REVIEW OF L8-L14

INT301 Bio-computation, Week 13, 2025

Review

- L8: Radial-Basis Function(RBF) Network
- L9-10:Time-series Prediction, Elman Network, Recurrent Network
- L11-L12: PCA and Unsupervised learning
- L13: Self-organizing Feature Map
- L14: Associative Memories & Hopfield network

Review:Radial-Basis Function(RBF) Network

- RBF network can be viewed as a **curve-fitting** or **function approximation** problem.
- For the problem *of* ***curve-fitting (approximation in high-dimensional spaces)***, learning is equivalent to finding an interpolating surface in the space that provides a best fit to the training data, measured by preselected statistical criteria.

Review:Radial-Basis Functions

- The ***radial-basis functions*** technique suggests constructing of interpolation functions *F* of the following form:

  $$F(\mathrm{x}) = \sum_{i=1}^{N} w_i \phi(\| \mathrm{x} - \mathrm{x}_i \|)$$

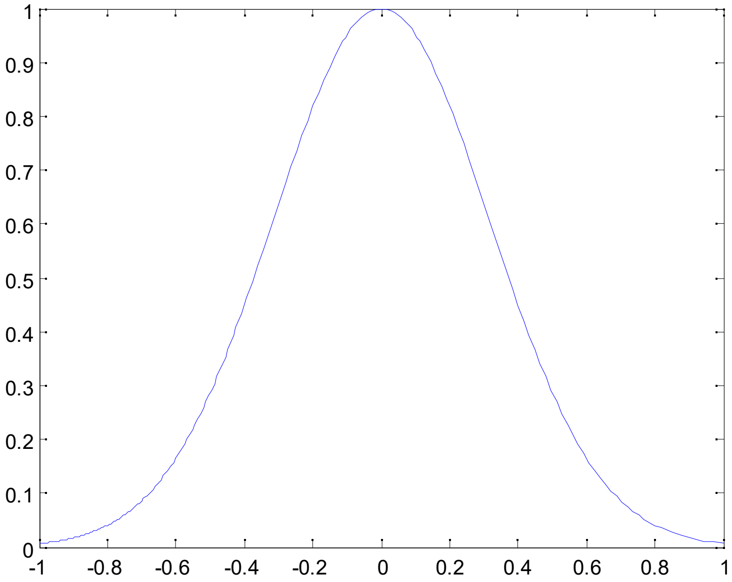
  where: $\phi(\| \mathrm{x} - \mathrm{x}_i \|)$ is a set of nonlinear radial-basis functions, $\mathbf{x}_i$ are the centers of these functions, and ||•|| is the Euclidean norm.
- A **radial basis function (RBF)** is a real-valued function whose value depends only on the distance from the origin
- $$\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$$
- Alternative form: depends on the distance from a ***center c***

  $$\phi(\mathbf{x}, \mathbf{c}) = \phi(\|\mathbf{x} - \mathbf{c}\|)$$
- Any function that satisfies the property is a radial basis function

Review:Radial-Basis Function(RBF) Network

A Gaussian RBF monotonically decreases with distance from the center. Gaussian--like RBFs are local (give a significant response only in a neighborhood near the center) and are more commonly used.

They are also more biologically plausible because their response is finite.



c = 0 and radius r = 1

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2}\right)$$

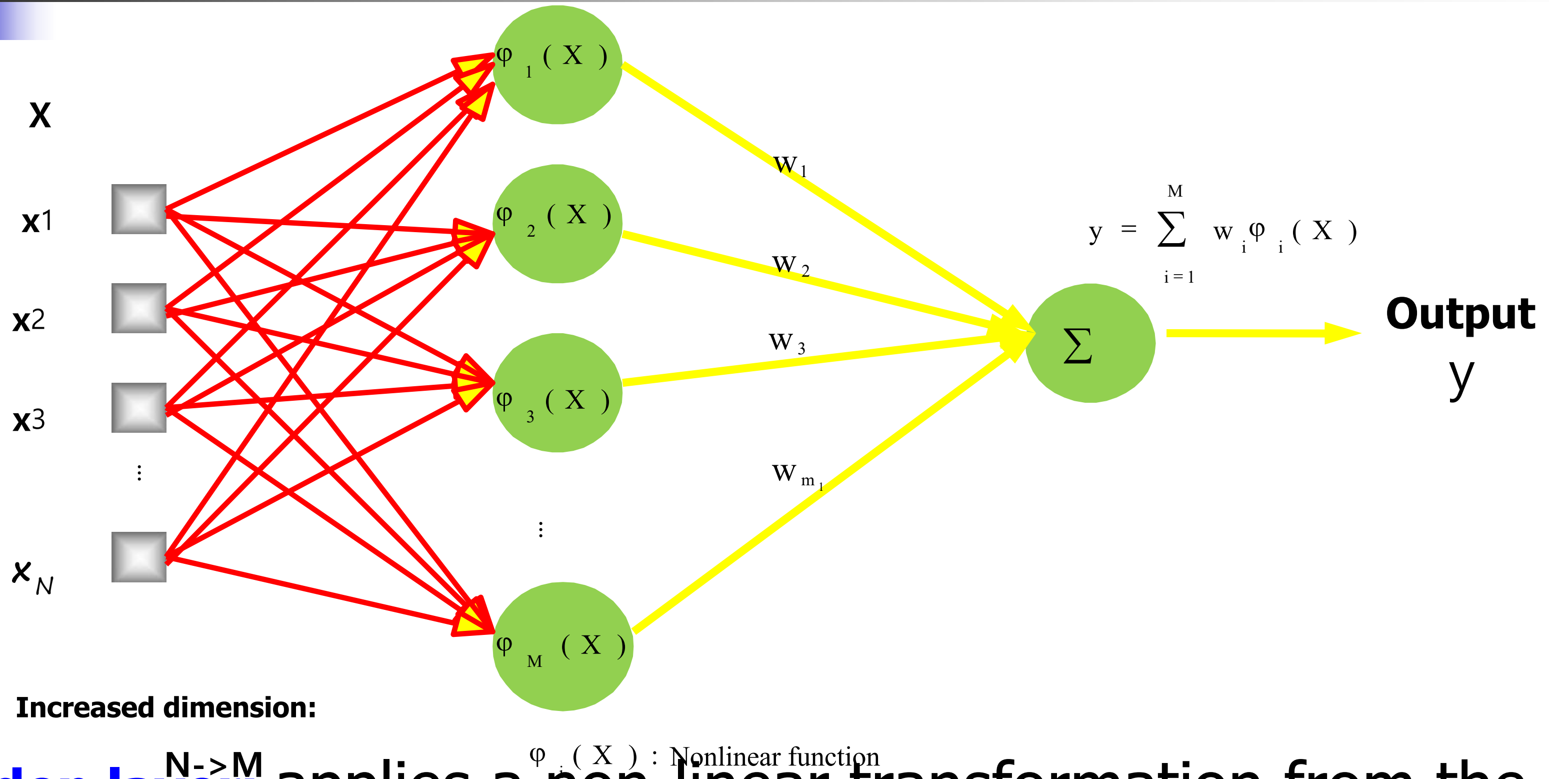
Review:Radial-Basis Function(RBF) Network

$$F(x) = \sum_{i=1}^{N} w_i \exp(-||x - x_i||^2 / 2\sigma_i^2)$$

The important characteristic:

- it is a ***universal approximator*** in that it can approximate arbitrarily well any multivariate continuous function on a compact set, given a sufficiently large number of units.

Review:Radial-Basis Function(RBF) Network



- **Hidden layer:** applies a non-linear transformation from the input space to the hidden space.
- **Output layer:** applies a linear transformation from the hidden space to the output space.

Review:Radial-Basis Function(RBF) Network

- **Φ** is the NxM design matrix of basis functions

$$\phi = \exp(-\frac{\|\mathrm{x}-\mathrm{t}_i\|^2}{2\sigma_i^2})$$

$$\Phi = \begin{bmatrix} \varphi_1(\| x_1 - t_1 \|) & \dots & \varphi_{\mathrm{M}}(\| x_1 - t_M \|) \\ & \dots & \\ \varphi_1(\| x_N - t_1 \|) & \dots & \varphi_M(\| x_N - t_M \|) \end{bmatrix}$$

Review:Radial-Basis Function(RBF) Network

$$\Phi W = \Phi \begin{bmatrix} w_1 \\ \ldots \\ w_\mathrm{M} \end{bmatrix} = \begin{bmatrix} d_1 \\ \ldots \\ d_N \end{bmatrix} = d$$

The unknown weights can be found by solving the following linear matrix equation using pseudo-inverse:

$$W = (\Phi^{\mathrm{T}}\Phi)^{-1}\Phi^{T} d$$

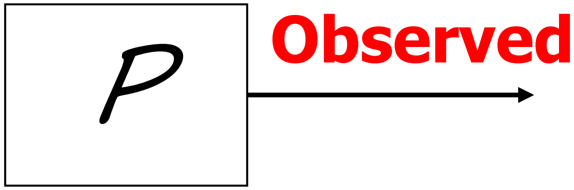
where: W is the weight vector, d is the response vector, and Φ is the **design matrix**

Review:Elman Network

- In statistics and machine learning, a time-series is often described by a sequence of vectors (or scalars) which depend on time t:

$$\left\{x(t_0), x(t_1), \cdots, x(t_{i-1}), x(t_i), x(t_{i+1}), \cdots\right\}$$

It's the output of some process P that we are interested in:



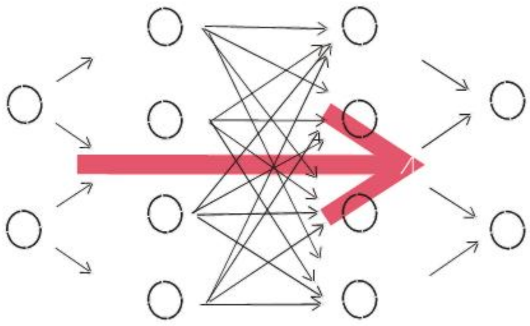
Observed time-series

Unknown dynamics (process)

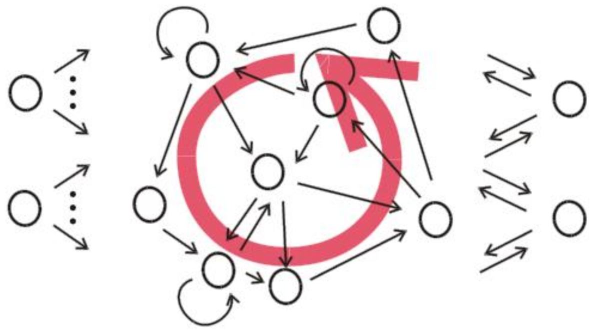
Review:Elman Network

- Sometimes, we require our model to be sensitive to inputs that were presented some time ago.
- In other words, our requirement is not met by a function (no matter how complex it may be) and demands **a model which maintains a *state***, or **memory**, over several presentations.
- Which applications need memory?
  - **sequential input** (e.g. language understanding, robot exploration, ...),
  - **sequential output** (e.g. speech, route planning, ...), or combinations of the above.

Review:Elman Network



- Connections only "from left to right", no connection cycle
- Activation is fed forward from input to output through "hidden layers"
- No memory



- At least one connection cycle
- Activation can "reverberate", persist even with no input
- System with memory
- Mathematically, RNNs implement dynamical systems.

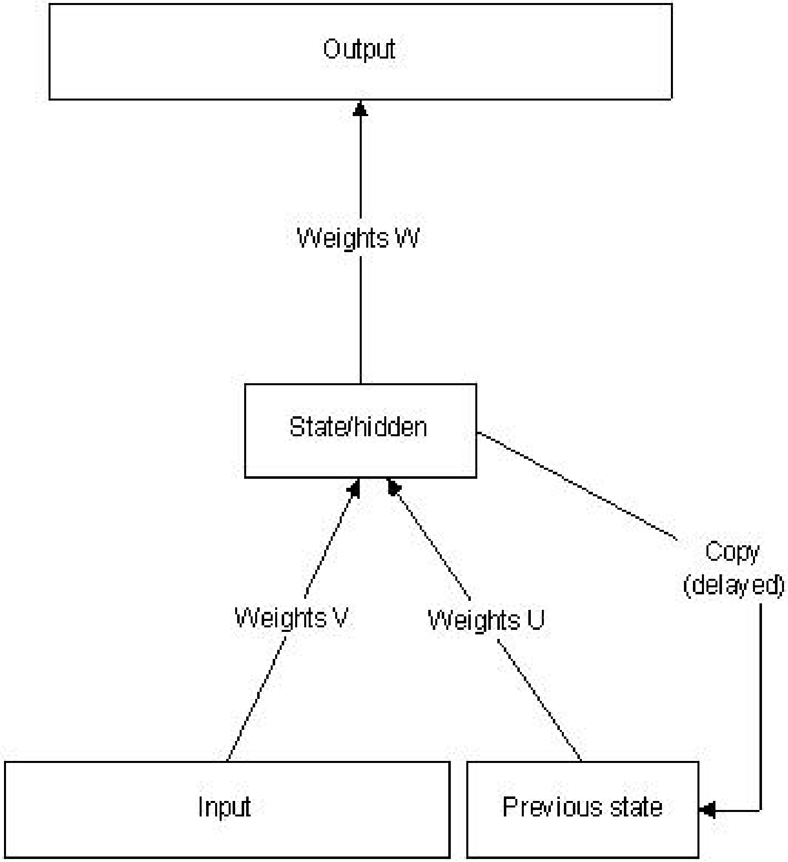
Review:Elman Network

- Elman network which typically distinguishes between **a state-output function and an input-state mapping**.

$$y = F_w(z)$$

$$z^t = F_v(z^{t-1}, x)$$

where *y* is the output, *x* is the input and *z* is the state. *t* is an index in time.

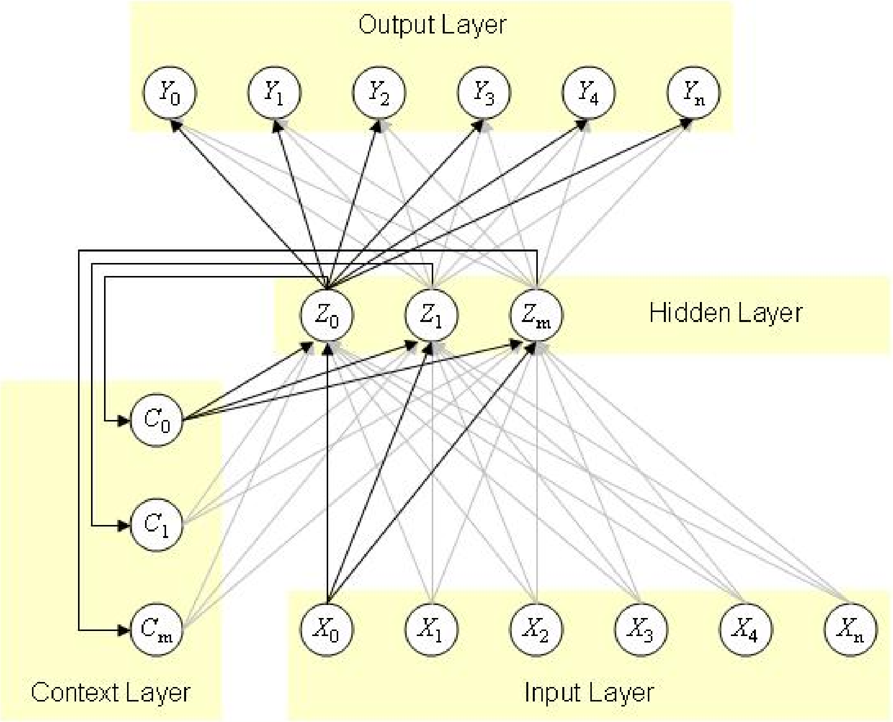


Review:Elman Network

Four layers of Elman network

- **Input layer**
- **Hidden layer** forming internal representation
- **Context layer**

  The value of the context neuron is used as an extra input signal for all the neurons in the hidden layer one time step later. In an Elman network, the weights from the hidden layer to the context layer are set to one and are fixed because the values of the context neurons have to be copied exactly
- **Output layer** linear combination



Review:Recurrent Neuron Network

- The network input at time t:

$$a_h(t) = Ux(t) + Ws(t\ -1)$$

- The activation of hidden unit at time t:

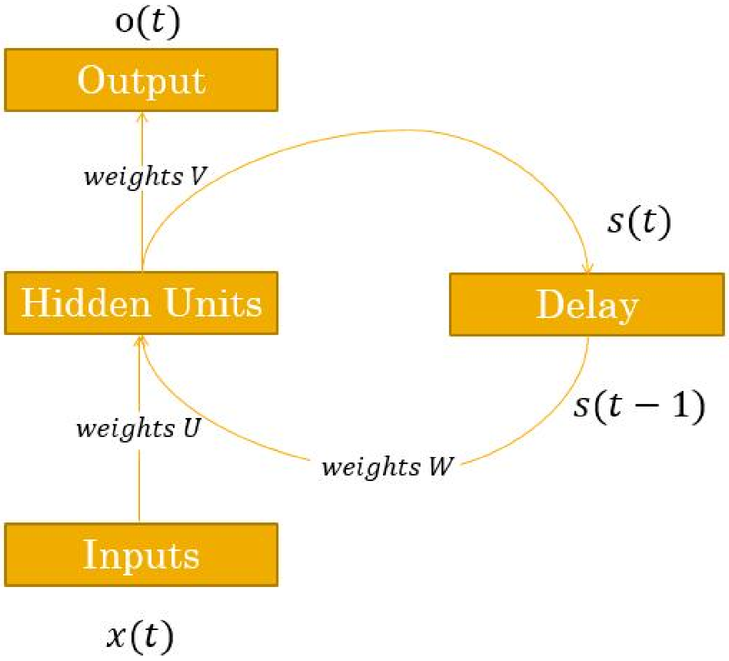
$$s(t) = f_h\big(a_h(t)\big)$$

- The hidden to the output at time t:

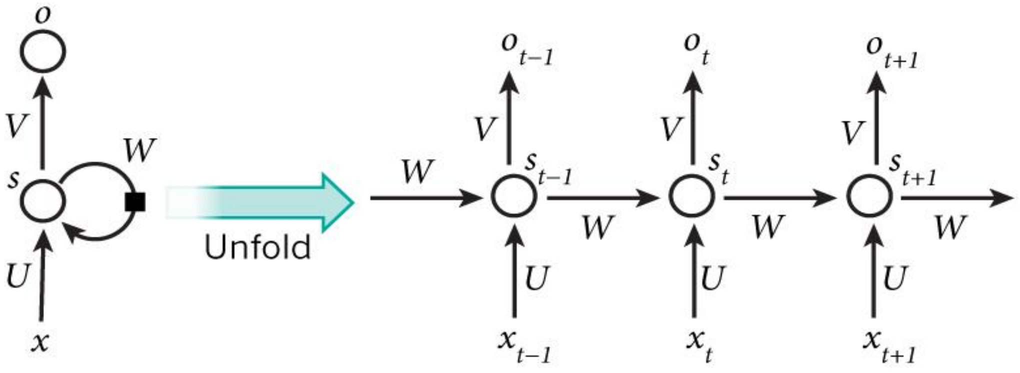
$$a_o(t) = Vs(t)$$

- The output of the network at time t is:

$$o(t) = f_o\big(a_o(t)\big)$$



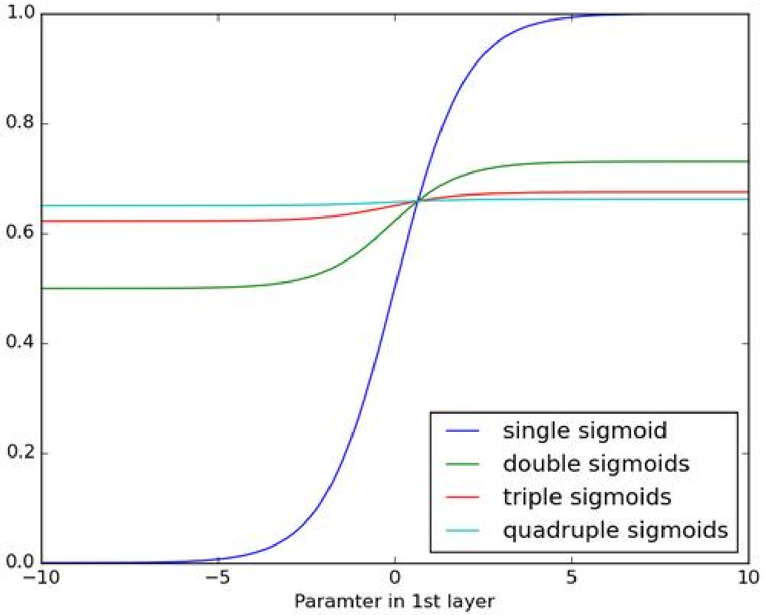
Review:Recurrent Neuron Network



- In RNN errors can be propagated more than 2 layers in order to capture longer history information.
  - This process is usually called unfolding.
  - In an unfolded RNN, recurrent weight is duplicated for arbitrary number of time steps

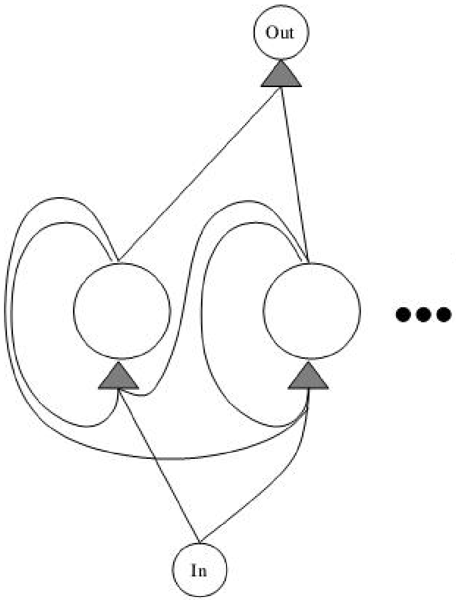
Review:Recurrent Neuron Network

- RNNs trained with BPTT have difficulties learning long-term dependencies (e.g. dependencies between steps that are far apart) due to what is called the vanishing gradients problem.
- The figure shows the effects of applying a sigmoid function over and over again. The data is flattened until, for large stretches, it has no detectable slope.
- Vanishing gradients can become too small for computers to work with or for networks to learn.

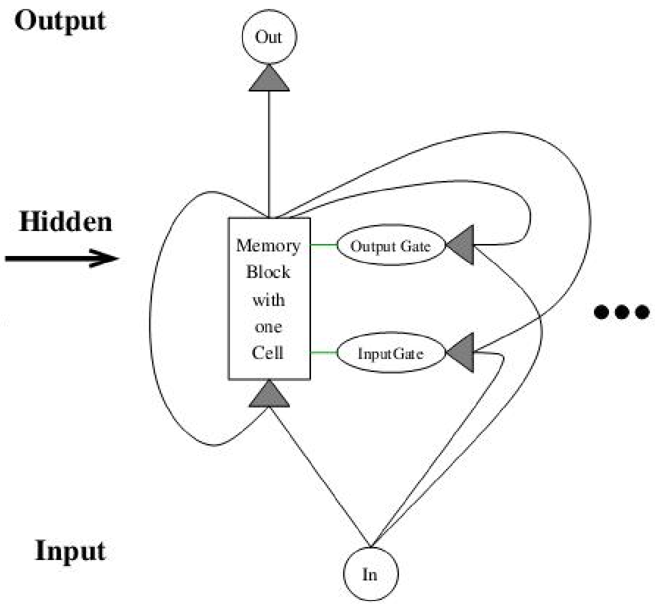


Review:Recurrent Neuron Network

- The basic unit in the hidden layer of an LSTM network is a memory block, it replaces the hidden unit in a traditional RNN.
- A memory block contains one or more memory cell and a pair of adaptive multiplicative gating units which gates input and output to all cells in the block.
- Memory blocks allow cells to share the same gates thus reducing the number of parameters.

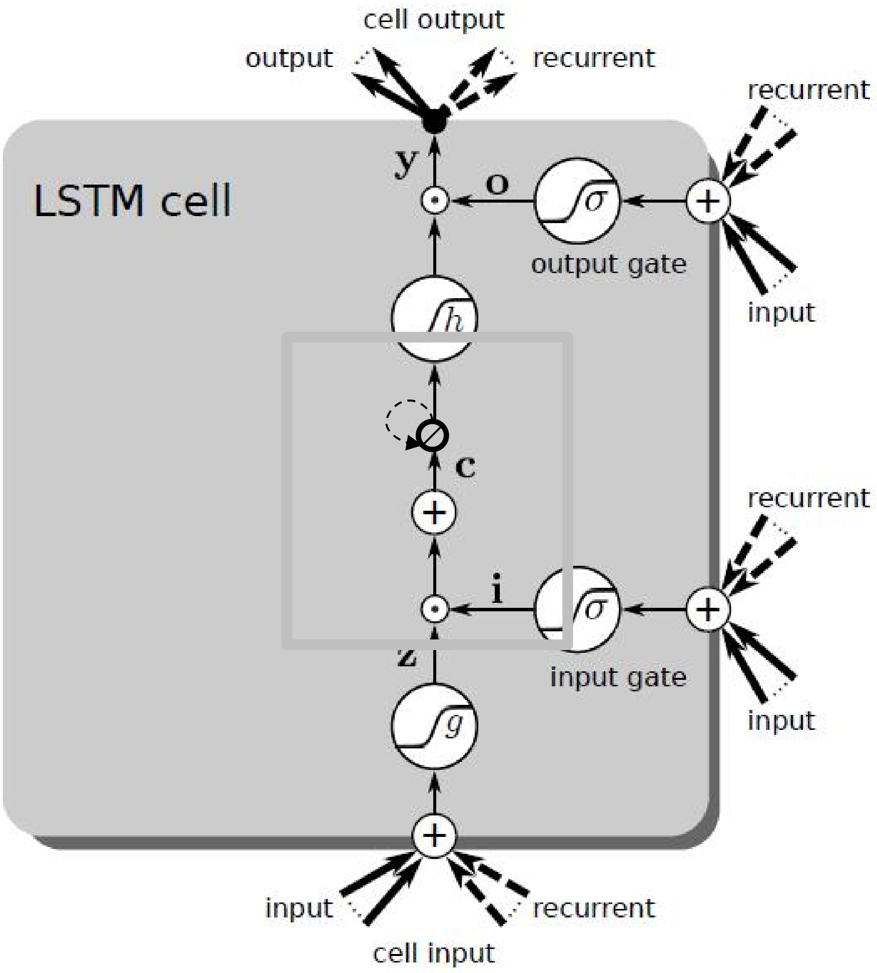


RNN with one FC hidden layer



LSTM network with memory blocks in hidden layer (only one is shown)

LSTM Architecture – CEC



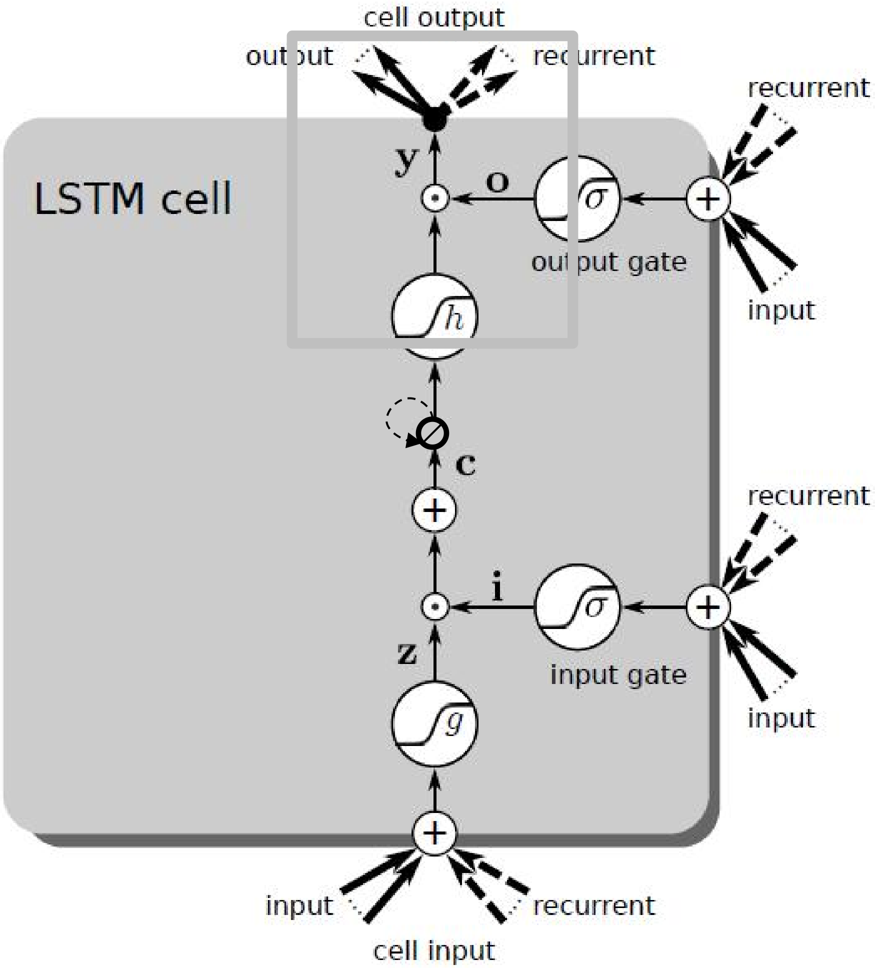
- Each memory cell contains a node with a self-connected recurrent edge of fixed weight one, ensuring that the gradient can pass across many time steps without vanishing – which is called CEC (constant error carousel)

- The CEC solves the vanishing error problem. In the absence of new input or error signals to the cell the CEC local error back flow remains constant, neither growing or decaying.

$$c(t) = z(t) \odot i(t) + c(t-1)$$

$\odot$ is the Hadamard product

Review:Recurrent Neuron Network



- The cell state c is updated based on its current state and 3 inputs: a_z , a_{in} , a_{out}

$$a_z(t) = W_z x(t) + R_z\big(y(t-1)\big)\,, z(t) = g\big(a_z(t)\big)$$

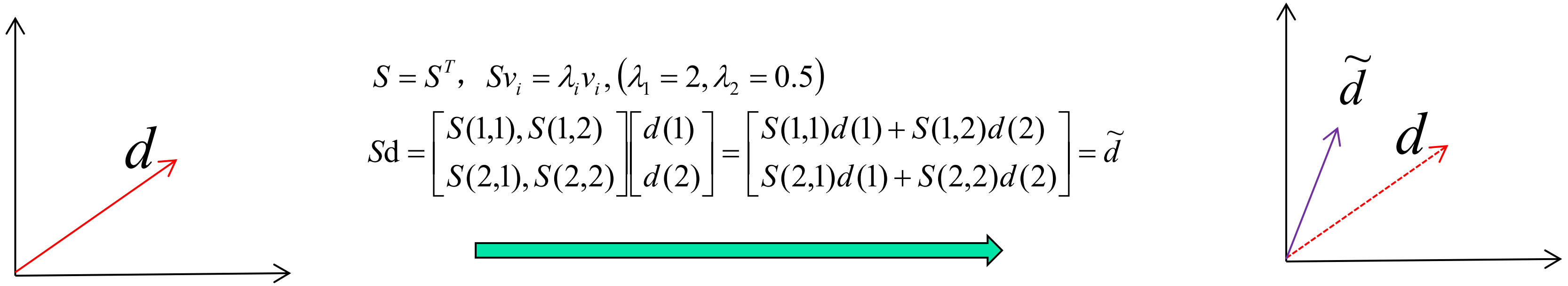
$$a_{in}(t) = W_{in} x(t) + R_{in}\big(y(t-1)\big)\,, i(t) = \sigma\big(a_{in}(t)\big)$$

$$c(t) = z(t) \odot i(t) + c(t-1)$$

$$a_{out}(t) = W_{out} x(t) + R_{out}\big(y(t-1)\big)\,, o(t) = \sigma\big(a_{out}(t)\big)$$

$$y(t) = h\big(c(t)\big) \odot o(t)$$

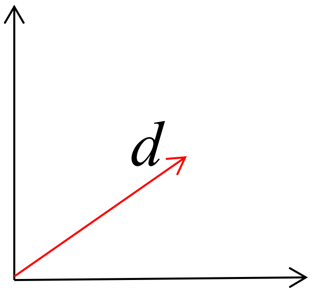
Review:PCA



$$S = S^T,\quad Sv_i = \lambda_i v_i, \left(\lambda_1 = 2, \lambda_2 = 0.5\right)$$

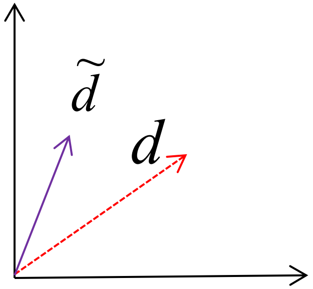
$$Sd = \begin{bmatrix} S(1,1), S(1,2) \\ S(2,1), S(2,2) \end{bmatrix} \begin{bmatrix} d(1) \\ d(2) \end{bmatrix} = \begin{bmatrix} S(1,1)d(1) + S(1,2)d(2) \\ S(2,1)d(1) + S(2,2)d(2) \end{bmatrix} = \tilde{d}$$

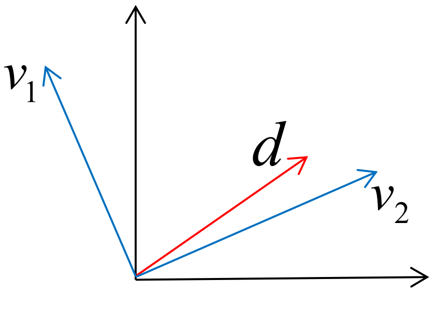
Review:PCA

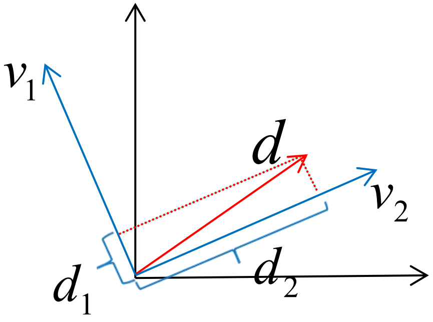


$$S = S^T,\quad Sv_i = \lambda_i v_i, (\lambda_1 = 2, \lambda_2 = 0.5)$$

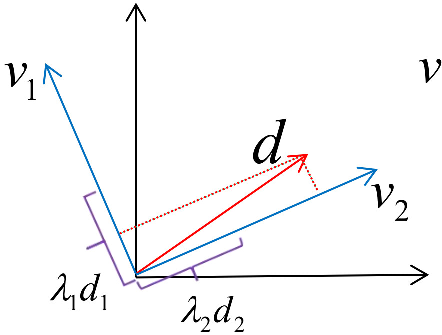
$$\mathrm{Sd} = \begin{bmatrix} S(1,1), S(1,2) \\ S(2,1), S(2,2) \end{bmatrix} \begin{bmatrix} d(1) \\ d(2) \end{bmatrix} = \begin{bmatrix} S(1,1)d(1) + S(1,2)d(2) \\ S(2,1)d(1) + S(2,2)d(2) \end{bmatrix} = \tilde{d}$$



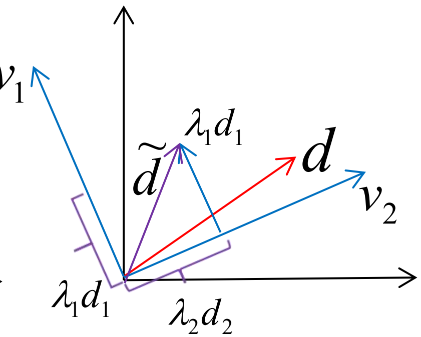




Projection on the eigenvectors



Scaling with eigenvalues



Vector addition

Review:PCA

- **Theorem**: If $\mathrm{S} \in \mathbb{R}^{m \times m}$ is a **symmetric** matrix, there exists an **eigen decomposition**, where ***Q*** is **orthogonal:**

$$S = Q\Lambda Q^T$$

  - $\boldsymbol{Q^{-1} = Q^T}$
  - Columns of ***Q*** are normalized eigenvectors
  - Columns are orthogonal.
  - (everything is real)

Review:PCA

For an *m*× *n* matrix **A** of rank *r*, there exists a factorization (Singular Value Decomposition = **SVD**) as follows:

$$A = \underset{m\times m}{U}\;\underset{m\times n,\ \text{nonnegative real}}{\Sigma}\;\underset{n\times n}{V^T}$$

The columns of ***U*** are orthogonal eigenvectors of ***AA^T***.

The columns of ***V*** are orthogonal eigenvectors of ***A^TA***.

Eigenvalues $\lambda_1 \ldots \lambda_r$ of ***AA^T*** are the eigenvalues of ***A^TA***.

$$\sigma_i = \sqrt{\lambda_i}$$

$$\Sigma = diag(\sigma_1 \ldots \sigma_r)$$

Singular values.

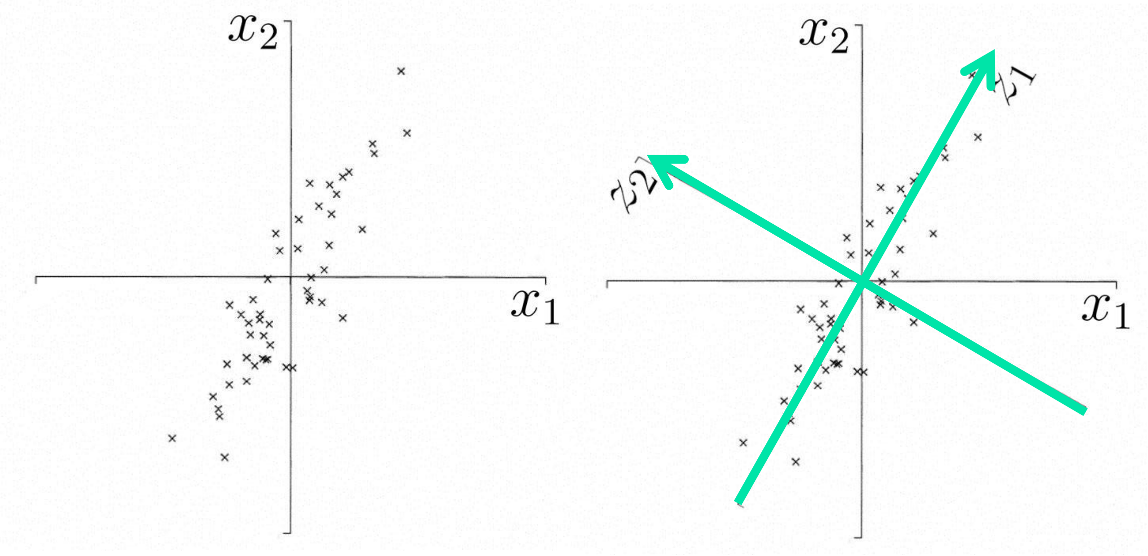
Review:PCA

- Each dimensionality reduction technique finds an *appropriate transformation* by satisfying certain criteria (e.g., *information loss, data discrimination*, etc.)

- The goal of PCA is to reduce the dimensionality of the data while ***retaining as much as possible of the variation present in the dataset***.

Review:PCA

- Motivation
  - Find bases which has high variance in data
  - Encode data with small number of bases with low MSE



- First PC is direction of maximum variance
- Subsequent PCs are orthogonal to 1st PC and describe maximum residual variance

Review:Unsupervised Learning

- A simple principle was proposed by Hebb in 1949 in the context of biological neurons
- Hebbian principle

When a neuron repeatedly excites another neuron, then the threshold of the latter neuron is decreased, or the ***synaptic weight between the neurons is increased***, in effect increasing the likelihood of the second neuron to excite

- Hebbian learning rule $\Delta w_{ji} = \eta y_j x_i$
  - There is no desired or target signal required in the Hebb rule, hence it is ***unsupervised learning***

Review:Unsupervised Learning

- Consider the update of a single weight **w** (x and y are the pre- and post-synaptic activities)

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta x(n)y(n)$$

- For a linear activation function

$$\mathbf{w}(n+1) = \mathbf{w}(n)[1 + \eta x(n)x^T(n)\ ]$$

  - **Weights increase without bounds.** If initial weight is negative, then it will increase in the negative. If it is positive, then it will increase in the positive range

- Hebbian learning is intrinsically unstable, unlike error-correction learning with BP algorithm

Review:Unsupervised Learning

- The simple Hebbian rule causes the weights to increase (or decrease) without bounds
- The weights need to be normalized to one as

$$\boxed{w_{ji}(n+1) = \frac{w_{ji}(n) + \eta x_i(n) y_j(n)}{\sqrt{\sum_i [w_{ji}(n) + \eta x_i(n) y_j(n)]^2}}}$$

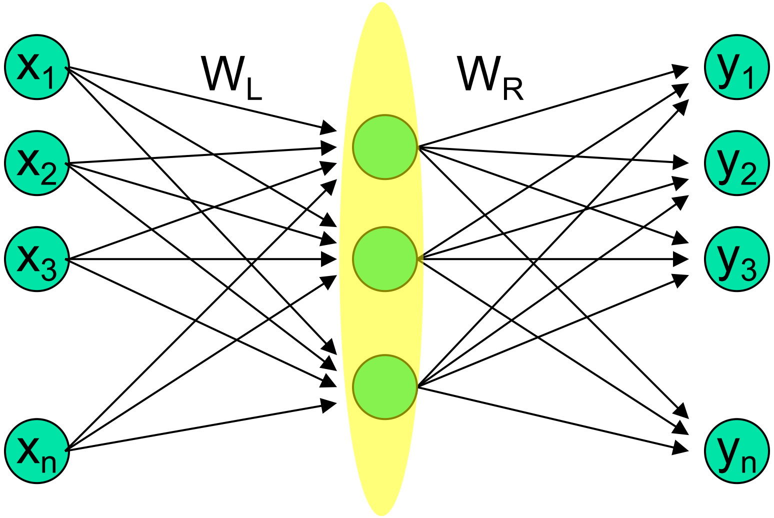
- Oja proves that, for small η<<1, the above normalization can be approximated as:

$$\boxed{w_{ji}(n+1) = w_{ji}(n) + \eta y_j(n)[x_i(n) - y_j(n) w_{ji}(n)]}$$

  - This is Oja's rule, or the normalized Hebbian rule
  - It involves a '**forgetting term**' that prevents the weights from growing without bounds

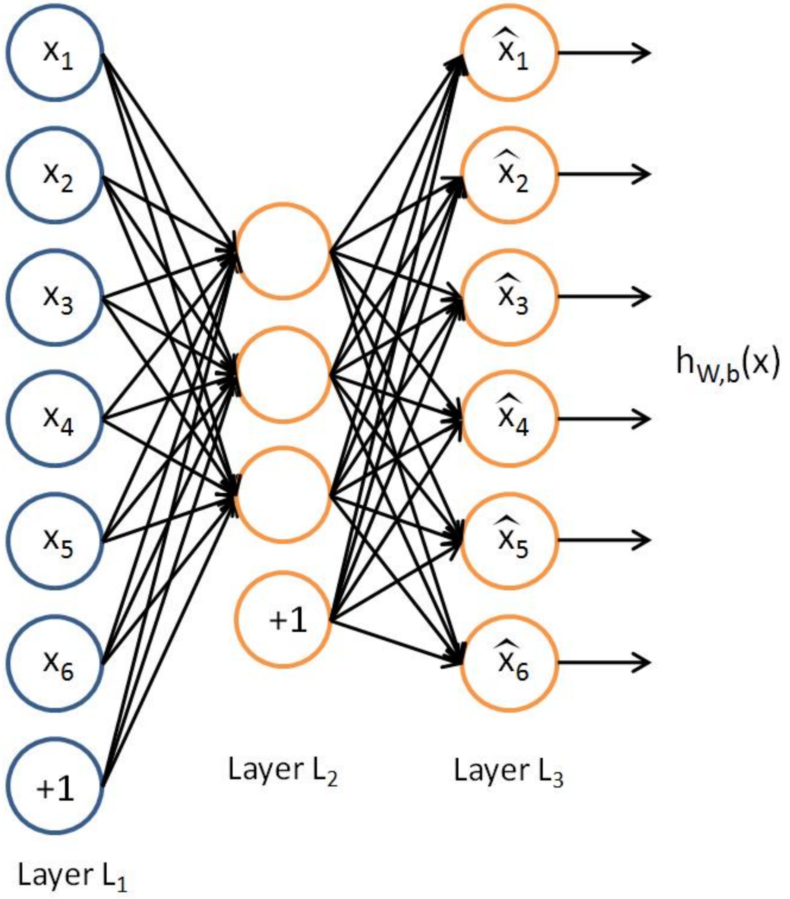
Review:Unsupervised Learning

- Multi-layer networks with bottleneck layer



  - Train using auto-associative output: e = x – y
  - WL spans the subspace of the first m principal eigenvectors.

Review:Unsupervised Learning



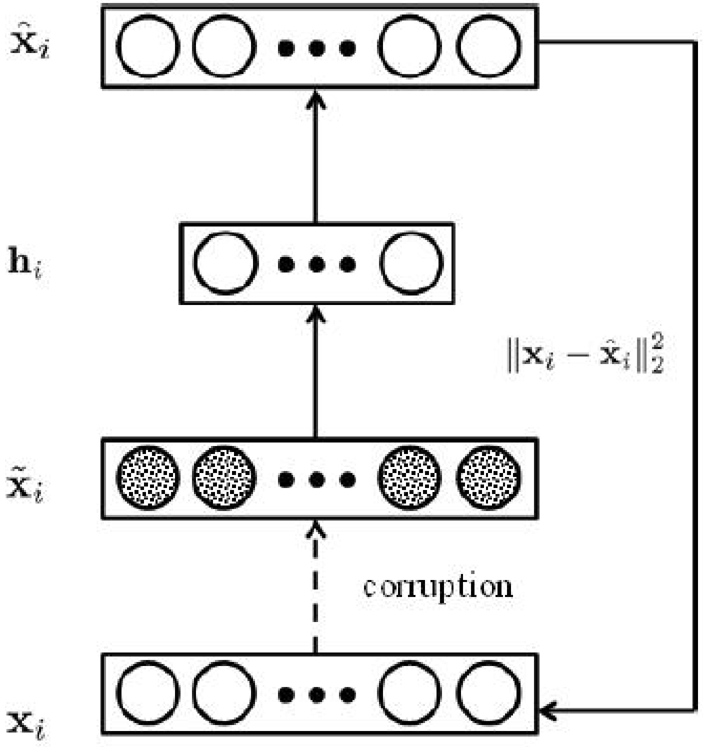
An Autoencoder is a feedforward neural network that learns to predict the input itself in the output.

$$y^{(i)} = x^{(i)}$$

- The input-to-hidden part corresponds to an encoder
- The hidden-to-output part corresponds to a decoder.

Review:Unsupervised Learning

- By adding stochastic noise, it can force auto-encoder to learn more robust features.
- The loss function



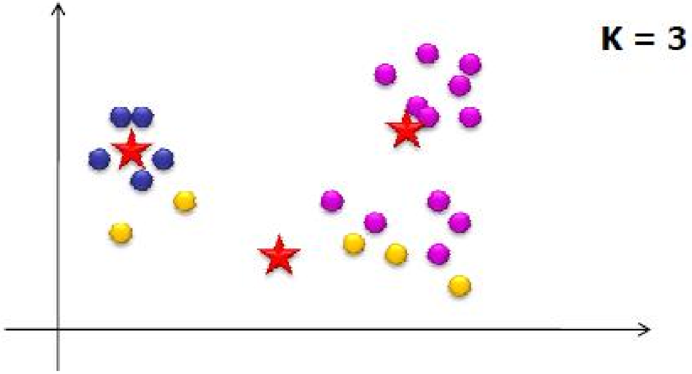
$$\mathbf{h}^{(\ell)} = \sigma(\mathbf{W}_1\tilde{\mathbf{x}}^{(\ell)} + \mathbf{b}_1)$$

$$\hat{\mathbf{x}}^{(\ell)} = \sigma(\mathbf{W}_2\mathbf{h}^{(\ell)} + \mathbf{b}_2)$$

$$\min_{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}_1,\mathbf{b}_2} \sum_{\ell} \|\mathbf{x}^{(\ell)} - \hat{\mathbf{x}}^{(\ell)}\|_2^2 + \lambda\left(\|\mathbf{W}_1\|_F^2 + \|\mathbf{W}_2\|_F^2\right)$$

Review: K-means clustering

- Choose the number of clusters - K
- Randomly choose initial positions of K centroids
- Assign each of the points to the "nearest centroid" (depends on distance measure)
- Re-compute centroid positions
- If solution converges → Stop!



Review:K-means clustering

- A good clustering method will produce high quality clusters with
  - high intra-class similarity
    - similar to one another within the same cluster
  - low inter-class similarity
    - dissimilar to the objects in other clusters

- The quality of a clustering method is also measured by its ability to discover some or all of the hidden patterns

Review:K-means clustering

- Distances are normally used to measure the similarity or dissimilarity between two data objects
- *Minkowski distance*:

$$d(i,j) = \sqrt[q]{(|x_{i_1} - x_{j_1}|^q + |x_{i_2} - x_{j_2}|^q + ... + |x_{i_p} - x_{j_p}|^q)}$$

where *i* = (x_{i1}, x_{i2}, …, x_{ip}) and *j* = (x_{j1}, x_{j2}, …, x_{jp}) are two *p*-dimensional data objects, *q* is a positive integer

- If q = 1, d is Manhattan distance

$$d(i,j) = |x_{i_1} - x_{j_1}| + |x_{i_2} - x_{j_2}| + ... + |x_{i_p} - x_{j_p}|$$

- If q = 2, d is Euclidean distance:

$$d(i,j) = \sqrt{(|x_{i_1} - x_{j_1}|^2 + |x_{i_2} - x_{j_2}|^2 + ... + |x_{i_p} - x_{j_p}|^2)}$$

Review:Unsupervised Learning

- **Competitive learning** means that only a single neuron from each group fires at each time step
- Output units compete with one another.
- These are **winner-takes-all (WTA)** units
- Competition is important for neural networks
  - Competition between neurons has been observed in biological nerve systems
  - Competition is important in solving many problems

Simple Competitive Learning

- Winner:

$$h_j = \sum_i w_{ji} x_i$$

Note: the inner product of two normal vectors is related to the cosine of the angle between them

$$w_{j*} \cdot x \geq w_j \cdot x \qquad \forall j \neq j^*$$

Winner = output node whose incoming weights are the shortest Euclidean distance from the input vector

- Lateral inhibition

Simple Competitive Learning

- (One possible) update rule for all neurons:

$$\Delta w_{ji} = \eta y_j \left(x_i - w_{ji} \right)$$

$$\begin{cases} y_{j*} = 1 \\ y_j = 0 \qquad if \qquad j \neq j* \end{cases}$$

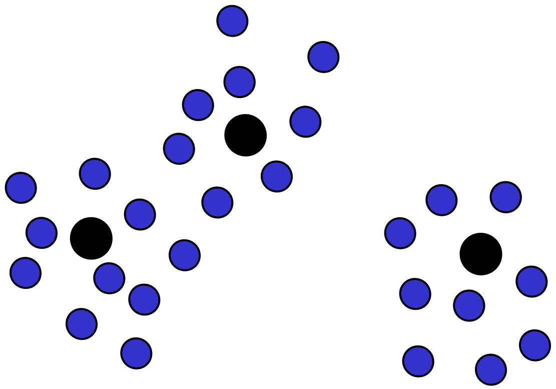
The neuron with largest activation is then adapted to be more likely the input that caused the excitation

Only the incoming weights of the winner node are modified.

Some units may never or rarely become a winner, and so weight vector may not be updated→**DEAD UNIT**

Review:Unsupervised Learning

- initialize K prototype vectors
- present a single example
- identify the closest prototype, i.e., the so-called ***winner***
- move the winner even closer towards the example



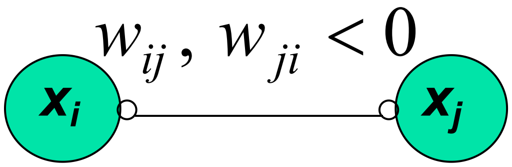
Intuitively clear, plausible procedure

- places prototypes in areas with high density of data
- identifies the most relevant combinations of features

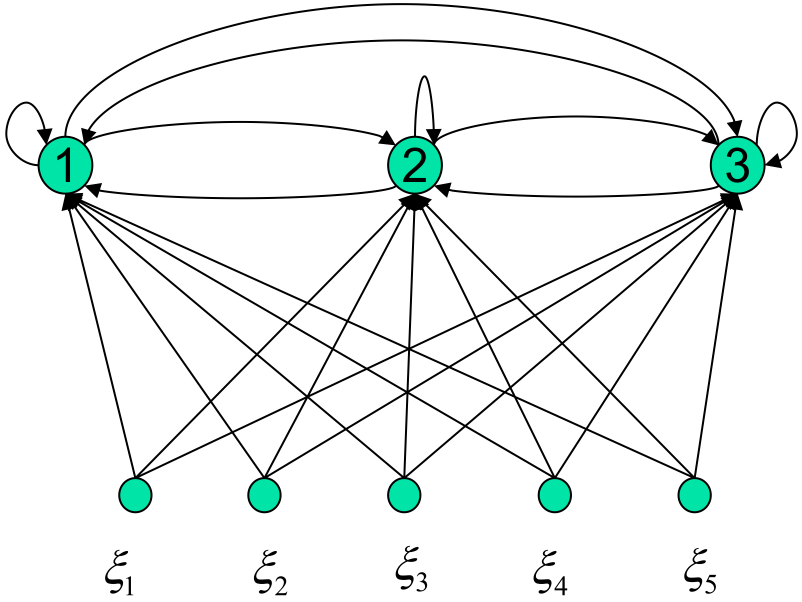
Review:Unsupervised Learning

Lateral inhibition

output of each node feeds to others through inhibitory connections (with negative weights)

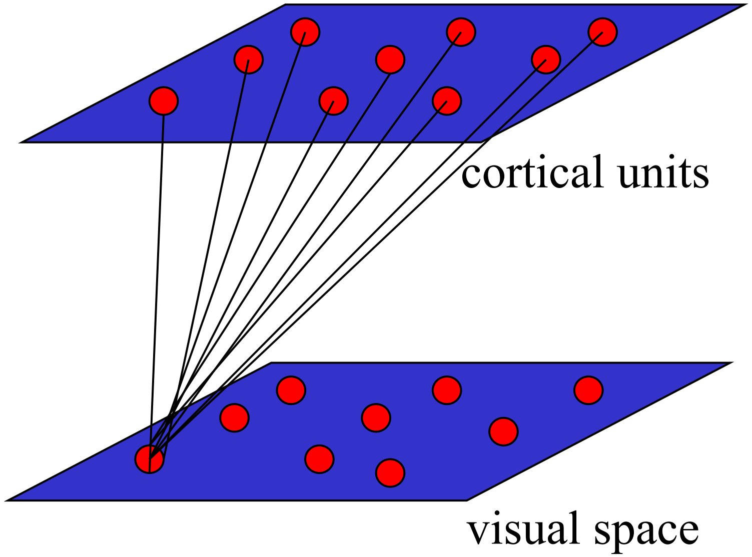


A specific competitive net that performs Winner Take All (WTA) competition is the **Maxnet**



Review: Self-organizing Feature Map

- Biologically motivated: how can activity-based learning using highly interconnected circuits lead to orderly mapping of visual stimulus space onto cortical surface?
- 2 layer network each cortical unit fully connect to visual space via Hebbian units
- Interconnections of cortical units described by ‘Mexican-hat’ function: short-range excitation and long-range inhibition



Review:Self-organizing Feature Map

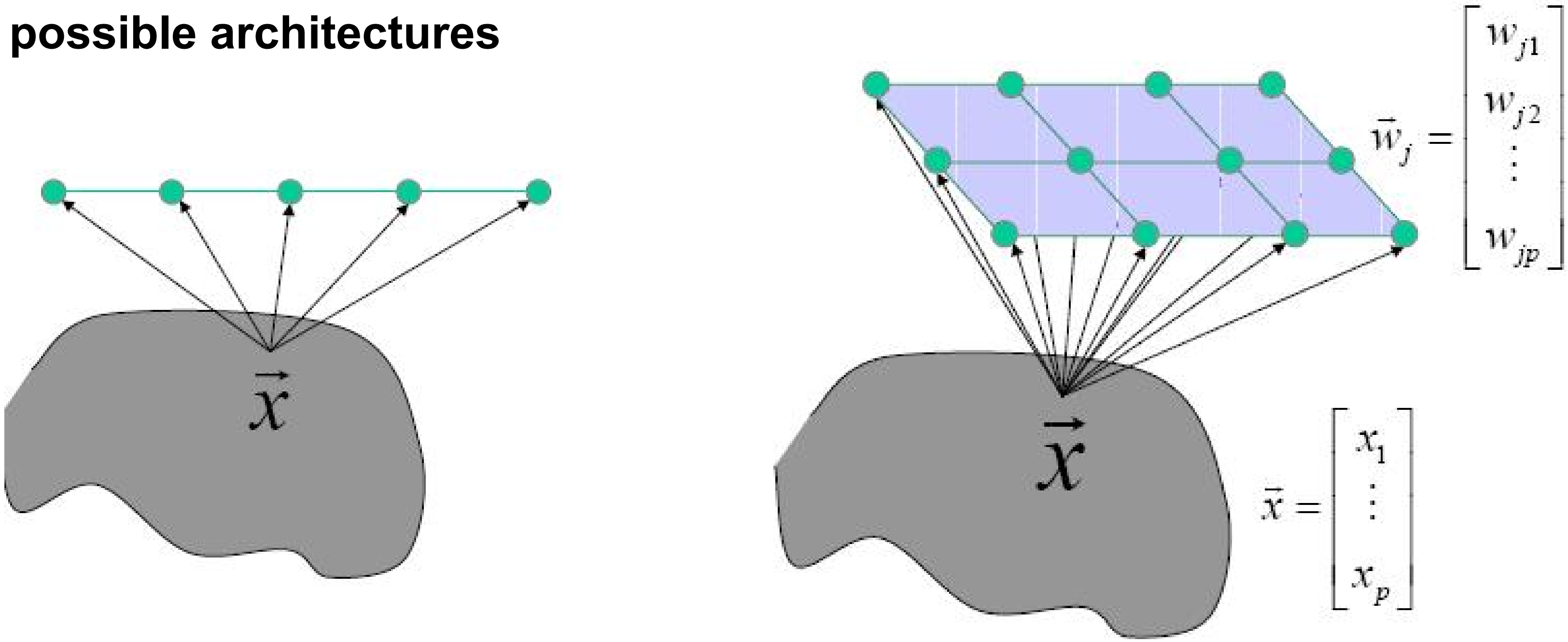
- Activity-based self-organization (von der Malsburg): after learning, a topographic map appears. However, input dimension is the same as output dimension

- Kohonen simplified this model and called it Kohonen's self-organizing map (SOM) algorithm
  - more general as it can perform **dimensionality reduction**
  - SOM can be viewed as a **vector quantization** type algorithm

Review:Self-organizing Feature Map

- The idea in an SOM is to transform an input of arbitrary dimension into a 1 or 2 dimensional discrete map

Two possible architectures

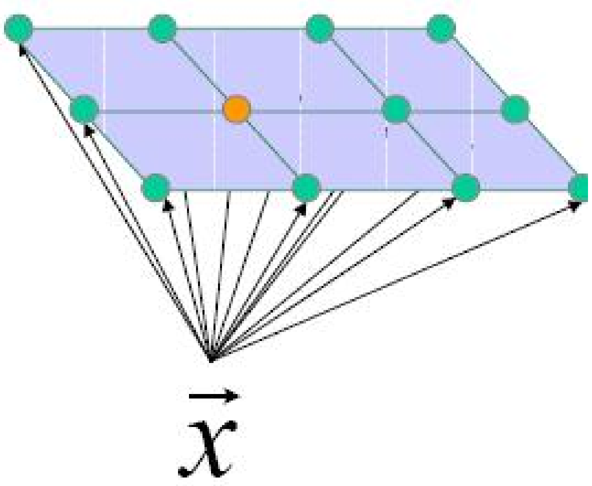


$\vec{x}$
$\vec{x}$
$\vec{w}_j = \begin{bmatrix} w_{j1} \\ w_{j2} \\ \vdots \\ w_{jp} \end{bmatrix}$
$\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$

Review:Self-organizing Feature Map

- Once weights of net initialized, algorithm comprises 3 processes:
- 1. Competition
  - Given an input pattern, outputs compete to see who is winner based on a discriminant function (e.g. similarity of input vector and weight vector)
- 2. Cooperation
  - Winning neuron determines spatial location of a topological neighborhood within which output neurons excited
- 3. Synaptic Adaptation
  - Excite neurons adapt weights so that value of discriminant function increases (a similar input would result in enhanced response from winner)

Review:Self-organizing Feature Map



Winner neuron

$$(1) \qquad j^* = argmax_j\left(w_j^T x\right)$$

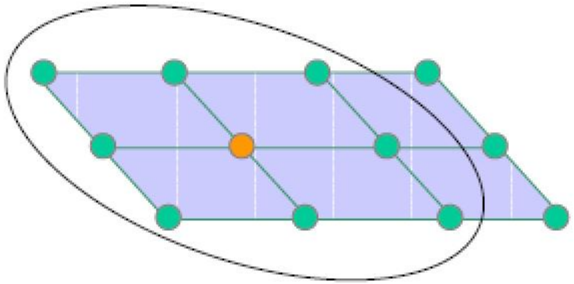
$$(2) \qquad j^* = argmin_j\left(\left\|x - w_j\right\|\right)$$

A continuous input space of activation patterns is mapped onto a discrete output space of neurons by a process of competition among the neurons in the network.

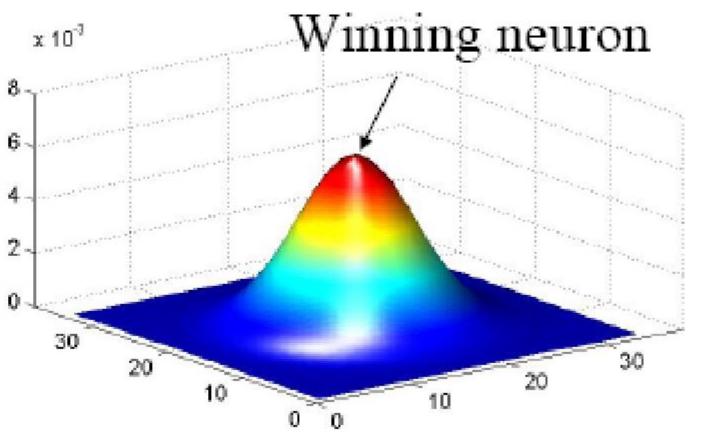
Review:Self-organizing Feature Map

The topological neighborhood *$h_{j,i}$*

- symmetric around the winning neuron and achieve its maximum value at the winning neuron
- the amplitude decreases monotonically with the increasing lateral distance



$$h_{j,i} = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2}\right)$$



Review:Self-organizing Feature Map

- The topological **neighborhood** *$h_{j,i}$ shrinks with time*

$$\sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_1}\right) \qquad h_{j,i}(t) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(t)}\right)$$

- **Neighbors** of the winning node are also allowed to update, even if they are not close to winning!

Adaptive Process

Update the weights in relation to the inputs

$$\mathrm{w}_j(t+1) = \mathrm{w}_j(t) + \boxed{\eta(t)}\boxed{h_{j,i}}(t)(\mathrm{x} - \mathrm{w}_j(t))$$

Learning rate

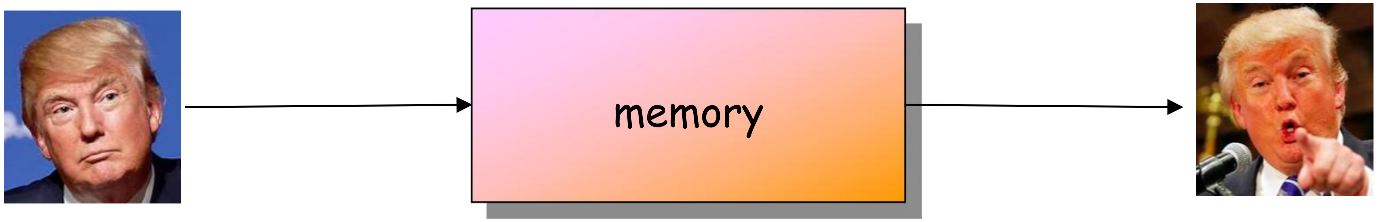
$$\eta(t) = \eta_0 \exp\left(-\frac{t}{\tau_1}\right)$$

Neighborhood function

$$h_{j,i}(t) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(t)}\right)$$

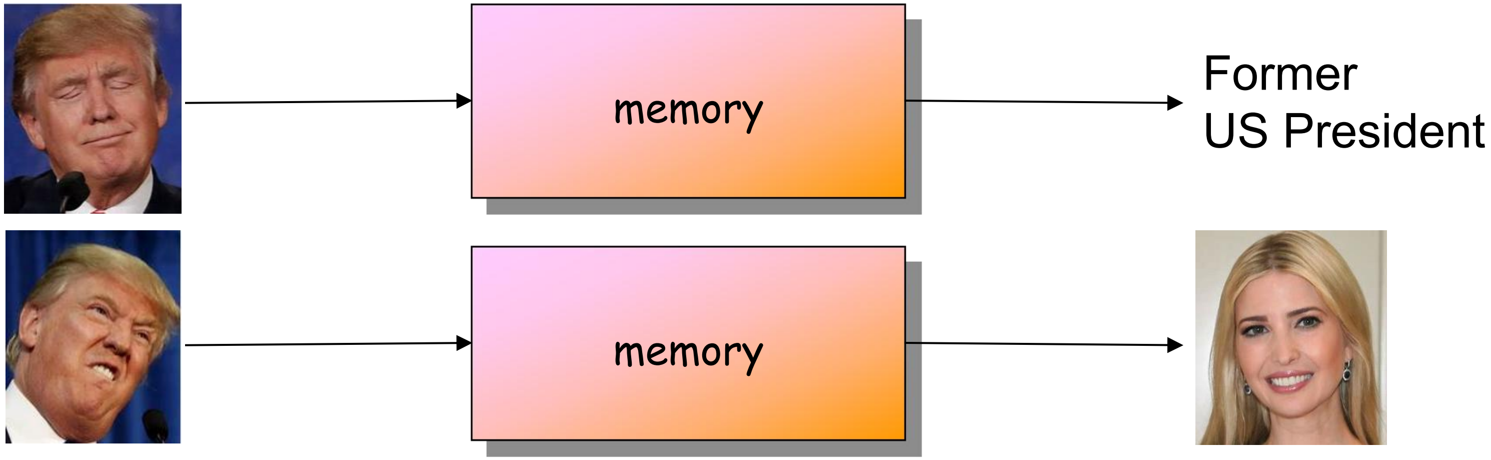
Review:Hopfield network

Auto-association



memory

Hetero-association



memory

Former
US President

memory

Review:Hopfield network

- Similar to Hebbian learning for classification
- Algorithm: (bipolar or binary patterns)
  - For each training samples ***s*:*t*** $\Delta w_{ij} = s_i \cdot t_j$
  - Δw_{ij} increases if both input and output are ON (binary) or have the same sign (bipolar)
- If $\Delta w_{ij} = 0$ initially, then after updates for all P training patterns

$$w_{ij} = \sum_{p=1}^{P} s_i(p) t_j(p) \quad W = \{ w_{ij} \}$$

- Instead of obtaining ***W*** by iterative updates, it can be computed from the training set by calculating the outer product of ***s*** and ***t***.

Review:Hopfield network

- **Outer product**: Let ***s*** and ***t*** be **row** vectors.

  Then for a particular training pair ***s:t***

$$\Delta W(p) = s^T(p) \otimes t(p) = \begin{bmatrix} s_1 \\ \\ \\ s_n \end{bmatrix} [t_1, \ldots\ldots t_m] = \begin{bmatrix} s_1 t_1 \ldots\ldots s_1 t_m \\ s_2 t_1 \ldots\ldots s_2 t_m \\ \\ s_n t_1 \ldots\ldots s_n t_m \end{bmatrix} = \begin{bmatrix} \Delta w_{11} \ldots\ldots \Delta w_{1m} \\ \\ \\ \Delta w_{n1} \ldots\ldots \Delta w_{nm} \end{bmatrix}$$

  and $W(P) = \sum_{p=1}^{P} s^T(p) \otimes t(p)$

- It involves 3 nested loops p, i, j (order of p is irrelevant)

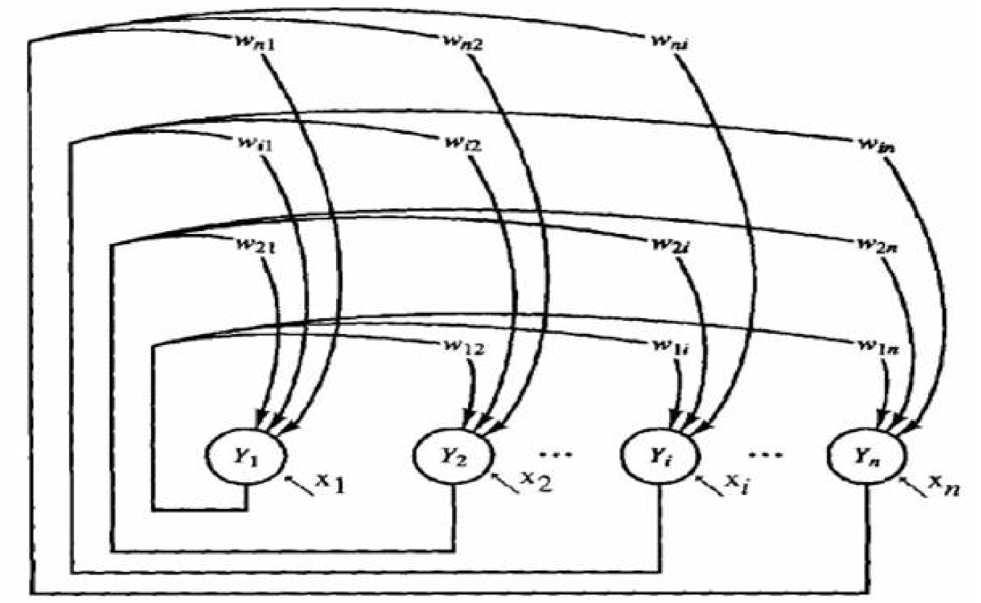
```
p= 1 to P                /* for every training pair */
    i = 1 to n           /* for every row in W     */
        j = 1 to m       /* for every element j in row i */
```

Review:Hopfield network

- **Architecture:**
  - single layer (units serve as both input and output)
  - nodes are threshold units (binary or bipolar)
  - weights: fully connected, symmetric, and zero diagonal

$$w_{ij} = w_{ji}$$
$$w_{ii} = 0$$

  - x_i are external inputs, which may be transient or permanent



Review:Hopfield network

- Storage is performed according to the following equation:

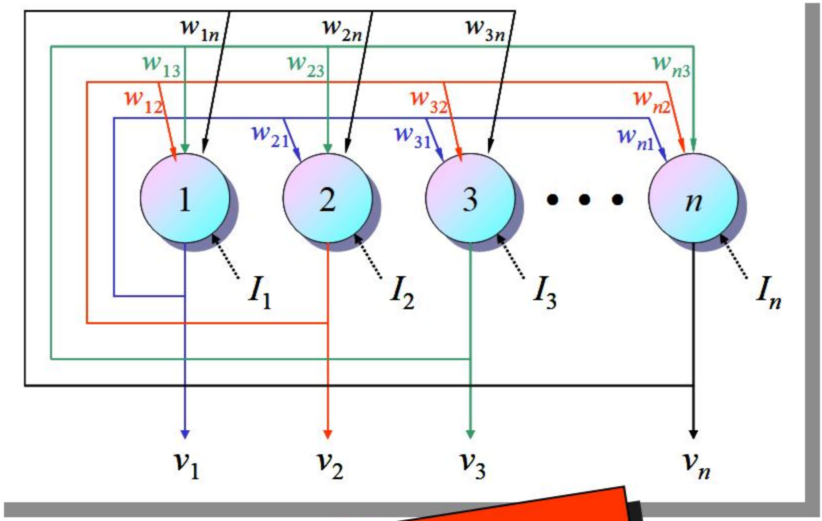
$$w_{ij} = \frac{1}{P} \sum_{p=1}^{P} x_i^p x_j^p$$

- The weight matrix is symmetrical, i.e., $w_{ij} = w_{ji}$.

- The constraint condition $w_{ii} = 0$ is important for the network behavior. It can be mathematically proven that *under these conditions the network will reach a stable activation state within an infinite number of iterations.*

Review:Hopfield network

- Recall
  - Use an input vector to recall a stored vector
  - Each time, randomly select a unit for update
  - Periodically check for convergence (stable state)
- Asynchronous mode update rule



Stable?

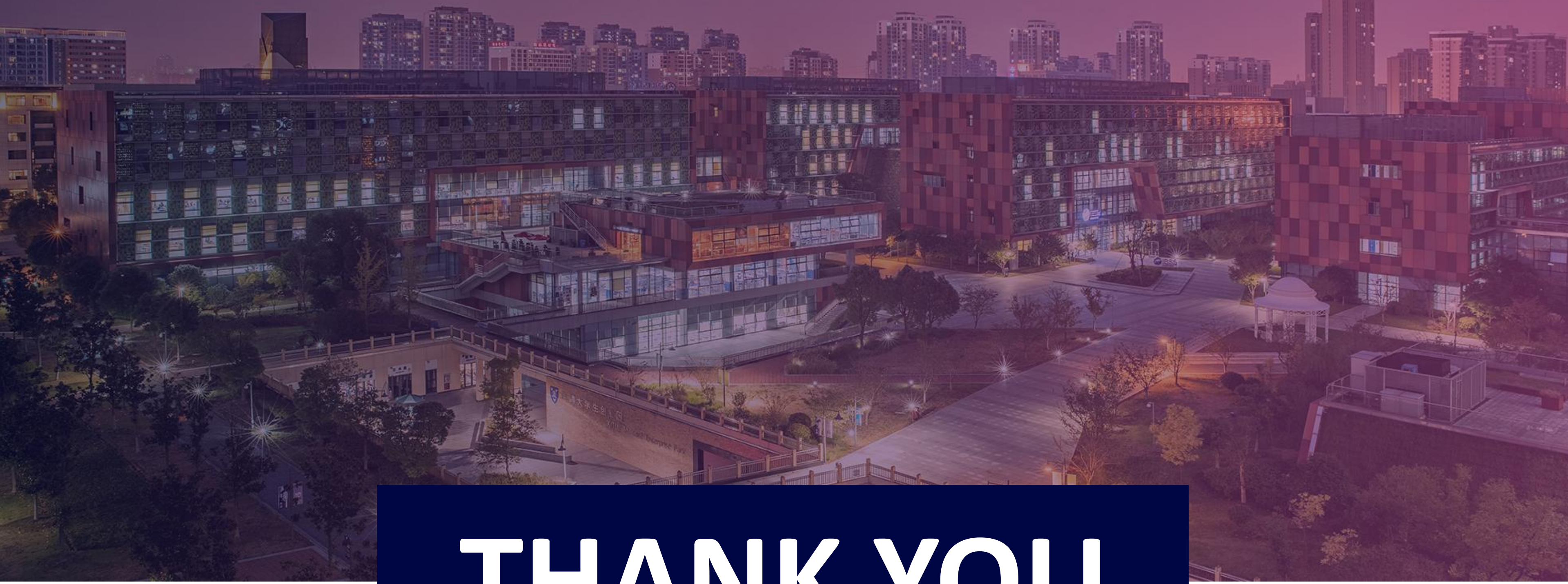
$$H_i(t+1) = \sum_{\substack{j=1 \\ j\neq i}}^{n} w_{ij} v_j(t) + I_i$$

$$v_i(t+1) = \operatorname{sgn}\left[H_i(t+1)\right] = \begin{cases} 1 & H_i(t+1) \geq 0 \\ -1 & H_i(t+1) < 0 \end{cases}$$

Review:Hopfield network

- The number of patterns that can be stored and accurately recalled is severely limited
  - net may converge to a novel spurious pattern

- Exemplar pattern will be unstable if it shares many bits in common with another exemplar pattern



THANK YOU

VISIT US

WWW.XJTLU.EDU.CN

FOLLOW US

@XJTLU

Xi'an Jiaotong-Liverpool University
西交利物浦大學