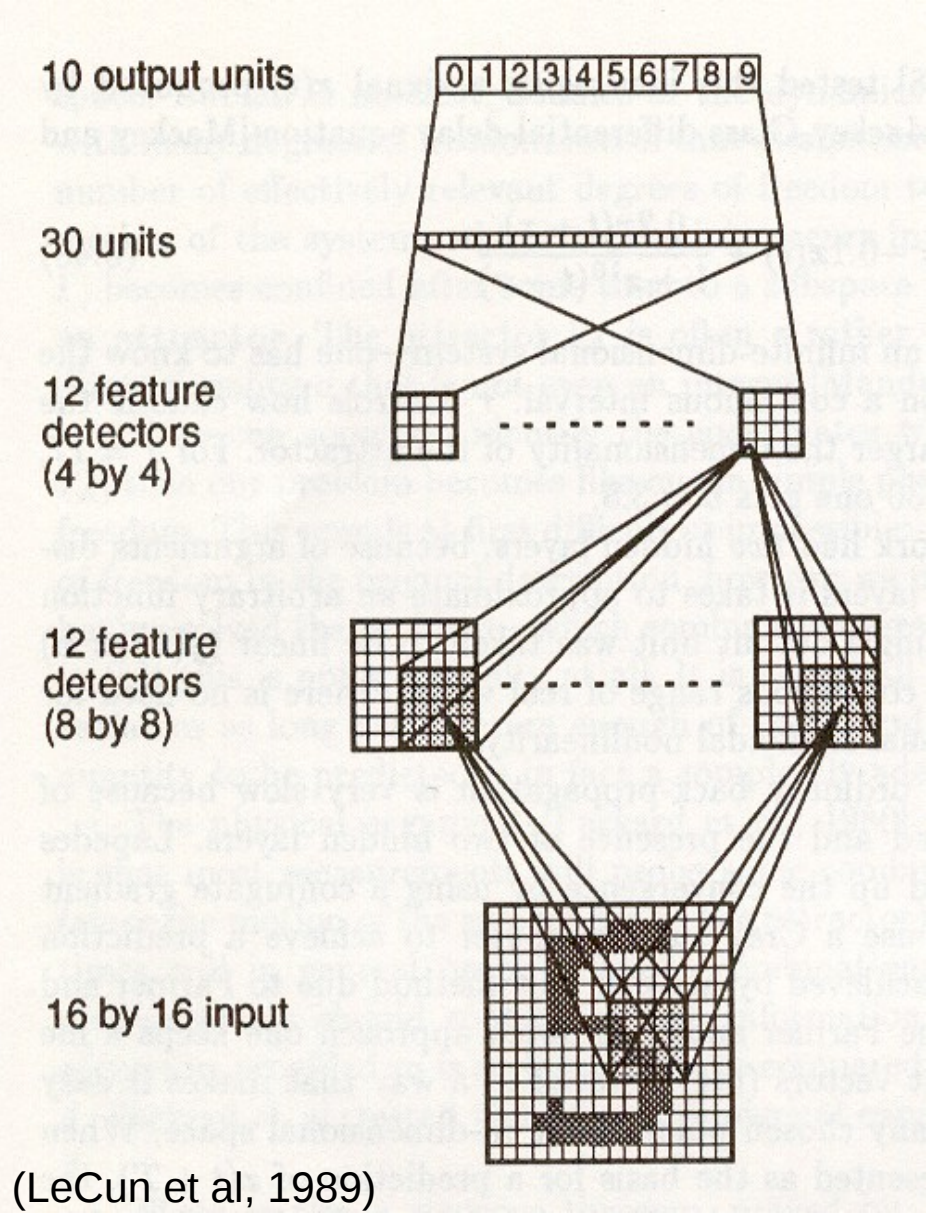
(Sejnowski & Rosenberg, 1987)



- training on 1024 words, intelligible speech after 10 epochs, 95% accuracy after 50 epochs
- NN first learned gross features (e.g. word boundaries) and then gradually refined its discrimination (psycholinguistic plausibility).
- 78% generalization accuracy, quite intelligible speech
- damage in NN (noise or unit removal) => graceful degradation
- meaningful internal representations (e.g. vowel-consonant distinction)
- only slightly worse than (rule-based) DEC-talk, with much lesser design effort involved

Application: Recognizing hand-written ZIP codes

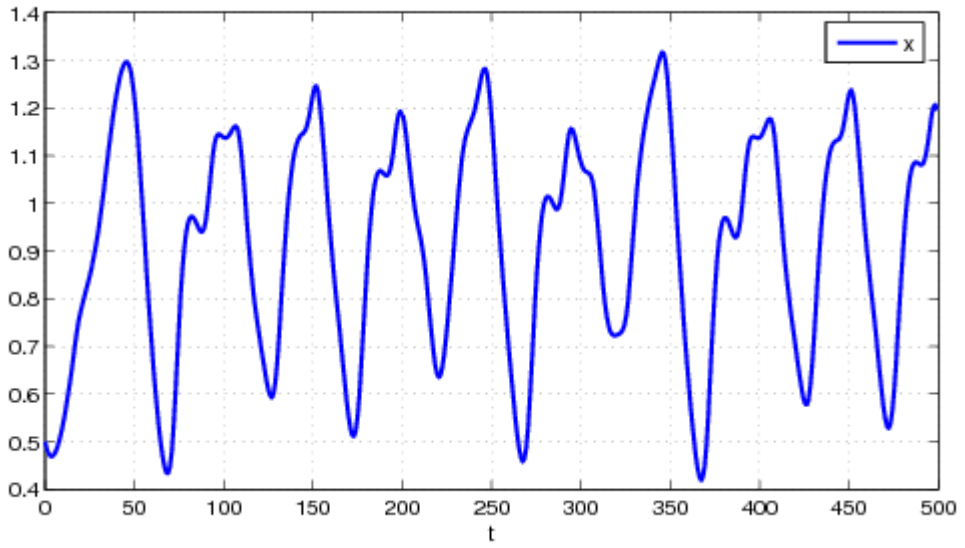
- Input: 16×16 units, (-1,1) range
- 3 hidden layers (HL)
- Reduction of free parameters by weight sharing on HL1: all 64 units in a group had the same 25 weights
- the same principle used in HL2
- 1256 units and 9760 weights
- BP used, accelerated with quasi-Newton rule
- 1% error on train set (7,300 digits), 5% on test set (2,000 digits). Test terror could be reduced to 1% with 12% rejection rate of marginal cases
- “optimal brain damage” - further elimination of weights to reduce test error



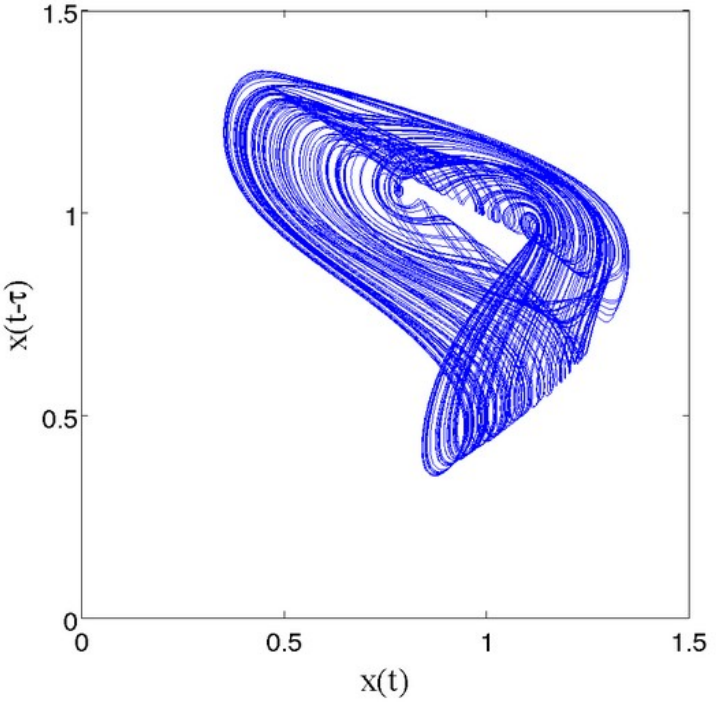
(LeCun et al, 1989)

Application: Signal prediction

- especially difficult case: chaotic time-series
- attractor embedding (Takens theorem) – in state space
- Input: $\boldsymbol{x}(t) = [x(t), x(t\text{-}k), ..., x(t\text{-}(m\text{-}1)k)]$, target $x(t+\Delta)$, $k = \text{int}$.
- Example: Mackey-Glass system

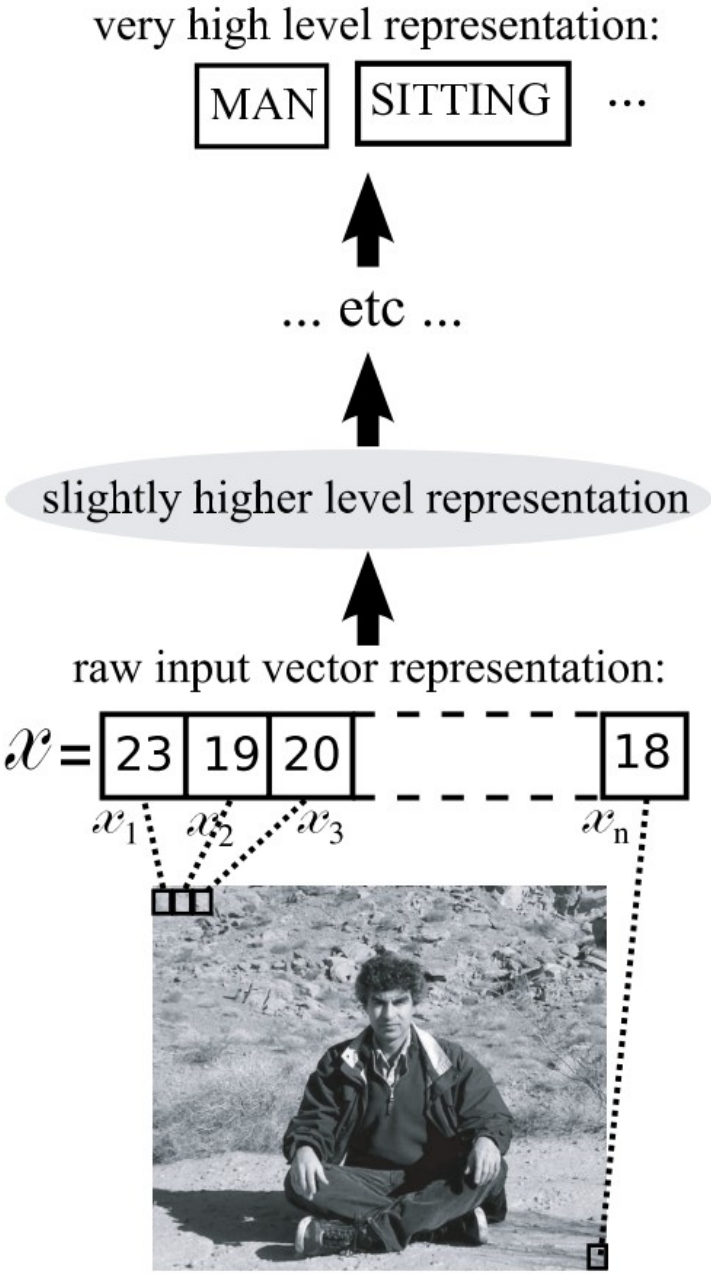


$$\frac{dx}{dt} = \beta \frac{x_\tau}{1 + x_\tau{}^n} - \gamma x, \quad \gamma, \beta, n > 0$$



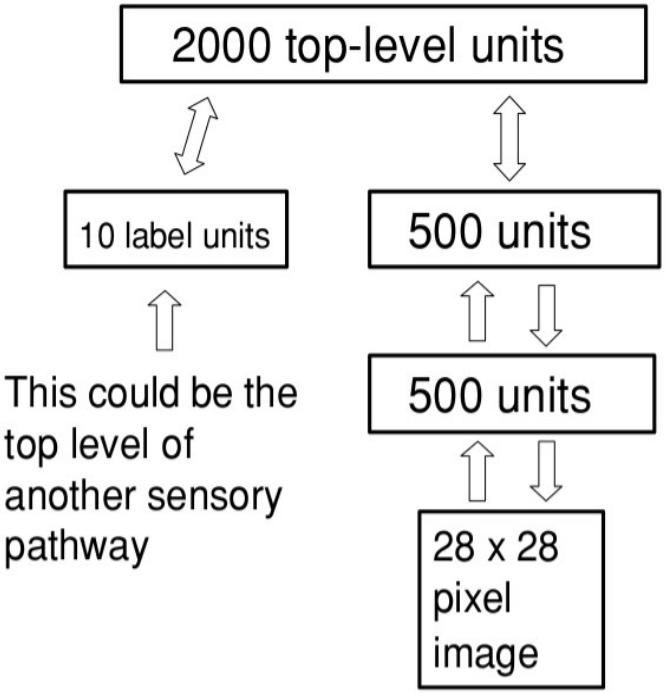
Deep learning

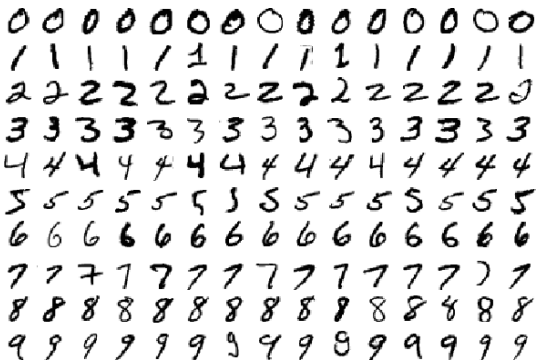
- multi-layer architectures (>3 layers)
- increasing abstractness
- with distributed representations emerging
- current discussion (connectionist ML) about its origins
- breakthrough: Deep Belief Networks (Hinton, Osindero &Teh, 2006)
- unsupervised + supervised learning possible
- biological relevance
- currently top results in various large-data domains: vision (object recognition), language modeling, speech recognition



Benchmark tests

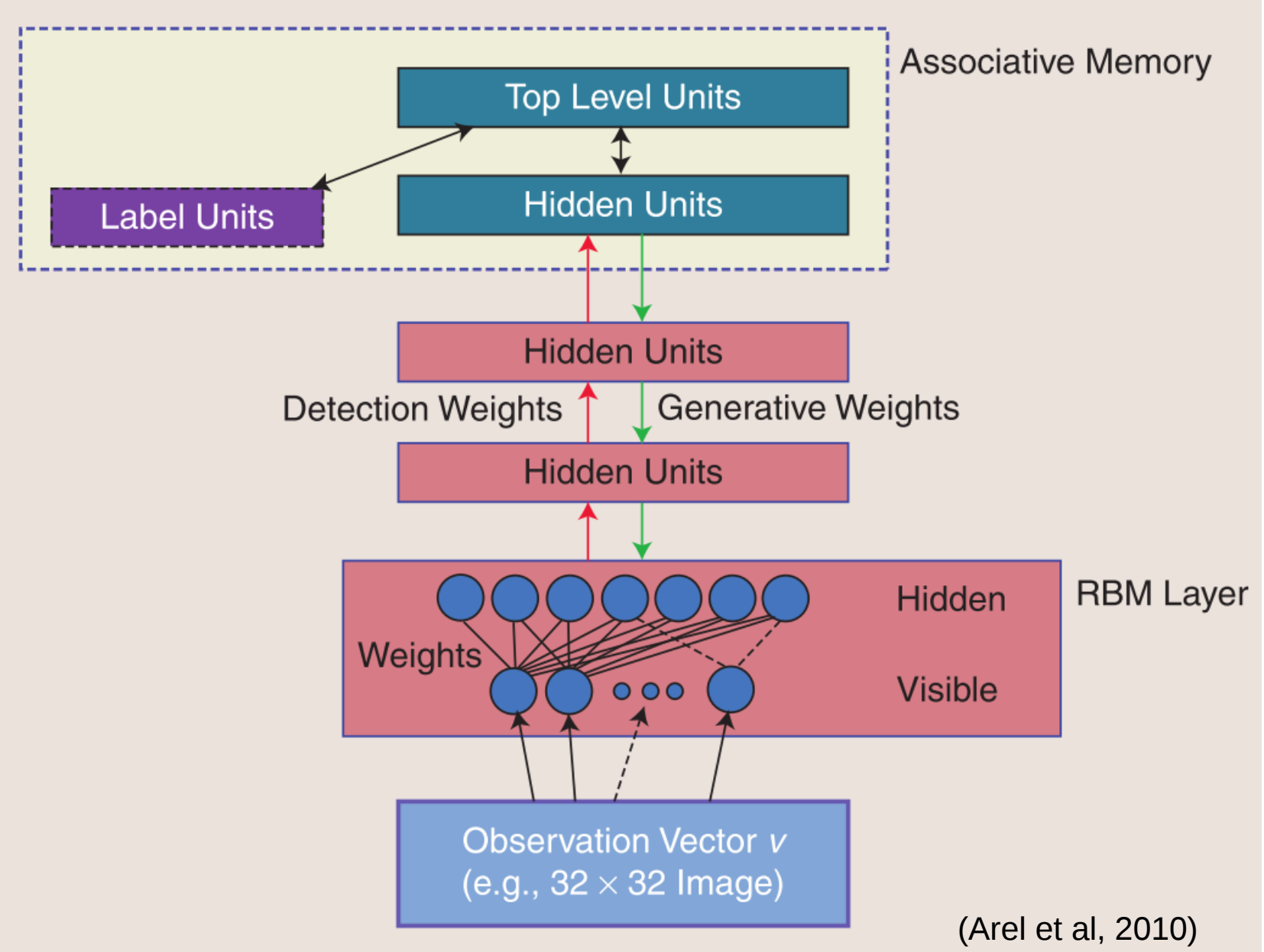
- MNIST database – handwritten digits
- Hinton et al (2006): DBN: layer-wise unsupervised pretraining + learning joint distributions (image-label pairs)
  - also top-down weights, symmetric matrices, 1.25% errors
- LeCun et al (1998): convolutional nets (<1%)
- Ciresan et al. (2012): (0.23%)
  - committee of 35 deep convolutional networks





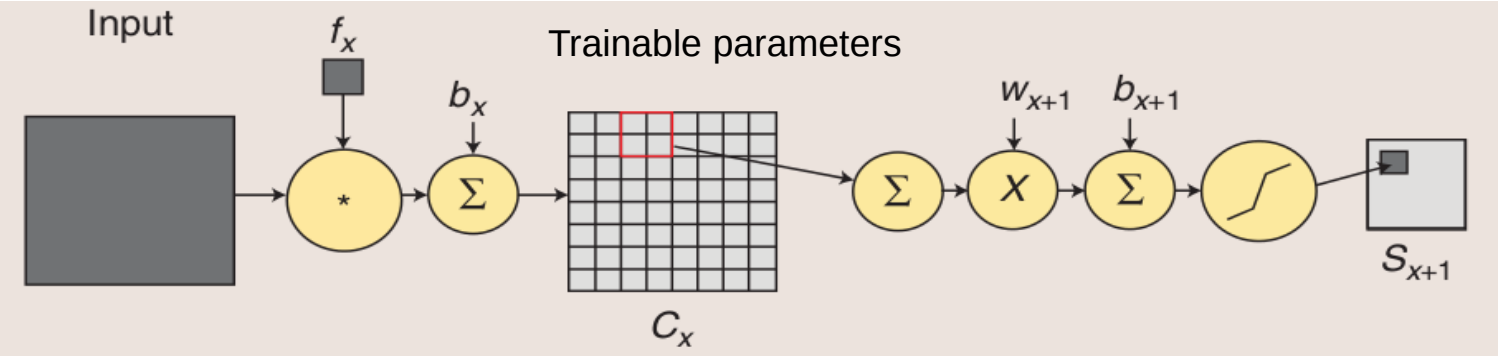
Records: http://yann.lecun.com/exdb/mnist/

Hinton's Deep Belief Network framework

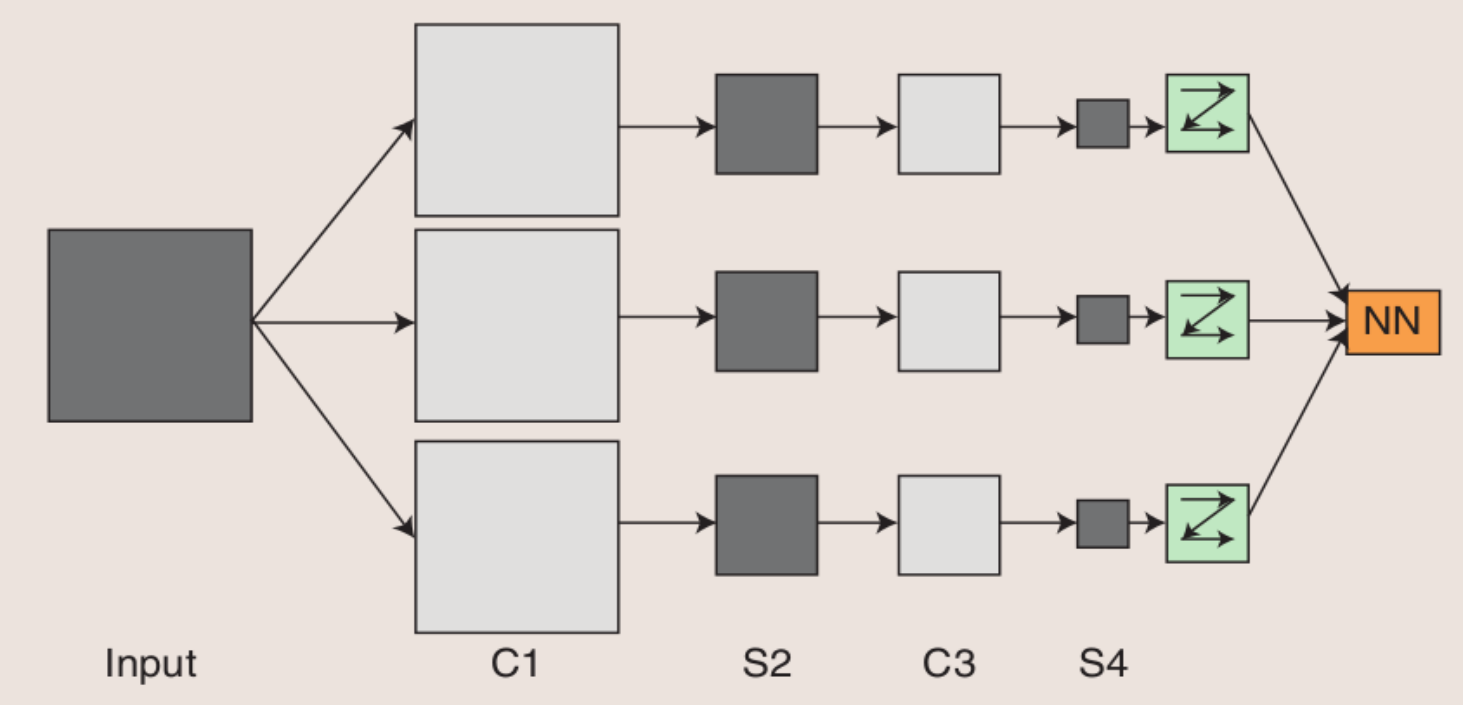


(Arel et al, 2010)

Convolutional Neural Networks



convolution subsampling



(Arel et al, 2010)

Deep nets training

- initialization
  - weights: uniform or (naïve) Gaussian distribution
  - problem of saturation => weight scaling = $f(deg_{in}, deg_{out})$
  - unsupervised pretraining (as in DBN)
- training
  - BP, momentum, Nesterov accelerated gradient (Hinton, 2013)
- activation functions
  - rectified linear: $y = \max(0, net)$, smoothed: $y = \log(1+e^{net})$, ...
- regularization
  - L1, L2, dropout, …
- input data set enlargement – by elastic deformations

Summary

- BP = standard supervised learning algorithm for MLPs in classification and/or regression tasks
- nonlinearity is essential, but for classification the softmax outputs usually perform better
- computationally efficient, complexity $O(W)$
- sequential mode preferential
- lacks biological plausibility
- finds locally optimal solutions, modifications may be of help (momentum, pruning)
- alternative, second-order optimization techniques provide superior solutions
- Modern stream: deep architectures

5

Principal component analysis

Self-organization

- Learning of a neural network without supervision (teacher), using only the environment
- Self-organizing neural networks (SONN) are biologically more plausible
- Architectures of SONN:
  - Single-layer with feedforward connections
  - Single-layer with feedforward and lateral connections
  - multi-layer with feedforward connections between layers

Self-organization (ctd)

"Global order can arise from local interactions." (Turing, 1952)

- Neural network learns without a teacher

Organization in a NN takes place at two interacting levels:

- Neuron activations – changes of unit activations
- Network connectivity - changes of synaptic connections

Principles of self-organization (von der Malsburg, 1990):

1. Self-amplification – weight modification tend to self-amplify
2. Competition - among neurons, due to limited resources
3. Cooperation - among neighboring neurons
4. Structural information - in input data is acquired as knowledge

Applications of self-organizing NNs

- SONNs are better at dealing with scaling problem, because their learning rules are simpler and are mostly local
- Types of tasks:
  - PCA – optimal linear encoding with respect to mean squared reconstruction error
  - Data clustering (vector quantization)
  - Topological feature mapping of input data
  - Data coding with maximum information preservation

Principal Component Analysis

- PCA (Pearson, 1901) – linear transformation into **feature space** – reduction of data dimensionality
- Assume n-dim. random (column) vector $\boldsymbol{x}$, with zero mean value, $\langle \boldsymbol{x} \rangle = \mathbf{0}$. Let $\boldsymbol{u}$: $\|\boldsymbol{u}\| = (\boldsymbol{u}^{\mathrm{T}}\boldsymbol{u})^{1/2} = 1$, with $\dim(\boldsymbol{u}) = n$.
- We project $\boldsymbol{x}$ onto $\boldsymbol{u}$, getting the projection $a = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{x}^{T}\boldsymbol{u}$.
- Since $\langle \boldsymbol{x} \rangle = \mathbf{0}$, then $\langle a \rangle = \boldsymbol{u}^{\mathrm{T}} \langle \boldsymbol{x} \rangle = 0$.
- Variance σ of a is then

$$\sigma^2 = \langle a^2 \rangle = \langle (\boldsymbol{u}^T \boldsymbol{x})(\boldsymbol{x}^T \boldsymbol{u}) \rangle = \boldsymbol{u}^T \langle \boldsymbol{x}\boldsymbol{x}^T \rangle \boldsymbol{u} = \boldsymbol{u}^T \mathbf{R}\, \boldsymbol{u}$$

  where $\mathbf{R}[n \times n]$ is the correlation matrix of input data.

- Then $\sigma^2 = \psi(\mathbf{u}) = (\mathbf{u}^T \mathbf{R}\, \mathbf{u})$

Structure of PCA eigenvalues

- It can be shown that $\boldsymbol{u}$ for which $\psi(\boldsymbol{u})$ reaches maximum satisfies condition $\mathbf{R}\boldsymbol{u} = \lambda\boldsymbol{u}$.
- For $\boldsymbol{u} \neq \mathbf{0}$ there exist nontrivial solutions, λ are **eigenvalues**, associated with eigenvectors $\boldsymbol{u}$ of matrix $\mathbf{R}$.
- Let us denote eigenvalues of $\mathbf{R}[n \times n]$ as $\lambda_1, \lambda_2, \ldots, \lambda_n$ and eigenvectors as $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n$. Then $\mathbf{R}\boldsymbol{u}_i = \lambda_i\boldsymbol{u}_i$, $i = 1,2,\ldots, n$
- Let $\lambda_1 > \lambda_2 > \ldots, > \lambda_n$, i.e. $\lambda_1 = \lambda_{\max}$. If we construct matrix $\mathbf{U}$ such that $\mathbf{U} = [\boldsymbol{u}_1\ \boldsymbol{u}_2 \ldots \boldsymbol{u}_n]$, then in matrix form we get

$$\mathbf{RU} = \mathbf{U}\Lambda$$

where Λ is the diagonal matrix $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$.

PCA eigenvalues (ctd)

- $\mathbf{U}$ is an orthogonal matrix, because

$$\boldsymbol{u}_i^{\mathrm{T}} \boldsymbol{u}_j = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases}$$

- Equivalent form is $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$, with $\mathbf{I}$ being an identity matrix, from which $\mathbf{U}^{\mathrm{T}} = \mathbf{U}^{-1}$.
- Then we can write $\mathbf{U}^{\mathrm{T}}\mathbf{R}\mathbf{U} = \mathbf{\Lambda}$, or for each eigen-pair

$$\boldsymbol{u}_i^{\mathrm{T}} \mathbf{R} \boldsymbol{u}_j = \begin{cases} \lambda_j, & j = i \\ 0, & j \neq i \end{cases}$$

from where $\psi(\boldsymbol{u}_j) = \lambda_j$, $j = 1, 2, \ldots, n$

Principal components

- For n different eigenvectors $\boldsymbol{u}_i$ we get different projections (*analysis*) that we call **principal components**:

$$a_j = \boldsymbol{u}_j^{\mathrm{T}} \boldsymbol{x} = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{u}_j \qquad j = 1,2,\ldots,n$$

- During reconstruction of $\boldsymbol{x}$ from projections a_j we combine these into the vector $\boldsymbol{a} = [a_1, a_2, \ldots, a_n]^T$ and then

$$\boldsymbol{a} = \left[\boldsymbol{u}_1^{\mathrm{T}} \boldsymbol{x}, \boldsymbol{u}_2^{\mathrm{T}} \boldsymbol{x}, \ldots, \boldsymbol{u}_n^{\mathrm{T}} \boldsymbol{x} \right]^T = \boldsymbol{U}^{\mathrm{T}} \boldsymbol{x}$$

- Reconstruction of $\boldsymbol{x}$ (synthesis):

$$\boldsymbol{x} = \mathbf{U}\, \boldsymbol{a} = \sum_{j=1}^{n} a_j \boldsymbol{u}_j$$

- PCA is a coordinate transformation of $\boldsymbol{x}$ into the feature space (point $\boldsymbol{a}$).

Dimensionality reduction

- Let $\lambda_1 > \lambda_2 > \ldots, > \lambda_p$ denote the p largest eigenvalues of $\mathbf{R}$. Reconstruction of $\boldsymbol{x}$ is then computed as:

$$\mathbf{x}' = \mathbf{U}\mathbf{a} = \sum_{j=1}^{p} a_j \mathbf{u}_j$$

- Reconstruction error:

$$\mathbf{e} = \mathbf{x} - \mathbf{x}' = \sum_{j=p+1}^{n} a_j \mathbf{u}_j$$

- It holds that:

$$\boldsymbol{e}^{\mathbf{T}} \boldsymbol{x}' = \left(\sum_{i=p+1}^{n} a_i \boldsymbol{u}_i^{\mathbf{T}}\right)\left(\sum_{j=1}^{p} a_j \boldsymbol{u}_j\right) = \sum_{i=p+1}^{n} \sum_{j=1}^{p} a_i a_j \boldsymbol{u}_i^{\mathbf{T}} \boldsymbol{u}_j = 0$$

Approximation + error

- The overall variance of all components

$$\sum_{i=1}^{n} \sigma_i^2 = \sum_{i=1}^{n} \lambda_i^2$$

where σ_i^2 is the variance of *i*-th component a_i

- Analogically, the overall variance of approximation $\boldsymbol{x}'$ with p components is

$$\sum_{i=1}^{p} \sigma_i^2 = \sum_{i=1}^{p} \lambda_i^2$$

- The remaining components $p+1, \ldots, n$ contribute to the reconstruction error.

Hebbian learning – single neuron

- Canadian psychologist Donald Hebb (1949) postulated:

"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."

- Let us assume one linear neuron with n inputs:

$$y = \sum_{j=1}^{n} w_j x_j = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}$$

- According to Hebb's postulate:

$$w_j(t+1) = w_j(t) + \alpha\, y\, x_j \quad \text{for} \quad j = 1,2,\ldots,n$$

where α is the learning rate.

Oja's rule

- The problem of weight divergence could be solved by explicit (L2-)normalization of weight vector (2nd principle of self-organization) (Oja, 1982):

$$w_j(t+1) = \frac{w_j(t) + \alpha\, y\, x_j}{\left\| \boldsymbol{w}(t) + \alpha\, y\, \boldsymbol{x} \right\|}$$

- For small α the rule can be rewritten with Taylor expansion as

$$w_j(t+1) = w_j(t) + \alpha\, y\left(x_j - y\, w_j\right) + O\left(\alpha^2\right)$$

- For small α we get Oja's (1982) rule:

$$w_j(t+1) = w_j(t) + \alpha\, y\, x_j - y^2 w_j(t) \qquad (*)$$

Hebbian term

Stabilizing term

- Oja's rule is opposite to delta rule (in terms of input-output)

Single-unit behavior

Properties of linear neuron trained by Oja's rule (*) with $\langle \mathbf{x} \rangle = \mathbf{0}$:

- Its weight vector $\boldsymbol{w}$ converges to match the eigenvector of $\mathbf{R} = \langle \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}} \rangle$
- After convergence, $\boldsymbol{w}$ maximizes $\langle y^2 \rangle$.
- After convergence, $\|\boldsymbol{w}\| = 1$.

- Hence, the linear Hebbian neuron projects input vectors to the direction that maximizes the discrimination capability (maximum information preservation).

GHA algorithm

- Assume single-layer NN with n inputs and p outputs. The Generalized Hebbian algorithm (Sanger, 1989) for updating weights has the form:

$$\Delta w_{ij} = y_i \left(x_j - \sum_{k=1}^{i} y_k w_{kj} \right)$$

- GHA extracts p principal components of the input data correlation matrix. Its advantages over classical PCA: (1) not necessary to compute $\mathbf{R}$, (2) computationally less demanding especially when $p \ll n$.
- Properties:
  - finds principal components, ordered according to magnitude
  - results are reproducible, up to the signs

Oja's algorithm

- Applies also to single-layer feedforward NN with p outputs:

$$\Delta w_{ij} = y_i \left(x_j - \sum_{k=1}^{p} y_k w_{kj} \right) \quad i = 1,2,\ldots,p$$

Properties:

- Does not extract principal components but only **principal subspace** (generated by them)
- The unit outputs are not ordered, their variance is roughly the same.
- Results are not reproducible; they depend on initial conditions and on the order of input presentation.
- Useful in applications when evenly distributed representations are needed.

APEX algorithm

- Adaptive Principal Component Extraction (Kung & Diamantaras, 1990) applies to single-layer NN with lateral connections. The *i*-th unit has the lateral weight vector:

$$\mathbf{u}_i = [u_{i1}, u_{i2}, \ldots, u_{i,i-1}]^T$$

  and the output:

$$y_i = \boldsymbol{w}_i^T \boldsymbol{x} + \boldsymbol{u}_i^T \boldsymbol{y}_{i-1} \quad \text{where} \quad \boldsymbol{y}_{i-1} = [y_1, y_2, \ldots, y_{i-1}]^T$$

- Feedforward weights are modified according to Oja's rule, lateral weights are trained by anti-Hebbian rule (inhibition):

$$\Delta u_{ik} = -\alpha y_i (y_k + y_i u_{ik})$$

- Feedforward weight vectors converge to eigenvectors of $\mathbf{R}$, whereas lateral weights to zero vectors.

Two classes of PCA algorithms

- Reestimation algorithms (e.g. GHA) – with feedforward connections only, trained by Hebb-like learning
- Decorrelation algorithms (e.g. APEX) – also with lateral connections, employ anti-Hebbian rule (to decorrelate outputs)

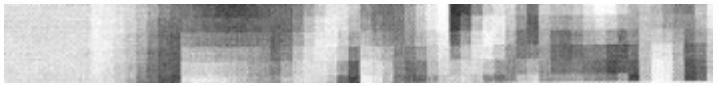
- Alternative: *n*-*p*-*n* network, with $n > p$, trained as an autoassociator (i.e. targets = inputs) by supervised back-propagation. After learning, hidden neurons extract PCA (e.g. Baldi & Hornik, 1989).

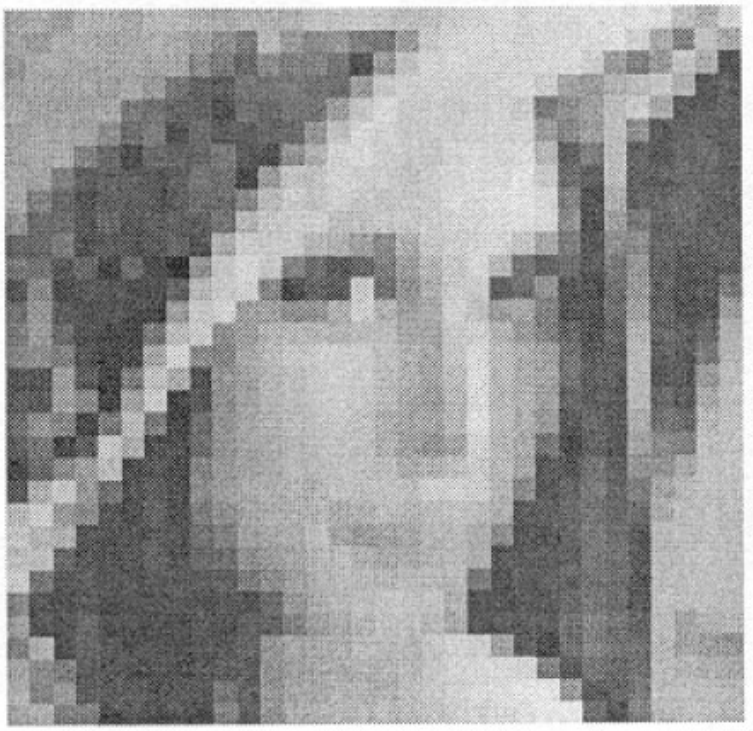
The use of PCA algorithms

- PCA is suitable for preprocessing the data that has approximately Gaussian distribution.
- PCA is equivalent to maximization of output information in case of Gaussian data (Linsker, 1988).
- Features of PCA-preprocessing:
  - decorrelation of input components
  - decreased data dimension (# components)
  - extraction of linear features (principal components)
- e.g. PCA-preprocessed data speed up back-propagation learning.

Example of PCA application

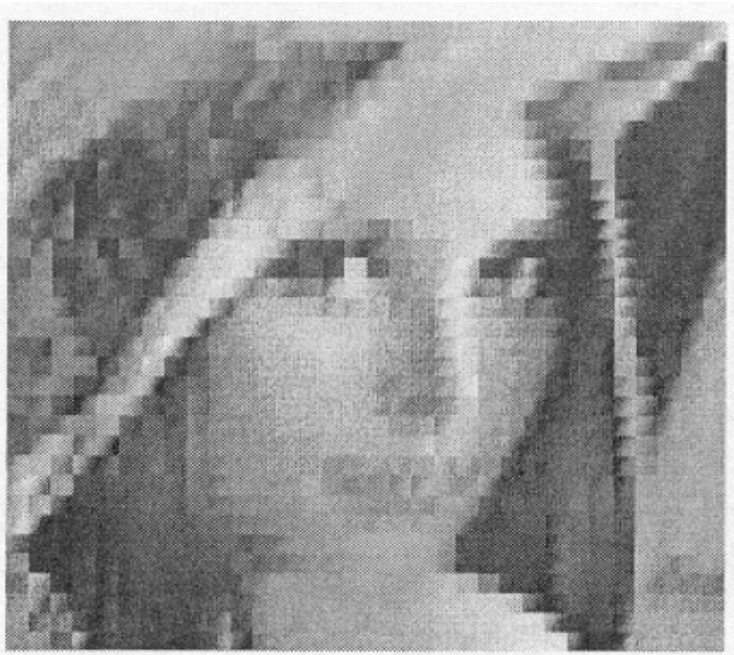
- Image of 256x256 pixels, converted to gray image (0...255)
- NN trained on subimages 8x8
- Eight largest eigenvectors:





Reconst-
ruction
with $p = 1$

GHA



a) *2 koeficienty (0,156 bit/bod)*

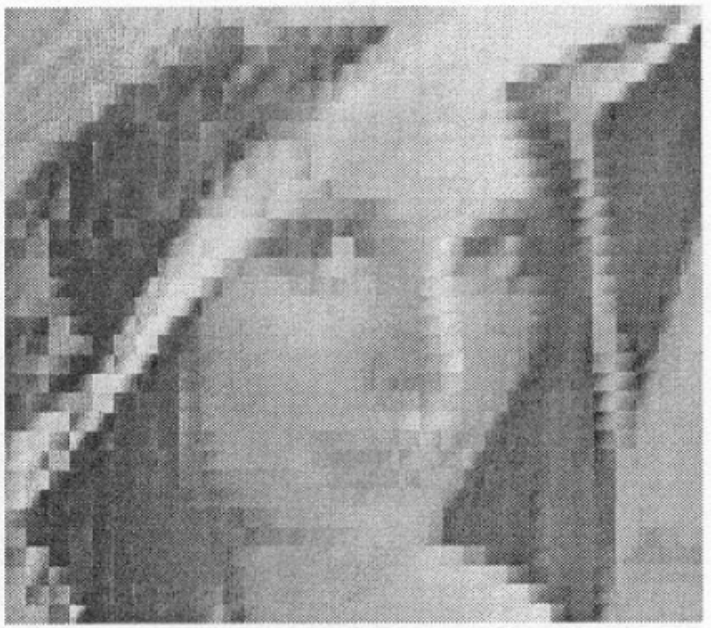
b) *4 koeficienty (0,313 bit/bod)*

c) *6 koeficientov (0,469 bit/bod)*

d) *8 koeficientov (0,625 bit/bod)*

(Courtesy of Oravec et al, 1998)

APEX



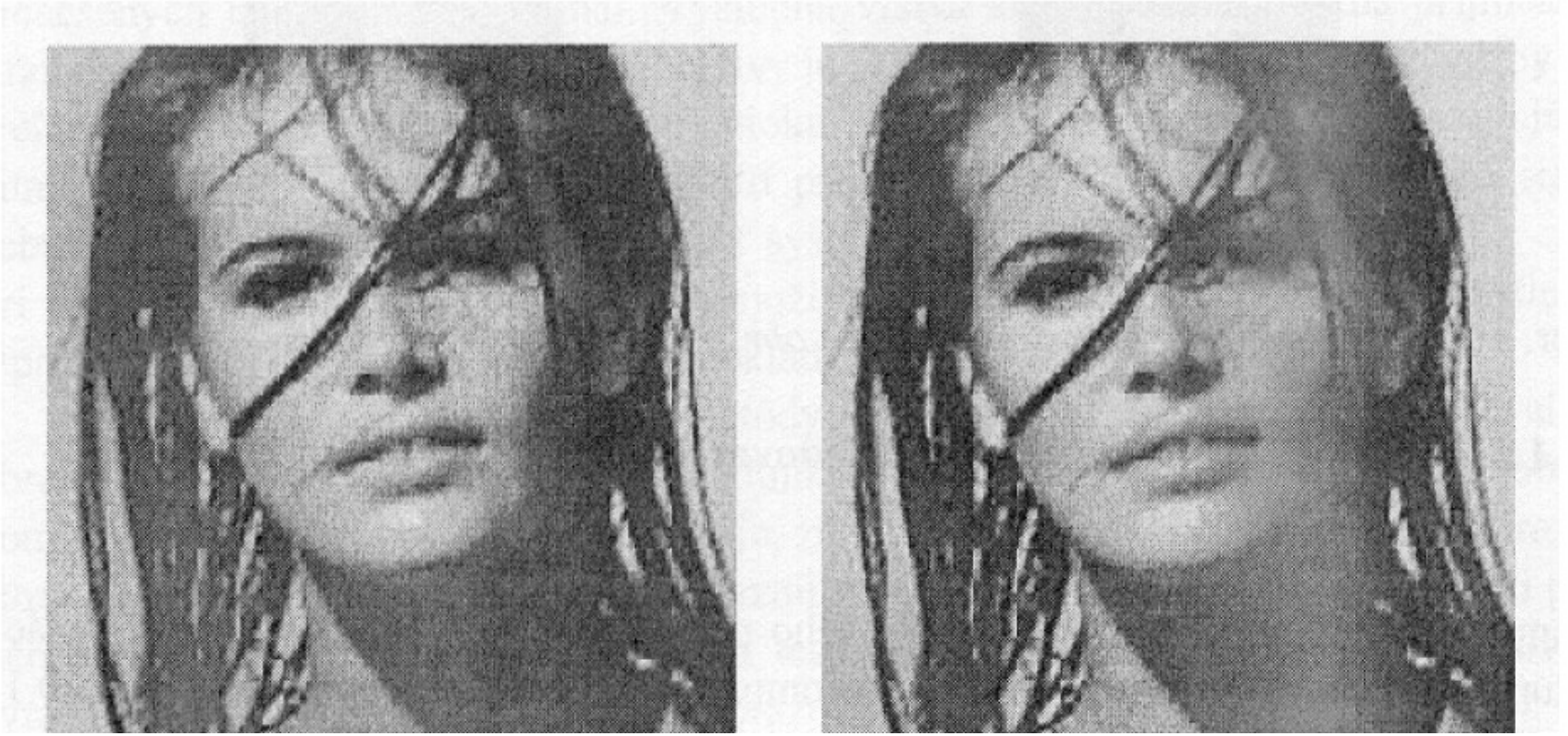
a) *2 koeficienty (0,156 bit/bod)*

b) *4 koeficienty (0,313 bit/bod)*

c) *6 koeficientov (0,469 bit/bod)*

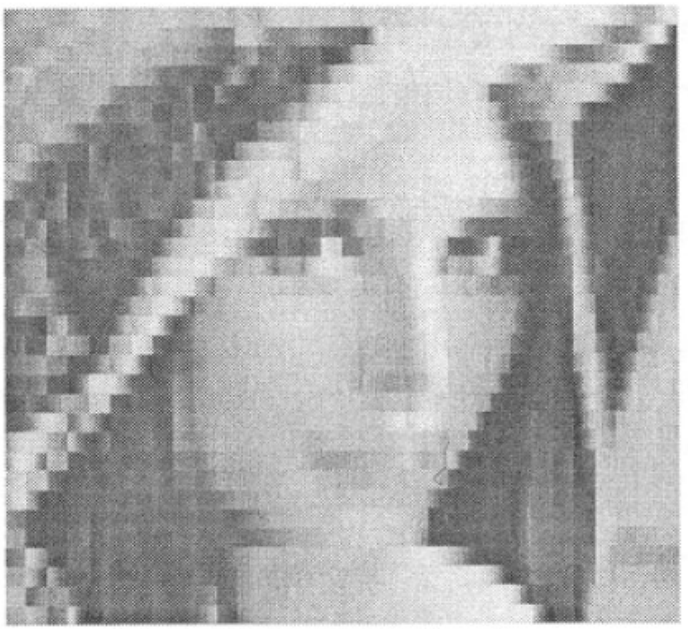
d) *8 koeficientov (0,625 bit/bod)*

APEX - generalization



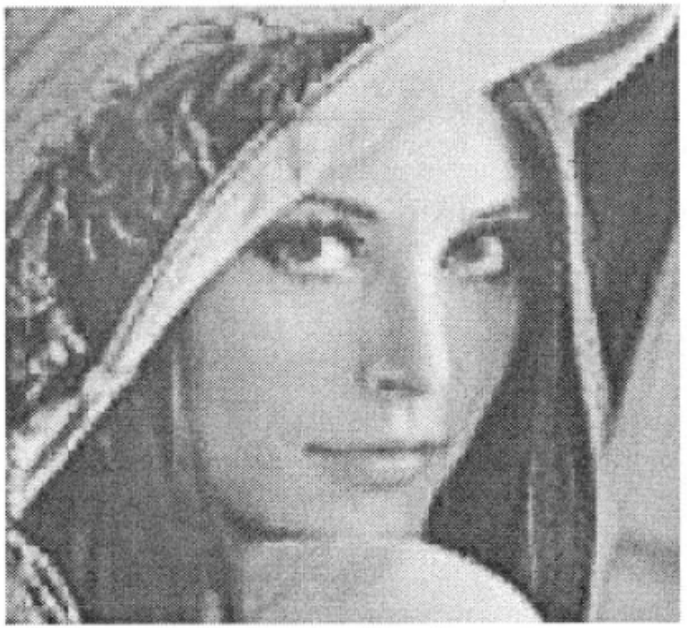
MLP

Performs equally well

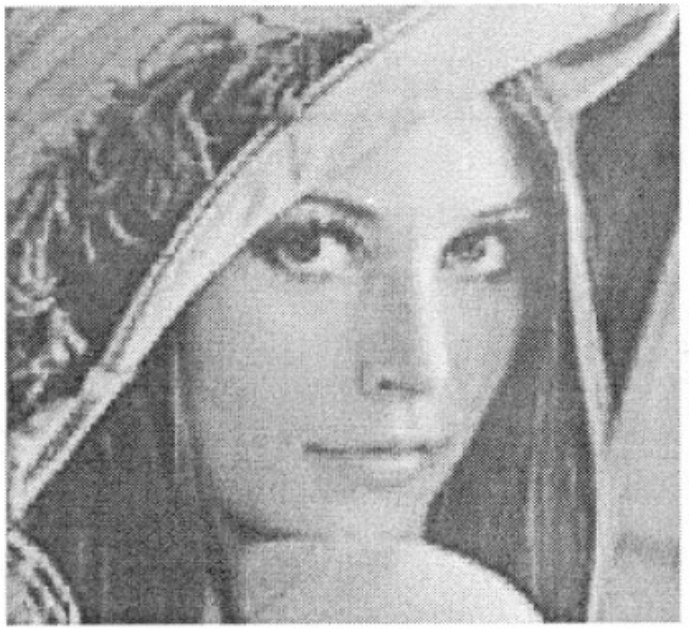


a) *64-2-64 (0,25 bit/bod)*

b) *64-6-64 (0,75 bit/bod)*



c) *64-8-64 (1 bit/bod)*



d) *64-12-64 (1,5 bit/bod)*

Summary

- Principal component analysis – standard method for linear reduction of data dimensionality
- Suitable for data with gaussian distribution (i.e. leading to minimal reconstruction error)
- PCA projects onto orthogonal directions with maximal variance
- A single-layer neural net trained by Hebbian-like learning rule can implement PCA
- No need to calculate autocorrelation matrix of input data

6

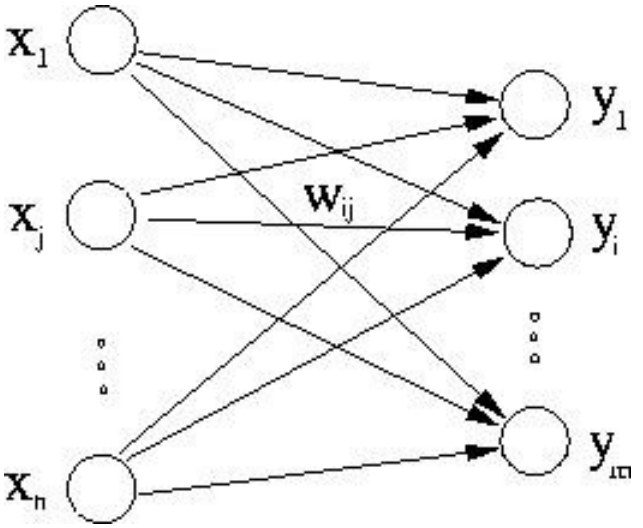
Self-organizing maps

Simple competitive learning

- a kind of unsupervised learning
- Features:
  - linear neurons
  - winner: $y_c = \max_i \{\boldsymbol{w}_i^T . \boldsymbol{x}\}$
    - i.e. best matching unit *c*
  - winner-take-all adaptation:

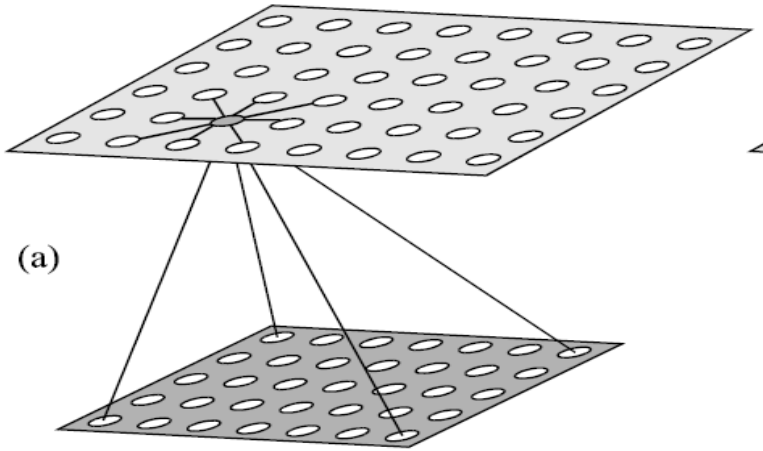
    $\Delta \boldsymbol{w}_c = \alpha(\boldsymbol{x} - \boldsymbol{w}_c) \quad \alpha \in (0,1)$

    $\|\boldsymbol{w}_c\| \leftarrow 1$
  - risk of “dead” neurons
- algorithm: in each iteration
  - find winner, adapt its weights
- useful for clustering

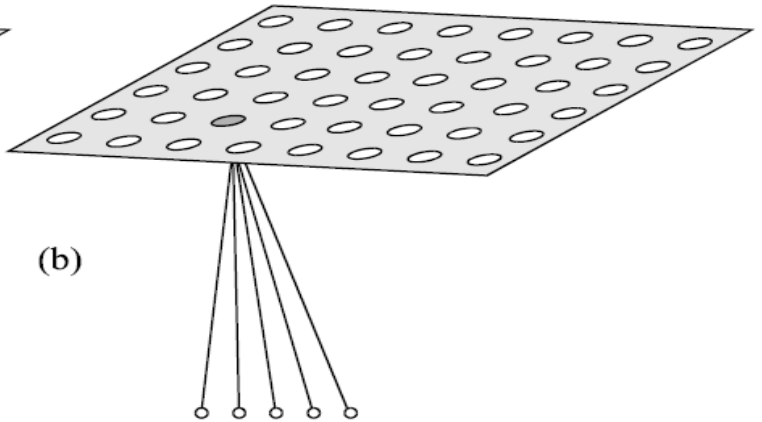


Feature mapping

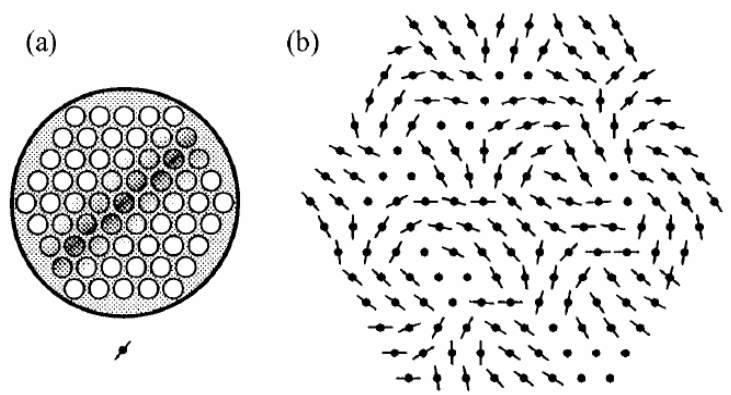
biologically motivated models



Self-organizing map (SOM)



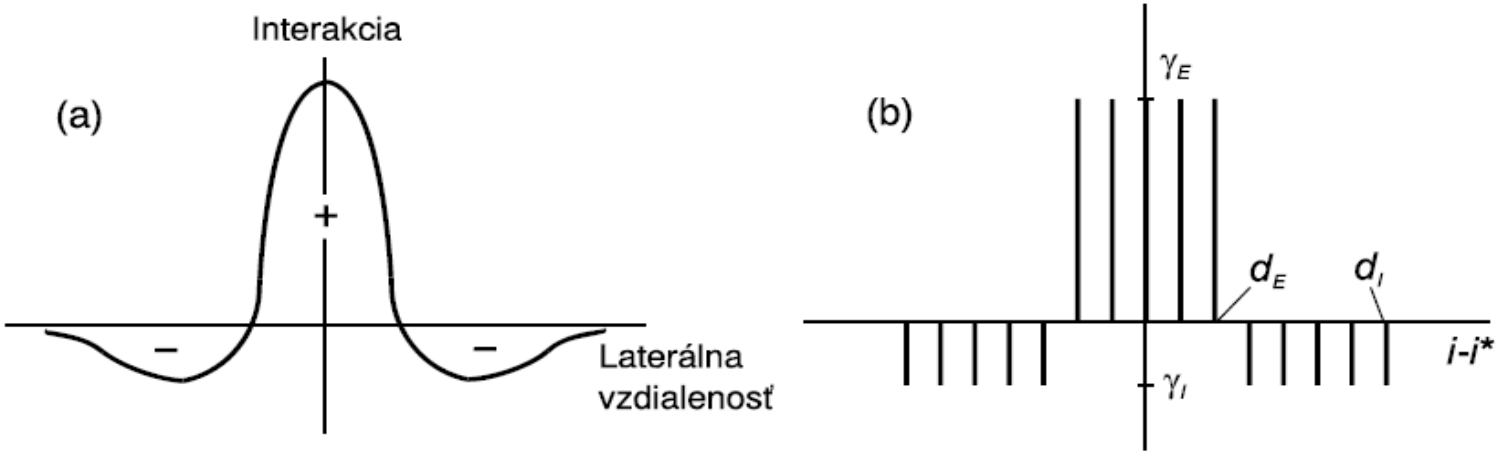
e.g. mapping from retina to cortex -> orientation map



- introduced topology of neurons in the map =>
- winner-take-most due to neuron cooperation

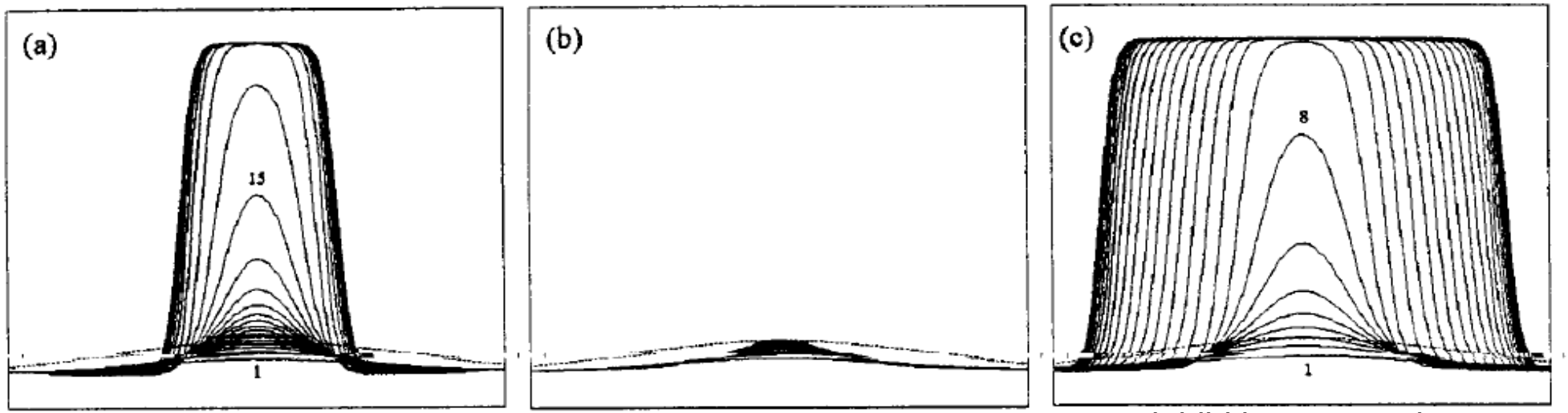
Lateral interactions in the map

Mexican hat function
(1D case)



$$y_i(t+1) = s(z_i + \sum_{k=-K}^{K} l_{ik} . y_{i+k}(t))$$

initial response $z_i = \boldsymbol{w}_i^T . \boldsymbol{x}$



inhibition too strong

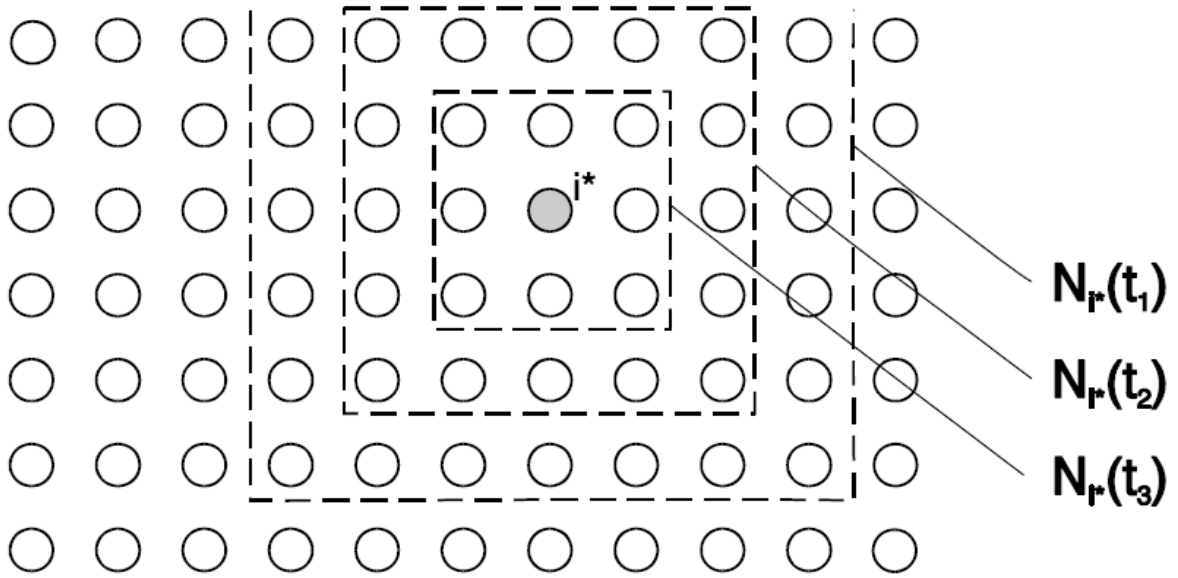
inhibition too weak

(Kvasnička et al, 1997)

Neighborhood function in SOM

- computationally efficient substitute for lateral interactions
- neurons adapt only within the winner neighborhood
- neighborhood radius decreases in time

- rectangular neighborhood (below)



- alternative: gaussian neighborhood

$$h(i^*, i) = \exp\{-\frac{d_E^2(i^*,i)}{\lambda^2(t)}\}$$

$$\lambda(t) = \lambda_i.(\lambda_f/\lambda_i)^{t/t_{max}}$$

SOM algorithm

(Kohonen, 1982)

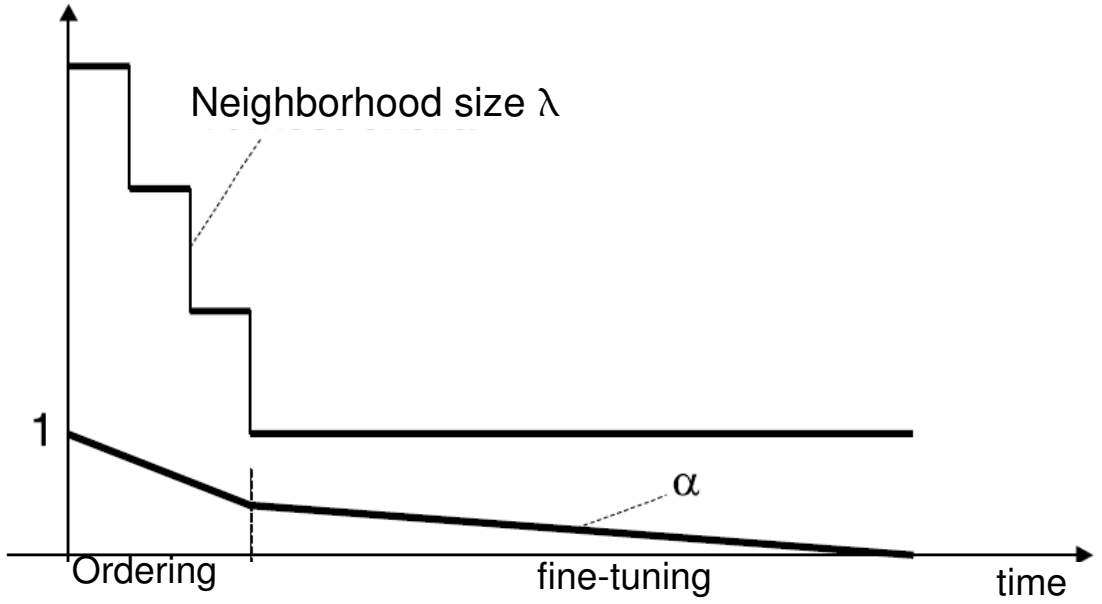
- randomly choose an input $\boldsymbol{x}$
- find winner i^* for $\boldsymbol{x}$

$$i^* = \arg\min_i \|\boldsymbol{x} - \boldsymbol{w}_i\|$$

- adapt weights within the neighborhood

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \alpha(t)\, h(i^*, i)[\boldsymbol{x}(t) - \boldsymbol{w}_i(t)]$$

- update SOM parameters (neighborhood, learning rate)
- repeat until stopping criterion is met



derived from general Hebbian form:

$$\Delta \boldsymbol{w}_i = \alpha\, y_i \boldsymbol{x} - g(y_i)\, \boldsymbol{w}_i$$

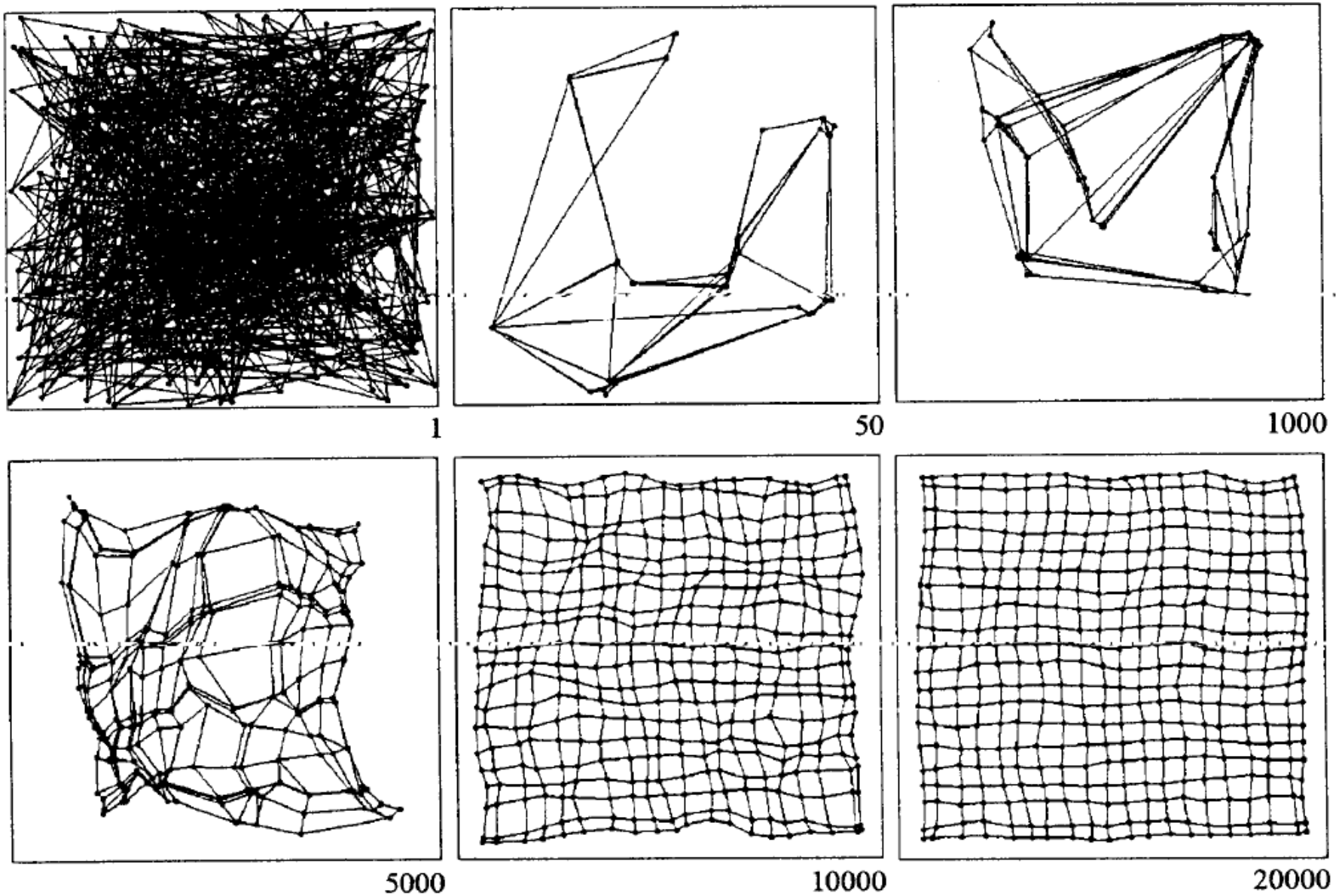
Input-output mapping:

$\mathbf{X} \rightarrow \{1,2,...,m\}$ or

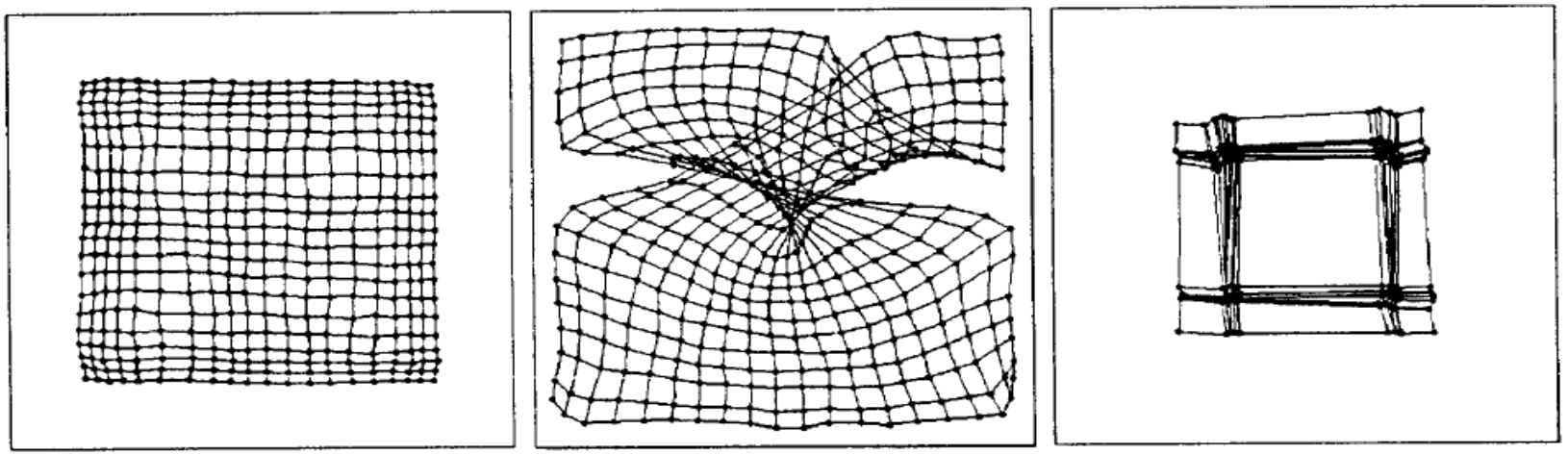
$\mathbf{X} \rightarrow \mathbf{Y}$ $\quad \boldsymbol{y} = [y_1, y_2, ..., y_m]$

where e.g. $y_i = \exp(-\|\boldsymbol{x} - \boldsymbol{w}_i\|)$

Example: 2D inputs, 20x20 neurons



Special effects

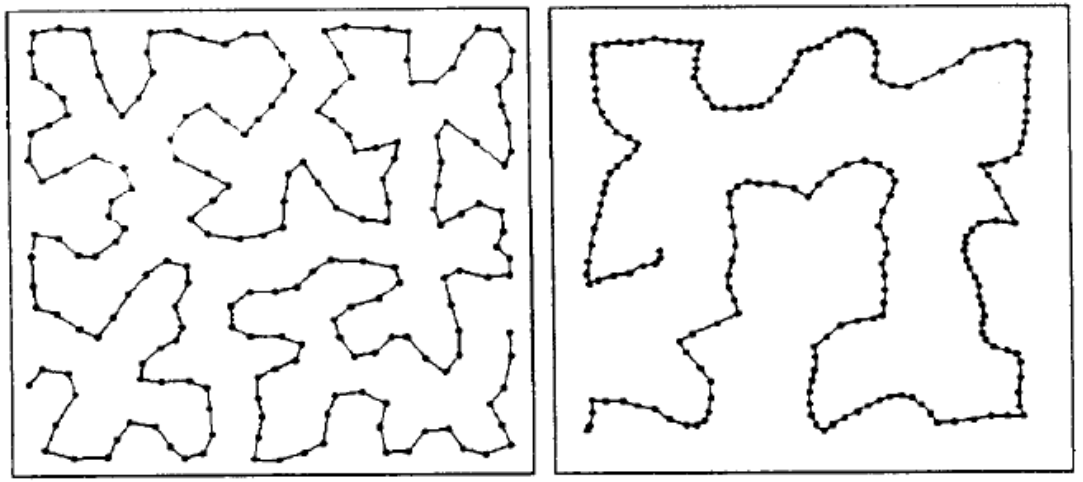


too fast $\alpha\downarrow$ too fast $\lambda\downarrow$ too slow $\lambda\downarrow$

Neighborhood effect

2D data,
1D map

rectangular

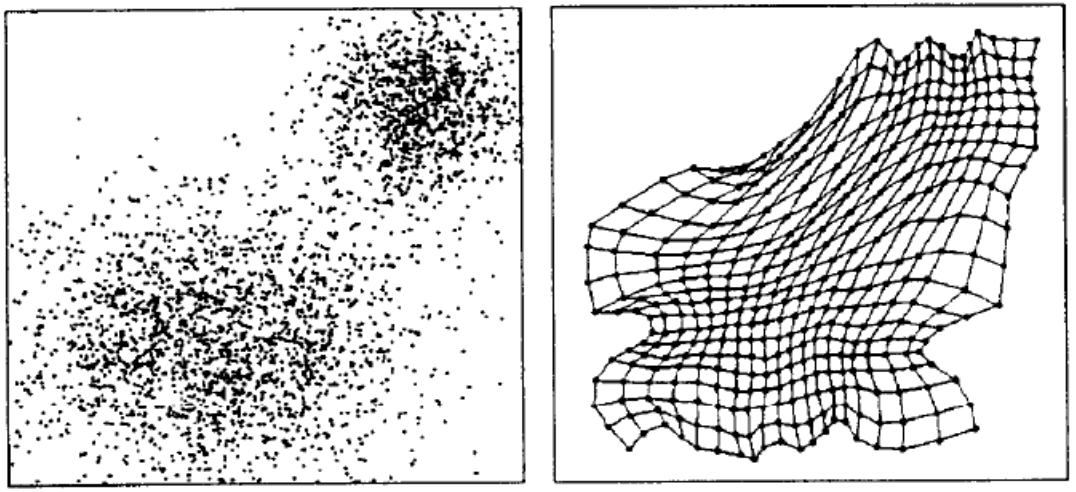


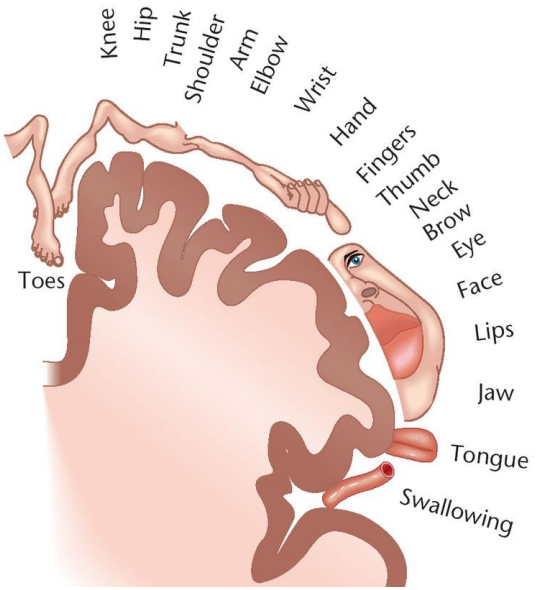
Gaussian

Magnification property

- SOM tends to approximate input distribution

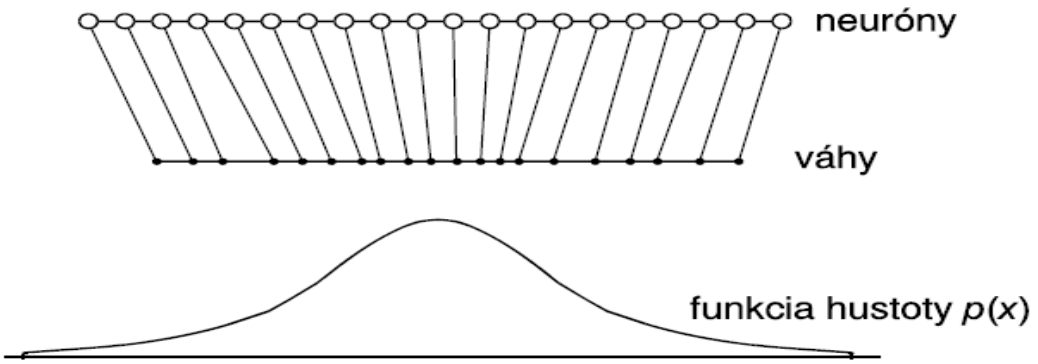
2D:

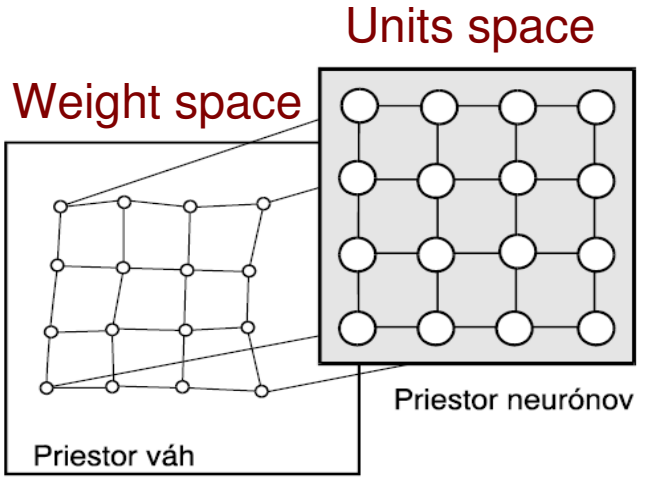




Somatosensory homunculus in the brain

1D:





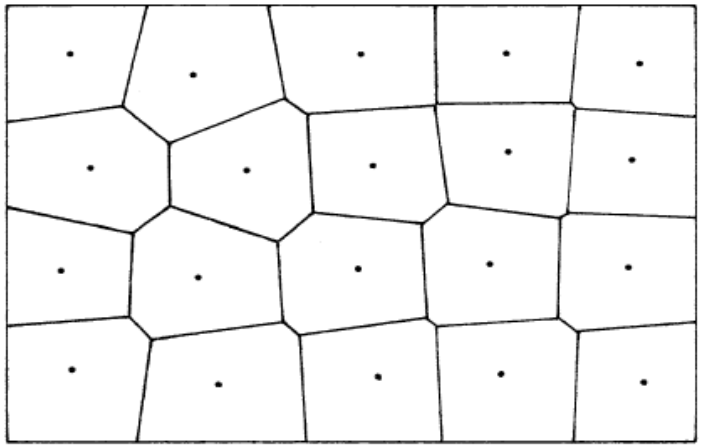
Theory for 1D: $\propto P(x)^{2/3}$

SOM performs 2 tasks simultaneously

Vector quantization

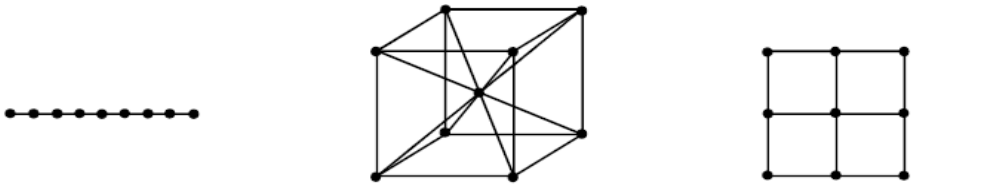
Voronoi compartments:

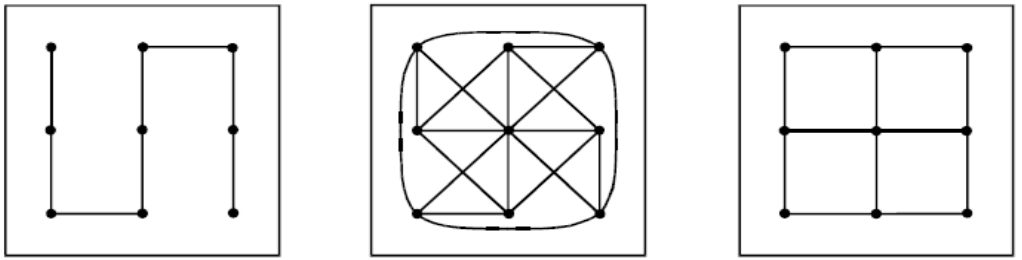
$$V_i = \{x \mid \|x - w_i\| < \|x - w_j\|\}, \quad \forall j \neq i$$



Voronoi tessellation

Topology preserving mapping





X←Y X→Y X↔Y

various measures of topology preservation proposed

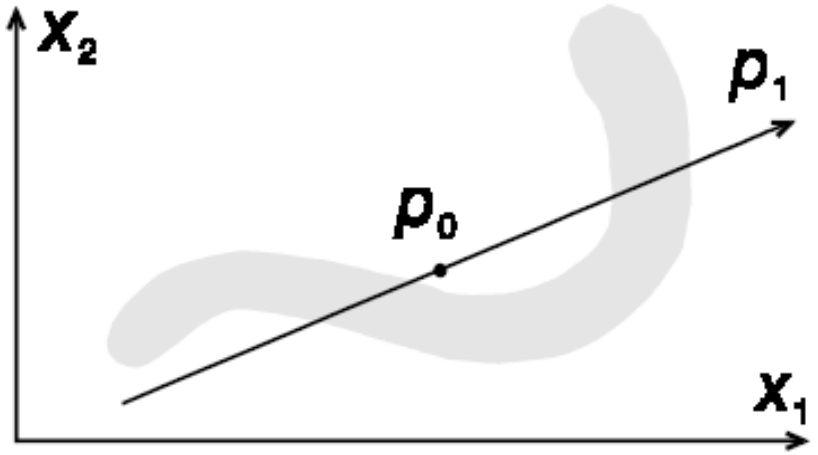
Main properties of SOM

- Approximation of the input space (input data) by the grid of neurons → Vector quantization theory
- Topological ordering (preservation of similarities between input and output spaces)
- Density matching – reflecting the variations in the statistics of input distribution
- Feature selection – via nonlinear mapping → principal curves or surfaces (Hastie and Stuetzle, 1989)
    - SOM as a nonlinear generalization of PCA

Comparison of SOM to PCA

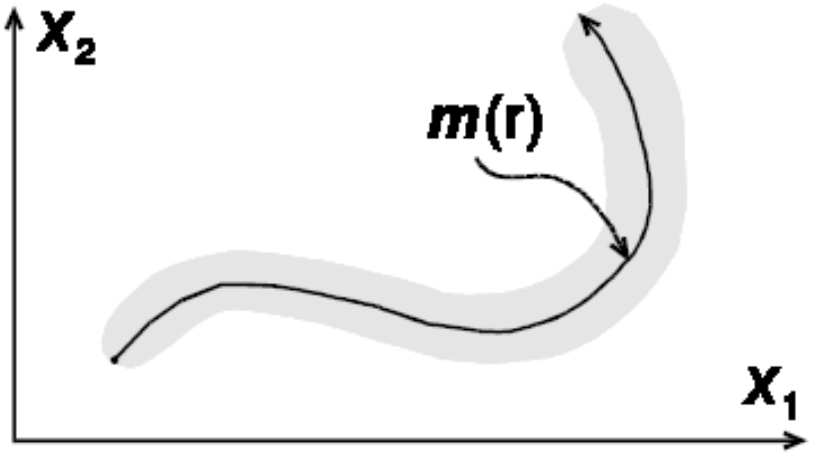
- feature extraction and mapping, difference in feature representation

PCA



(linear) principal components
One unit represents 1 dimension

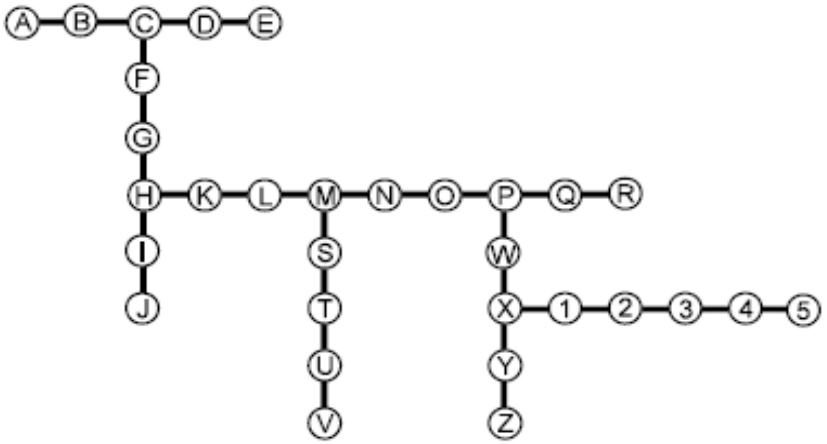
SOM

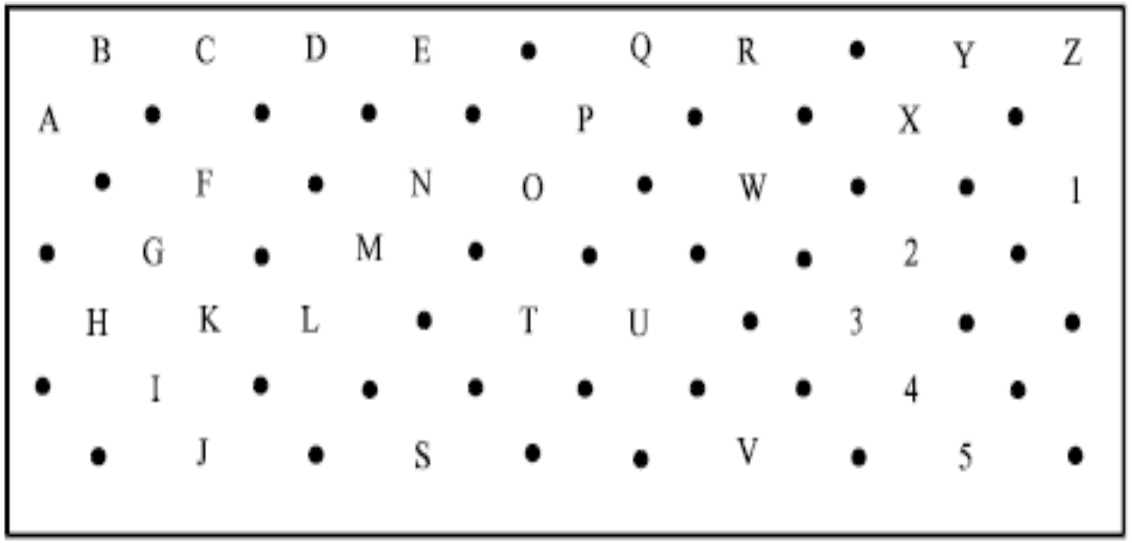


(nonlinear) principal manifold
More units represent 1 dimension

Application: Minimum spanning tree

(Kohonen,1990)



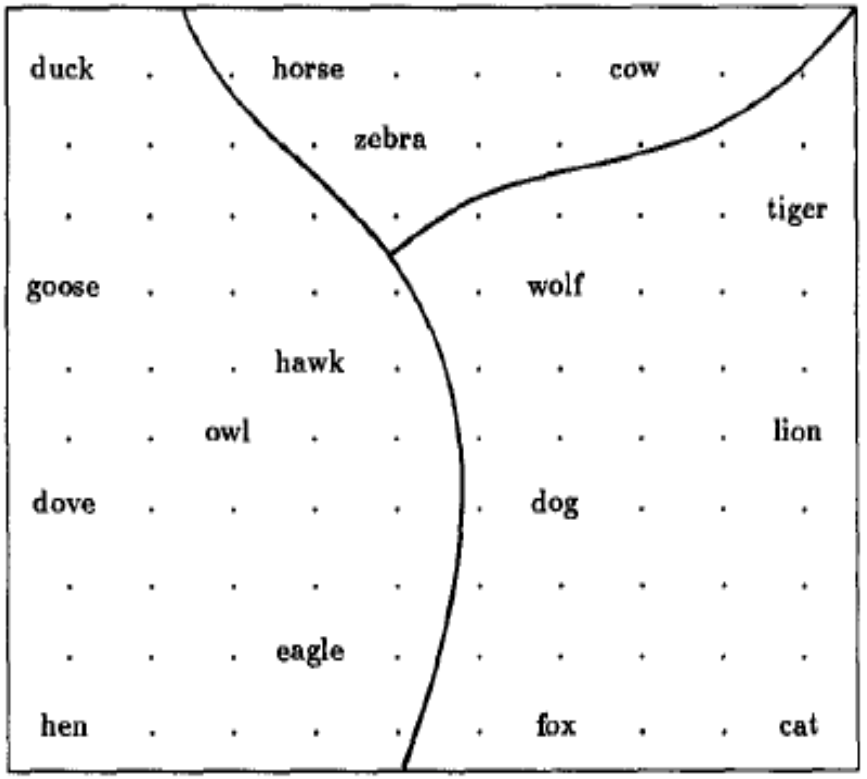


Input vector encoding:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5
1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0	0	0	0	0	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	3	6	6	6	6	6	6	6	6	6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	1	2	3	4	2	2	2	2	2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5

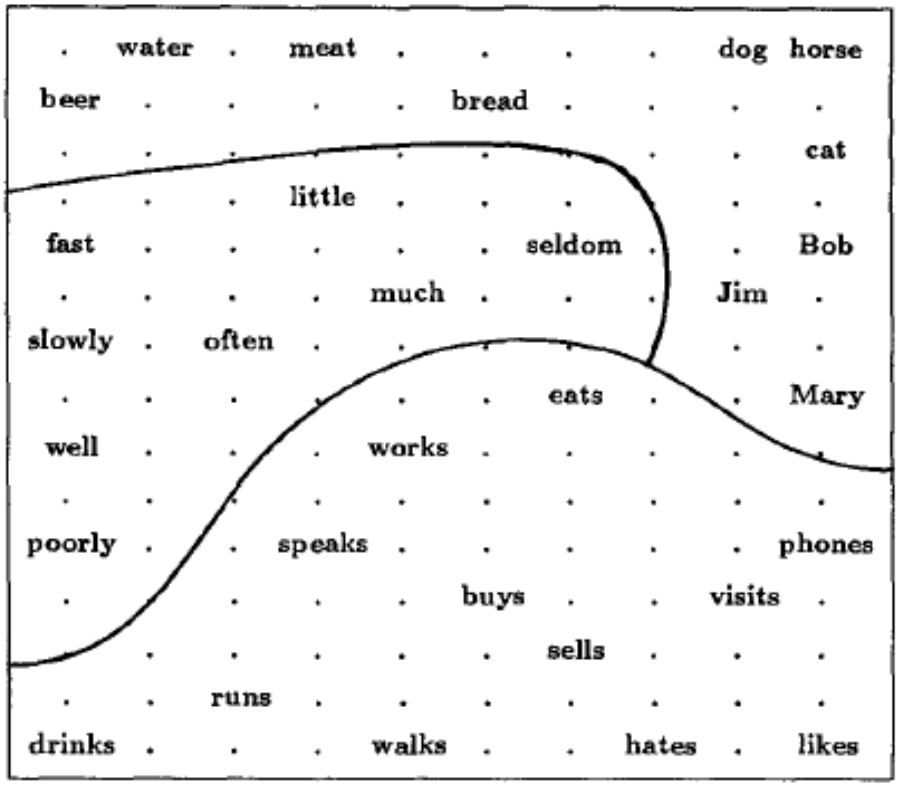
Application: Semantic maps

Attribute map



Words $\boldsymbol{x} = [\boldsymbol{x}_{symbol}; \boldsymbol{x}_{attributes}]$, i.e. combo of symbolic (one-hot) and binary attribute vectors (13-dim), of 3 categories (is, has, likes to). Testing done using $\boldsymbol{x} = [\boldsymbol{x}_{symbol}; \boldsymbol{0}]$.

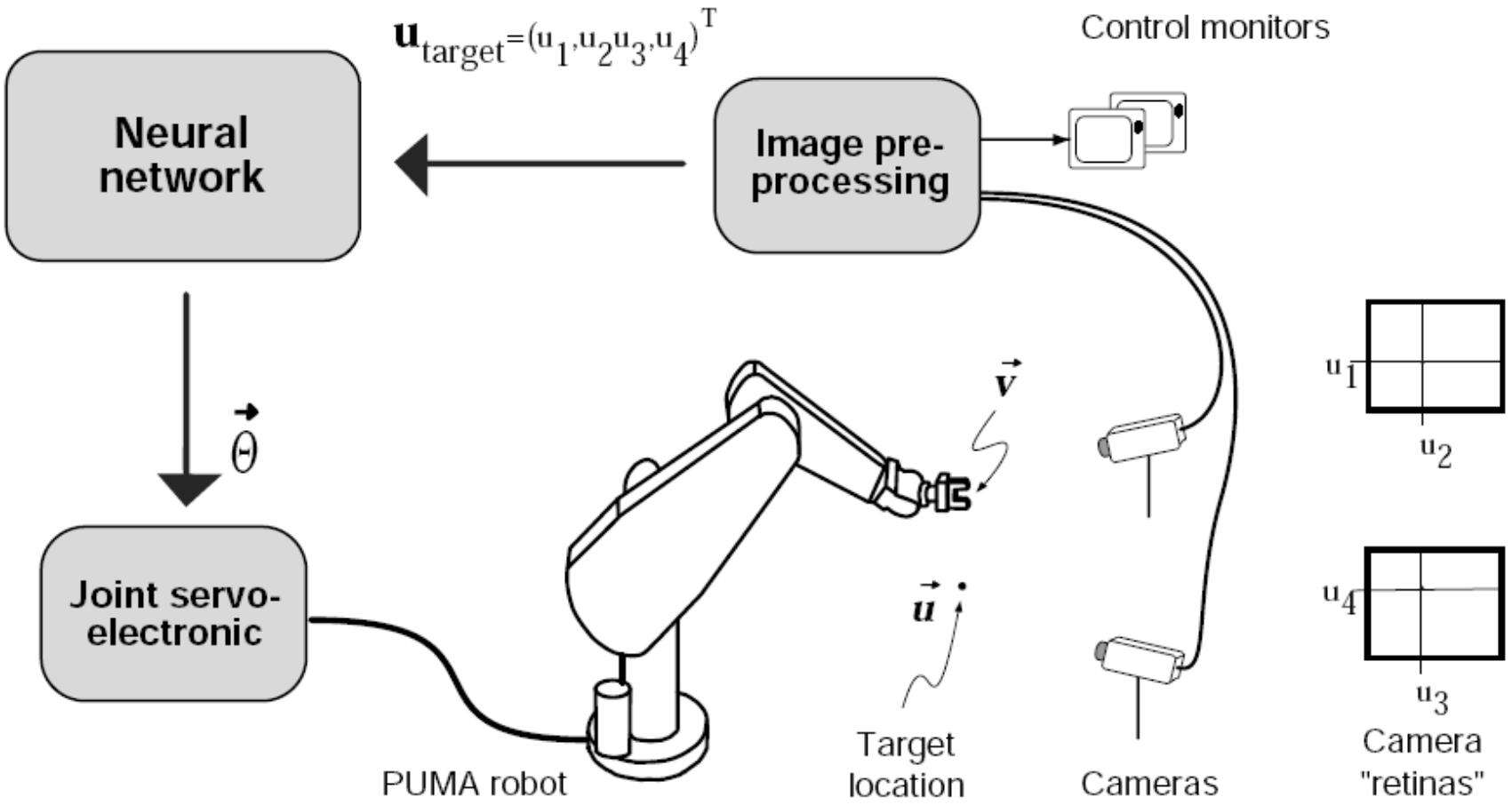
Role-based map



Words represented in contexts, i.e. $\boldsymbol{x} = [\boldsymbol{x}_{symbol}; \boldsymbol{x}_{context}]$, using sentence templates [noun - verb - adverb/noun]. Symbol ~ 7-dim. vector (of unit length), (average LR) context ~ 14-dim. (random projection used)

(Ritter & Kohonen, 1989)

Application: Robotic arm control



$$\theta(\mathbf{u}) = \theta_i + \mathbf{A}_i.(\mathbf{u} - \mathbf{w}_i)$$

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \varepsilon.h(i,i^*).(\mathbf{u} - \mathbf{w}_i)$$
$$\theta_i \leftarrow \theta_i + \varepsilon.h(i,i^*).\Delta\theta_i$$
$$\mathbf{A}_i \leftarrow \mathbf{A}_i + \varepsilon.h(i,i^*).\Delta\mathbf{A}_i$$

(Walter & Schulten 1993)

Related self-organizing NN algorithms

- Can be viewed as unsupervised data approximation with undirected graph $G = (V,C)$, $V = \{\mathbf{w}_i\}$ ~ vertices, $C[n \times n]$ ~ (symmetric) connection matrix

Examples:

- Topology-Representing network (Martinetz & Schulten 1994)
  - Flexible net topology, fixed number of units
- Growing Cell Structures (Bruske & Sommer, 1995)
  - Flexible topology and number of units (they can be removed or added based on max. quantization error)
- useful for non-stationary data distributions

Summary

- self-organizing map – a very popular algorithm
  - principles of competition and cooperation, unsupervised learning
  - performs vector quantization and topology-preserving mapping
- useful for data clustering and visualization
- theoretical analysis of SOM limited to simple cases
- various self-organizing algorithms developed
  - main purpose: data clustering
  - not all implement dimensionality-reducing mapping
  - flexible architectures possible

7

Radial-basis function networks

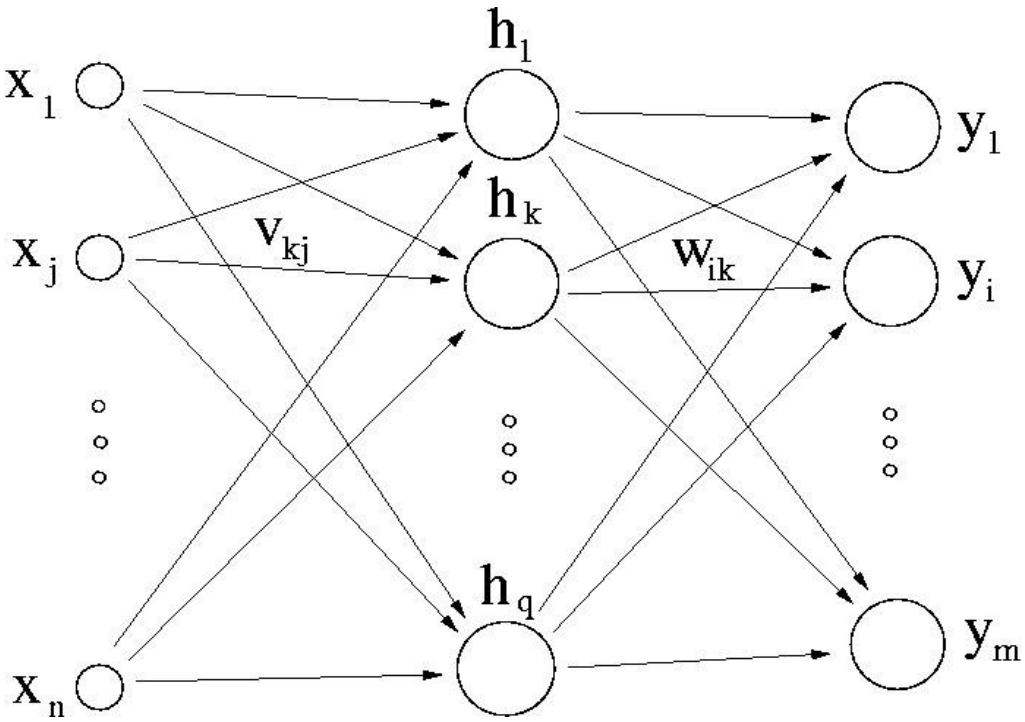
Combined NN models

- combination of unsupervised and supervised learning
- independent optimization, can be much faster than gradient descent, with similar results
- works well if similar inputs are to produce similar outputs
- unsupervised learning ⇡ clustering
- more hidden units may be needed (compared to a completely supervised model)
- Examples:
  - counter-propagation networks (Hecht-Nielsen, 1987)
  - learning vector quantization (Kohonen, 1990)
  - radial-basis-function networks (Moody & Darken, 1989)

Radial-Basis-Function neural net

- Inputs $\boldsymbol{x}$, weights $\boldsymbol{w}$, outputs $\boldsymbol{y}$
- Output activation:

$$y_i = \sum_{k=1}^{q} w_{ik} h_k(\boldsymbol{x}) + w_{i0}$$



- h_k = radial activ. function, e.g.

$h_k(\boldsymbol{x}) = \varphi(\boldsymbol{x}) = \exp(-\|\boldsymbol{x}-\boldsymbol{v}_k\|^2/\sigma_k^2)$

$\boldsymbol{v}_k \sim$ center k, $\sigma_k \sim$ its width

$\varphi(d)$ are (usually) local functions because for $d \to \infty \;\; \varphi(d) \to 0$

σ affects generalization

- $\boldsymbol{v}_k$ used for approximation of unconditional probability density of input data $p(\boldsymbol{x})$
- RBF as a receptive field (easier than that of an MLP)

Separability of patterns

- Cover's theorem (1965):

  A complex pattern classification problem cast in a high-dim. space nonlinearly is more likely to be linearly separable than in a low-dim. space.

- Consider binary partitioning (dichotomy) for $\boldsymbol{x}_1,\boldsymbol{x}_2,...,\boldsymbol{x}_N$ (classes C_1,C_2). Dichotomy $\{C_1,C_2\}$ is $\boldsymbol{\phi}$-separable, where $\boldsymbol{\phi}(\boldsymbol{x}) = [\varphi_1(\boldsymbol{x}), \varphi_2(\boldsymbol{x}),...,\varphi_q(\boldsymbol{x})]$, if $\exists \boldsymbol{w} \in \Re^q$ such that for $\forall \boldsymbol{x} \in C_1$: $\boldsymbol{w}^{\mathrm{T}}.\boldsymbol{\phi}(\boldsymbol{x}) > 0$ and for $\forall \boldsymbol{x} \in C_2$: $\boldsymbol{w}^{\mathrm{T}}.\boldsymbol{\phi}(\boldsymbol{x}) < 0$.

- $\{\varphi_k(\boldsymbol{x})\}$ – feature functions (hidden space), $k = 1,2,...,q$

- Sometimes, non-linear transformation can result in linear separability without having to increase data dimension (e.g. XOR problem):

$\boldsymbol{v}_1 = [0\ 0]$, $\boldsymbol{v}_2 = [1\ 1]$

$\varphi_k(\boldsymbol{x}) = \exp(-\|\boldsymbol{x}-\boldsymbol{v}_k\|^2)$

$\boldsymbol{x}$	$\varphi_1(\boldsymbol{x})$	$\varphi_2(\boldsymbol{x})$
(1,1)	1	0.14
(0,1)	0.37	0.37
(1,0)	0.37	0.37
(0,0)	0.14	1

Interpolation problem

- Mapping data into higher dimensions can be useful
- We deal with multivariate interpolation in high-dim. space (Davis, 1963):

  Given the sets $\{\boldsymbol{x}_i \in \Re^q,\ d_i \in \Re\}$ find a function F that satisfies the condition: $F(\boldsymbol{x}_i) = d_i,\ i=1,2,...,N.$ (in strict sense)
- For RBF, we get the set of linear equations: $\boldsymbol{w}^{\mathrm{T}} \boldsymbol{h}_i = d_i,\ i = 1,2,...,N.$
- If $\mathbf{H}^{-1}$ exists, the solution is $\boldsymbol{w} = \mathbf{H}^{-1} \boldsymbol{d}$
- How can we be sure that interpolation matrix $\mathbf{H}$ is nonsingular?
- Michelli's theorem (1986): Let $\{\boldsymbol{x}_i \in \Re^n\}$ be a set of distinct points $(i=1,2,...,N)$. Then $\mathbf{H}$ $[N \times N]$, whose $h_{ij} = \varphi_{ij}(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$, is nonsingular.
- a large class of RBFs satisfies this condition

Various types of radial-basis functions

Gaussian: $\varphi(r) = \exp(-r^2/\sigma^2)$

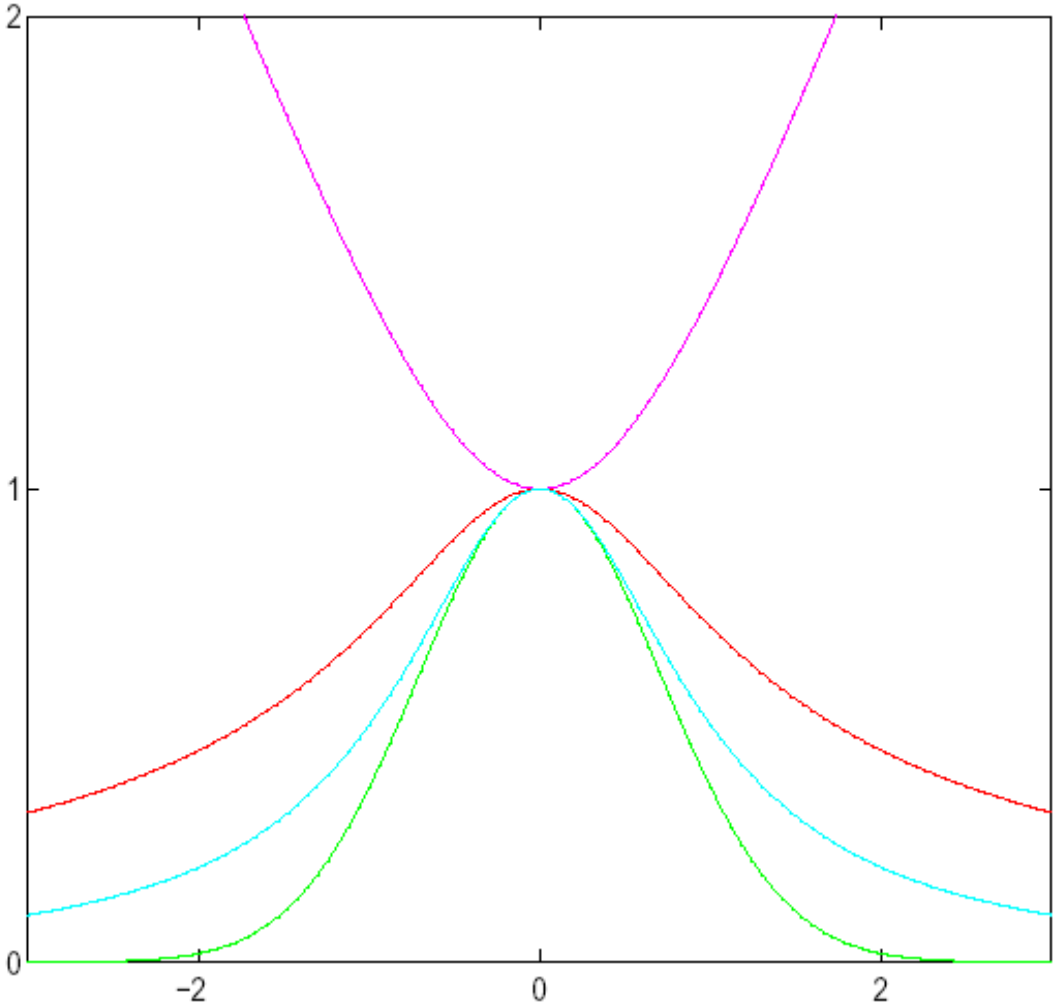
Multiquadrics: $\varphi(r) = (r^2+c^2)^{1/2}$

Inverse multiquadrics: $\varphi(r) = (r^2+c^2)^{-1/2}$

Cauchy: $\varphi(r) = 1/(1+r^2)$

$$r \in \Re, \quad c>0$$

RBFs that grow at infinity (multiquadrics) can be used to approximate a smooth I/O mapping with greater accuracy than those that yield positive-definite interpolation matrix (Powell, 1988).



Approximation properties of RBF networks

Theorem: (Park & Sandberg, 1991) Let G: $\Re^q \rightarrow \Re$ be an integrable bounded function such that G is continuous and $\int_{\Re^q} G(\boldsymbol{x})\, d\boldsymbol{x} \neq 0$. The family of RBF networks consists of functions F: $\Re^q \rightarrow \Re$:

$$F(\boldsymbol{x}) = \sum_{k=1}^{q} w_k\, G((\boldsymbol{x}-\boldsymbol{v}_k)/\sigma)$$

where $\sigma > 0$, $w_k \in \Re$ and $\boldsymbol{v}_k \in \Re^q$.

Then for any continuous function $f(\boldsymbol{x})$ there exists an RBF network with a set of centers $\boldsymbol{v}_k \in \Re^q$ and a common width $\sigma > 0$ such that $F(\boldsymbol{x})$ realized by RBF network is close to $f(\boldsymbol{x})$ in L_p norm, $p \in [1,\infty]$.

Note: Theorem does not require radial symmetry for kernel G: $\Re^q \rightarrow \Re$.

Comparison of RBF and MLP

- both are nonlinear layered feedforward networks
- both are universal approximators, using parametrized compositions of functions of single variables.
- localized vs. distributed representations on hidden layer =>
  - convergence of RBF may be faster
  - MLPs are global, RBF are local => MLP need fewer parameters
- different designs of a supervised network:
  - MLP = stochastic approximation problem
  - RBF = hypersurface-fitting problem in a high-dim. space
- one-stage (MLP) vs. two-stage (RBF) training scheme

Training RBF nets

- two-stage process
- nonlinear (layer 1) and linear (layer 2) optimization strategies are applied to different learning tasks => different time scales
- Approaches for layer 1:
  - fixed centers selected at random
  - self-organized selection of centers
- Approaches for layer 2
  - online optimization (e.g. steepest descent), or RLS
  - via pseudoinverse $\mathbf{H}'$: then $\mathbf{W} = \mathbf{H}'\mathbf{D}$
- Another method: supervised selection of centers and output weight setting

Strategy 1: Fixed centers selected at random

- "sensible" approach if training data are distributed in a representative manner:

$$G(\|\boldsymbol{x}-\boldsymbol{v}_i\|^2) = \exp(-K\|\boldsymbol{x}-\boldsymbol{v}_i\|^2/d^2_{\max})$$

K – number of centers, $d_{\max} = \max_{kl}\{\|\boldsymbol{v}_k-\boldsymbol{v}_l\|\}$, $\Rightarrow \sigma = d_{\max}/(2K)^{1/2}$

- RBFs become neither too flat nor too wide
- Alternative: individual widths σ, inversely proportional to density $p(\boldsymbol{x})$ – requires experimentation with data
- relatively insensitive to regularization, for larger data sets

Strategy 2: Self-organized selection of centers

- Hybrid learning process – useful for smaller data sets
- Self-organization: *K*-means clustering algorithm:
  1. Initialization: random (different) setting of $\{\boldsymbol{v}_1(0), \boldsymbol{v}_2(0), ..., \boldsymbol{v}_K(0)\}$
  2. take a sample $\boldsymbol{x}(t)$ randomly
  3. find "winner": $c = \arg\min_j \|\boldsymbol{x}(t) - \boldsymbol{v}_j(t)\| \quad j = 1, ..., K$
  4. Update winner's center: $\boldsymbol{v}_c(t+1) = \boldsymbol{v}_c(t) + \alpha\,[\boldsymbol{x}(t) - \boldsymbol{v}_c(t)]$
  5. Increment t, go to 2, until *stopping_criterion* is met.

Cost function: $min_{\{\boldsymbol{v}_j\}} \sum_{j=1}^{K} \sum_{C(i)=j} \|\boldsymbol{x}_i - \boldsymbol{v}_j\|^2$ for the given encoder *C*

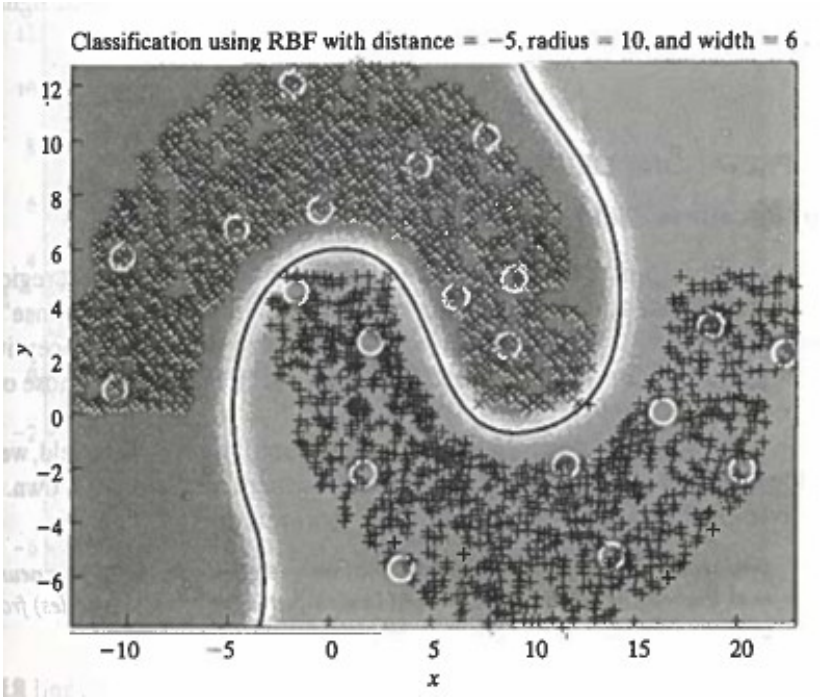
Strategy 3: Completely supervised method

- both stages are supervised
- use error function: $E = \frac{1}{2} \sum_{(p)} (e^{(p)})^2$, $e^{(p)} = d^{(p)} - \sum_k w_k \, G(\|\boldsymbol{x} - \boldsymbol{v}_k\|)$
- find parameters w_k, $\boldsymbol{v}_k$ and cov_k^{-1} to minimize E
- unlike MLP, gradient descent procedure does not involve error back-propagation
- different learning rates used
- Comparison for NETtalk: generalized (unlike standard) RBF net can substantially outperform MLP (Wettscherek & Dietterich, 1992)

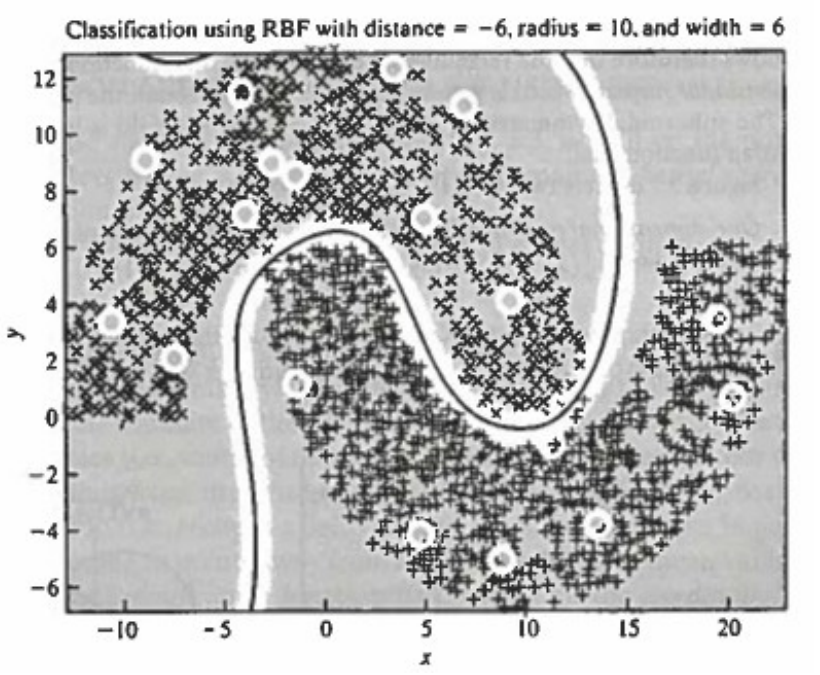
Example using RBF

Two-moons classification task: 20 Gaussian units, 1000 points used for training, 2000 for testing. Different widths (σ) used.

$\sigma = 2.6$



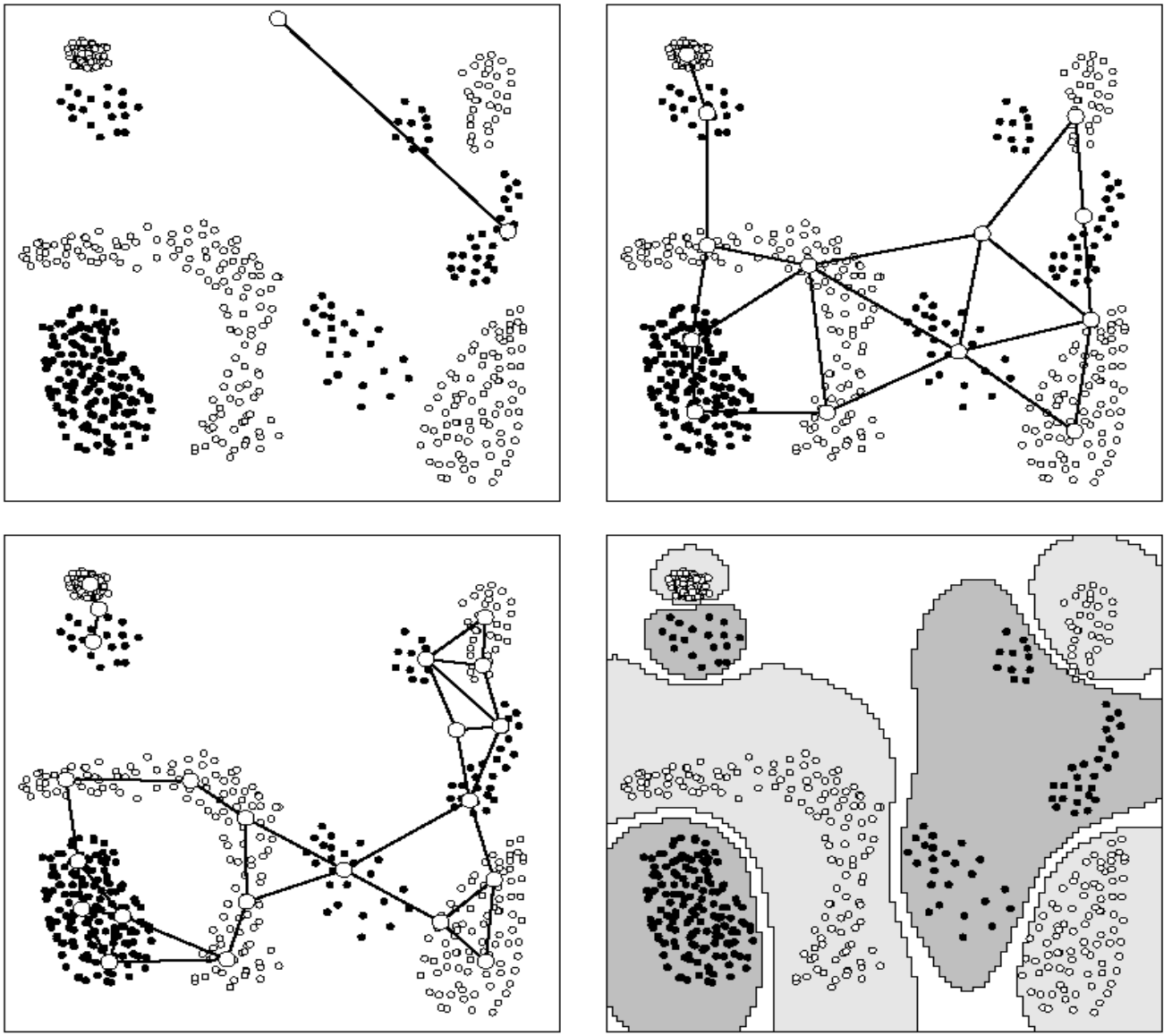
$\sigma = 2.4$



Alternative self-organizing modules for center allocation

- Can be useful for input data
  - with varying dimensionality across input domain (e.g. Topology Representing Network)
  - with non-stationary distributions – dynamic networks (Dynamic Cell Structures, Growing CS)
- to be coupled with dynamic linear part
- all based on competitive learning

Example: binary classification with a growing RBF net



(Fritzke, 1994)

Summary

- RBF – hybrid feedforward NN model
  - first layer (typically) unsupervised, second layer supervised
- we deal with interpolation problem – as an ill-posed hypersurface reconstruction problem
- universal approximator
- various training algorithms for RBF centers
- applicable for function approximation and for classification

8

Neural networks for sequential data

Temporal processing with neural networks

Required for tasks with temporal structure – where the same input can be associated with different outputs

e.g. $A \rightarrow \alpha$, $B \rightarrow \beta$, $B \rightarrow \alpha$, $B \rightarrow \gamma$, ...

Types of tasks:

- ➔ Sequence recognition (classification)
- ➔ Sequence prediction
- ➔ Sequence generation

Incorporating time into a NN:

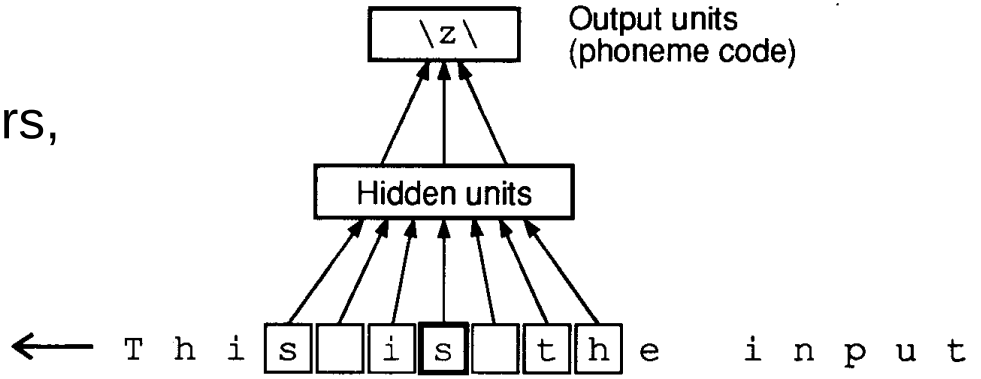
- tapped-delay input ~ time as spatial dimension
- using recurrent NN architecture
    - temporal input-output mapping
    - associative memory

Time as spatial dimension: NETtalk

(Sejnowski & Rosenberg, 1987)

NN learns to read English text

Input 7×29 units encoding 7 characters,
80 hidden and 26 output units
encoding phonemes.



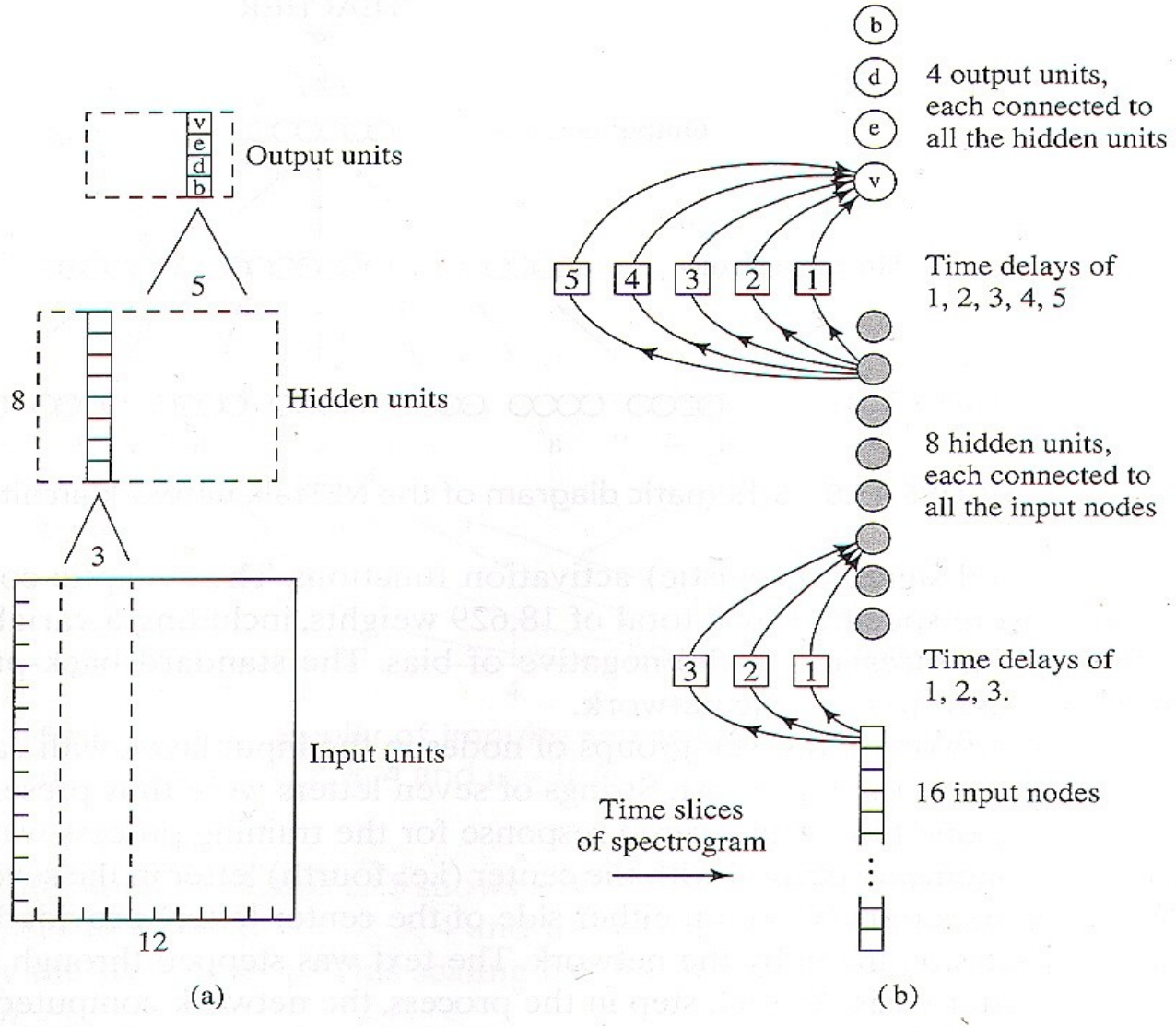
- training on 1024 words, intelligible speech after 10 epochs, 95% accuracy after 50 epochs
- 78% generalization accuracy, quite intelligible speech
- NN first learned gross features (e.g. word boundaries) and then gradually refined its discrimination (psycholinguistic plausibility).
- graceful degradation
- meaningful internal representations (e.g. vowel-consonant distinction)

- General question: How to estimate delay from training data?

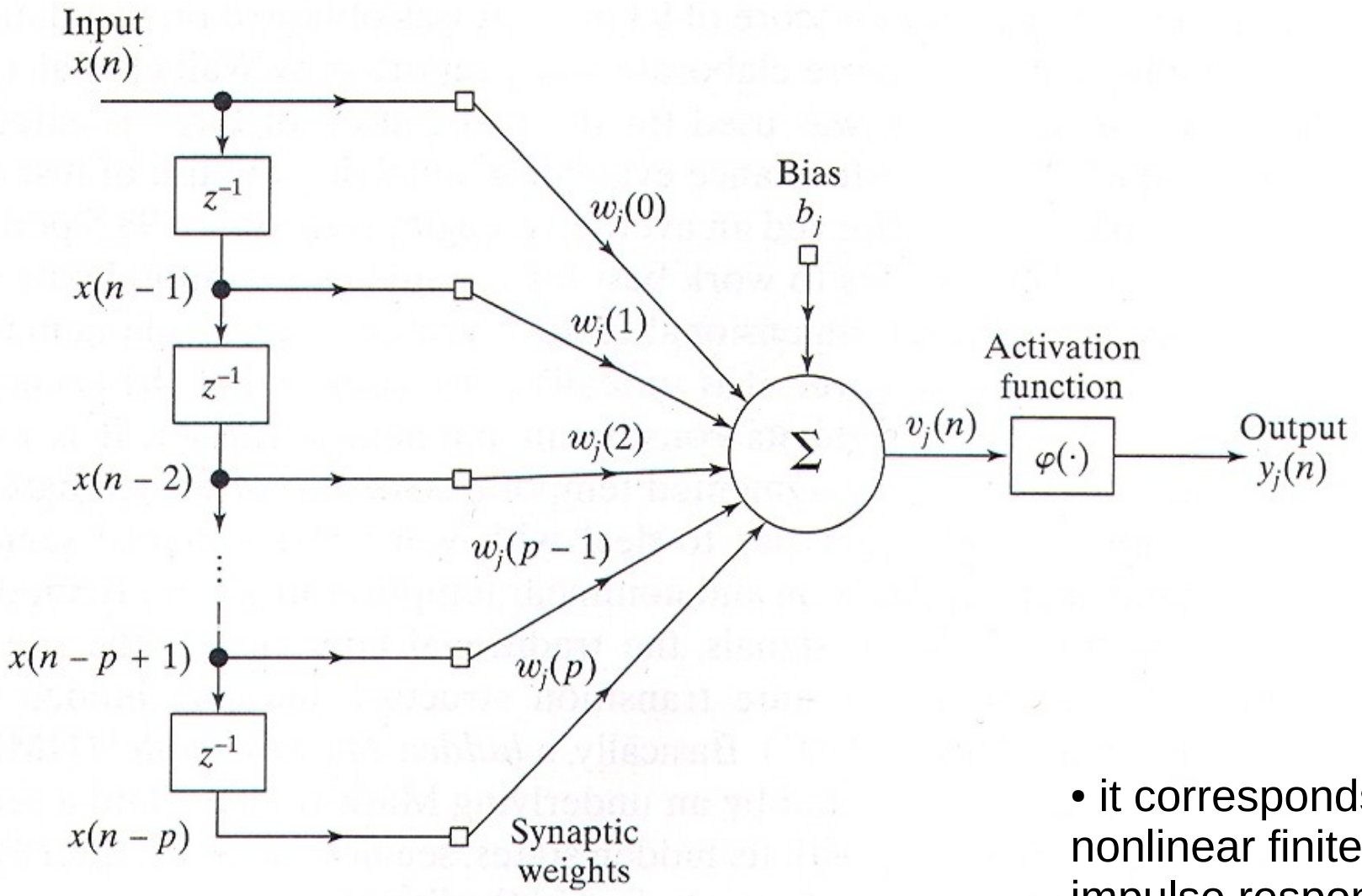
Time-Delay Neural Network

(Lang & Hinton, 1988)

- feed-forward NN
- Input: spectrogram (16×12)
- Hidden: 10 copies of 8 units
- Output: 6 copies of 4 units
- trained as feed-forward NN
- Weight sharing
- Weights = 16×8×4 = 544
- unsuitable for tasks that might require very long memory



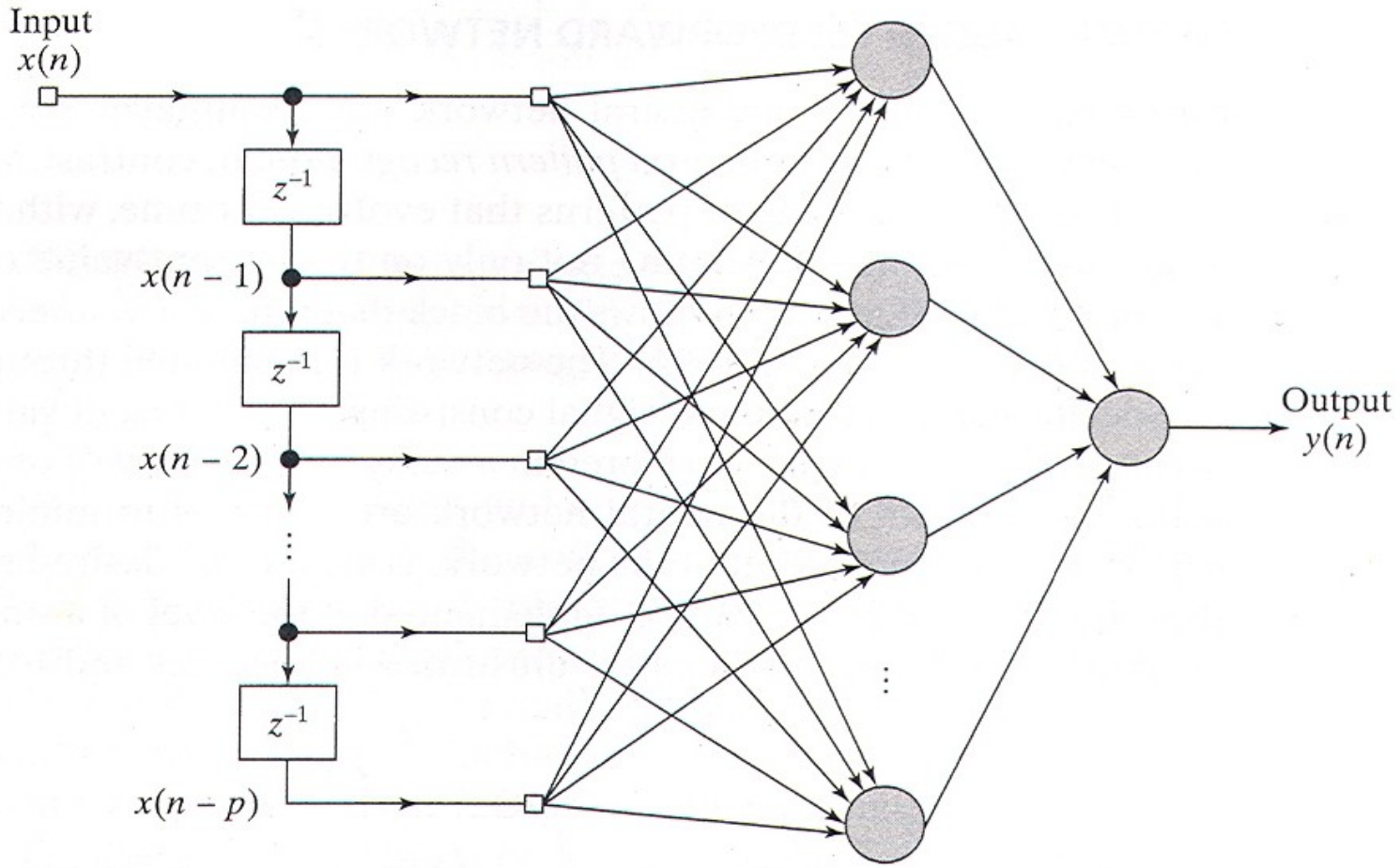
Focused neuronal filter



- focused because all memory comes from the input
- can be trained as ordinary feedforward NN

- it corresponds to a nonlinear finite impulse response (FIR) filter (in digital signal processing)

Focused time feedforward NN



- usable for stationary input-output mapping tasks
- can be trained as ordinary feedforward NN

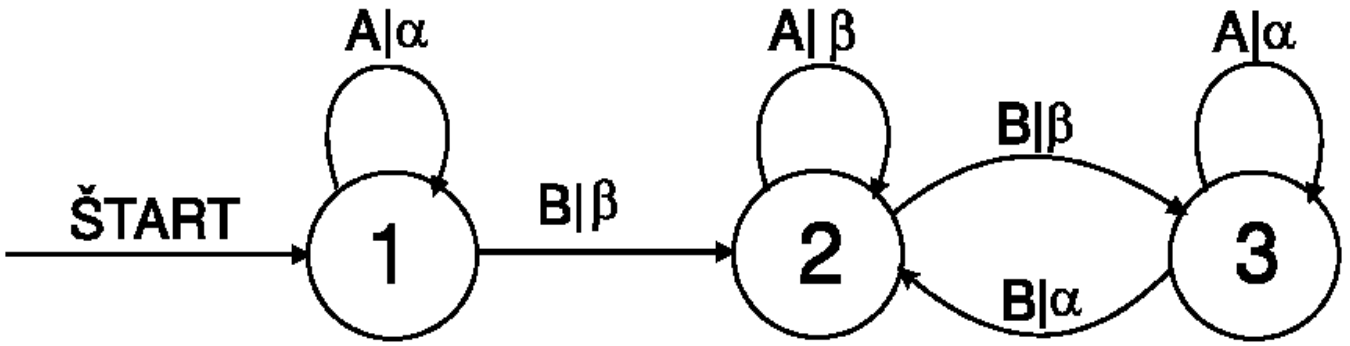
Applications of time-lagged NNs

- prediction and modeling of time series
- noise cancellation
- adaptive control
- speech recognition
- system identification

BUT... what if the required memory may be unlimited?

- time-lagged NNs have no feedback

Example: Mealy automaton



- Inputs: $\{A,B\}$, outputs: $\{\alpha,\ \beta\}$
- Training set: $A\rightarrow\alpha,\ A\rightarrow\alpha,\ B\rightarrow\beta,\ B\rightarrow\beta,\ B\rightarrow\alpha,\ A\rightarrow\beta,\ A\rightarrow\beta,\ A\rightarrow\beta$
- no sufficient tapped-line can reliably be set, so as to learn the behavior
- State representation of temporal context more appropriate than “past window” → towards recurrent models

Partially recurrent networks (with context units)

a) Elman (1990) - feedback from hidden layer

- can recognize sequences, make predictions, produce short sequence completion

b) Jordan (1986) - feedback from output layer

- option: decay units $c_i(t+1) = \alpha\, c_i(t) + y_i(t) \qquad \alpha<1$
- with fixed input, can generate various output sequences
- with input sequences, can recognize sequences

c) Stornetta (1986) - decay loop on input =>

- moving average of past activations (IIR): $c_i(t+1) = \alpha\, c_i(t) + x_i(t)$
- better suited for recognizing input sequences, than generating or reproducing them

d) Mozer (1986) – input $c_i(t+1) = \alpha\, c_i(t) + f(\sum_j v_{ij} x_j(t))$

- differs from c) in two features: full connectivity b/w inputs and context units, trainable decay links (recurrent)
- requires a learning rule different from BP, similar applicability as c)

Learning algorithms for fully recurrent NNs

- dynamically driven recurrent NNs, global feedback
  - acquire (internal) state representations
- (similarly to spatial tasks) two modes:
  - epochwise training: epoch ~ sequence
  - continuous training
- We mention two gradient based algorithms: BPTT and RTRL
- Heuristics:
  - start with shorter sequences, then increase length
  - update weights only if training error is larger than threshold
  - consider regularization (e.g. weight decay)

Back-propagation through time

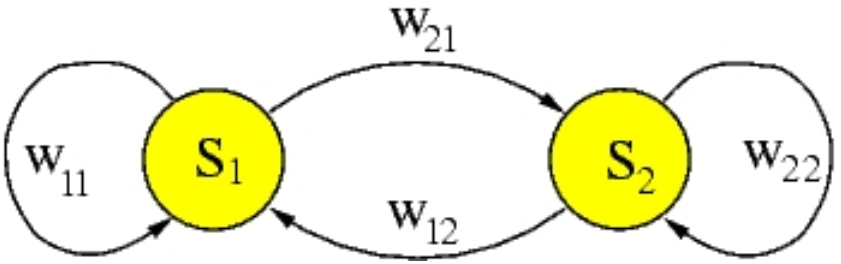
(Werbos, 1990)

- extension of standard BP algorithm – unfolding in time into a feedforward NN (with identical weights)
- sequence with inputs $\boldsymbol{x}(1), \boldsymbol{x}(2), ..., \boldsymbol{x}(T)$

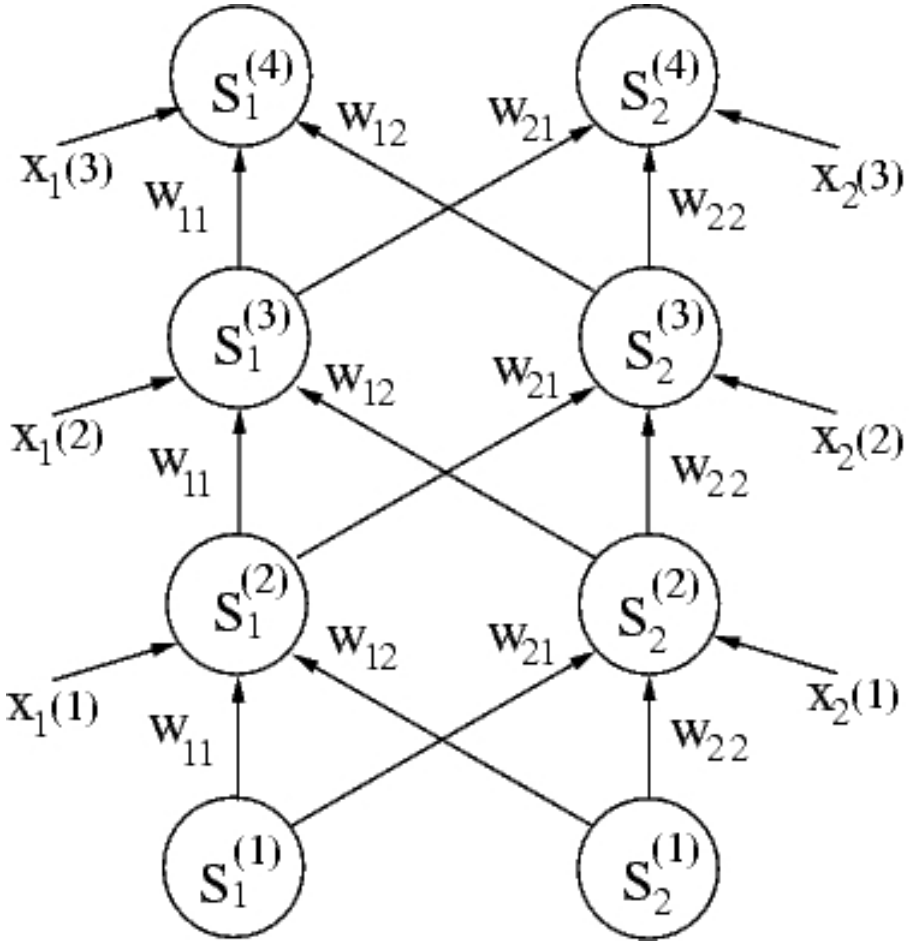
State equation:

$$s_i(t+1) = f\left(\sum_j w_{ij} s_j(t) + x_i(t)\right),$$

$$[in\ our\ example\ \ i, j = 1,2]$$



$T = 3$



BPTT algorithm

- applied after processing each sequence (of length *T*)
- during single forward pass through sequence:
  - record inputs, local gradients δ
- Overall error: $E_{\text{total}}(T) = \frac{1}{2} \sum_{t=1}^{T} \sum_{i \in O} e_i^2(t)$
- for $t = T$: $\delta_i(t) = f'(net_i)\, e_i(t)$

  for $1 < t < T$: $\delta_i(t) = f'(net_i)\, [e_i(t) + \sum_{l \in O} w_{li}\, \delta_l(t+1)]$
- Update weights: $\Delta w_{ij} = - \alpha\, \partial E_{\text{total}}(T) / \partial w_{ij} = \alpha \sum_{t=2}^{T} \delta_i(t)\, s_j(t-1)$

- impractical for longer sequences (of unknown length)

Real-time recurrent learning (RTRL)

(Williams & Zipser, 1989)

- Instantaneous output error: $e_i(t) = d_i(t) - s_i(t); \;\; i \in O$ (targets exist)

  $E(t) = \frac{1}{2} \sum_{i \in O} e_i^2(t)$

- Update weights: $\Delta w_{ij} = - \alpha \, \partial E(t) / \partial w_{ij} = \alpha \sum_{k \in O} e_k(t) \, \partial s_k(t) / \partial w_{ij}$

  $\partial s_k(t) / \partial w_{ij} = f'(net_k(t)) \, [\delta^{kr}_{ki} \, s_j(t\text{-}1) + \sum_l w_{kl} \, \partial s_l(t\text{-}1) / \partial w_{ij}]$

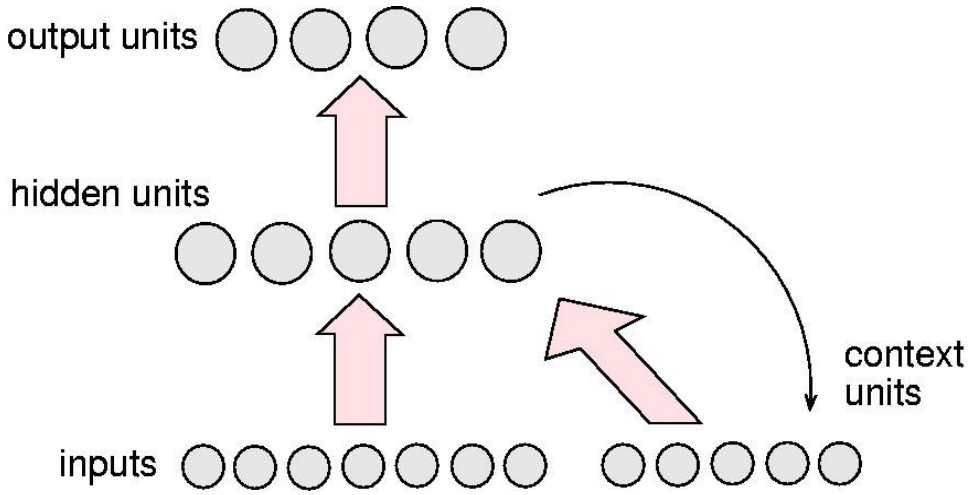
  $l \in$ units feeding to unit k, and $\delta^{kr}_{ki} = 1$, if $k = i$, else 0.

  - if j pertains to external input, $x_j(t\text{-}1)$ is used instead

- Smaller α recommended, BP "tricks" applicable (e.g. momentum)
- Teacher forcing – replace actual output with desired whenever available
  - may lead to faster training and enhance learning capability
- Very large time and memory requirements (with N neurons, each iteration): N^3 derivatives, $O(N^4)$ updates to maintain

Simple recurrent network

(Elman, 1990)



Implicit representation of time

Hidden state activation:

$$h_k(t+1) = f\left(\sum_j w_{kj} x_j(t) + \sum_l c_{kl} h_l(t)\right)$$

Output: $$y_i(t) = f\left(\sum_k v_{ik} h_k(t)\right)$$

Unit's activation function:

$$f(u) = \frac{1}{1+\exp(-u)}$$

can be trained by BP, BPTT, RTRL,...

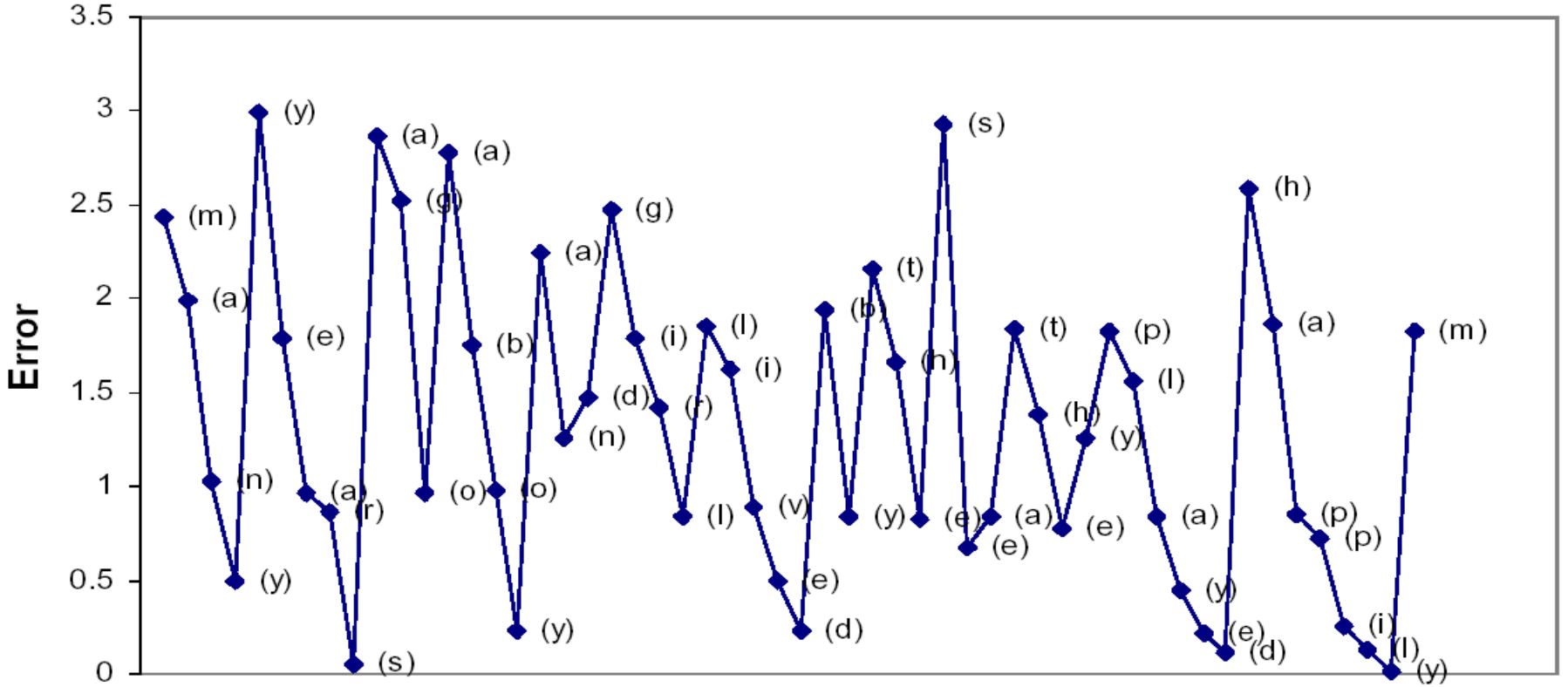
The following examples: symbolic dynamics

Example: Next letter prediction task

Task: letter-in-word prediction, 5-bit inputs
Data: 200 sentences, 4 to 9 words in a sentence
SRN: 5-20-5 units, trained by back-propagation (Rumelhart, Hinton & Williams, 1986)
- NN discovers the notion “word”

Many years ago boy and girl lived by the sea ...

(Elman, Cog. Sci., 1990)



Example: Next word prediction task

Categories of lexical items used

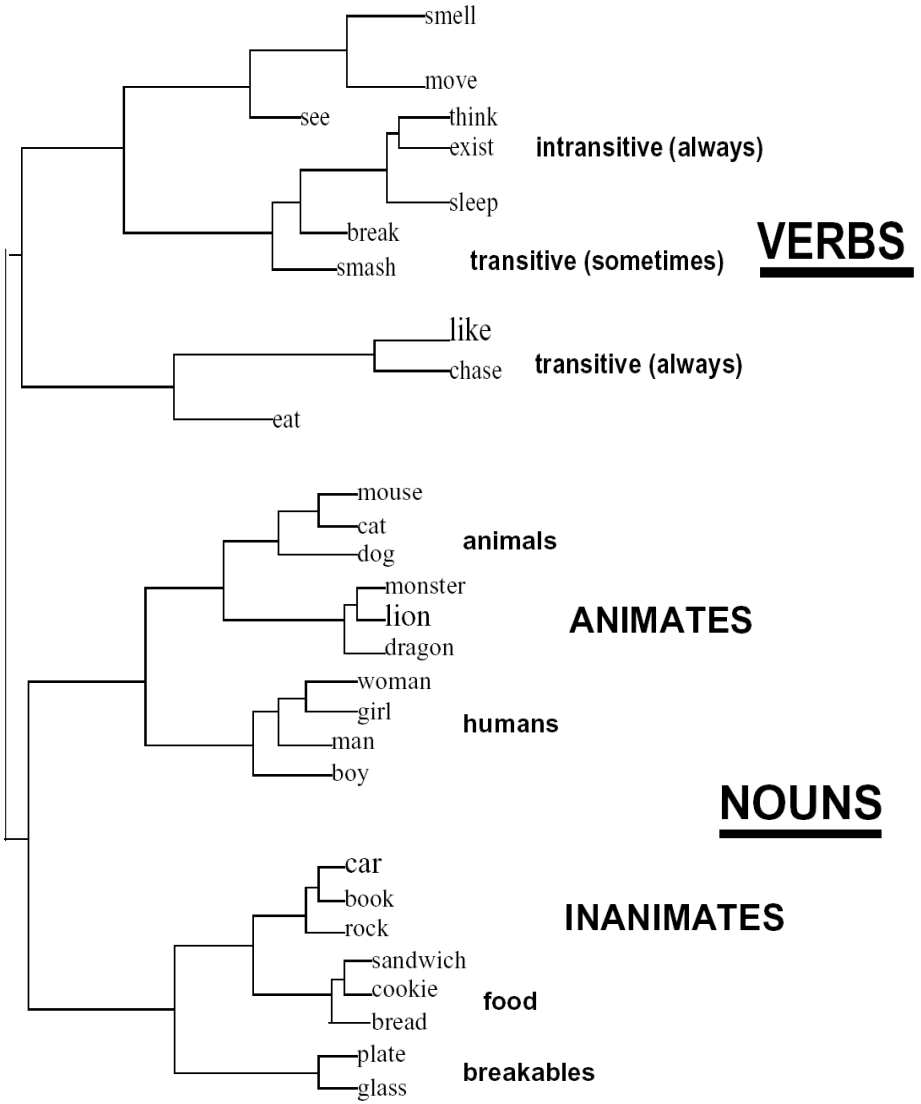
Category	Examples
NOUN-HUM	man, woman
NOUN-ANIM	cat, mouse
NOUN-INANIM	book, rock
NOUN-AGRESS	dragon, monster
NOUN-FRAG	glass, plate
NOUN-FOOD	cookie, sandwich
VERB-INTRAN	think, sleep
VERB-TRAN	see, chase
VERB-AGPA	move, break
VERB-PERCEPT	smell, see
VERB-DESTROY	break, smash
VERB-EA	eat

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

SRN: 31-150-31, localist encoding of words
no context reset b/w sentences

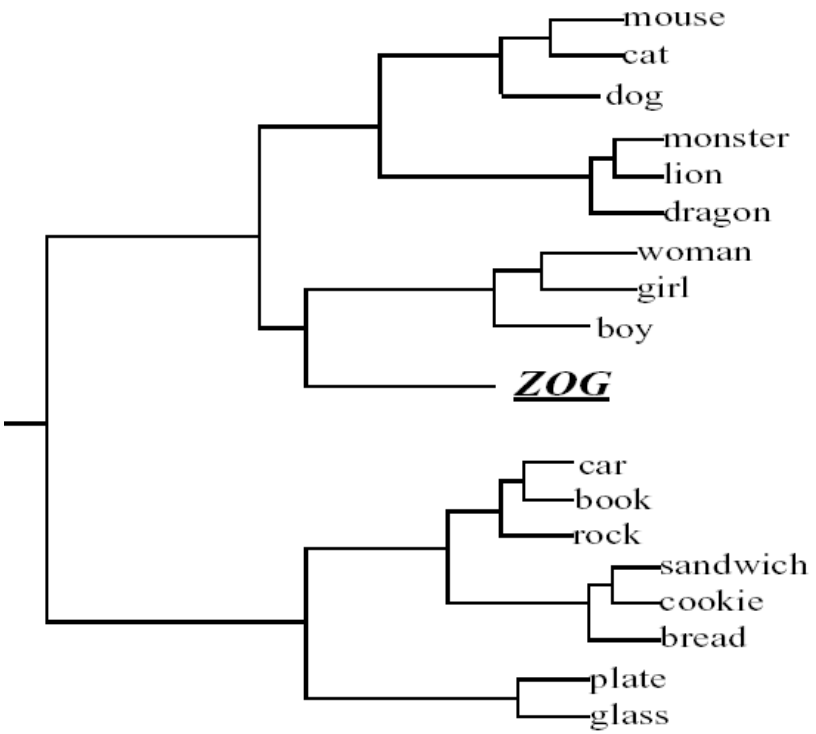
Averaged hidden-unit activation vectors



(Elman, Cog. Sci., 1990)

Properties of hidden-unit activations after training

- activations show structure (clusters)
- types/tokens distinction: types = centroids of tokens
- representations are hierarchically structured
- type vector for a novel word (*zog*) consistent with previous knowledge
- representation space would not grow with a growing lexicon



Example: Modeling recursive processing in humans

(Christiansen & Chater, 1999)

A. Counting recursion *aabb* *NNVV*

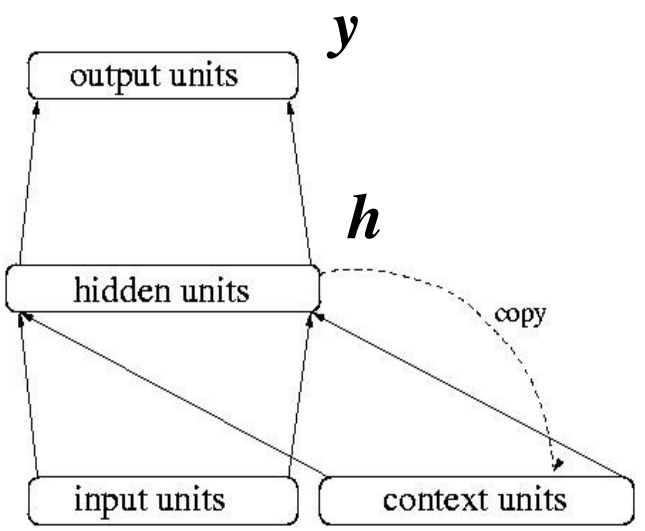
B. Center-embedding recursion *a b b a* $S_N P_N P_V S_V$ *the boy girls like runs*

C. Cross-dependency recursion *a b a b* $S_N P_N S_V P_V$ *the boy girls runs like*

D. Right-branching recursion *a a b b* $P_N P_V S_N S_V$ *girls like the boy that runs*

- Qualitative performance of SRN model matches human behavior, both on relative difficulty of B and C, and between their processing and that of D.
- This work suggests a novel explanation of people's limited recursive performance, without assuming the existence of a mentally represented competence grammar allowing unbounded recursion.
- They compare the performance of the network before and after training – pointing to architectural bias, which facilitates the processing of D over B and C.

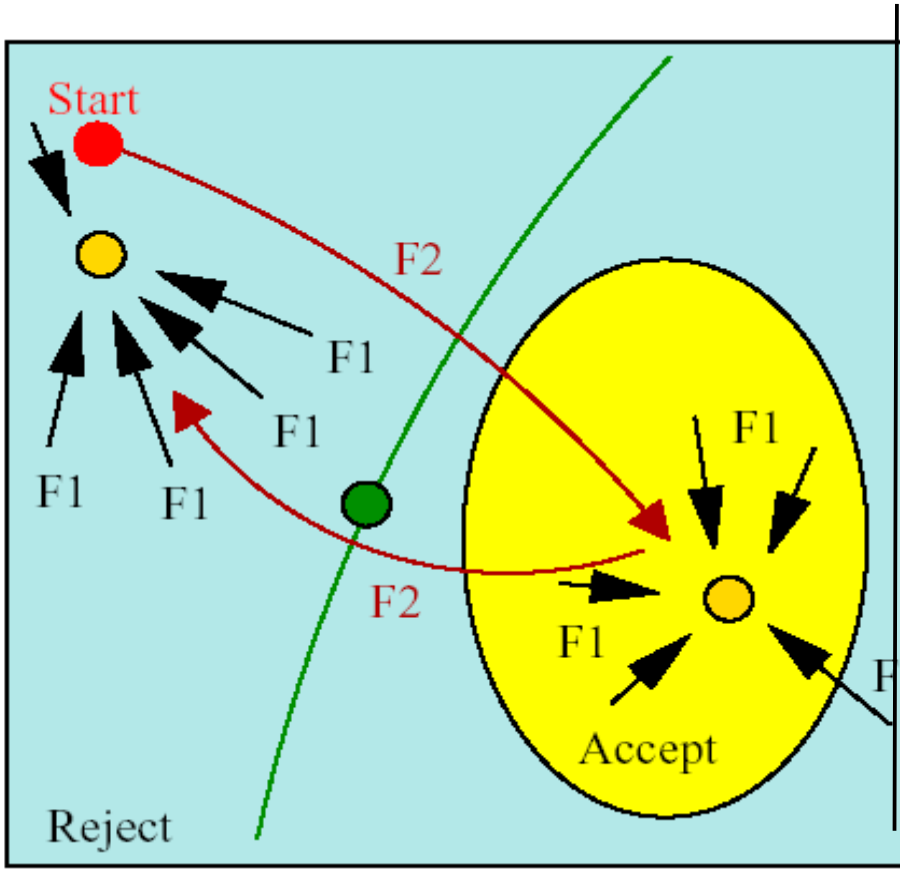
RNN state space organization



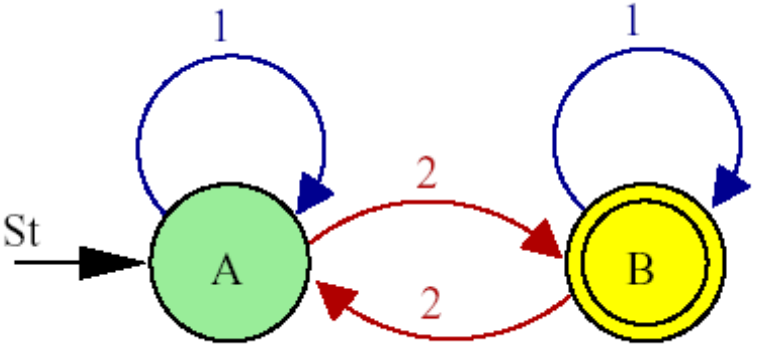
- ***y*** = **F(*h*)** is a squashed version of a linear transformation => it is smooth and monotonic
- Activations ***h*** leading to the same/similar output ***y*** are forced to lie close to each other in the RNN state space
- Heuristics for enhancing RNN generalization: cluster RNN state space into a finite number of clusters
- Each cluster will represent an abstract information-processing state => knowledge extraction from RNN (e.g. learning finite state automata with RNNs)

Example: Learning a finite state automaton

- State-space activations in RNN – neural memory – code the entire history of symbols we have seen so far.
- There exists information latching problem for gradient learning
- To latch a piece of information for a potentially unbounded number of time steps, we need attractive sets.



RNN state space

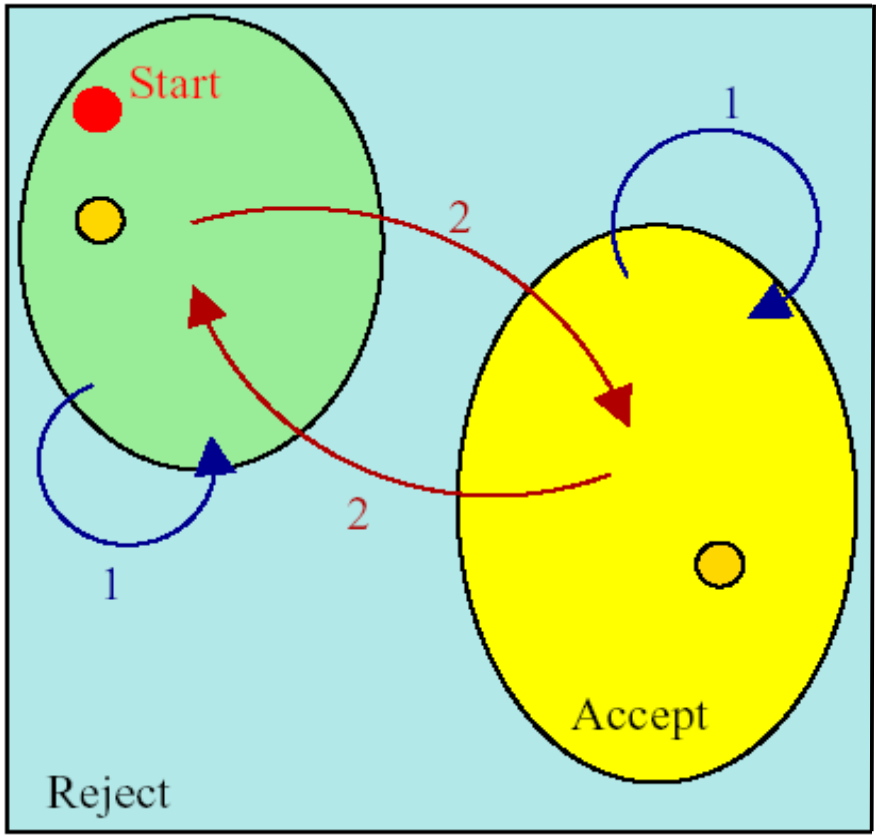


Tiňo (2003)

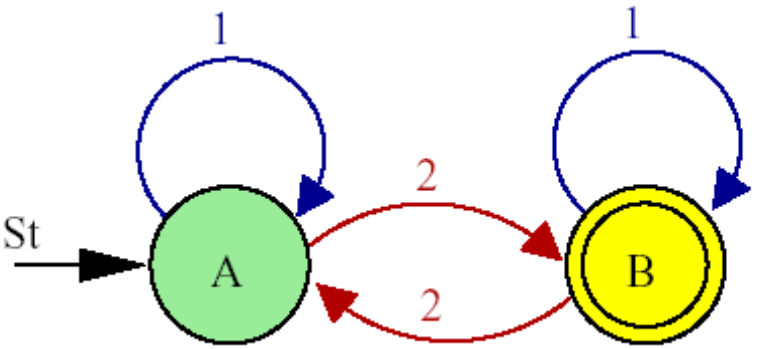
Grammatical:
all strings containing odd number of 2's

Example: string classification task

RNN has a continuous state-space



Automaton can be extracted from trained RNN (clusters of hidden-unit activations correspond to automaton states)

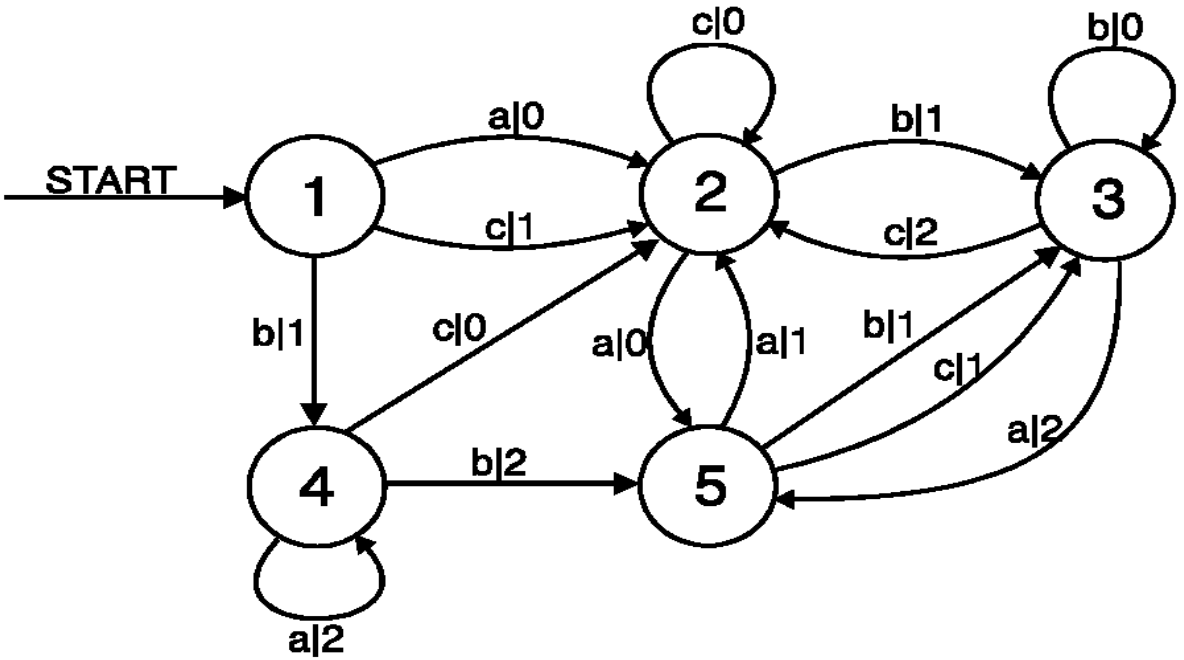


Extracted FSM:
all strings containing odd number of 2's

Tiňo (2003)

Example: temporal association task

3+1 input symbols
3+1 output symbols



Training sequences:

$$acccb! \rightarrow 00001x,\ caaab! \rightarrow 10101x,\ bbbbbb! \rightarrow 121000x,\ \text{atd'}.$$

$$acccb!caaab!bbbbbb!... \rightarrow 00001x10101x121000x...$$

Suitable scenario: start with simpler sequences
Extracted automaton can generalize training data

Computational power of recurrent networks

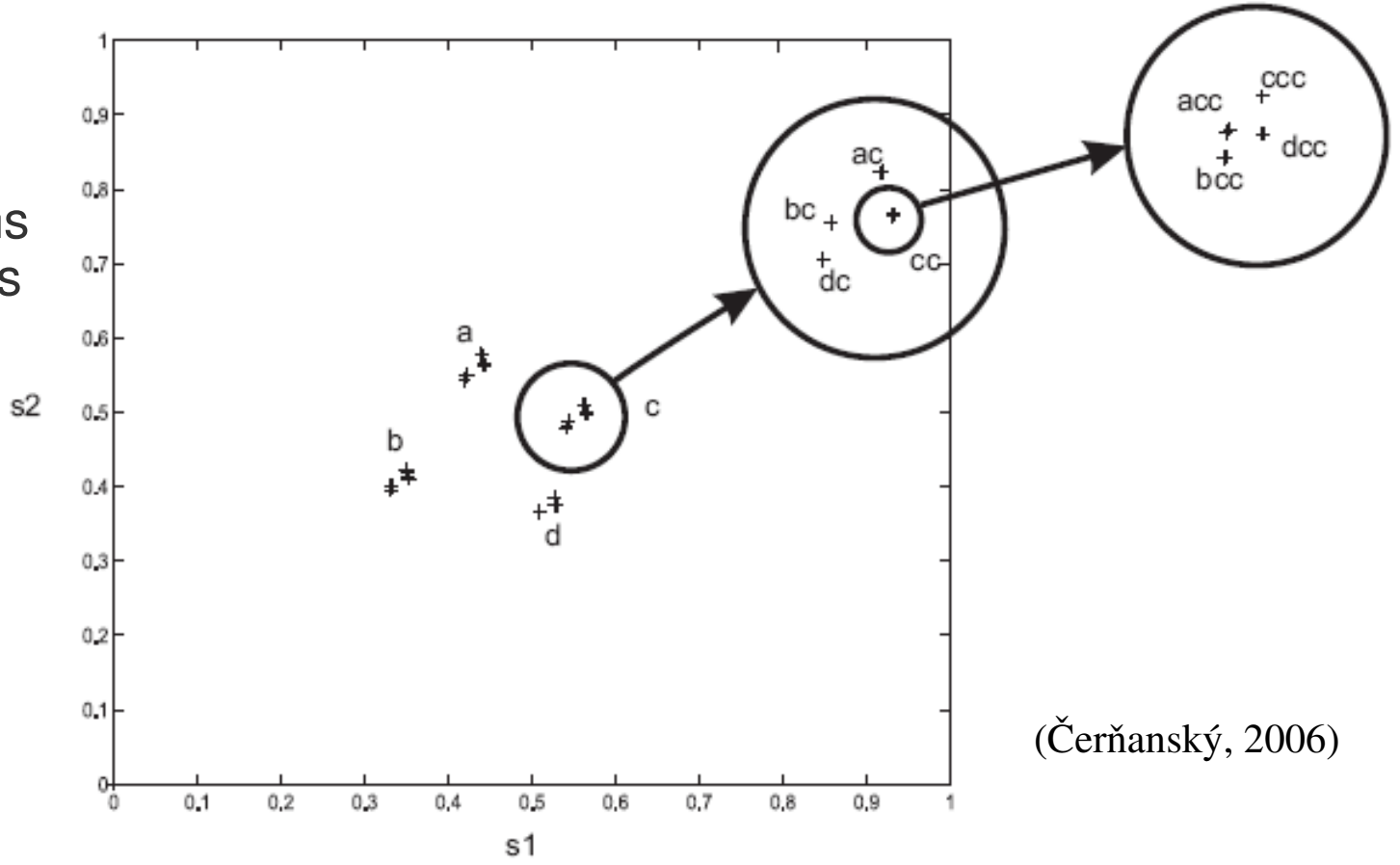
- recurrent NNs can learn to simulate formal automata
  - regular grammars (finite-state machine)
  - context-free grammars (e.g. $a^n b^n$ ~ saddle-point attractive set)
- All Turing machines may be simulated by fully connected recurrent networks built on neurons with sigmoidal activation functions. (Siegelmann & Sontag, 1991)
- Practically, we may encounter problems with convergence in more complex tasks.
- vanishing gradients problem – difficulty to learn long-term dependencies by gradient-based algorithms
  - approaches: apply heuristics, extended Kalman filtering, elaborate optimization techniques

Architectural bias

- Structured hidden-unit activations in RNN exist prior to training

(Kolen, 1994; Tiňo, Čerňanský, Beňušková, 2004; Čerňanský, Makula, Beňušková, 2007)

Hidden unit activations (2D space), 4 symbols in input alphabet



(Čerňanský, 2006)

Explanation of architectural bias in RNNs

In RNNs with sigmoid activation functions and initialized with small weights (Tiňo et al. 2004):

1) clusters of recurrent activations that emerge prior to training correspond to Markov prediction contexts – histories of symbols are grouped according to the number of symbols they share in their suffix, and

2) based on activation clusters, one can extract from untrained RNNs predictive models – variable memory length Markov models (VLMMs).

RNNs have a potential to outperform finite memory models, but to appreciate how much information has really been induced during training, RNN performance should always be compared with that of VLMMs extracted before training as the “null” base models.

Iterated Function Systems

(Barnsley, 1988)

IFS consists of a complete metric space (X,d), where $X = [0,1]^N$, d is Euclidean metric, together with a set of contractive mappings $w_i: X \to X$

$$w_i(x) = k\,x + (1\text{-}k)\,s_i \qquad i = 1,2,\ldots, A$$

Symbols $s_i \in \{0,1\}^N$ $\quad N = ceil(\log_2 A)$ Contraction coef. $k \in (0,0.5]$

Each n-symbol sequence $S = s_1 s_2 \ldots s_n$ is represented by IFS as a point

$$w(x) = w_n(w_{n-1}(\ldots(w_2(w_1(x)))\ldots)), \quad x \in X.$$

Recurrent NN with small random weights also performs contractive mappings in state space (using various *k*'s for each symbol).

IFS – topographic mapping property

Let $S_i^j = s_i\, s_{i+1} \ldots s_j$, then given a sequence $S = s_1 s_2 \ldots$ over A, the chaotic n-block representation of S is defined as a set of points

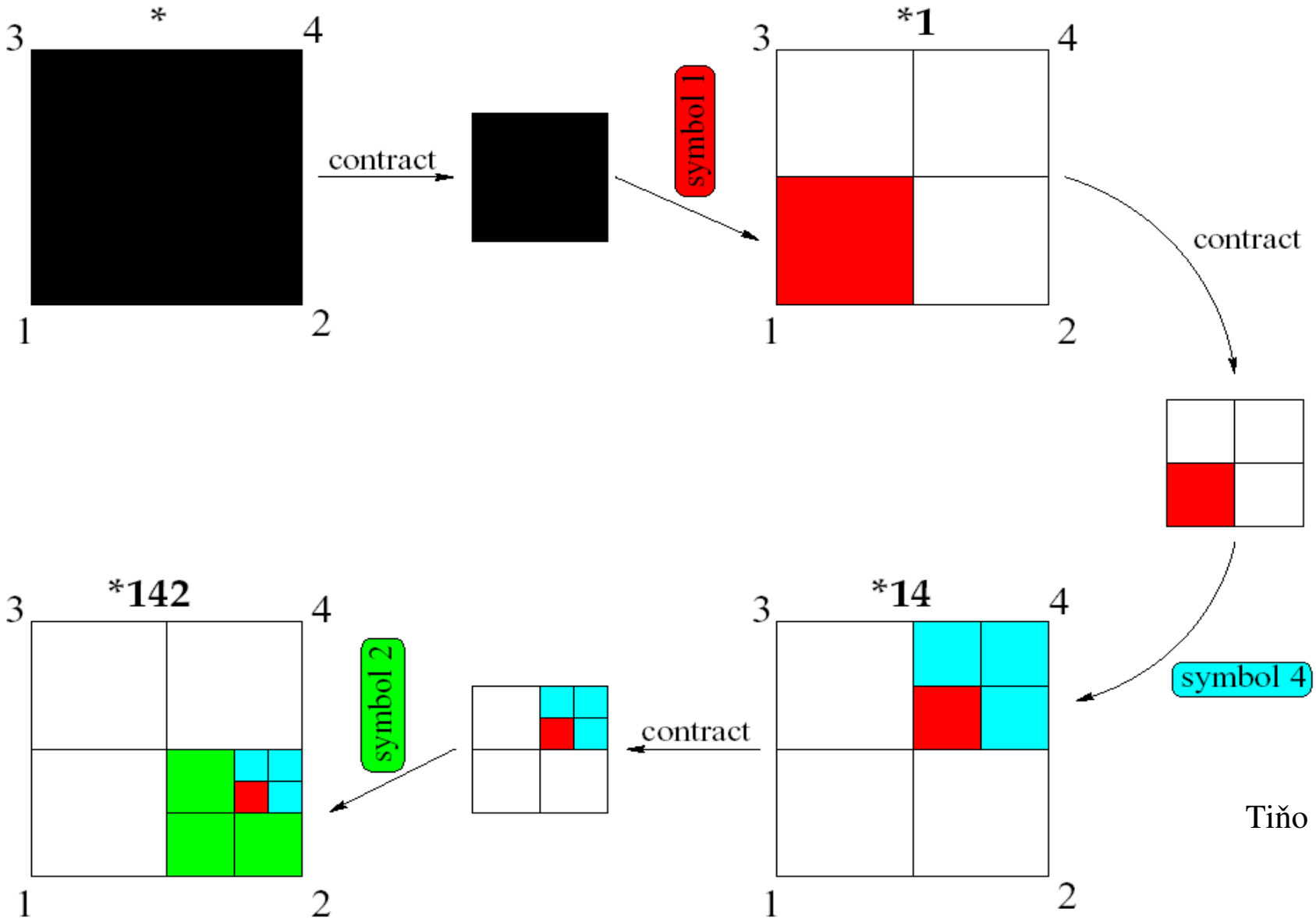
$$CBR_{n,k}(S) = \{S_i^{i+n-1}(x_*)\}_{i \geq 1}$$

where $x_* = \{1/2\}^M$ is the center of X.

- *CBR* has the property that is temporal analogue of topographic mapping: the longer is common suffix of two sequences, the closer they are mapped in *CBR*.
- On the other hand, the Euclidean distance between points representing two n-sequences that have the same prefix of length $n - 1$ and differ in the last symbol, is at least $1 - k$.

Illustration of contractive mapping (IFS)

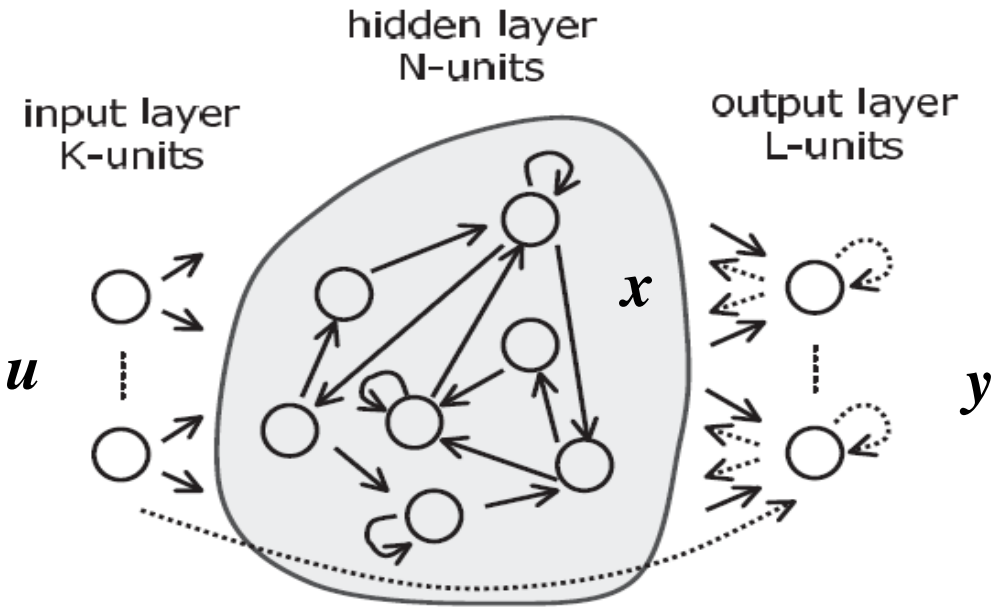
Topographic ordering with respect to suffixes



Tiňo (2003)

Reservoir computing – echo-state network

(Jaeger, 2001)



ESN can have an SRN architecture, but also additional connections are possible (useful for some tasks).

Reservoir units: usually nonlinear (logistic sigmoid or tanh)

System equations:

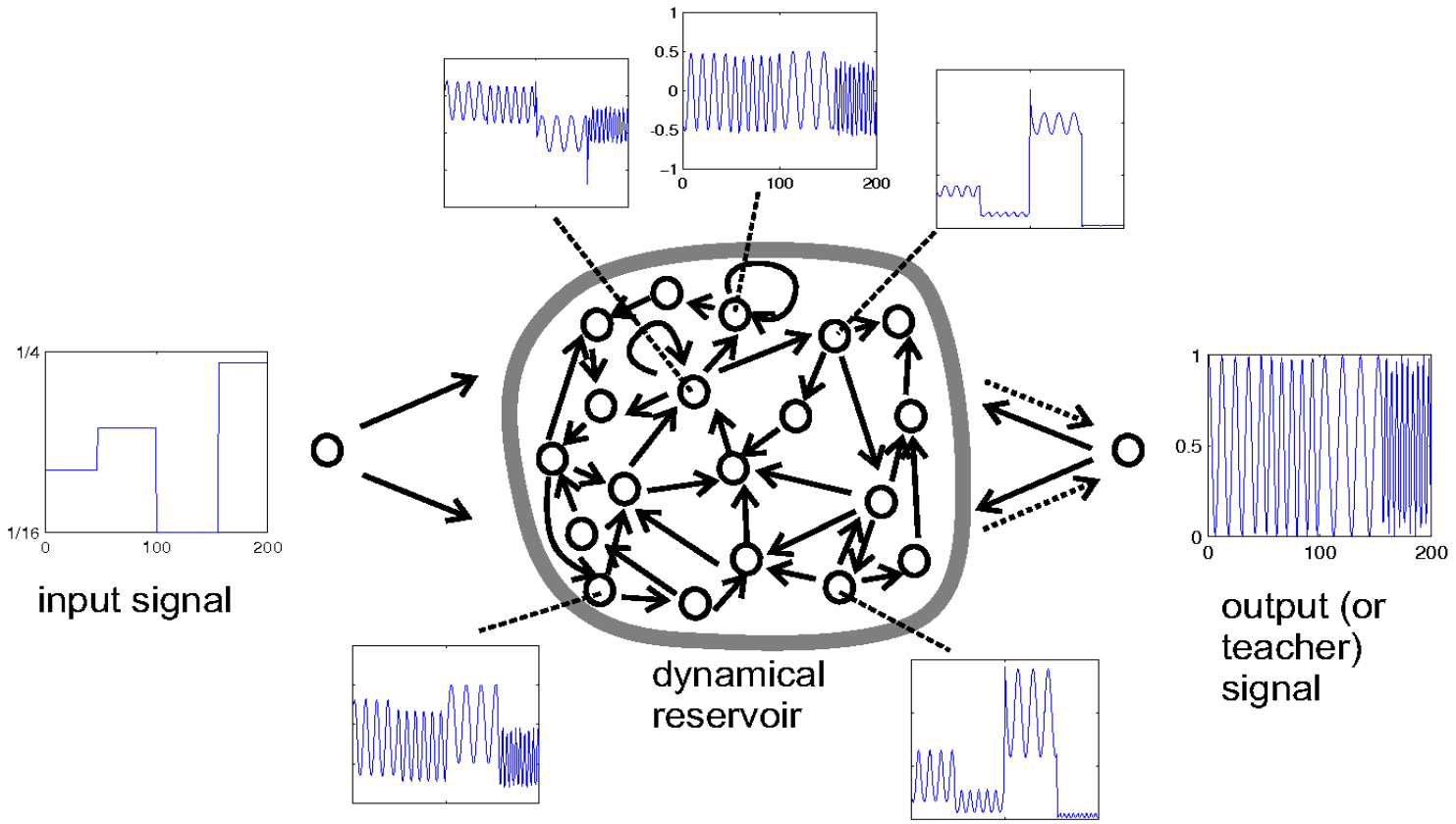
$$\boldsymbol{x}(t) = f\left(\boldsymbol{W}\,\boldsymbol{x}(t-1) + \boldsymbol{W}^{inp}\,\boldsymbol{u}(t) + \boldsymbol{W}^{fb}\,\boldsymbol{y}(t)\right)$$

$$\boldsymbol{y}(t) = f^{out}\left(\boldsymbol{W}^{out}\,\boldsymbol{z}(t)\right)$$

$$\boldsymbol{z}(t) = [\boldsymbol{x}(t); \boldsymbol{u}(t)]$$

$$\boldsymbol{W}^{out} \sim K \times (K+N)]$$

Echo State Network (ctd)



- studied issues: memory capacity, information transfer, ...
- edge of chaos = interesting regime (may be optimal)

ESN training

- Initialize
  - create the reservoir with echo-state property, i.e. reservoir states are only determined by input sequence: (Jaeger, 2001)

    Definition 2.1 (*Echo State Property*). A network $F : X \times U \to X$ (with the compactness condition) has the echo state property with respect to U: if for any left infinite input sequence $u^{-\infty} \in U^{-\infty}$ and any two state vector sequences $x^{-\infty}, y^{-\infty} \in X^{-\infty}$ compatible with $u^{-\infty}$, it holds that $x_0 = y_0$.

  - small random input weights (with uniform, or gaussian distribution)
- Collect reservoir states
  - feed the input sequence into the network (apply state equation)
- Compute output weights
  - supervised, via pseudoinverse of $\boldsymbol{X}$

- ESN behaves like a Markov model (in symbolic dynamics)

ESN properties

Echo-state property (ESP): depends on spectral properties of $\boldsymbol{W}$ = (typically) random *sparse* matrix, measures:

- spectral radius = largest absolute eigenvalue $|\lambda_{max}| = \rho(\boldsymbol{W})$,
- spectral norm = largest singular value $s_{max}(\boldsymbol{W})$, relation: $0 \leq s_{max}(\boldsymbol{W}) \leq \rho(\boldsymbol{W})$

- Criteria for ESP: $s_{max}(\boldsymbol{W}) < 1 \rightarrow$ too strict, $\rho(\boldsymbol{W}) < 1$ not satisfctory
- $\rho(\boldsymbol{W}) \approx 1$ tends to optimize ESN behaviour (e.g. memory capacity)

Memory capacity (MC): reflects the ability to retrieve input data from the reservoir

- scalar i.i.d. inputs assumed, MC depends on $\boldsymbol{W}$, $\boldsymbol{W}^{inp}$,N

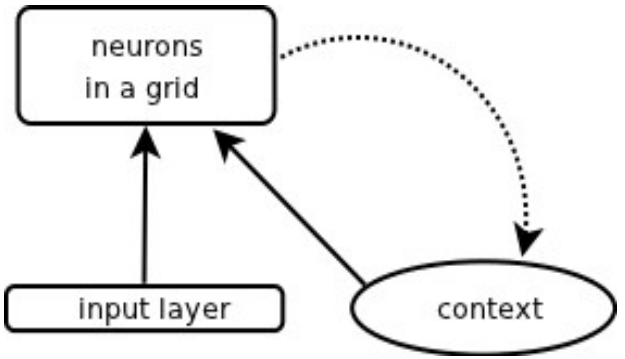
$$\mathrm{MC} = \sum_{k=1}^{k_{\max}} \mathrm{MC}_k = \sum_{k=1}^{k_{\max}} \frac{cov^2(u(t-k), y_k(t))}{var(u(t)) \cdot var(y_k(t))}$$

$$y_k(t) = \mathbf{w}_k^{\mathrm{out}} \mathbf{x}(t) = \tilde{u}(t-k)$$

$$k_{\max} = L$$

Reservoir stability – measured by characteristic Lyapunov exponent (Sprott, 2003).

SOMs for symbolic sequences



- (Markovian) map of suffixes

Neurons: $i = 1,2, ..., N$, input dim d Winner $i^* = \arg\min_j \{d_j(t)\}$, or $\{ \|\boldsymbol{d}_j(t)\| \}$

Weight update for: $\boldsymbol{w}_i$ (input weights) $\boldsymbol{c}_i$ (context weights, optional)

Temporal Kohonen map: $d_i(t) = a.\|\boldsymbol{x}(t) - \boldsymbol{w}_i\|^2 + (1-a)\, d_i(t-1)$

Recurrent SOM: $\boldsymbol{d}_i(t) = a.[\boldsymbol{x}(t) - \boldsymbol{w}_i] + (1-a)\, \boldsymbol{d}_i(t-1)$

Merge SOM: $d_i(t) = (1-a).\|\boldsymbol{x}(t) - \boldsymbol{w}_i\|^2 + a.\|\boldsymbol{r}(t) - \boldsymbol{c}_i\|^2$ $\boldsymbol{x}, \boldsymbol{r} \in R^d$

$\boldsymbol{r}(t) = b.\boldsymbol{w}_{i^*}(t-1) + (1-b)\, \boldsymbol{r}_{i^*}(t-1)]$

Recursive SOM: $d_i(t) = a.\|\boldsymbol{x}(t) - \boldsymbol{w}_i\|^2 + b.\|\boldsymbol{y}(t-1) - \boldsymbol{c}_i\|^2$ $y_i = \exp(-d_i)$

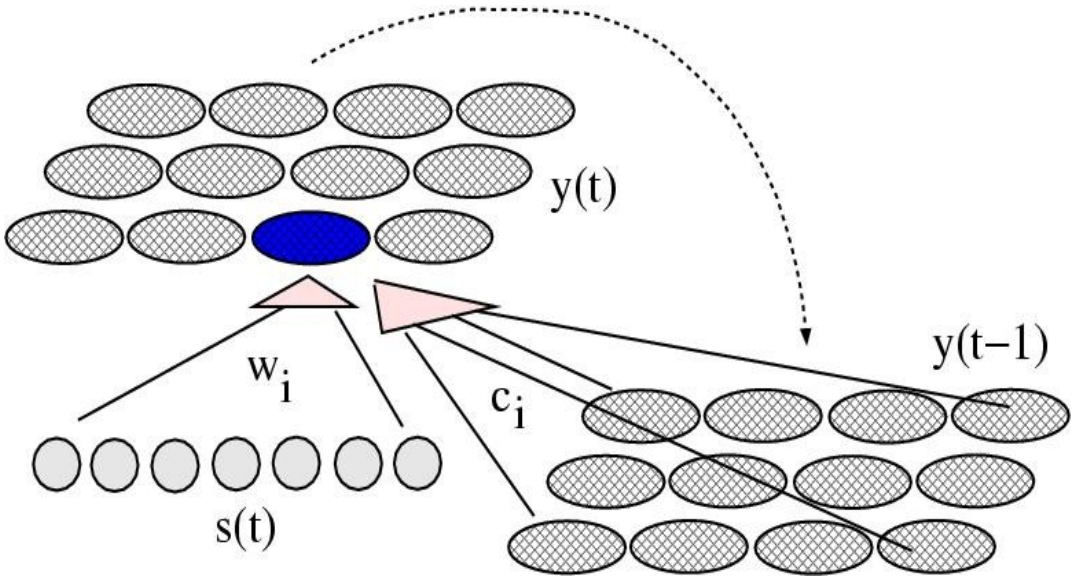
$\boldsymbol{c}_i, \boldsymbol{y} \in R^N$

SOMSD: $d_i(t) = a.\|\boldsymbol{x}(t) - \boldsymbol{w}_i\|^2 + b.\|\boldsymbol{p}_{i^*}(t-1) - \boldsymbol{c}_i\|^2$ $\boldsymbol{p}_{i^*}$ ~ winner coord in map

Recursive self-organizing map (RecSOM)

RecSOM can lead to more complex dynamics compared to other unsupervised models.
(Tiňo, Farkaš, van Mourik, 2006)

(Voegtlin, 2002)



Quantization error on unit *i*:

$$d_i = a\|\boldsymbol{s}(t)-\boldsymbol{w}_i\|^2 + b\|\boldsymbol{y}(t-1)-\boldsymbol{c}_i\|^2$$

Output of unit *i*:

$$y_i = \exp(-d_i)$$

Learning rules:

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + z\,h_{ik}[\boldsymbol{s}(t)-\boldsymbol{w}_i(t)]$$

$$\boldsymbol{c}_i(t+1) = \boldsymbol{c}_i(t) + z\,h_{ik}[\boldsymbol{y}(t-1)-\boldsymbol{c}_i(t)]$$

$0 < z << 1$ (constant) learning rate

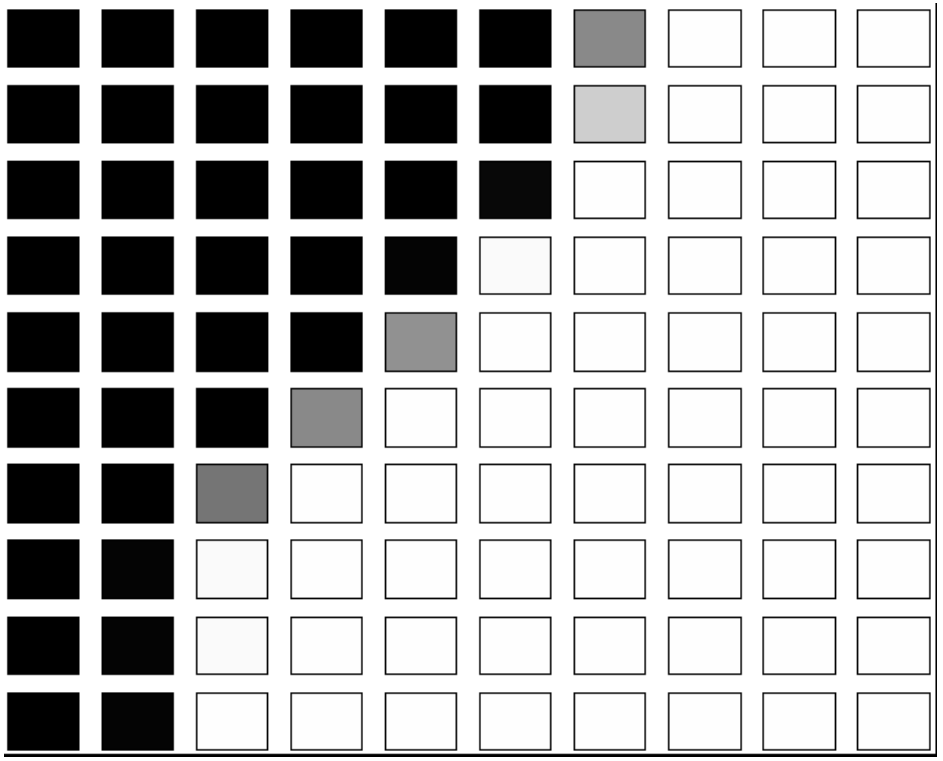
h – (shrinking) neighborhood function

RecSOM: trained on stochastic 2-state automaton: weights

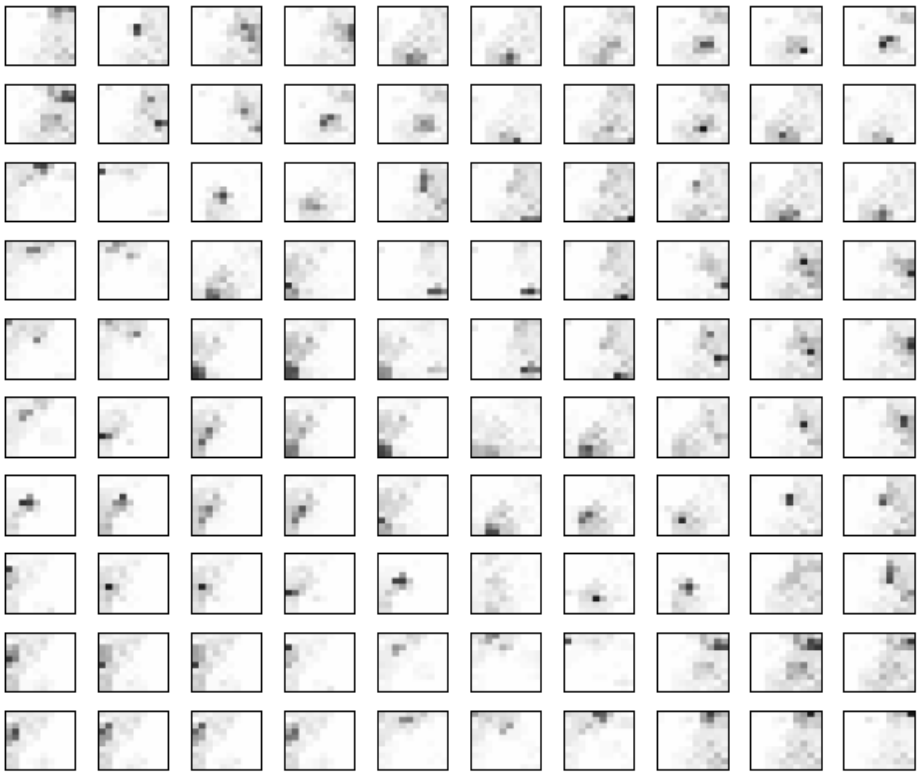
RecSOM: 10x10 units, 1D inputs

Input source: $P(b|a) = 0.3$, $P(a|b) = 0.4$

Input weights



Context weights



RecSOM: trained on stochastic 2-state automaton:
topographic map of suffixes

Summary

- two classes of architectures (time-lagged, partially or fully recurrent)
- time-lagged models are good for tasks with limited memory
- recurrent models with global feedback (via tapped-delay-lines) learn their internal state representations
- existing links to the theory of nonlinear dynamical systems, signal processing and control theory
- More complex learning algorithms: BPTT, RTRL (gradient-based)
- second-order neurons possible – higher computational power
- despite theoretical potential, difficulties to learn more complex tasks
- existence of architectural bias
- echo-state networks
- self-organizing recursive maps

9

Hopfield auto-associative memory

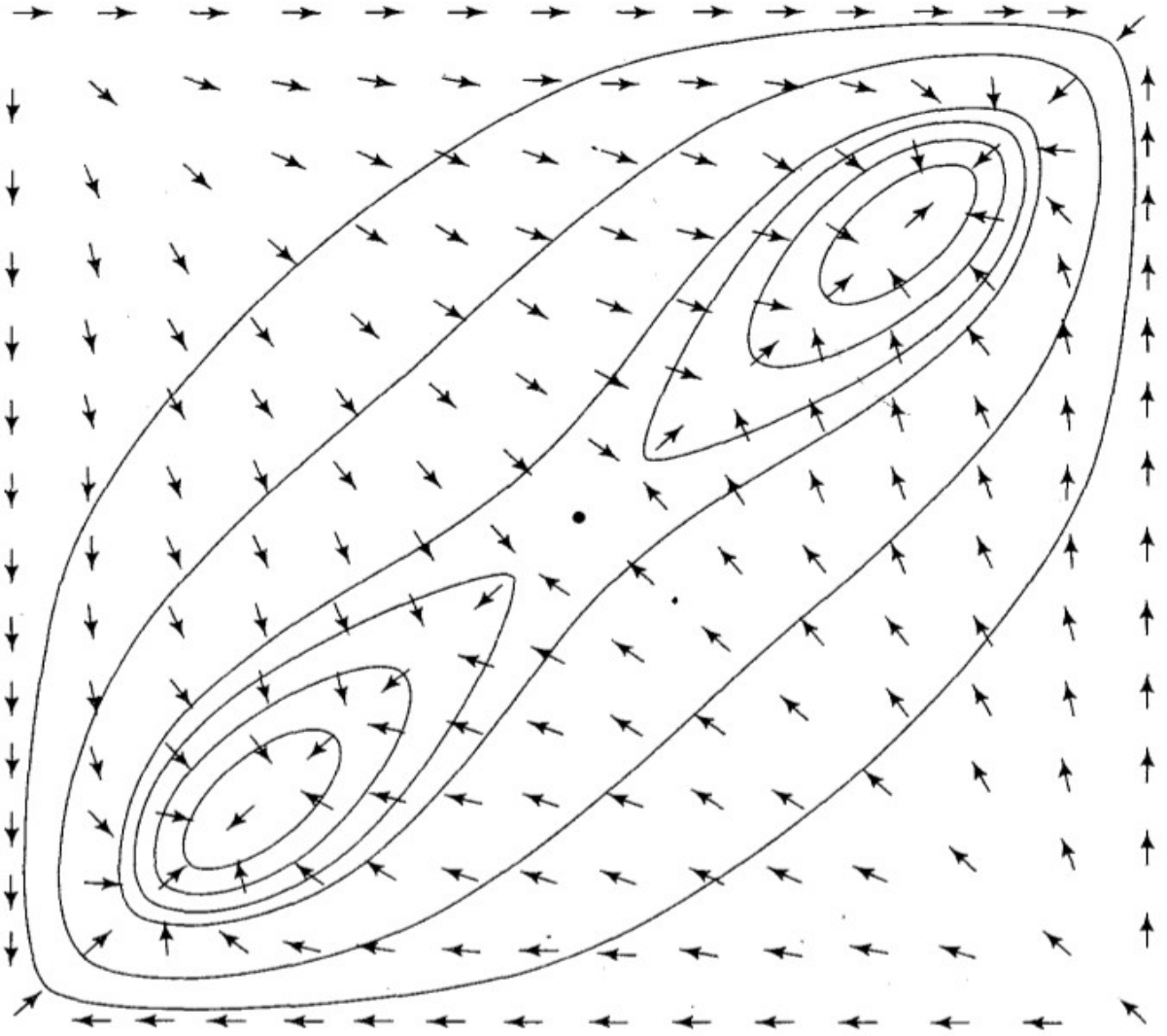
Introduction

- One class of recurrent models that can deal with sequential data are associative memories
- Key concept – stability (related to neurodynamics):
- The presence of stability implies some coordination among elements of the system
- bounded-input-bounded-output (BIBO) stability criterion
- Stability in nonlinear dynamic systems – in the sense of Lyapunov (1892)
- Neurodynamics: deterministic or stochastic dynamic systems

Dynamic systems

- State-space model – uses state variables that unfold in time
- the state $\boldsymbol{x}(t) = [x_1(t),\ x_2(t), ..., x_n(t)]^{\mathrm{T}}$, n = the order of system
- In continuous time: $d\boldsymbol{x}(t)/dt = \mathbf{F}(\boldsymbol{x}(t))$
- In continuous time: $\boldsymbol{x}(t+1) = \mathbf{F}(\boldsymbol{x}(t))$
- $\mathbf{F}$ is a vector function, whose each component is a nonlinear function (whose arguments are any elements of $\boldsymbol{x}$)
- System unfolding ~ trajectory in the state space
- State portrait ~ all trajectories superimposed
- Stability analysis – identification of equilibria

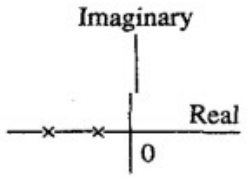
Example: State portrait of a dynamic system

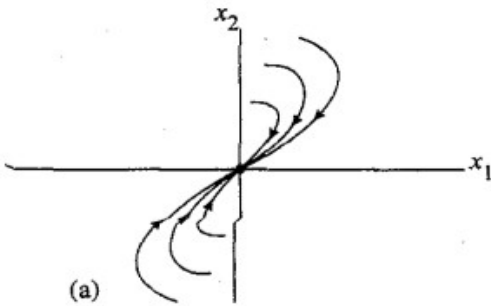


(Hopfield, 1984)

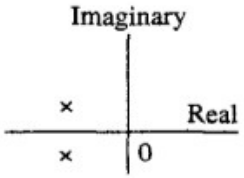
Attractor types in 2D space

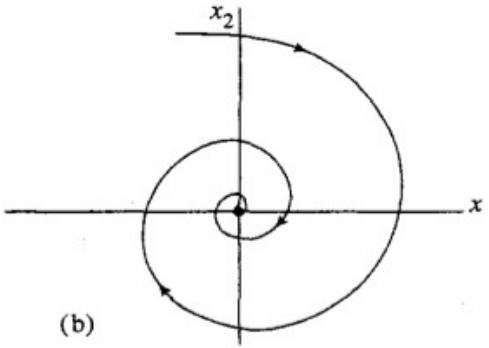
stable node



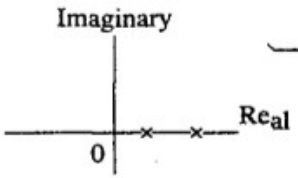


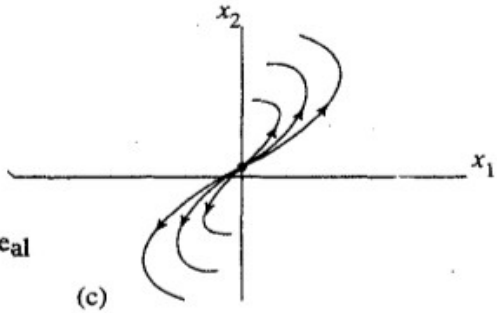
stable focus

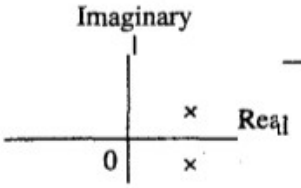


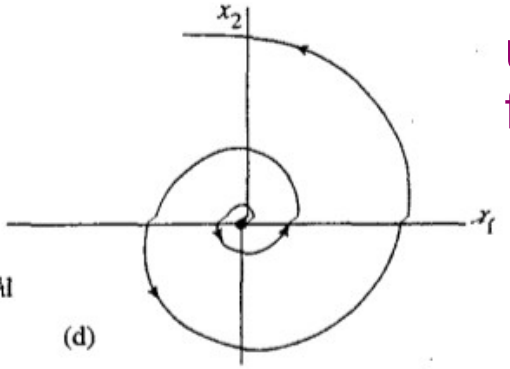


unstable node

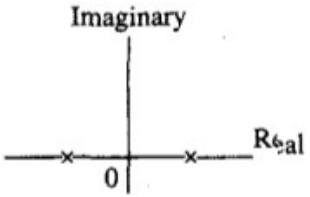


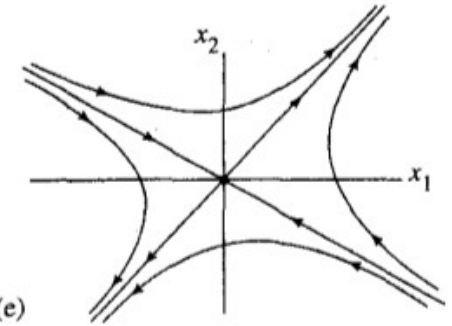




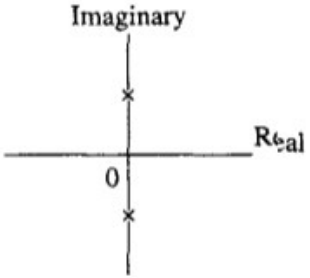


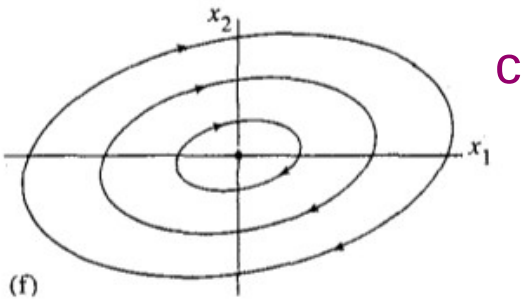
unstable focus





saddle point





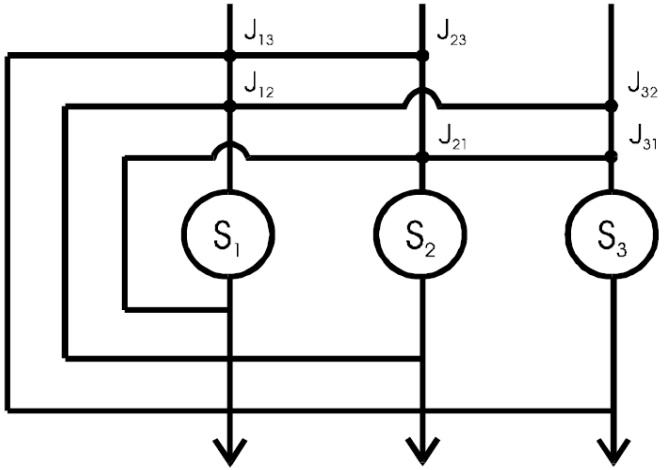
center

Towards Hopfield model

- physical inspiration (ordering states in magnetic materials)
- model of spin glasses (Kirkpatrick a Sherrington, 1978)
- Hopfield (1982) network:
  - content-addressable memory (“retrieve a pattern given a cue”)
  - an example of cellular automaton
- Attractive features of AAM:
  - model of a cognitive processing (attractors)
  - emergent behavior
- emphasis is on pattern retrieval dynamics, rather than learning

Hopfield model: basic concepts

- One (fully connected) layer with n neurons
- Neuron: two states $S_i \in \{-1,+1\}, \; i = 1..n$
- Configuration: $\mathbf{S} = [S_1, S_2, ..., S_n]$
- Weight: $J_{ij} \sim j \to i$, if $J_{ij} > 0$, then excitatory, if $J_{ij} < 0$, inhibitory, $J_{ii} = 0$



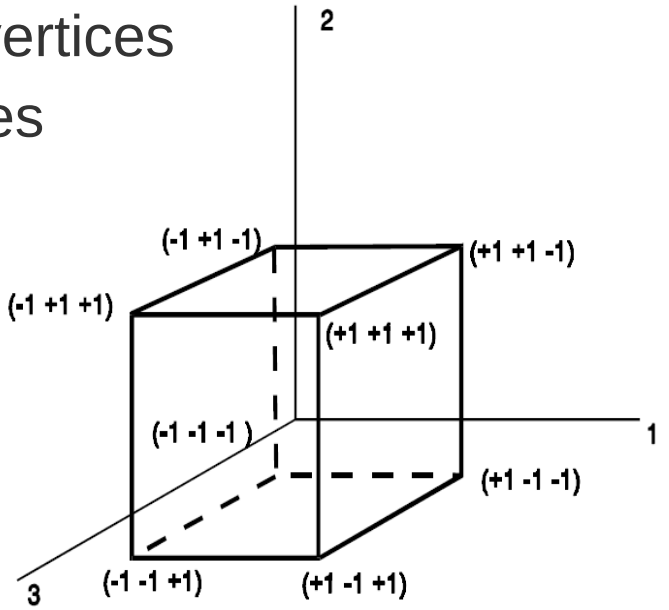
- Postsynaptic potential: $h_i^{\text{int}} = \sum_j J_{ij} S_j$ (~ internal magnetic field)
- Neuron excitation threshold: h_i^{ext} (~ external field)
- Effective postsynaptic potential: $h_i = h_i^{\text{int}} - h_i^{\text{ext}}$ (~ postsynaptic potential)
- Neuron state update (deterministic version): $S_i \to S_i' = \text{sgn}(h_i) \in \{-1,+1\}$
  if $h_i =\geq 0, \; \text{sgn}(h_i) = 1$, else -1.

Model dynamics and energy

- Synchronous (parallel): $S_i(t) = \mathrm{sgn}(\sum_{j \neq i} J_{ij} S_j(t\text{-}1) - h_i^{\mathrm{ext}}) \qquad \forall i$
  - one relaxation cycle = update of all neurons: $\mathbf{S}(t\text{-}1) \rightarrow \mathbf{S}(t)$
- Asynchronous (sequential): $S_i(t) = \mathrm{sgn}(\sum_{j \neq i} J_{ij} S_j(t\text{-}1) - h_i^{\mathrm{ext}}) \quad i \sim \mathrm{rnd}$
  - randomly chosen neurons (one at a time)

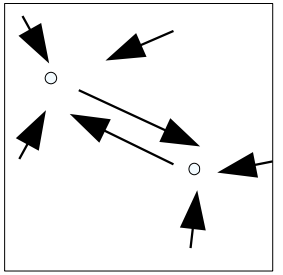
- Evolution of configuration: $\mathbf{S}(0) \rightarrow \mathbf{S}(1) \rightarrow \mathbf{S}(2) \rightarrow \ldots$ (relaxation process)
  - sync dynamics: trajectory over hypercube vertices
  - aSync dynamics: along the hypercube edges
- Energy of configuration: (state-dependant)

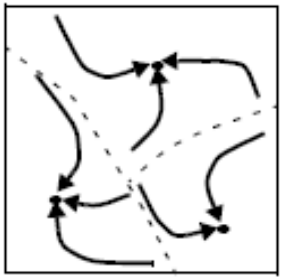
$$E(\mathbf{S}) = -\tfrac{1}{2} \sum_i \sum_j J_{ij} S_i S_j - \sum_i S_i h_i^{\mathrm{ext}}$$



Asymptotic behavior

- Randomly set initial config. $(\mathrm{P}_{\mathrm{bit}=1} = 0.5)$, $h_i^{\mathrm{ext}} \in [-1,1]$, $J_{ij} \in [-1,1]$
- Chaotic behavior: E rises and descends
  - typical par.: $\mathbf{J} \neq \mathbf{J}^{\mathrm{T}}$, syncDyn, arbitrary $h_i^{\mathrm{ext}} \in [-1,1]$
- Limit cycles: (with period 2, 4, ...)
  - typical par.: syncDyn (rarely for AsyncDyn)
- (Fixed) points: local minima of E
  - typical par.: $\mathbf{J} = \mathbf{J}^{\mathrm{T}}$, asyncDyn, $h_i^{\mathrm{ext}} = 0$
  - E descends only



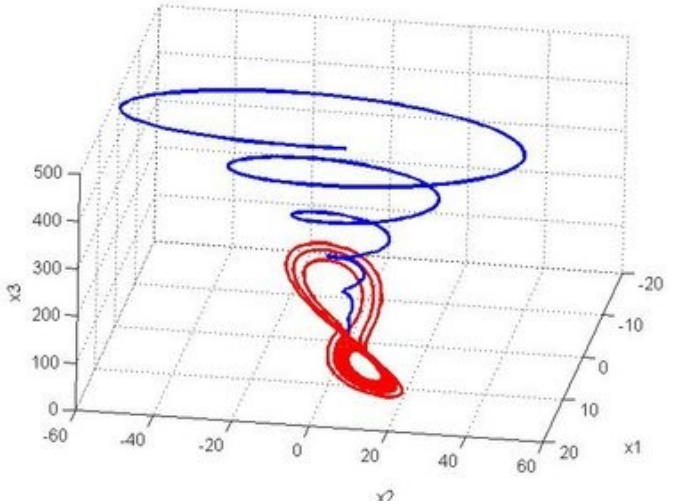


Asymptotic state: attractor dynamics

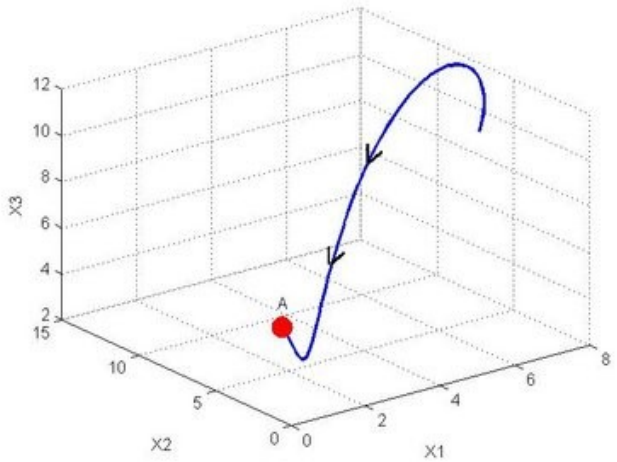
Depending on

- dynamics (a/synchronous)
- connectivity matrix (a/symmetric)

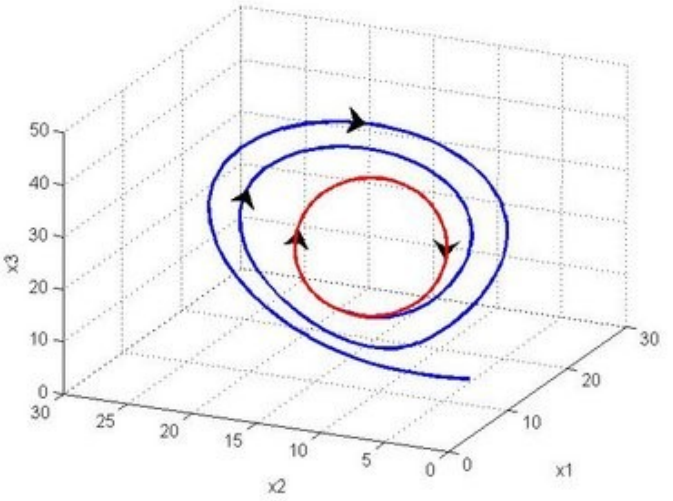
the network state ends up in
one of the three attractors



Chaotic attractor



Point attractor



Limit cycle attractor

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

Chaotic attractor: conditions

- Synchronous or asynchronous dynamics
- Arbitrary randomly generated excitation thresholds for $\forall\, i$:

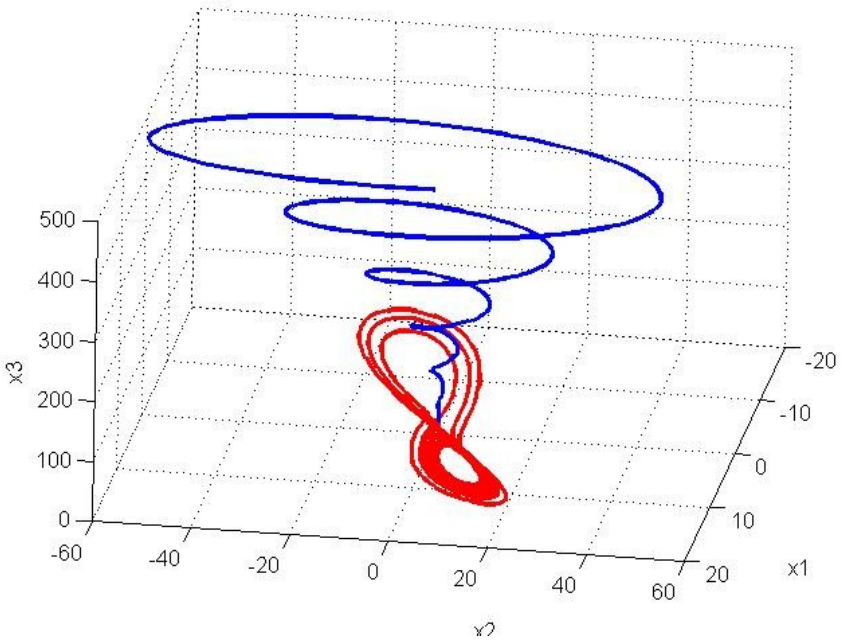
$$h_i^{\text{ext}} \in (-1,+1)$$

- Synaptic weights generated randomly for $\forall\, i,j$:

$$J_{ij} \in (-1,+1)$$

- Assymmetric connectivity, i.e. $\mathbf{J} \neq \mathbf{J}^{\mathrm{T}}$ (or $J_{ij} \neq J_{ji}$)
- E rises and descends

General example in a continuous state space



Example of a 3-dimensional chaotic attractor (Lorenz attractor) embedded in an N-dim. network. Example trajectory is shown in blue. Attractor manifold is shown in red. This manifold is bounded, though may not be repeatedly traversed in the stable state.

Sensitivity to the initial conditions

Limit cycle attractor: conditions

- Synchronous dynamics
- Arbitrary randomly generated excitation thresholds for $\forall\, i$:

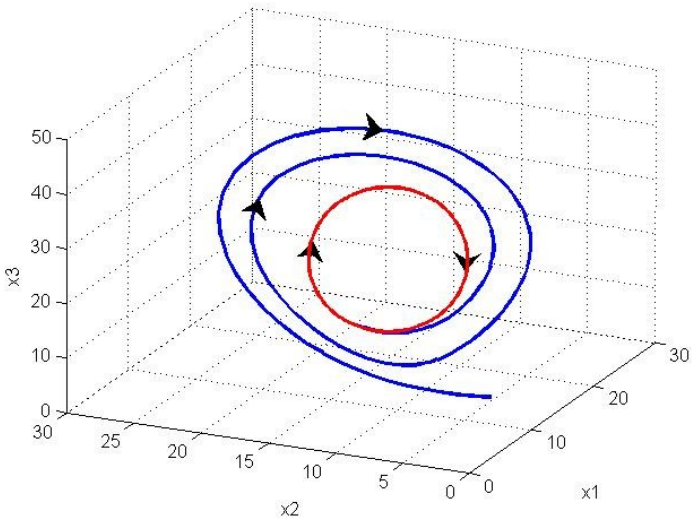
$$h_i^{\text{ext}} \in (-1,+1)$$

- Synaptic weights generated randomly for $\forall\, i, j$:

$$J_{ij} \in (-1,+1)$$

- Symmetric connectivity, i.e. $\mathbf{J} = \mathbf{J}^{\mathrm{T}}$ (or $J_{ij} = J_{ji}$)

General example in a continuous state space



Example of a 2-dimensional cyclic attractor embedded in an N-dim. network. Example trajectory is shown in blue. Attractor manifold is shown in red. This manifold is repeatedly traversed in the stable state.

Point attractor: conditions

- Asynchronous dynamics
- Zero excitation thresholds for $\forall\, i$:

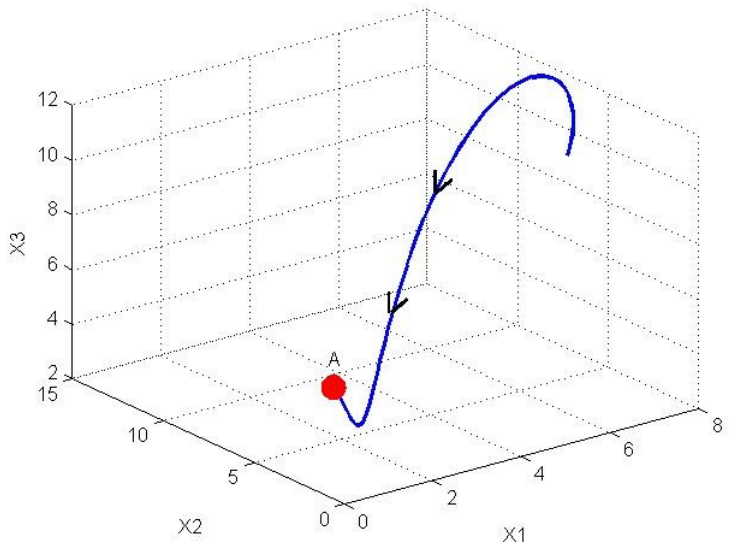
$$h_i^{\text{ext}} = 0$$

- Synaptic weights generated randomly for $\forall\, i, j$:

$$J_{ij} \in (-1, +1)$$

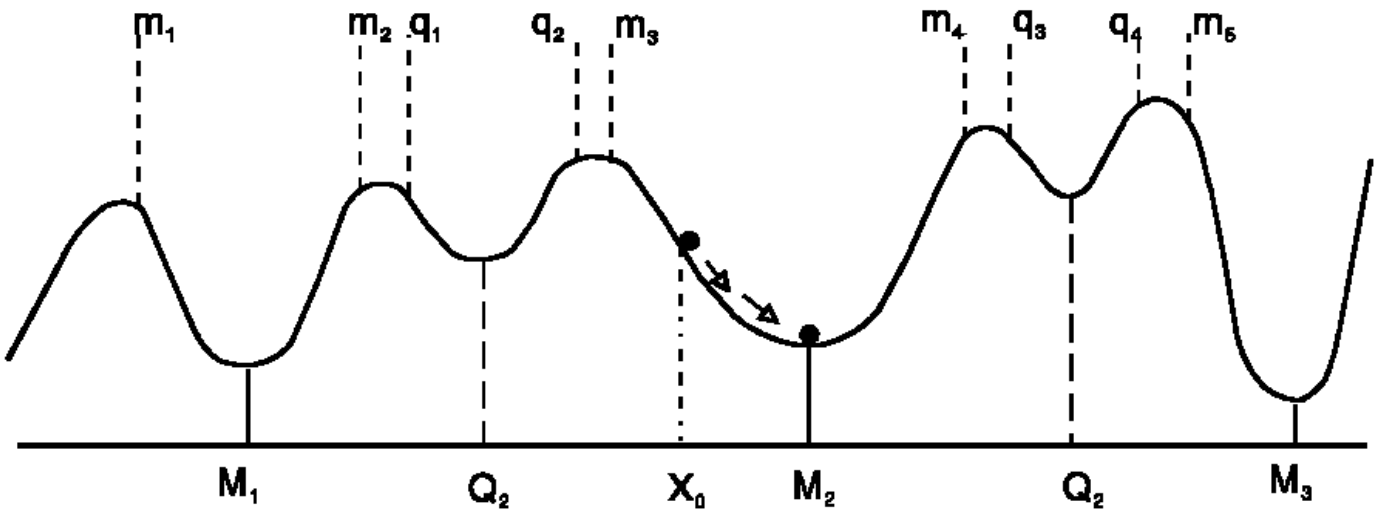
- Symmetric connectivity, i.e. $\mathbf{J} = \mathbf{J}^{\mathrm{T}} (\text{or } J_{ij} = J_{ji})$
- E only descends

General example in a continuous state space

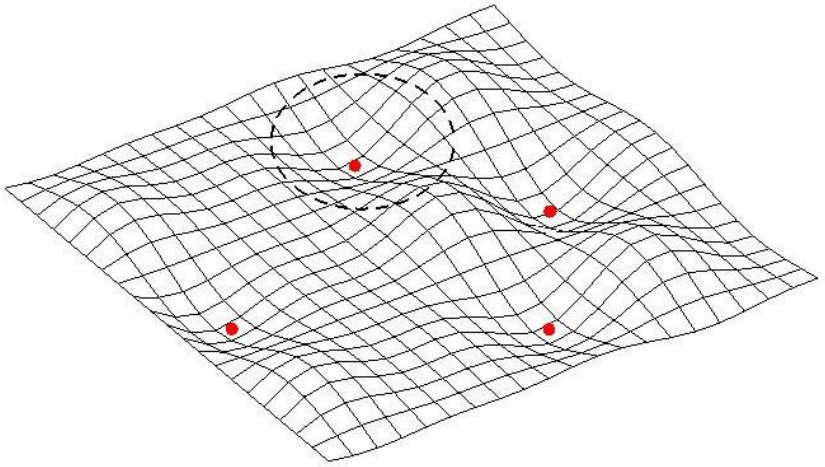


Insensitive to initial conditions, i.e. many different initial states (from the so-called basin of attraction) converge to the same point attractor.

Energy landscape



- basin of attraction, depends on **J**
- attractors: true (M_k), spurious (Q_l)
- energy decreases monotonously (for point attractors)
- spurious attractors – undesirable (linear combination of odd number of patterns)



Point attractor = minimum of energy

- Energy (Hamiltonian) of the state of Hopfield network is defined as:

$$E(\boldsymbol{s}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} J_{ij} S_i S_j - \sum_{i=1}^{n} S_i h_i^{ext}$$

- We can show that for asynchronous dynamics, zero excitation thresholds for $\forall\, i$, and symmetric connectivity, i.e. $\mathbf{J} = \mathbf{J}^{\mathrm{T}}$ energy always monotonically decreases during relaxation.

- Energy for $h_i^{\mathrm{ext}} = 0$ reduces to $E(\boldsymbol{s}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} J_{ij} S_i S_j$

- We need to show that the next state never has an energy *E'* higher than is *E* of the current state, i.e.

$$\Delta E = E' - E \leq 0$$

State evolution = energy minimisation

- Energy of the current state (with m-th unit taken out, and using $J_{mj}=J_{jm}$):

$$E(\boldsymbol{S})=-\frac{1}{2}\sum_{i\neq m}^{n}\sum_{j\neq m}^{n}J_{ij}S_iS_j-\frac{1}{2}\sum_{i=1}^{n}J_{mi}S_iS_m-\frac{1}{2}\sum_{j=1}^{n}J_{mj}S_mS_j=-\frac{1}{2}\sum_{i\neq m}^{n}\sum_{j\neq m}^{n}J_{ij}S_iS_j-\sum_{i=1}^{n}J_{mi}S_mS_i$$

- Energy of a new state obtained by updating a neuron m, i.e. $S_m \rightarrow S'_m$

$$E'=-\frac{1}{2}\sum_{i\neq m}^{n}\sum_{j\neq m}^{n}J_{ij}S_iS_j-\sum_{i=1}^{n}J_{mi}S_m^{'}S_i$$

- The difference reads:

$$\Delta E=E'-E=\left(S_m-S_m^{'}\right)\sum_{i\neq m}^{n}J_{mi}S_i=\left(S_m-S_m^{'}\right)h_m=-\Delta S_m h_m$$

Energy minimisation

- Energy difference reads: $\Delta E = E' - E = \left(S_m - S_m' \right) h_m$

- Reminder of the update rule: $S_m' = \mathrm{sign}\left(h_m \right) = \begin{cases} +1 & \text{for } h_m \geq 0 \\ -1 & \text{for } h_m < 0 \end{cases}$

- There are 4 possibilities for the change of the state of neuron *m*:

$$S_m = +1 \wedge S_m' = +1 \left(when \;\; h_m > 0 \right) \;\Rightarrow\; \Delta E = 0$$

$$S_m = -1 \wedge S_m' = -1 \left(when \;\; h_m < 0 \right) \;\Rightarrow\; \Delta E = 0$$

$$S_m = +1 \wedge S_m' = -1 \left(when \;\; h_m < 0 \right) \;\Rightarrow\; \Delta E < 0$$

$$S_m = -1 \wedge S_m' = +1 \left(when \;\; h_m > 0 \right) \;\Rightarrow\; \Delta E < 0$$

Q.E.D.

HN as an autoassociative memory

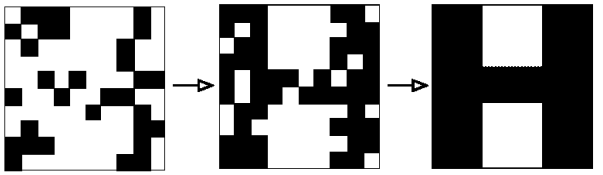
- It is possible to design the connectivity matrix **J** in such a way that only the chosen states (memory states) become the point attractors.
- Point attractors are stationary states, which means that once the network gets into one of them then the network stays there forever.
- Each attractor has a basin of attraction, i.e. a (big) region of initial states, that all converge to the same point attractor. It is possible to design **J** in such a way that only the states that are similar to the attractor (the memory) will converge into it.
- This means that the network will find the memory state even when it is presented with a noisy, incomplete or damaged version of it.
- This is called an auto-associative or content-addressable memory.

Autoassociative memory

- Point attractors = stationary states ⇔ memorized patterns
- Content-addressable memory
- Assume (binary) patterns: $\boldsymbol{x}^{(p)} = [x_1^{(p)}, x_2^{(p)}, ..., x_n^{(p)}]$ $p = 1...N$
- Set weights: $J_{ij} = 1/n \sum_p x_i^{(p)} x_j^{(p)}$ for $i \neq j$, $J_{ii} = 0$
- $J_{ij} \in \{-N/n , ..., 0, ..., N/n\}$
- Recall (retrieval) of pattern $\boldsymbol{x}^{(r)}$ occurs ⇔ $\mathbf{S}(0) \rightarrow \ \ldots \ \rightarrow \boldsymbol{x}^{(r)}$
- Stability requirement for $\boldsymbol{x}^{(r)}$: $x_i^{(r)} . h_i^{(r)} > 0$ for $i = 1..n$
- $x_i^{(r)} . h_i^{(r)} = x_i^{(r)} \sum_j J_{ij} x_j^{(r)} = ... = 1 + C_i^{(r)} > 0$

in limit for large n

crosstalk



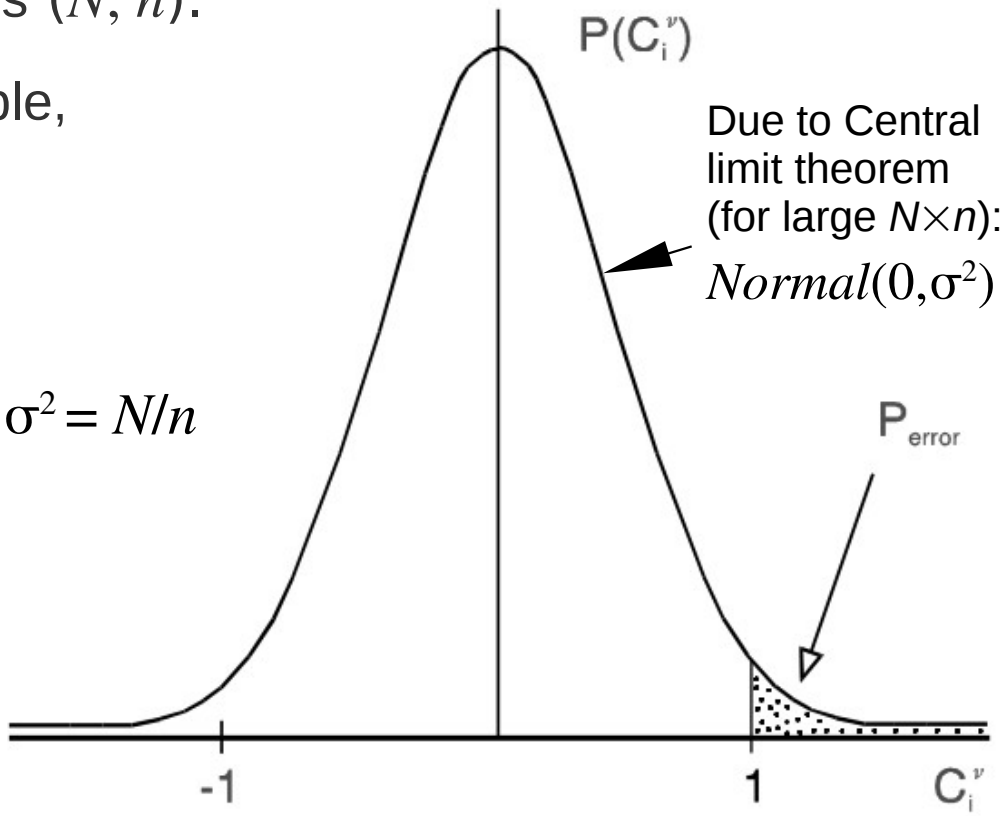
Memory capacity

- for orthogonal patterns $C_i^{(r)} = 0 \Rightarrow N_{\max} = n$
- for pseudoorthogonal patterns: i.e. if $\langle \boldsymbol{x}^{(p)}.\, \boldsymbol{x}^{(r)} \rangle \approx 0 \wedge |C_i^{(r)}| < 1$ (stability)
  - What's the capacity in this case?
- Treat pattern bits as random variables (N, n).
- What's the prob. that *i*-th bit is unstable,

  i.e. $P_{\text{error}} = P(C_i^{(r)} > 1)$?
- for large N and n,

  $C_i^{(r)} \sim$ binomial distrib. $Bin(0,\sigma^2)$
- $Bin \approx Normal(0,\sigma^2)$

$\sigma^2 = N/n$

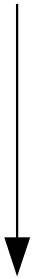


(Beňušková, 1997)

Memory capacity (ctd)

- Relationship between P_{error} and capacity:

P_{error}	$N_{\max}/n$
0.001	0.105
0.0036	0.138
0.01	0.185
0.05	0.37
0.1	0.61



increasing blackout in retrieval

- Stable memorized states are
  - true attractors
  - reverse configurations
  - spurious states

Stochastic Hopfield model

- How to get rid of spurious attractors?
- Introduction of noise into the model
  - more biologically plausible
  - narrows down basins of attraction of spurious attractors
- Interpretation from statistical physics: noise (T)
- Probability distribution of states: $P(\mathbf{S}) = 1/Z \exp(-\beta E(\mathbf{S}))$,
- Normalization term $Z = \sum_{\mathbf{S}'} \exp(-\beta E(\mathbf{S}'))$, $\beta = 1/T$, inverse temperature
- Transition probability $P(\mathbf{S} \rightarrow \mathbf{S}') = 1/(1+\exp(\beta \Delta E))$ $\quad \Delta E = E(\mathbf{S}') - E(\mathbf{S})$
- Transitions to states with higher energy possible (but less likely)
- Link to simulated annealing (Kirkpatrick, Gellat, Vecchi, 1983; Černý, 1985)
  - Mixed cooling down and heating (exploration)

Stochastic Hopfield model (ctd)

- Stochastic rule that unit changes its state

  $P(S_m = \pm 1) = 1/(1+\exp(-2\beta h_m S_m))$ (m-th bit was changed)

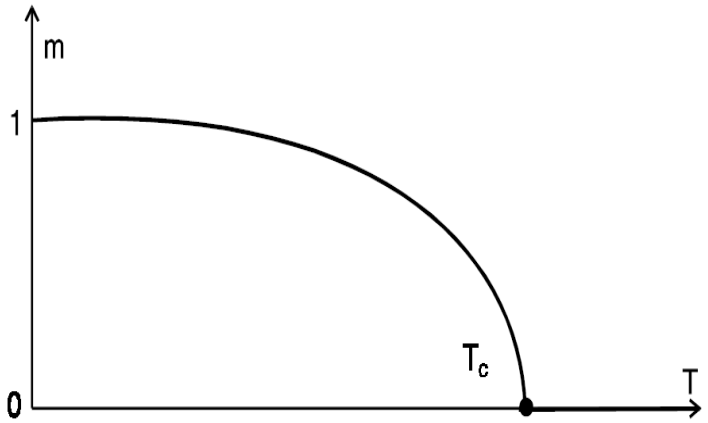
- Then $P(S_m = +1) = 1/(1+\exp(-2\beta h_m))$
- and $P(S_m = -1) = 1 - P(S_m = +1)$
- probabilities of transitions depend on T
- For $T \to 0$ we get deterministic model
- For $T \to \infty$ we get an ergodic system, $P(S_m = +1) = 0.5, \quad m = 1 \ldots n$
  - no stable memories

Memory stability of stochastic model

- Consider $N \ll n$, stochastic model, $h_i^{\text{ext}} = 0$: (Amit et al., 1985)
- mean-field approximation => self-consistent equations:

$$\langle S_i \rangle = \tanh \sum_j J_{ij} \langle S_j \rangle, \quad \forall i, \langle S_j \rangle = \sum_j S_j P(S_j) = 1.P(S_j=+1) + (-1).P(S_j=-1)$$

- assumption for solution: $\langle S_j \rangle = m.x_j^{(r)}, \quad \forall j, \quad m \in \Re$
- leads to equation: $m = \tanh(\beta m)$ for $N \ll n$
  - for $T > T_c$: no pattern is stable ($T_c = 1$)
  - each spurious state has its critical temperature $0 < T_{sc} \leq 0.46$ =>
  - for $0.46 < T < T_c$: all spurious attractors become destabilized



Properties of stochastic model

- only true attractors can remain stable
- lower accuracy of retrieved memorized patterns
  - overlap $m^{(p)} = 1/N \sum_j x_j^{(p)} S_j^{(p)}$ with one of the patterns > 0.9
  - we talk about probability distribution of attractor states
- longer relaxation times
- psychologically and neurobiologically more plausible

- Yet another version: analogue Hopfield model (diff. equations)

Applications

- Modeling neurobiological and psychological effects:
  - autoassociative pattern recall (given a cue)
  - recognition of sequences (relaxation times in cognitive modeling)
  - generation of sequences (e.g. melodies)
- Optimization problems:
  - combinatorial
    - e.g. traveling salesman problem
  - image processing (filtering) – reconstruction of an image from its noisy version

Summary

- Hopfield's (1982) work has had a significant impact on NNs
- symmetric weights – novel feature introduced
- Defining network state in terms of energy – to be minimized
- Hopfield model shows that it is possible for a structured behavior to emergent from evolution of a complex, nonlinear dynamic system over time.
- In context of autoassociative memory, stochastic model can overcome the limitations of the deterministic version, by properly destabilizing spurious attractors.

References

- Haykin S.: *Neural Networks and Learning Machines*, Prentice Hall, 2009.
- Kvasnička V., Beňušková Ľ., Pospichal J., Farkaš I., Tiňo P., Král A.: *Introduction to the Theory of Neural Networks.* IRIS, Bratislava, 1997.
- Oravec M., Polec J., Marchevský S.: *Neurónové siete pre číslicové spracovanie signálov*. Faber Bratislava, 1998.