9.54
class 4

Supervised learning

Shimon Ullman + Tomaso Poggio

Danny Harari + Daneil Zysman + Darren Seibert

Intro

An old and simple model of supervised learning

associate b to a and store:

$$\phi_{b,a}(x) = b * a = \int b(\xi)a(x-\xi)d\xi$$

retrieve output b from input a – if $a \odot a \approx a$

$$a \odot \phi_{b,a}(x) = \int a(\tau)\phi_{b,a}(\tau + x)d\tau \approx a$$

An old and simple model of supervised learning

when

$$\phi(x) = \sum b_i * a_i$$

retrieve output b from input a — if $a_j \odot a_i \approx \delta_{i,j}$

$$a_j \odot \phi \approx b_j$$

It is a special case…

Linear

"Linear" learning

Suppose

$$x_i \in \mathbb{R}^n \; and \; y_i \in \mathbb{R}^m, \; i = 1, \cdots, N$$

Define

$$(x_1, \cdots, x_N) = X \; and \; (y_1, \cdots, y_N) = Y$$

Find linear operator (eg a matrix) such that

$$MX = Y$$

"Linear" learning

If X^{-1} exists, then

$$MX = Y \quad \Longrightarrow \quad M = YX^{-1}$$

If X^{-1} does not exists, then

$$MX = Y \quad \Longrightarrow \quad M = YX^{\dagger}$$

where the pseudo inverse is the solution of

$$min \; ||MX - Y||_F \quad \text{with} \quad ||A||_F = \sqrt{(\sum_{i,j} |a_{i,j}|^2)}$$

and if X is full column rank $X^{\dagger} = (X^T X)^{-1} X^T$

"Linear" learning is linear regression

If $m = 1$ e.g. the output y is scalar, then

$$Mx = y \implies y = m^T x = \sum_i m_i x_i$$

with $M = XY^{-1}$

Nonlinear

Nonlinear learning

Suppose

$$x_i \in \mathbb{R}^n \; and \; y_i \in \mathbb{R}^m, \; i = 1, \cdots, N$$

Define

$$(x_1, \cdots, x_N) = X \; and \; (y_1, \cdots, y_N) = Y$$

Find operator N such that

$$N \circ X = Y$$

In general impossible but…assume N is in the class of polynomial mappings of degree k in the vector space V (over the real field)…eg N has a convergent Taylor series expansion

Weierstrass theorem ensures approximation of any continuous function

Nonlinear learning

$$Y = L_o + L_1(X) + L_2(X, X) + ... + L_k(X, ..., X)$$

f(x) is a polynomial with all monomials as in this 2D example

$$y = a_1 x_1 + a_2 x_2 + b_1 x_1^2 + b_{12} x_1 x_2 + \cdots$$

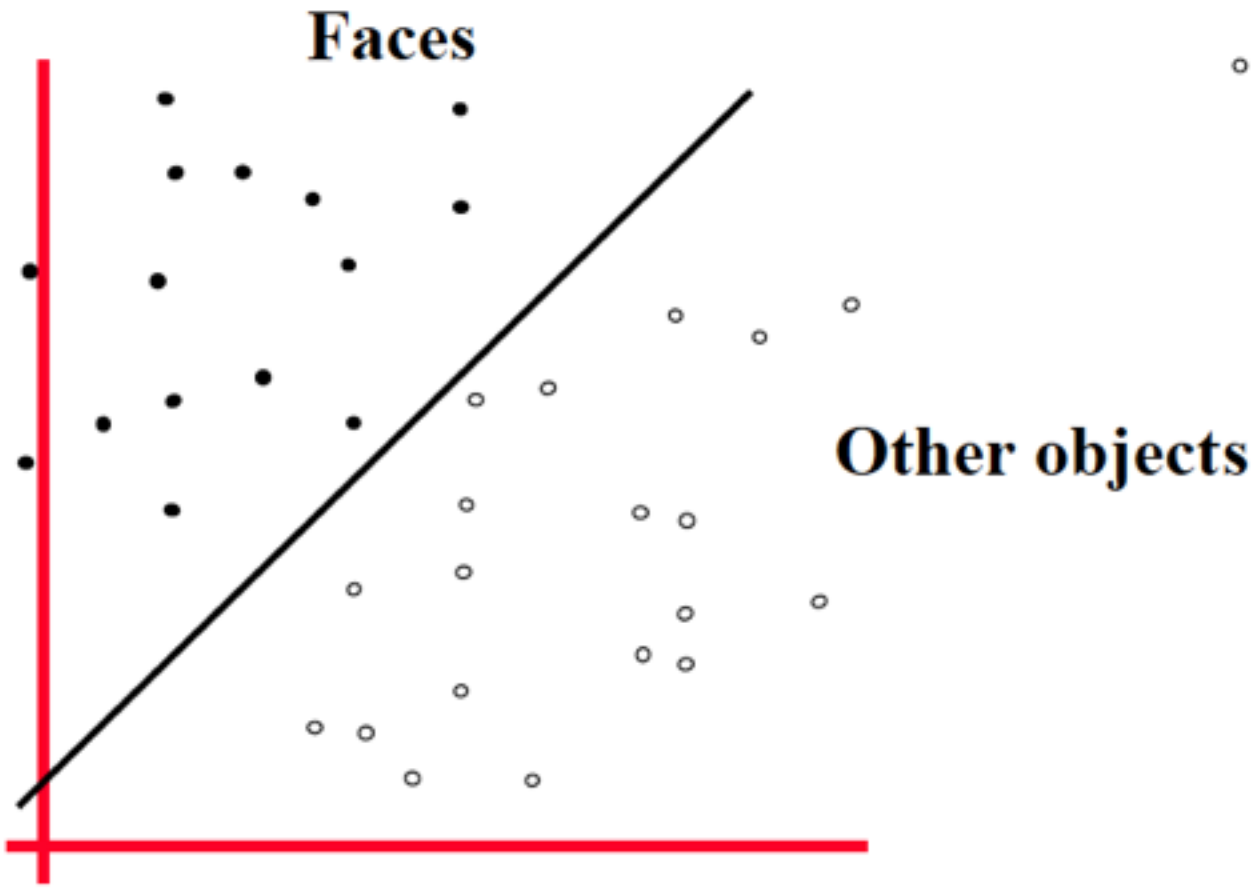
Classification and Regression

Supervised Learning

- Two Primary Tasks
  1. **Classification**
     - Inputs u_1, u_2, … and discrete classes C_1, C_2, …, C_k
     - Training examples: (u_1, C_2), (u_2, C_7), etc.
     - Learn the mapping from an arbitrary input to its class
     - Example: Inputs = images, output classes = face, not a face
  2. **Function Approximation (regression)**
     - Inputs u_1, u_2, … and continuous outputs v_1, v_2, …
     - Training examples: (input, desired output) pairs
     - Learn to map an arbitrary input to its corresponding output
     - Example: Highway driving
       Input = road image, output = steering angle

The Classification Problem

- denotes +1 (faces)
- denotes -1 (other)

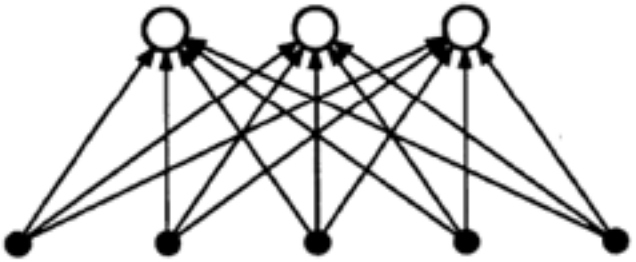


Idea: Find a separating hyperplane (line in this case)

Classification using “Perceptrons”

- Fancy name for a type of layered feedforward networks
- Uses artificial neurons (“units”) with binary inputs and outputs

Single-layer



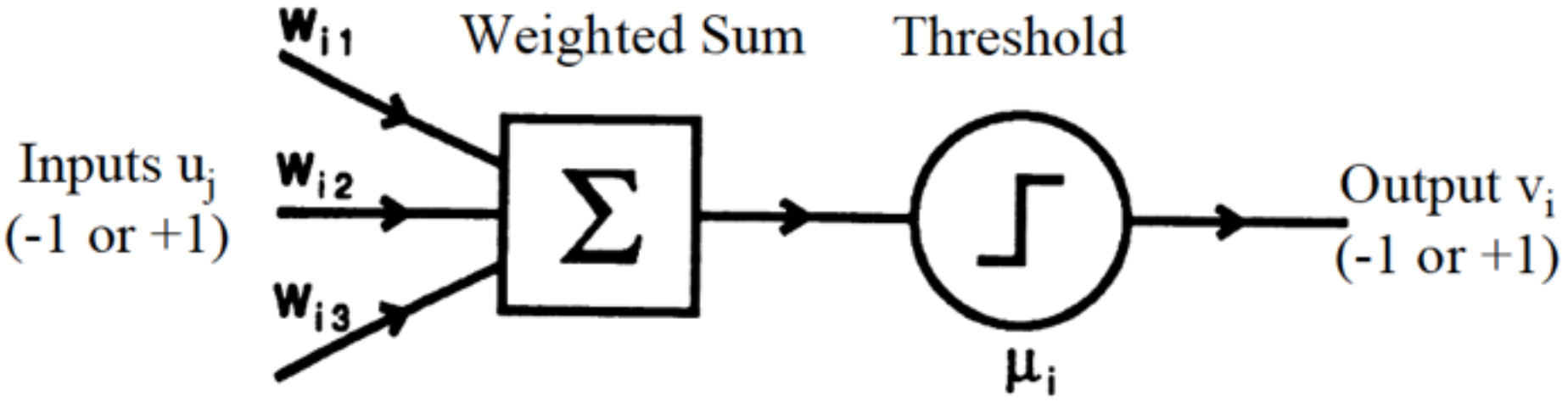
$$y = sign(Mx)$$

Perceptrons use "Threshold Units"

- Artificial neuron:
  - m binary inputs (-1 or 1) and 1 output (-1 or 1)
  - Synaptic weights w_{ij}
  - Threshold μ_i

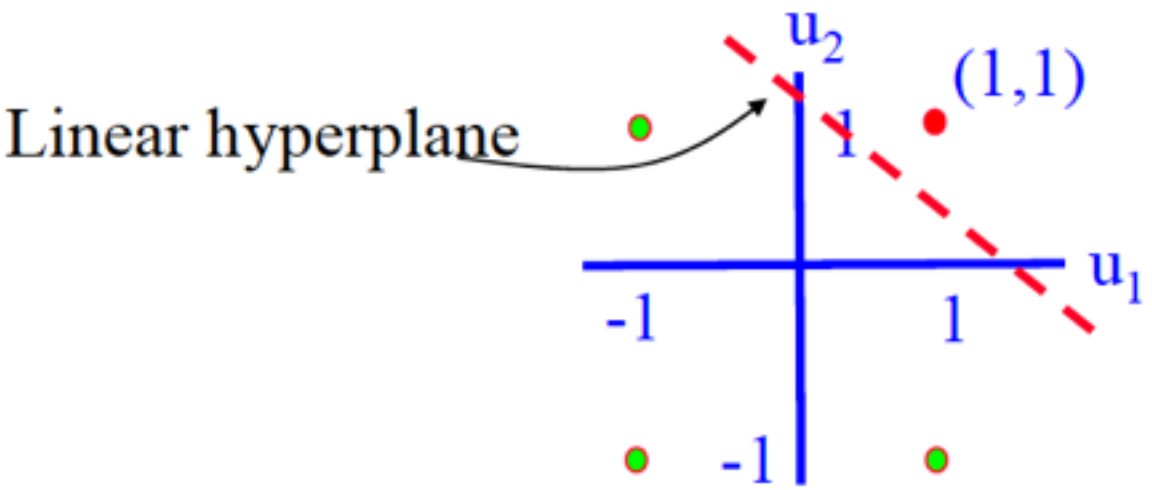
$$v_i = \Theta\left(\sum_j w_{ij} u_j - \mu_i\right)$$

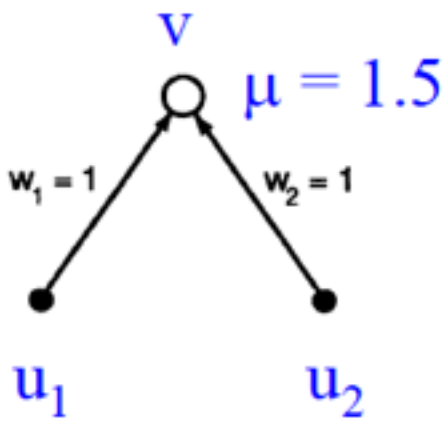
$\Theta(x) = 1$ if $x \geq 0$ and -1 if $x < 0$



Linear Separability

- Example: AND is linearly separable
  - a AND b = 1 if and only if a = 1 and b = 1

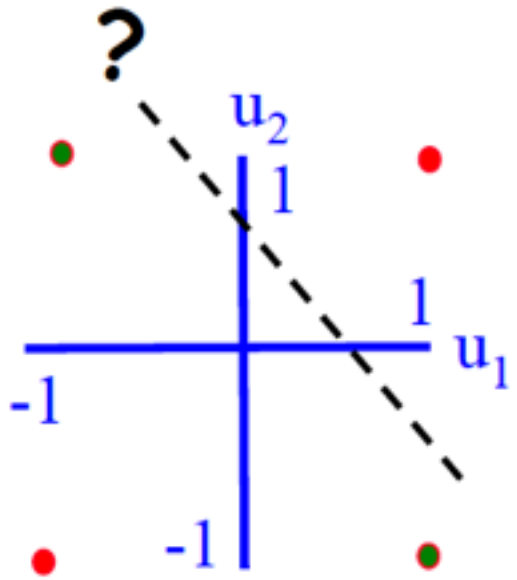




Perceptron for AND

What about the XOR function?

u_1	u_2	XOR
-1	-1	1
1	-1	-1
-1	1	-1
1	1	1

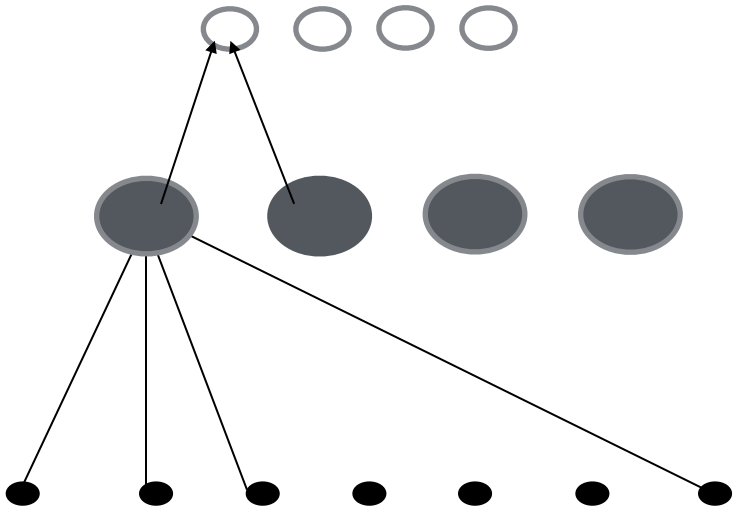


In our language: is L_1 enough?

XOR function

$$y = sign(L_1 x + L_2(x, x)) = sign(a_1 u_1 + a_2 u_2 + b u_1 u_2) = sign(u_1 u_2)$$

is in fact enough. This corresponds to a universal, one-hidden layer network



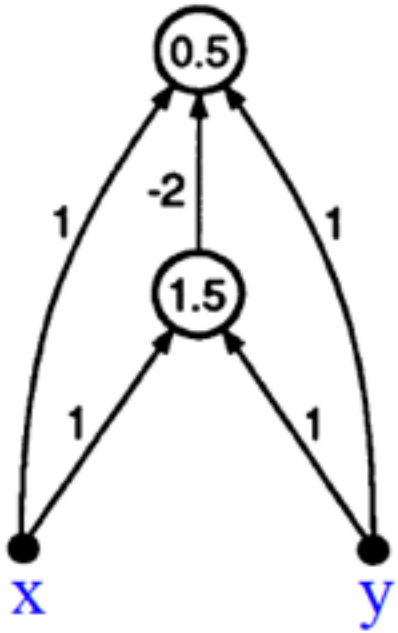
output layer

all monomials

input variables

Solution in 1980s: Multilayer perceptrons

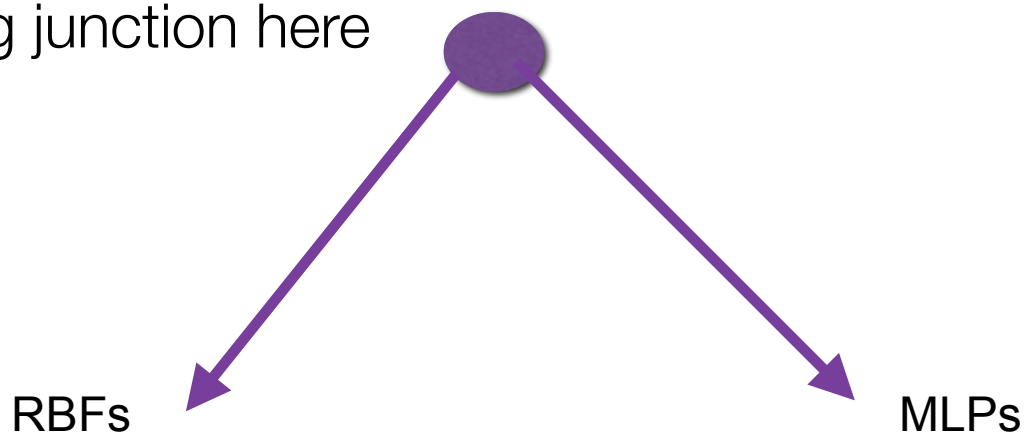
- Removes limitations of single-layer networks
  - Can solve XOR
- An example of a two-layer perceptron that computes XOR



- Output is +1 if and only if $x + y - 2\Theta(x + y - 1.5) - 0.5 > 0$

A few non-standard remarks

- Regression is king, Gauss knew everything…
- Perhaps no need of multiple layers…are 2 layers universal?
- An interesting junction here

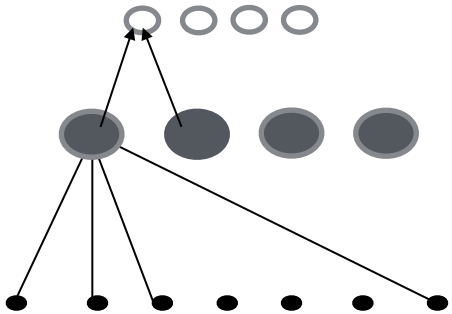


Radial Basis Functions

Nonlinear learning

Later we will see that RBF expansions are a good approximation of functions in high dimensions:

$$\sum_{k=1}^{N} c_k e^{-||x_k - x||^2}$$

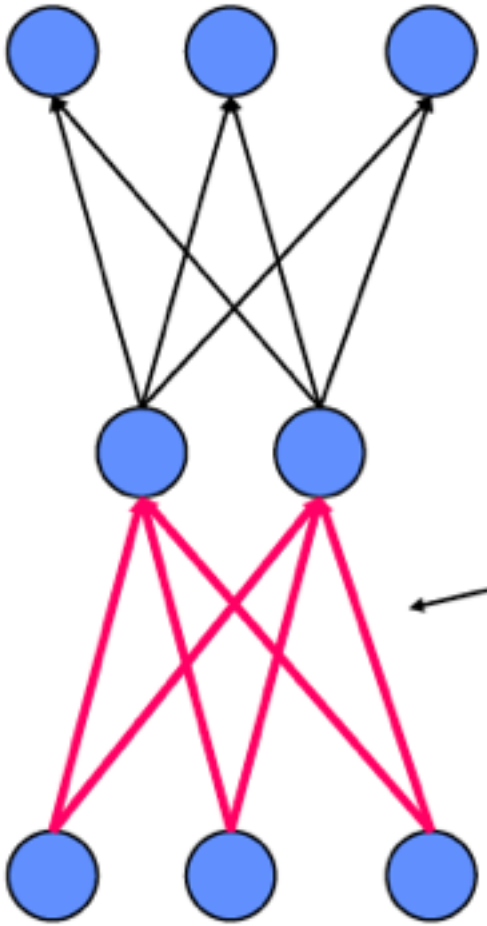


- RBF can be written as a 1-hidden layer network

- RBF is a rewriting of our polynomial (infinite radius of convergence)

$$e^{||\hat{x}_k - x||^2} = \sum_{n=0}^{\infty} \frac{||\hat{x}_k - x||^{2n}}{n!}$$

output neurons

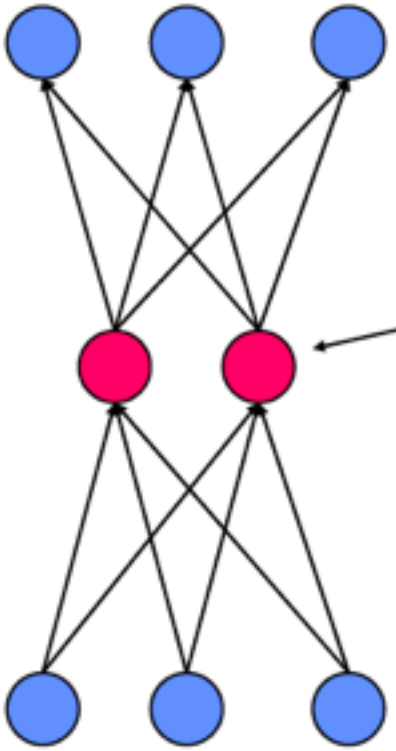


input nodes

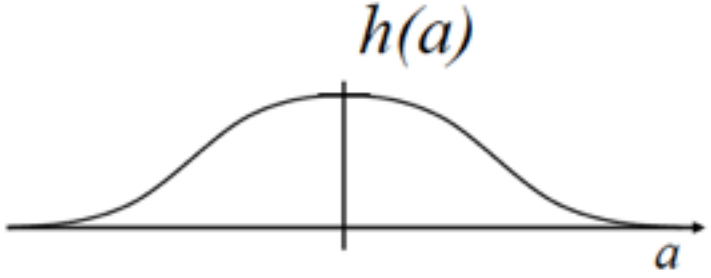
"activation" function:

$$a_j = \sqrt{\sum_{i=1}^{n} (x_i - \mu_{i,j})^2}$$

output neurons



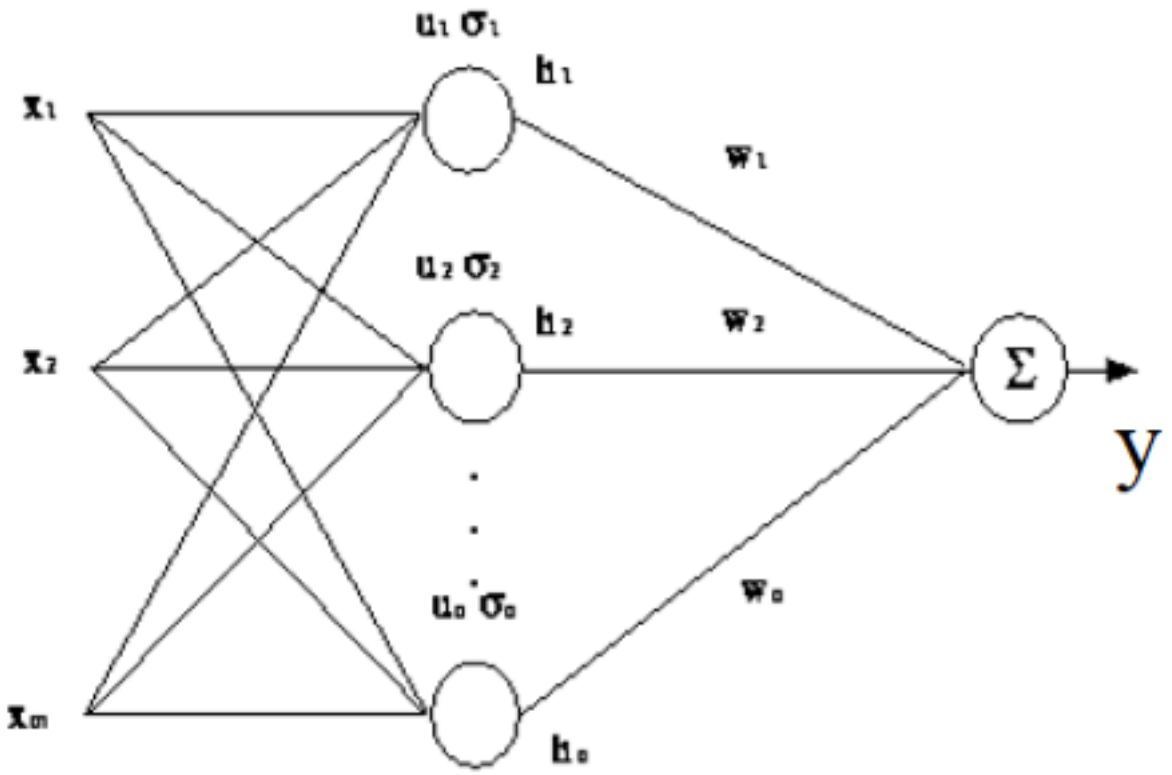
Hidden layer:
(Gaussian bell-shaped function)



$$h(a) = e^{-\frac{a^2}{2\sigma^2}}$$

input nodes

INPUTS HIDDEN LAYER OUTPUT



$$h_i = exp[-\frac{(\mathbf{x}-\mathbf{u}_i)^T(\mathbf{x}-\mathbf{u}_i)}{2\sigma^2}],\ y = \sum_i h_i w_i$$

Memory-based computation

$$f(x) = \sum_i c_i G(x, x_i) = \sum_i c_i e^{-\frac{||x - x_i||^2}{2\sigma^2}}$$

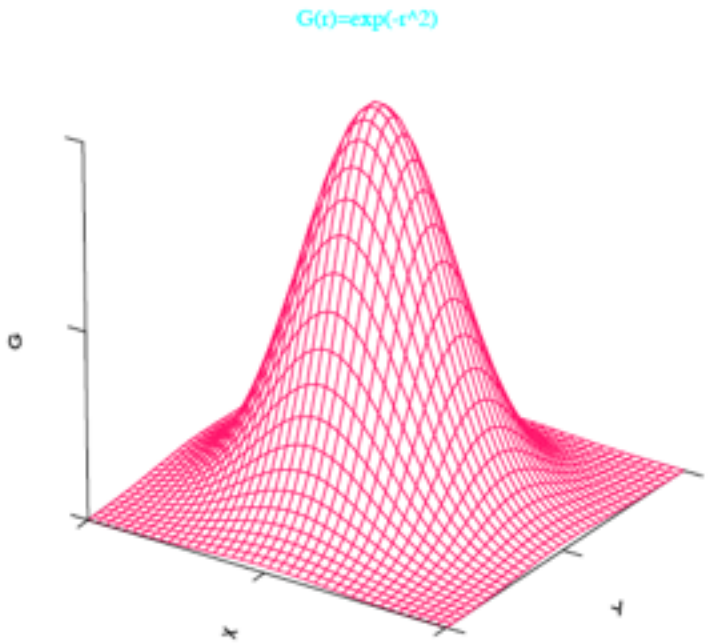
The training set is $(x_1, \cdots, x_N) = X \; and \; (y_1, \cdots, y_N) = Y$

Suppose now that $e^{-\frac{||x - x_i||^2}{2\sigma^2}} \rightarrow \delta(x - x_i)$: then it is a memory, a lookup table

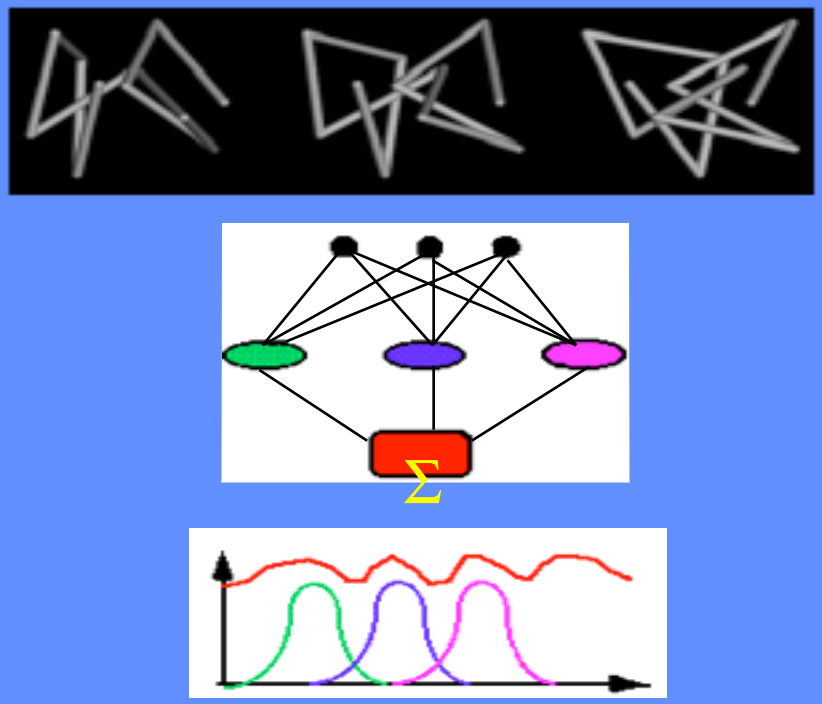
$$f(x) = \begin{cases} y, & \text{if } x = x_i \\ 0, & \text{if } x \neq x_i \end{cases}$$

Memory-based computation

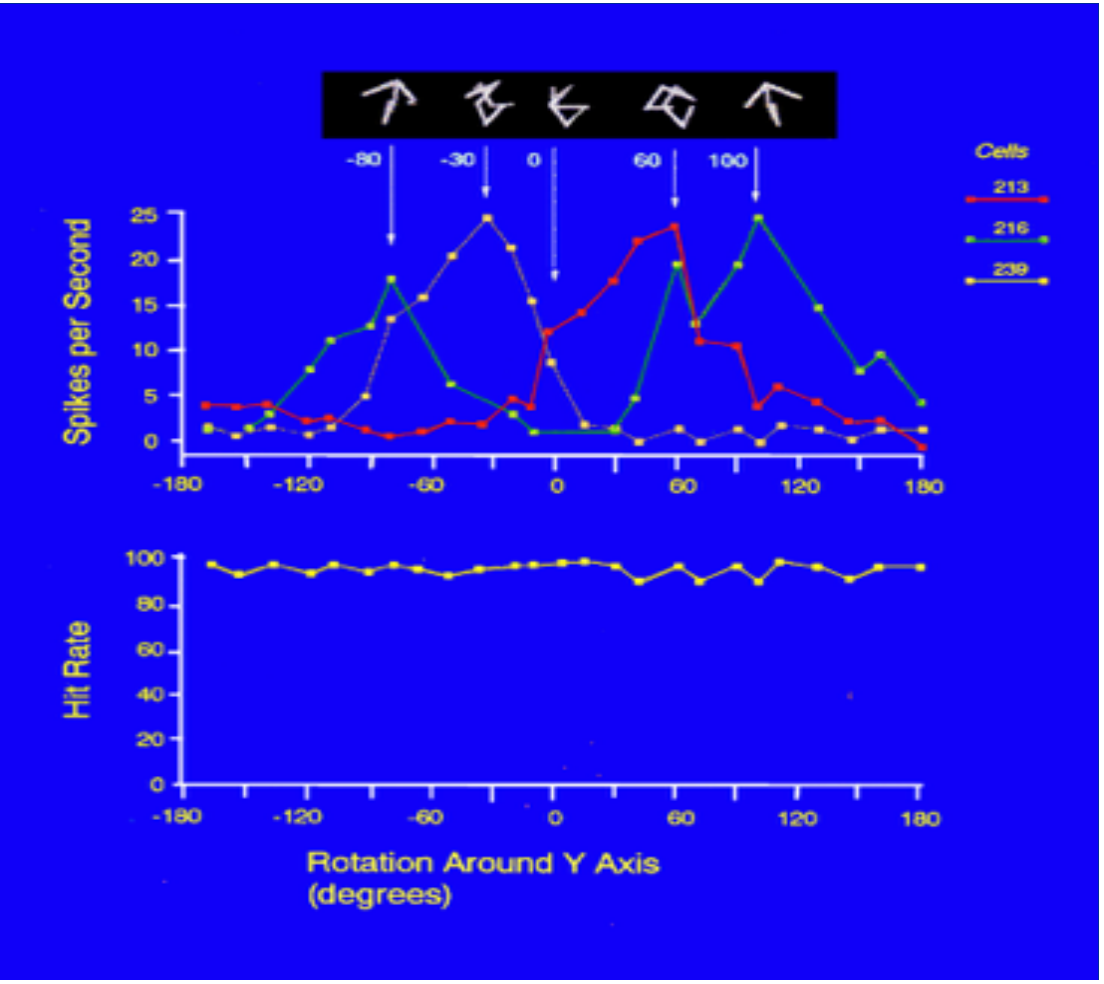
Of course learning is much more than memory but in this model the difference is between a Gaussian and a delta function

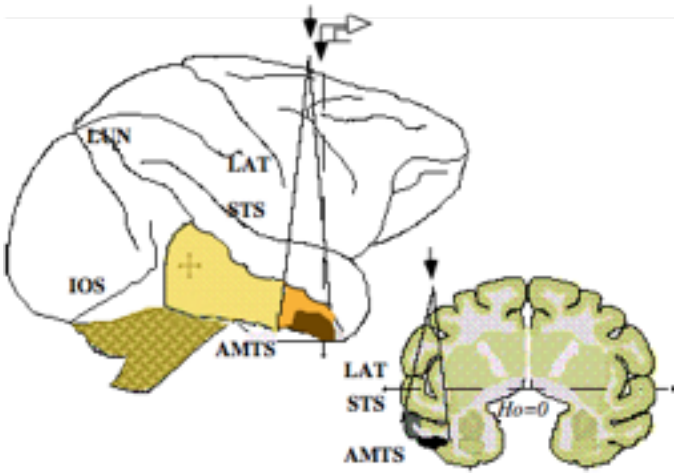


$$f(x) = \sum_i c_i G(x, x_i) = \sum_i c_i e^{-\frac{||x - x_i||^2}{2\sigma^2}}$$



Poggio, Edelman
***Nature*, 1990.**



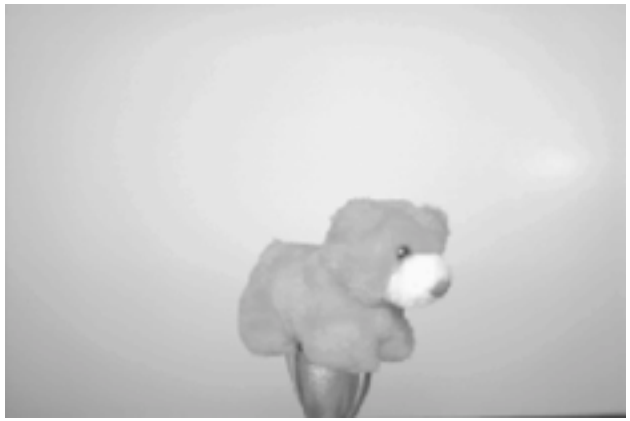


Logothetis, Pauls, and Poggio, 1995

Garfield

Image Analysis

⇒ **Bear (0° view)**



⇒ **Bear (45° view)**

Image Synthesis

UNCONVENTIONAL GRAPHICS

Θ = 0° view $\Rightarrow$

Θ = 45° view $\Rightarrow$

Hyperbf

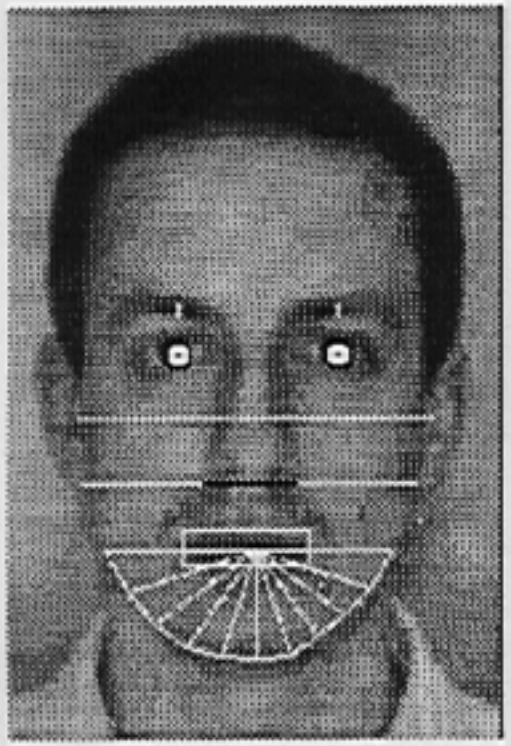
$$f^*(\mathbf{x}) = \sum_{\alpha=1}^{n} c_\alpha G(\|(\mathbf{x} - \mathbf{t}_\alpha)\|_W^2) + p(\mathbf{x}) \qquad (3)$$

where the parameters $\mathbf{t}_\alpha$, which we call "centers," and the coefficients c_α are unknown, and are in general many fewer than the data points ($n \leqslant N$). The norm is a *weighted norm*

$$\|(\mathbf{x} - \mathbf{t}_\alpha)\|_W^2 = (\mathbf{x} - \mathbf{t}_\alpha)^T W^T W (\mathbf{x} - \mathbf{t}_\alpha) \qquad (4)$$

Cartooon male

A toy problem: Gender Classification



1	pupil to eyebrows separation
2	pupil to nose vertical distance
3	pupil to mouth vertical distance
4	pupil to chin vertical distance
5	eyebrows thickness
6	nose width
7	mouth width
8	bizygomatic breadth
9	bigonial breadth
10-15	six chin radii
16	mouth height

Figure 1: Geometrical features (white) used in the face recognition experiments

Brunelli, Poggio ’91 (IRST, MIT)

An example: HyperBF and gender classification

Some of the geometrical feature (white) used in the gender classification experiments

HyperBF and gender classification

Typical stimuli used in the (informal!) psychophysical experiments of gender classification (about 90% correct)

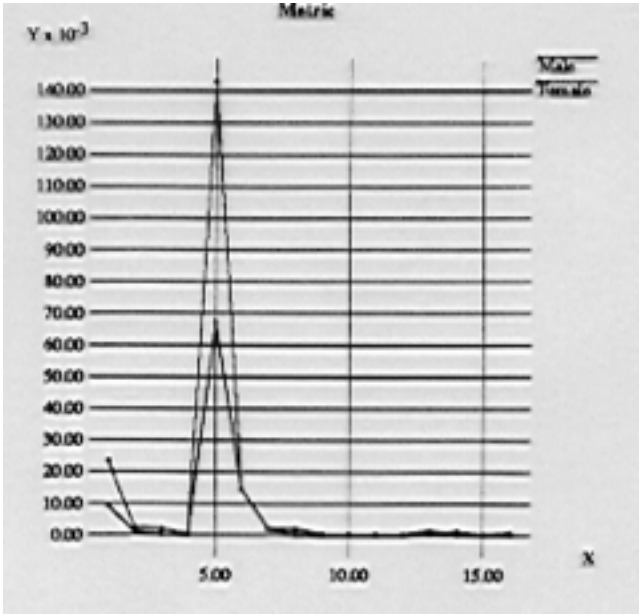
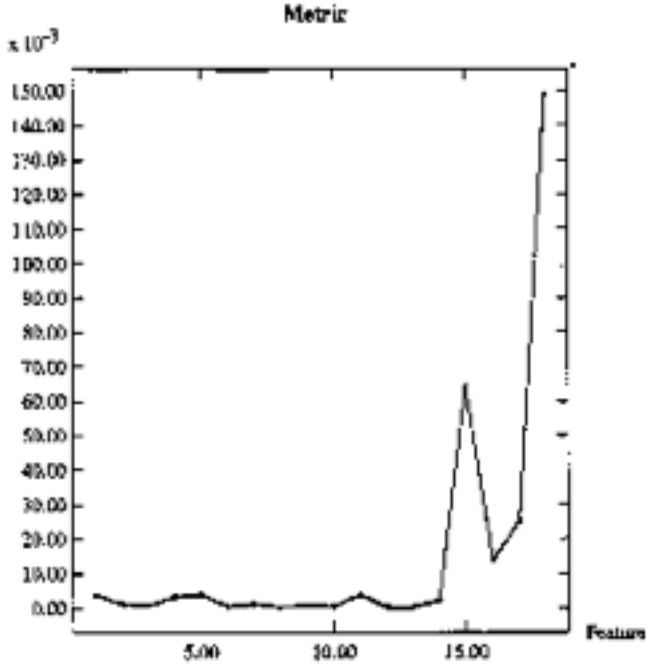


Figure 3: Feature weights for gender classification as computed by the HyperBF networks

An example: HyperBf and gender classification (Brunelli and Poggio, 1990)



Feature weights (i.e. the elemnts of the (diagonal) **W** as computed by the HyperBF Network for gender classification

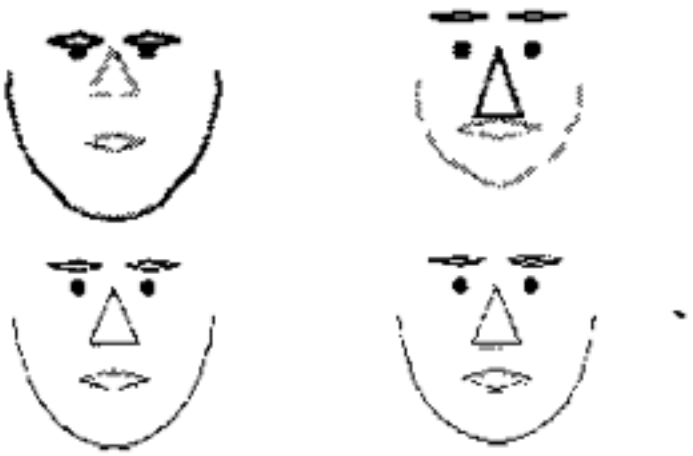
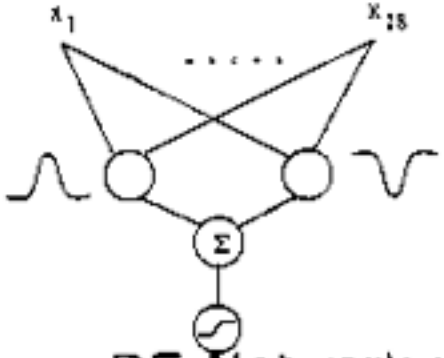


Figure 4: TOP. The male prototype (left) and the female prototype (left) as synthesized by the HyperBF Networks with movable coefficients, centers and metric. The darker the feature, the more important it is according to the corresponding entries in the diagonal metric W. BOTTOM. The average male face (left) and the average female face (right)



The HyperBF Network used for gender classification

- Data base of 200 images, 48 people
- Testing set (of "novel" people): about 88 percent correct classification

Radial Basis Functions and MLPs

Sigmoidal units are radial basis functions (for normalized inputs)

Since $||x - w||^2 = ||x||^2 + ||w||^2 - 2(x \cdot w)$

If $||x|| = 1$

$$(x \cdot w) = \frac{1 + ||w||^2 - ||x - w||^2}{2}$$

and thus $\sigma(w \cdot x + b)$ is a radial function

Consider the MLP units

$$\sigma(x \cdot w - \theta) = \frac{1}{1 + e^{-(x \cdot w - \theta)}}$$

Sigmoidal units are radial basis functions (for normalized inputs)

The corresponding radial function is

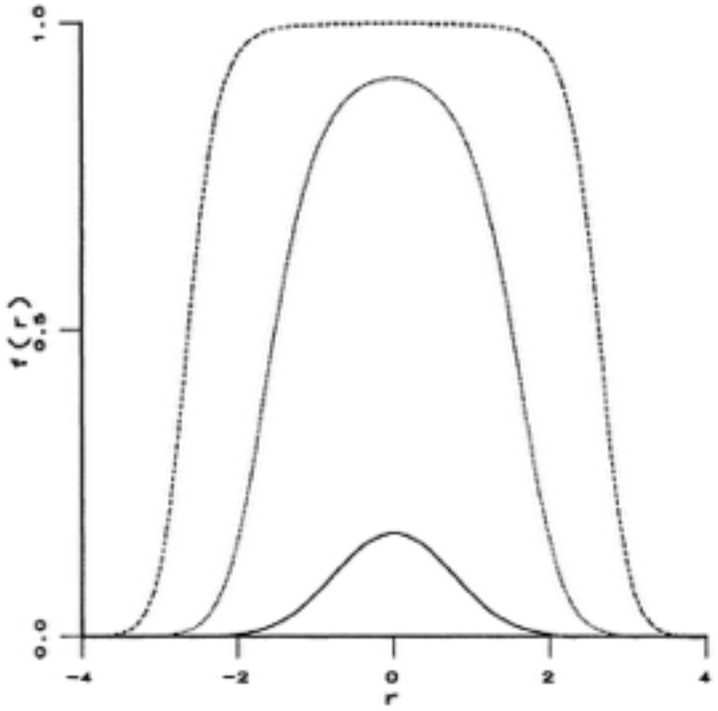
$$s\sigma(\mathbf{w}\cdot\mathbf{x}+\theta) = \frac{s}{1+\exp(-(\mathbf{w}\cdot\mathbf{x}+\theta))} \quad (s\in\mathbf{R},\ \mathbf{w}\in\mathbf{R}^d,\ \theta\in\mathbf{R})$$

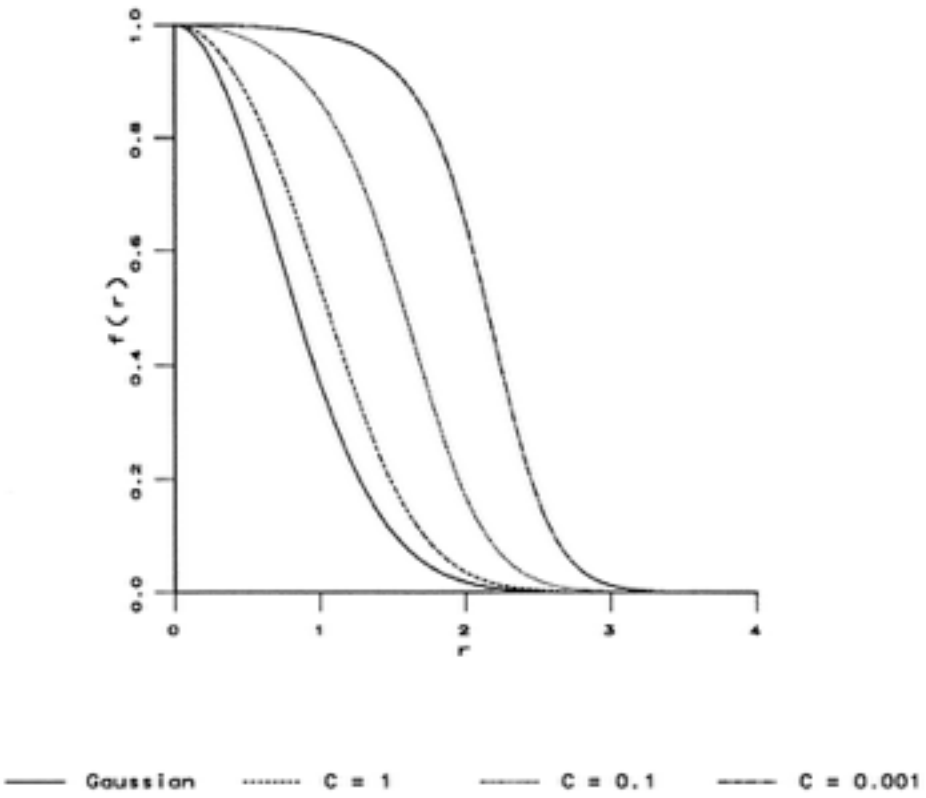
$$= \frac{s}{1+C(\lambda)\exp(\lambda\|\mathbf{x}-\mathbf{t}\|^2)} \quad (C(\lambda)>0)$$

where we have defined :

$$\mathbf{t}=\frac{\mathbf{w}}{2\lambda},\quad C(\lambda)=\exp(-(\theta+\lambda(1+\frac{\|\mathbf{w}\|^2}{4\lambda^2})))$$

Sigmoidal units are radial basis functions (for normalized inputs)





What if you want to approximate a continuous function?

Can a network learn to drive?

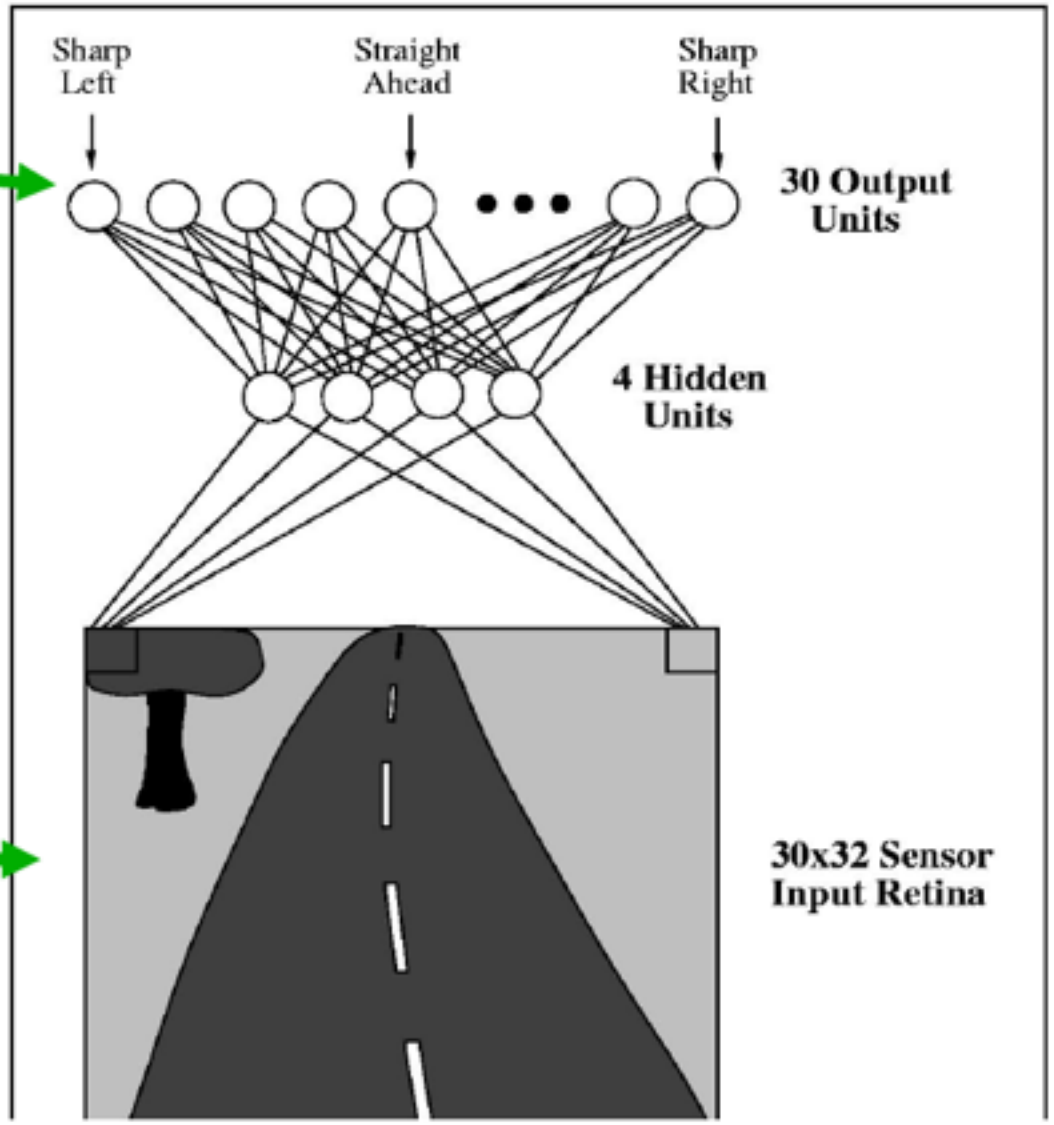
Example Network

Steering angle

Desired Output:

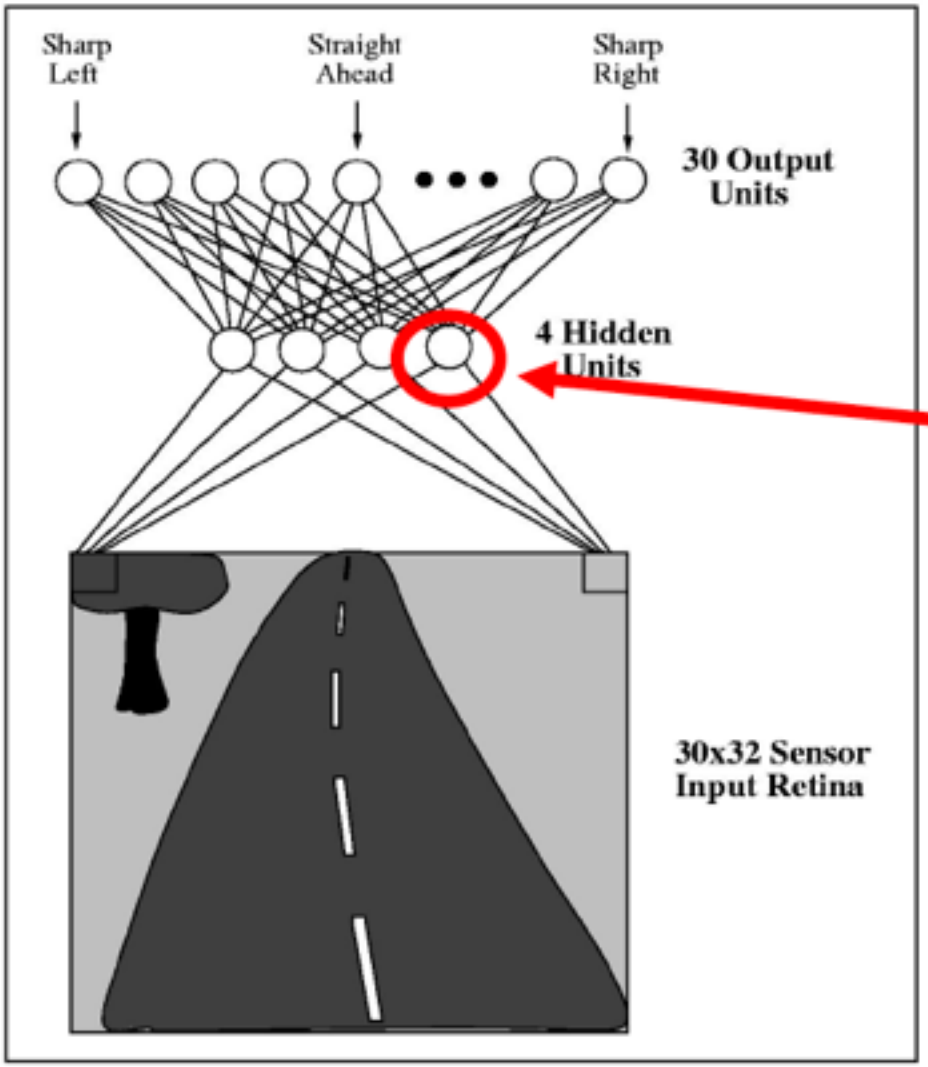
$\mathbf{d} = (d_1\ d_2 \ldots d_{30})$

Current image

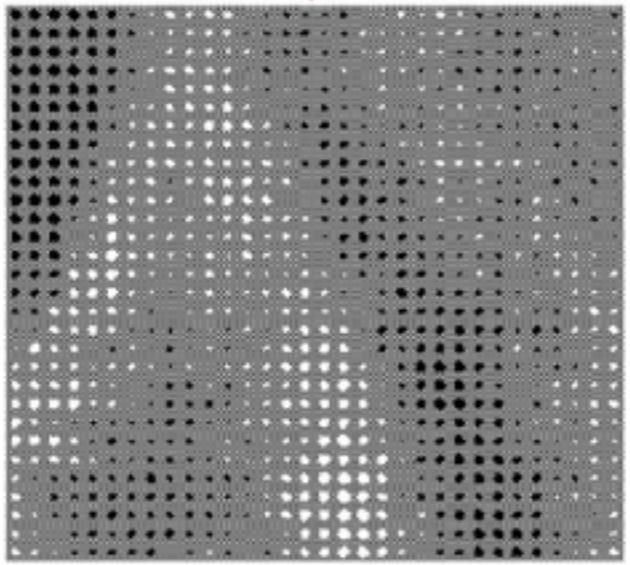


Input $\mathbf{u} = (u_1\ u_2 \ldots u_{960})$ = image pixels

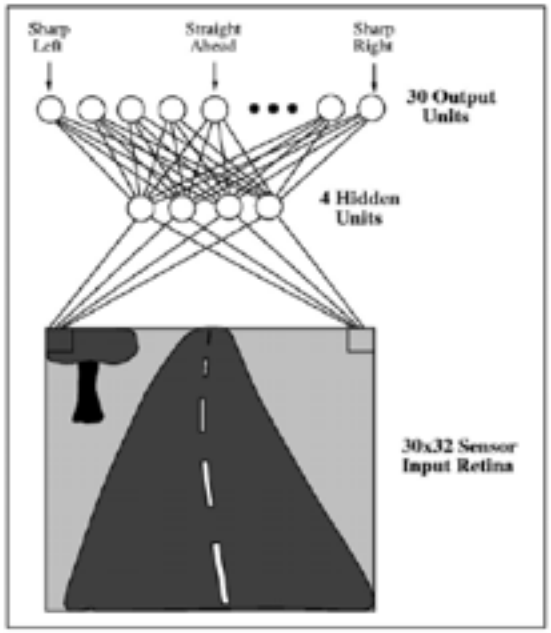
Learning to Drive using Backprop



One of the learned "road features" w_i



ALVINN (Autonomous Land Vehicle in a Neural Network)



CMU Navlab

Trained using human driver + camera images

After learning:

- Drove up to 70 mph on highway
- Up to 22 miles without intervention
- Drove cross-country largely autonomously

(Pomerleau, 1992)