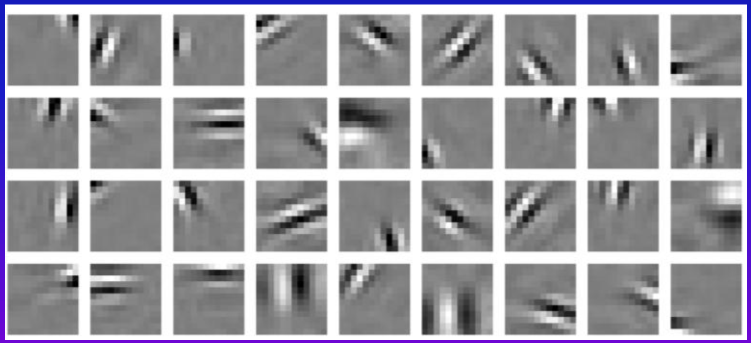
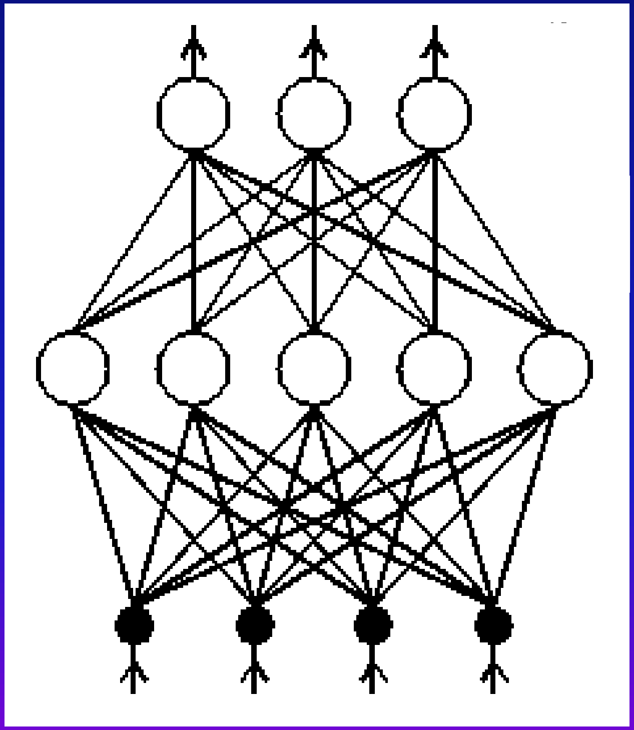
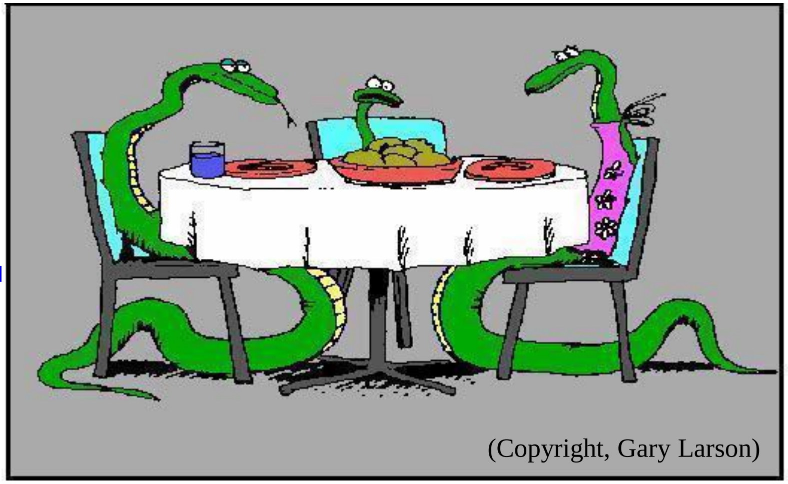
CSE/NB 528
Lecture 13: From Unsupervised Learning to Supervised Learning
(Chapters 8 & 10)





What's on the menu today?


(Copyright, Gary Larson)

"Oh, brother! . . . Not hamsters again!"

- Unsupervised Learning
  - Sparse coding and Predictive coding
- Supervised Learning
  - Classification versus Function Approximation/Regression
  - Perceptrons & Learning Rule
  - Linear Separability: Minsky-Papert deliver the bad news
  - Multilayer networks to the rescue
  - Radial Basis Function Networks

Recall: Generative Models for Unsupervised Learning

Causes **v**

Generative model

Data **u**

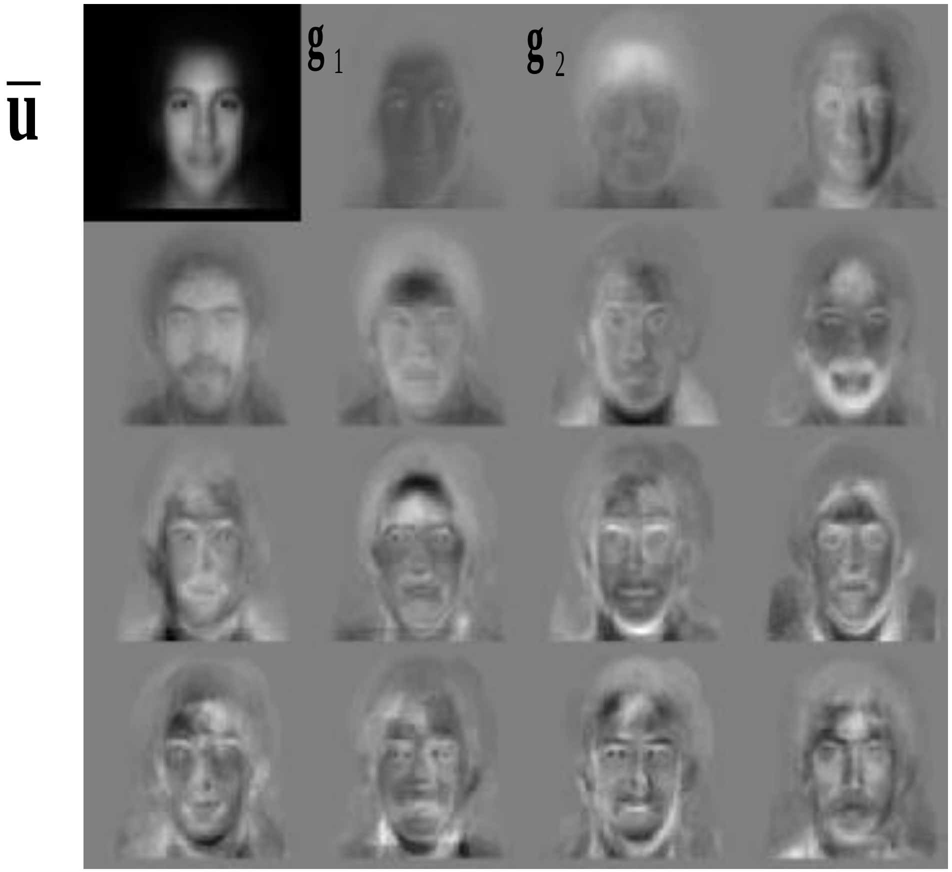
Suppose input **u** was generated by a linear superposition of causes $v_1, v_2, \ldots, v_k$ with basis vectors (or "features") $\mathbf{g}_i$

$$\mathbf{u} = \sum_i \mathbf{g}_i v_i + noise$$

(e.g., an image composed of several features, or audio containing several voices)

Example: "Eigenfaces"

- Suppose your basis vectors or "features" $\mathbf{g}_i$ are the eigenvectors of input covariance matrix of face images



Linear combination of eigenfaces

$\mathbf{g}_1 v_1$ $\mathbf{g}_2 v_2$ $\cdots$ $\mathbf{g}_8 v_8$

$$\bar{\mathbf{u}} + \sum_i \mathbf{g}_i v_i$$

Image

Linear Generative Model

- Suppose input **u** was generated by linear superposition of causes $v_1, v_2, \ldots, v_k$ and basis vectors or "features" $\mathbf{g}_i$:

$$\mathbf{u} = \sum_i \mathbf{g}_i v_i + noise = G\mathbf{v} + noise$$

- Problem: For a set of inputs **u**, estimate causes v_i for each **u** and learn feature vectors $\mathbf{g}_i$
  - Suppose number of causes is much lesser than size of input
- Idea: Find **v** and G that minimize reconstruction errors:

$$E = \frac{1}{2}\left\|\mathbf{u} - \sum_i \mathbf{g}_i v_i\right\|^2 = \frac{1}{2}(\mathbf{u} - G\mathbf{v})^T(\mathbf{u} - G\mathbf{v})$$

Probabilistic Interpretation

- E is the same as the *negative log likelihood* of data:
  Likelihood = Gaussian with mean $G\mathbf{v}$ and identity covariance matrix I

$$p[\mathbf{u} \mid \mathbf{v}; G] = N(\mathbf{u}; G\mathbf{v}, I)$$

$$E = -\log p[\mathbf{u} \mid \mathbf{v}; G] = \frac{1}{2}(\mathbf{u} - G\mathbf{v})^T(\mathbf{u} - G\mathbf{v}) + C$$

Minimizing error function E is the same as maximizing log likelihood of the data

Bayesian approach

- Would like to maximize posterior:

$$p[\mathbf{v} \mid \mathbf{u}; G] \propto p[\mathbf{u} \mid \mathbf{v}; G]\, p[\mathbf{v}; G]$$

- Equivalently, find **v** and *G* that maximize:

$$F(\mathbf{v}, G) = \left\langle \log p[\mathbf{u} \mid \mathbf{v}; G] + \log p[\mathbf{v}; G] \right\rangle$$

Prior for causes (what should this be?)

What do we know about the causes **v**?

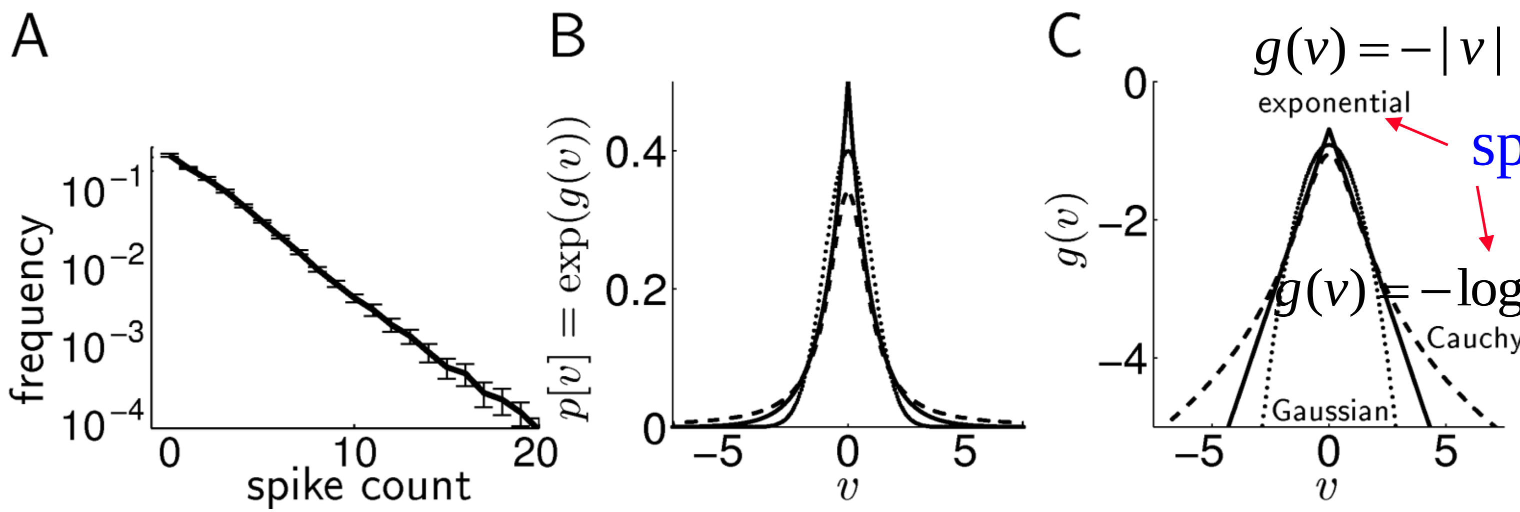
- We would like the causes to be *independent*
  - If cause A and cause B always occur together, then perhaps they should be treated as a single cause AB?
- Examples:
  - Image: Composed of several independent edges
  - Sound: Composed of independent spectral components
  - Objects: Composed of several independent parts

What do we know about the causes **v**?

- We would like the causes to be *independent*
- Idea 1: We would like: $p[\mathbf{v};G] = \prod_a p[v_a;G]$
- Idea 2: If causes are independent, only a few of them will be active for any input
  - v_a will be 0 most of the time but high for a few inputs
  - Suggests a sparse distribution for the prior $p[v_a;G]$

Prior Distributions for Causes

Spikes in area IT in monkey viewing TV	Possible prior distributions	Log prior



$g(v) = -|v|$

sparse

$g(v) = -\log(1+v^2)$

$$p[\mathbf{v}; G] \propto \prod_a \exp(g(v_a))$$

Finding the optimal **v** and *G*

- Want to maximize:

$$F(\mathbf{v},G) = \left\langle \log p[\mathbf{u}\,|\,\mathbf{v};G] + \log p[\mathbf{v};G] \right\rangle$$

$$= \left\langle -\frac{1}{2}(\mathbf{u}-G\mathbf{v})^T(\mathbf{u}-G\mathbf{v}) + \sum_a g(v_a) \right\rangle + K$$

- Approximate EM algorithm:
  - E step: Maximize *F* with respect to **v** keeping G fixed
    - Set d**v**/dt $\propto$ d*F*/d**v** ("gradient ascent/hill-climbing")
  - M step: Maximize *F* with respect to G, given the **v** above
    - Set d*G*/dt $\propto$ d*F*/d*G* ("gradient ascent/hill-climbing")

(During implementation, let **v** converge for each input before changing synaptic weights *G*)

E Step: Estimating **v**

Gradient ascent

$$\frac{d\mathbf{v}}{dt} \propto \frac{dF}{d\mathbf{v}} = G^T(\mathbf{u} - G\mathbf{v}) + g'(\mathbf{v})$$

Reconstruction (prediction) of **u**

$$\tau \frac{d\mathbf{v}}{dt} = G^T(\mathbf{u} - G\mathbf{v}) + g'(\mathbf{v})$$

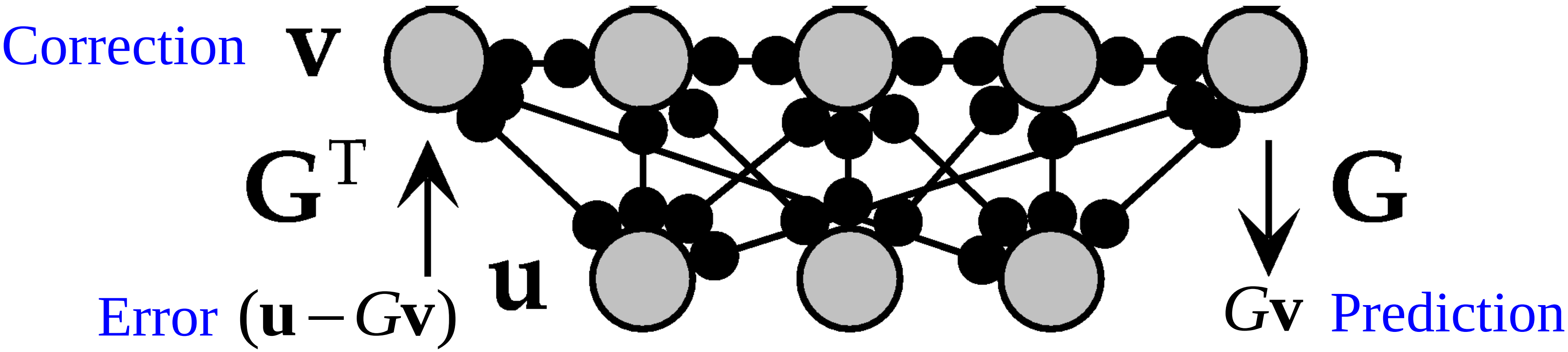
Firing rate dynamics (Recurrent network)

Error

Sparseness constraint

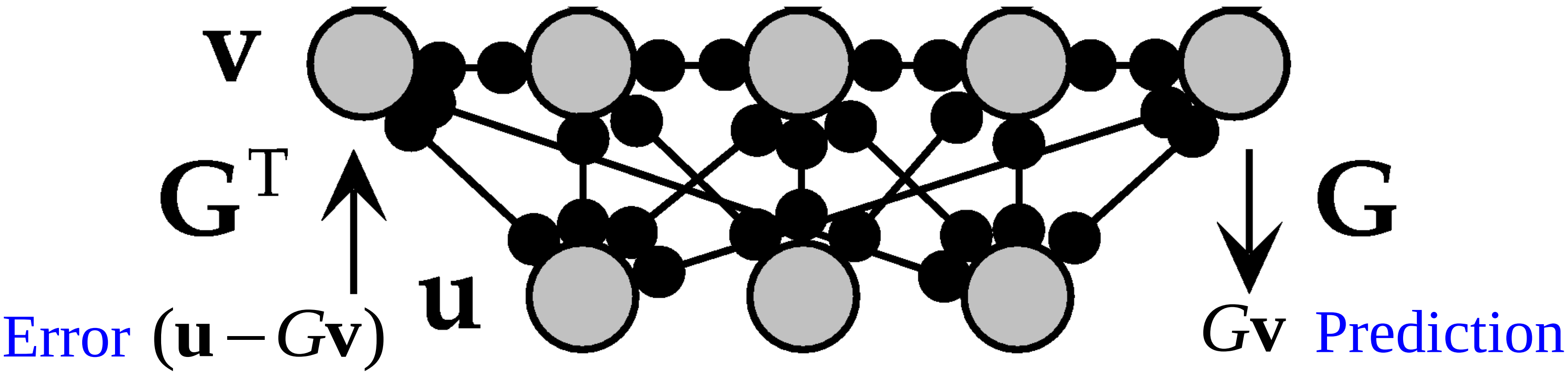
Recurrent network for estimating **v**

$$\tau \frac{d\mathbf{v}}{dt} = G^T(\mathbf{u} - G\mathbf{v}) + g'(\mathbf{v})$$



[Suggests a role for feedback pathways in the cortex (Rao & Ballard, 1999)]

M step: Learning the Synaptic Weights G

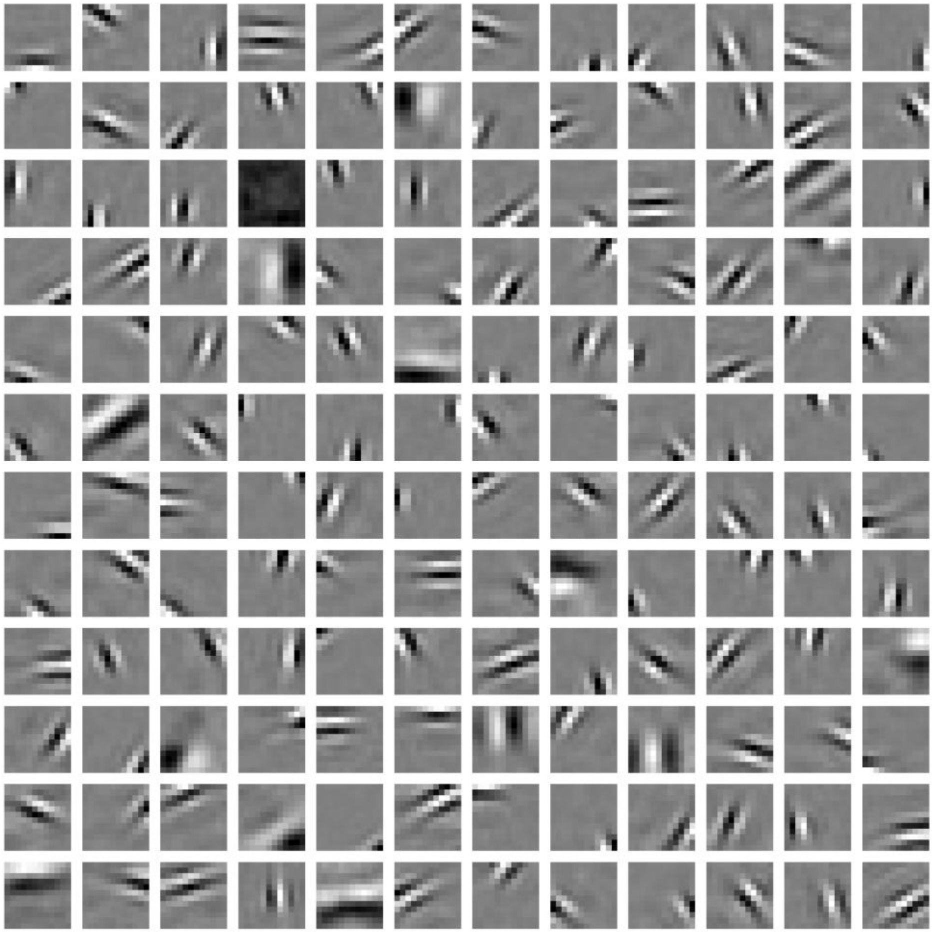


Gradient ascent $$\frac{dG}{dt} \propto \frac{dF}{dG} = (\mathbf{u} - G\mathbf{v})\mathbf{v}^T$$

Learning rule $$\tau_G \frac{dG}{dt} = (\mathbf{u} - G\mathbf{v})\mathbf{v}^T$$

Hebbian!
(similar to Oja's rule)

Result: Learning G for Natural Images



(Olshausen & Field, 1996)

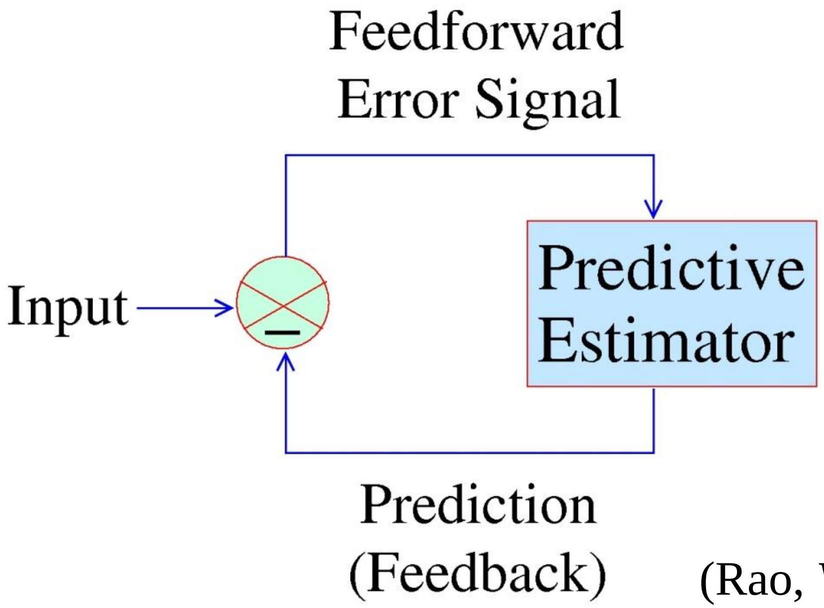
Each square is a column $\mathbf{g}_i$ of G (obtained by collapsing rows of the square into a vector)

Almost all the $\mathbf{g}_i$ represent local edge features

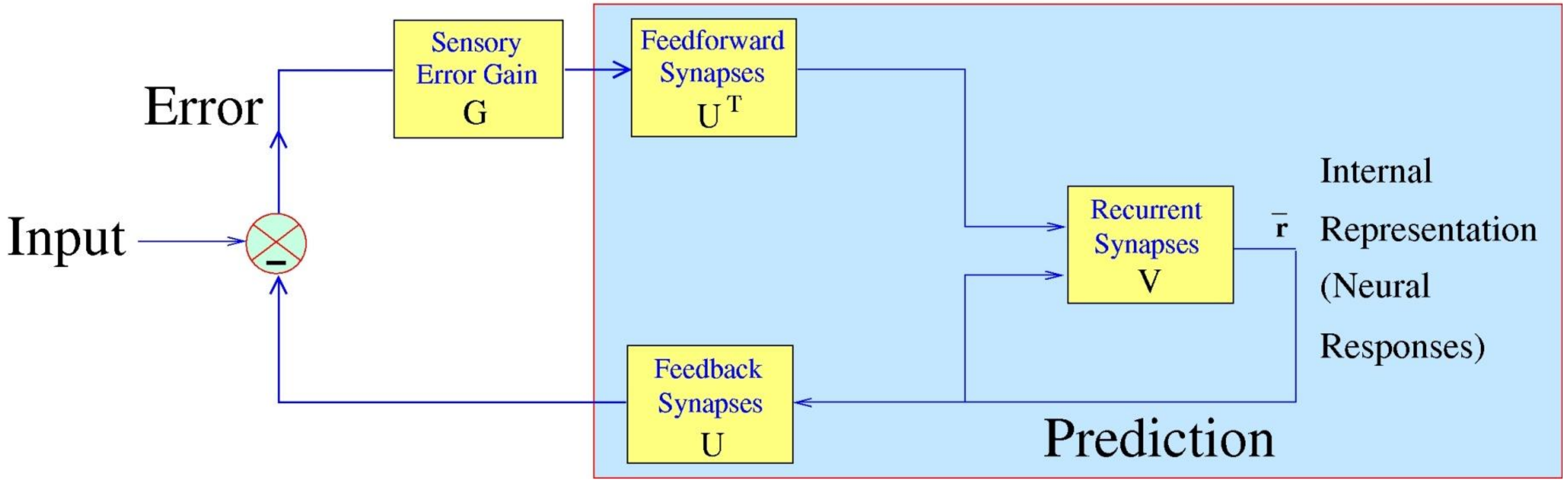
Any image patch $\mathbf{u}$ can be expressed as:

$$\mathbf{u} = \sum_i \mathbf{g}_i v_i = G\mathbf{v}$$

Sparse Coding Network is a special case of Predictive Coding Networks

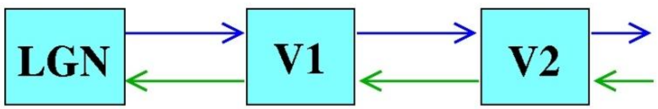


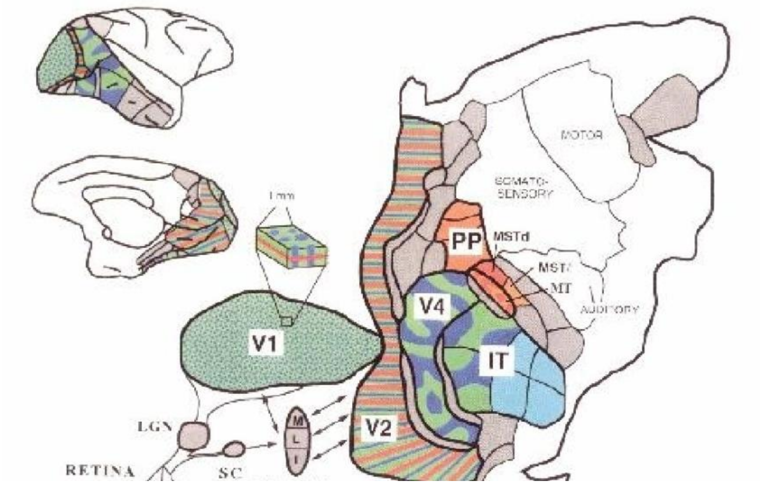
(Rao, *Vision Research*, 1999)

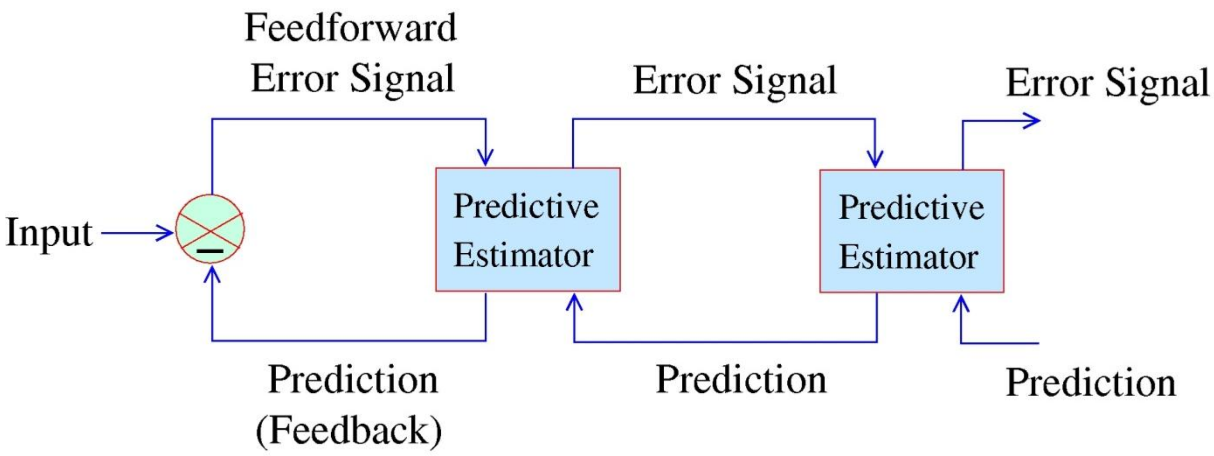


(See also Chapter 12 in the Anastasio textbook)

Predictive Coding Model of Visual Cortex





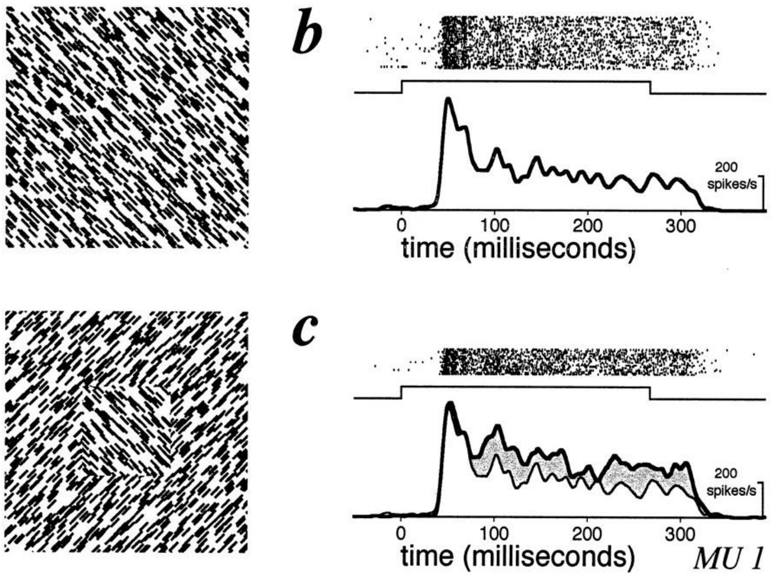


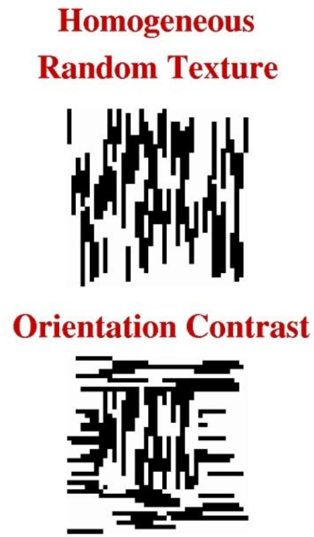
(Rao & Ballard, *Nature Neurosci.*, 1999)

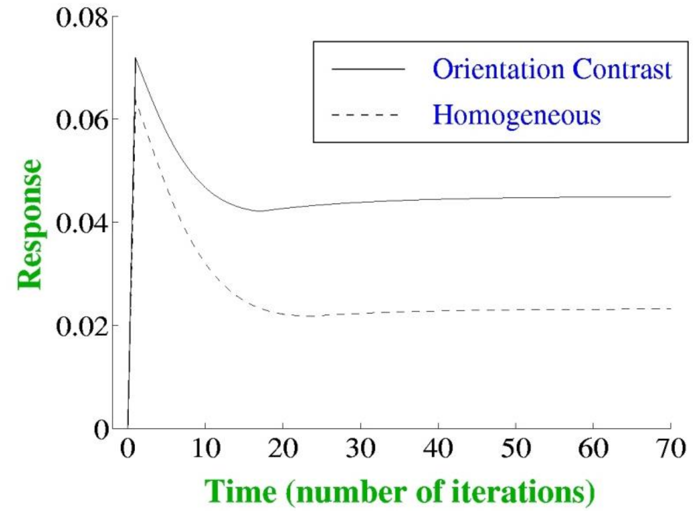
Predictive coding model explains contextual effects

Monkey Primary Visual Cortex

Model



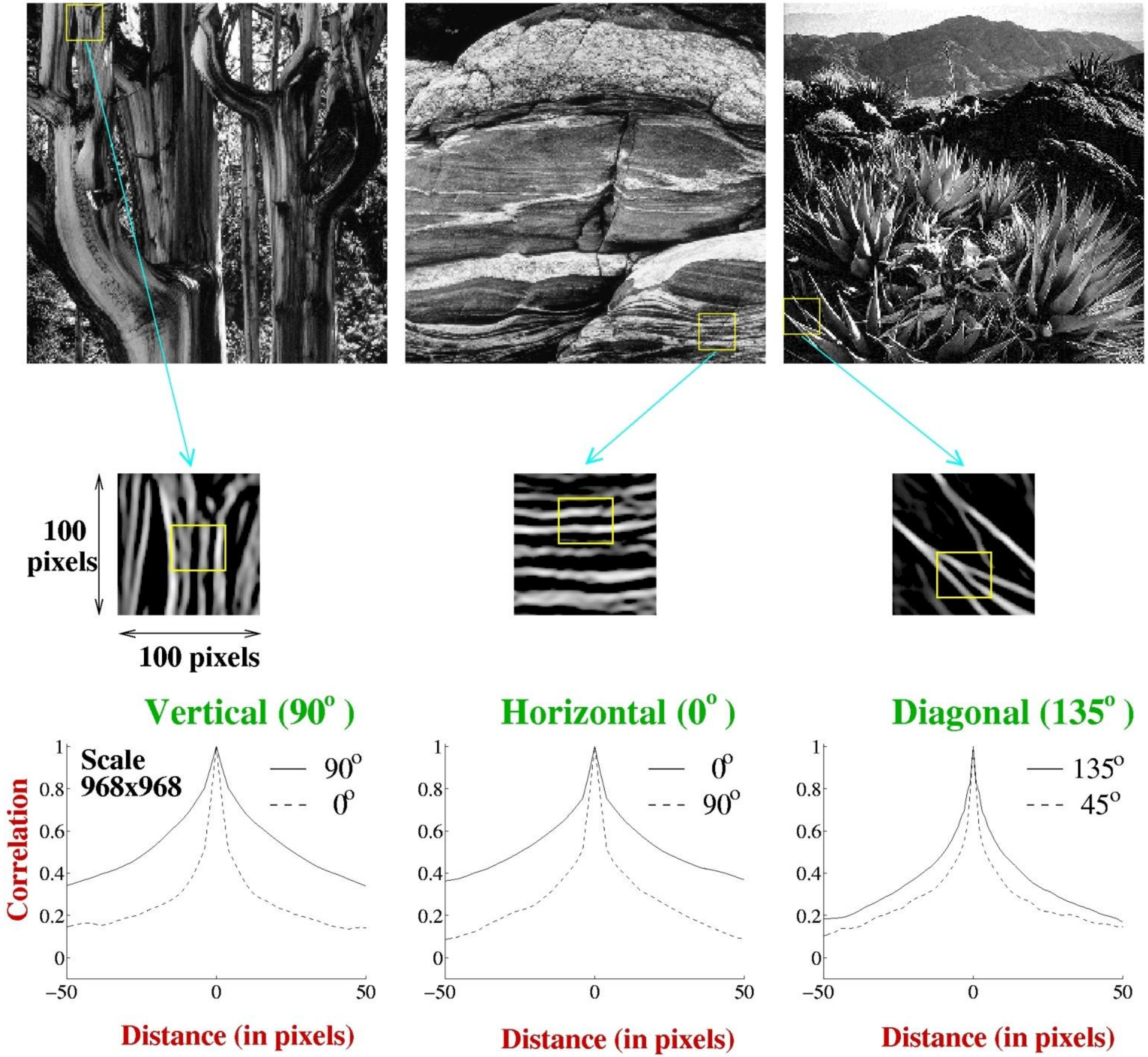




(Zipser et al., *J. Neurosci*., 1996)

(Rao & Ballard, *Nature Neurosci*., 1999)

Contextual effects arise from Natural Image properties



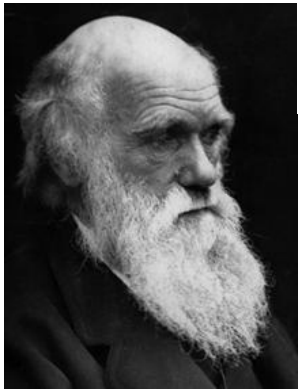
(Rao & Ballard, *Nature Neurosci.*, 1999)

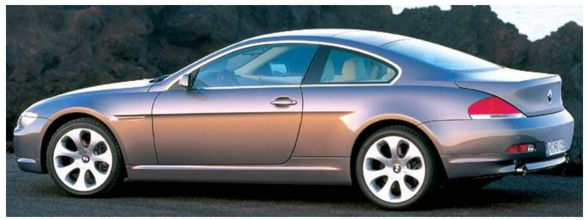
What if your data comes with not just inputs but also outputs?

Enter…Supervised Learning

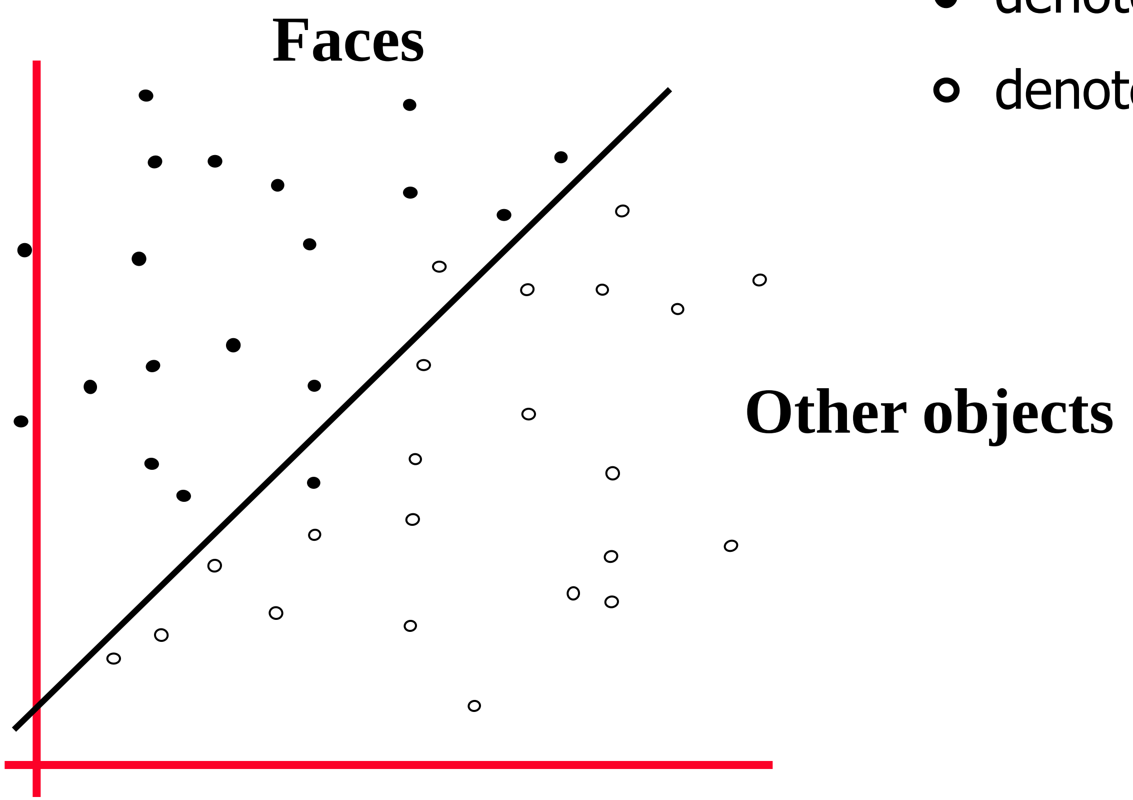
Example: Supervised Learning for Face Detection

Can we learn a network to distinguish faces from other objects?





The Classification Problem



- ● denotes output of +1 (faces)
- ○ denotes output of -1 (other)

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

Supervised Learning

- Two Primary Tasks

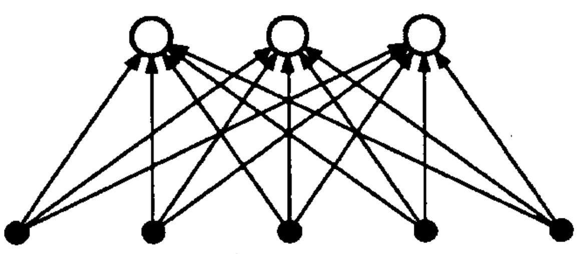
  1. Classification
  - Inputs u_1, u_2, … and discrete classes C_1, C_2, …, C_k
  - Training examples: (u_1, C_2), (u_2, C_7), etc.
  - Learn the mapping from an arbitrary input to its class
  - Example: Inputs = images, output classes = face, not a face

  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

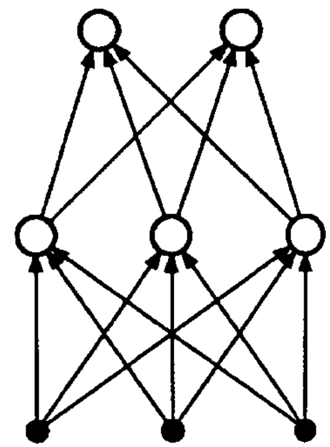
Classification using "Perceptrons"

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

Single-layer



Multilayer

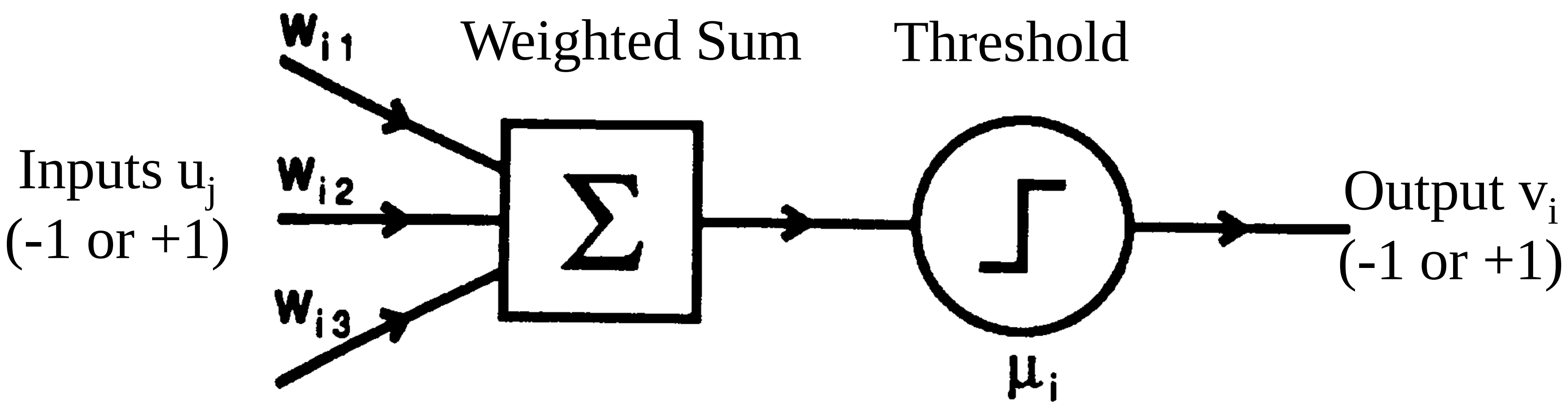


Perceptrons use “Threshold Units”

- 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 $x < 0$



What does a Perceptron compute?

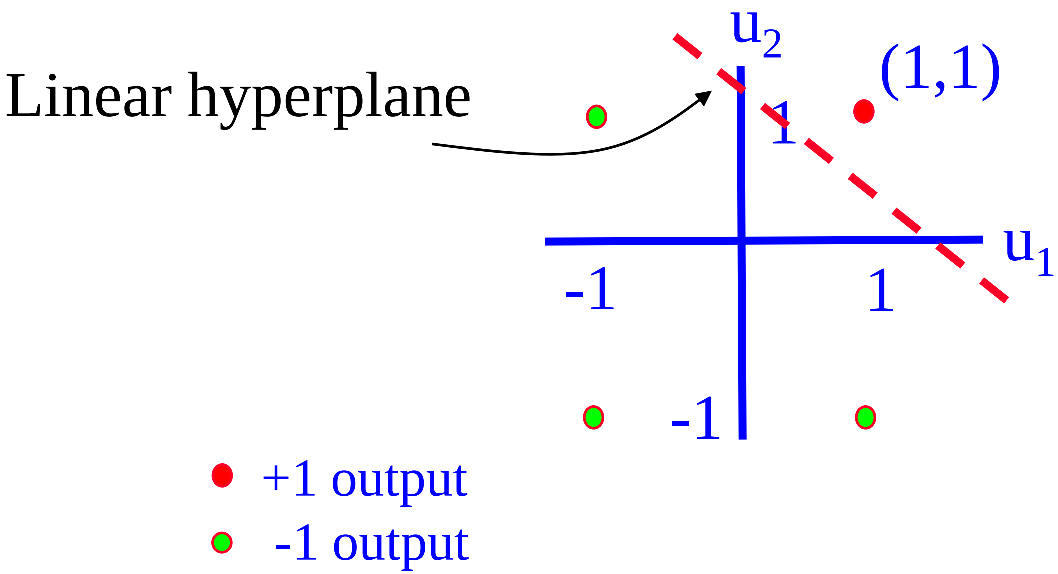
- Consider a single-layer perceptron
  - Weighted sum forms a *linear hyperplane (line, plane, …)*

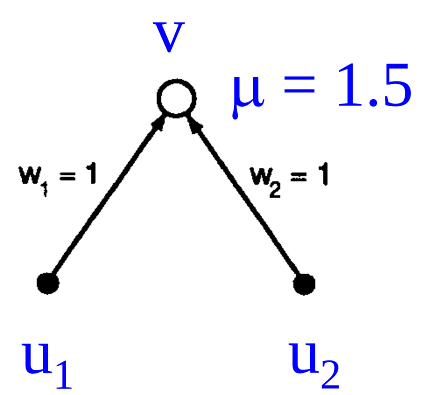
$$\sum_j w_{ij} u_j - \mu_i = 0$$

  - Everything *on one side* of hyperplane is in class 1 (output = +1) and everything *on other side* is class 2 (output = -1)
  - <u>Any function that is linearly separable can be computed by a perceptron</u>

Linear Separability

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





Perceptron for AND

Perceptron Learning Rule

- Given inputs **u** and desired output v^d, adjust **w** as follows:

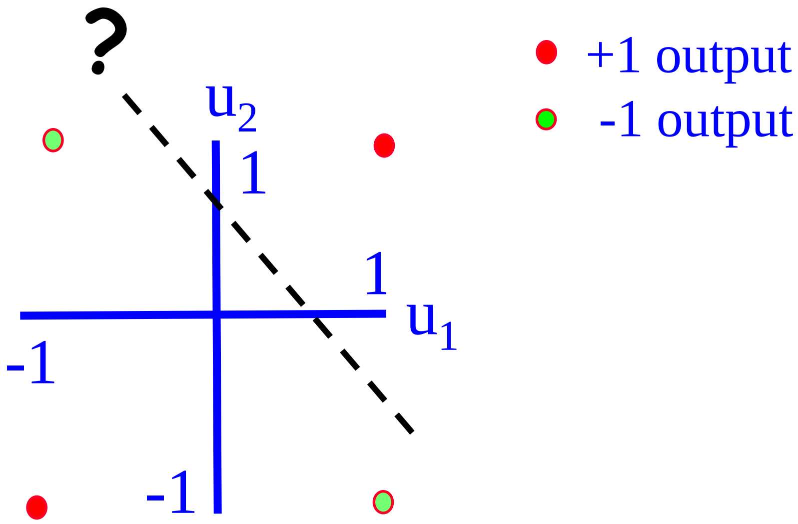
1. Compute error signal $e = (v^d - v)$ where v is the current output
2. Change weights according to the error:

$$\mathbf{w} \rightarrow \mathbf{w} + \varepsilon(v^d - v)\mathbf{u} \qquad A \rightarrow B \text{ means replace } A \text{ with } B$$

⇒ E.g., for positive inputs, this increases weights if error is positive and decreases weights if error is negative (opposite for negative inputs)

What about the XOR function?

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

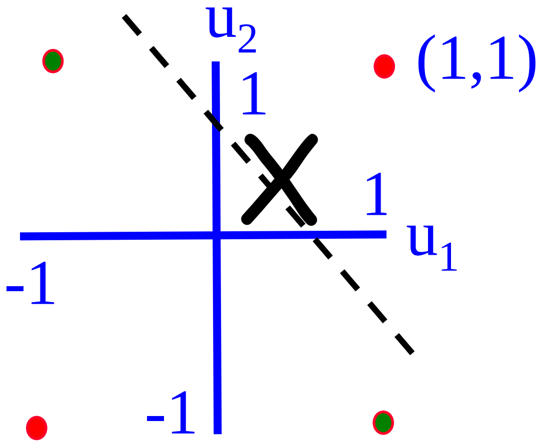


Can a straight line separate the +1 outputs from the -1 outputs?

Linear Inseparability

- Single-layer perceptron with threshold units fails if classification task is not linearly separable
  - Example: XOR
  - No single line can separate the "yes" (+1) outputs from the "no" (−1) outputs!

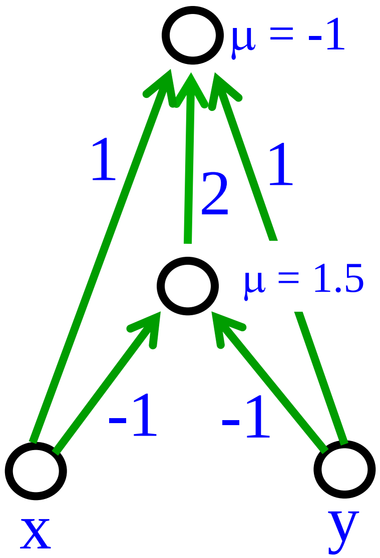
- Minsky and Papert's book showing such negative results put a damper on neural networks research for over a decade!



How do we deal with linear inseparability?

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 $x + y + 2\Theta(-x - y - 1.5) > -1$

(Inputs x and y can be +1 or -1)

What if you want to approximate a *continuous* function?

Can a network learn to drive?

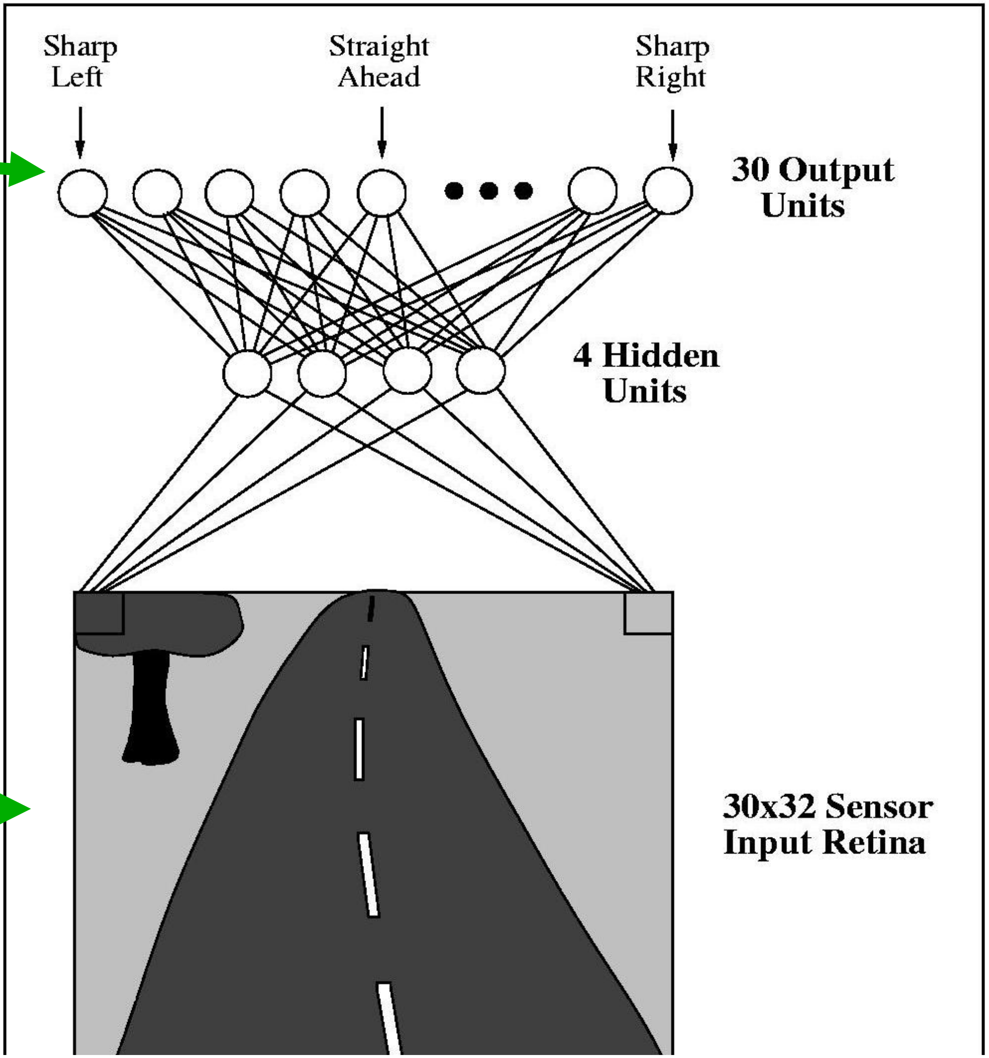
Example Network

Steering angle

Desired Output:

$\mathbf{d} = [d_1\ d_2\ \ldots\ d_{30}]$

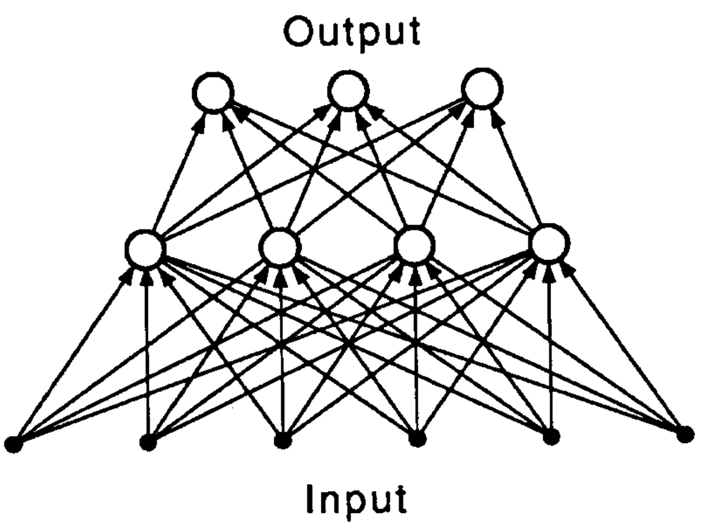
Current image



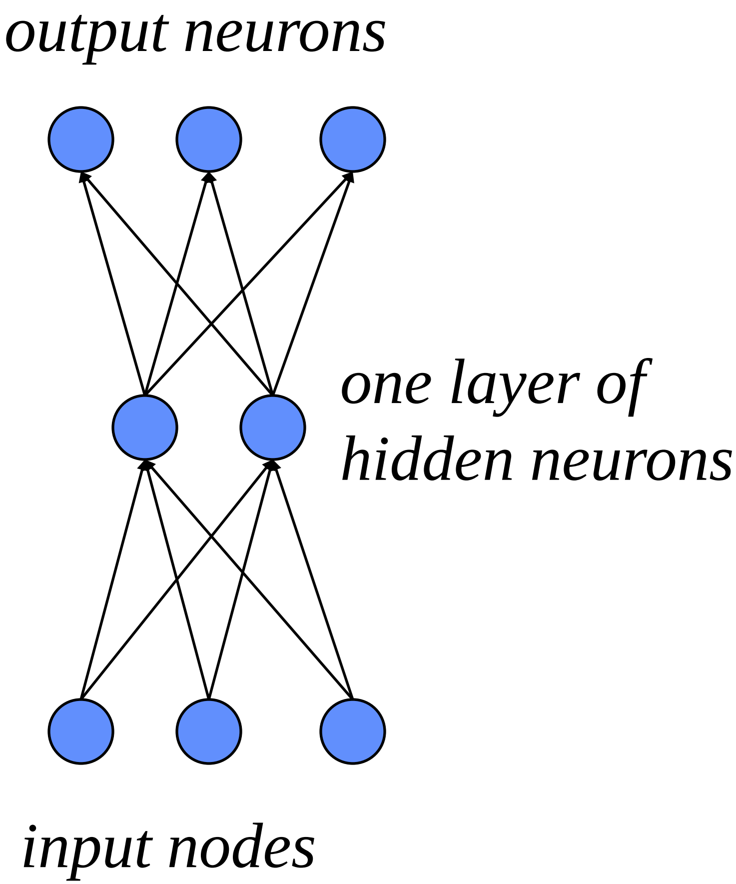
Input $\mathbf{u} = [u_1\ u_2\ \ldots\ u_{960}]$ = image pixels

Function Approximation

- We want networks that can learn a function
  - Network maps real-valued inputs to real-valued outputs
  - Want to generalize to predict outputs for new inputs
  - Idea: Given input data, map input to desired output by *adapting weights*

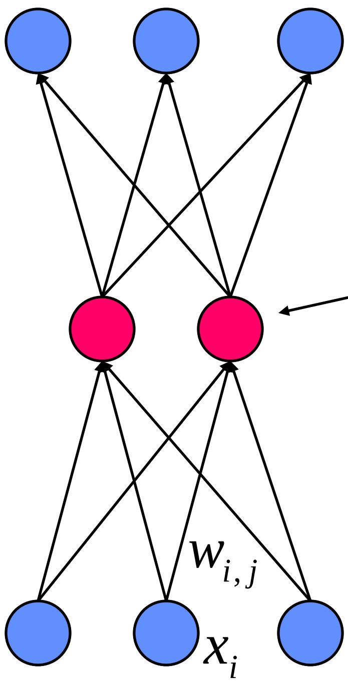


Example: Radial Basis Function (RBF) Networks



Radial Basis Function Networks

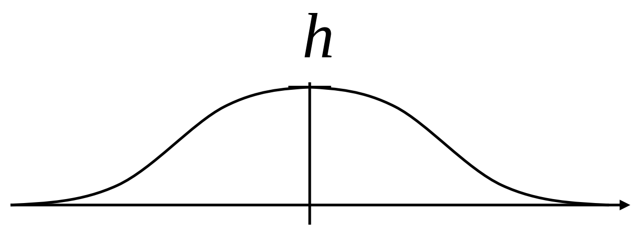
output neurons



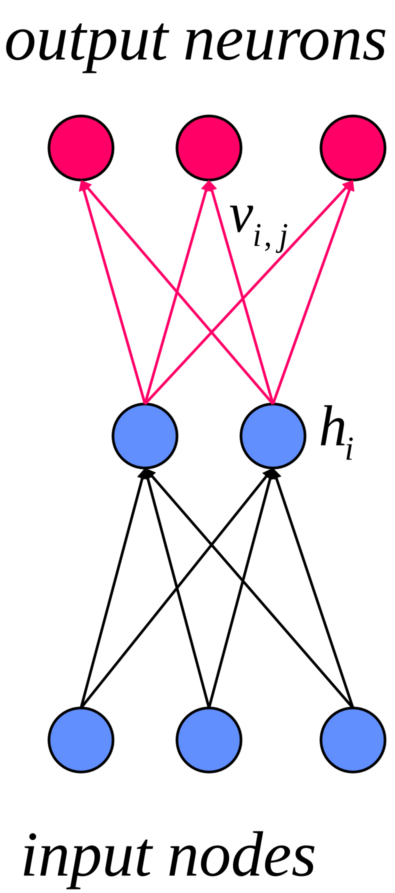
input nodes

Hidden layer output: $h_j = e^{-\frac{\sum_{i=1}^{n}(x_i - w_{i,j})^2}{2\sigma^2}}$

(Gaussian bell-shaped function)



Radial Basis Function Networks



output of network:

$$\text{out}_j = \sum_i v_{i,j} h_i$$

- Main Idea: Use a mixture of Gaussian functions h_i to approximate the output
- Gaussians are called "basis functions"

RBF networks

- Each hidden unit stores a mean (in its weights) and a variance
- Each hidden unit computes a Gaussian function of input $\mathbf{x}$
- Can derive learning rules for output weights v_i, means $\mathbf{w}_i$, and variances σ^2_i by minimizing squared output error function (via gradient descent learning)
- See http://en.wikipedia.org/wiki/Radial_basis_function_network for more details and links.

Next Class: Backpropagation and Reinforcement Learning

- Things to do:
  - Read Chapter 9
  - Finish Homework 3 (due Friday, May 20)
  - Work on group project