DD2437 – Artificial Neural Networks and Deep Architectures (annda)

Course summary

Pawel Herman

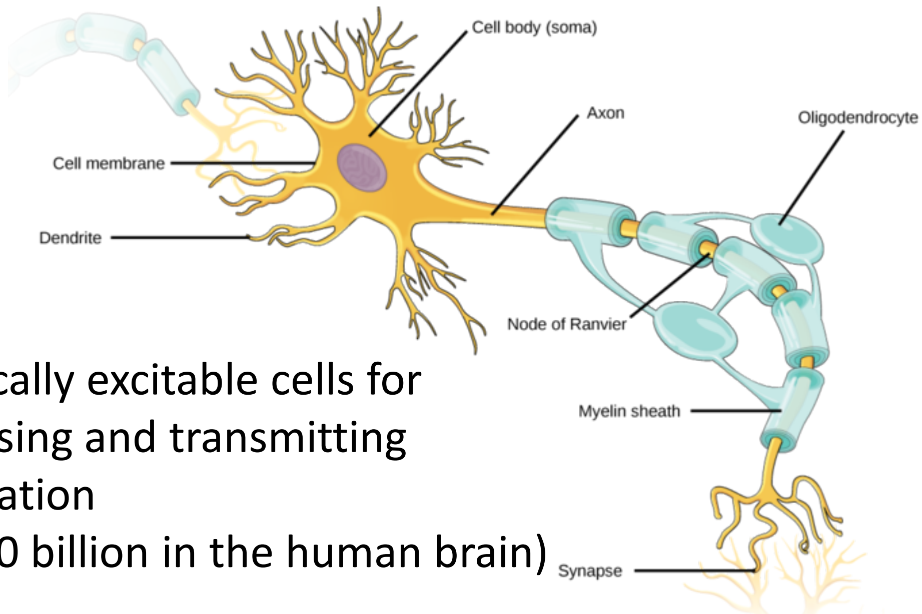
Computational Science and Technology (CST)

KTH Royal Institute of Technology

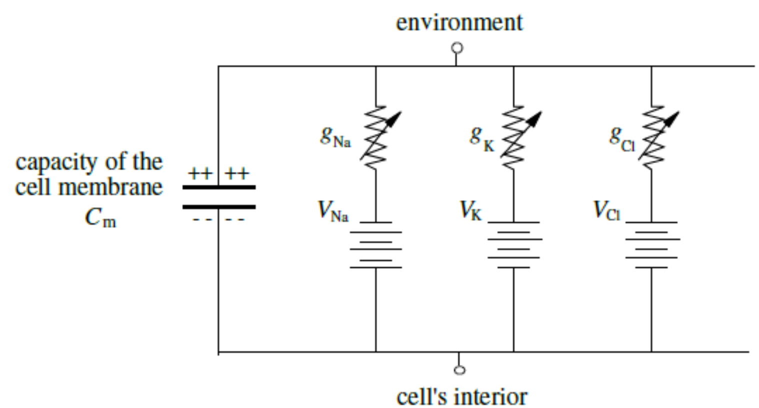
March 2018

Biomimetic nature of ANNs

Inspirations from biology

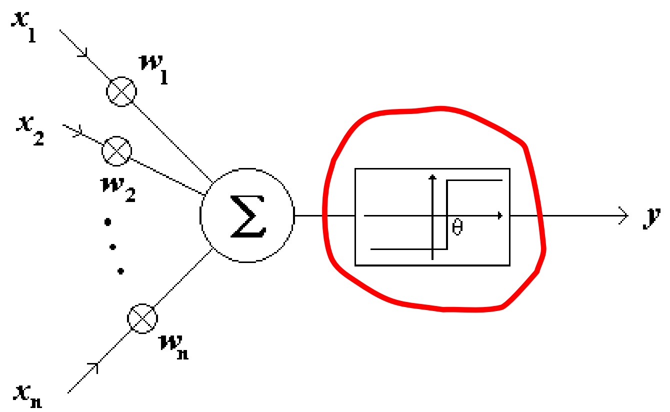


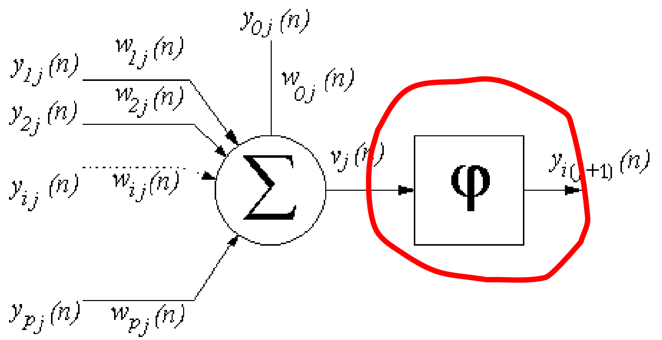
electrically excitable cells for processing and transmitting information
(ca. 100 billion in the human brain)



Fundamental characteristics

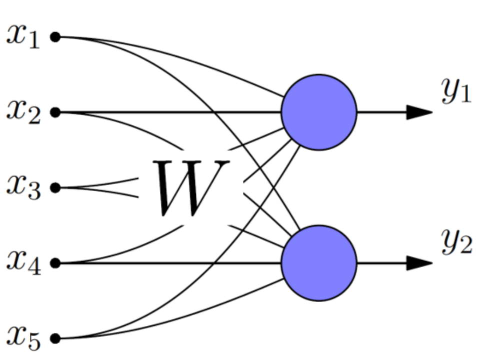
- nodes, units
- **activation function**
- learning rule
- topology, network architecture
- data

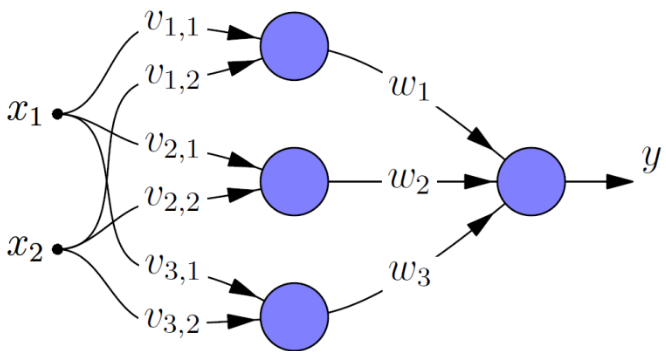




Linear networks

What can be computed?





$$y = \vec{w}^{\mathrm{T}} \cdot \vec{x}$$

$$y = \mathrm{W} \cdot \vec{x}$$

$\vec{w}$ - weight vector

W - weight matrix

Storing mappings (memorising)

Storing a mapping using Hebb's rule

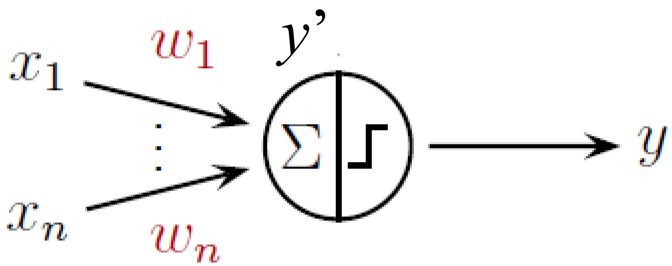
$$\vec{x}_1 \rightarrow \vec{y}_1 \qquad \vec{x}_2 \rightarrow \vec{y}_2 \qquad \vec{x}_3 \rightarrow \vec{y}_3 \quad \ldots \quad \vec{x}_n \rightarrow \vec{y}_n$$

Hebb's rule $\Delta w_{ij} = x_i y_j$

Result $\mathrm{W} = \sum_{p=1}^{n} \vec{y}_p \cdot \vec{x}_p^{\mathrm{T}}$

Correlational memory!

Threshold in TLU



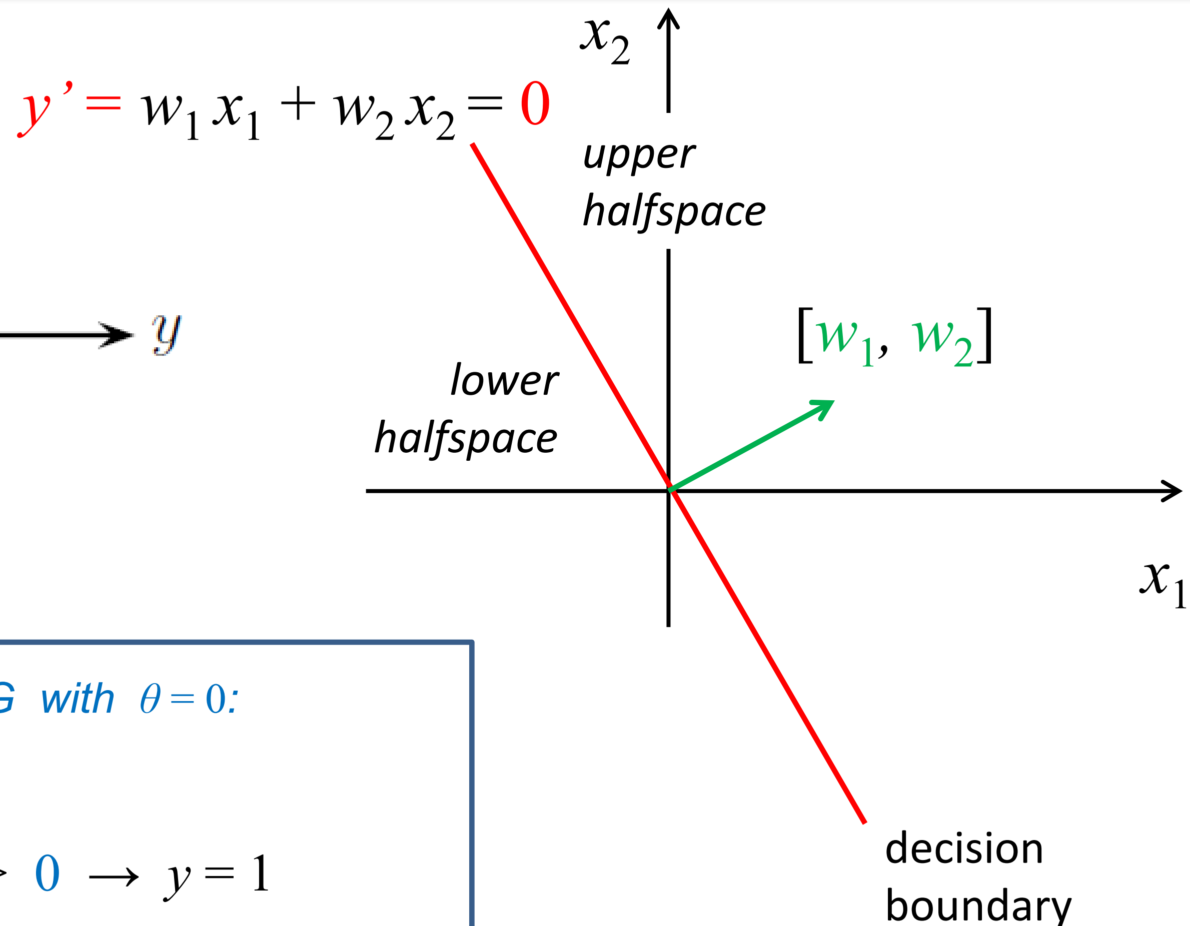
$$y' = w_1 x_1 + w_2 x_2$$

THRESHOLDING with $\theta = 0$:

$$y = f_{step}(y')$$

$$w_1 x_1 + w_2 x_2 > 0 \rightarrow y' > 0 \rightarrow y = 1$$

$$w_1 x_1 + w_2 x_2 <= 0 \rightarrow y' <= 0 \rightarrow y = 0$$



Perceptron learning for classification

Training of a Thresholded Network: Perceptron Learning

Basic Principle: Weights are changed whenever a pattern is erroneously classified

When the result $= 0$, should be $= 1$

$$\Delta \vec{w} = \eta \vec{x}$$

When the result $= 1$, should be $= 0$

$$\Delta \vec{w} = -\eta \vec{x}$$

Delta rule

Delta rule (Widrow-Hoff rule, ADALINE)

1. Symmetric target values: {-1, 1}
2. Error is measured before thresholding

$$e = t - \vec{w}^{\mathrm{T}} \vec{x}$$

3. Find weights that minimise the error cost function

$$\varepsilon = \frac{e^2}{2}$$

Training of thresholded single-layer networks

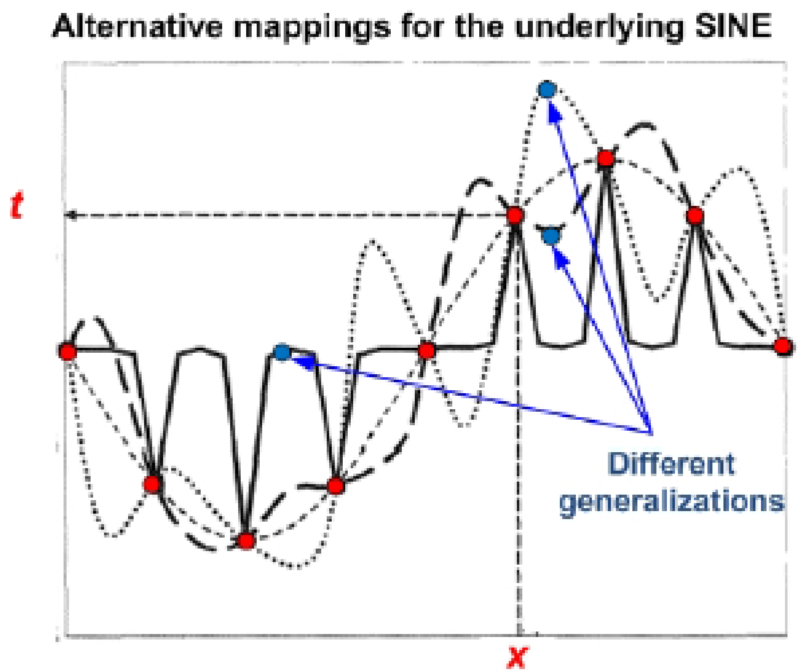
Perceptron learning:

$$\Delta\vec{w} = \eta e \vec{x} \qquad \text{where} \quad e = t - y$$

Delta rule:

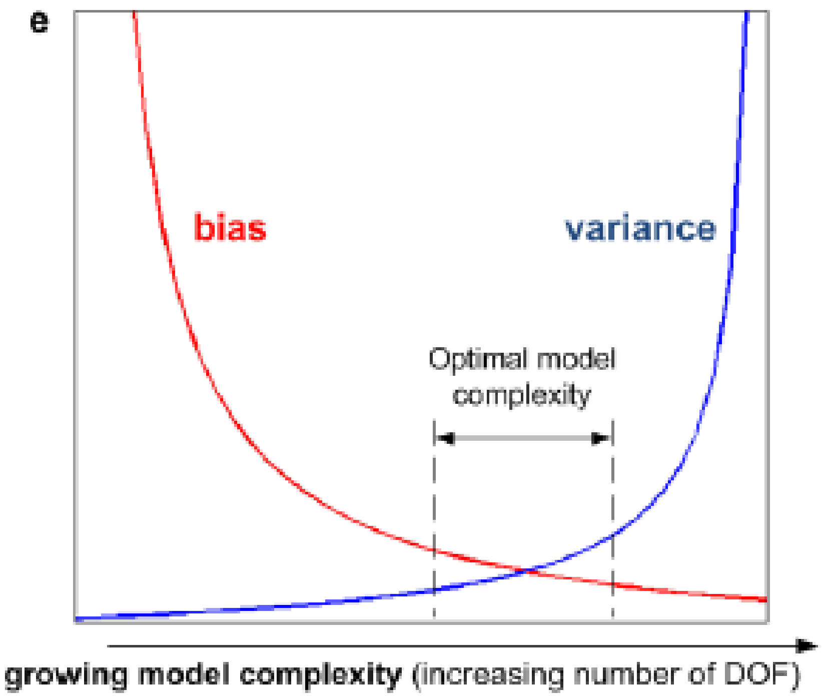
$$\Delta\vec{w} = \eta e \vec{x} \qquad \text{where} \quad e = t - \vec{w}^{\mathrm{T}}\vec{x}$$

Generalisation and overfitting phenomenon



- network memorises the training data (noise fitting)
- instead it should learn the underlying function
- on the other hand, the danger of underfitting/undertraining

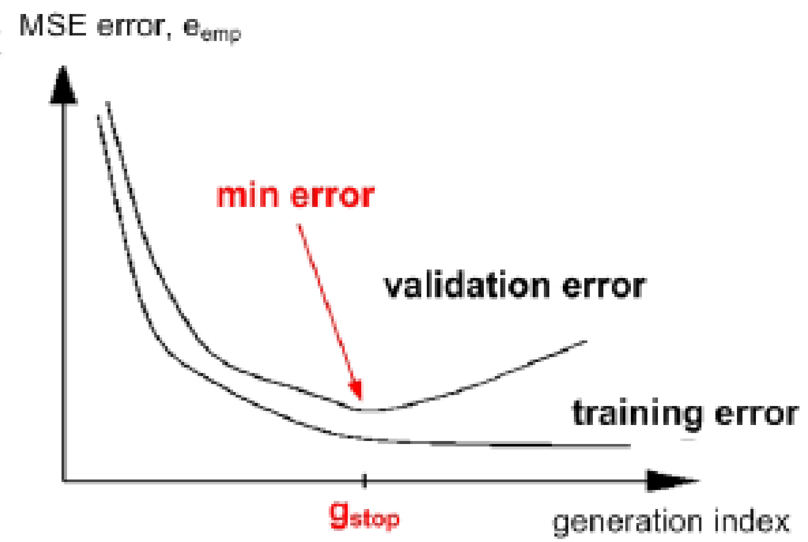
Bias and variance trade-off



- the problem can be alleviated by increasing training data size
- otherwise, the complexity of the model has to be restricted
- for NNs, identification of the optimal network architecture

Early stopping

- An additional data set is required - **validation set** (split of original data)
- The network is trained with BP until the error monitored on the validation set reaches minimum (further on only noise fitting)

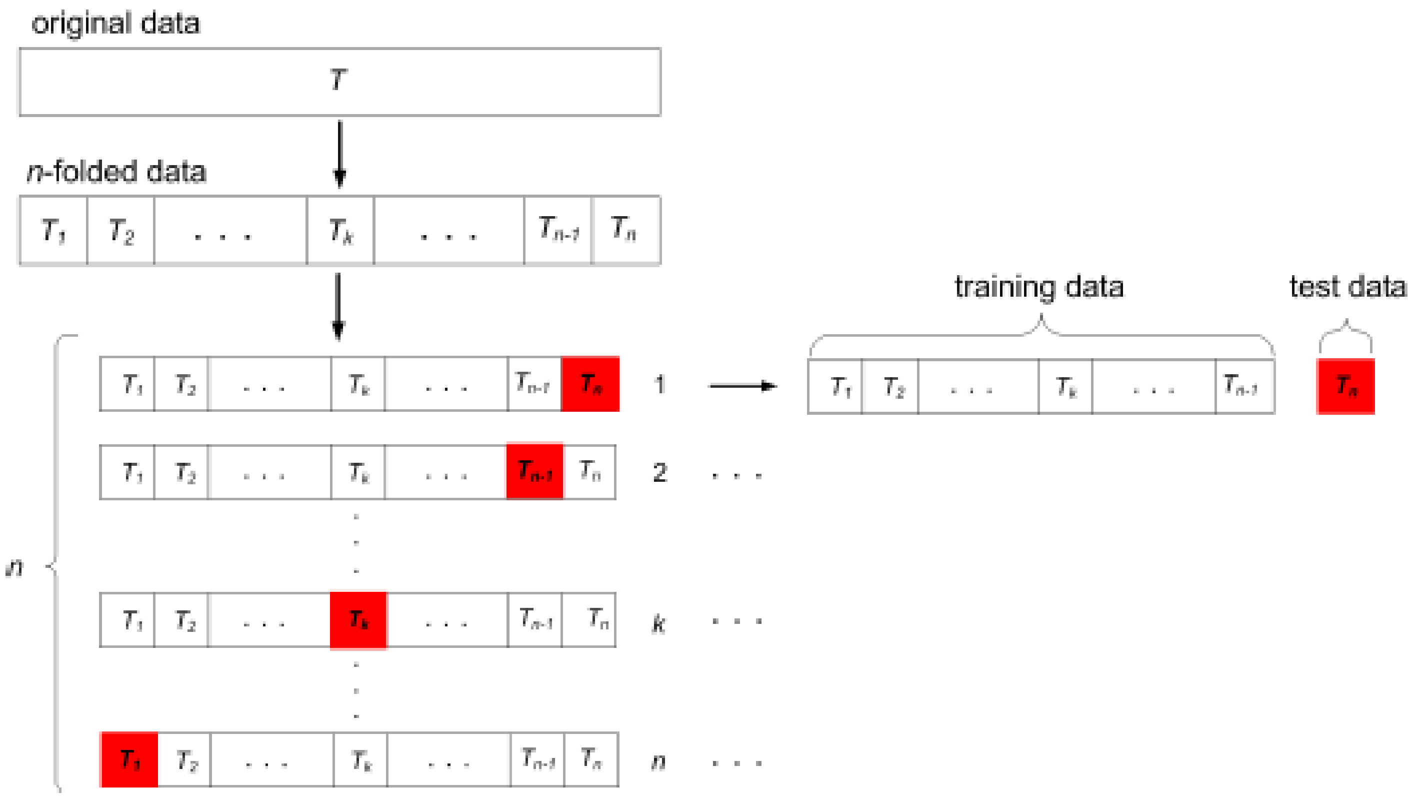


- For quadratic error, it corresponds to learning with weight decay

Model validation and selection

- Empirical assessment of generalisation capabilities
  - allows for model verification, comparison and thus selection
  - separate training and test sets - the simplest approach
- Basic ***hold-out*** (also referred to as *cross-validation*) method often relies on 3 sets and is commonly combined with early stopping
- The cost of sacrificing original data for testing (>10%) can be too high for small data sets

N-fold crossvalidation



$$E_{\text{true}}^{\text{CV}} = \frac{1}{n} \sum E_{\text{emp}}^{T \setminus T_k \to T_k}$$

Regularisation

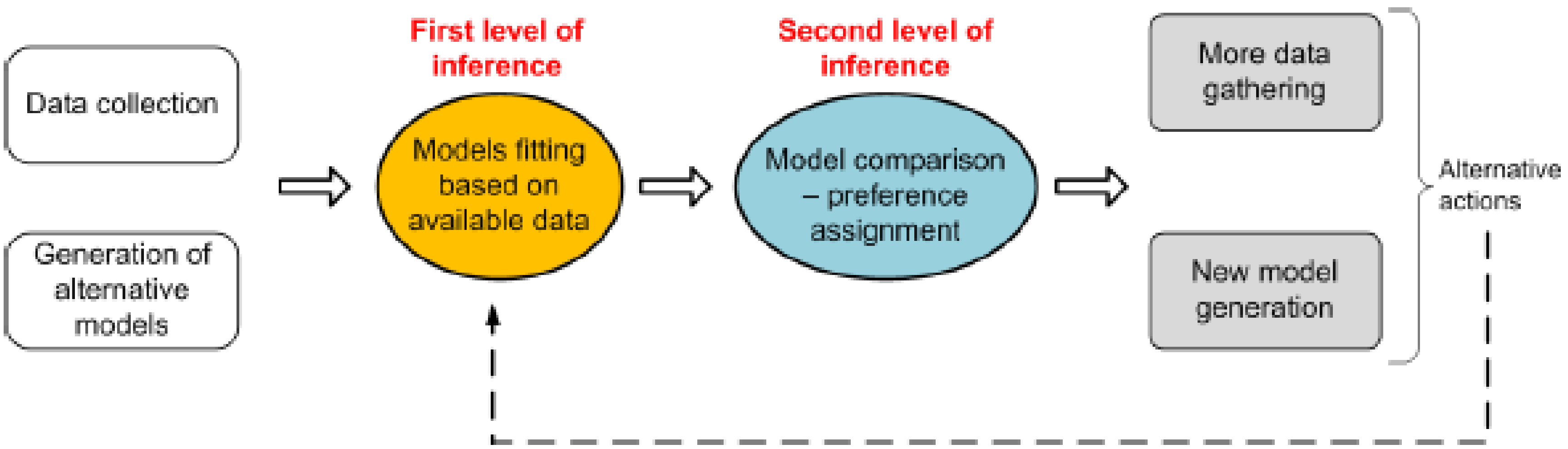
- Regularisation as an approach to controlling the complexity of the model
  - constraining an ill-posed problem (existence, uniqueness and continuity)
  - striking bias-variance balance (SRM)

- Penalised learning (penalty term in the error function)

$$\tilde{E} = E + \lambda \Omega$$

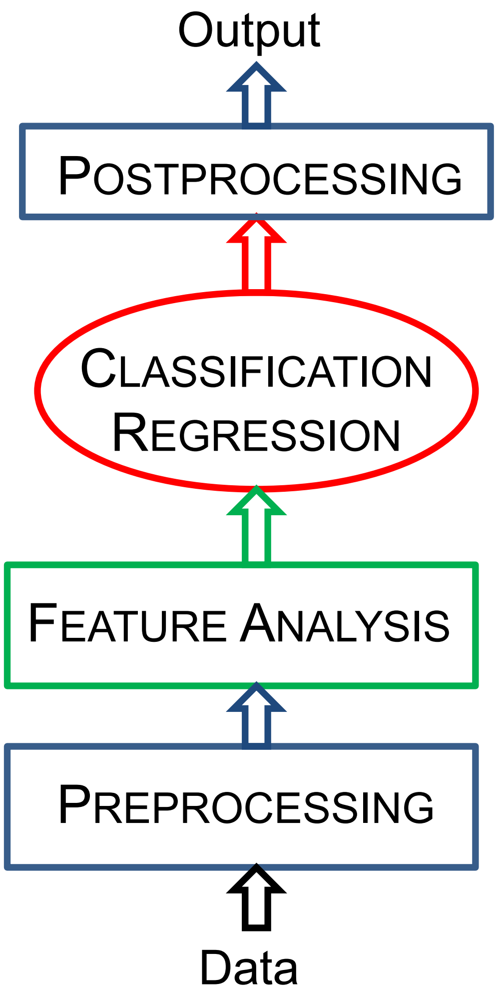
  - trade-off controlled by the regularisation parameter λ
  - smooth stabilisation of the solution due to the complexity term
  - in classical "penalised ridging", $\Omega = \frac{\partial^n Y}{\partial \mathbf{w}^n} \quad \left(\|\mathbf{D} Y\|^2 \right)$

Bayesian regularisation



- model fitting (weights in ANNs) based on likelihood function and priors (1st level)
- model comparison by evaluating the evidence (2nd level)

Pattern recognition pipeline



1. Preprocessing
2. Features, low-level data representation
3. Classification / regression with ANN
4. Postprocessing (alternative)

Error measures – performance metrics

- Decide on the target measure of performance (potentially related to key performance indicators) and specific metric
  - sum square error (with or without normalisation), root-mean-square
  - accuracy for classification tasks
  - precision, recall, ROC curve (area under the curve, AUC)
  - F-score: $\boldsymbol{F = 2pr / (p+r)}$, where: p - precision, r – recall
- More advanced measures
  - weighted errors, e.g. weighted sum of squares
  - probabilistic measures for classification, e.g. cross-entropy for two or multiple classes (if the output represents probabilities by *softmax* activation)

Outline of optimisation algorithms

Beyond gradient descent

- Extensions to gradient descent
- Linear search methods
- Conjugate gradients (+ scaled conjugate gradients)
- Newton's method (making explicit use of Hessian) and quasi-Newton approach
- The Levenberg-Marquardt algorithm

Committee of networks

- Basic idea: combine weak learners and boost performance
- Concept in opposition to best model selection
- Question of extra computational effort
- Key questions:
  - Which learners? How to train them, on what data?
  - How to combine learners?

Ensemble methods – simple averaging

Model averaging as a general strategy for ensemble methods

The expected square error of the ensemble:

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c.$$

where: k – the number of weak learners

ε_i – error committed by the i-th learner (MVN(0, C))

C is defined by $\mathbb{E}\left[\varepsilon_i^2\right] = v, \quad \mathbb{E}\left[\varepsilon_i\varepsilon_j\right] = c$

If the errors are uncorrelated, i.e. $c=0$:

$$E_{COM} = \frac{1}{k}v = \frac{1}{k}\mathbb{E}\left[\varepsilon_i^2\right] = \frac{1}{k}\left(\frac{1}{k}\left(E_{INDIV}^{(1)} + \ldots + E_{INDIV}^{(k)}\right)\right) = \frac{1}{k}\overline{E}_{INDIV}$$

Ensemble approaches

Static approaches that do not account for input

- ensemble averaging, bagging
- boosting

Approaches dependent in input

- mixture of experts
- hierarchical mixtures

The radial-basis-function (RBF) technique

Nonlinear mapping with the use of *radial-basis-functions* (RBFs):

$$\varphi_i\left(\left\|\mathbf{x}-\mathbf{x}_i\right\|\right)$$

$\mathbf{x}_i$ – RBF centre

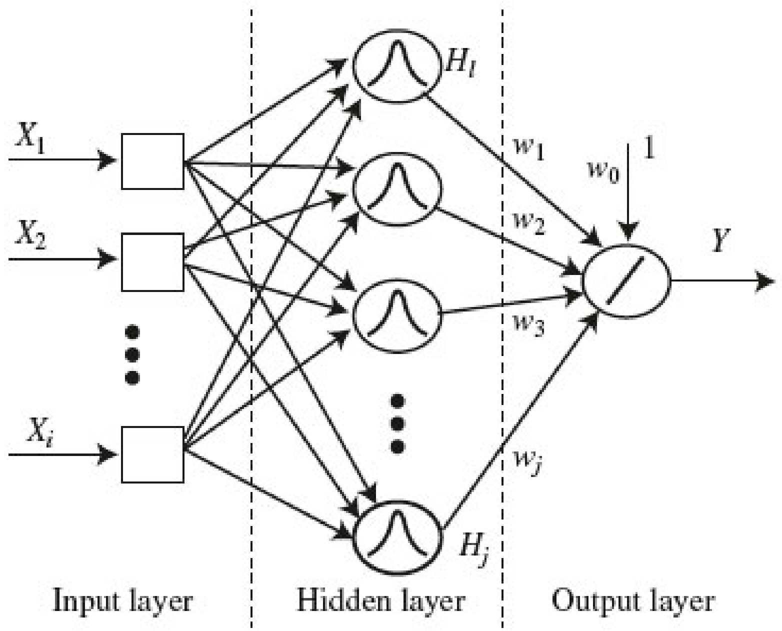
$\|\cdot\|$ – vector norm, often Euclidean

$\varphi_i(r)$ – kernel function, often Gaussian

Linear operation in N-dimensional space:

$$F(\mathbf{x})=\sum_{i=1}^{N} w_i\varphi_i\left(\left\|\mathbf{x}-\mathbf{x}_i\right\|\right)$$

The RBF neural network concept



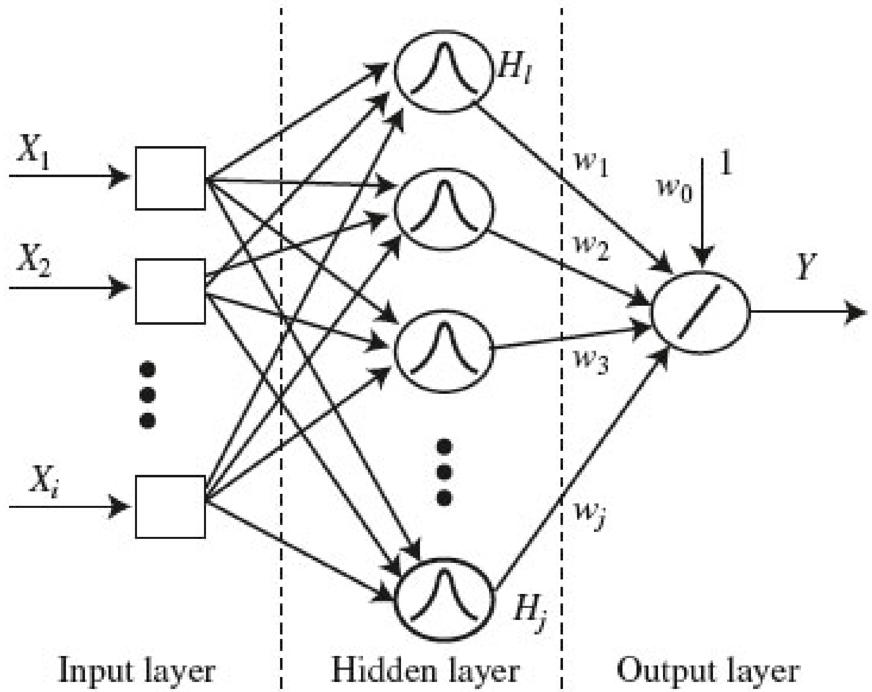
In the *exact interpolation*, the size of the hidden layer, N, is equal to the number of samples n

BUT this is not robust especially if a lot of samples are corrupted with noise!

We need modifications:

1) $N < n$
2) centres $\mathbf{x}_i$ different from samples
3) widths, σ, also differ across RBF nodes
4) it is possible to include biases

Hybrid learning algorithm



M **input** units

N **hidden** units

n **training** samples (each sample is N-dimensional)

Size of the **input layer** determined by the dimensionality of the input.

Hidden layer

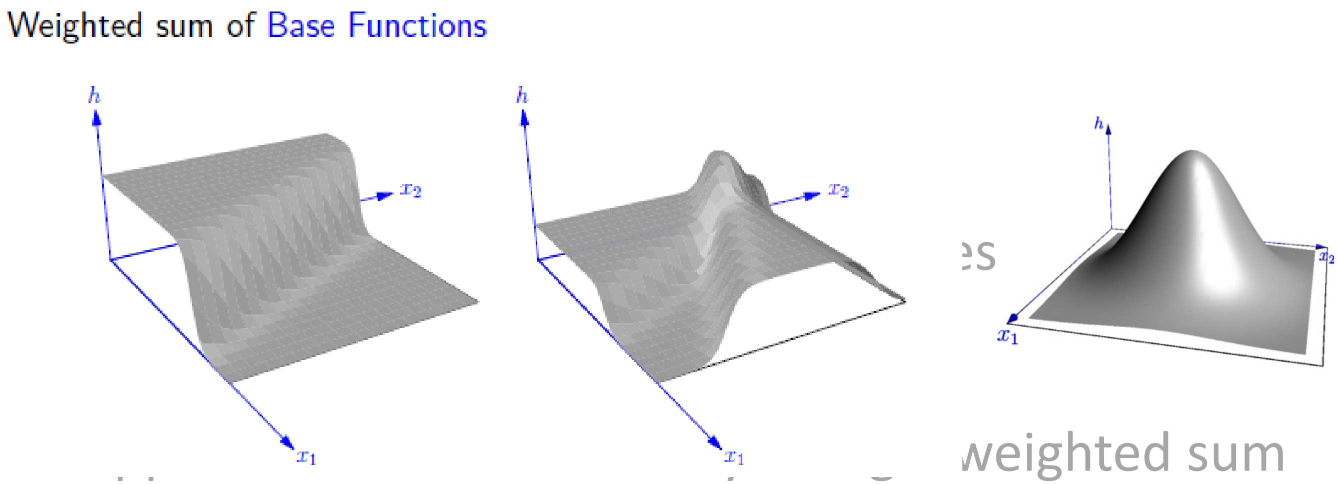
- N has to be decided
- centres and widths of hidden units have to be identified

Output layer performs a linear mapping – e.g., training with least square methods

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ . & . & . & . \\ \varphi_{n1} & \varphi_{n2} & \cdots & \varphi_{nN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ . \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ . \\ d_n \end{bmatrix}$$

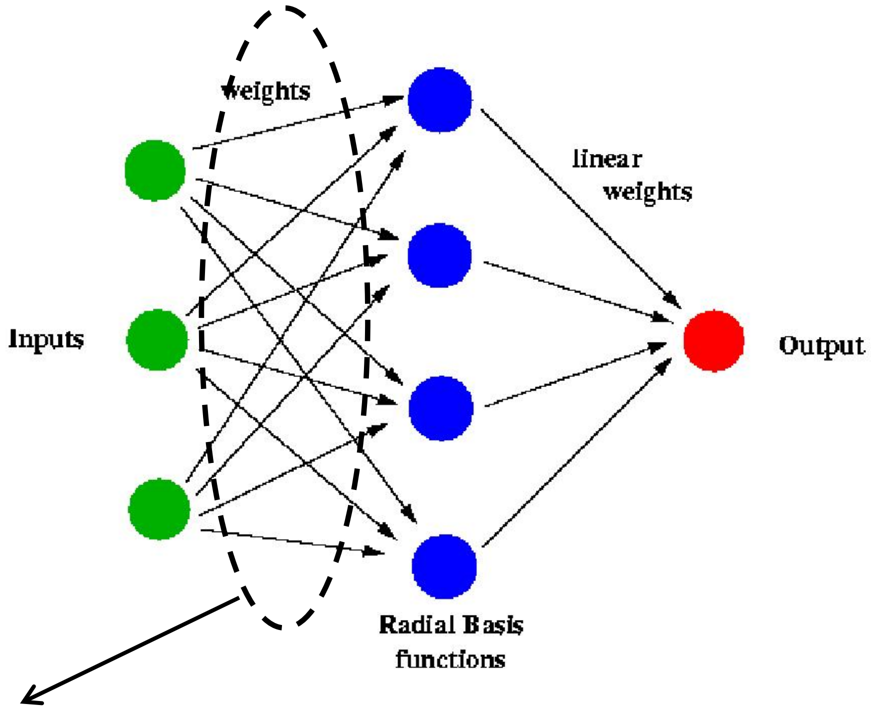
RBF NN vs MLP

- Hidden units in MLP rely on weighted linear summations of inputs (a matter of i
  - RBFs rely
- In MLPs, fun ...es weighted su
  - In RBFs, ... weighted sum



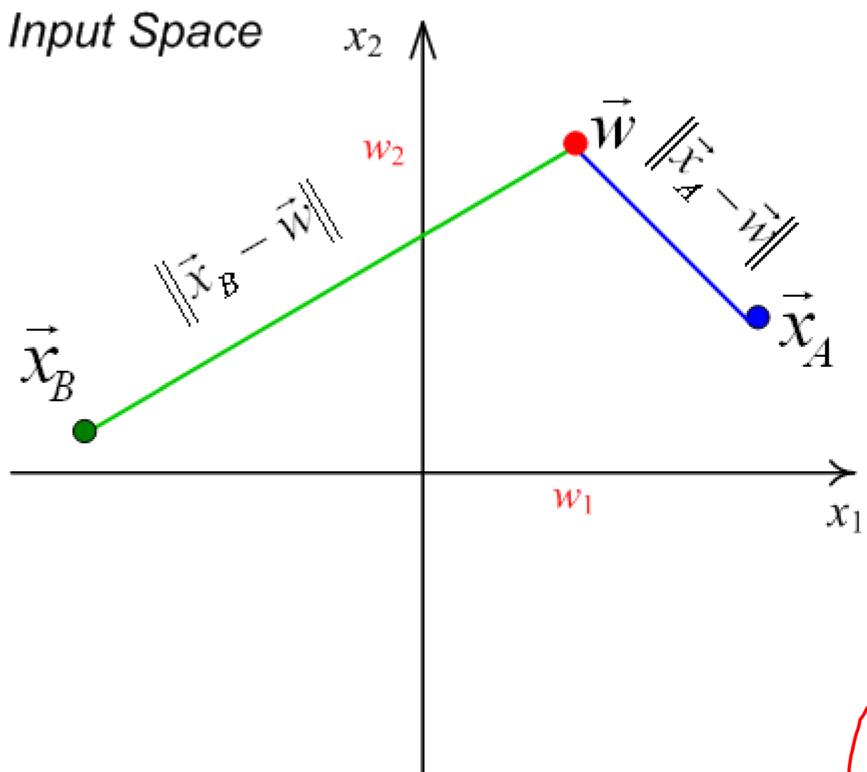
- MLPs form distributed activations (many hidden units contribute to the output for a given input, which partly leads to local minima etc.)
  - In RBFs, very few local basis functions (wrt. input) are activated for a given input

Interpretation of the weights to hidden layer

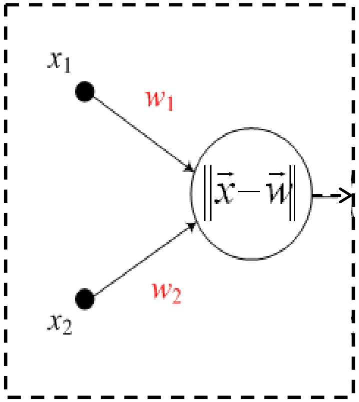


Do these connections have weights?

Weights in the input space – Euclidean distance



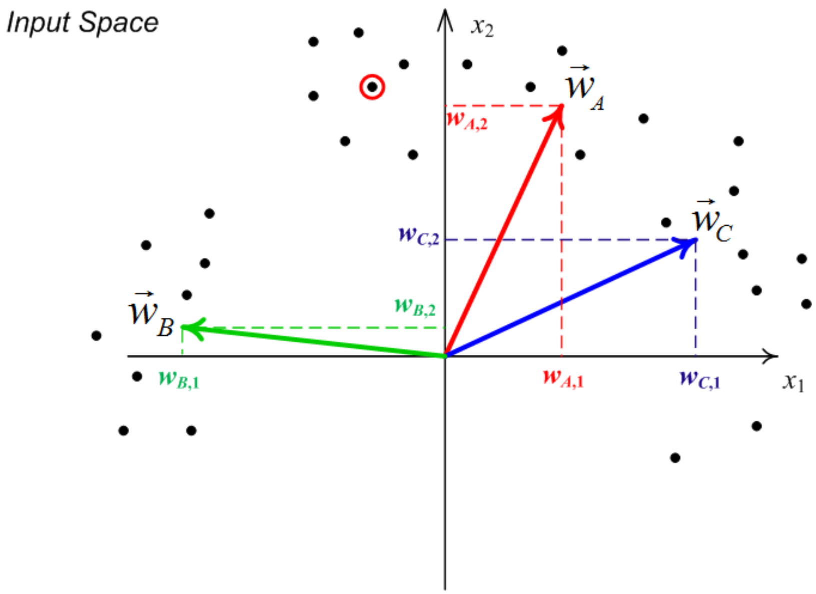
$$\|\vec{x} - \vec{w}\| = \sqrt{\sum_{i=1}^{2} (x_i - w_i)^2}$$



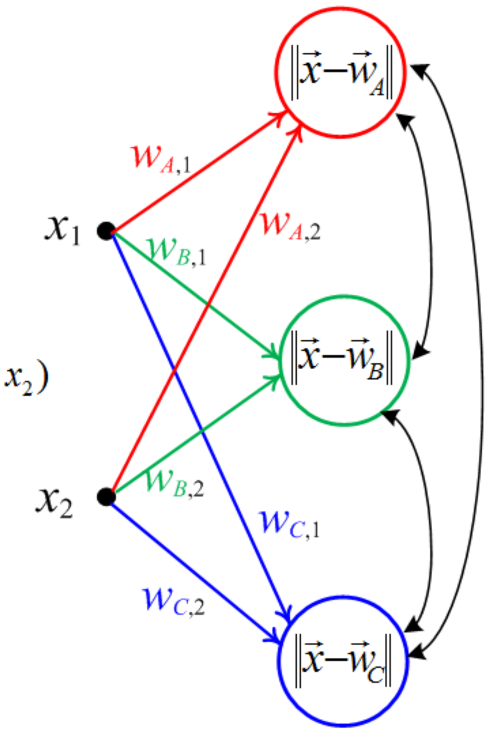
Euclidean distance measure: $\|\vec{x} - \vec{w}\|$

$$\|\vec{x}_A - \vec{w}\| < \|\vec{x}_B - \vec{w}\|$$

Competition – update of the input weights



$\vec{x} = (x_1, x_2)$



$$\vec{w}_A = (w_{A,1}, w_{A,2})$$
$$\vec{w}_B = (w_{B,1}, w_{B,2})$$
$$\vec{w}_C = (w_{C,1}, w_{C,2})$$

If the **red** node w_A wins, then:

$$\Delta\vec{w}_A = \eta\vec{x}$$

OR

$$\Delta\vec{w}_A = \eta(\vec{x} - \vec{w}_A)$$

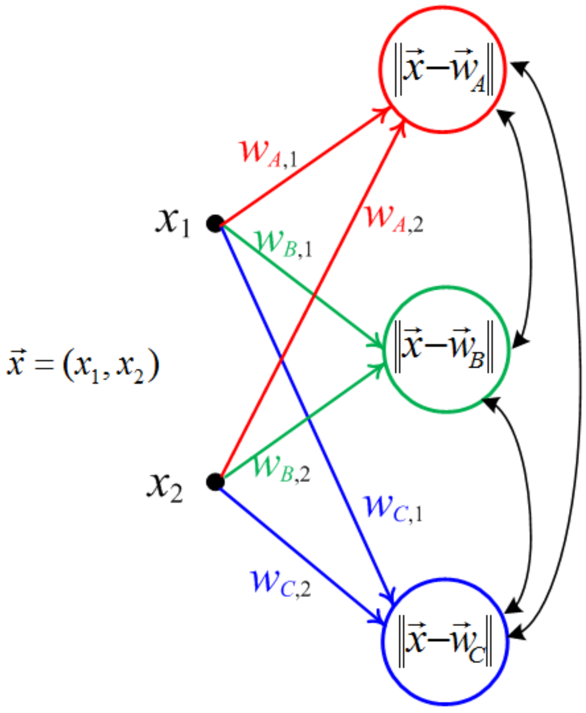
Competitive learning – basic principle

The fundamental principle

Update the *winning* unit (prototype/code vector) to make it more specialised ("even better").

Properties

- the algorithm finds cluster in data – purely unsupervised approach
- each node protects its "territory"
- there is also a batch version



If the **red** node w_A wins, then:

$$\Delta\vec{w}_A = \eta\vec{x}$$

OR

$$\Delta\vec{w}_A = \eta(\vec{x}-\vec{w}_A)$$

Competitive learning – problem with dead units

Dead unit problem

Prototype vectors far from actual data will never become better.

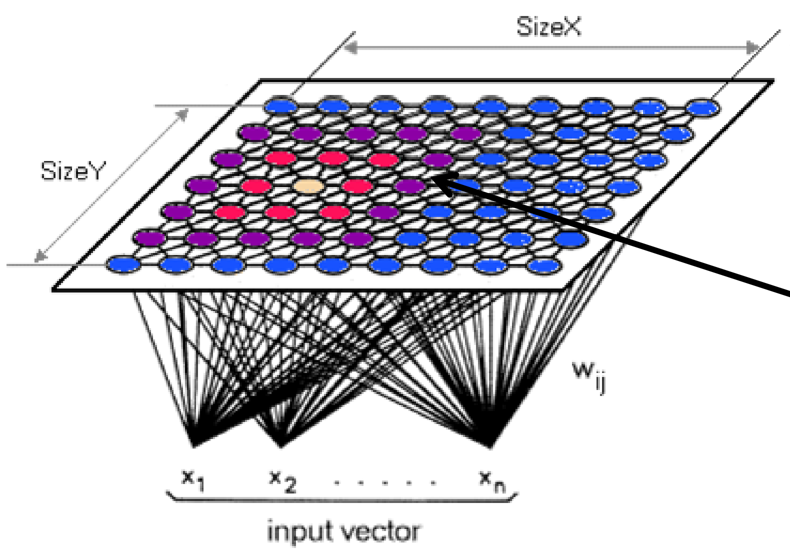
Methods to avoid dead units

- Initialise algorithm (the weights) with data samples
- "Leaky learning" (soft-competition) – some updates for all
- Learning with conscience – balanced update allowing losers to win
- Introduce noise to data

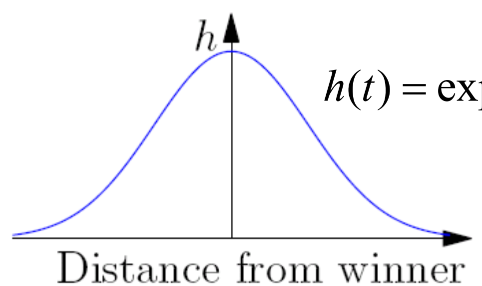
Distance and neighbourhood in the *output* space

In the OUTPUT space the distance between the nodes (their neigbourhood) is defined

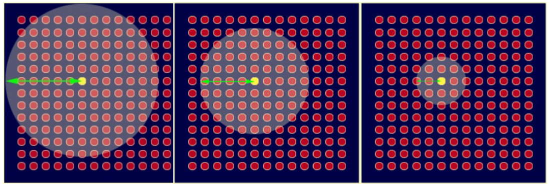
The links between the nodes are commonly used to define some discrete distance measure, *d*, in the *discrete* OUTPUT space



THEN we can use a neighbourhood function, *h*, based on the distance between the discrete nodes in the OUTPUT space



$$h(t) = \exp\left(-\frac{d^2}{2\sigma^2(t)}\right)$$

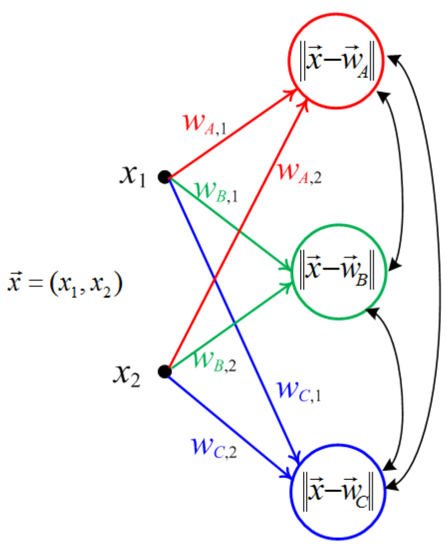


h shrinks since *σ* exponentially decreases over time: $\sigma(t) = \sigma_0 \exp\left(-\frac{t^2}{\tau}\right)$

Topology preserving maps

Learning principle

Competitive learning where winning "spills over" to neighbours.



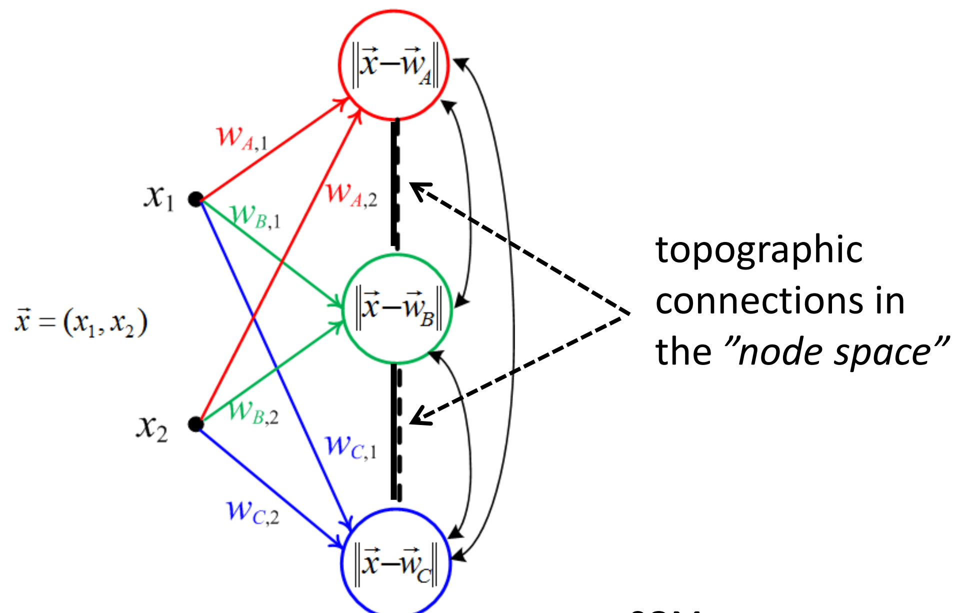
If the **red** node w_A wins, then:

$$\Delta\vec{w}_A = \eta\vec{x}$$

OR

$$\Delta\vec{w}_A = \eta(\vec{x} - \vec{w}_A)$$

Competitive learning



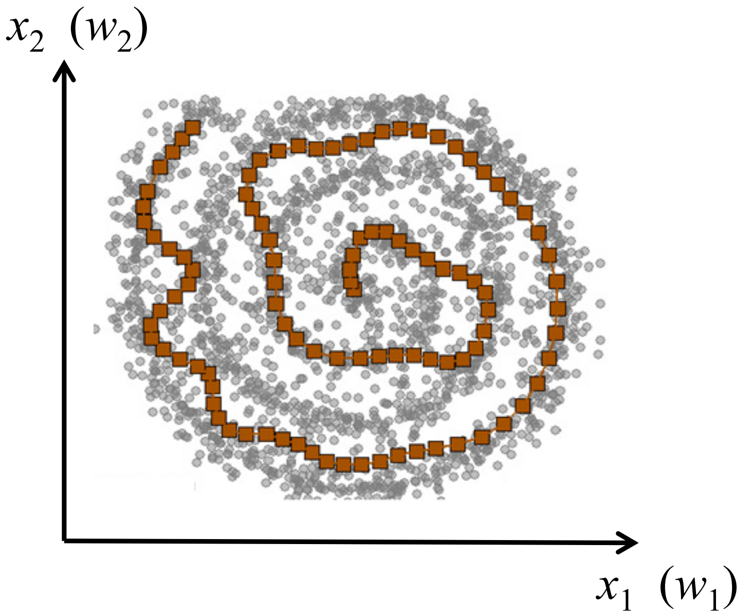
SOM

SOM visualisation – showing lattices in *input* space

... it is very useful to **show links between nodes** as they illustrate neighbourhood (topographical relationship) in the OUTPUT space

2 inputs – **2D** INPUT space (x_1,x_2) corresponding to **2D** WEIGHT space (w_1,w_2)

1D arrangement of nodes
in the OUTPUT space



Learning Vector Quanitsation (LVQ)

Learning Vector Quantisation (LVQ) is a *supervised* competitive learning algorithm (classes are known)

$$\Delta\vec{w} = +\eta(\vec{x} - \vec{w})$$

if the winner belongs to the *right* class

$$\Delta\vec{w} = -\eta(\vec{x} - \vec{w})$$

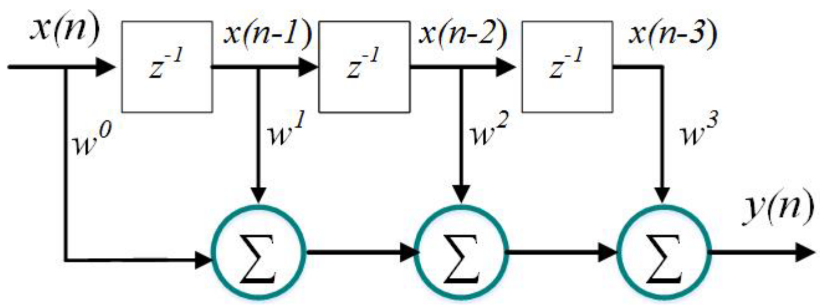
if the winner belongs to the *wrong* class

Static MLP for handling dynamics

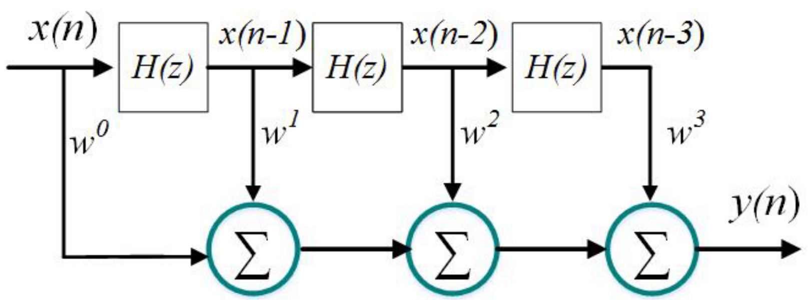
The use of a static MLP to account for temporal dimension

- short-term memory function
- nonlinear regression capabilities

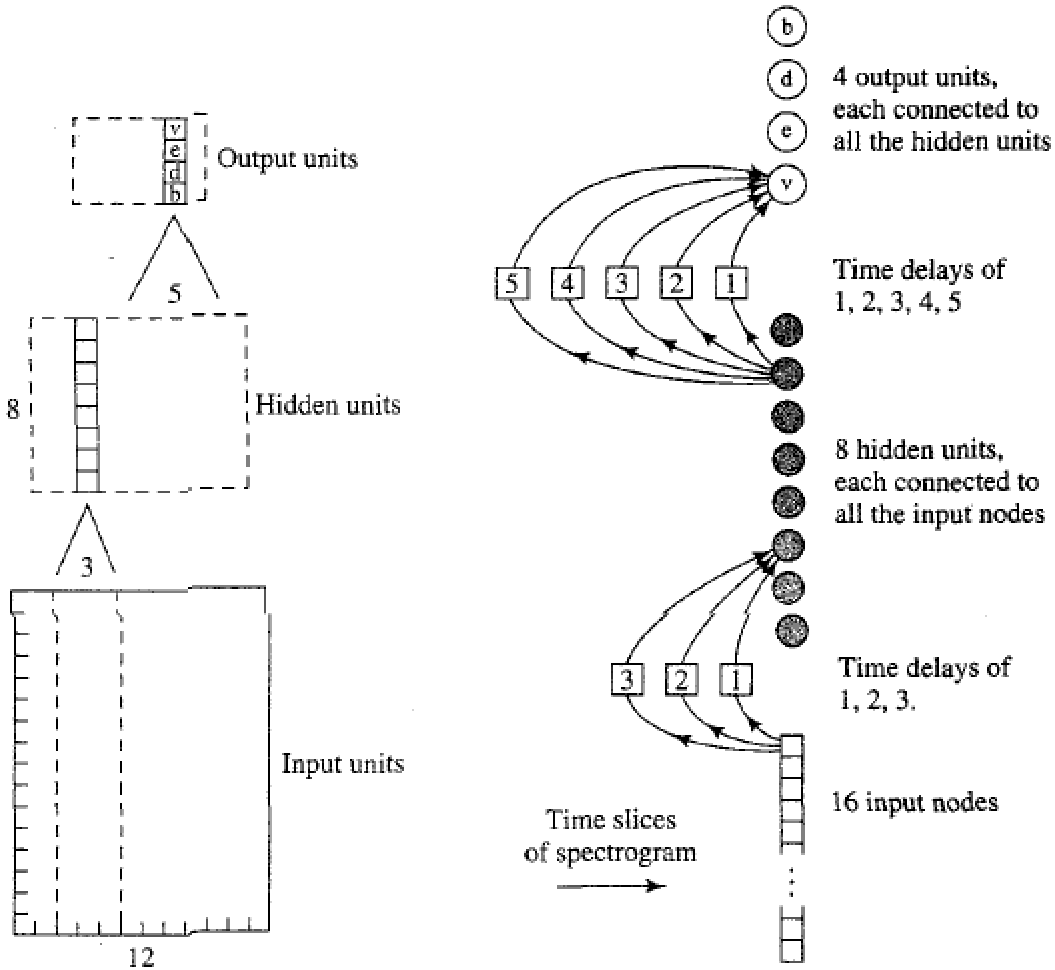
Tapped delay line memory



Generalized tapped delay line memory

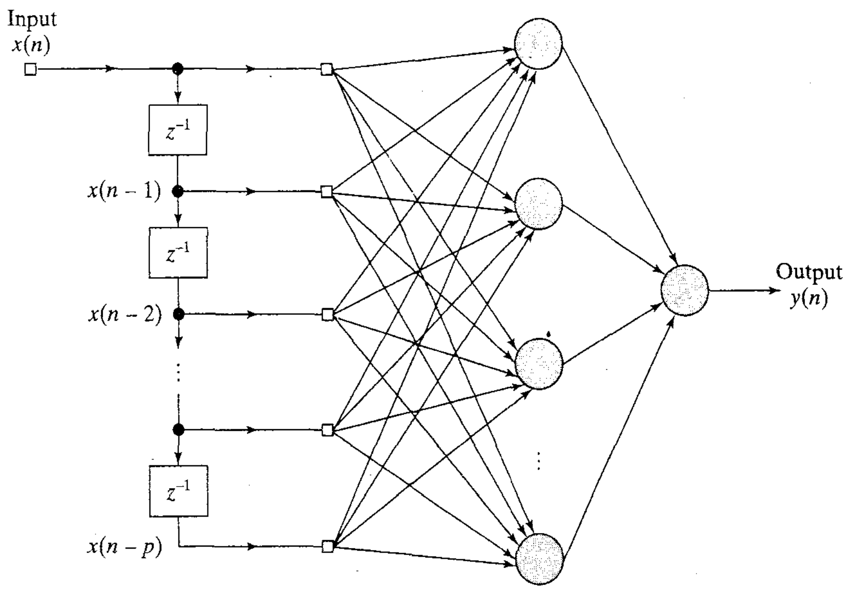


Time delay neural network



Lang and Hinton, 1988

Learning approach to TLFN



Backprop can be used with relatively simple *focused TLFNs* .

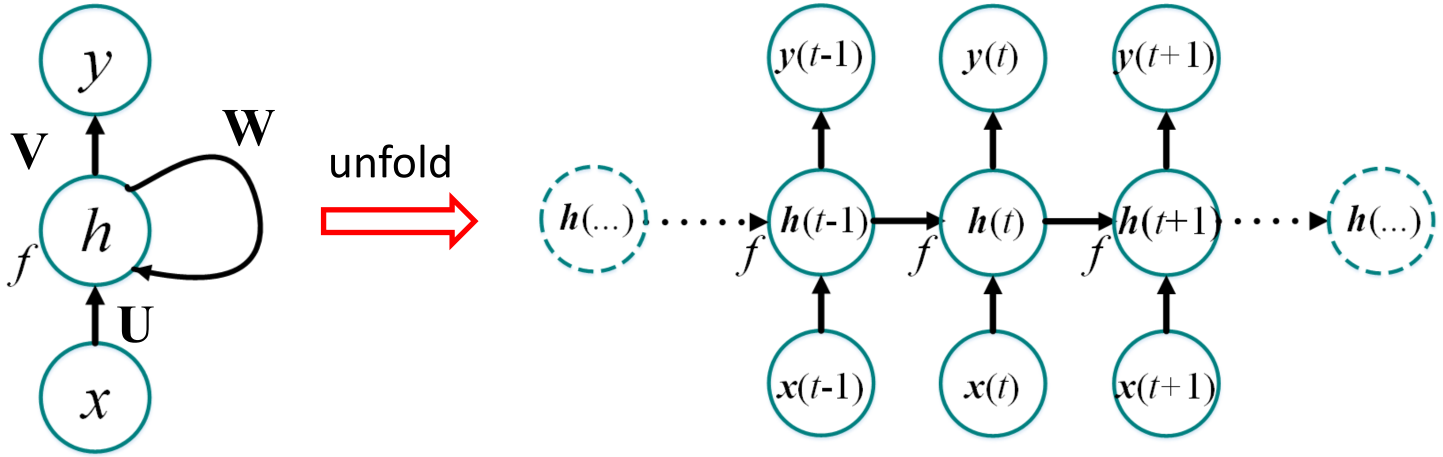
A general principle to unfold the network: form a large "static" network, and apply backprop.

For *distributed TLFNs*, using standard backprop is neither practical nor elegant.

Temporal backpropagation

Haykin, 1999

Recurrent connections between hidden units

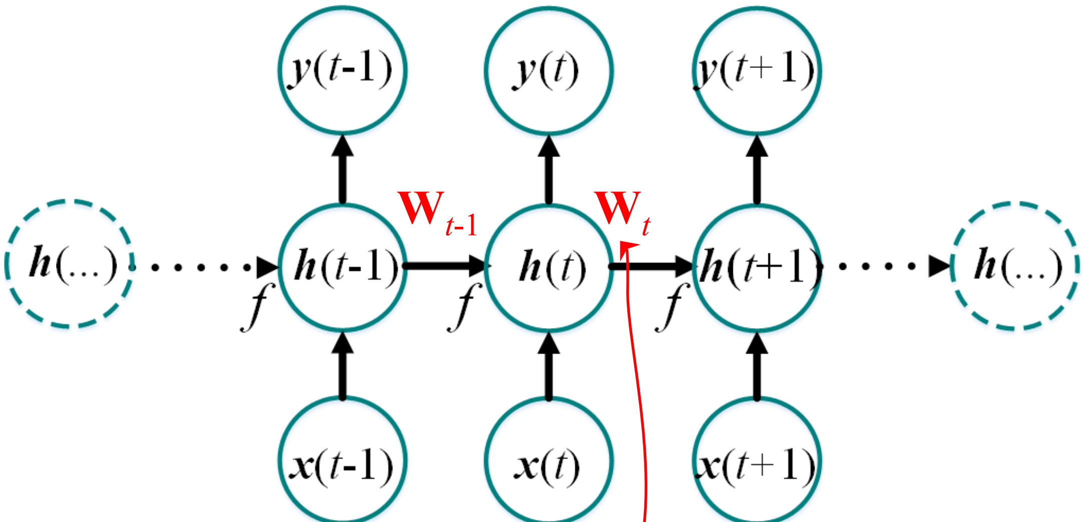


$$\boldsymbol{h}(t) = f(\mathbf{W}\boldsymbol{h}(t-1) + \mathbf{U}\boldsymbol{x}(t) + bias)$$

$$\boldsymbol{y}(t) = \mathbf{V}\boldsymbol{h}(t) + bias$$

state-space description

Backpropagation through time



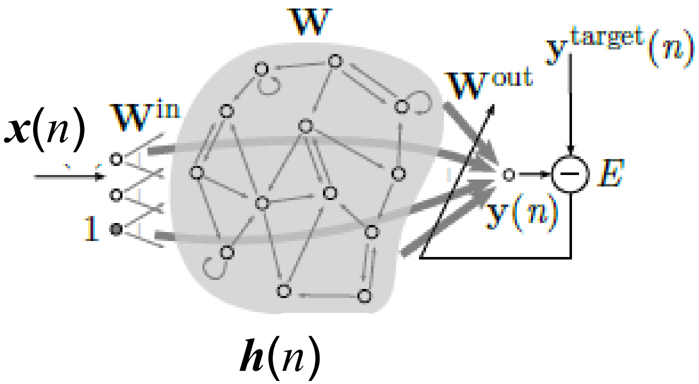
shared duplicated weights

Essentially,

$\mathbf{W}=\mathbf{W}_t$, $\mathbf{W} = \mathbf{W}_{t\text{-}1}$ etc.

So: $$\underbrace{\frac{\partial E}{\mathrm{W}} = \frac{\partial E}{\mathrm{W}_t} + \frac{\partial E}{\mathrm{W}_{t-1}} + \ldots}_{\text{\#time steps in a training sample}}$$

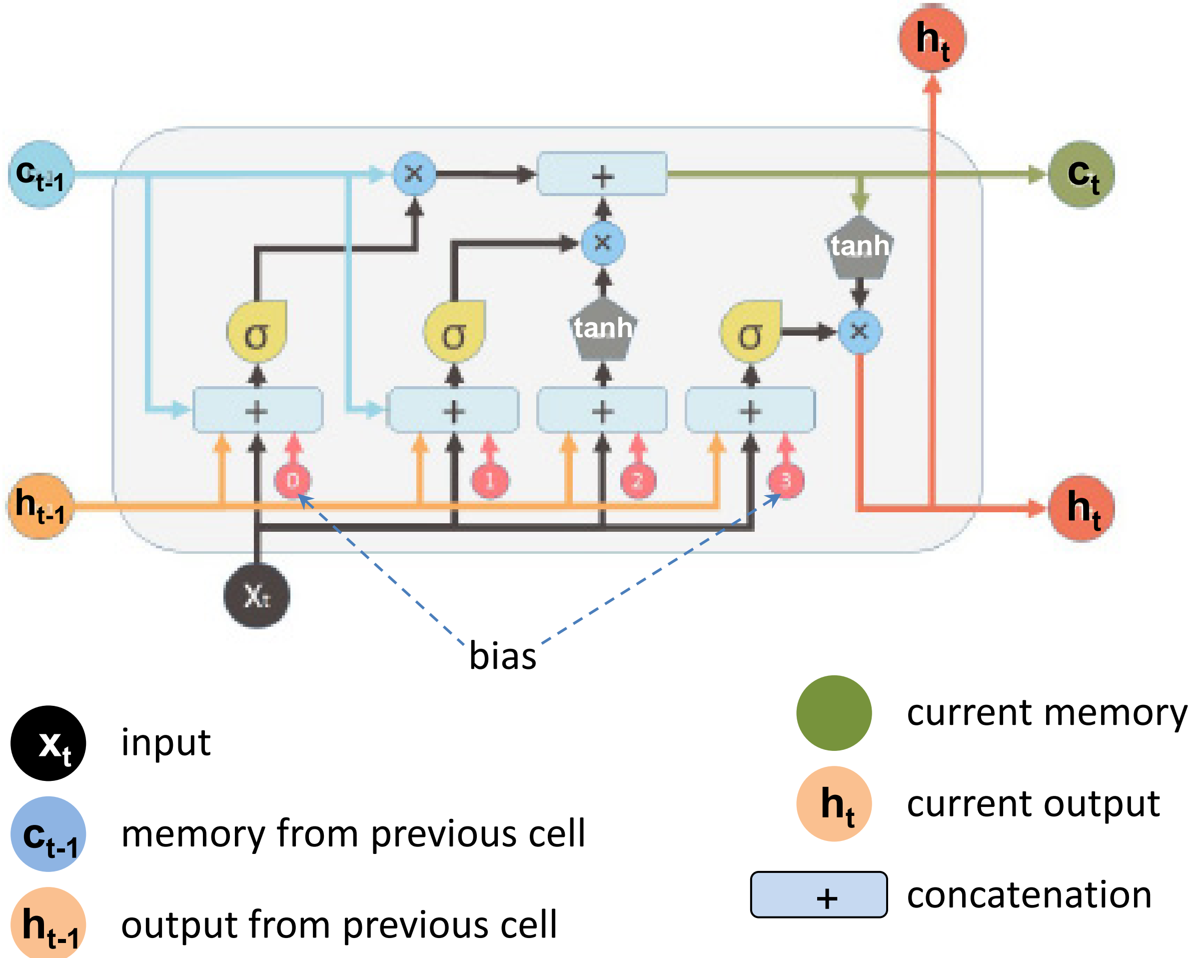
Reservoir computing – overall recipe

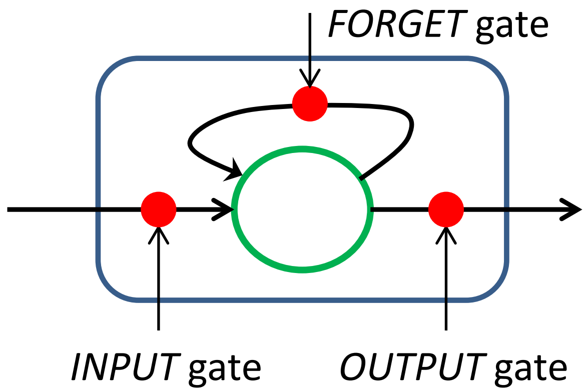


Recipe

1. Generation of the dynamic reservoir ($\boldsymbol{W}_{in}$, $\mathbf{W}$)
2. Application of inputs, $\boldsymbol{x}(n)$, and collecting the corresponding activation states, $\boldsymbol{h}(n)$.
3. Computation of the linear output weights from the reservoir with a linear regression approach (MSE error to be minimised).
4. Use of the RNN for new data – predictions.

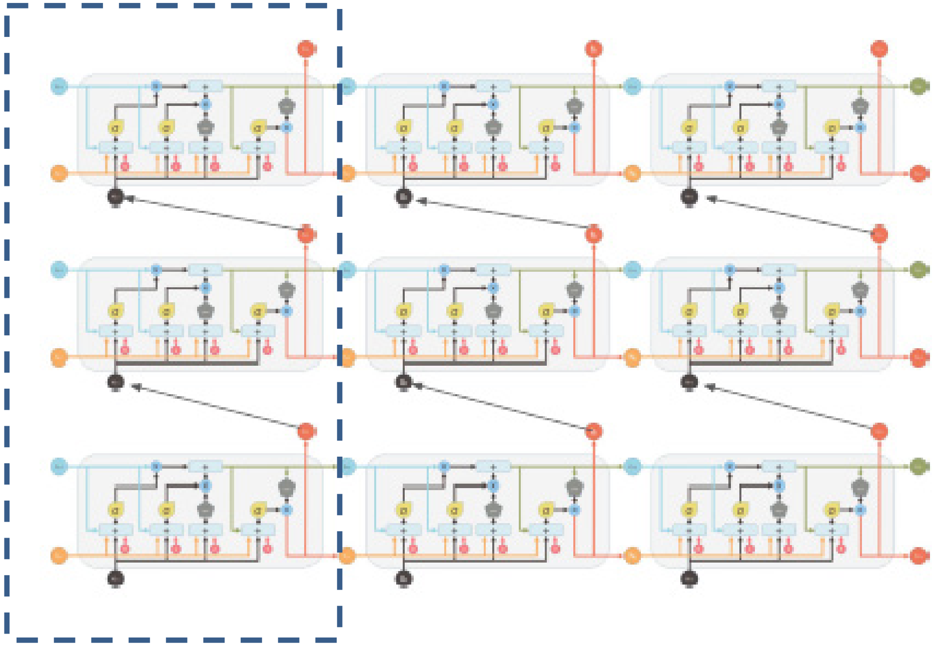
Long short-term memory (LSTM) cell





adapted from A. Sood and colah's blog

Deep LSTM



deep stacking

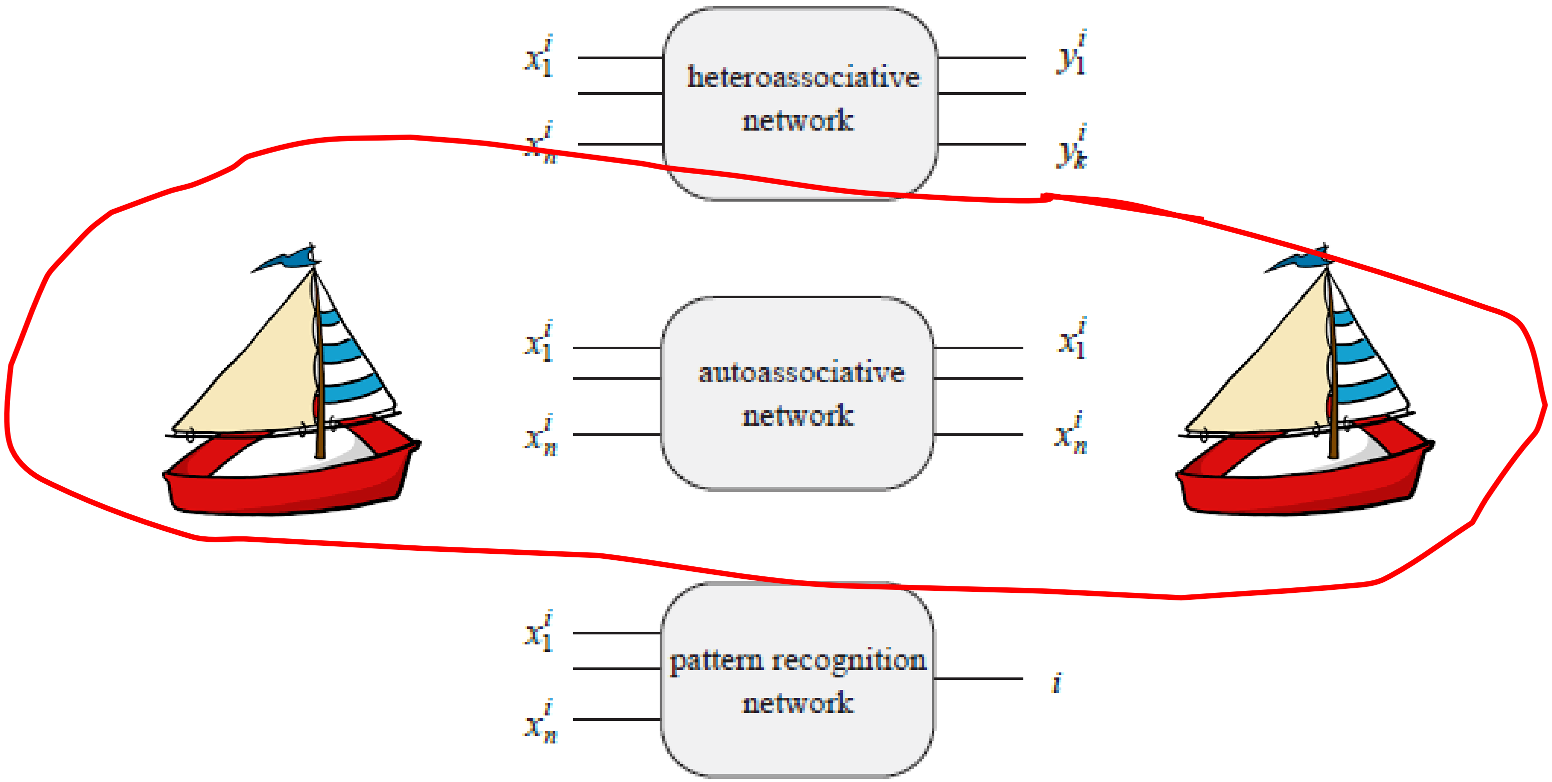
output sequence of one layer constitutes the input sequence to another layer

Why do we go deep?

- has the potential to perform better at handling temporal information at wide varying scales
- requires however many more parameters to be learnt

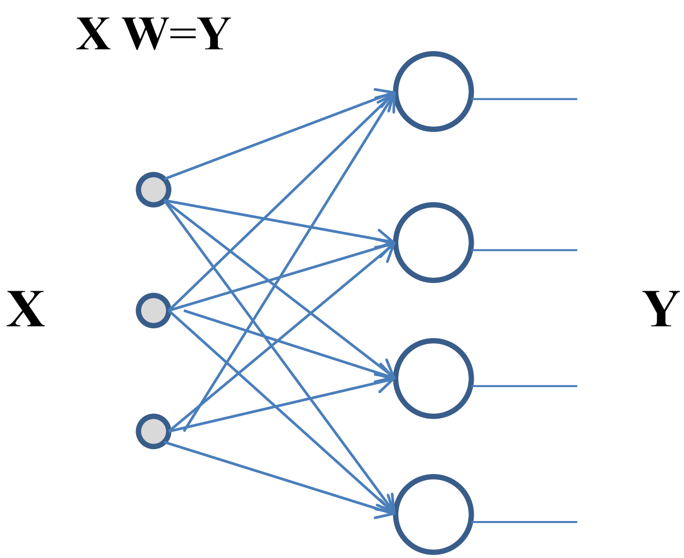
adapted from A. Sood

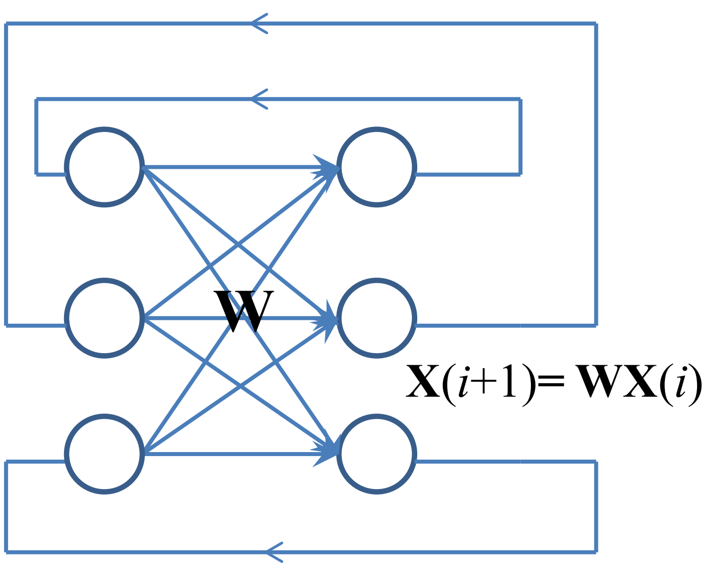
Associative pattern recognition



Linear associative memory networks

- Simple single layer or recurrent networks





without feedback
(recall is a feedforward step)

autoassociative recurrent network, with feedback
(recall is an iterative process)

Learning for associative memory networks

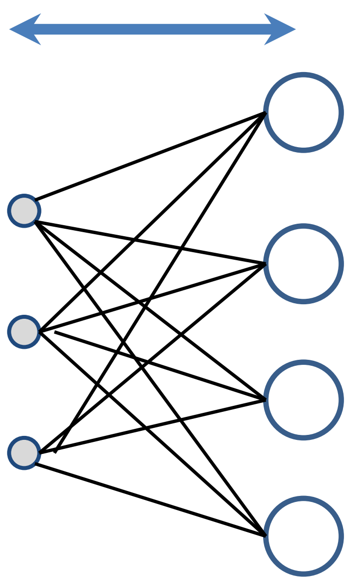
Hebbian (outer product) vs pseudoinverse matrix approach

$$\mathbf{W} = \mathbf{X}^{\mathrm{T}}\mathbf{X} \quad vs \quad \mathbf{W} = \mathbf{X}^{+}\mathbf{Y}$$

Fast computations and direct biological interpretation but non-orthogonal projection causing memory recall problems

Better reliability and storage capacity (less problems with crosstalk) with orthogonal projections mitigating crosstalk problems when recalling memories

Concept of energy in BAM



If $(\vec{x},\vec{y})$ is a stable point, then nearby points like $(\vec{x}_0,\vec{y}_0)$ should converge.

$$\vec{y}_0 = \mathbf{W}\vec{x}_0 \text{ , next } \quad \vec{e} = \mathbf{W}^{\mathrm{T}}\vec{y}_0$$

How far is $\vec{e}$ from $\vec{x}_0$?

$$E = -\vec{x}_0^{\ \mathrm{T}}\vec{e} = -\vec{x}_0^{\ \mathrm{T}}\mathbf{W}^{\mathrm{T}}\vec{y}_0 = -\vec{y}_0^{\ \mathrm{T}}\mathbf{W}\vec{x}_0$$

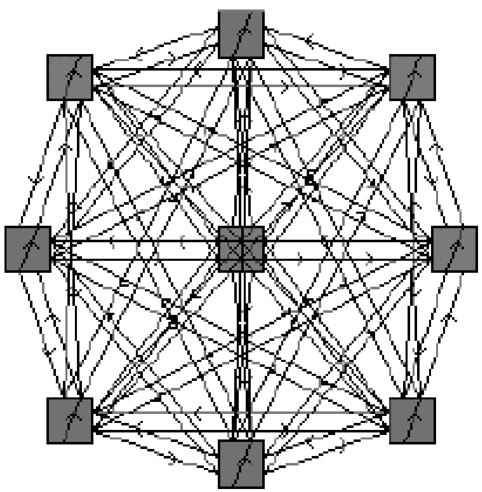
For the autoassociative BAM with $\mathbf{W}$, energy in the state $\vec{x}$:

$$E(\vec{x},\vec{x}) = -\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x} + \vec{x}^{\mathrm{T}}\vec{\theta}$$

If bias is added

$$E(\vec{x}) = -\frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}x_i x_j + \sum_{i=1}^{n}\theta_i x_i$$

Hopfield network

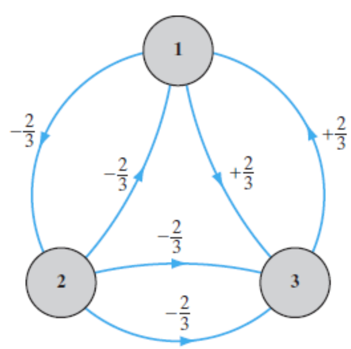


$$\forall_i \, w_{i,i} = 0$$ no self-connections

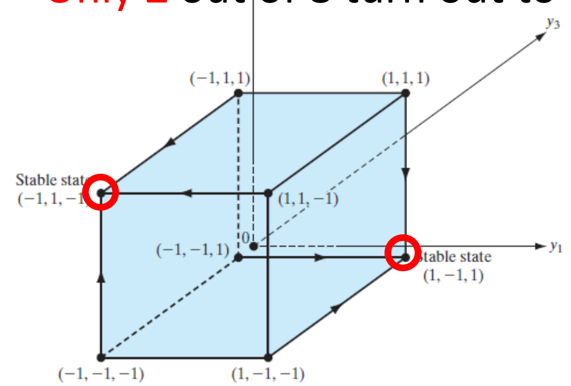
$$\vec{x}' = \mathrm{sgn}(\mathbf{W}\vec{x} + \vec{\theta})$$

$$E(state = \vec{x}) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} x_i x_j + \sum_{i=1}^{n} \theta_i x_i$$

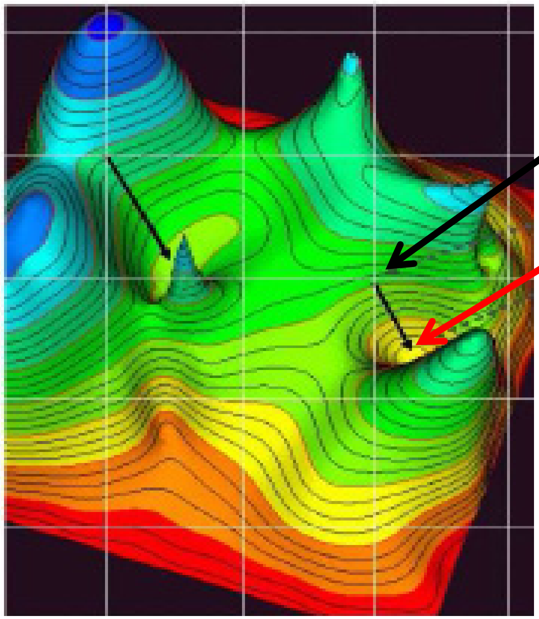
$\mathbf{W}$ should be symmetric with diag=0 for convergence



Only **2** out of 8 turn out to be stable!

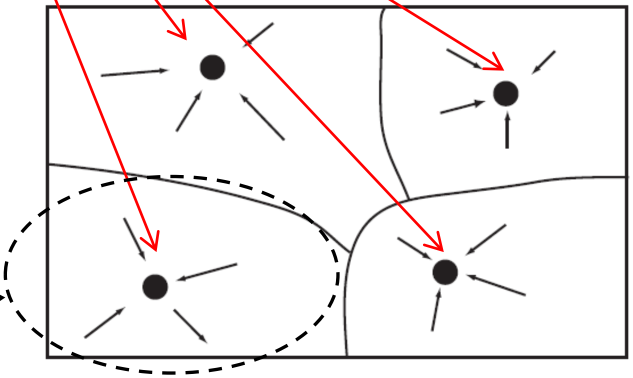


Attractor dynamics



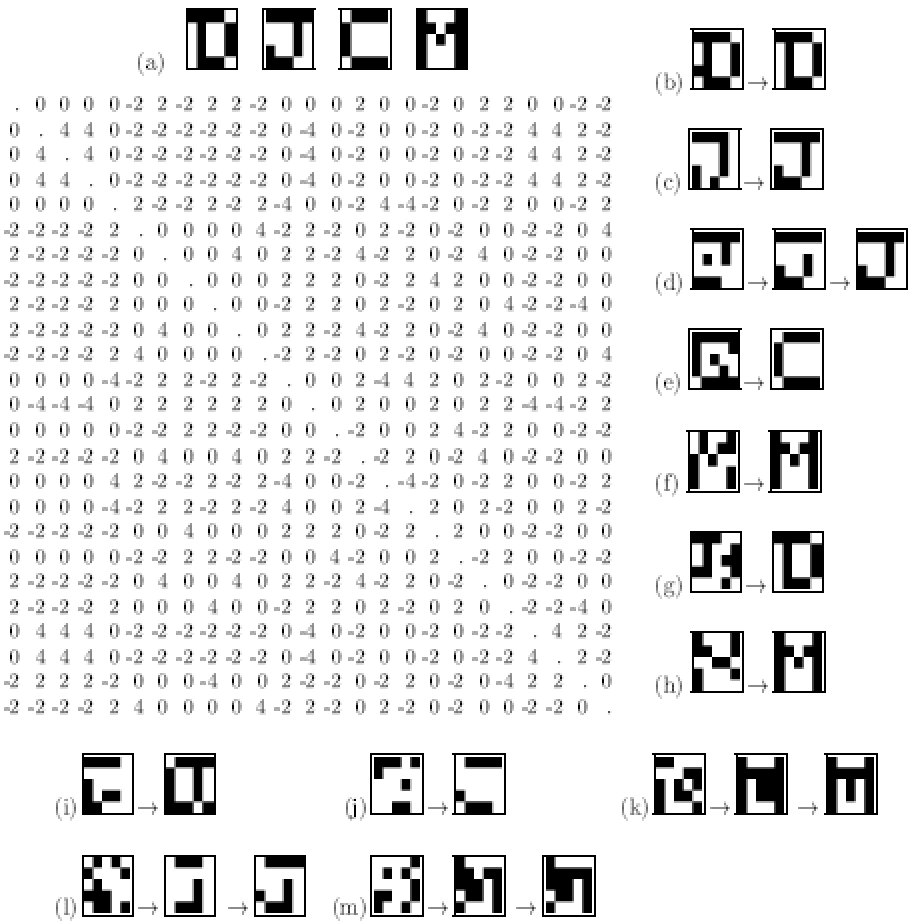
Memory cue
(within the basin of attractor)

Memory state
(local energy minimum,
stable point, **fixed-point attractor**)



Around each fixed point (attractor), there is
a ***basin of attraction***

Pattern storage and recall example

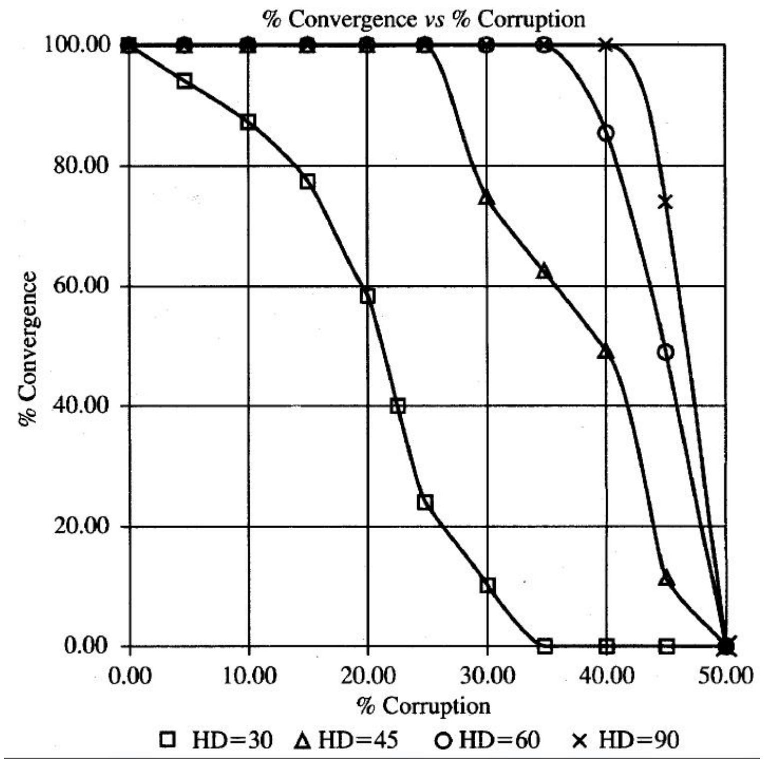


Common problems

1. Corruption of individual bits.
2. Lack of encoded memory or a very small basin of attraction.
3. Appearance of spurious additional memories.

adapted from McKay

Catastrophic forgetting effect



Convergence rate is defined based on the convergence criterion, often expressed as the upper bound on *Hamming distance*.

Network properties are not robust for synchronous updates.

Also, problems for continuous networks.

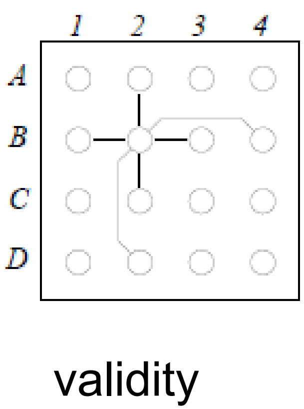
$$a_i = \sum_j w_{ij} x_j \qquad x_i = \tanh(a_i).$$

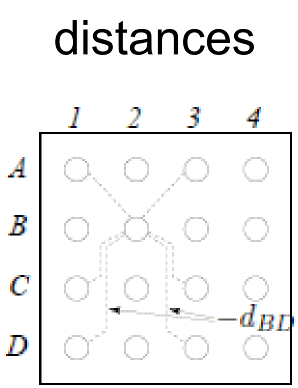
Better behaviour for continuous continuous –time Hopfield network

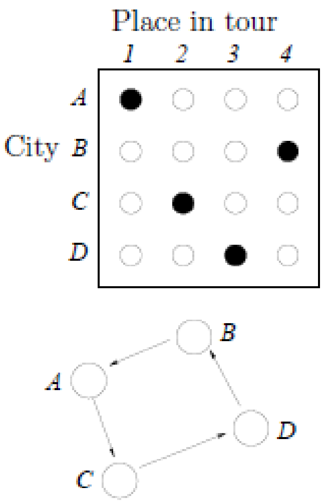
$$a_i(t) = \sum_j w_{ij} x_j(t). \qquad \frac{\mathrm{d}}{\mathrm{d}t} x_i(t) = -\frac{1}{\tau}(x_i(t) - f(a_i)),$$

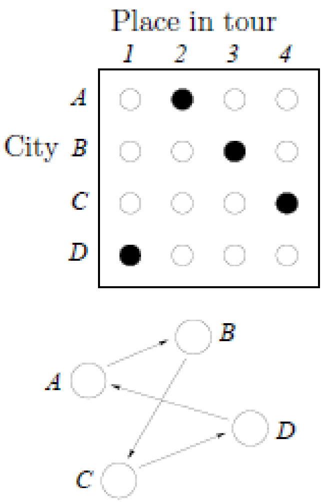
Hopfield networks for optimisation problems

- Hopfield network's dynamics minimises an energy function
- Some optimisation problems could be mapped to the quadratic energy function (particularly constrain satisfaction problems(CSPs))
- Travelling salesman problem (TSP) as a classic CSP problem









From Hopfield networks to Boltzmann machines

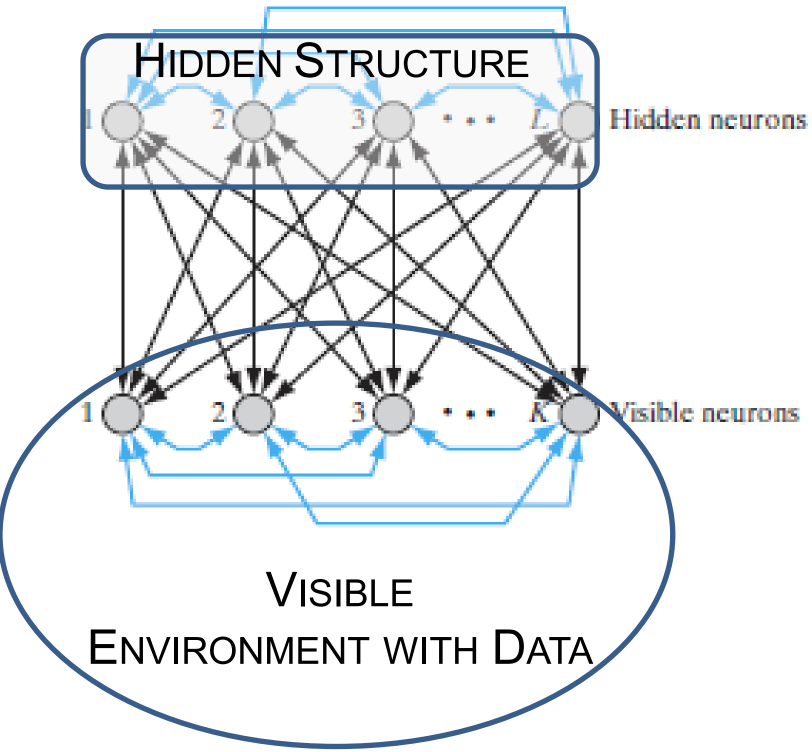
- Energy of this stochastic network is the same as before

$$E = -\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}x_i x_j$$

- Given a set of examples $\{\vec{x}_i\}_1^m$ the idea is to adjust $\mathbf{W}$ to describe data distribution (well matched to these examples)

$$P(\vec{x} \mid \mathbf{W}) = \frac{e^{-E}}{Z} = \frac{1}{Z(\mathbf{W})}\exp\left(\frac{1}{2}\vec{x}^{\mathrm{T}}\mathbf{W}\vec{x}\right)$$

Hidden and visible units



- Symmetric connections between visible, $\boldsymbol{v}$, and hidden neurons, $\boldsymbol{h}$
- Hidden neurons help account for higher-order correlations in the input vectors (data)
- Visible units provide interface to the external world – environment (data, $\boldsymbol{v}=\boldsymbol{x}$)
- Hidden units operate freely and are used to explain environmental input vectors

Boltzmann learning – two phases

$$\frac{\partial L(\mathbf{W})}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \log P\left(\{\boldsymbol{x}^{(p)}\}_1^M \mid \mathbf{W}\right) = \sum_p \left\{ \boxed{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{h}|\boldsymbol{v}=\boldsymbol{x}^{(p)},\mathbf{W})}} - \boxed{\left\langle y_i y_j \right\rangle_{P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W})}} \right\}$$

$$\Delta w_{i,j} \propto \left\langle y_i, y_j \right\rangle_{\text{data}} - \left\langle y_i, y_j \right\rangle_{\text{model}}$$

- **Positive** phase implies clamping the inputs (relative fast)

"Hebbian learning" $\left\langle y_i, y_j \right\rangle_{data}$

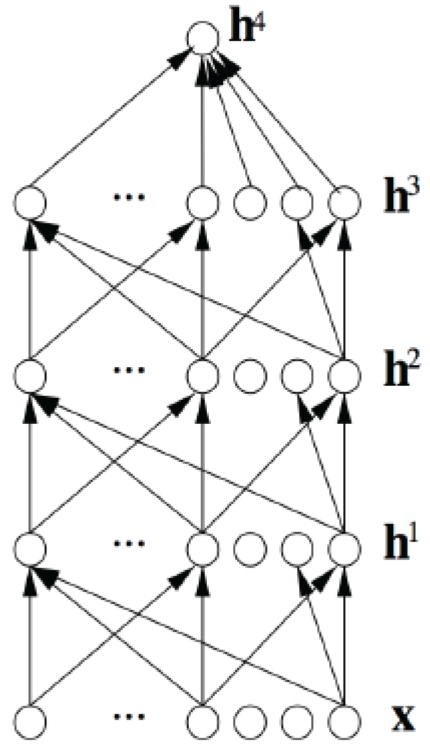
- **Negative** phase involves updating all the units (can be very slow)

"Hebbian forgetting" $\left\langle y_i, y_j \right\rangle_{\text{model}}$

prevent from learning false, spontaneously generated states

What is depth in ML?

- Depth of architecture
  - the number of levels of composition of nonlinear operations in the function learnt
  - the length of the longest path from input to output in the graph

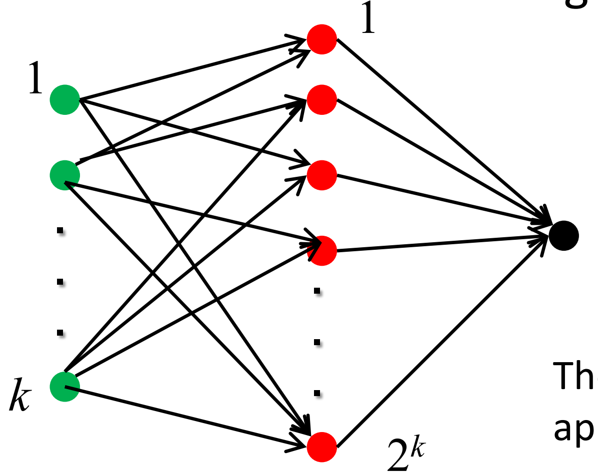


- Deep learning
  - using multiple layers of inf. processing stages in hierarchical architectures for pattern recognition and representation learning
  - focus on (incremental) learning of feature hierarchies

Motivation for deep structures

Why go deep? Do we need deep structures?

- Expressive power and compactness of models (*expressibility* and *efficiency*)
  - enhances generalisation, especially with limited training examples
  - less degrees of freedom when handling complexity and nonlinearity – exponential gain

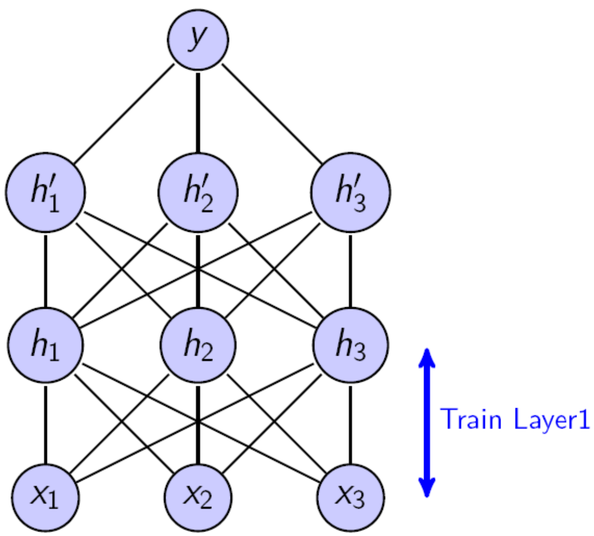


Shallow structure may need exponential size of hidden layer(s)

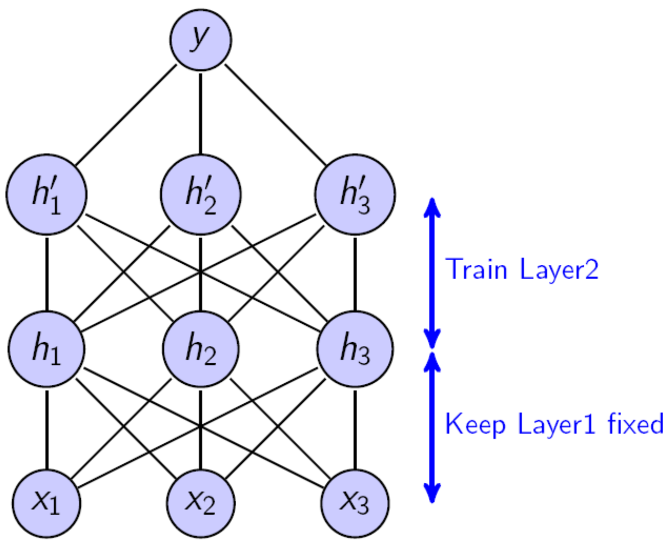
The universal approximation theorem and approximation costs.

General theme of the older deep learning protocol – deep belief networks, stacked autoencoders

- Greedy layer-wise unsupervised pre-training + supervised tuning (the legacy of Hinton, Bengio and LeCun)



Single layer at a time

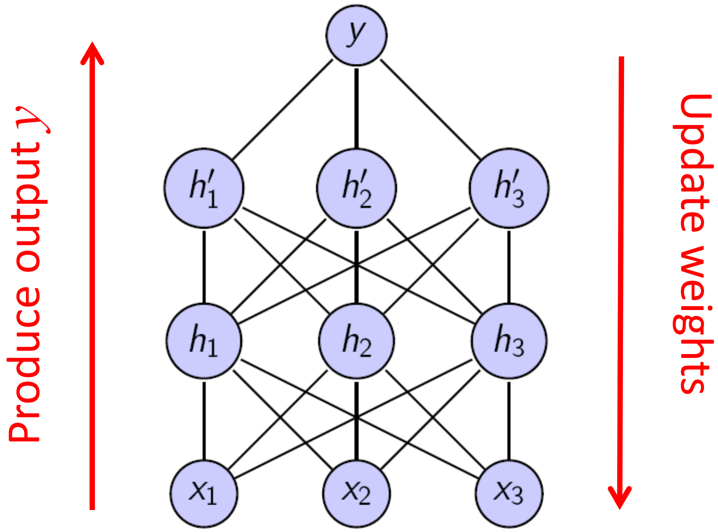


Train another layer while keeping the lower layer fixed

Hinton et al., 2006
Duh, 2013

General theme of the older deep learning protocol – deep belief networks, stacked autoencoders

- Greedy layer-wise unsupervised pre-training + supervised tuning (the legacy of Hinton, Bengio and LeCun)



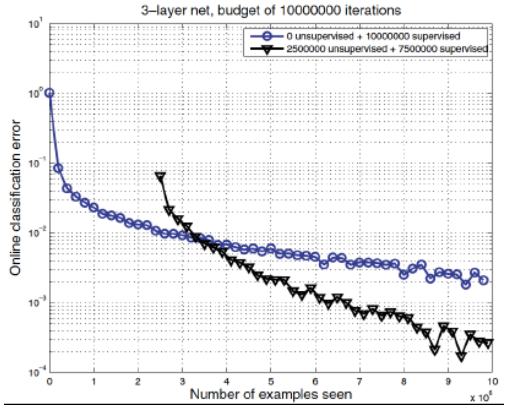
Gradient-based fine tuning

1. Add a classifier layer and retrain globally the entire structure.
2. Train only a supervised classifier on top and keep other layers fixed.

Hinton et al., 2006
Duh, 2013
LeCun & Ranzato, 2013

Hypothetical role of unsupervised pre-training

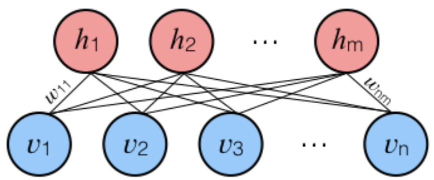
- Regularisation hypothesis (Erhan et al., 2010)
  - Pre-training minimises variance
  - It also helps to control complexity for architectures with large sizes of hidden layers
  - Acts like an implicit penalisation term
- Optimisation hypothesis (Bengio et al., 2007)
  - pre-training finds a better initial condition for further gradient-based optimisation
  - it facilitates training of the entire architecture (lower and higher layers benefit from tuning)



Most common network architecture and learning types

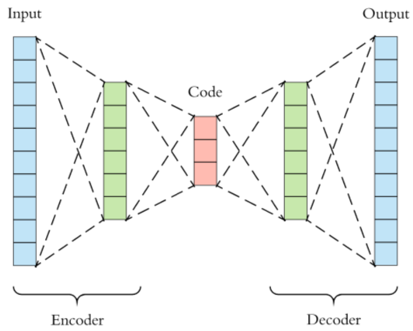
Restricted Boltzmann machine (RBM) layer

(contrastive divergence for pre-training)

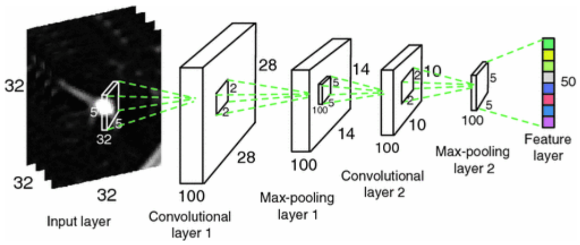


Auto-encoder (AE) layer

(gradient descent based algorithms for pre-training)



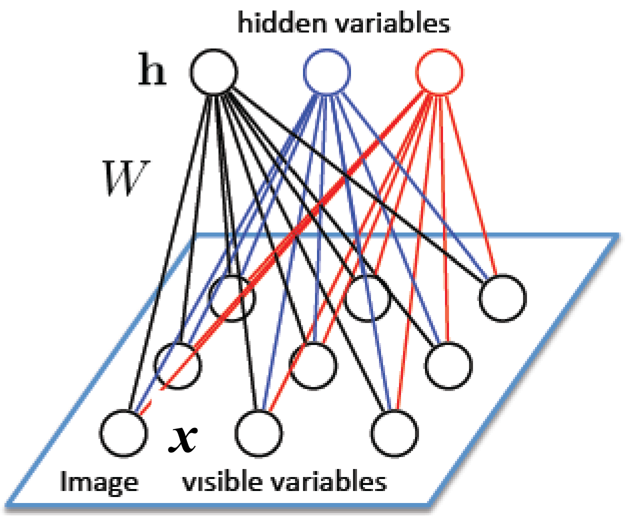
Convolutional neural networks (CNNs)



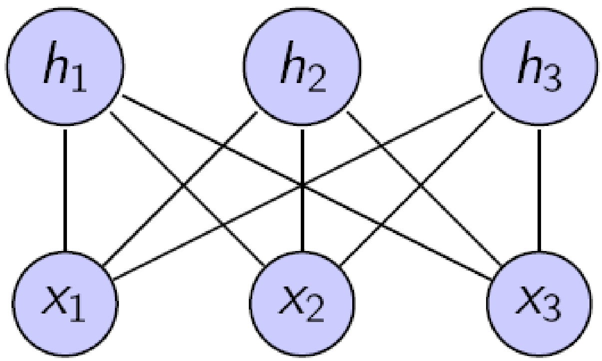
Greedy layer-wise unsupervised pre-training, which is increasingly omitted once **ReLU** units are employed

Network initialised without any pre-training

Restricted Boltzmann machine (RBM)



Simple energy-based model



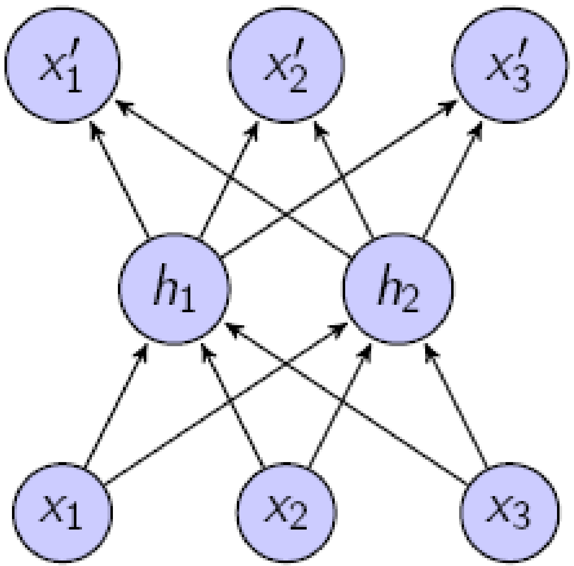
$$p(x,h) \sim e^{-E_\theta(x,h)}$$

$$E_\theta(x,h) = -x'Wh - b'x - d'h$$

In traditional RBM, x_i and h_j are binary variables

The idea is to optimise log-likelihood with the use of approximative Gibbs sampling – Constrastive Divergence algorithm

Autoencoders



Decoder: $x' = \sigma(W'h + d)$

Encoder: $h = \sigma(Wx + b)$

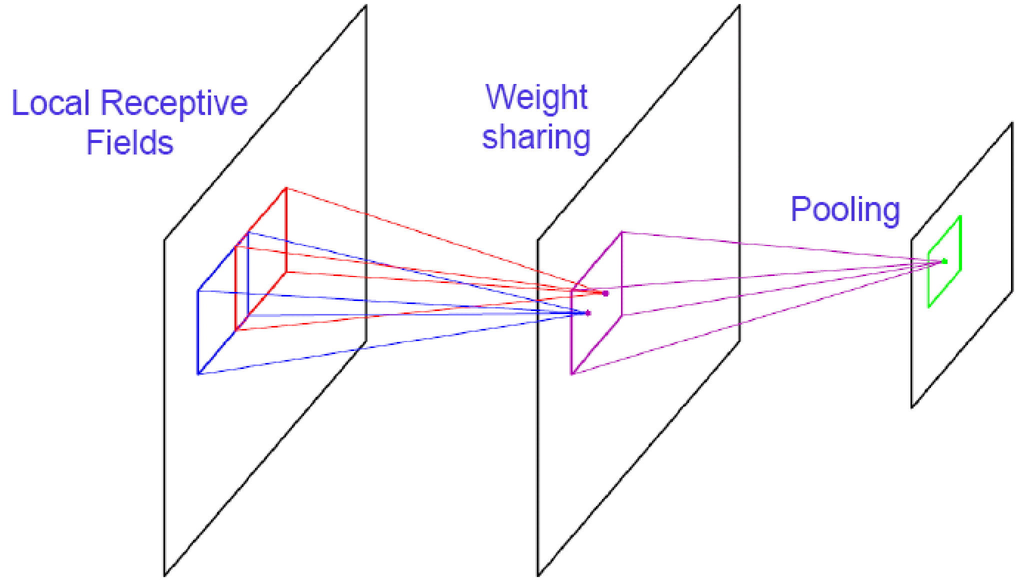
Encourage h to give small reconstruction error:

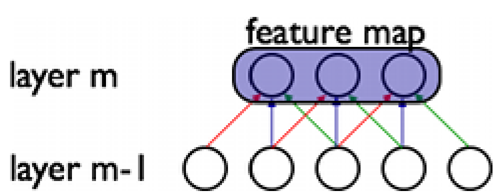
- e.g. $Loss = \sum_m ||x^{(m)} - DECODER(ENCODER(x^{(m)}))||^2$
- Reconstruction: $x' = \sigma(W'\sigma(Wx + b) + d)$

(REF)

Convolutional neural networks (CNNs)

Input image convolutional layer pooling





LeCun et al., 1989

Why should we be bothered with deep learning?

- Learning representations
  - learning features as part of DL algorithms
  - multiple levels of abstraction and complexity (hierarchy)
- Distributed feature representations
  - multi-task or transfer learning (multi-clustering)
  - mitigates the curse of dimensionality, allows for non-local generalisation
  - sparse coding
- Multiple levels of latent variables allow combinatorial sharing of statistical strength

Why does deep learning seem to work?

- the notion of *"cheap learning"*
  - exponentially fewer parameters than "generic" degrees of freedom ("swindle")
  - we take advantage of the special nature of problems at hand:

    the laws of physics select a particular class of functions that are sufficiently "mathematically simple" to allow "cheap learning" to work

    benefitting from *smoothness*, *symmetry*, *invariance*, *locality* (local interactions boosting sparseness)
- *"no-flattening" theorems*
  - "flattening polynomials is exponentially expensive, with 2n neurons required to multiply n numbers using a single hidden layer, a task that a deep network can perform using only ~ 4n neurons"

Henry W. Lin and Max Tegmark, Why does deep and cheap learning work so well?, arXiv:1608.08225

Data representations

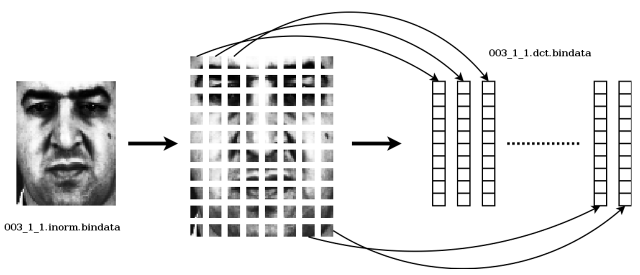
- Multiple ways of representing information – what is the difference? Why should we care?

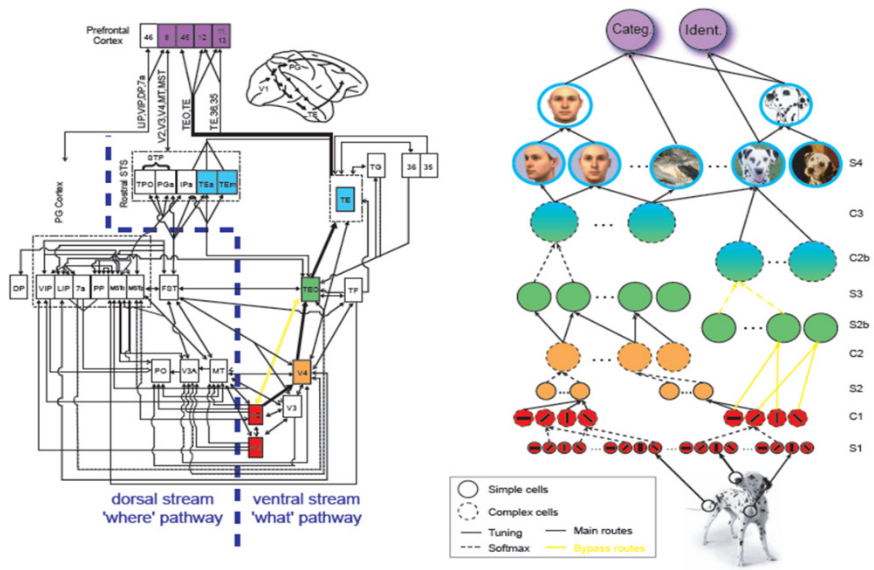
Data parameterisation

12102

10111101000110 (bin)

2f46 (hex)

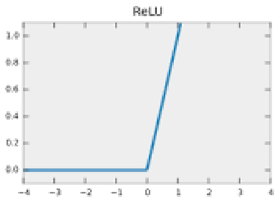




Hypothetical hierarchical representations of visual objects in the brain

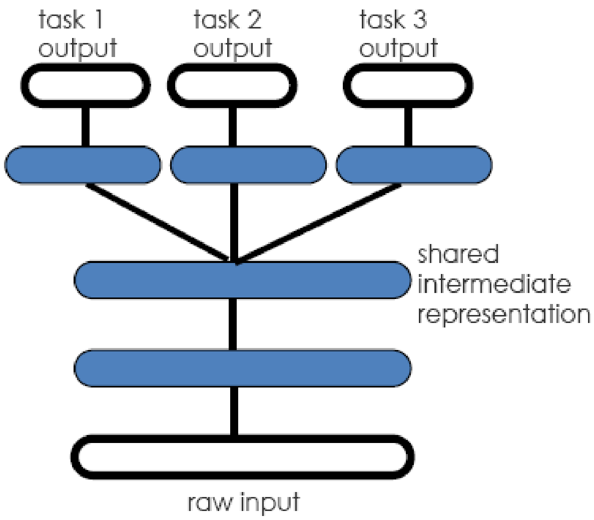
Representation learning in deep models

- The concept of layer-by-layer pretraining
  - greedy layer-wise unsupervised representation learning
    - intuitively, learning about the input distribution should help in learning the *mapping* between the input and output space
    - BUT having two separate phases has *disadvantages*
  - ULTIMATELY, the approach with <u>unsupervised pretraining</u> is largely ***abandoned*** (except word embeddings in NLP)
    - new regularisation techniques: dropout, batch normalisation
    - smaller datasets -> Bayesian methods
    - units with ReLU activation



Transfer learning – sharing factors across tasks

- Assumption that factors explaining the variations in different tasks are shared/common
- Especially low-level features are expected to be the same
- The concept of *one-shot* and *zero-shot* learning
- Zero-shot learning as a specific form of *multi-modal learning* (capturing the relationship between representations in different modalities)

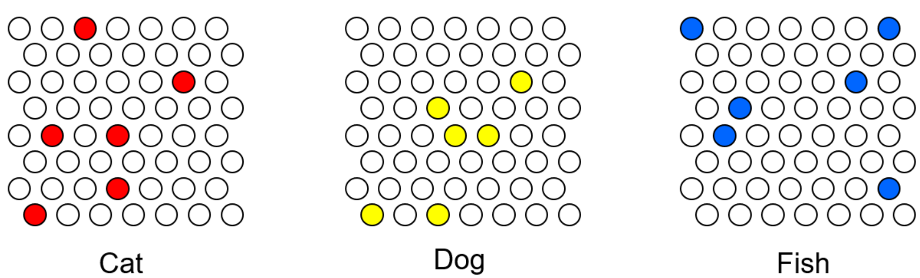


Distributed (and sparse) representations

Information is distributed across many units that account for information about features that are not mutually exclusive…..

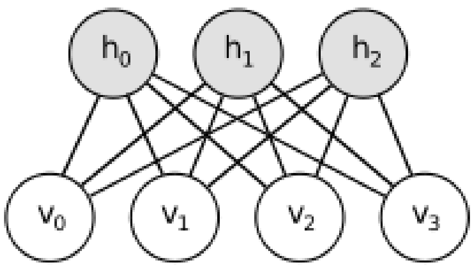
… unlike in clustering with distinct regions where *local generalisation* is observed.

Locality in input space implies different behaviour of the learned function in different regions of data space.



sparse not distributed	not sparse distributed	sparse distributed
0 .2 0 0 0	.1 .8 .7 .5 .7	0 .8 0 .5 0
0 0 0 0 .1	.8 .9 .6 .2 .4	0 0 .6 0 .4
0 0 0 .4 0	.3 .1 .6 .3 .3	.3 0 0 .3 0

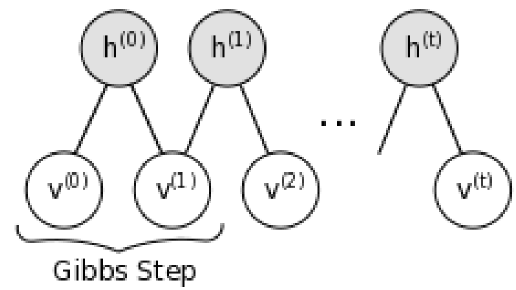
RBM learning with Contrastive Divergence (CD)

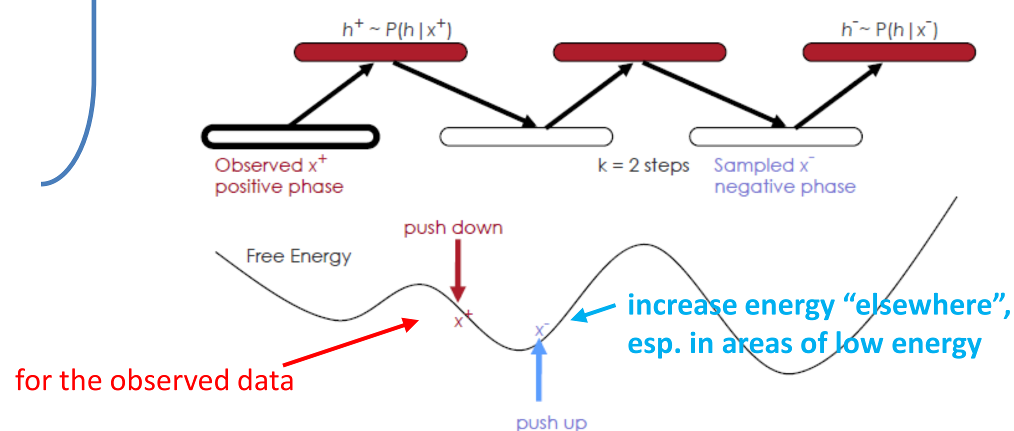


$$P(h_i = 1 \mid \boldsymbol{v}) = \frac{1}{1 + \exp\left(-bias_{h_i} - \boldsymbol{v}^{\mathrm{T}} \mathbf{W}_{:,i}\right)}$$

$$P(v_j = 1 \mid \boldsymbol{h}) = \frac{1}{1 + \exp\left(-bias_{v_j} - \mathbf{W}_{j,:} \boldsymbol{h}\right)}$$

Gibbs sampling





Hinton, 2003

GOOD TO KNOW:

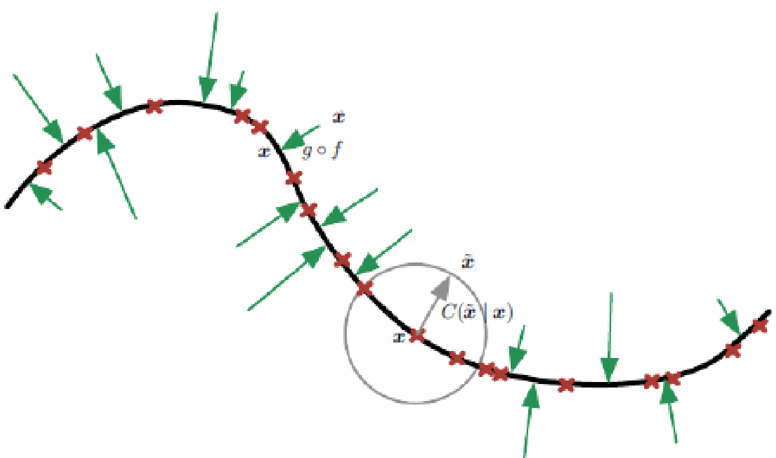
Contrastive Divergence does not optimise the likelihood but it works effectively!

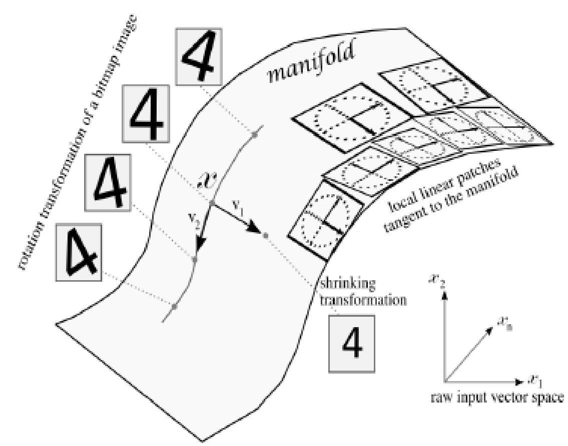
Denoising autoencoders

When training autoencoders there is a **compromise**

I. Need to approximately recover ***x*** – *reconstruction* force

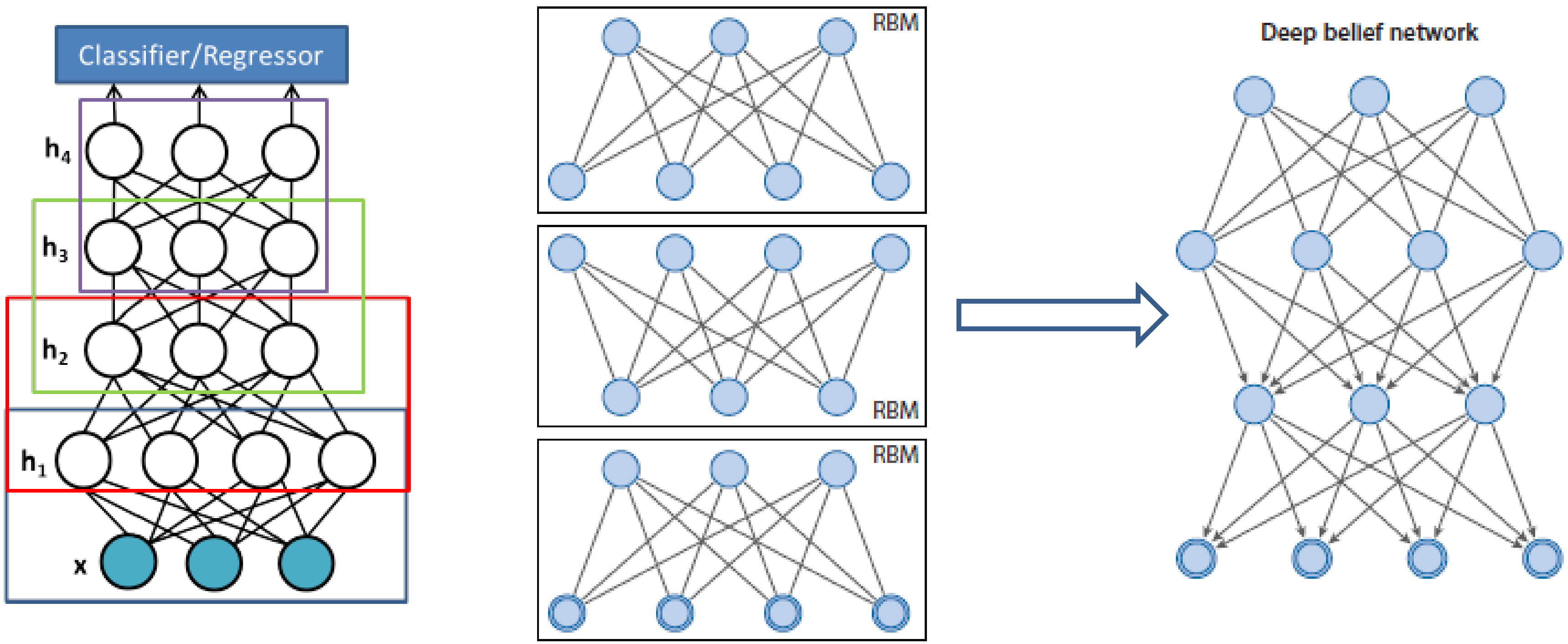
II. Need to satisfy the regularization term – *regularisation* force.



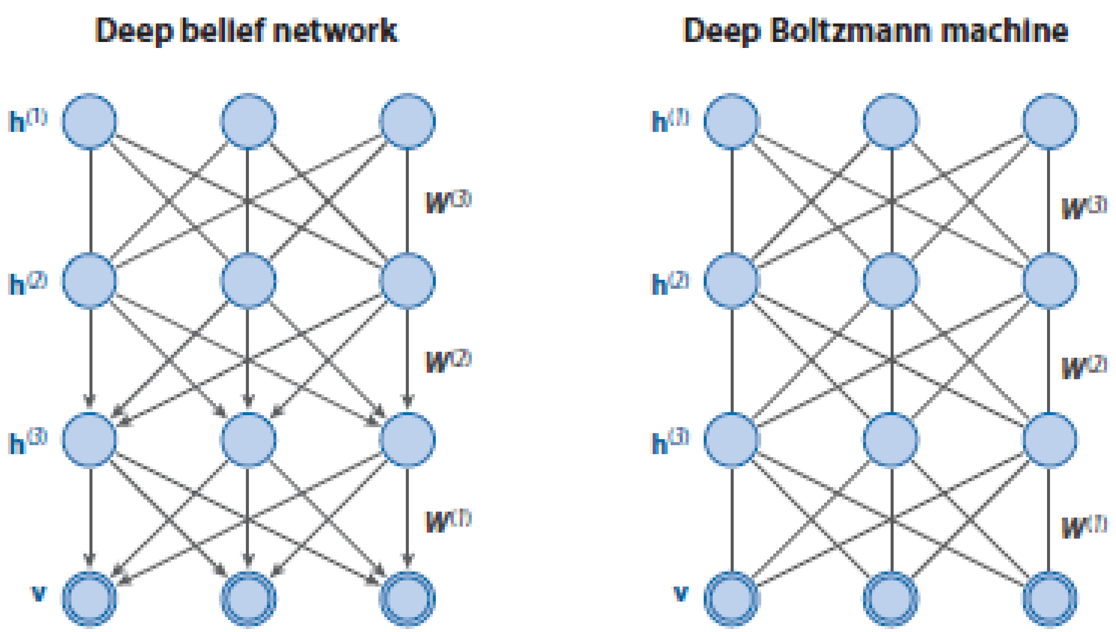


Goodfellow et al.

Deep belief nets



DBN vs DBM



Deep belief network
h(1)
h(2)
h(3)
v
W(3)
W(2)
W(1)
Deep Boltzmann machine
h(1)
h(2)
h(3)
v
W(3)
W(2)
W(1)

Generative adversarial networks (GANs)