9.54
class 9

Supervised learning

MLP and backpropagation
Hilbert and Rao
Neurons, synapses

Shimon Ullman + Tomaso Poggio

Danny Harari + Daneil Zysman + Darren Seibert

Center for Brains,
Minds & Machines

Example: representer theorem in the linear case

Math

Some simple linear algebra shows that

$$w^T = YX^T(XX^T)^{-1} = Y(X^TX)^{-1}X^T = CX^T$$

since $X^T(XX^T)^{-1} = (X^TX)^{-1}X^T$

Then

$$f(x) = w^Tx = CX^Tx = \sum_i^n c_i x_i^T x$$

We can compute C_n or w_n depending whether $n \leq p$.

The above result is the most basic form of the <u>Representer Theorem.</u>

Stability and (Tikhonov) Regularization

Math

Consider $f(x) = w^T x = \sum_{j=1}^{p} w^j x^j$, and $R(f) = w^T w$,

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$w^T = Y X^T (X X^T)^{-1}$$

$$\min_{f \in \mathcal{H}} \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \|f\|^2 \right\}$$

$$w^T = Y X^T (X X^T + \lambda I)^{-1}$$

$\lambda ||w||^2$ in the case of linear functions

From Linear to Nonparametric Models

Math

We can now consider a truly non parametric model

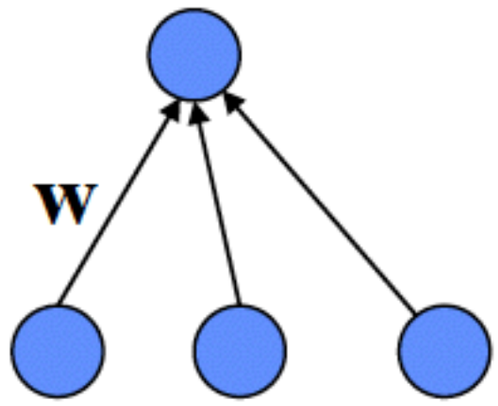
$$f(x) = \sum_{j \geq 1} w^j \Phi(x)^j = \sum_{i=1}^{n} \underbrace{K(x, x_i)}_{\sum_{j \geq 1} \Phi(x_i)^j \Phi(x)^j} c_i$$

We have

$$C_n = (\underbrace{X_n X_n^T}_{(X_n X_n^T)_{i,j} = x_i^T x_j} + \lambda n I)^{-1} Y_n \longrightarrow C_n = (\underbrace{K_n}_{(K_n)_{i,j} = K(x_i, x_j)} + \lambda n I)^{-1} Y_n$$

Sigmoidal networks ~ MLPs

$$Output \quad v = g(\mathbf{w}^T\mathbf{u}) = g(\sum_i w_i u_i)$$

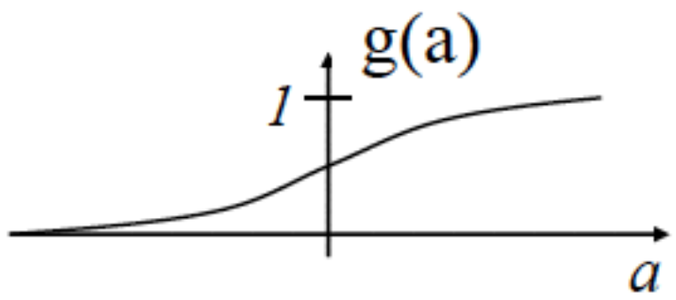


Input nodes

$\mathbf{u} = (u_1 \quad u_2 \quad u_3)^T$

Sigmoid function:

$$g(a) = \frac{1}{1 + e^{-\beta a}}$$



Sigmoid is a non-linear "squashing" function: Squashes input to be between 0 and 1. The parameter β controls the slope.

Gradient Descent

Given training examples $(\mathbf{u}^m, d^m)$ (m = 1, …, N), define a sum of squared output errors function (also called a cost function or "energy" function)

$$E(\mathbf{w}) = \frac{1}{2}\sum_m (d^m - v^m)^2$$

where $v^m = g(\mathbf{w}^T \mathbf{u}^m)$

Thus $$Y - MX = 0$$

More in general look for M such that

$$min||Y - MX||^2$$

The solution is given by putting the gradient to zero

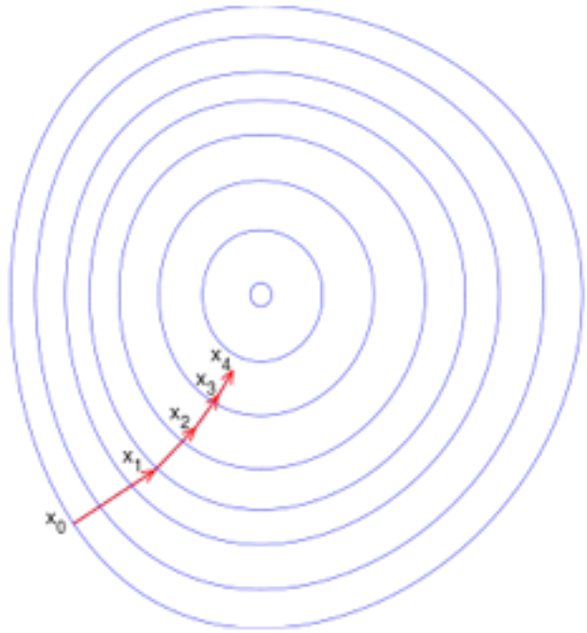
$$\nabla V(M) = 2(Y - MX)X^T = 0 \text{ yielding } YX^T = MXX^T \text{ that is } M = YX^T(XX^T)^{-1}$$

which is the same we had derived earlier…

How could minimization done in general, in practice, by the brain?
Probably not by analytic solution….

The gradient offers a general way to compute a solution to a minimization problem

$$\frac{dM}{dt} = -\gamma \nabla V(M)$$



finds the elements of M which correspond to $minV(M)$

As an example let us look again at

$$min||Y - MX||^2$$

Using $$\nabla V(M) = 2(Y - MX)X^T$$

Let us make the example more specific. Assume that y_i are scalar

Then $M = w^T$ and

$min_{m_{i,j}} ||MX - Y||^2$ becomes $min_{w \in R^d} \frac{1}{n} \sum_{1=1}^{n} (y_i - w^T x_i)^2$

yielding

$$\nabla V(M) = \nabla V(w_i^T) = \frac{2}{n} \sum_{i=1}^{n} (y_i - w^T x_i) x_i^T$$

and thus

$$\frac{dw^T}{dt} = -\gamma_t \sum_{i=1}^{n} (y_i - w_t^T x_i) x_i^T$$

Discretizing time in

$$\frac{dw^T}{dt} = -\gamma_t \sum_{i=1}^{n} (y_i - w_t^T x_i) x_i^T$$

we obtain

$$w_{t+1}^T = w_t^T - \gamma_t \sum_{i=1}^{n} (y_i - w_t^T x_i) x_i^T$$

Gradient descent has several nice properties but it is still not “biological”…

$$w_{t+1}^T = w_t^T - \gamma_t \sum_{i=1}^{n} (y_i - w_t^T x_i) x_i^T$$

can be written as

$$\frac{dw^T}{dt} = -\gamma_t \sum_{i=1}^{n} \nabla V_i(w)$$

Stochastic gradient descent is…

$$\frac{dw^T}{dt} = -\gamma_t \nabla V_i(w) \ , \ i = 1, \cdots , n$$

I use now E(w)=V(w)

Gradient Descent

- Would like to change $\mathbf{w}$ so that $E(\mathbf{w})$ is minimized
  - ➪ Gradient Descent: Change $\mathbf{w}$ in proportion to $-dE/d\mathbf{w}$ (why?)

$$\mathbf{w} \rightarrow \mathbf{w} - \varepsilon \frac{dE}{d\mathbf{w}}$$

$$\frac{dE}{d\mathbf{w}} = -\sum_m (d^m - v^m) \frac{dv^m}{d\mathbf{w}} = -\sum_m (d^m - v^m) g'(\mathbf{w}^T \mathbf{u}^m) \mathbf{u}^m$$

Derivative of sigmoid

Stochastic Gradient Descent

- What if the inputs only arrive one-by-one?
- Stochastic gradient descent approximates sum over all inputs with an "on-line" running sum:

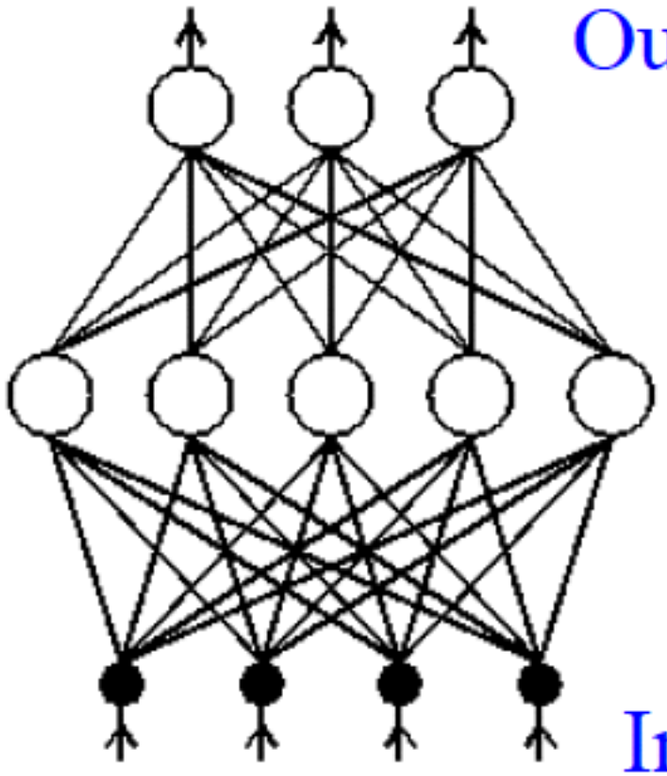
$$\mathbf{w} \rightarrow \mathbf{w} - \varepsilon \frac{dE_1}{d\mathbf{w}}$$

$$\frac{dE_1}{d\mathbf{w}} = -\underbrace{(d^m - v^m)}_{\text{delta = error}} g'(\mathbf{w}^T \mathbf{u}^m)\mathbf{u}^m$$

Also known as the "delta rule" or "LMS (least mean square) rule"

Stochastic Gradient Descent: chain rule ~ Backpropagation

- What if we have multiple layers?



Output $\mathbf{v} = (v_1\ v_2\ \ldots\ v_J)^T$; Desired = $\mathbf{d}$

Delta rule can be used to adapt these weights

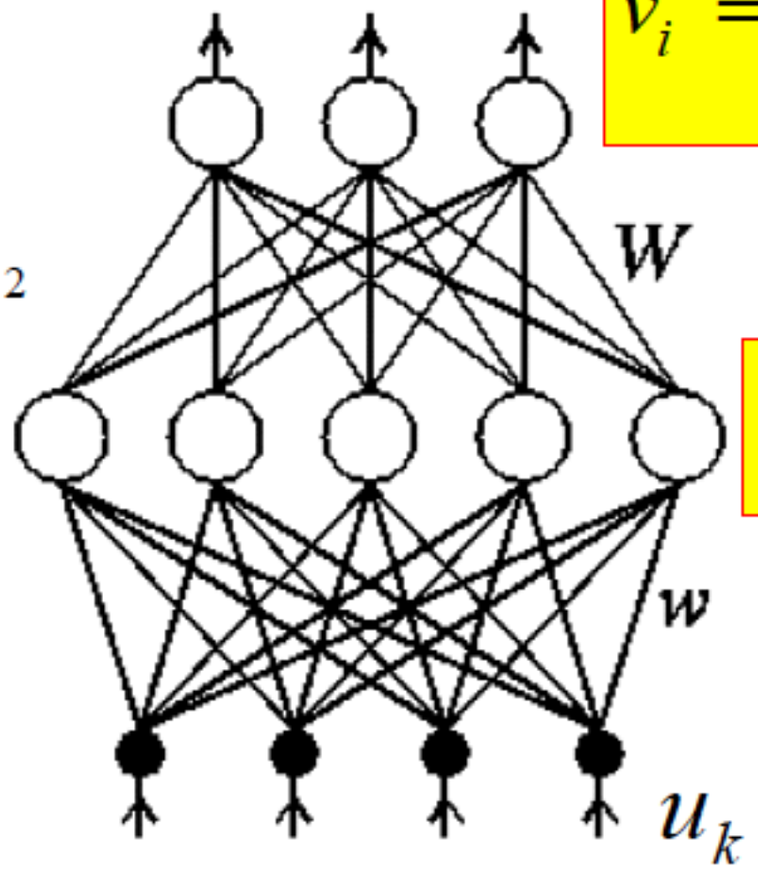
How do we adapt these?

Input $\mathbf{u} = (u_1\ u_2\ \ldots\ u_K)^T$

Backpropagation: Uppermost layer (delta rule)

$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

$$v_i = g(\sum_j W_{ji} x_j)$$



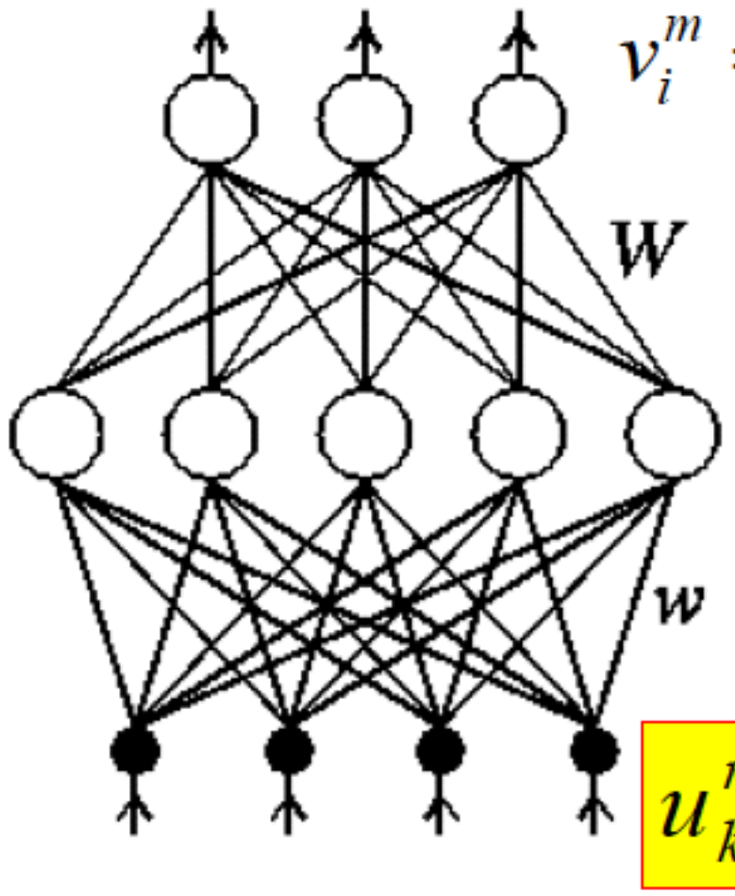
Learning rule for hidden-output weights **W**:

$$W_{ji} \rightarrow W_{ji} - \varepsilon \frac{dE}{dW_{ji}} \qquad \{\textit{gradient descent}\}$$

$$\frac{dE}{dW_{ji}} = -(d_i - v_i) g'(\sum_j W_{ji} x_j) x_j \qquad \{\textit{delta rule}\}$$

Backpropagation: Inner layer (chain rule)

$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$



$$v_i^m = g(\sum_j W_{ji} x_j)$$

$$x_j^m = g(\sum_k w_{kj} u_k^m)$$

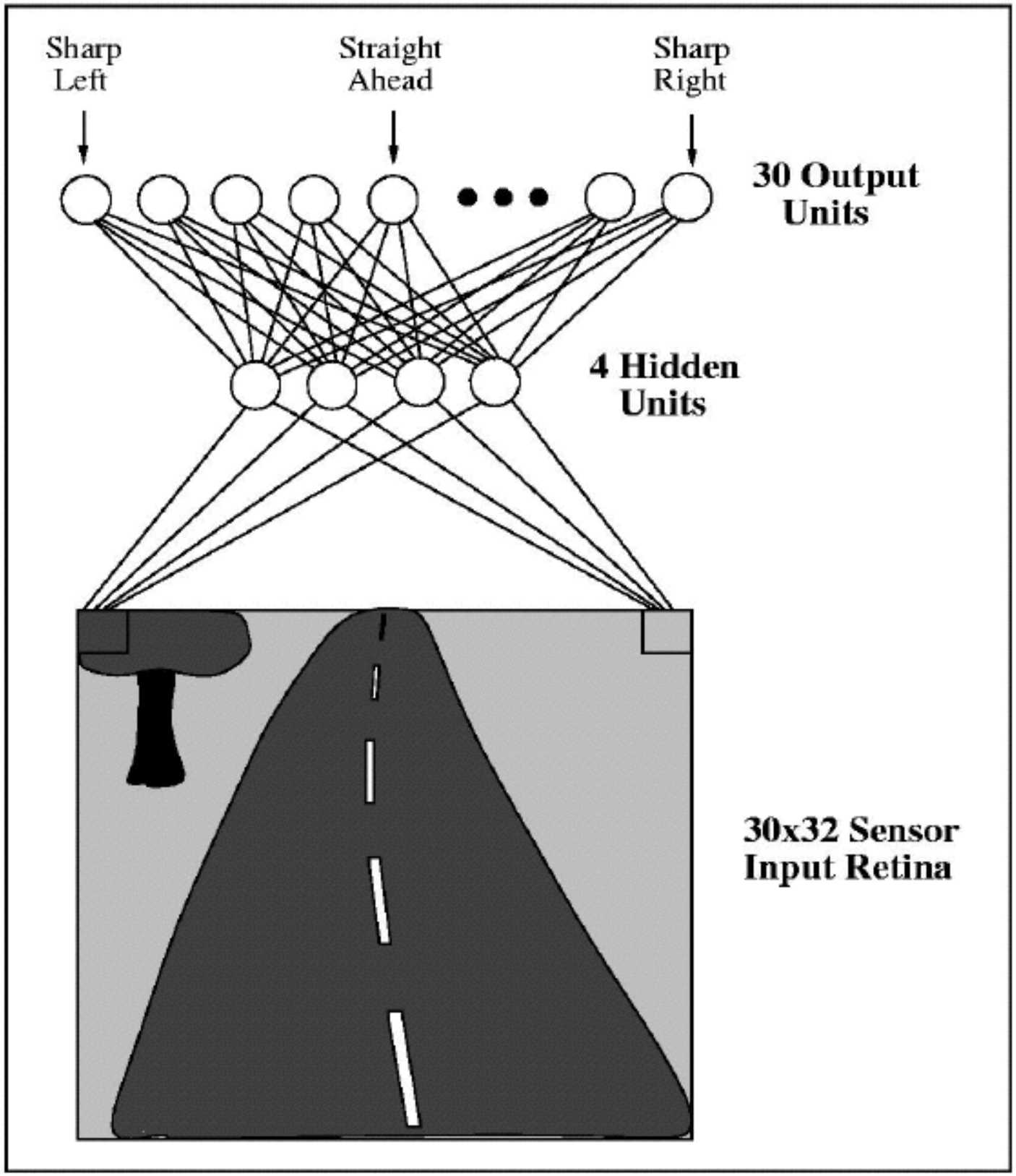
Learning rule for input-hidden weights **w**:

$$w_{kj} \rightarrow w_{kj} - \varepsilon \frac{dE}{dw_{kj}} \quad \text{But}: \frac{dE}{dw_{kj}} = \frac{dE}{dx_j} \cdot \frac{dx_j}{dw_{kj}} \quad \{\textit{chain rule}\}$$

$$\frac{dE}{dw_{kj}} = \left[-\sum_{m,i} (d_i^m - v_i^m) g'(\sum_j W_{ji} x_j^m) W_{ji} \right] \cdot \left[g'(\sum_k w_{kj} u_k^m) u_k^m \right]$$

http://en.wikipedia.org/wiki/Automatic_differentiation

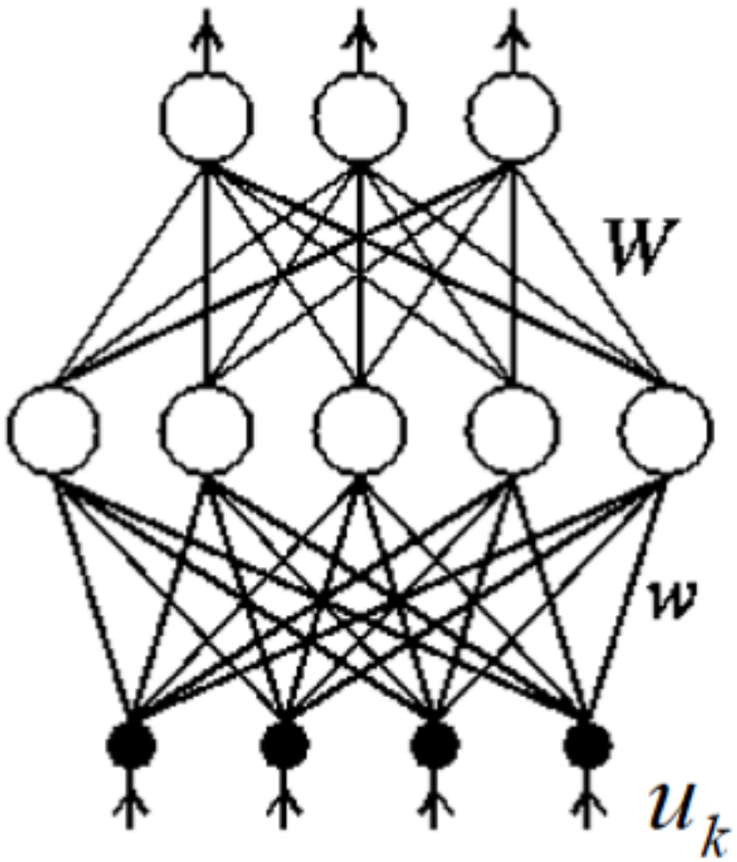
Example Network



(Pomerleau, 1992)

Training the network using backprop

$$v_i = g(\sum_j W_{ji} g(\sum_k w_{kj} u_k))$$



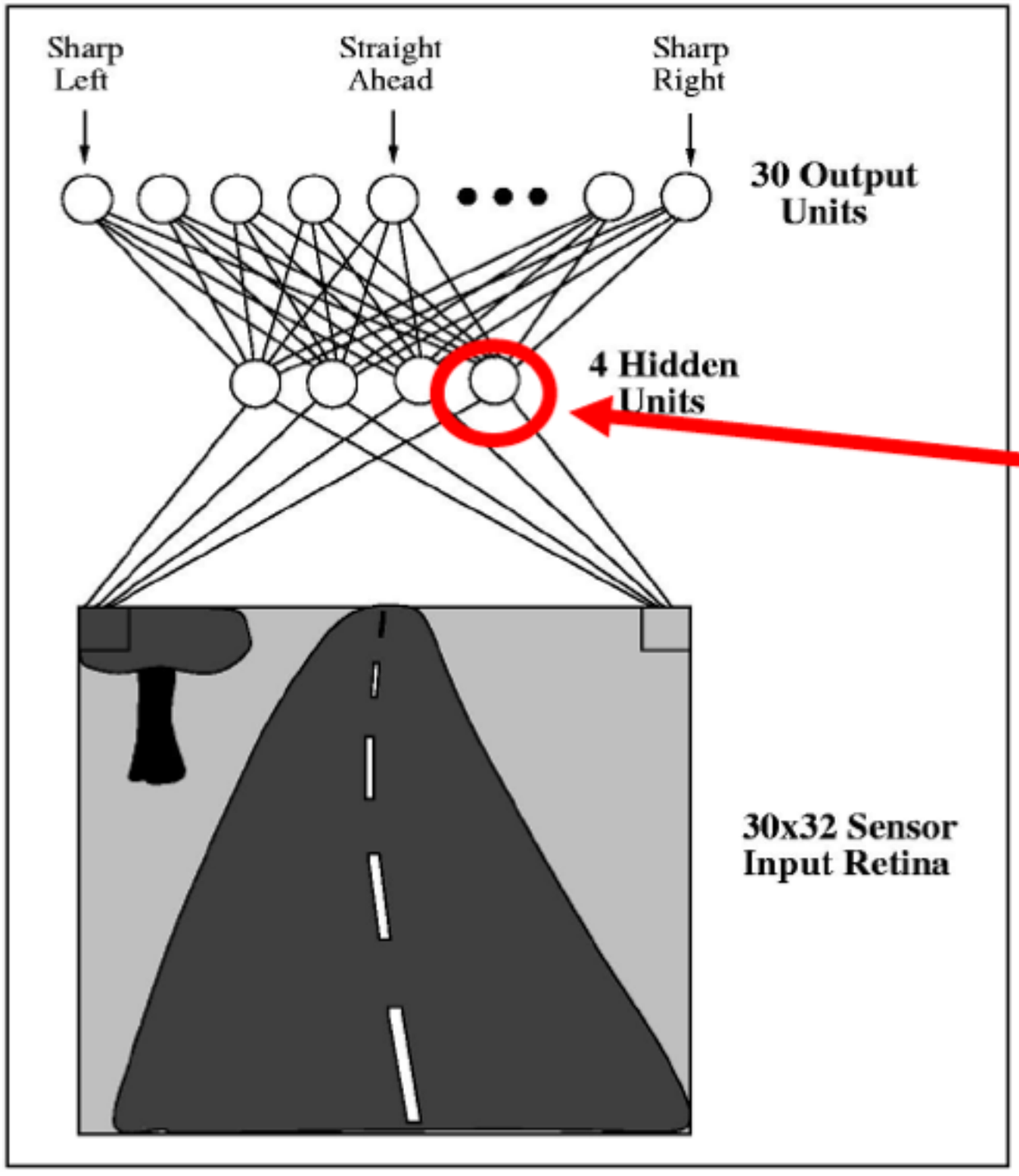
Start with random weights **W**, **w**

Given input **u**, network produces output **v**

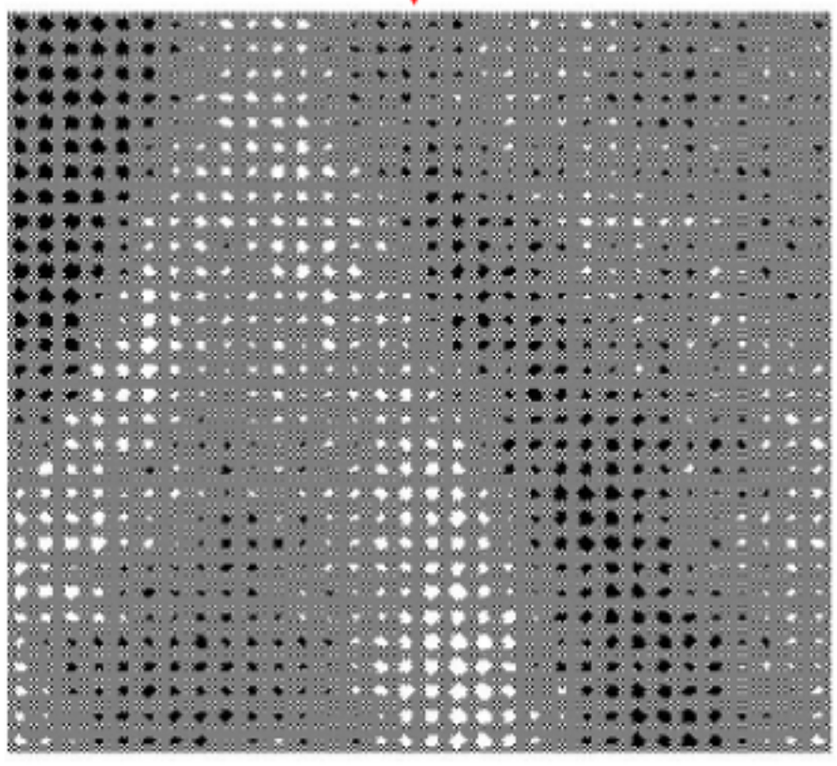
Use backprop to learn **W** and **w** that minimize total error over all output units (labeled *i*):

$$E(\mathbf{W},\mathbf{w}) = \frac{1}{2}\sum_i (d_i - v_i)^2$$

Learning to Drive using Backprop



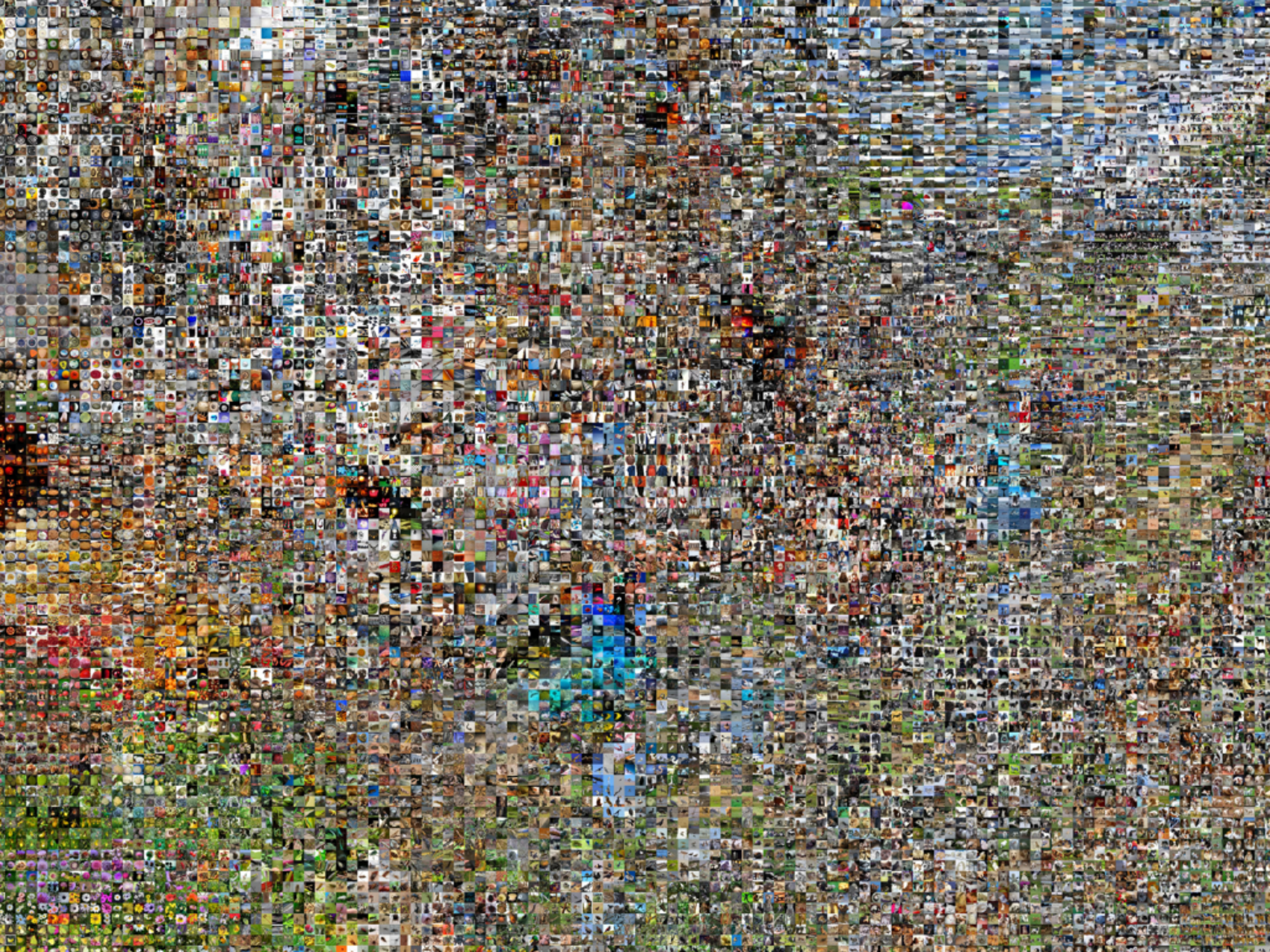
One of the learned "road features" w_i



Convolutional networks trained with backpropagation

Human-like performance!

http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

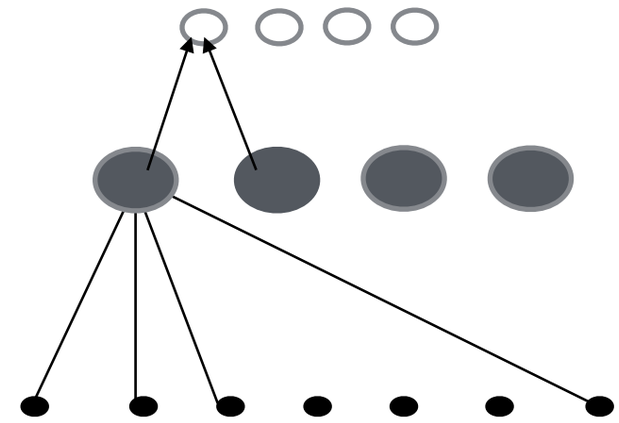


What are MLPs doing?

Nonlinear learning (RBFs)

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

$$f(x) = \sum_{k=1}^{N} e^{\frac{-||x-x_k||^2}{2\sigma^2}}$$



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

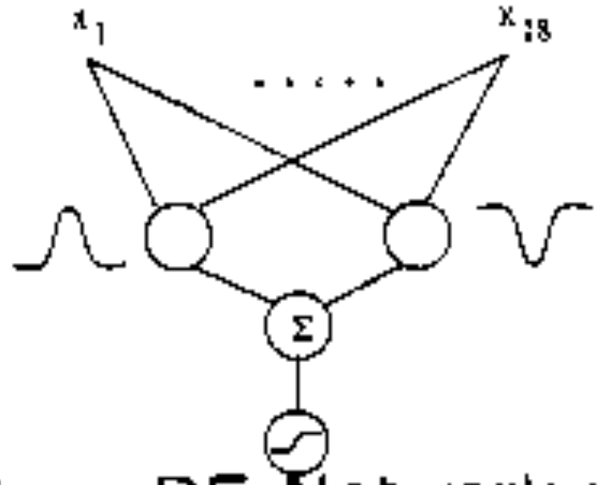
- RBF is a rewriting of our polynomial (infinite radius of convergence)

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

Nonlinear learning (HyperBFs)

$$f(x) = \sum_{k=1}^{M} c_k e^{-||x-t_k||_W^2} \text{ with } ||z||_W^2 = z^T W^T W z$$

where t_k are called centers. There are 3 sets of parameters: t_k, c_k, W. Usually $M << N$



The HyperBF Network used for gender classification

- Data base of 200 images, 48 people

- Testing set (of "novel" people): about 88 percent correct classification

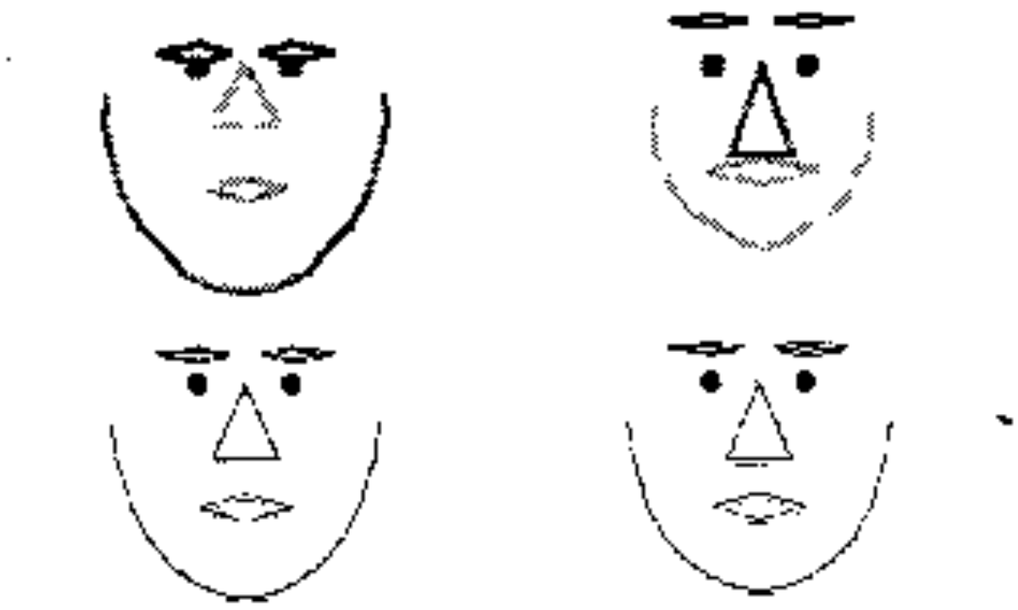


Figure 4: TOP. The male prototype (left) and the female prototype (left) as synthesized by the HyperBF Networks with movable coefficients, centers and metric. The darker the feature, the more important it is according to the corresponding entries in the diagonal metric W. BOTTOM. The average male face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

Non-standard remarks: Backpropagation

- Networks with one hidden layer and normalized inputs: equivalence MLPs and HyperBFs

- Key “original” feature of MLPs is stochastic gradient descent (which requires chain rule for more than one hidden layer); SGD is standard online learning algorithm; SGD saves computation and memory

- HyperBFs need much fewer centers than RBF: thus the equivalent MLPs save memory

- Disadvantages:
  - non-convex optimization problem, local minima
  - generalization not guaranteed (stability cannot be proven?)

- Important open question: why do MLPs with more than one hidden layer (nowdays called DLN) seem to perform better? What is their representational power?

Batch learning and Online Learning

A batch learning algorithm maps a training set to a function in the hypotheses space, that is

$$f_n = A(S_n) \in \mathcal{H},$$

and is typically assumed to be symmetric, that is, invariant to permutations in the training set.

An online learning algorithm is defined recursively as

$$f_0 = 0 \qquad\qquad f_{n+1} = A(f_n, z_n).$$

Batch learning and Online Learning

The prototype example of batch learning algorithm is empirical risk minimization, defined by the variational problem

$$\min_{f \in \mathbb{H}} I_n(f),$$

where $I_n(f) = \mathbb{E}_n V(f, z)$ is the empirical average of the loss on the sample

The prototype example of online learning algorithm is stochastic gradient descent, defined by

$$f_{n+1} = f_n - \gamma_n \nabla V(f_n, z_n)$$

Stochastic Gradient Descent

Under standard conditions SDG is stable and generalizes; these conditions are probably not satisfied by MLPs (loss is not convex; H may not be compact…)

Multiple layers and Hilbert's 13th problem

- Hilbert's 13th problem conjectures that there are functions of 3 variables that cannot be represented as compositions of functions of one variable.

- Kolmogorov proved that Hilbert's conjecture is wrong: he proved that every continuous function of several variables can be represented as a superposition of continuous functions of one variable and the operation of addition.

- Unfortunately, the representing functions are pretty bad, they are at best only continuous, and they cannot be chosen to be differentiable.

Multiple layers and Hilbert's 13th problem

Theorem 1.1. (Kolmogorov 1957). *There exist fixed increasing continuous functions $h_{pq}(x)$, on $I = [0, 1]$ so that each continuous function f on I^n can be written in the form*

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} g_q\left(\sum_{p=1}^{n} h_{pq}(x_p)\right),$$

where g_q are properly chosen continuous functions of one variable.

This should be compared with

$$f(x_1, \cdots, x_n) = \sum_q w_q \sigma_q\left(\sum_i w_i^q x_i + b^q\right)$$

Multiple layers and Hilbert's 13th problem

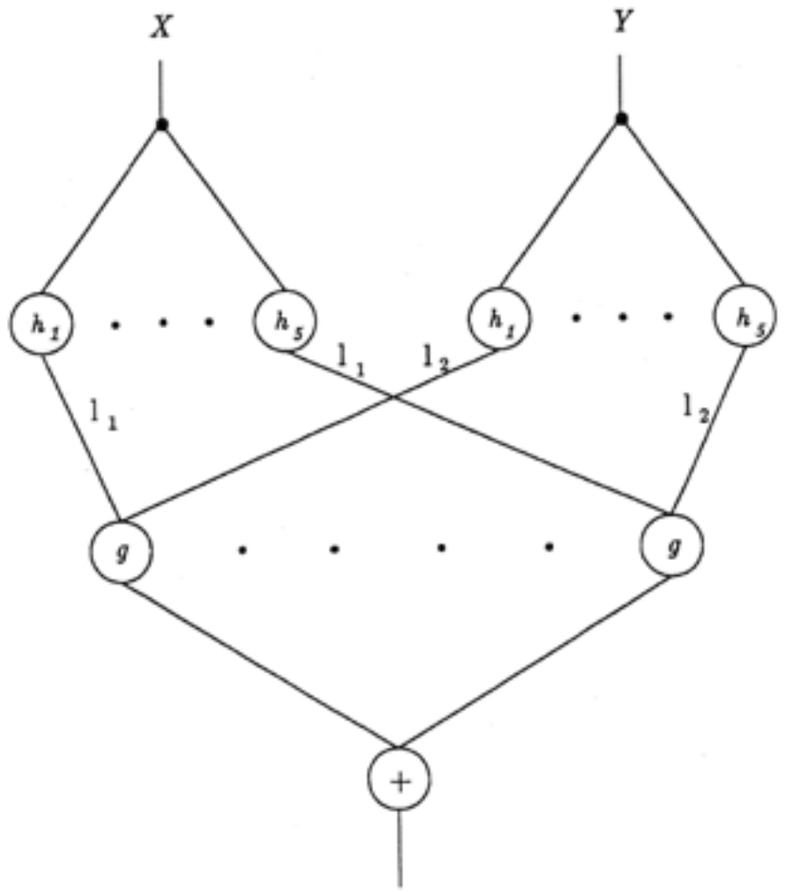


Figure 1: The network representation of an improved version of Kolmogorov's theorem, due to Kahane (1975). The figure shows the case of a bivariate function. The Kahane's representation formula is $f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} g[\sum_{p=1}^{n} l_p h_q(x_p)]$ where h_q are strictly monotonic functions and l_p are strictly positive constants smaller than 1.

Multiple layers and Hilbert's 13th problem

Problem with Kolmogorov solution:

The functions h are independent of f but the functions g depend on it and are as complex as f and possibly discontinuous. In fact

Theorem 2.1. (Vitushkin 1954). *There are $r (r = 1, 2, \ldots)$ times continuously differentiable functions of $n \geq 2$ variables, not representable by superposition of r times continuously differentiable functions of less than n variables; there are r times continuously differentiable functions of two variables that are not representable by sums and continuously differentiable functions of one variable.*

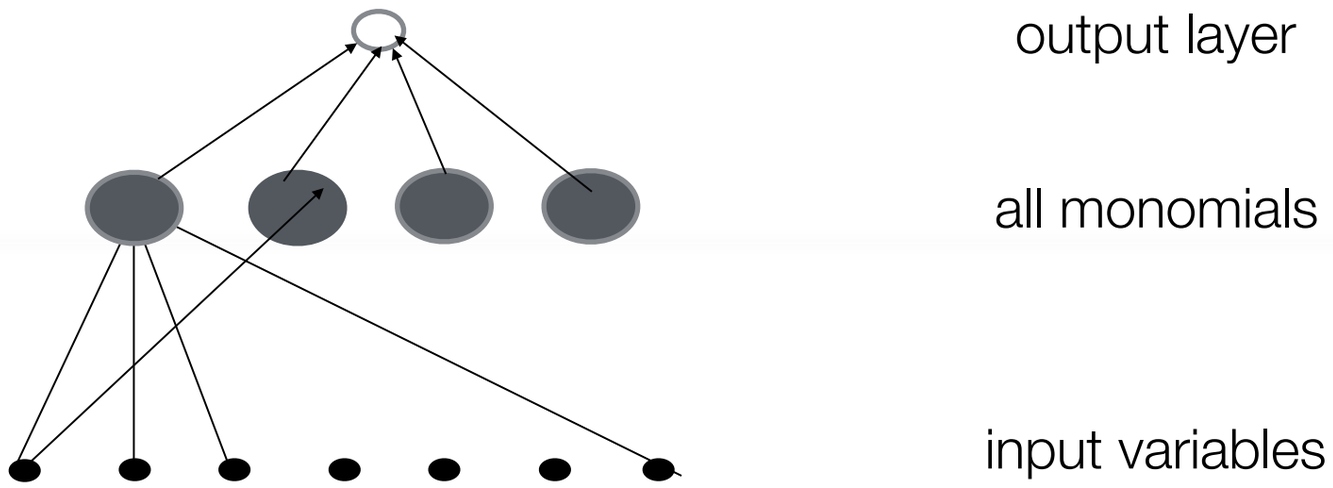
Plan

- One-hidden layer nets are universal; multiple layers can be equivalent
- For normalized inputs Gaussian units and sigmoidal units can be equivalent
- In particular, MLPs are equivalent to HyperBF; HyperBF need much fewer units than RBFs
- With SGD HyperBFs (and therefore MLPs) can be trained using much less memory than RBFs (which need to keep in memory all labeled examples vs the actual centers)
- Instead of storing 1M of 256x256 pixel images —> 2 10^11 numbers the Imagenet DLN has 6 10^7 parameters: this is a factor 2000 less memory
- Why multiple hidden layers instead on one? Intuition from analogy of HyperBF with look-up tables: hierarchical look-up tables, KD trees/balls,…

One-hidden layer nets are universal

- We know (theorems exist) that Gaussian RBFs can approximate arbitrarily well any continuous function

- An insight is provided by Weierstrass theorem rewritten here in terms of networks



Multiple layers can be equivalent

- A fixed sigmoidal can be used in multiple layers: the network is universal

- I do not prove "theorem" above. An insight is provided by the following lemma (remember: sigmoids and Gaussian are analytic and can be represented/defined in terms of their power series)

  Lemma *A fixed power can be used in a multiple layers net to represent exactly any polynomial (1-hidden layer net)*

Sketch proof: define $(\cdot)^2 = \Downarrow$ *, then every monomial can be obtained from it*

$$x^2 = \begin{array}{c} x \\ \Downarrow \end{array} \qquad 2xy = \begin{array}{ccc} x+y & & \\ \Downarrow & x & y \\ & \Downarrow & \Downarrow \end{array} \qquad xyz = \begin{array}{ccc} xy+z & & \\ \Downarrow & xy & z \\ & \Downarrow & \Downarrow \end{array}$$

For normalized inputs linear combinations of Gaussian units and sigmoidal units are equivalent

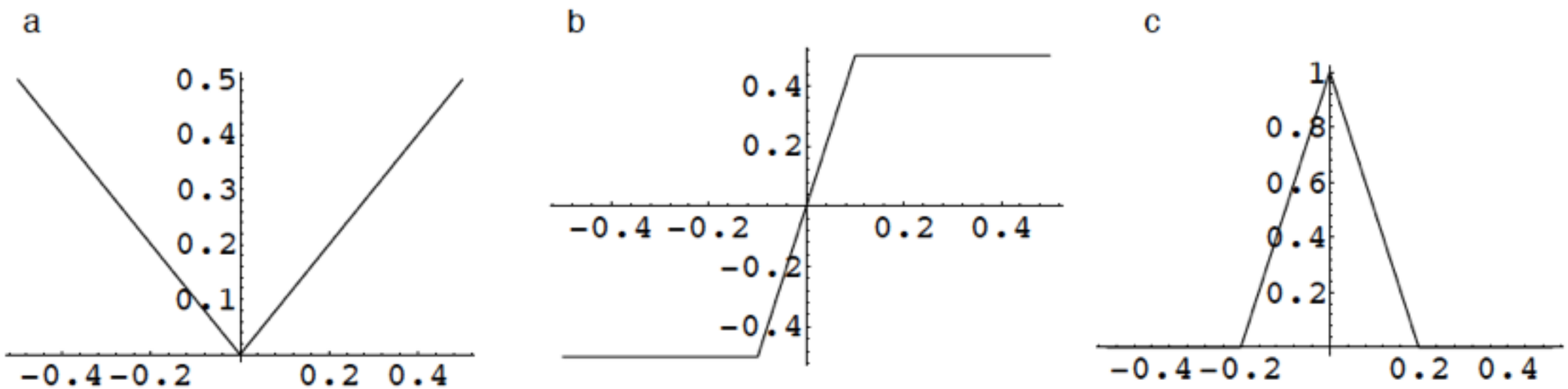


Figure 5: a) Absolute value basis function, $|x|$, b) "Sigmoidal" basis function $\sigma_L(x)$ c) Gaussian-like basis function $g_L(x)$

In particular, MLPs are equivalent to HyperBF; HyperBF need much fewer units than RBFs

$$f(x) = \sum_{i=1}^{N} c_i G(x, x_i) \qquad \text{vs.} \qquad f(x) = \sum_{i=1}^{K} c_i G(x, t_i), \;\; K << N$$

$$\frac{\partial H}{\partial c_\alpha} = -2 \sum_{i=1}^{N} \Delta_i h(\|\mathbf{x}_i - \mathbf{t}_\alpha\|^2) \;\; ,$$

$$\frac{\partial H}{\partial \mathbf{t}_\alpha} = 4 c_\alpha \sum_{i=1}^{N} \Delta_i h'(\|\mathbf{x}_i - \mathbf{t}_\alpha\|^2)(\mathbf{x}_i - \mathbf{t}_\alpha)$$

The use of prototypes by HyperBFs suggest that, in a sense, HyperBFs networks are an extension of massive template matchers or look-up tables: if memory is cheap, look-up tables are a good starting point.

Why multiple hidden layers instead on one? Intuition from analogy of HyperBF with look-up tables: hierarchical look-up tables, KD trees/balls, hierarchical VQ…

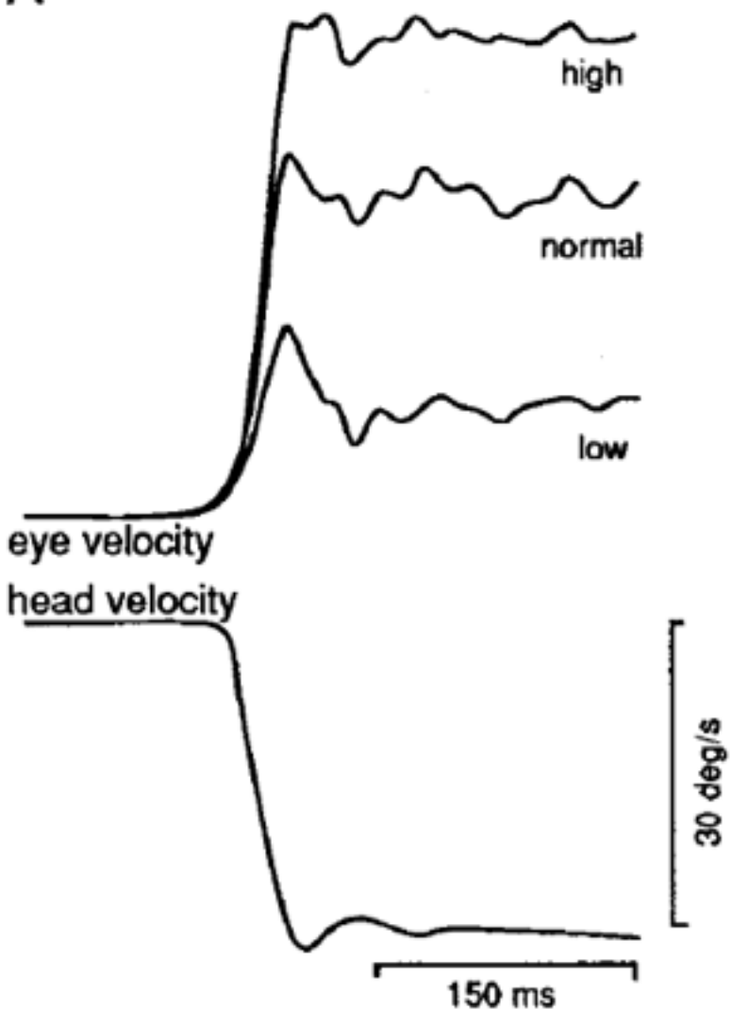
- RBFs can be regarded in the limit as look-up tables
- HyperBFs can be regarded as RBFs where the centers are learned from a kind of supervised k-means (from gradient equations)
- Hierarchical VQ can save a lot of memory —> this corresponds to factorization of Gaussian centers into lower dimensional centers which is what DLPs do when they reduce number of output units from layer to layer

An interesting metaphor:
DLN as memories

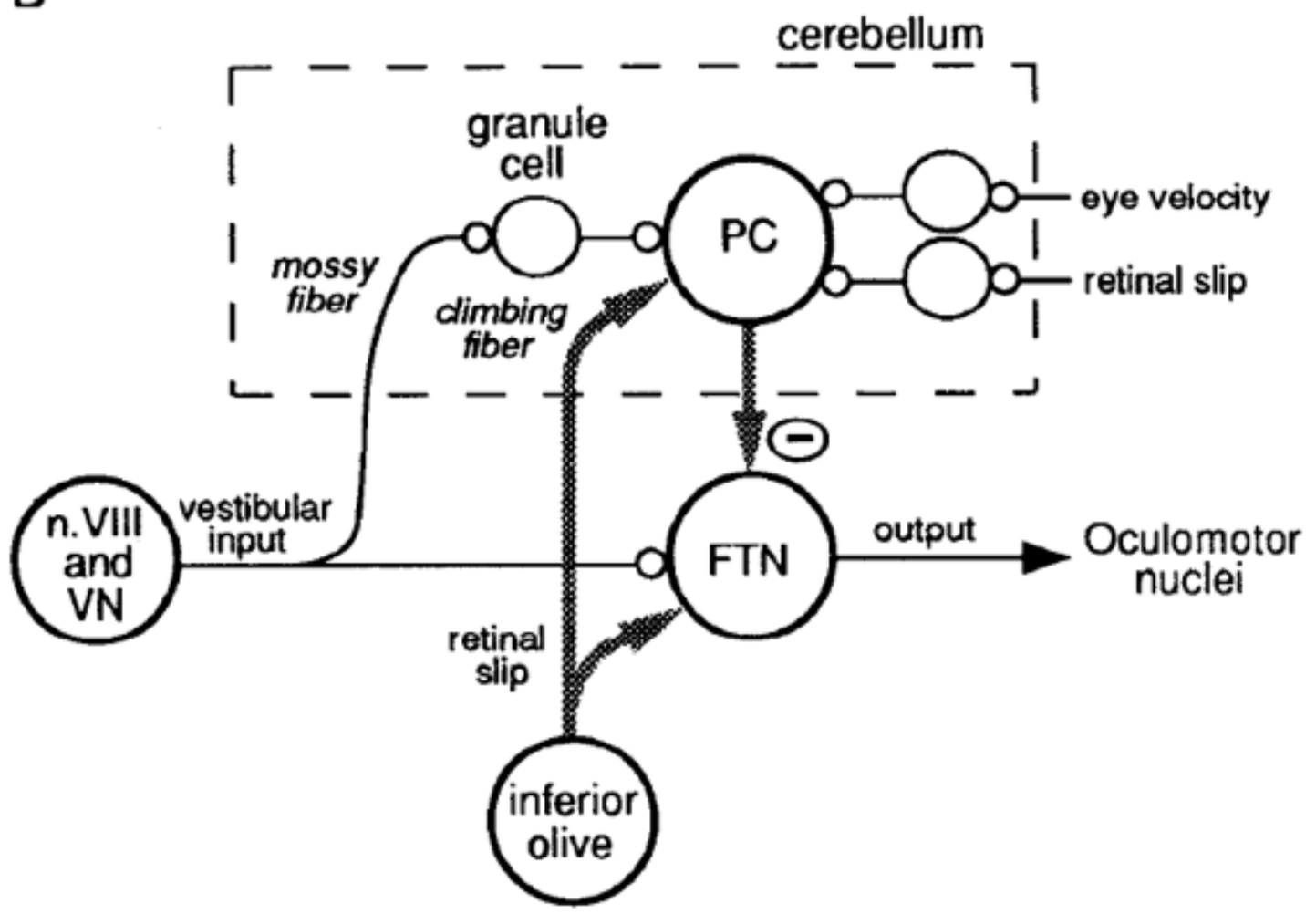
- Hierarchical VQ: equivalent to factorization of Gaussians in layers? see old memo with Federico….bin# = N^d but factorizing Gaussians I have bin# = N d (because of radial)

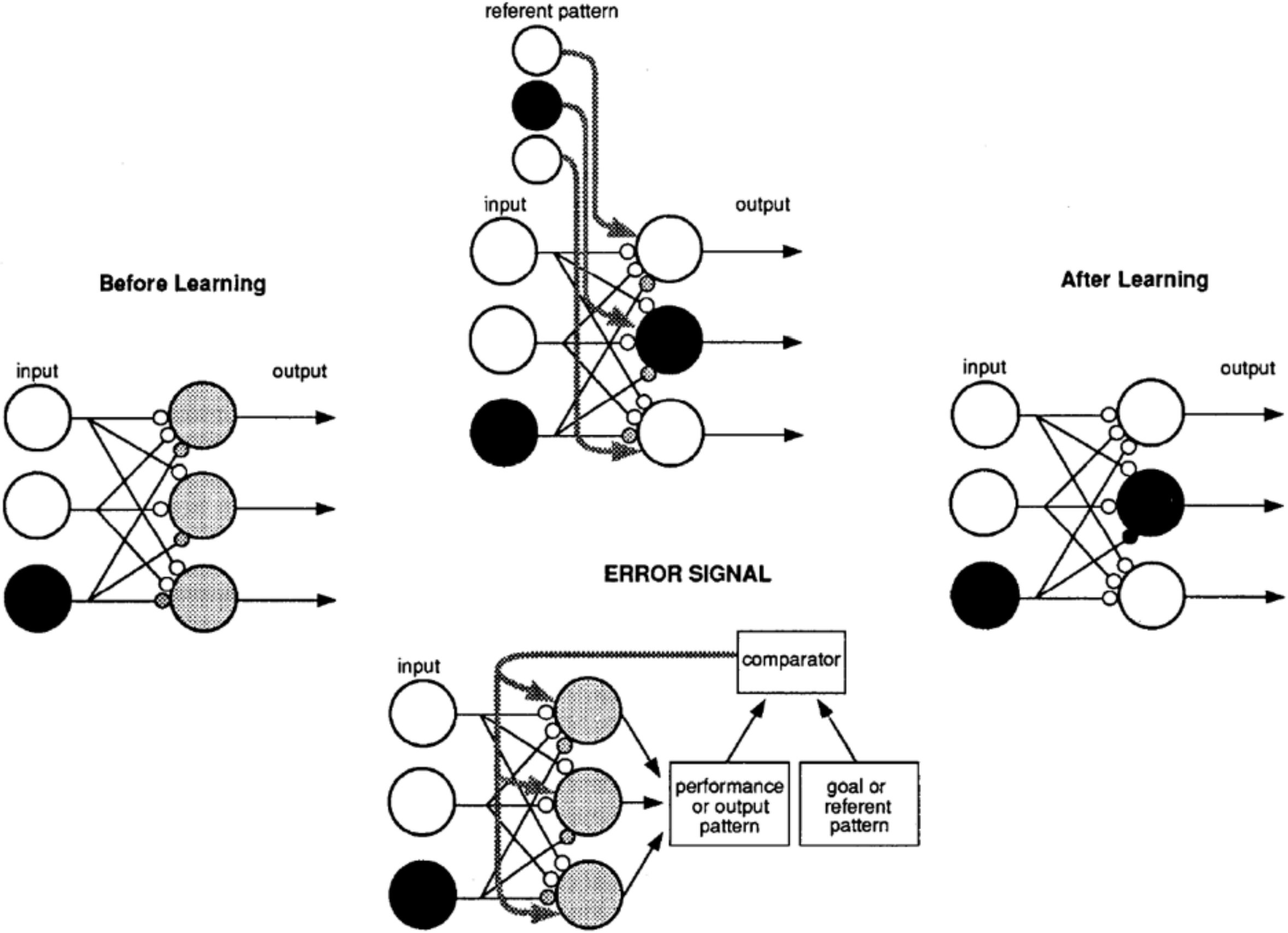
- Gaussians with sigma —>0 become memory …the basic unit in the hierarchical VQ is a gaussian with 2 inputs and a center. A Gaussian with 1 input is \delta(x-m-1); a G with 2 inputs is the product of 2 one-dim Gaussians…thus I need only one hidden layer to compute f_1 AND f_2 AND…but I can also compute it in multiple layers similar to HVQ…thus each center can have a small dim

A

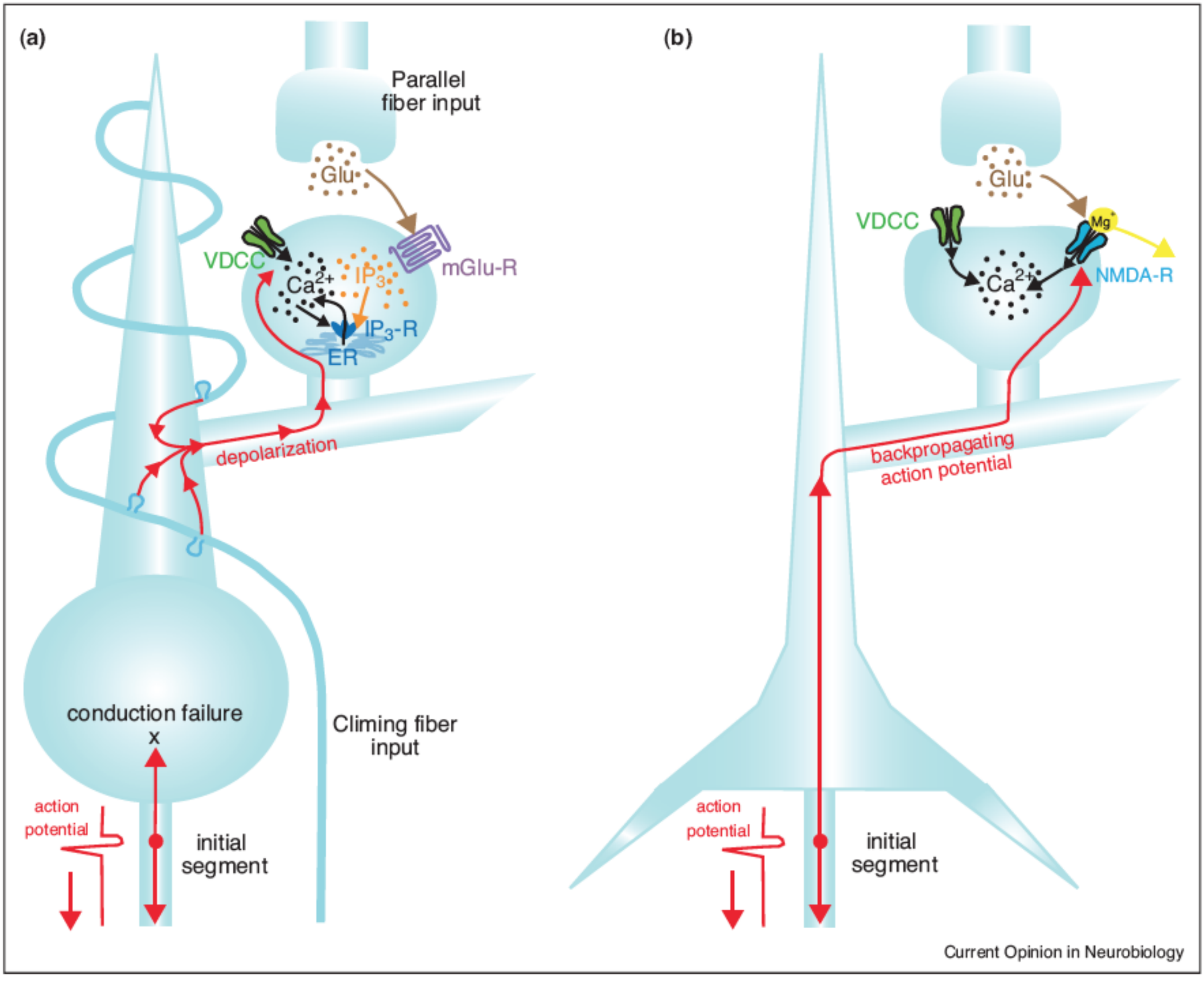

high
normal
low
eye velocity
head velocity
30 deg/s
150 ms

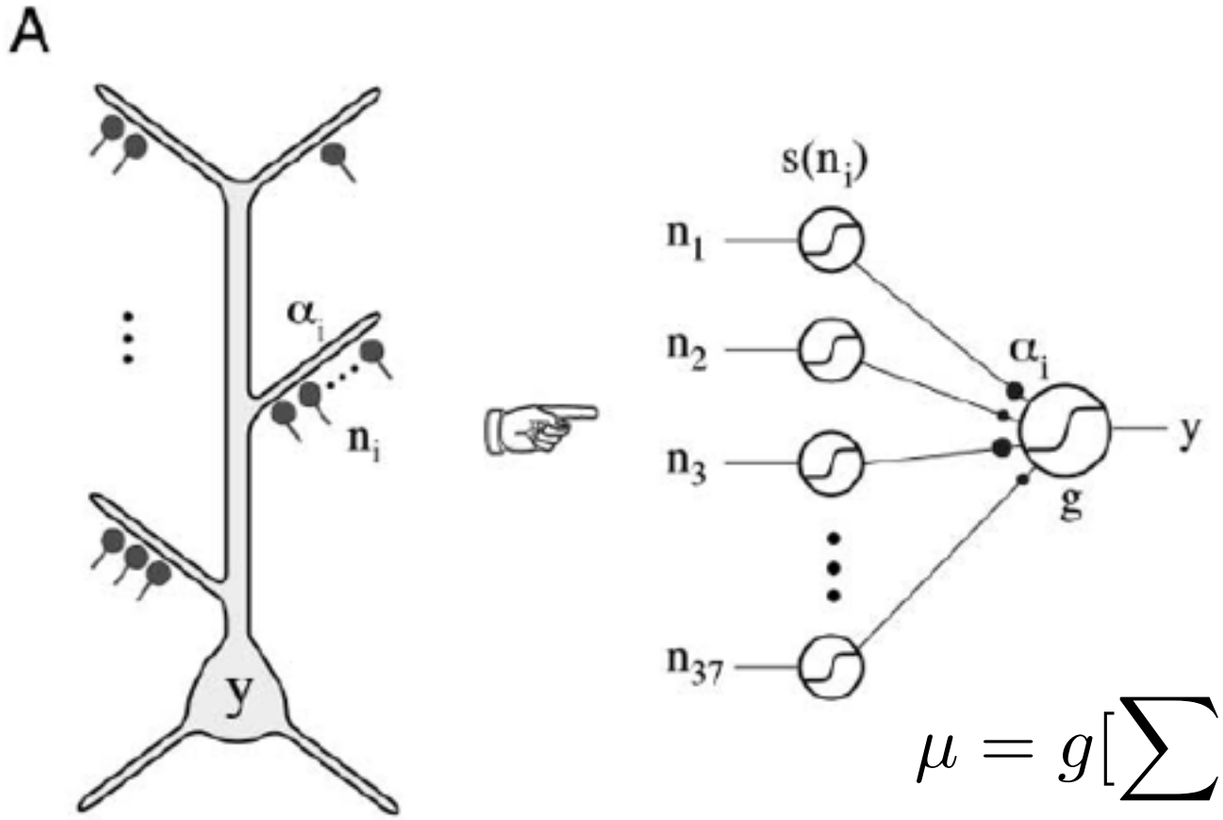
B


cerebellum
granule cell
PC
eye velocity
retinal slip
mossy fiber
climbing fiber
n. VIII and VN
vestibular input
FTN
output
Oculomotor nuclei
retinal slip
inferior olive



Hebbian mechanisms can be used for biological supervised learning (Knudsen, 1990)


(a)
Parallel
fiber input
Glu
VDCC
Ca2+
IP3
mGlu-R
IP3-R
ER
depolarization
conduction failure
x
Climing fiber
input
action
potential
initial
segment
(b)
Glu
VDCC
Mg+
Ca2+
NMDA-R
backpropagating
action potential
action
potential
initial
segment
Current Opinion in Neurobiology



$$\mu = g[\sum_i \alpha_i \sigma(< w^i, x^i >)]$$

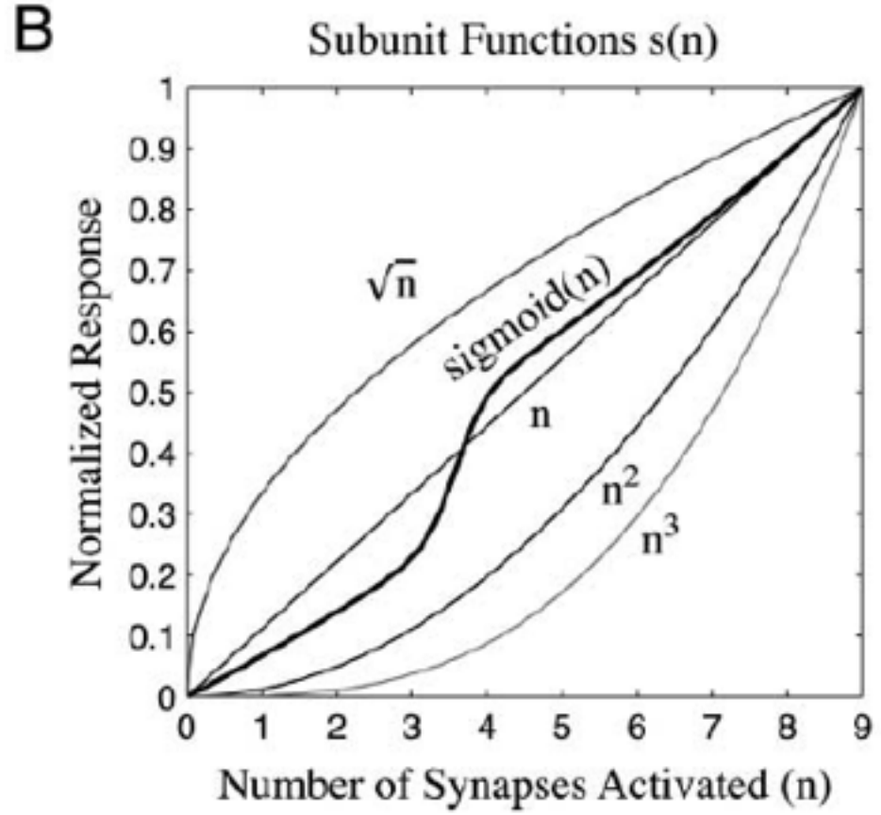


Figure 1. Pyramidal Neuron as Two-Layer Neural Network

Fully Dispersed

Heterogeneous

Fully Clustered