CONNECTIONIST METHODS

Non Symbolic Representations

- Decision trees can be easily read
  - A disjunction of conjunctions (logic)
  - We call this a symbolic representation
- Non-symbolic representations
  - More numerical in nature, more difficult to read
- Artificial Neural Networks (ANNs)
  - A Non-symbolic representation scheme
  - They embed a giant mathematical function
    - ◆To take inputs and compute an output which is interpreted as a categorisation
  - Often shortened to “Neural Networks”
    - ◆Don’t confuse them with real neural networks (in heads)

Function Learning

- Map categorisation learning to numerical problem
  - Each category given a number
  - Or a range of real valued numbers (e.g., 0.5 - 0.9)
- Function learning examples
  - Input = 1,2,3,4   Output = 1,4,9,16
    - Here the concept to learn is squaring integers
  - Input = [1,2,3], [2,3,4], [3,4,5], [4,5,6], Output = 1, 5, 11, 19
    - Here the concept is: [a,b,c] -> a*c - b
    - The calculation is more complicated than in the first example
- Neural networks:
  - Calculation is much more complicated in general
  - But it is still just a numerical calculation

Complicated Example:Categorising Vehicles

- Input to function: pixel data from vehicle images
  - Output: numbers: 1 for a car; 2 for a bus; 3 for a tank

INPUT INPUT INPUT INPUT

OUTPUT = 3 OUTPUT = 2 OUTPUT = 1 OUTPUT=1

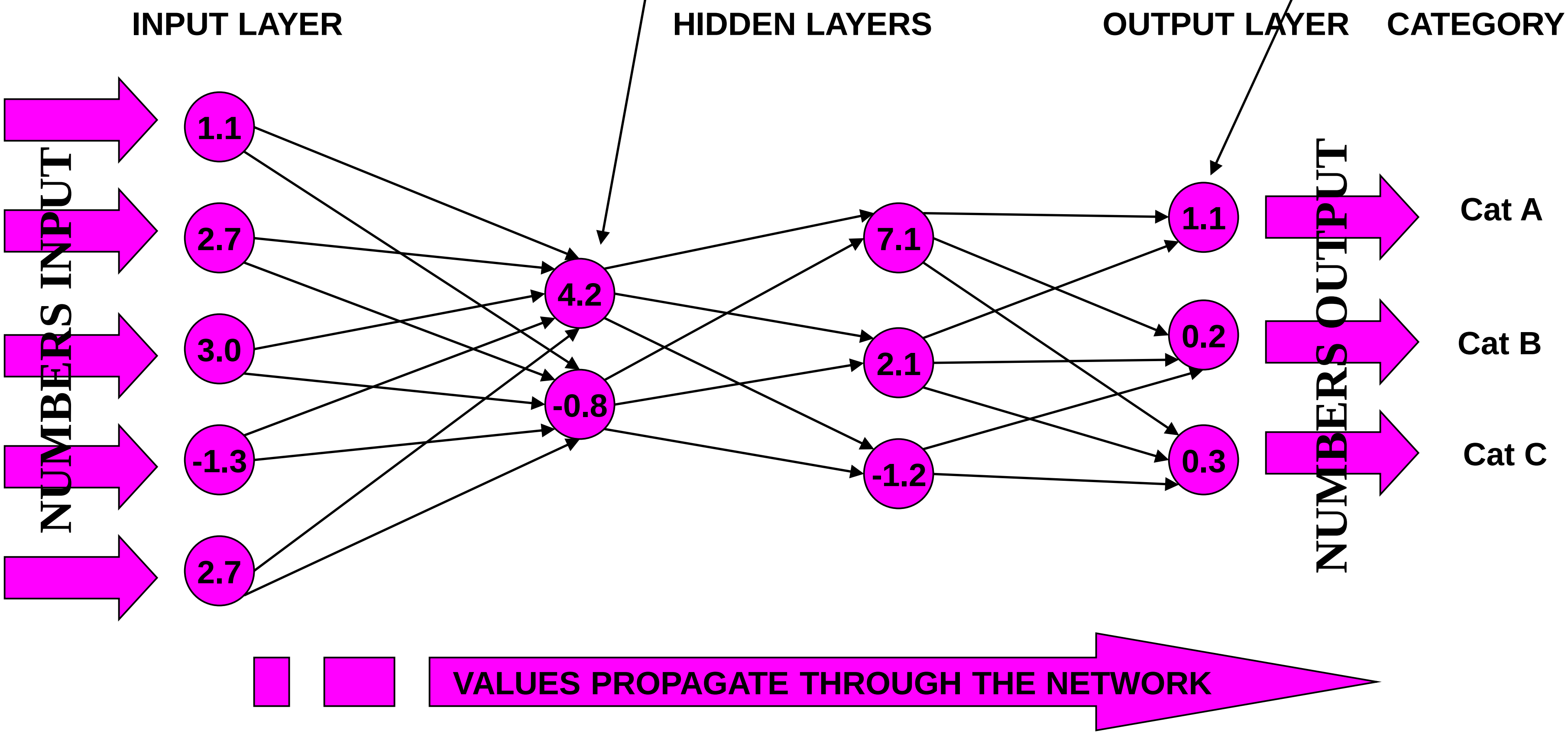
So, what functions can we use?

- Biological motivation:
  - The brain does categorisation tasks like this easily
  - The brain is made up of networks of neurons
- Naturally occurring neural networks
  - Each neuron is connected to many others
    - Input to one neuron is the output from many others
    - Neuron “fires” if a weighted sum S of inputs > threshold
- Artificial neural networks
  - Similar hierarchy with neurons firing
  - Don’t take the analogy too far
    - Human brains: 100,000,000,000 neurons
    - ANNs: < 1000 usually
    - ANNs are a gross simplification of real neural networks

General Idea

Value calculated using all the input unit values

Choose CAT output (for a given output value)



Representation of Information

- If ANNs can correctly identify vehicles
  - They then contain some notion of “car”, “bus”, etc.
- The categorisation is produced by the **units** (nodes)
  - Exactly how the input reals are turned into outputs!
- But, in practice:
  - Each unit does the same calculation
    - ◆But it is based on the **weighted sum** of inputs to the unit
  - So, the weights in the weighted sum
    - ◆Is where the information is really stored
- “Black Box” representation:
  - Useful knowledge about learned concept is difficult to extract

ANN learning problem

- Given a categorisation to learn (expressed numerically)
  - And training examples represented numerically
    - With the correct categorisation for each example
- Learn a neural network using the examples
  - which produces the correct output for unseen examples
- Boils down to

  (a) Choosing the correct network architecture

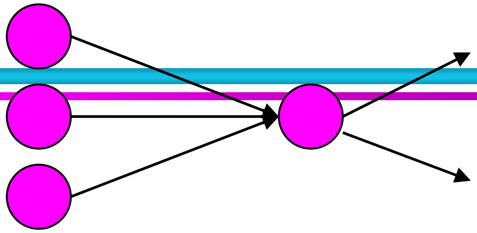
    - Number of hidden layers, number of units, etc.

  (b) Choosing (the same) function for each unit

  (c) Training the weights between units to work correctly

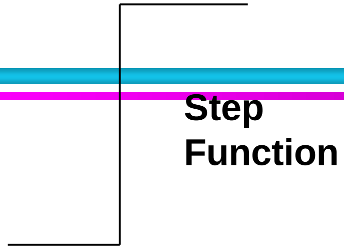
Perceptron ➡

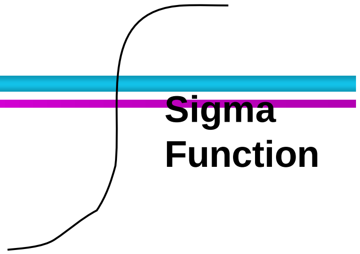
Simplest ANN - Perceptron



- Multiple input nodes
- Single output node
  - Takes a weighted sum of the inputs, call this S
  - Unit function calculates the output for the network
- Useful to study because
  - We can use perceptrons to build larger networks
- Perceptrons have limited representational abilities

Unit Functions





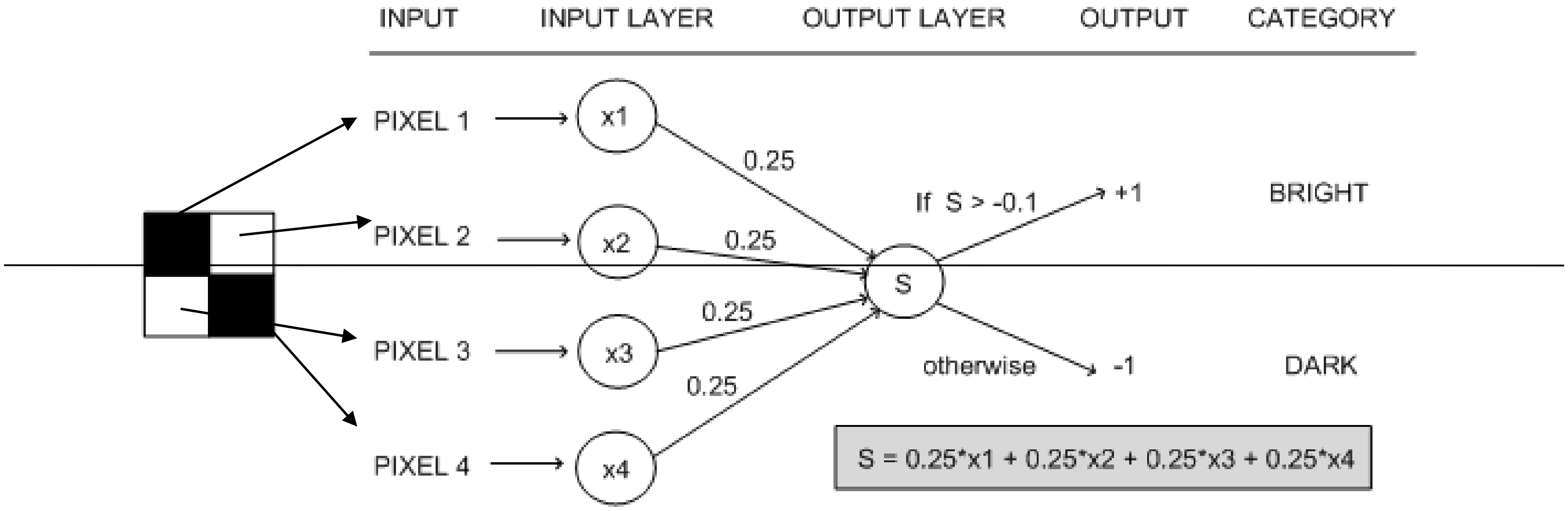
- Linear Functions
  - Simply output the weighted sum
- Threshold Functions
  - Output low values
    - Until the weighted sum gets over a threshold
    - Then output high values
    - Equivalent of “firing” of neurons
- Step function:
  - Output +1 if S > Threshold T
  - Output –1 otherwise
- Sigma function:
  - Similar to step function but differentiable

Example Perceptron

- Categorisation of 2x2 pixel black & white images
  - Into “bright” and “dark”
- Representation of this rule:
  - If it contains 2, 3 or 4 white pixels, it is “bright”
  - If it contains 0 or 1 white pixels, it is “dark”
- Perceptron architecture:
  - Four input units, one for each pixel
  - One output unit: +1 for white, -1 for dark

Example Perceptron

- Example calculation: x_1=-1, x_2=1, x_3=1, x_4=-1
  - S = 0.25*(-1) + 0.25*(1) + 0.25*(1) + 0.25*(-1) = 0
- 0 > -0.1, so the output from the ANN is +1
  - So the image is categorised as “bright”



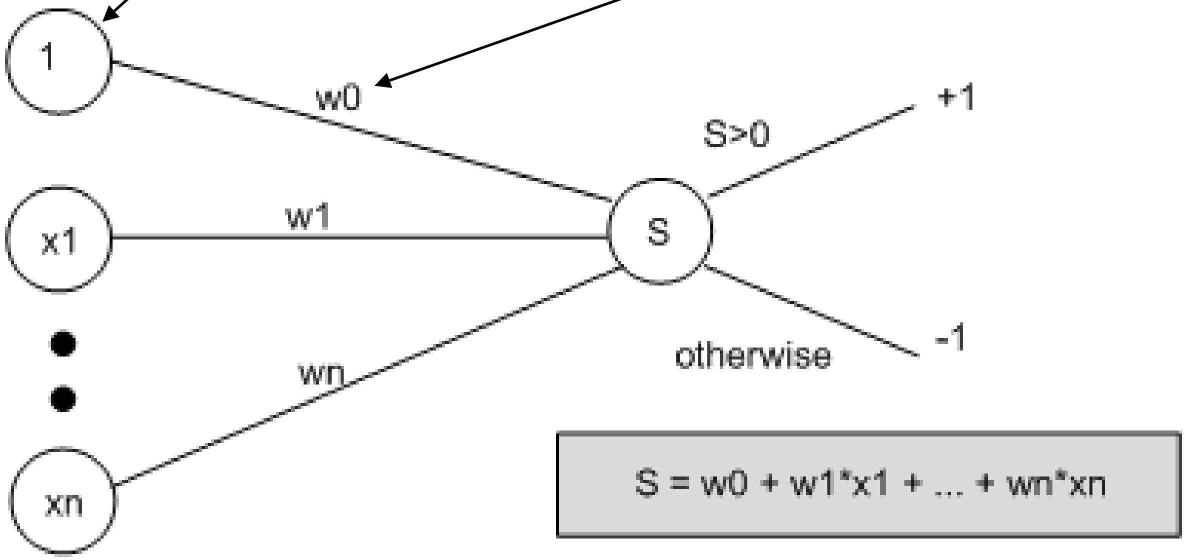
Learning in Perceptrons

- Need to learn
  - Both the weights between input and output units
  - And the value for the threshold
- Make calculations easier by
  - Thinking of the threshold as a weight from a special input unit where the output from the unit is always 1
- Exactly the same result
  - But we only have to worry about learning weights

New Representation for Perceptrons

Special Input Unit
Always produces 1

Threshold function
has become this



S = w0 + w1*x1 + ... + wn*xn

Learning Algorithm

- Weights are set randomly initially
- For each training example E
  - Calculate the observed output from the ANN, o(E)
  - If the target output t(E) is different to o(E)
    - Then tweak all the weights so that o(E) gets closer to t(E)
    - Tweaking is done by perceptron training rule
- This routine is done for every example E
- Don’t necessarily stop when all examples used
  - Repeat the cycle again (an ‘epoch’)
  - Until the ANN produces the correct output
    - For *all* the examples in the training set (or good enough)

Perceptron Training Rule

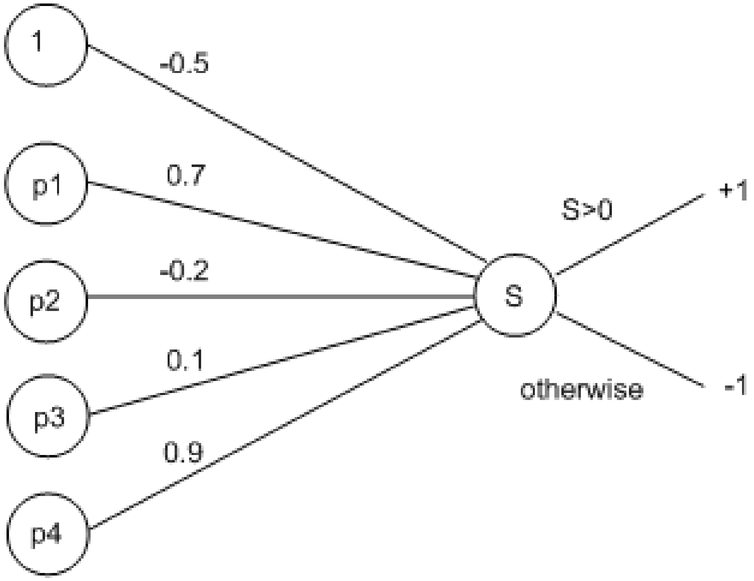
- When t(E) is different to o(E)
  - Add on Δ_i to weight w_i
  - Where $\Delta_i = \eta(t(E)-o(E))x_i$
  - Do this for every weight in the network
- Interpretation:
  - (t(E) – o(E)) will either be +2 or –2 [cannot be the same sign]
  - So we can think of the addition of Δ_i as the movement of the weight in a direction
    - Which will improve the networks performance with respect to E
  - Multiplication by xi
    - Moves it more if the input is bigger

The Learning Rate

- η is called the learning rate
  - Usually set to something small (e.g., 0.1)
- To control the movement of the weights
  - Not to move too far for one example
  - Which may over-compensate for another example
- If a large movement is actually necessary for the weights to correctly categorise E
  - This will occur over time with multiple epochs

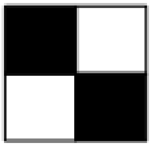
Worked Example

- Return to the “bright” and “dark” example
- Use a learning rate of η = 0.1
- Suppose we have set random weights:



Worked Example

- Use this training example, E, to update weights:



- Here, x1 = -1, x2 = 1, x3 = 1, x4 = -1 as before
- Propagate this information through the network:
  - S = (-0.5 * 1) + (0.7 * -1) + (-0.2 * +1) + (0.1 * +1) + (0.9 * -1) = -2.2
- Hence the network outputs o(E) = -1
- But this should have been “bright”=+1
  - So t(E) = +1

Calculating the Error Values

- $\Delta_0 = \eta(t(E)-o(E))x_0$

  $= 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2$

- $\Delta_1 = \eta(t(E)-o(E))x_1$

  $= 0.1 * (1 - (-1)) * (-1) = 0.1 * (-2) = -0.2$

- $\Delta_2 = \eta(t(E)-o(E))x_2$

  $= 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2$

- $\Delta_3 = \eta(t(E)-o(E))x_3$

  $= 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2$

- $\Delta_4 = \eta(t(E)-o(E))x_4$

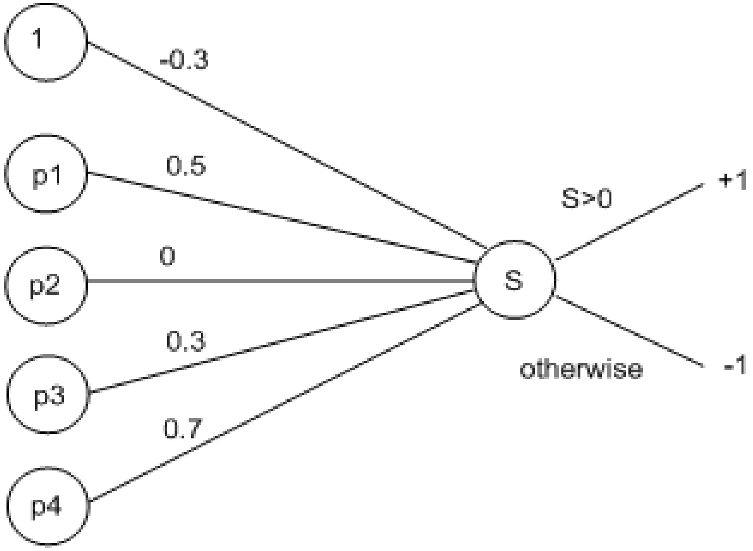
  $= 0.1 * (1 - (-1)) * (-1) = 0.1 * (-2) = -0.2$

Calculating the New Weights

- $w'_0 = -0.5 + \Delta_0 = -0.5 + 0.2 = -0.3$
- $w'_1 = 0.7 + \Delta_1 = 0.7 + -0.2 = 0.5$
- $w'_2 = -0.2 + \Delta_2 = -0.2 + 0.2 = 0$
- $w'_3= 0.1 + \Delta_3 = 0.1 + 0.2 = 0.3$
- $w'_4 = 0.9 + \Delta_4 = 0.9 - 0.2 = 0.7$

New Look Perceptron

- Calculate for the example, E, again:
  - S = (-0.3 * 1) + (0.5 * -1) + (0 * +1) + (0.3 * +1) + (0.7 * -1) = -1.2
- Still gets the wrong categorisation
  - But the value is closer to zero (from -2.2 to -1.2)
  - In a few epochs time, this example will be correctly categorised



Unified Perceptron Learning Rule

$$\text{If } t = 1 \text{ and } a = 0, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$$

$$\text{If } t = 0 \text{ and } a = 1, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$$

$$\text{If } t = a, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$$

These three rules can be rewritten as a single expression. First we will define a new variable, the perceptron error e:

$$e = t - a$$

$$\text{If } e = 1, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$$

$$\text{If } e = -1, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$$

$$\text{If } e = 0, \text{ then } {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$$

$$\boxed{{}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + e\mathbf{p} = {}_1\mathbf{w}^{old} + (t - a)\mathbf{p}}$$

Unified Learning Rule cont'd

- This rule can be extended to train the bias by noting that a bias is simply a *weight* whose input is always 1 i.e. p=1

•We can thus replace the input p in the Eq. below with the input to the bias, which is 1.

$$ {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + e\mathbf{p} = {}_1\mathbf{w}^{old} + (t - a)\mathbf{p} $$

•The result is the perceptron rule for a bias:

$$ b^{new} = b^{old} + e $$

Multiple-Neuron Perceptrons

To update the ith row of the weight matrix:

$${}_{i}\mathbf{w}^{new} = {}_{i}\mathbf{w}^{old} + e_{i}\mathbf{p}$$

$$b_i^{new} = b_i^{old} + e_i$$

Matrix form:

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

$$\mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e}$$

Apple/Banana Example

Training Set

$$\left\{\mathbf{p}_1 = \begin{bmatrix}-1\\1\\-1\end{bmatrix}, t_1 = \begin{bmatrix}1\end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix}1\\1\\-1\end{bmatrix}, t_2 = \begin{bmatrix}0\end{bmatrix}\right\}$$

Initial Weights

$$\mathbf{W} = \begin{bmatrix}0.5 & -1 & -0.5\end{bmatrix} \qquad b = 0.5$$

First Iteration

$$a = hardlim(\mathbf{W}\mathbf{p}_1 + b) = hardlim\left(\begin{bmatrix}0.5 & -1 & -0.5\end{bmatrix}\begin{bmatrix}-1\\1\\-1\end{bmatrix} + 0.5\right)$$

$$a = hardlim(-0.5) = 0 \qquad \boldsymbol{e = t_1 - a = 1 - 0 = 1}$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix}0.5 & -1 & -0.5\end{bmatrix} + (1)\begin{bmatrix}-1 & 1 & -1\end{bmatrix} = \begin{bmatrix}-0.5 & 0 & -1.5\end{bmatrix}$$

$$b^{new} = b^{old} + e = 0.5 + (1) = 1.5$$

Second Iteration

$$a = hardlim\,(\mathbf{W}\mathbf{p}_2 + b) = hardlim\,\left(\begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + (1.5)\right)$$

$$a = hardlim\,(2.5) = 1$$

$$e = t_2 - a = 0 - 1 = -1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix} -0.5 & 0 & -1.5 \end{bmatrix} + (-1)\begin{bmatrix} 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} -1.5 & -1 & -0.5 \end{bmatrix}$$

$$b^{new} = b^{old} + e = 1.5 + (-1) = 0.5$$

Check

$$a = hardlim(\mathbf{W}\mathbf{p}_1 + b) = hardlim\left(\begin{bmatrix} -1.5 & -1 & -0.5 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} + 0.5\right)$$

$$a = hardlim(1.5) = 1 = t_1$$

$$a = hardlim(\mathbf{W}\mathbf{p}_2 + b) = hardlim\left(\begin{bmatrix} -1.5 & -1 & -0.5 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + 0.5\right)$$

$$a = hardlim(-1.5) = 0 = t_2$$

Perceptron Rule Capability

Proof of Convergence: can consider it as exercise!

The perceptron rule will always converge to weights which accomplish the desired classification, assuming that such weights exist.

Learning Abilities of Perceptrons

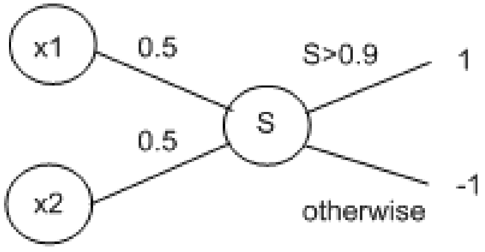
- Perceptrons are a very simple network
- Computational learning theory
  - Study of which concepts can and can’t be learned
    - By particular learning techniques (representation, method)
- Cannot learn some simple **boolean functions**
- Caused a “winter” of research for ANNs in AI
  - People thought it represented a fundamental limitation
  - But perceptrons are the simplest network
  - ANNS were revived by used of layered networks and backpropagation to solve non-linear problems.

Boolean Functions

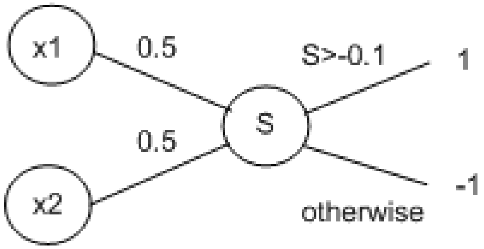
- Take in two inputs (-1 or +1)
- Produce one output (-1 or +1)
- In other contexts, use 0 and 1
- Example: AND function
  - Produces +1 only if *both* inputs are +1
- Example: OR function
  - Produces +1 if *either* inputs are +1

Boolean Functions as Perceptrons

- Problem: XOR boolean function
  - Produces +1 only if inputs are different
  - Cannot be represented as a perceptron
  - Because it is not linearly separable



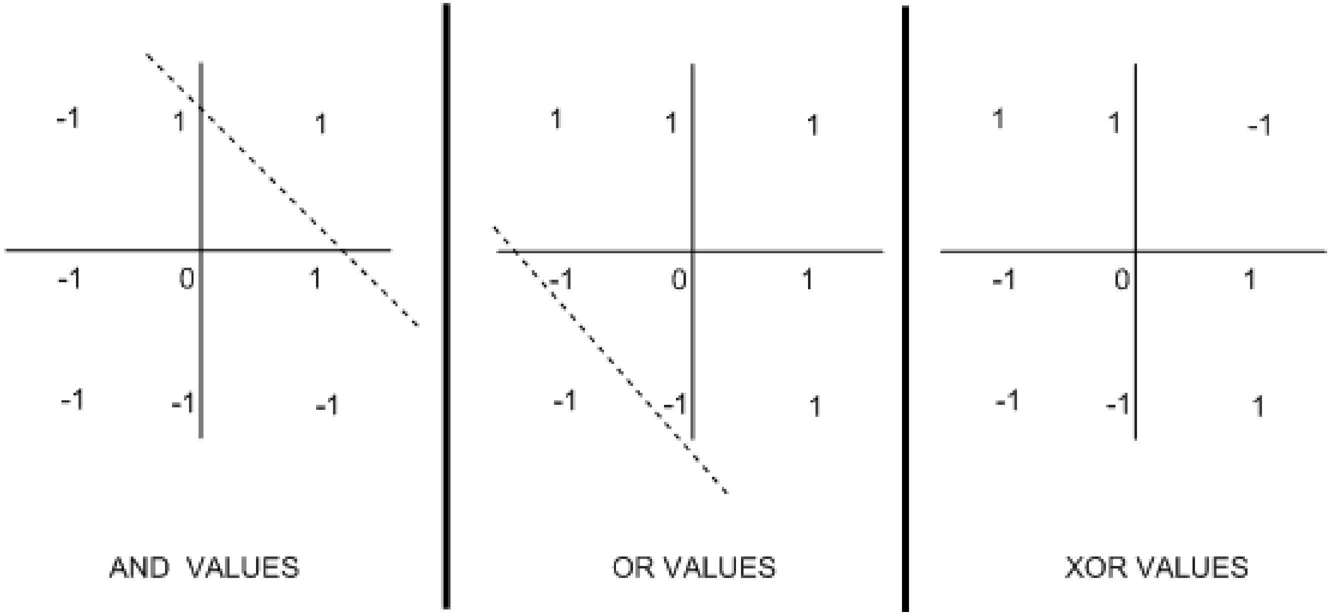
An ANN for AND



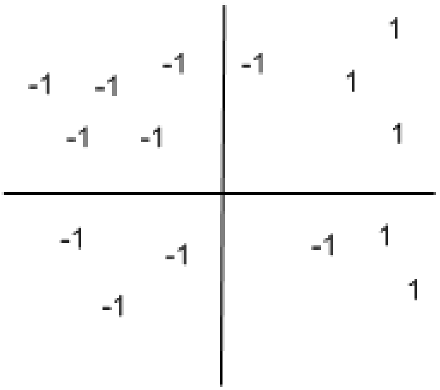
An ANN for OR

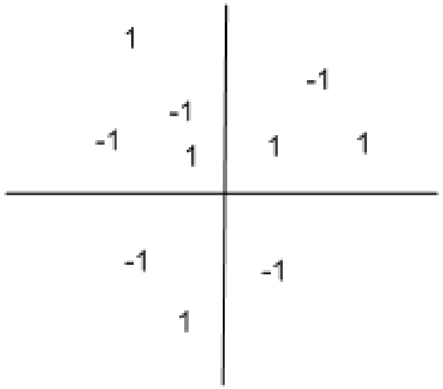
Linearly Separable Boolean Functions

- Linearly separable:
  - Can use a line (dotted) to separate +1 and –1
- Think of the line as representing the threshold
  - Angle of line determined by two weights in perceptron
  - Y-axis crossing determined by threshold



Linearly Separable Functions



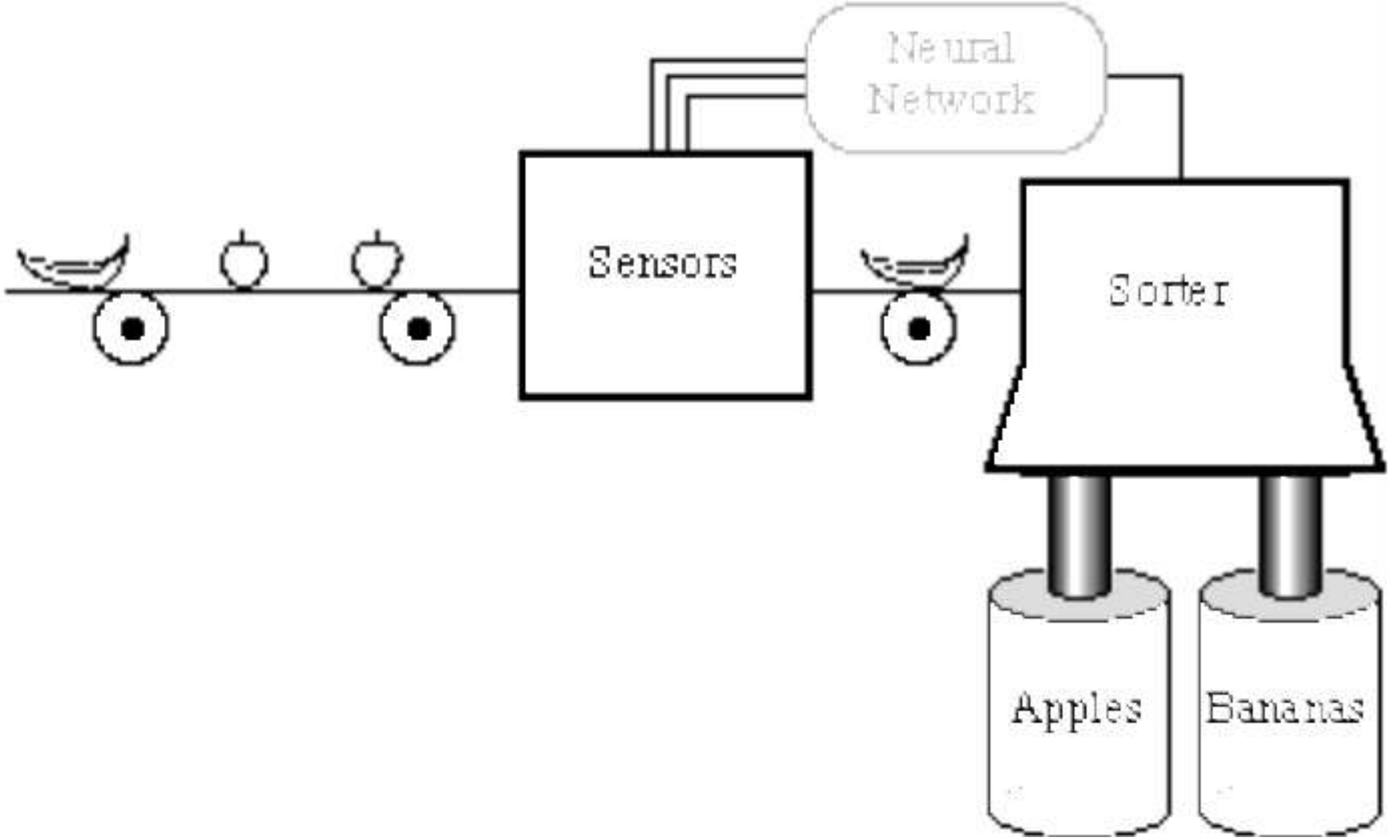


- Result extends to functions taking many inputs
  - And outputting +1 and –1
- Also extends to higher dimensions for outputs

Competitive Networks

Apple/Banana Sorter

Recall the Banana/Apple problem



Prototype Vectors

Measurement Vector

$$\mathbf{p} = \begin{bmatrix} \text{shape} \\ \text{texture} \\ \text{weight} \end{bmatrix}$$

Prototype Banana **Prototype Apple**

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

Shape: {1 : round ; -1 : eliptical}
Texture: {1 : smooth ; -1 : rough}
Weight: {1 : > 1 lb. ; -1 : < 1 lb.}

The neural network will receive one three-dimensional input vector for each fruit on the conveyer and must make a decision as to whether the fruit is a banana (p_1) or an apple (p_2).

Feedforward network

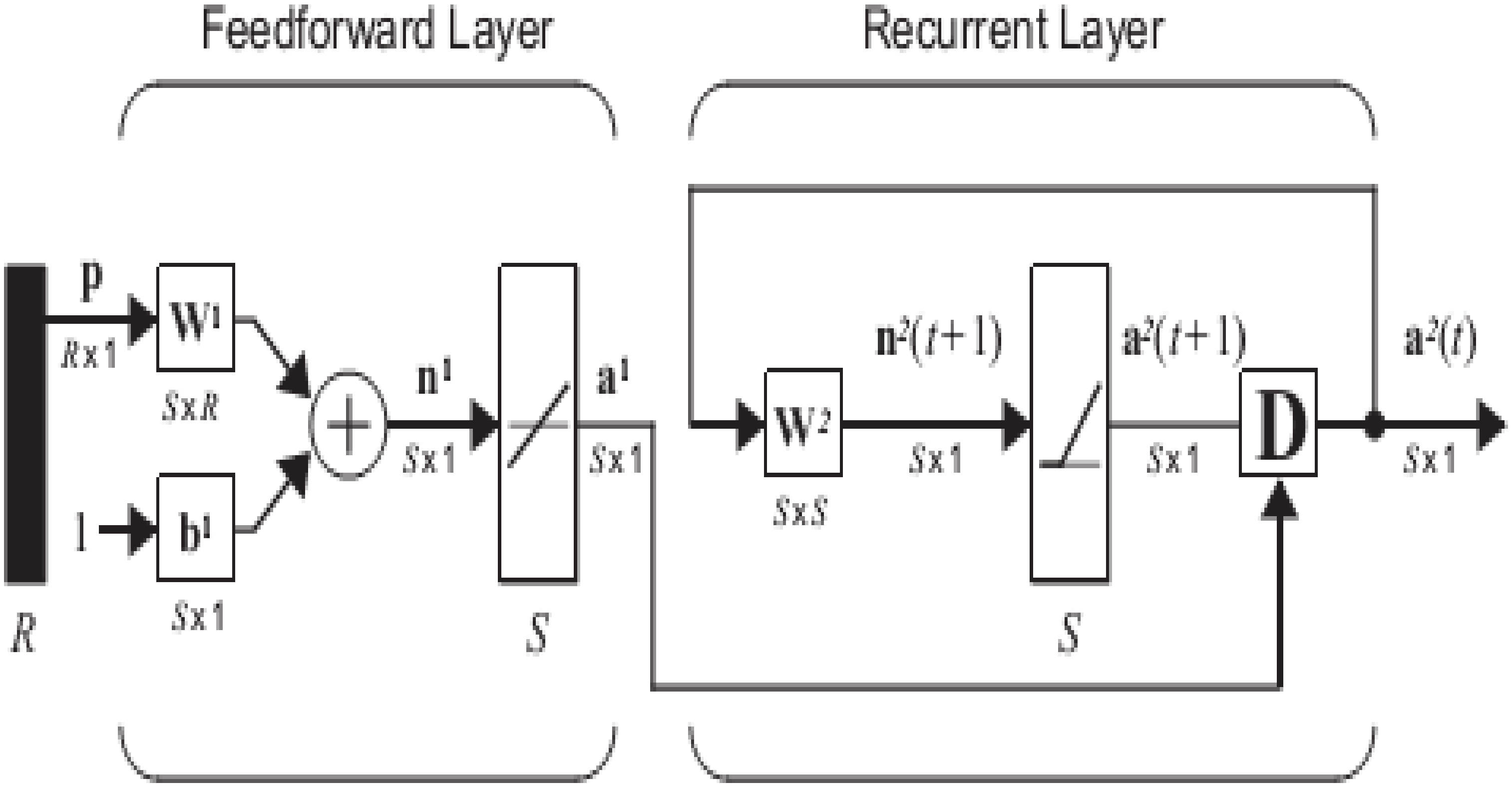
- Recall for a Perceptron (an example of a feedforward network) it can be shown that the classification (decision boundary) can be achieved by developing weights **w** and a bias **b** such that:

$${}_i\mathbf{w}^T\mathbf{p} + b_i = 0$$

Hamming Network

- It was designed explicitly to solve binary pattern recognition problems (where each element of the input vector has only two possible values — in the banana/apple example we have 1 or -1).
- it uses both feedforward and recurrent (feedback) layers
- Note that the number of neurons in the first layer is the same as the number of neurons in the second layer.

Hamming Network cont'd



$$\mathbf{a}^1 = \text{purelin}(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) \qquad \mathbf{a}^2(0) = \mathbf{a}^1 \qquad \mathbf{a}^2(t+1) = \text{poslin}(\mathbf{W}^2\mathbf{a}^2(t))$$

Hamming Network cont’d

General Concepts on Hamming network operation:

•The objective of the Hamming network is to decide which prototype vector is closest to the input vector.

•This decision is indicated by the output of the recurrent layer.

•There is one neuron in the recurrent layer for each prototype pattern. When the recurrent layer converges, there will be only one neuron with nonzero output.

•This neuron indicates the prototype pattern that is closest to the input vector.

Hamming Network cont'd

Feedforward Layer

- The feedforward layer performs a correlation, or inner product, between each of the prototype patterns and the input pattern
- In order for the feedforward layer to perform this correlation, the rows of the weight matrix in the feedforward layer, represented by the connection matrix $\mathbf{w}^1$, are set to the prototype patterns.

Hamming Network cont'd

Based on the Banana/Apple example this would mean:

Prototype Banana **Prototype Apple**

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

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

Hamming Network cont'd

- The feedforward layer uses a linear transfer function, and each element of the bias vector ***b*** is equal to *R*, where *R* is the number of elements in the input vector
- For our banana/apple example the bias vector would be:

$$\mathbf{b}^1 = \begin{bmatrix} R \\ R \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

- **With these choices for the weight matrix and bias vector, the output of the feedforward layer is:**

$$\mathbf{a}^1 = \mathbf{W}^1\mathbf{p} + \mathbf{b}^1 = \begin{bmatrix} \mathbf{p}_1^{\mathrm{T}} \\ \mathbf{p}_2^{\mathrm{T}} \end{bmatrix} \mathbf{p} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^{\mathrm{T}}\mathbf{p} + 3 \\ \mathbf{p}_2^{\mathrm{T}}\mathbf{p} + 3 \end{bmatrix}$$

Hamming Network cont'd

Remark:

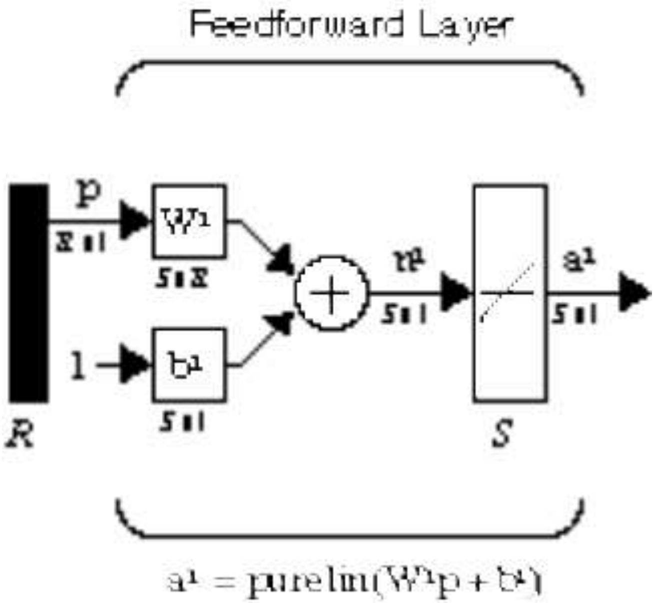
- Note that the outputs of the feedforward layer are equal to the inner products of each prototype pattern with the input, plus *R*. For two vectors of equal length, their inner product will be largest when the vectors point in the same direction, and will be smallest when they point in opposite directions.

- By adding *R* to the inner product we guarantee that the outputs of the feedforward layer can never be negative.

Hamming Network cont’d

- This network is called the Hamming network because the neuron in the feedforward layer with the largest output will correspond to the prototype pattern that is closest in Hamming distance to the input pattern.
- The Hamming distance between two vectors is equal to the number of elements that are different, defined only for binary vectors.

Summary - Feedforward Layer



For Banana/Apple Recognition

$$S = 2$$

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

$$\mathbf{b}^1 = \begin{bmatrix} R \\ R \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{a}^1 = \mathbf{W}^1\mathbf{p} + \mathbf{b}^1 = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix}\mathbf{p} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T\mathbf{p} + 3 \\ \mathbf{p}_2^T\mathbf{p} + 3 \end{bmatrix}$$

Hamming Network cont'd

Recurrent Layer

- The recurrent layer of the Hamming network is what is known as a "competitive" layer.
- The neurons in this layer are initialized with the outputs of the feedforward layer, which indicate the correlation between the prototype patterns and the input vector.
- Then the neurons compete with each other to determine a winner.
- After the competition, only one neuron will have a nonzero output. The winning neuron indicates which category of in-put was presented to the network

Hamming Network cont’d

- The competition is governed by the equation:

$$\mathbf{a}^2(t+1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(t))$$

Where;

$$\mathbf{a}^2(0) = \mathbf{a}^1 \qquad \text{(Initial Condition)}$$

- Here 2 represents the layer number not a power!
- The transfer function *poslin* is linear for positive values and zero for negative values

Hamming Network cont'd

- The weight matrix W^2 has the form:

$$\mathbf{W}^2 = \begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix}$$

where *ε* is some number less than *1 / (S-1)*, and *S* is the number of neurons in the recurrent layer.

•An iteration of the recurrent layer proceeds as follows:

$$\mathbf{a}^2(t+1) = \mathbf{poslin}\left(\begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix}\mathbf{a}^2(t)\right) = \mathbf{poslin}\left(\begin{bmatrix} a_1^2(t) - \varepsilon a_2^2(t) \\ a_2^2(t) - \varepsilon a_1^2(t) \end{bmatrix}\right)$$

Hamming Network cont’d

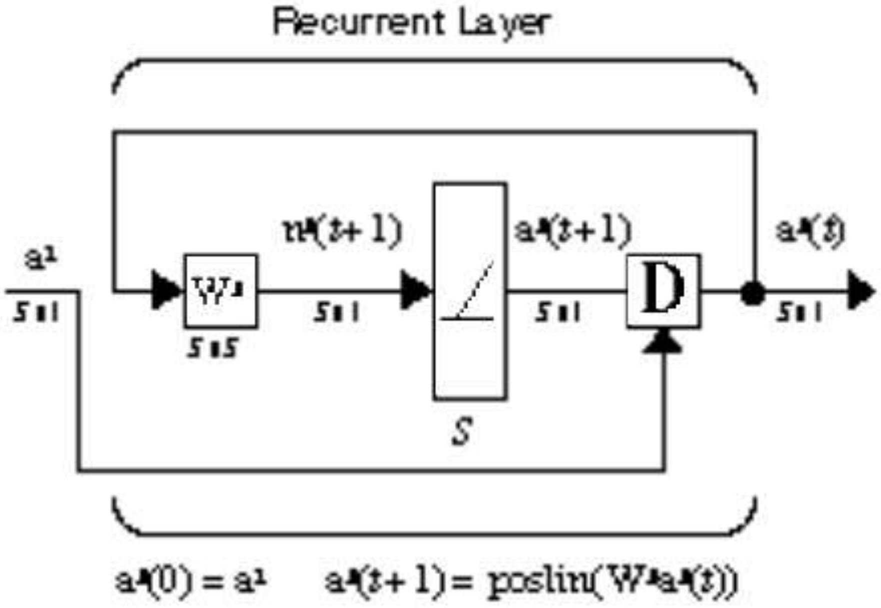
$$\mathbf{a}^2(t+1) = \mathbf{poslin}\left(\begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix}\mathbf{a}^2(t)\right) = \mathbf{poslin}\left(\begin{bmatrix} a_1^2(t) - \varepsilon a_2^2(t) \\ a_2^2(t) - \varepsilon a_1^2(t) \end{bmatrix}\right)$$

From above it is clear that:

•Each element is reduced by the same fraction of the other. The larger element will be reduced by less, and the smaller element will be reduced by more, therefore the difference between large and small will be increased.

•The effect of the recurrent layer is to *zero out* all neuron outputs, except the one with the largest initial value (which corresponds to the prototype pattern that is closest in Hamming distance to the input!).

Summary - Recurrent Layer



$$\mathbf{W}^2 = \begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix} \qquad \varepsilon < \frac{1}{S-1}$$

$$\mathbf{a}^2(t+1) = \mathbf{poslin}\left(\begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix}\mathbf{a}^2(t)\right) = \mathbf{poslin}\left(\begin{bmatrix} a_1^2(t) - \varepsilon a_2^2(t) \\ a_2^2(t) - \varepsilon a_1^2(t) \end{bmatrix}\right)$$

Illustrative Example

Prototype Banana **Prototype Apple**

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

Shape: {1 : round ; -1 : eliptical}
Texture: {1 : smooth ; -1 : rough}
Weight: {1 : > 1 lb. ; -1 : < 1 lb.}

First Layer

Input (Rough Banana)

$$\mathbf{p} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

$$\mathbf{a}^1 = \mathbf{W}^1\mathbf{p} + \mathbf{b}^1 = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix}\mathbf{p} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T\mathbf{p} + 3 \\ \mathbf{p}_2^T\mathbf{p} + 3 \end{bmatrix}$$

$$\mathbf{a}^1 = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}\begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} (1 + 3) \\ (-1 + 3) \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

Illustrative Example

Second Layer

$$\mathbf{W}^2 = \begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix} \qquad \varepsilon < \frac{1}{S-1} \qquad \mathbf{a}^2(t+1) = \mathbf{poslin}\left(\begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix}\mathbf{a}^2(t)\right) = \mathbf{poslin}\left(\begin{bmatrix} a_1^2(t) - \varepsilon a_2^2(t) \\ a_2^2(t) - \varepsilon a_1^2(t) \end{bmatrix}\right)$$

$$\mathbf{a}^2(0) = \mathbf{a}^1 \qquad \text{(Initial Condition)}$$

S = 2 , ε = 0.5

$$\mathbf{a}^2(1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(0)) = \begin{cases} \mathbf{poslin}\left(\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}\begin{bmatrix} 4 \\ 2 \end{bmatrix}\right) \\ \mathbf{poslin}\left(\begin{bmatrix} 3 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \end{cases}$$

$$\mathbf{a}^2(2) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(1)) = \begin{cases} \mathbf{poslin}\left(\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ 0 \end{bmatrix}\right) \\ \mathbf{poslin}\left(\begin{bmatrix} 3 \\ -1.5 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \end{cases}$$

Illustrative Example

- Since the outputs of successive iterations produce the same result, the network has converged.
- Prototype pattern number one, the banana, is chosen as the correct match, since neuron number one has the only nonzero output.

Exercise:

- show that the outputs of the feedforward layer are equal to:

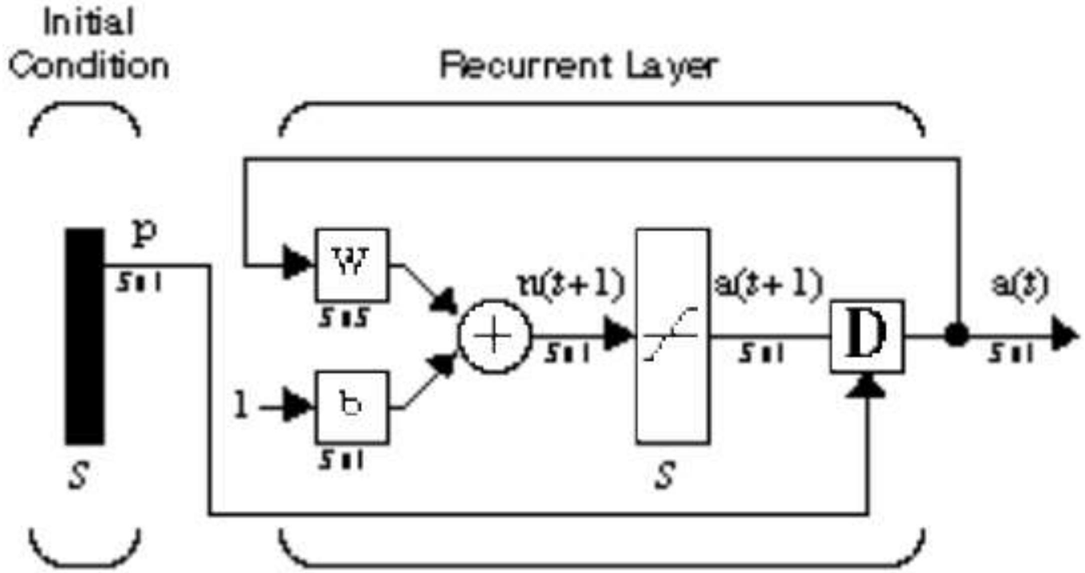
  = 2*R* – 2*distance* , where*; distance* = Hamming distances from the prototype patterns to the input pattern.
- Consider a different value for ε provided $\varepsilon < \frac{1}{S-1}$

Reccurent Networks

Hopfield Network

- This is a recurrent network that is similar in some respects to the recurrent layer of the Hamming network, but which can effectively perform the operations of both layers of the Hamming network.

Hopfield Network



$a(0) = p \qquad a(t+1) = \mathrm{satlins}(Wa(t) + b)$

Hopfield Network

- The neurons in this network are initialized with the input vector,then the network iterates until the output converges.
- When the network is operating correctly, the resulting output should be one of the *prototype vectors*.
- Therefore, whereas in the Hamming network the nonzero neuron indicates which prototype pattern is chosen, the Hopfield network actually produces the selected prototype pattern at its output.

Hopfield Network

- The network is governed by the equation:

$$\mathbf{a}(t+1) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(t) + \mathbf{b})$$

Where;

$$\mathbf{a}(0) = \mathbf{p}$$

and

satlins is the transfer function that is linear in the range [-1, 1] and saturates at 1 for inputs greater than 1 and at -1 for inputs less than -1.

Hopfield Network

- The design of the weight matrix and the bias vector for the Hopfield network is a more complex procedure than it is for the Hamming network.
- Just to illustrate the operation of the network, we use a weight matrix and a bias vector that can solve our banana and apple pattern recognition problem.

$$\mathbf{W} = \begin{bmatrix} 1.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ 0.9 \\ -0.9 \end{bmatrix}$$

Hopfield Network

Remark:

- We want the network output to converge to either the banana pattern,p$_1$, or the apple pattern,p$_2$. In both patterns, the second element is 1, and the third element is -1. The difference between the patterns occurs in the first element. Therefore, no matter what pattern is input to the network, we want the second element of the output pattern to converge to 1,the third element to converge to -1, and the first element to go to either 1or -1, whichever is closer to the first element of the input vector!

Prototype Banana **Prototype Apple**

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

Apple/Banana Problem

Thus the equations of operation of the Hopfield network are:

$$\mathbf{W} = \begin{bmatrix} 1.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ 0.9 \\ -0.9 \end{bmatrix}$$

$$\mathbf{a}(t+1) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(t) + \mathbf{b})$$

$$a_1(t+1) = satlins(1.2a_1(t))$$

$$a_2(t+1) = satlins(0.2a_2(t) + 0.9)$$

$$a_3(t+1) = \mathrm{satlins}(0.2a_3(t) - 0.9)$$

- Regardless of the initial values,$a_1(0)$, the second element will be increased until it saturates at 1, and the third element will be decreased until it saturates at -1. The first element is multiplied by a number larger than 1. Therefore, if it is initially negative, it will eventually saturate at -1; if it is initially positive it will saturate at 1.

Apple/Banana Problem

- Thus;

Test: “Rough” Banana

$$\mathbf{a}(0) = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \qquad \mathbf{a}(t+1) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(t) + \mathbf{b})$$

$$\mathbf{W} = \begin{bmatrix} 1.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.2 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ 0.9 \\ -0.9 \end{bmatrix} \qquad \mathbf{a}(1) = \begin{bmatrix} -1 \\ 0.7 \\ -1 \end{bmatrix} \qquad \mathbf{a}(2) = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

$$\mathbf{a}(3) = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

(Banana)