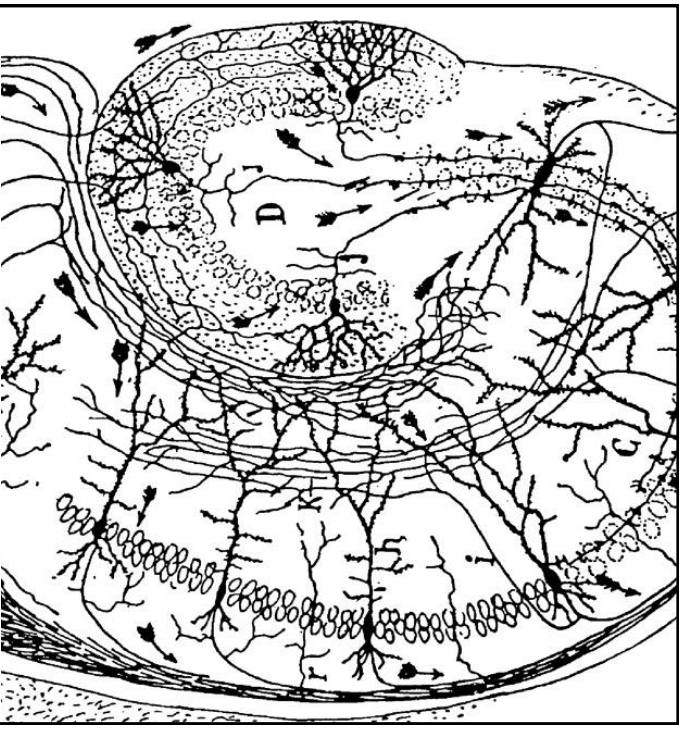
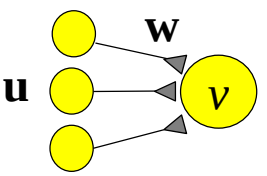
CSE/NEURO 528
Lecture 11: Unsupervised Learning
(Chapters 8 & 10)

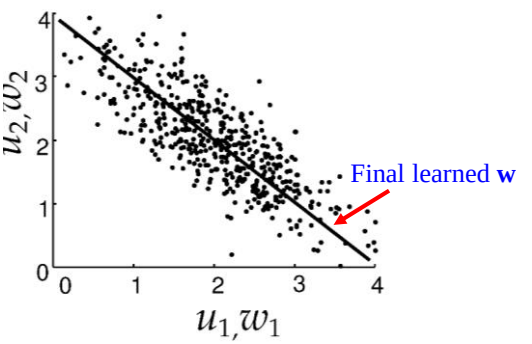


Last Time: Hebbian Learning implements *Principal Component Analysis* (PCA)



Covariance Rule

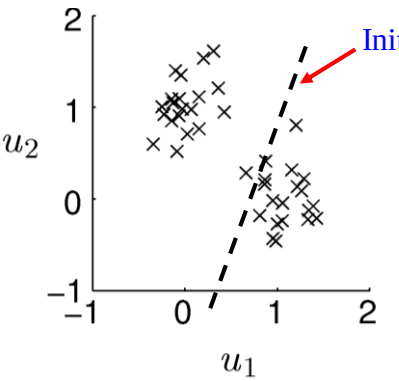
$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{u}(v - \langle v \rangle)$$



Hebbian learning learns a weight vector aligned with the principal eigenvector of input correlation/covariance matrix (i.e., direction of maximum variance)

Image Source: Dayan & Abbott textbook

What about this input data?



?

What **w** does the covariance rule learn?

Image Source: Dayan & Abbott textbook

PCA does not correctly describe the data

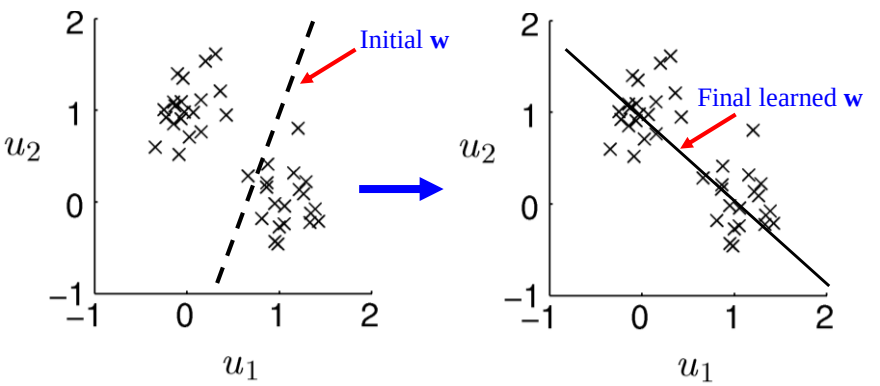
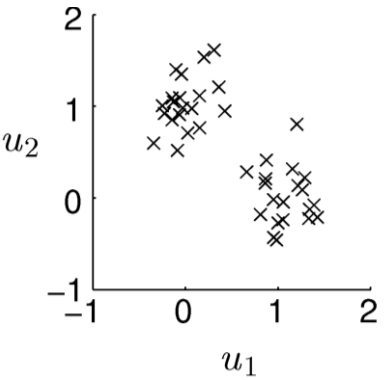


Image Source: Dayan & Abbott textbook

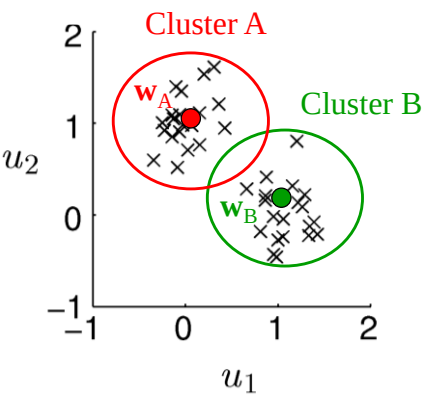
Introduction to Unsupervised Learning

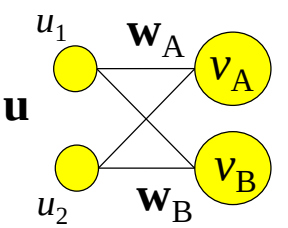


Input data seems to be made up of two clusters of points
Can neurons learn such clusters?

Image Source: Dayan & Abbott textbook

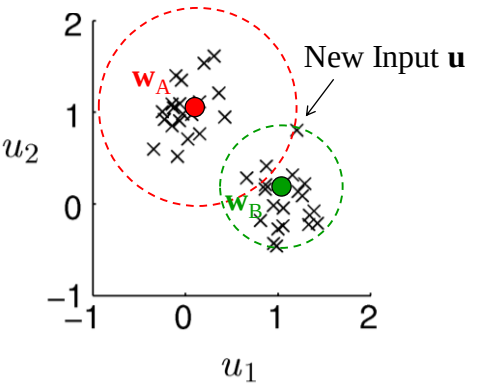
Using Neurons to represent Clusters

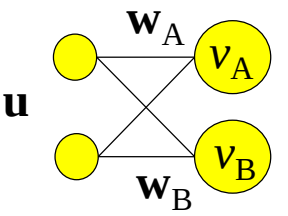




$$v_i = \mathbf{w}_i \cdot \mathbf{u} = \mathbf{w}_i^T \mathbf{u} = \mathbf{u}^T \mathbf{w}_i$$

What is the relation between *most active neuron* versus one whose *weight vector is closest to the input* ?



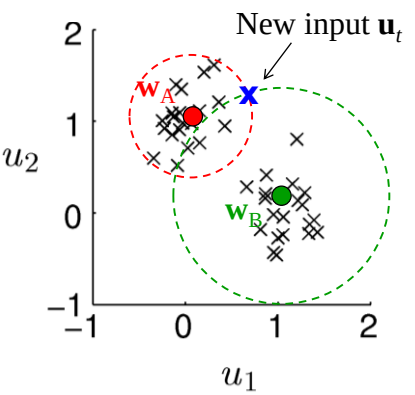


$$\|\mathbf{u}-\mathbf{w}_i\|^2 = (\mathbf{u}-\mathbf{w}_i)^T(\mathbf{u}-\mathbf{w}_i)$$
$$= \mathbf{u}^T\mathbf{u} + \mathbf{w}_i{}^T\mathbf{w}_i - \mathbf{u}^T\mathbf{w}_i - \mathbf{w}_i{}^T\mathbf{u}$$
$$= \|\mathbf{u}\|^2 + \|\mathbf{w}_i\|^2 - 2v_i$$

Pick neuron i (= A or B) based on $\arg\min_i \|\mathbf{u}-\mathbf{w}_i\|^2 = \arg\max_i v_i$

since $\|\mathbf{u}\|=1$ and $\|\mathbf{w}_i\|=1$

Updating the Weights given a New Input

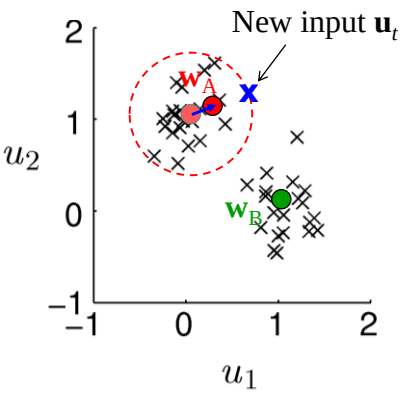


- Given new input, pick a cluster (weight vector closest to input)
- Set weight vector to **running average** of all inputs $\mathbf{u}_i$ *in that cluster*

$$\mathbf{w}_A(t) = \left(\sum_{i=1}^{t}\mathbf{u}_i\right)/t = \left(\sum_{i=1}^{t-1}\mathbf{u}_i + \mathbf{u}_t\right)/t$$
$$= \frac{(t-1)}{t}\mathbf{w}_A(t-1) + \frac{\mathbf{u}_t}{t}$$
$$= \mathbf{w}_A(t-1) + (1/t)(\mathbf{u}_t - \mathbf{w}_A(t-1))$$

$$\Delta\mathbf{w}_A = \varepsilon\cdot(\mathbf{u}_t - \mathbf{w}_A)$$

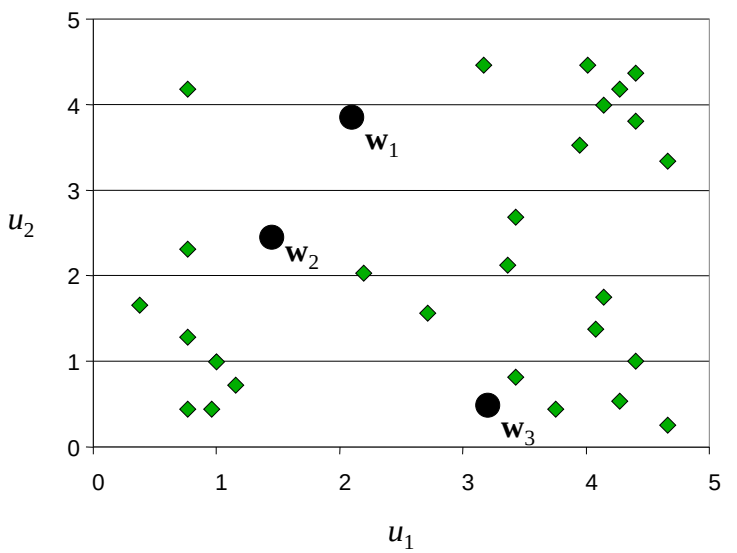
Competitive Learning



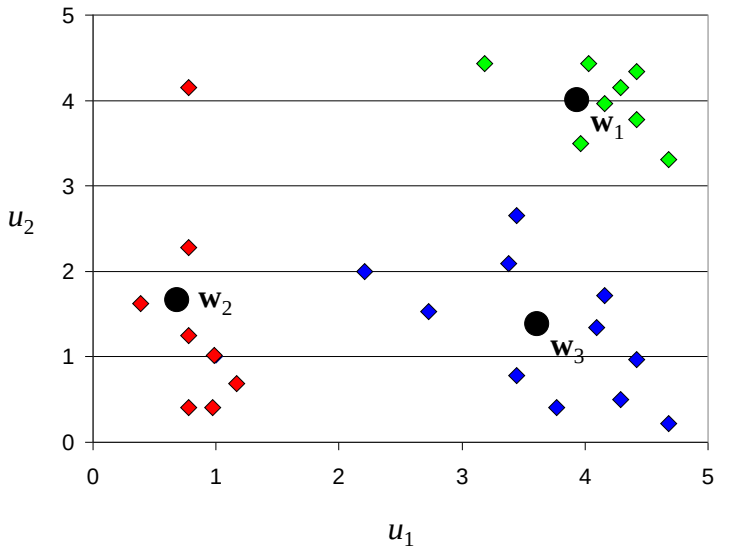
- Given a new input, pick the most active neuron ("winner-takes-all")
  ⇒ One whose weights are closest to new input
- Move weight vector for that neuron a bit closer to new input:

$$\Delta \mathbf{w} = \varepsilon \cdot (\mathbf{u}_t - \mathbf{w})$$

Competitive Learning Example: Initial Weights

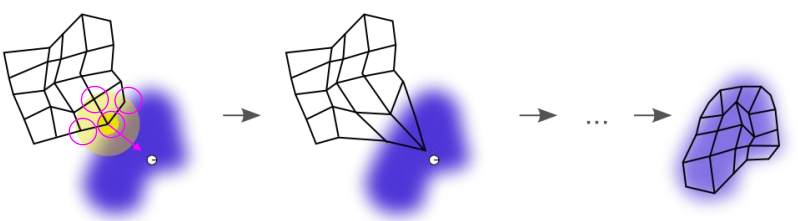


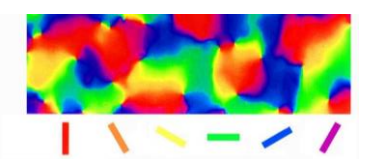
Competitive Learning Example: After Updates



Competitive Learning and Self Organizing Maps (a.k.a. Kohonen Maps)

- Given an input, pick the winning neuron
- Update weights for that neuron AND other neurons *in the neighborhood* of the winner

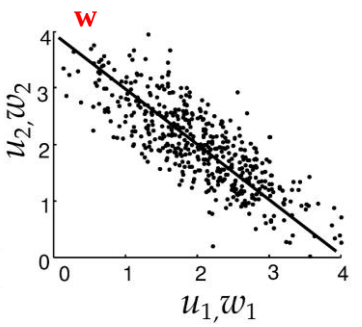




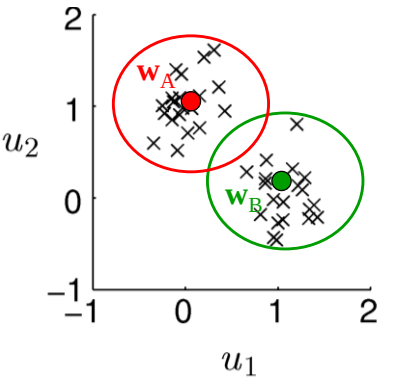
Orientation preference map in the primary visual cortex (V1)

Image Source: Wikimedia Commons

PCA/Hebb/Covariance rule



Competitive Learning



Is there a principled way of learning models of input data?

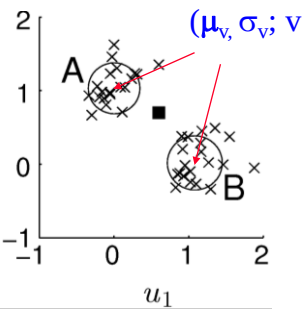
Image Source: Dayan & Abbott textbook

Enter…Unsupervised Learning

Causes v — Prior $p[v;G]$

Generative model

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



Mixture of Gaussians Model:

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

Parameters $G = (\mu_v, \sigma_v, \gamma_v)$ Gaussian Prior = γ_v

The Goal of Unsupervised Learning

Prior $p[\mathbf{v};G]$ — Causes **v**

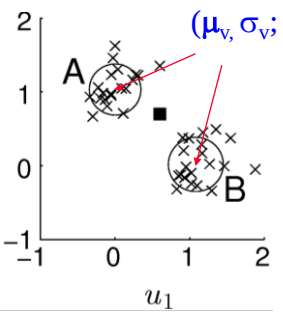
Generative model

Likelihood $p[\mathbf{u}\,|\,v;G]$ — Data **u**

Posterior $p[\mathbf{v}\,|\,\mathbf{u};G]$

- **Goal:** Learn a good "Generative Model" for the data you are seeing
  - Mimic the data generation process
- **General Approach:** Given data **u**, need to solve two sub-problems:
  - Estimate causes **v** (compute Posterior)
    - Same as "Recognition" problem
  - Learn parameters *G*

Clustering as Unsupervised Learning



$(\mu_v, \sigma_v;\ v = A, B)$

How do we learn the model parameters $G = (\mu_v, \sigma_v, \gamma_v)$?

Mixture of Gaussians Model:

$$p[\mathbf{u};G] = \sum_v p[\mathbf{u}\,|\,v;G]\,p[v;G]$$

Gaussian Prior = γ_v

EM algorithm for Unsupervised Learning

- Stands for Expectation-Maximization algorithm
- *Iterate* the following two steps until convergence:
  - E step: Compute posterior distribution of v (= A, B) for each $\mathbf{u}$:

$$p[v \mid \mathbf{u}; G] = \frac{p[\mathbf{u} \mid v; G]\,p[v; G]}{p[\mathbf{u}; G]} = \frac{N(\mathbf{u}; \boldsymbol{\mu}_v, \sigma_v I) \cdot \gamma_v}{\sum_v N(\mathbf{u}; \mu_v, \sigma_v I) \cdot \gamma_v}$$

(Bayes rule)

"Soft" competition (not winner-takes-all)

  - M step: Change parameters G using results from E step

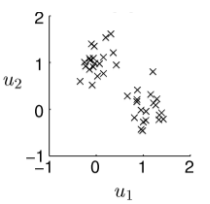
$$\boldsymbol{\mu}_v = \frac{\sum_{\mathbf{u}} p[v \mid \mathbf{u}; G] \cdot \mathbf{u}}{\sum_{\mathbf{u}} p[v \mid \mathbf{u}; G]}, \quad \sigma_v^{\,2} = \frac{\sum_{\mathbf{u}} p[v \mid \mathbf{u}; G]\,|\mathbf{u} - \boldsymbol{\mu}_v|^2}{\sum_{\mathbf{u}} p[v \mid \mathbf{u}; G]}$$

(Update parameters)

$$\gamma_v = \sum_{\mathbf{u}} p[v \mid \mathbf{u}; G] / N_u$$

Results from the EM algorithm

Input data:



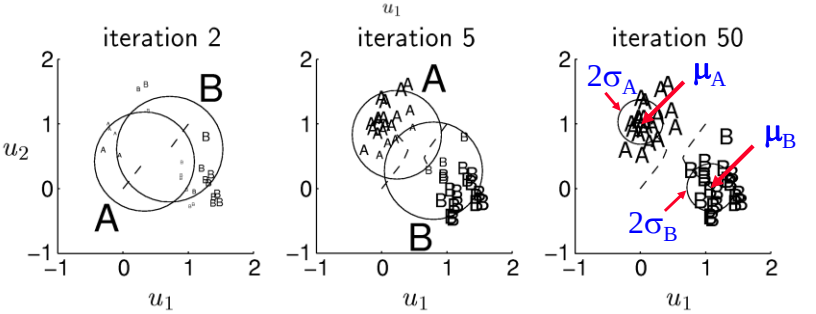
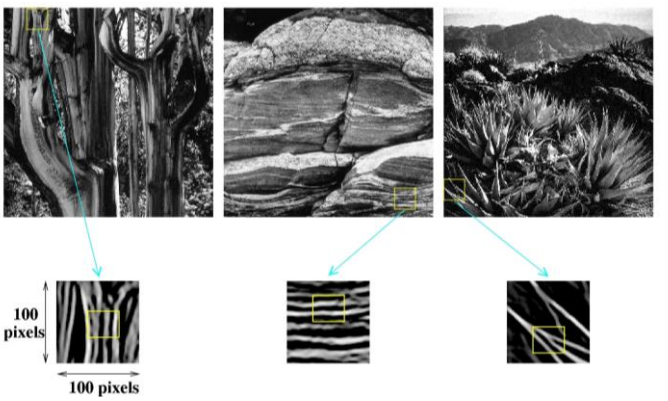


Image Source: Dayan & Abbott textbook

Suppose the Data are Natural Images…



Causes **v**

$p[\mathbf{v};G]$

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

Data **u**

What kind of generative model would you use?
How do you learn the "causes" of such images?

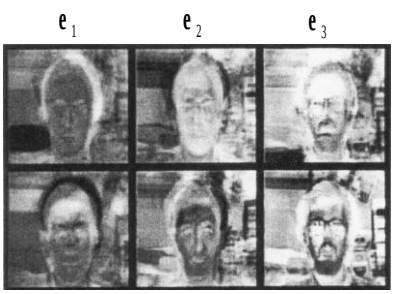
Image Source: Rao & Ballard, 1999

Eigenvectors strike again?

Input data **u**:
Face images
(*N* pixels total)

.....

Eigenvectors of the Input Covariance Matrix



.....

"Eigenfaces"

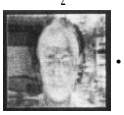
Image Source: Turk & Pentland, 1991

A Linear Model of Images using Eigenvectors

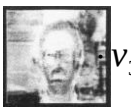
Input image **u**

$=$

$\cdot v_1 +$



$\cdot v_2 +$



$v_3 + \cdots$

$$\mathbf{u} = \sum_{i=1}^{N} \mathbf{e}_i v_i$$

Suppose you use only the first M principal eigenvectors:

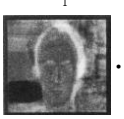
$$\mathbf{u} = \sum_{i=1}^{M} \mathbf{e}_i v_i + noise \qquad (M < N)$$

E.g., $M = 10, N = 10^6$ pixels

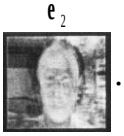
Not so fast, Eigenvectors!

Input image **u**

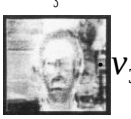
$=$



$\cdot v_1 +$



$\cdot v_2 +$



$v_3 + \cdots$

$$\mathbf{u} = \sum_{i=1}^{M} \mathbf{e}_i v_i + noise \qquad (M < N)$$

Eigenvector representation is good for compression but not so good if you want to extract the *local components* (or *parts*) of an image (e.g., parts of a face, local edges in a scene, etc.)

A Linear Model for Natural Images

Input image **u**

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

$$\mathbf{u} = \sum_{i=1}^{M} \mathbf{g}_i v_i + noise \qquad \text{(Note: } M \text{ can be larger than } N\text{)}$$

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

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

How do we learn $\mathbf{g}_i$ that are local components/parts of images?

Next Class: From Unsupervised to Supervised and Reinforcement Learning

- Things to do:
  - Read Chapter 9
  - Homework 4 due on Monday, February 27
  - Work on mini-project