Adaptive Resonance Theory

ART Networks

Stephen Grossberg &
Gail Carpenter

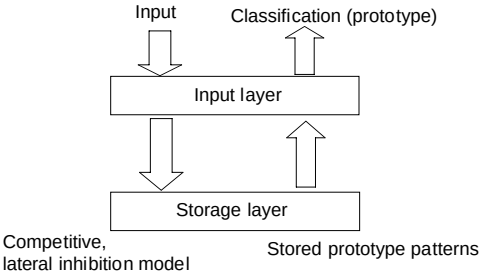
ART Networks

- Work by clustering + nuances
- Several varieties:
  - ART1: Discrete patterns
  - ART2: Continuous patterns …
  - Fuzzy ARTMAP
- Attempt to address the "stability/plasticity dilemma":
  - stability: recognized patterns should be insensitive to noise
  - plasticity: system should be capable of learning new patterns

ART Networks

- Combine supervised and unsupervised (competitive, clustering)
- Dynamically create new categories
- Biologically motivated by an ODE model
- Models short- and long-term memory

ART Layers



ART1

- Two kinds of explanations:
  - algorithmic
  - neural
- We concentrate on the first.
- The second is more complicated, since it involves ways to achieve the control aspects of the algorithmic approach.

Basic ART Operation

- Input pattern presented to input layer
- Storage layer indicates tentative hypothetical classification
- Inut layer decides if hypothetical is **close enough**; if so, done.
- If not, storage layer indicates alternate hypothesis.
- The above two steps are repeated until the hypothetical classification is accepted.

ART Operation

- All hypotheses could be rejected; in this case, a **new** class is created in the storage layer.
- "resonance" = mutual reinforcement between input and storage layers
- "adaptive" = weights are adjusted when resonance occurs

ART1 (Discrete Patterns)

- training pattern $x \in \{0, 1\}^n$
- prototype patterns $w_j \in \{0, 1\}^n$
- Storage unit computes $y_j = w_j x / || w_j ||^2$ for each prototype
- The winner is the prototype with the largest y_j
- For acceptance, $y_j > || x ||^2 / n$, where n is the number of dimensions.
- This means that sufficiently-many bits must **match**.

ART1

- Assuming that the acceptance test is passed, it is also required that

  $$w_j x / || x ||^2 > \rho$$

  where ρ is an adjustable parameter called **vigilance**.

  This means that the input and the pattern share a sufficient fraction of 1's.

ART1

- If the acceptance test is passed, but the vigilance test is not:
  - the prototype in question is temporarily omitted from consideration;
  - a new competition takes place
- until a prototype is found for which both tests are passed.
- If no prototype is found, a new class k is created with

  $$w_k = x$$

ART1

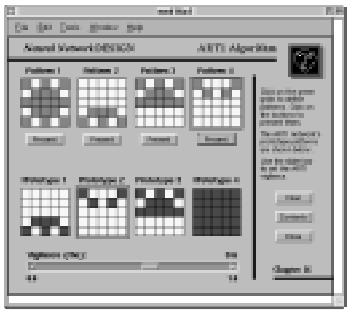
- The **higher** the **vigilance**, the more likely a new pattern is to be introduced.
- Lower vigilance will allow one input to pass as another pattern.

ART1 Issues

- Subset-Superset dilemma:
  - If one pattern is **contained in** another, then a given input may have the same inner product with two different prototypes.
  - Can be resolved by allowing weights other than {0, 1} and **normalizing** the prototypes.
  - Neural normalization can be achieved by an on-center, off-surround competition.

ART1 Demo

Increasing **vigilance** causes the network to be more selective, to introduce a new prototype when the fit is not good.



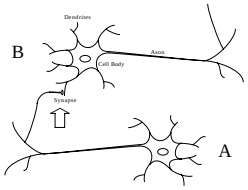
Try different patterns

Hebbian Review

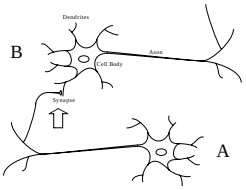
Hebb's Postulate

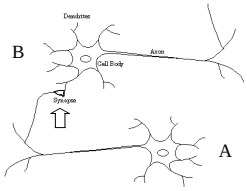
"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

D. O. Hebb, 1949



In other words, when a weight contributes to firing a neuron, the weight is increased. (If the neuron doesn't fire, then the weight is not).





Colloquial Corollaries

- "Neurons that fire together wire together."
- "Use it or lose it".
- "Taking notes helps you remember."
- "Absence makes the heart grow fonder".

Generalized Hebb Rule

- When a weight contributes to firing a neuron, the weight is increased.
- When firing one neuron acts to *inhibit* the firing of another, the weight is *decreased*.
- When neurons don't fire together, the weight is *decreased*.
- "Those that don't won't."

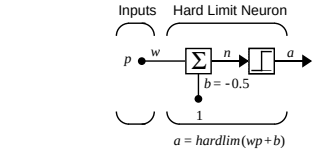
Flavors of Hebbian Learning

- **Unsupervised**
  - Weights are strengthened by their similarity to the ***actual*** response to a stimulus.
- **Supervised**
  - Weights are strengthened by similarity to the ***desired*** response.

Unsupervised Hebbian Learning

(aka "Associative Learning")

Simple Associative Network

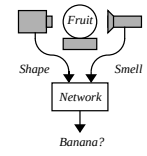


$$a = hardlim(wp + b) = hardlim(wp - 0.5)$$

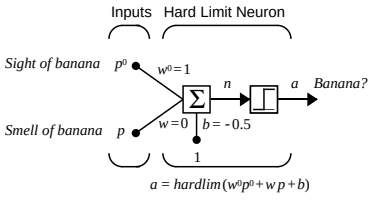
input: $p = \begin{cases} 1, & \text{stimulus} \\ 0, & \text{no stimulus} \end{cases}$

output: $a = \begin{cases} 1, & \text{response} \\ 0, & \text{no response} \end{cases}$

Banana Associator



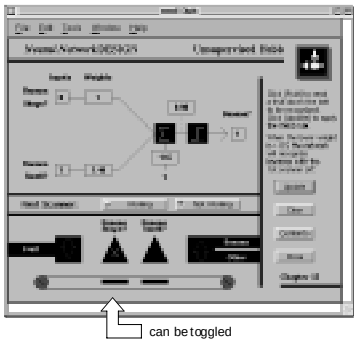
Did Pavlov anticipate this?



Unconditioned Stimulus: $p^0 = \begin{cases} 1, & \text{shape detected} \\ 0, & \text{shape not detected} \end{cases}$

Conditioned Stimulus: $p = \begin{cases} 1, & \text{smell detected} \\ 0, & \text{smell not detected} \end{cases}$

Banana Associator Demo



Unsupervised Hebb Rule

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q)$$

input

actual response

Vector Form:

$$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha \mathbf{a}(q)\mathbf{p}^T(q)$$

Training Sequence:

$$\mathbf{p}(1), \mathbf{p}(2), \ldots, \mathbf{p}(Q)$$

Learning Banana Smell

Initial Weights:

$$w^0 = 1, w(0) = 0$$

unconditioned (shape)

Training Sequence:

$$\{p^0(1) = 0, p(1) = 1\}, \{p^0(2) = 1, p(2) = 1\}, \ldots$$

conditioned (smell)

$\alpha = 1$

$$w(q) = w(q-1) + a(q)p(q)$$

First Iteration (sight fails, smell present):

$$a(1) = hardlim(w^0 p^0(1) + w(0)p(1) - 0.5)$$
$$= hardlim(1 \cdot 0 + 0 \cdot 1 - 0.5) = 0 \quad \text{(no banana)}$$

$$w(1) = w(0) + a(1)p(1) = 0 + 0 \cdot 1 = 0$$

Example

Second Iteration (sight works, smell present):

$$a(2) = hardlim(w^0 p^0(2) + w(1)p(2) - 0.5)$$
$$= hardlim(1 \cdot 1 + 0 \cdot 1 - 0.5) = 1 \quad \text{(banana)}$$

$$w(2) = w(1) + a(2)p(2) = 0 + 1 \cdot 1 = 1$$

Third Iteration (sight fails, smell present):

$$a(3) = hardlim(w^0 p^0(3) + w(2)p(3) - 0.5)$$
$$= hardlim(1 \cdot 0 + 1 \cdot 1 - 0.5) = 1 \quad \text{(banana)}$$

$$w(3) = w(2) + a(3)p(3) = 1 + 1 \cdot 1 = 2$$

Banana will now be detected if either sensor works.

Problems with Hebb Rule

- Weights can become arbitrarily large.
- There is no mechanism for weights to decrease.

Hebb Rule with Decay

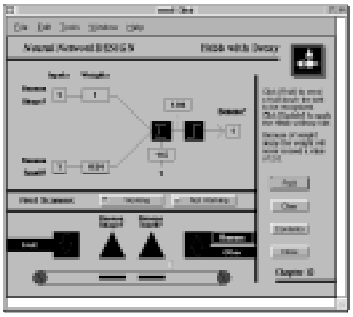
$$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha \mathbf{a}(q)\mathbf{p}^T(q) - \gamma \mathbf{W}(q-1)$$

$$\mathbf{W}(q) = (1-\gamma)\mathbf{W}(q-1) + \alpha \mathbf{a}(q)\mathbf{p}^T(q)$$

This keeps the weight matrix from growing without bound, which can be demonstrated by setting both a_i and p_j to 1:

$$w_{ij}^{max} = (1-\gamma)w_{ij}^{max} + \alpha a_i p_j$$
$$w_{ij}^{max} = (1-\gamma)w_{ij}^{max} + \alpha$$
$$w_{ij}^{max} = \frac{\alpha}{\gamma}$$

Banana Associator with Decay



Example: Banana Associator

$\alpha = 1 \qquad \gamma = 0.1$

First Iteration (sight fails, smell present):

$$a(1) = hardlim(w^0 p^0(1) + w(0)p(1) - 0.5)$$
$$= hardlim(1 \cdot 0 + 0 \cdot 1 - 0.5) = 0 \quad \text{(no banana)}$$

$$w(1) = w(0) + a(1)p(1) - 0.1w(0) = 0 + 0 \cdot 1 - 0.1(0) = 0$$

Second Iteration (sight works, smell present):

$$a(2) = hardlim(w^0 p^0(2) + w(1)p(2) - 0.5)$$
$$= hardlim(1 \cdot 1 + 0 \cdot 1 - 0.5) = 1 \quad \text{(banana)}$$

$$w(2) = w(1) + a(2)p(2) - 0.1w(1) = 0 + 1 \cdot 1 - 0.1(0) = 1$$

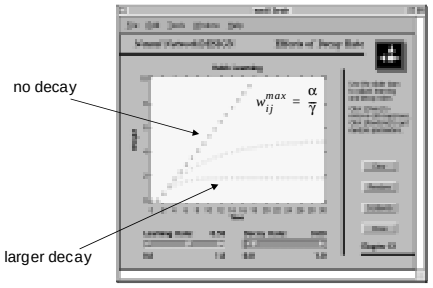
Example

Third Iteration (sight fails, smell present):

$$a(3) = hardlim(w^0 p^0(3) + w(2)p(3) - 0.5)$$
$$= hardlim(1 \cdot 0 + 1 \cdot 1 - 0.5) = 1 \quad \text{(banana)}$$

$$w(3) = w(2) + a(3)p(3) - 0.1w(3) = 1 + 1 \cdot 1 - 0.1(1) = 1.9$$

General Decay Demo



Problem of Hebb with Decay

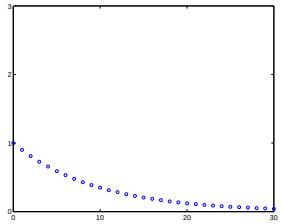
Associations will be lost if stimuli are not occasionally presented.

If $a_i = 0$, then

$$w_{ij}(q) = (1 - \gamma)w_{ij}(q - 1)$$

If $\gamma = 0.1$, this becomes

$$w_{ij}(q) = (0.9)w_{ij}(q - 1)$$

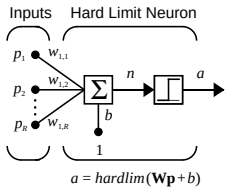


Therefore the weight decays by 10% at each iteration where there is no stimulus.

Solution to Hebb Decay Problem

- Don't decay weights when there is no stimulus of any kind.
- We have seen rules like this (Instar)

Instar *Recognition* Network



$a = hardlim(\mathbf{W}\mathbf{p} + b)$

Instar Operation

$$a = hardlim(\mathbf{W}\mathbf{p} + b) = hardlim({}_1\mathbf{w}^T\mathbf{p} + b)$$

The instar will be active when

$${}_1\mathbf{w}^T\mathbf{p} \geq -b$$

or

$${}_1\mathbf{w}^T\mathbf{p} = \|{}_1\mathbf{w}\|\|\mathbf{p}\|\cos\theta \geq -b$$

For normalized vectors, the largest inner product occurs when the angle between the weight vector and the input vector is zero -- the input vector is equal to the weight vector.

The rows of a weight matrix represent patterns to be recognized.

Vector Recognition

If we set

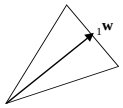
$$b = -\|{}_1\mathbf{w}\|\|\mathbf{p}\|$$

the instar will only be active when $\theta = 0$.

If we set

$$b > -\|{}_1\mathbf{w}\|\|\mathbf{p}\|$$

the instar will be active for a range of angles.



As b is increased, the more patterns there will be (over a wider range of θ) which will activate the instar.

Instar Rule

Hebb with Decay

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q)$$

Modify so that **learning and forgetting will only occur when the neuron is active** - Instar Rule:

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q) - \gamma a_i(q) w_{ij}(q-1)$$

or

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q)(p_j(q) - w_{ij}(q-1))$$

Vector Form:

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha a_i(q)(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$$

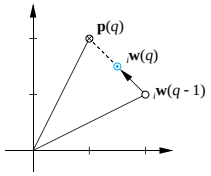
Graphical Representation

For the case where the instar is active ($a_i = 1$):

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$$

or

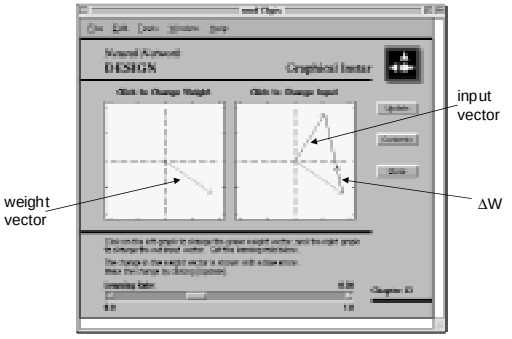
$${}_i\mathbf{w}(q) = (1-\alpha){}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q)$$



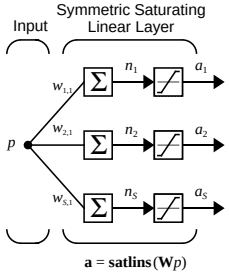
For the case where the instar is inactive ($a_i = 0$):

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1)$$

Instar Demo



Outstar (*Recall* Network)



$$\mathbf{a} = \mathbf{satlins}(\mathbf{W}p)$$

Outstar Operation

Suppose we want the outstar to recall a certain pattern $\mathbf{a}^*$ whenever the input $p = 1$ is presented to the network. Let

$$\mathbf{W} = \mathbf{a}^*$$

Then, when $p = 1$

$$\mathbf{a} = \mathbf{satlins}(\mathbf{W}p) = \mathbf{satlins}(\mathbf{a}^* \cdot 1) = \mathbf{a}^*$$

and the pattern is correctly recalled.

The columns of a weight matrix represent patterns to be recalled.

Outstar Rule

For the instar rule we made the weight decay term of the Hebb rule proportional to the *output* of the network.

For the outstar rule we make the weight decay term proportional to the *input* of the network.

$$w_{ij}(q) = w_{ij}(q-1) + \alpha a_i(q) p_j(q) - \gamma p_j(q) w_{ij}(q-1)$$

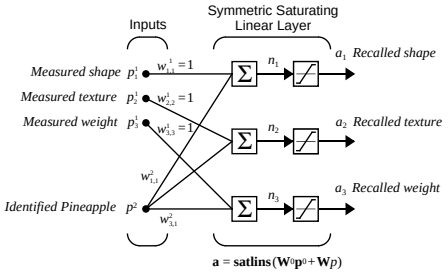
If we make the decay rate γ equal to the learning rate α,

$$w_{ij}(q) = w_{ij}(q-1) + \alpha(a_i(q) - w_{ij}(q-1)) p_j(q)$$

Vector Form:

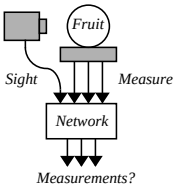
$$\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha(\mathbf{a}(q) - \mathbf{w}_j(q-1)) p_j(q)$$

Example - Pineapple Recall



Definitions

$$\mathbf{a} = \mathbf{satlins}(\mathbf{W}^0\mathbf{p}^0 + \mathbf{W}p)$$

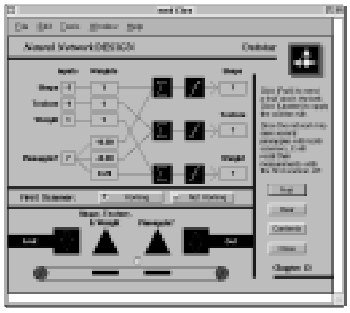


$$\mathbf{W}^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{p}^0 = \begin{bmatrix} shape \\ texture \\ weight \end{bmatrix} \qquad \mathbf{p}^{pineapple} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$p = \begin{cases} 1, & \text{if a pineapple can be seen} \\ 0, & \text{otherwise} \end{cases}$$

Outstar Demo



Iteration 1

$$\left\{ \mathbf{p}^0(1) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, p(1) = 1 \right\}, \left\{ \mathbf{p}^0(2) = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, p(2) = 1 \right\}, \ldots$$

$$\alpha = 1$$

$$\mathbf{a}(1) = \mathbf{satlins}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} 1 \right) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{(no response)}$$

$$\mathbf{w}_1(1) = \mathbf{w}_1(0) + (\mathbf{a}(1) - \mathbf{w}_1(0)) p(1) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right) 1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Convergence

$$\mathbf{a}(2) = \mathbf{satlins}\left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} + \begin{bmatrix}0\\0\\0\end{bmatrix}1\right) = \begin{bmatrix}-1\\-1\\1\end{bmatrix} \quad \text{(measurements given)}$$

$$\mathbf{w}_1(2) = \mathbf{w}_1(1) + (\mathbf{a}(2) - \mathbf{w}_1(1))p(2) = \begin{bmatrix}0\\0\\0\end{bmatrix} + \left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} - \begin{bmatrix}0\\0\\0\end{bmatrix}\right)1 = \begin{bmatrix}-1\\-1\\1\end{bmatrix}$$

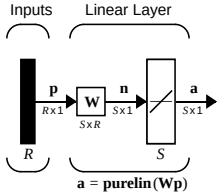
$$\mathbf{a}(3) = \mathbf{satlins}\left(\begin{bmatrix}0\\0\\0\end{bmatrix} + \begin{bmatrix}-1\\-1\\1\end{bmatrix}1\right) = \begin{bmatrix}-1\\-1\\1\end{bmatrix} \quad \text{(measurements recalled)}$$

$$\mathbf{w}_1(3) = \mathbf{w}_1(2) + (\mathbf{a}(2) - \mathbf{w}_1(2))p(2) = \begin{bmatrix}-1\\-1\\1\end{bmatrix} + \left(\begin{bmatrix}-1\\-1\\1\end{bmatrix} - \begin{bmatrix}-1\\-1\\1\end{bmatrix}\right)1 = \begin{bmatrix}-1\\-1\\1\end{bmatrix}$$

Supervised Hebbian Learning

Linear Associative Memory

Linear Associator



$$\mathbf{a} = \mathbf{Wp} \qquad a_i = \sum_{j=1}^{R} w_{ij}p_j$$

Training Set:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Hebb Rule

$$w_{ij}^{new} = w_{ij}^{old} + \alpha\, f_i(a_{iq})g_j(p_{jq})$$

Postsynaptic Signal

Presynaptic Signal

Simplified Form:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha a_{iq}p_{jq}$$

actual output

input pattern

Supervised Form:

$$w_{ij}^{new} = w_{ij}^{old} + t_{iq}p_{jq}$$

Matrix Form:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{t}_q\mathbf{p}_q^T$$

desired output

Batch Operation

$$\mathbf{W} = \mathbf{t}_1\mathbf{p}_1^T + \mathbf{t}_2\mathbf{p}_2^T + \cdots + \mathbf{t}_Q\mathbf{p}_Q^T = \sum_{q=1}^{Q}\mathbf{t}_q\mathbf{p}_q^T$$

(Zero Initial Weights)

Matrix Form:

$$\mathbf{W} = \begin{bmatrix}\mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q\end{bmatrix}\begin{bmatrix}\mathbf{p}_1^T\\\mathbf{p}_2^T\\\vdots\\\mathbf{p}_Q^T\end{bmatrix} = \mathbf{TP}^T$$

$$\mathbf{P} = \begin{bmatrix}\mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_Q\end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix}\mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q\end{bmatrix}$$

Performance Analysis

$$\mathbf{a} = \mathbf{Wp}_k = \left(\sum_{q=1}^{Q}\mathbf{t}_q\mathbf{p}_q^T\right)\mathbf{p}_k = \sum_{q=1}^{Q}\mathbf{t}_q(\mathbf{p}_q^T\mathbf{p}_k)$$

Case I, input patterns are orthogonal.

$$(\mathbf{p}_q^T\mathbf{p}_k) = 1 \qquad q = k$$
$$= 0 \qquad q \neq k$$

Therefore the network output equals the target:

$$\mathbf{a} = \mathbf{Wp}_k = \mathbf{t}_k$$

Case II, input patterns are normalized, but not orthogonal.

$$\mathbf{a} = \mathbf{Wp}_k = \mathbf{t}_k + \sum_{q \neq k}\mathbf{t}_q(\mathbf{p}_q^T\mathbf{p}_k)$$

Error term

Example

Banana Apple Normalized Prototype Patterns

$$\mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \quad \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \quad \left\{ \mathbf{p}_1 = \begin{bmatrix} -0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix} \right\} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix} \right\}$$

Weight Matrix (Hebb Rule):

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} -0.5774 & 0.5774 & -0.5774 \\ 0.5774 & 0.5774 & -0.5774 \end{bmatrix} = \begin{bmatrix} 1.1548 & 0 & 0 \end{bmatrix}$$

Tests:

Banana $\mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 1.1548 & 0 & 0 \end{bmatrix} \begin{bmatrix} -0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} -0.6668 \end{bmatrix}$

Apple $\mathbf{W}\mathbf{p}_2 = \begin{bmatrix} 0 & 1.1548 & 0 \end{bmatrix} \begin{bmatrix} 0.5774 \\ 0.5774 \\ -0.5774 \end{bmatrix} = \begin{bmatrix} 0.6668 \end{bmatrix}$

Pseudoinverse Rule - (1)

Performance Index: $\mathbf{W}\mathbf{p}_q = \mathbf{t}_q \qquad q = 1, 2, \ldots, Q$

$$F(\mathbf{W}) = \sum_{q=1}^{Q} \|\mathbf{t}_q - \mathbf{W}\mathbf{p}_q\|^2$$ ⟸ **Mean-squared error**

Matrix Form: $\mathbf{W}\mathbf{P} = \mathbf{T}$

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_Q \end{bmatrix}$$

$$F(\mathbf{W}) = \|\mathbf{T} - \mathbf{W}\mathbf{P}\|^2 = \|\mathbf{E}\|^2$$

$$\|\mathbf{E}\|^2 = \sum_i \sum_j e_{ij}^2$$

Pseudoinverse Rule - (2)

$$\mathbf{W}\mathbf{P} = \mathbf{T}$$

Minimize:

$$F(\mathbf{W}) = \|\mathbf{T} - \mathbf{W}\mathbf{P}\|^2 = \|\mathbf{E}\|^2$$

If an inverse exists for $\mathbf{P}$, $F(\mathbf{W})$ can be made zero:

$$\mathbf{W} = \mathbf{T}\mathbf{P}^{-1}$$

When an inverse does not exist $F(\mathbf{W})$ can be minimized using the pseudoinverse:

$$\mathbf{W} = \mathbf{T}\mathbf{P}^+$$

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T$$

Relationship to the Hebb Rule

Hebb Rule

$$\mathbf{W} = \mathbf{T}\mathbf{P}^T$$

Pseudoinverse Rule

$$\mathbf{W} = \mathbf{T}\mathbf{P}^+$$

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T$$

If the prototype patterns are orthonormal:

$$\mathbf{P}^T\mathbf{P} = \mathbf{I}$$

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T = \mathbf{P}^T$$

Example

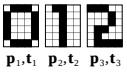
$$\left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix} -1 \end{bmatrix} \right\} \quad \left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix} 1 \end{bmatrix} \right\} \quad \mathbf{W} = \mathbf{T}\mathbf{P}^+ = \begin{bmatrix} -1 & 1 \end{bmatrix} \left(\begin{bmatrix} -1 & 1 \\ 1 & 1 \\ -1 & -1 \end{bmatrix} \right)^+$$

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}^{-1} \begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -0.5 & 0.25 & -0.25 \\ 0.5 & 0.25 & -0.25 \end{bmatrix}$$

$$\mathbf{W} = \mathbf{T}\mathbf{P}^+ = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} -0.5 & 0.25 & -0.25 \\ 0.5 & 0.25 & -0.25 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

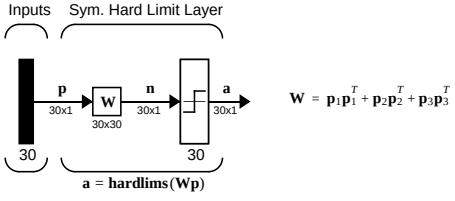
$$\mathbf{W}\mathbf{p}_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \end{bmatrix} \qquad \mathbf{W}\mathbf{p}_2 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix}$$

Autoassociative Memory



$\mathbf{p}_1, \mathbf{t}_1 \quad \mathbf{p}_2, \mathbf{t}_2 \quad \mathbf{p}_3, \mathbf{t}_3$

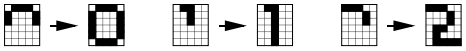
$$\mathbf{p}_1 = \begin{bmatrix} -1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & \ldots & 1 & -1 \end{bmatrix}^T$$



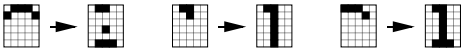
$$\mathbf{W} = \mathbf{p}_1\mathbf{p}_1^T + \mathbf{p}_2\mathbf{p}_2^T + \mathbf{p}_3\mathbf{p}_3^T$$

Tests

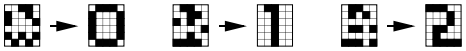
50% Occluded



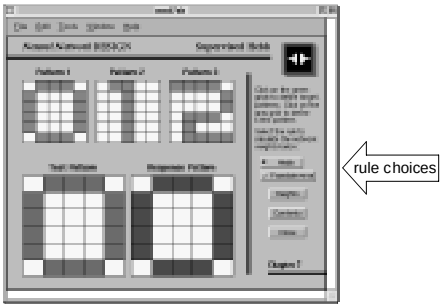
67% Occluded



Noisy Patterns (7 pixels)



Supervised Hebbian Demo



Spectrum of Hebbian Learning

Unsupervised: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{a}_q \mathbf{p}_q^T$

Supervised: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T$

Smoothing: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T - \gamma \mathbf{W}^{old} = (1-\gamma)\mathbf{W}^{old} + \alpha \mathbf{t}_q \mathbf{p}_q^T$

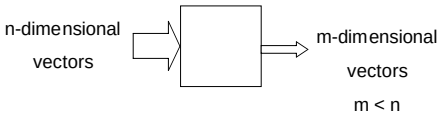
Delta Rule: $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha(\mathbf{t}_q - \mathbf{a}_q)\mathbf{p}_q^T$ (target, actual)

Principal Component Analysis (PCA) Neural Networks

What is PCA?

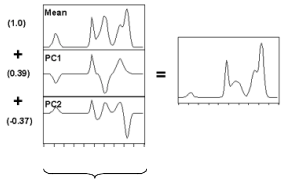
- A standard statistical technique for reducing dimensionality of data
- Purpose: Better understanding or communication of data; used a lot in the sciences to select the most important features
- In so reducing, we want to lose as little information as possible, given the before- and after- dimensions.
- Also known as Karhuenen-Loeve (K-L) transformation (Watanabe, 1969)

PCA



Principal Components

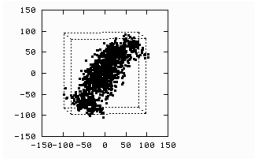
(cf http://www.galactic.com/algorithms/pca.htm)



vector components

Scientific Uses

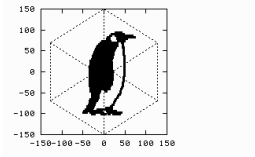
(cf. http://144.124.112.51/auj/scattering/demo/page4.active.html)



Transform coordinates to get a better understanding of the underlying phenomenon.

Scientific Uses

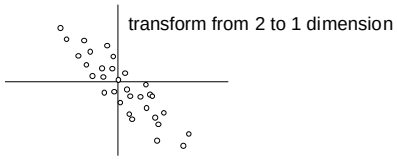
(cf. http://144.124.112.51/auj/scattering/demo/page4.active.html)



Transform coordinates to get a better understanding of the underlying phenomenon.

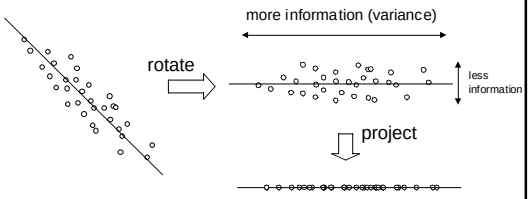
The main idea

- Transform the input data into fewer dimensions
- Preserve as much of the variance as possible



Transformation

- Preserve as much of the variance as possible



Comparison with Linear Regression

- Linear regression requires one to ***pre-identify*** dependent vs. independent variables.
- PCA does not.

How to Accomplish PCA?

- First center data by subtracting the mean in each dimension.
- Multiply each data point (column vector) by an m x n matrix W, having orthonormal rows, to be determined.
- (Orthonormal rows means $W W^T = I$.)

Determining Matrix W

- We want to choose W so that the amount of **"information" lost** in multiplying by W is minimized.
- Since W is orthonormal, W^T is the pseudo-inverse of W, i.e.
- if $Wx = y$, then $W^T y$ is an approximation to x, i.e.
- $W^T Wx$ should be "close" to x.
- We want $|| x - W^T Wx ||$ to be minimized.

Determining Matrix (2)

- From previous page, choose W so that $E(|| x - W^T Wx ||)$ where E denotes the **expectation** (mean) taken over all samples x.
- In other words, minimize the sum
  $$\sum_{\text{training samples}} || x_i - W^T W x_i ||^2$$
- PCA is a method for choosing W (given target # of dimensions) to minimize this sum.

Determining Matrix (3)

- Let T be the set of training vectors.
- The **covariance matrix** S is defined by computing, for each pair of components i, j, the value
  $E[(x_i - m_i)(x_j - m_j)]$
  taken over x in T where is the mean for that component.
- A **high i, j entry** in this matrix means that variables i and j are **positively correlated**.
- Among vectors w, the **variance** of wx over the set T is **maximized** when
  - w is an eigenvector, corresponding to the maximum eigenvalue of S.

Determining Matrix (4)

- So, to compute W, compute the covariance matrix S.
- Compute the eigenvalues of S.
- For a target dimension of m, choose the m largest eigenvalues, and form W out of the corresponding eigenvectors as rows.

Example

- Training set T:
  - (1.3, 3.2, 3.7)
  - (1.4, 2.8, 4.1)
  - (1.5, 3.1, 4.6)
  - (1.2, 2.9, 4.8)
  - (1.1, 3.0, 4.8)
- Component means:
  - (1.3, 3.0, 4.4)
- Mean-Adjusted Training set T:
  - (0, 0.2, -0.7)
  - (0.1, -0.2, -0.3)
  - (0.2, 0.1, 0.2)
  - (-0.1, -0.1, 0.4)
  - (-0.2, 0.0, 0.4)

Example (2)

- Covariance Matrix (T has 3 dimensions)
- S = $0.2 \begin{pmatrix} 0.1 & 0.01 & -0.11 \\ 0.01 & 0.10 & -0.1 \\ -0.11 & -0.1 & 0.94 \end{pmatrix}$
- Eigenvalues are $\gamma_1 = 0.965$, $\gamma_2 = 0.090$, $\gamma_3 = 0.084$

Example (3)

- $\gamma_1 / (\gamma_1 + \gamma_2 + \gamma_3) = 0.84$, meaning that one component captures 84% of the variance
- Eigenvectors corresponding to the two largest eigenvalues are:
  - (-0.823, -0.54, -0.169)
  - (0.553, -0.832, -0.026)
- These two vectors would be the rows of the matrix mapping the training set into two dimensions.

Matlab: pcacov function

```
>> cv = 0.2*[0.1,    0.01, -0.11;
             0.01,   0.10, -0.1;
            -0.11,  -0.1,   0.94];

>> [pc, latent, explained] = pcacov(cv);

>> explained

explained =

   84.7212
    7.9114
    7.3674
```

Matlab: pcacov function

```
>> latent

latent =

    0.1932
    0.0180
    0.0168
```

Largest two eigenvalues

```
>> pc
```

Corresponding eigenvectors

```
pc =

    0.1265    0.5534    0.8232
    0.1153   -0.8325    0.5419
   -0.9852   -0.0263    0.1691
```

Perspective

- According to B. D. Ripley, 1996 (Pattern Recognition and Neural Networks, Oxford U. Press), the method just described is inferior to another one using *singular value decomposition*.

An Application

(D. Morrison, via Diamantaras & Kung)

- Characterize turtle shells based upon measured length(L), width(W), and height(H) (in mm).
- Covariance matrix for 24 female turtles:

	L	W	H
L	451.33	271.17	168.7
W	271.17	171.73	103.29
H	168.7	103.29	66.65

- Principal Components:

	1	2	3
L	0.8126	-0.5454	-0.2054
W	0.4955	0.8321	-0.2491
H	0.3068	0.1006	0.9465
variance	680.4	6.5	2.9

Turtle Application (D. Morrison)

- Principal Components:

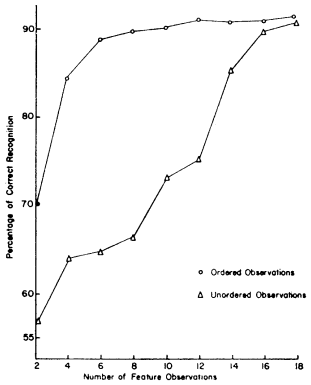
	1	2	3
L	0.8126	-0.5454	-0.2054
W	0.4955	0.8321	-0.2491
H	0.3068	0.1006	0.9465
variance	680.4	6.5	2.9

- The first component accounts for 98.6% of the variance. So the following measure can be used:

$Y_1 = 0.81L + 0.5W + 0.31H$

Application: Handwritten Character recognition (K.S. Fu)

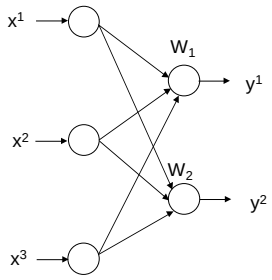
- Attempt recognition based upon **18 radial measurements**, spaced at 20 degree increments, of characters (240 samples).
- Recognition rate was computed vs. the number of measurements used (2, 4, 6, … out of 18) in no particular order.
- The same computation was done with the measurements **ordered by principal components**.
- The next slide compares the two approaches.



What is a PCA Network?

- The matrix W of PCA can be interpreted as a 1-layer neural network.
- Each row of W corresponds to weights of a neuron.

3->2 PCA Network



PCA Network Learning

- PCA network can *learn* weights rather than have them precomputed.
- This is a form of *unsupervised* Hebbian learning.
- However, the choice of output variables has to be pre-made from the data.

PCA Learning Rules

- $\Delta W = \eta\, y_i\, x_i^T - K_i\, W$, where
  - x_i is an input vector
  - y_i is the corresponding output vector
  - $y_i x_i^T$ is the Hebbian component
  - K_i is one of the following:
    - 0 pure Hebbian
    - $y_i y_i^T$ (Williams' rule, 1985)
    - $3D(y_i y_i^T) + 2L(y_i y_i^T)$ (Oja and Karhunen rule, 1985)
    - $L(y_i y_i^T)$ (Sanger's rule, 1989)
- where D(M) maps the **diagonal** entries of M to themselves, and other entries to 0.
- L(M) maps entries below the main diagonal to themselves, and other entries to 0.

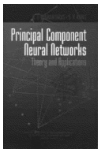
PCA Shortcomings

- The PCA user must say how many target dimensions to use.
- PCA only finds linear sub-spaces.
- It works best if the individual components are Gaussian-distributed.
- ICA (e.g. Bell & Senjowski) does not rely on such a distribution.

Other means of learning for dimensionality reduction

- Self-Organizing Map
  (# of dimensions in superstructure = reduced dimension)
- LVQ
  (# of neurons = reduced dimension)
- Projection pursuit, another statistical technique

Book on PCA NN



Diamantaras, K.I. / Kung, S.Y.

Principal Component Neural Networks
Theory and Applications

1996. 256 pages.
ISBN 0-471-05436-4 John Wiley & Sons

The coverage in this book is extensive. It gives several additional learning algorithms, including ones with lateral connections as well as feed-forward ones.

PCA Demo (M. Hassoun):

http://neuron.eng.wayne.edu/java/PCA/PCA.html