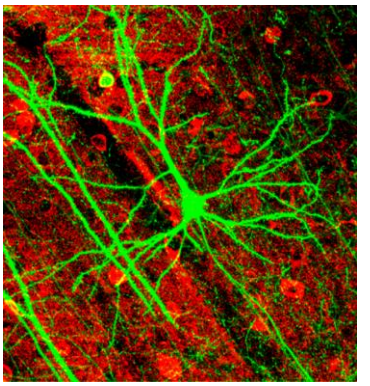
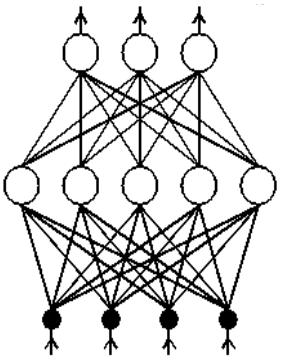
# CSE/NEURO 528
# Lecture 12: Unsupervised to Supervised Learning
(Chapters 8 & 10)







## Last Time: A Linear Model for Natural Images

Input image **u**

$$= \boxed{?} \cdot v_1 + \boxed{?} \cdot v_2 + \boxed{?} \cdot v_3 + \cdots$$

($\mathbf{g}_1$, $\mathbf{g}_2$, $\mathbf{g}_3$ above the boxes)

$$\mathbf{u} = \sum_{i=1}^{M} \mathbf{g}_i v_i + noise$$

(Note: $M$ can be larger than $N$)

$$= G\mathbf{v} + noise$$

$G$ = matrix whose columns are $\mathbf{g}_i$
$\mathbf{v}$ = vector whose elements are $v_i$

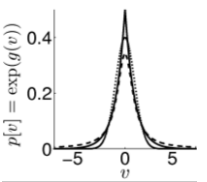
## Defining the Generative Model: Prior

Causes **v**

Prior $p[\mathbf{v}]$

Generative model

Input Data **u**




Assuming causes $v_i$ are independent:

$$p[\mathbf{v}] = \prod_i p[v_i]$$

$$\log p[\mathbf{v}] = \sum_i \log p[v_i; G]$$

For any input, we want only a few causes $v_i$ to be active

- $v_i$ = 0 most of the time but high for some inputs
- Suggests sparse distribution for $p[v_i]$: peak at 0 but with heavy tail (also called *super-Gaussian* distribution)



## Defining the Generative Model: Likelihood

Causes **v**

Prior $p[\mathbf{v}]$

Generative model

Input Data **u**

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

Linear model:

$$\mathbf{u} = G\mathbf{v} + noise$$

Assume *noise* is Gaussian white noise:

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

$$\propto \exp(-\frac{1}{2}\|\mathbf{u} - G\mathbf{v}\|^2)$$

Log likelihood:

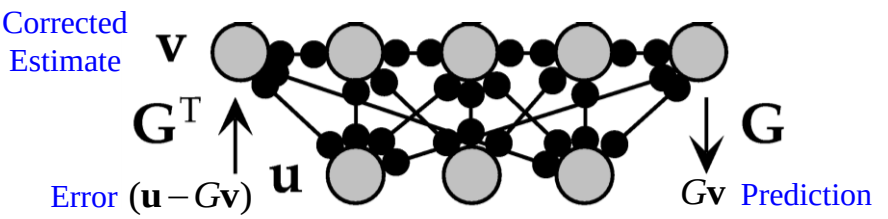
$$\log p[\mathbf{u} \mid \mathbf{v}; G] = -\frac{1}{2}\|\mathbf{u} - G\mathbf{v}\|^2 + C$$

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

$$= -\frac{1}{2}\|\mathbf{u} - G\mathbf{v}\|^2 + \sum_i g(v_i) + K$$

Find **v** and $G$ that maximize log of posterior probability

## Recurrent Network for Sparse Coding




Firing rate dynamics

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

*Sparseness penalty*

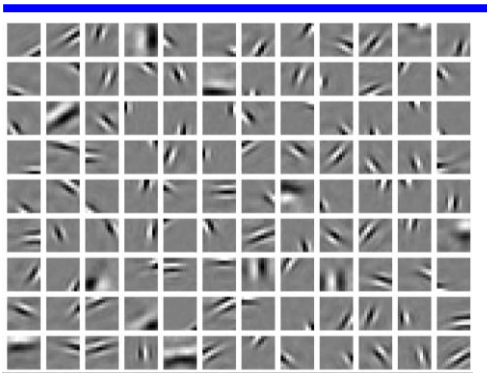
Learning rule

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

*Hebbian*



## Result: Sparse Coding of Natural Images



*G*

(Olshausen & Field, 1996)

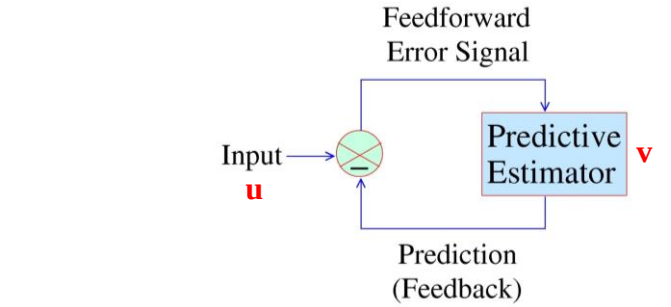
Each square is a column $\mathbf{g}_i$ of $G$ (obtained by collapsing rows of the square into a vector)

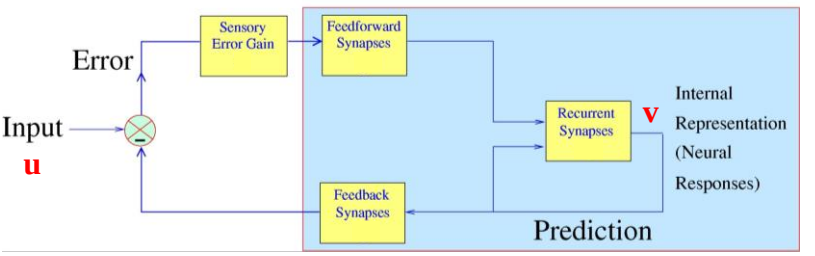
The $\mathbf{g}_i$ look like local edge or bar features similar to receptive fields in primary visual cortex (V1)

Image patch **u**

$$= \boxed{\mathbf{g}_1} \cdot v_1 + \boxed{\mathbf{g}_2} \cdot v_2 + \boxed{\mathbf{g}_3} \cdot v_3 + \cdots$$

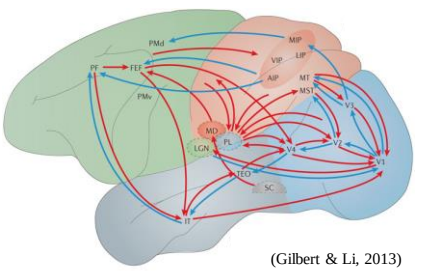
## Sparse Coding is a special case of Predictive Coding



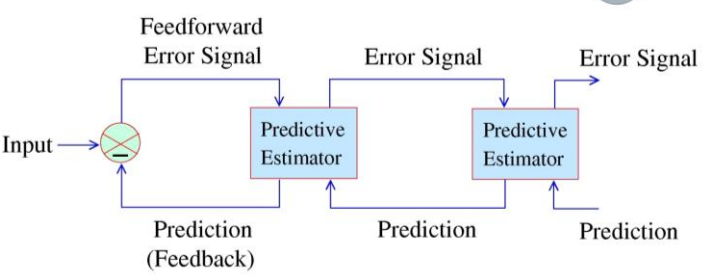




For more details, see: (Rao, *Vision Research*, 1999; Rao & Ballard, *Nature Neurosci*., 1999)

## Predictive Coding Model of the Visual Cortex

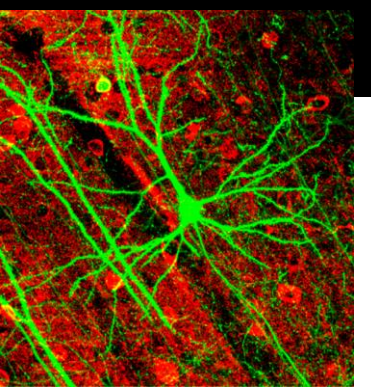



(Gilbert & Li, 2013)




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

# From Unsupervised to Supervised Learning: Neurons as Classifiers





# The Classification Problem

How do we build a classifier to distinguish between faces and other objects?











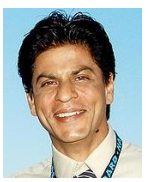
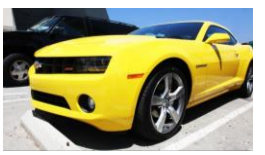
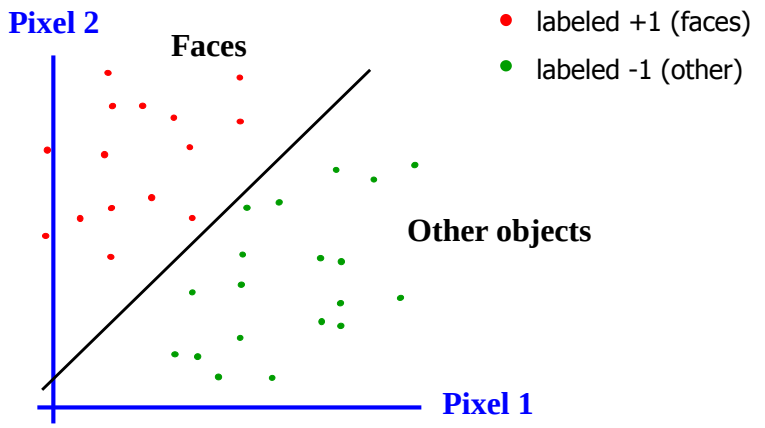






Image Source: Wikimedia Commons

# The Classification Problem

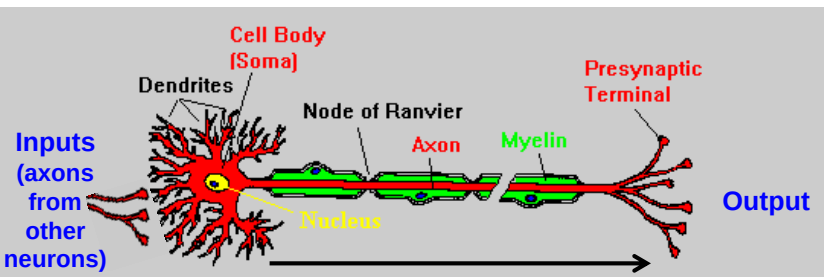



**Idea: Find a separating hyperplane (line in this case)**

**Can neurons do that?**

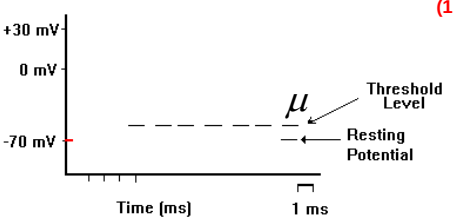


# The Idealized Neuron



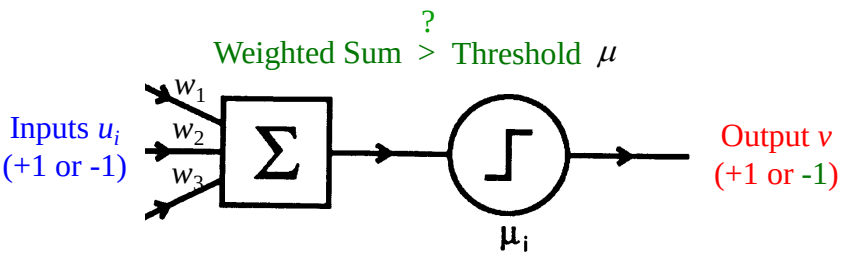

$u_i \quad w_i \quad \sum_i w_i u_i > \mu ? \quad v$

Spike or no spike
(1 or -1)




Images by Eric Chudler, UW

## The "Perceptron"




$$v = \Theta(\sum_i w_i u_i - \mu)$$

$\Theta(x)$ = +1 if $x > 0$ and -1 if $x \leq 0$

[Introduced by Rosenblatt (1958) building on McCulloch and Pitts (1943)]

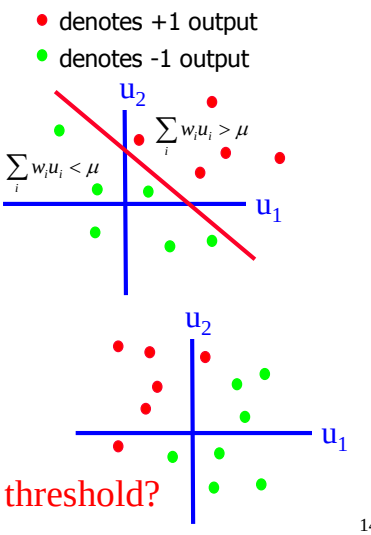


## What does a Perceptron do?

- Weighted sum defines a *hyperplane (line, plane, ...)*

$$\sum_i w_i u_i - \mu = 0$$

- All inputs *on one side* of hyperplane have output = +1 ("class 1"); all inputs *on other side* have output = -1 ("class 2")
- Perceptrons can classify!
  - Can perform linear classification




How do we learn the weights and threshold?

## Perceptron Learning Rule

Given input **u**, output $v = \Theta(\sum_i w_i u_i - \mu)$, and desired output $v^d$:

Adjust $w_i$ and $\mu$ according to output error $(v^d - v)$:

$$\Delta w_i = \varepsilon(v^d - v)u_i$$

+1 -1
-1 +1

For positive input ($u_i$ = +1):
Increases weight if error is positive
Decreases weight if error is negative
(opposite for $u_i$ = -1)

$$\Delta \mu = -\varepsilon(v^d - v)$$

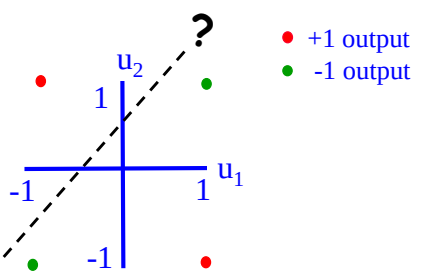
+1 -1
-1 +1

Decreases threshold if error is positive
Increases threshold if error is negative



## Can Perceptrons learn any function?

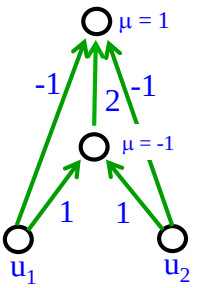
| $u_1$ | $u_2$ | XOR |
|---|---|---|
| -1 | -1 | -1 |
| 1 | -1 | +1 |
| -1 | 1 | +1 |
| 1 | 1 | -1 |




Perceptrons can only classify linearly separable data
How do we handle linear inseparability?

## Multilayer Perceptrons

- Can classify linearly inseparable data
  - Can solve XOR
- An example of a two-layer perceptron that computes XOR




(Inputs and outputs are +1 or -1)



What if you want *continuous* outputs rather than +1/-1 outputs (i.e., regression)?

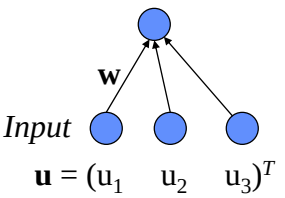
E.g., Teaching a network to drive a truck



Image Source: Wikimedia Commons

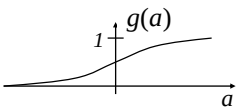
# Continuous Outputs with Sigmoid Networks

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

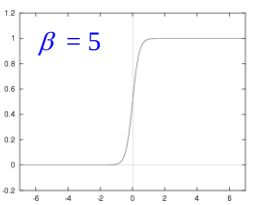


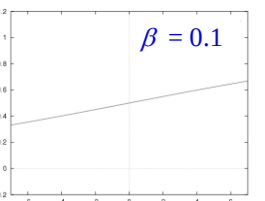

Sigmoid output function:

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




Parameter $\beta$ controls the slope






# Learning Multilayer Sigmoid Networks

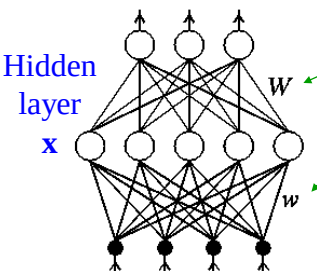
$$v_i = g(\sum_j W_{ij} g(\sum_k w_{jk} u_k)) = g(\sum_j W_{ij} x_j)$$

Desired output **d** also given

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

Learn weights that minimize output error:

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




Use gradient descent!

$$\Delta W_{ij} = -\varepsilon \frac{dE}{dW_{ij}} = \varepsilon \cdot (d_i - v_i) g'(\sum_j W_{ij} x_j) x_j$$

*Delta rule*

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

$$\Delta w_{jk} = -\varepsilon \frac{dE}{dw_{jk}} = -\varepsilon \frac{dE}{dx_j}\frac{dx_j}{dw_{jk}}$$

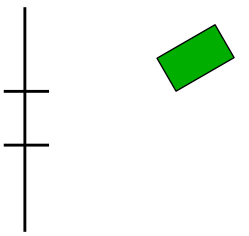
*Backpropagation learning rule*

(see Supplementary Materials for details)

## Example: Backing up a Truck (courtesy of Keith Grochow)

Teaching a Network to back a truck into a loading dock

- Input: x, y, θ of truck
- Output: Steering angle





Forget trucks doin' supervised learning – how do I find some food in a barn?

Enter…Reinforcement Learning

Image Source: Wikimedia Commons

# Next Class: Reinforcement Learning

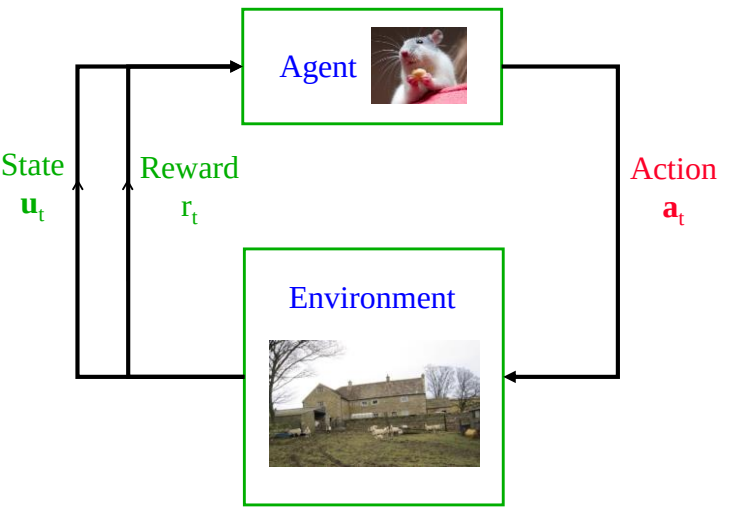





Image Source: Wikimedia Commons