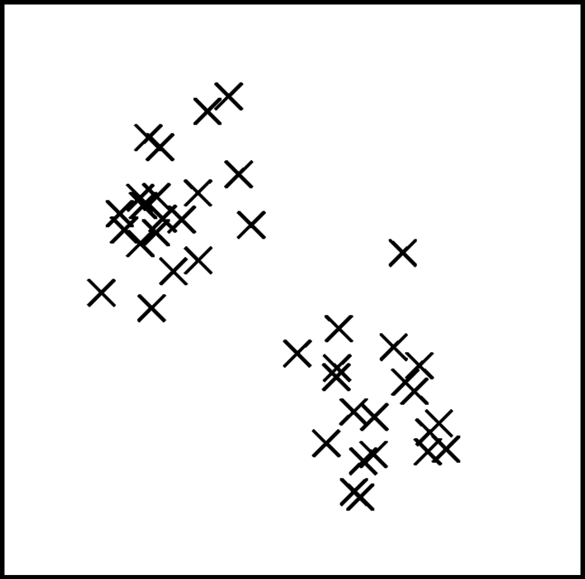
CSE/NB 528

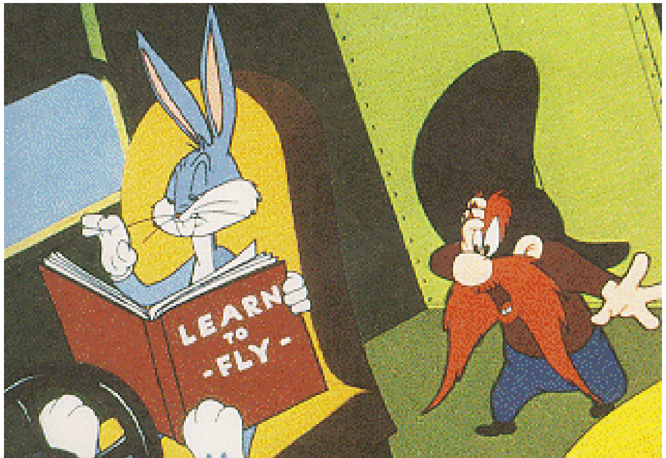
Lecture 12: Unsupervised Learning and Probability Density Estimation

(Chapters 8 & 10)



Today's Agenda: Learning about Learning

- Hebbian learning and its variants (Covariance, Oja rule)
  - Relation to Principal Component Analysis (PCA)
- Unsupervised Learning and Density Estimation
  - K-means Clustering and Mixture of Gaussians
  - EM algorithm



Flashback: Hebbian Learning

- Linear neuron: $v = \mathbf{w}^T \mathbf{u} = \mathbf{u}^T \mathbf{w}$
- Basic Hebb Rule: $\tau_w \dfrac{d\mathbf{w}}{dt} = \mathbf{u}v$ (or $\mathbf{w} \rightarrow \mathbf{w} + \varepsilon \cdot \mathbf{u}v$)
- What is the average effect of this rule?

$$\tau_w \frac{d\mathbf{w}}{dt} = \langle \mathbf{u}v \rangle_{\mathbf{u}} = \langle \mathbf{u}\mathbf{u}^T \mathbf{w} \rangle_{\mathbf{u}} = \langle \mathbf{u}\mathbf{u}^T \rangle_{\mathbf{u}} \mathbf{w} = Q\mathbf{w}$$

- Q is the input correlation matrix: $Q = \langle \mathbf{u}\mathbf{u}^T \rangle$

Variants of Hebb's Rule

- Hebb:

$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{u}v$$

Unstable

- Covariance rule:

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

Unstable

- Oja's rule:

$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{u}v - \alpha v^2 \mathbf{w}$$

Stable $\|\mathbf{w}\| \to \frac{1}{\sqrt{\alpha}}$

What does the Hebb rule do anyway?

Eigenvector analysis of Hebb rule…

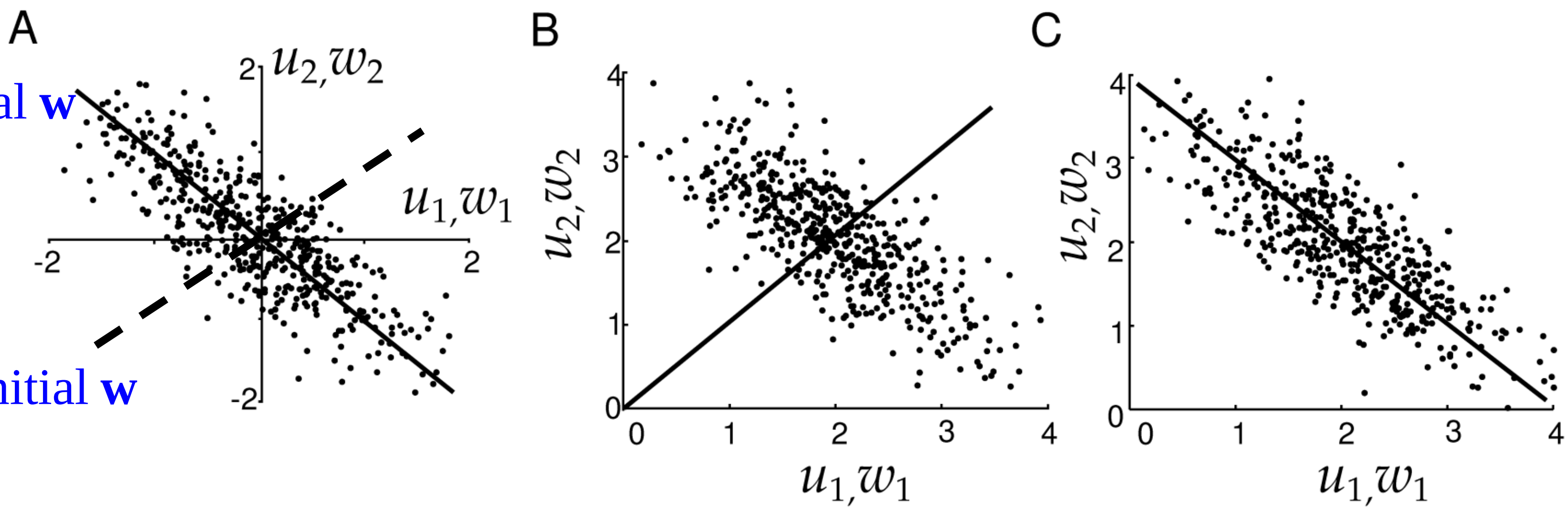
Hebb Rule implements Principal Component Analysis (PCA)!

Pure Hebb
Input mean = (0,0)

Pure Hebb
Input mean = (2,2)

Covariance Rule
Input mean = (2,2)

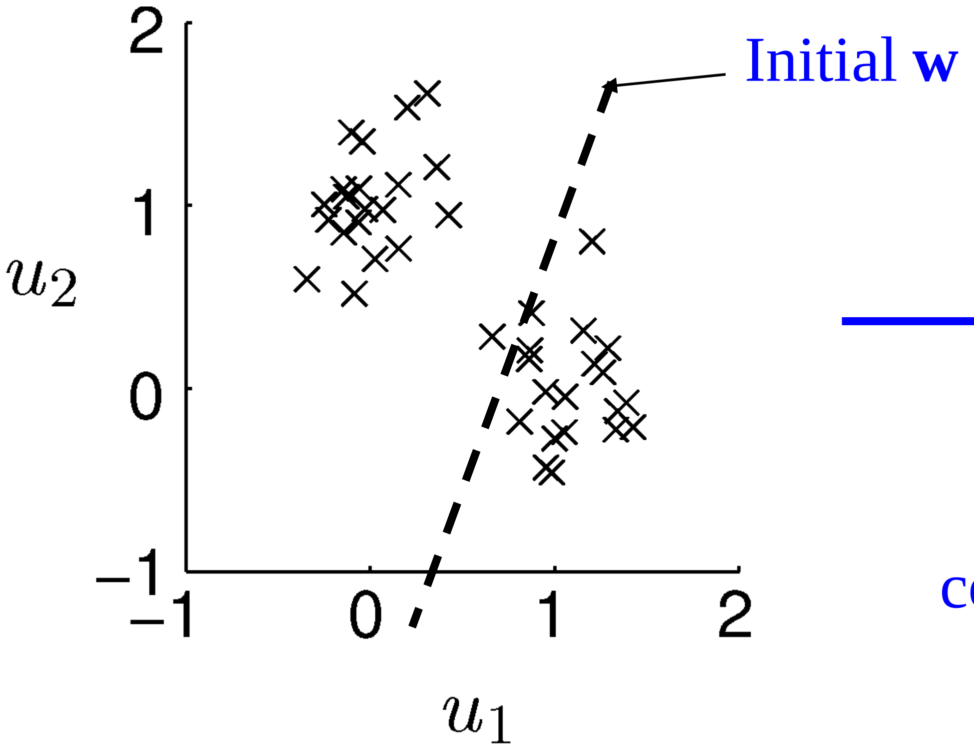
Final **w**



Initial **w**

Hebb rule *rotates* weight vector to align with principal eigenvector of input correlation/covariance matrix (i.e. direction of maximum variance)

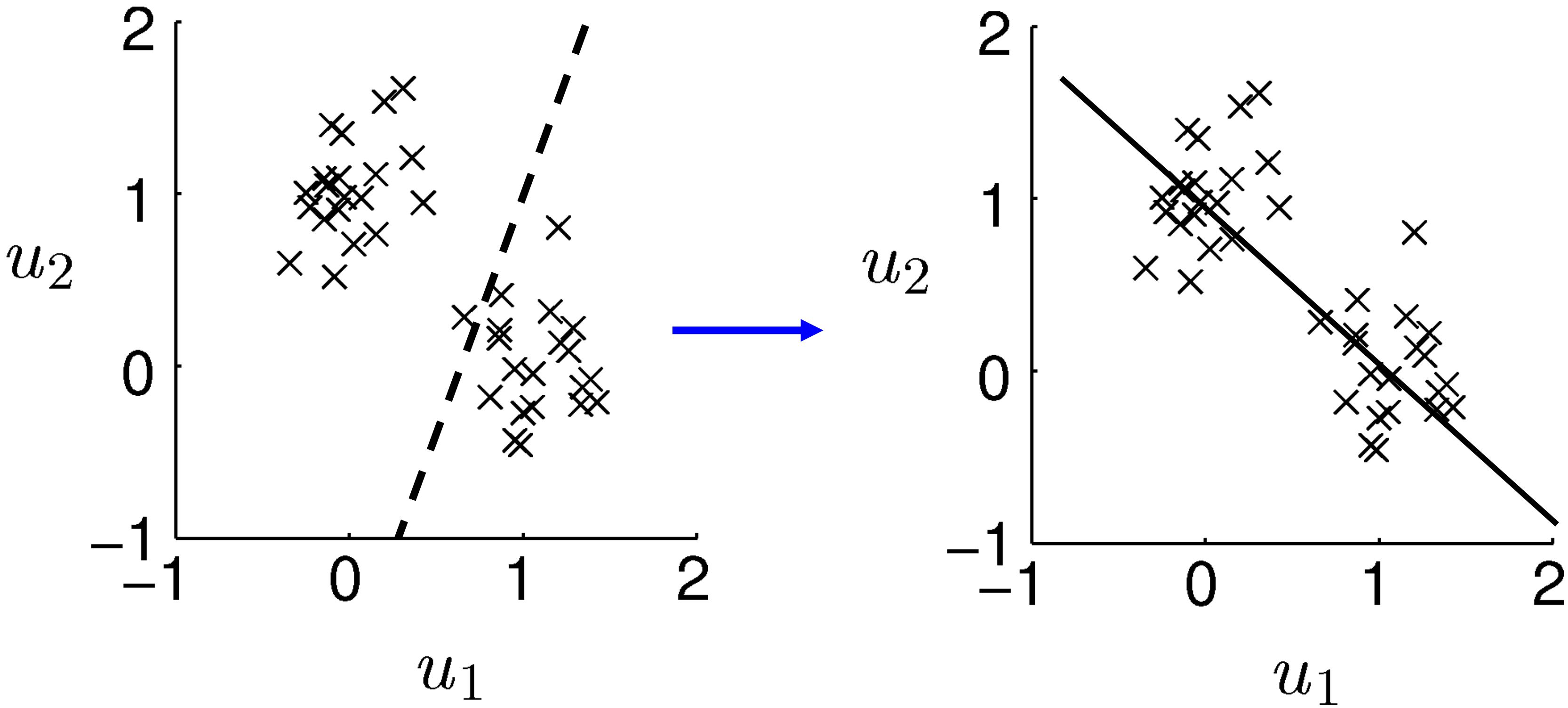
What about this data?



?

What does the covariance rule learn?

PCA does not correctly describe the data

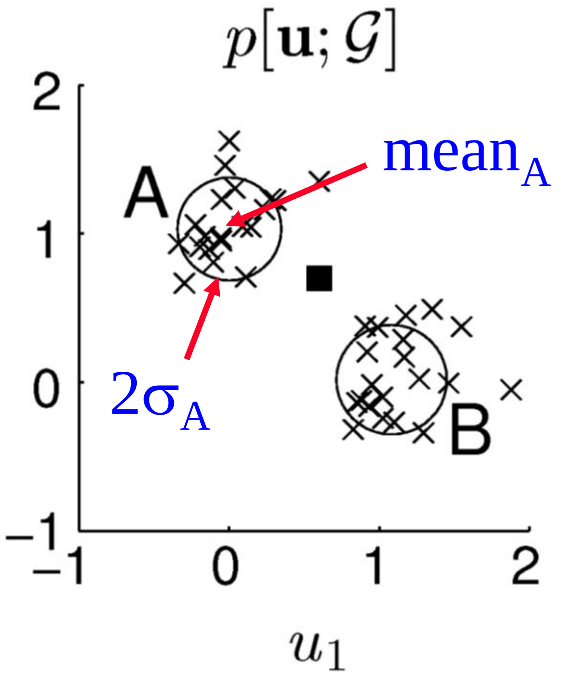


BUT…Input data is made up of two clusters (Gaussians)

→ two "causes"

Causal Models

- Main goal of unsupervised learning: Learn the "Causes" underlying the input data
- Example: Learn the means and variances of the two Gaussians A and B that generated this data
- Want: Two neurons A and B that learn the means and variances based solely on input data (which are samples from the distribution)



Generative versus Recognition Models

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

(posterior)

Generative model

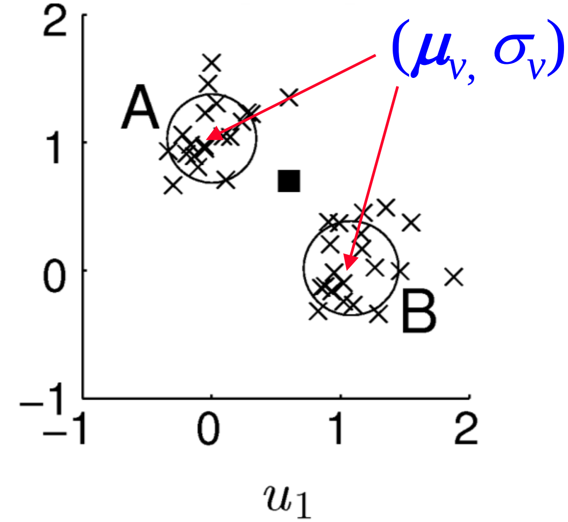
Recognition model

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

(likelihood)

(similar to decoding vs. encoding)

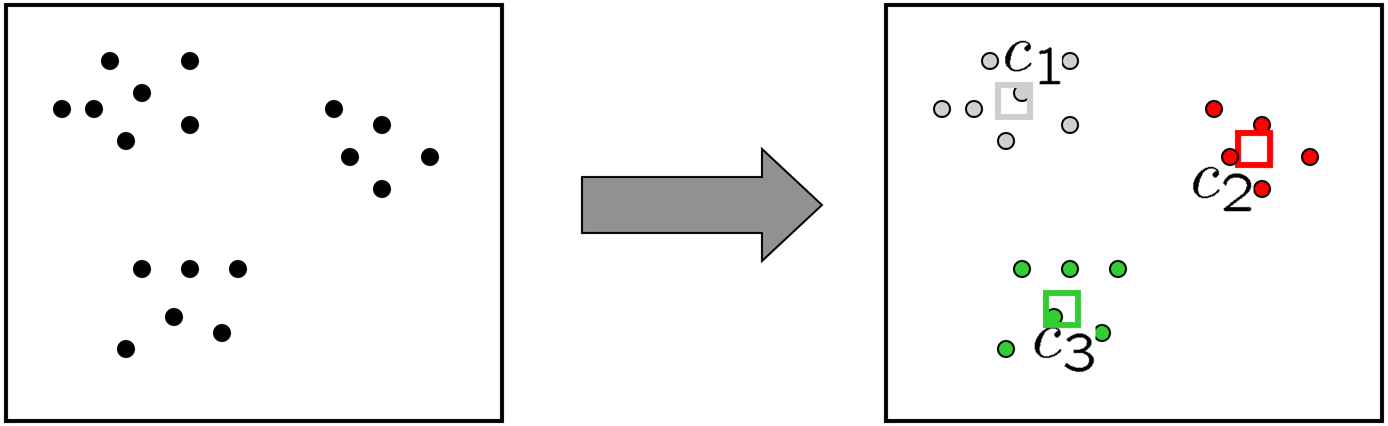
Parameters $G = (\boldsymbol{\mu}_v, \sigma_v, \gamma_v)$



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

Prior $= \gamma_v$

How do we learn the parameters (e.g., mean)?



Idea: Use one neuron to represent one cluster

Find cluster center (mean) by averaging all points in neuron's cluster

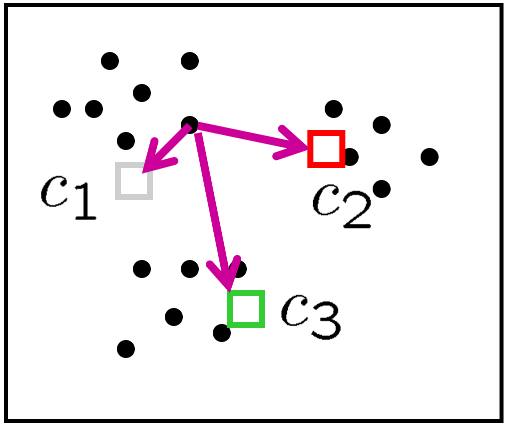
How do you find which point belongs to which cluster?

Break it down into 2 subproblems

Suppose you are given the cluster centers c_i

Q: how do you assign points to a cluster?

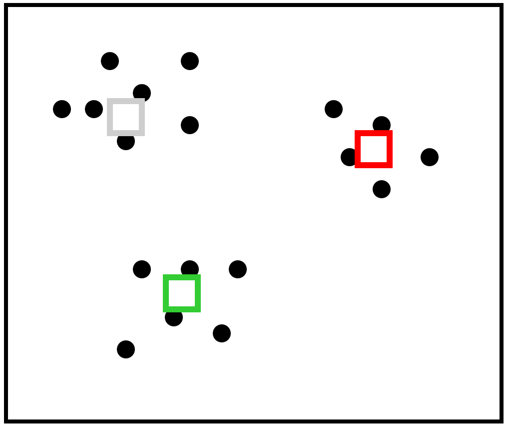
A: for each point p, choose closest c_i



Suppose you are given the points in each cluster

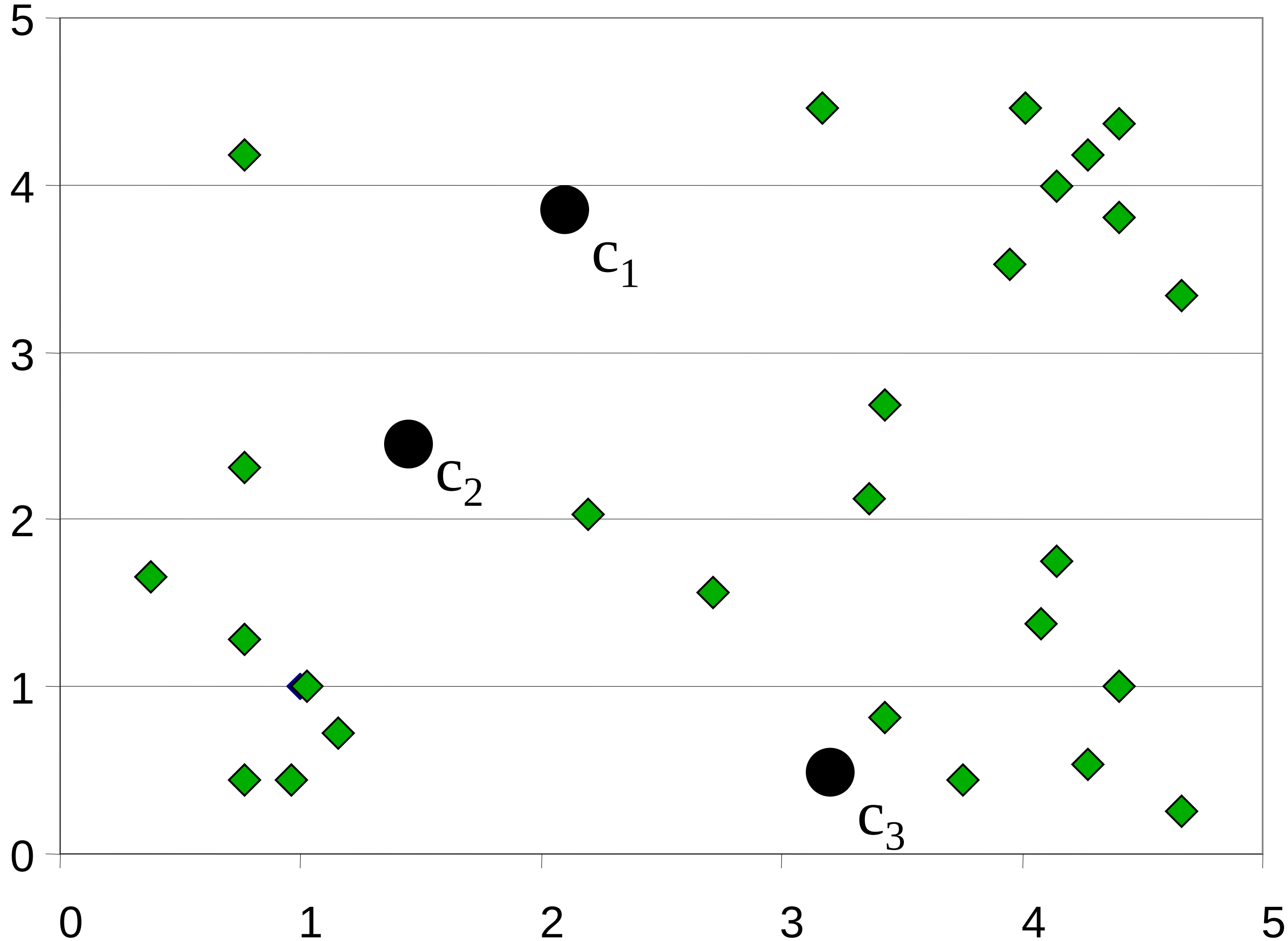
Q: how to re-compute each cluster's center?

A: choose c_i to be the mean of all the points in that cluster



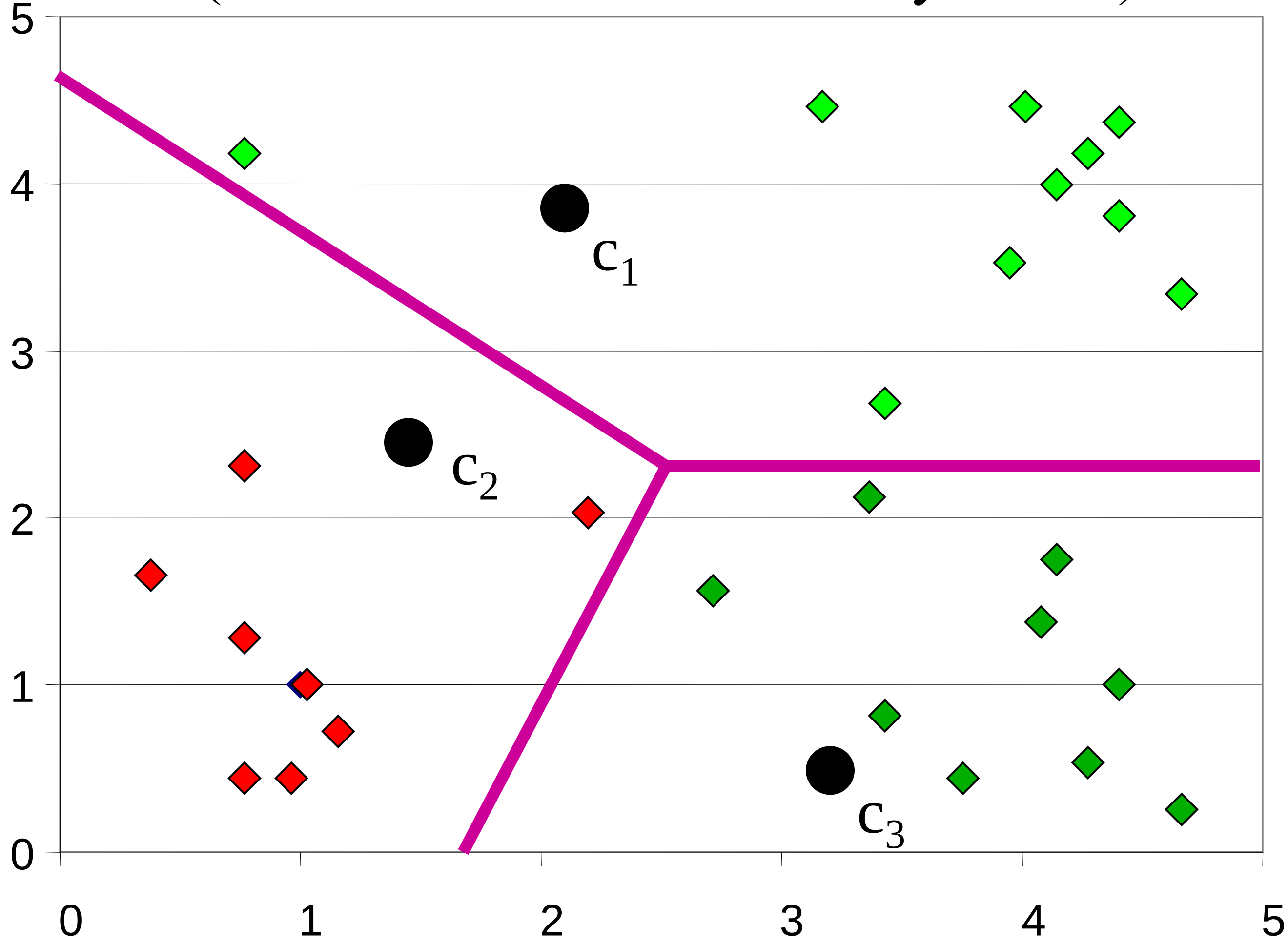
“K-means” clustering: Example

Randomly initialize the cluster centers (synaptic weights)



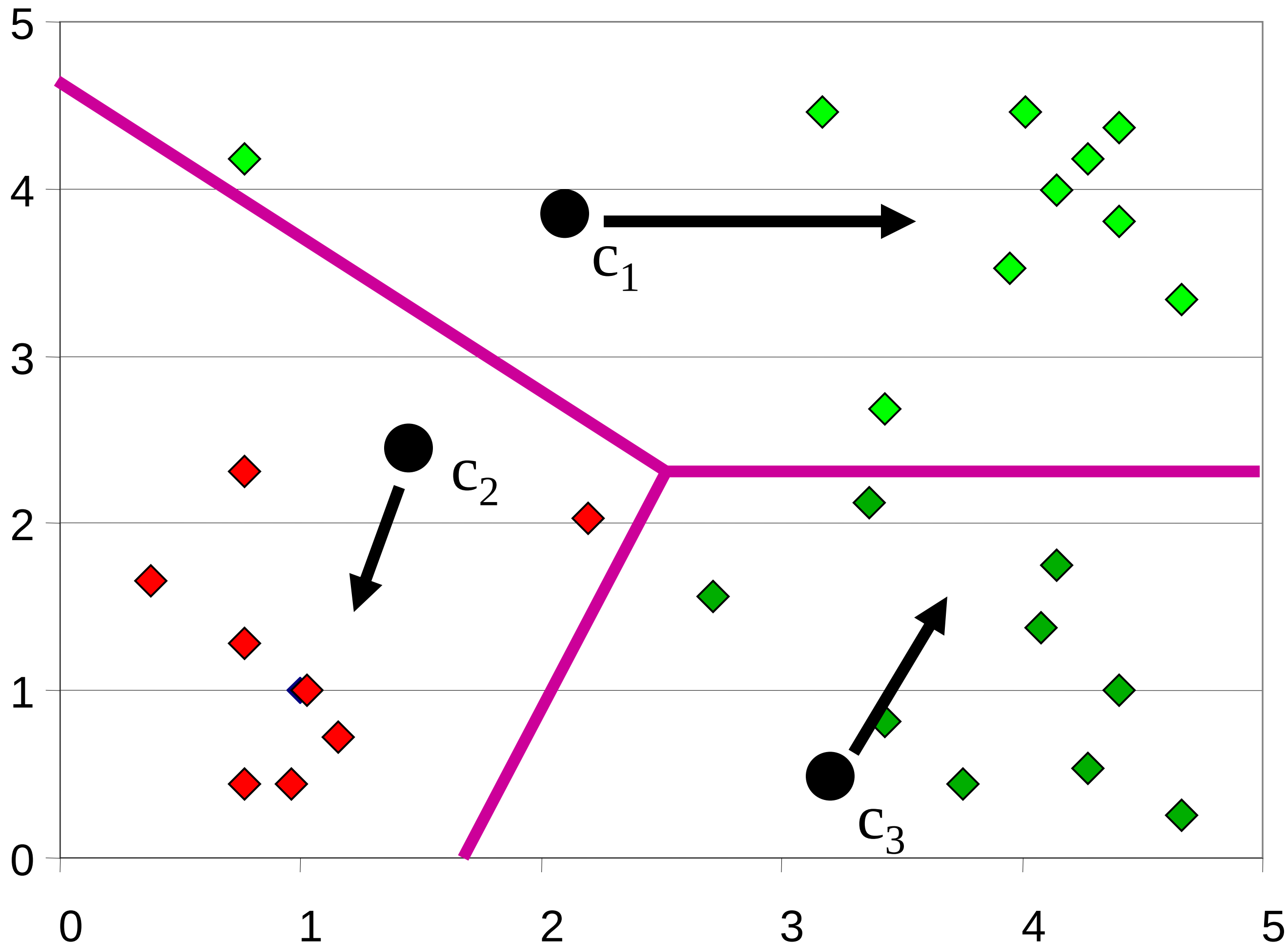
"K-means" clustering: Example

Determine cluster membership for each input
("winner-takes-all" inhibitory circuit)

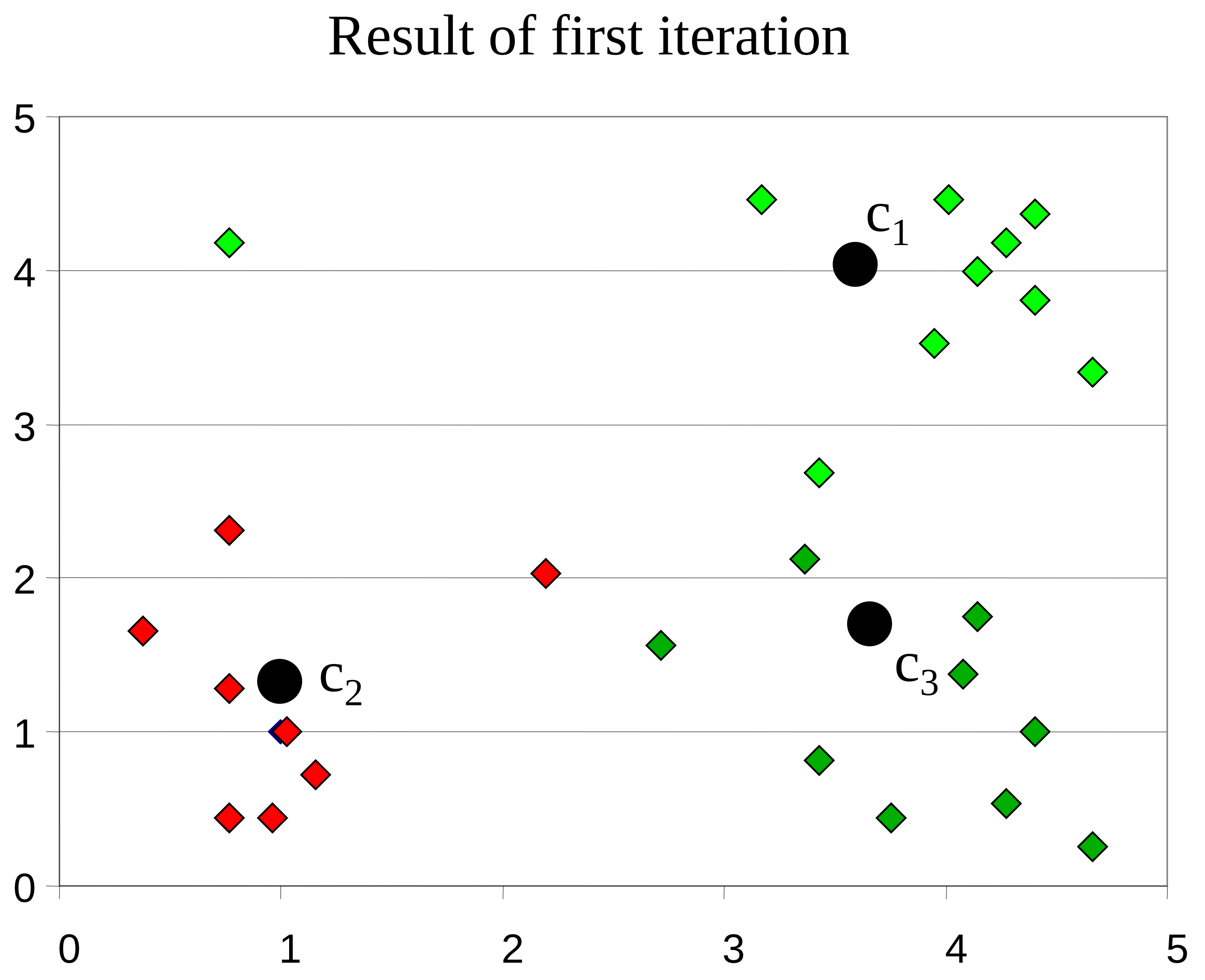


“K-means” clustering: Example

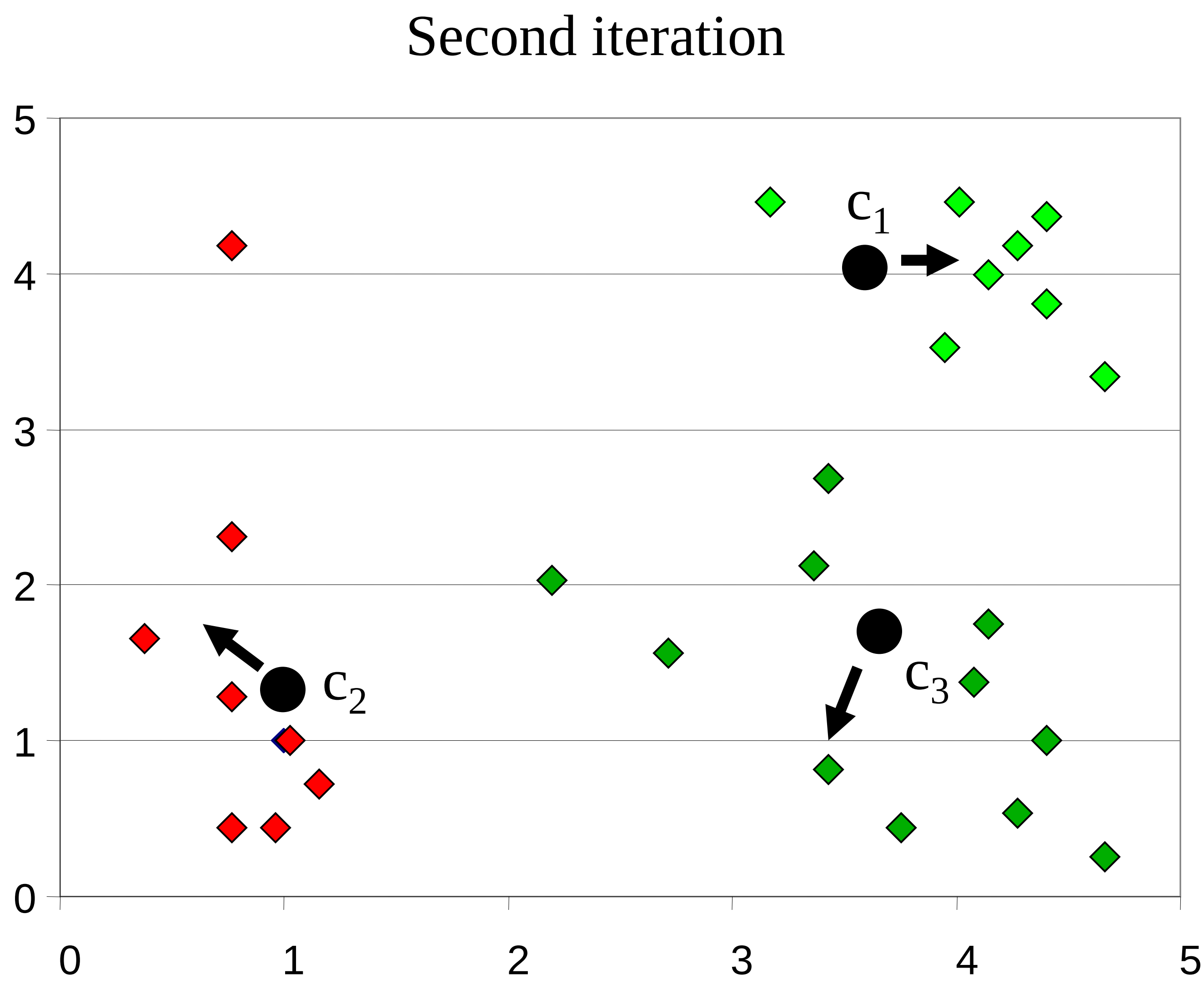
Re-estimate cluster centers (adapt synaptic weights)



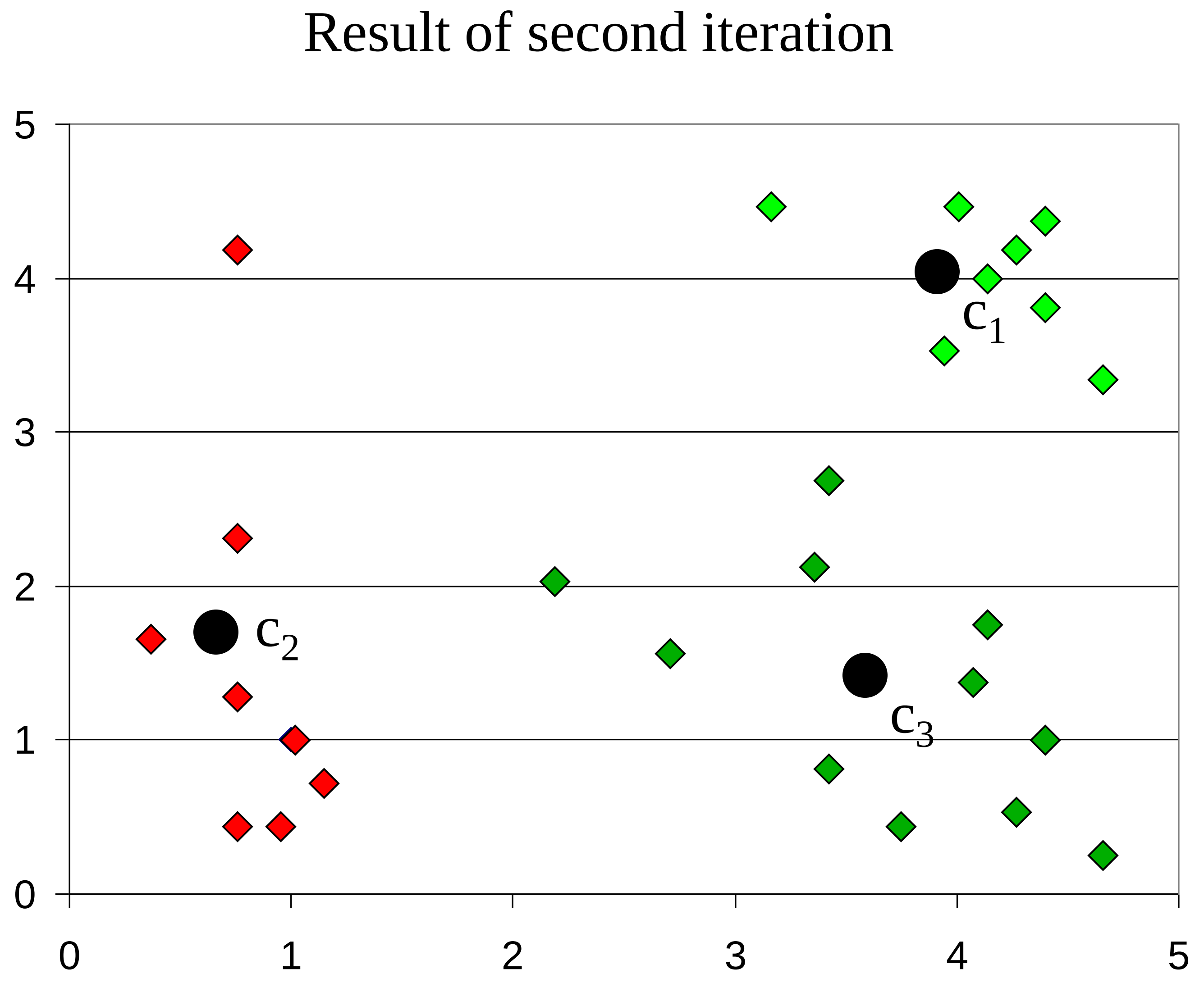
“K-means” clustering: Example



“K-means” clustering: Example



"K-means" clustering: Example



K-means clustering

- Properties
  - Will always converge to *some* solution
  - Can be a "local minimum"
    - does not always find the global minimum of the overall objective function:

$$\sum_{\text{clusters } i} \sum_{\text{points p in cluster } i} ||p - c_i||^2$$

K-means and probability density estimation

- Can formalize K-means as *probability density estimation*
- Model data as a **mixture of K Gaussians:**

$$p[\mathbf{u};G] = \sum_{j=1}^{K} p[\mathbf{u} \mid j;G]\, p[j;G]$$

- Estimate not only means but also covariances

K-means and the EM algorithm

Expectation Maximization (EM) Algorithm overview:

- Initialize K clusters: $C_1, \ldots, C_K$
  (μ_j, Σ_j) and $P(C_j)$ for each cluster j

1. Estimate which cluster each data point belongs to

   $$p(C_j \mid x_i)$$ ➡ Expectation step

2. Re-estimate cluster parameters

   $$(\mu_j, \Sigma_j), p(C_j)$$ ➡ Maximization step

3. Repeat 1 and 2 until convergence

EM algorithm for Mixture of Gaussians

- E step: Compute probability of membership in cluster based on output of previous M step ($p(x_i|C_j)$ = Gaussian(μ_j, Σ_j))

$$p(C_j \mid x_i) = \frac{p(x_i \mid C_j) \cdot p(C_j)}{p(x_i)} = \frac{p(x_i \mid C_j) \cdot p(C_j)}{\sum_j p(x_i \mid C_j) \cdot p(C_j)}$$

(Bayes rule)

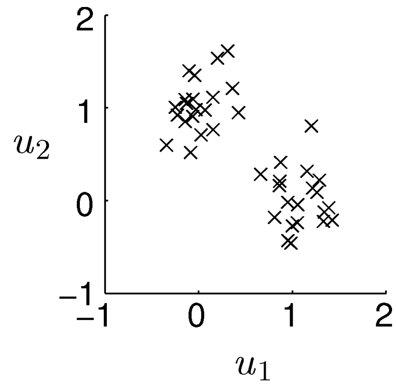
- M step: Re-estimate parameters based on output of E step

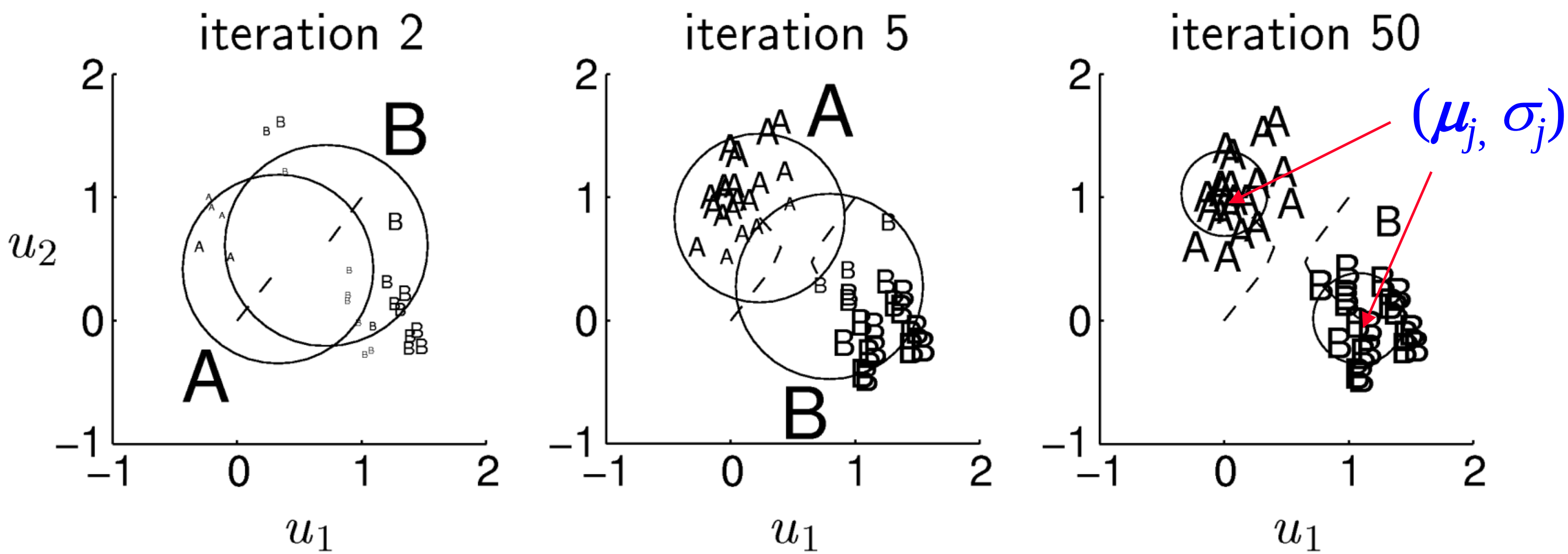
$$\mu_j = \frac{\sum_i p(C_j \mid x_i) \cdot x_i}{\sum_i p(C_j \mid x_i)} \qquad \Sigma_j = \frac{\sum_i p(C_j \mid x_i) \cdot (x_i - \mu_j) \cdot (x_i - \mu_j)^T}{\sum_i p(C_j \mid x_i)} \qquad p(C_j) = \frac{\sum_i p(C_j \mid x_i)}{N}$$

(Learn parameters)

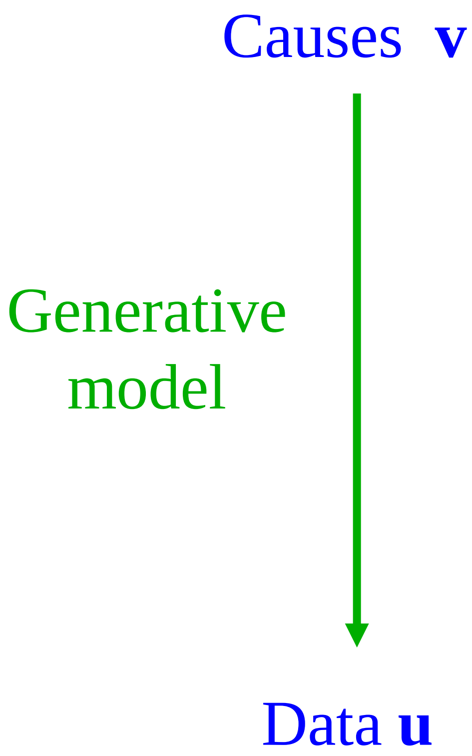
Results from the EM algorithm

Input data:





Recall: Generative Models



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

(likelihood)

Instead of clusters, what if data was generated by linear superposition of causes?
(e.g., an image composed of several features, or audio containing several voices)

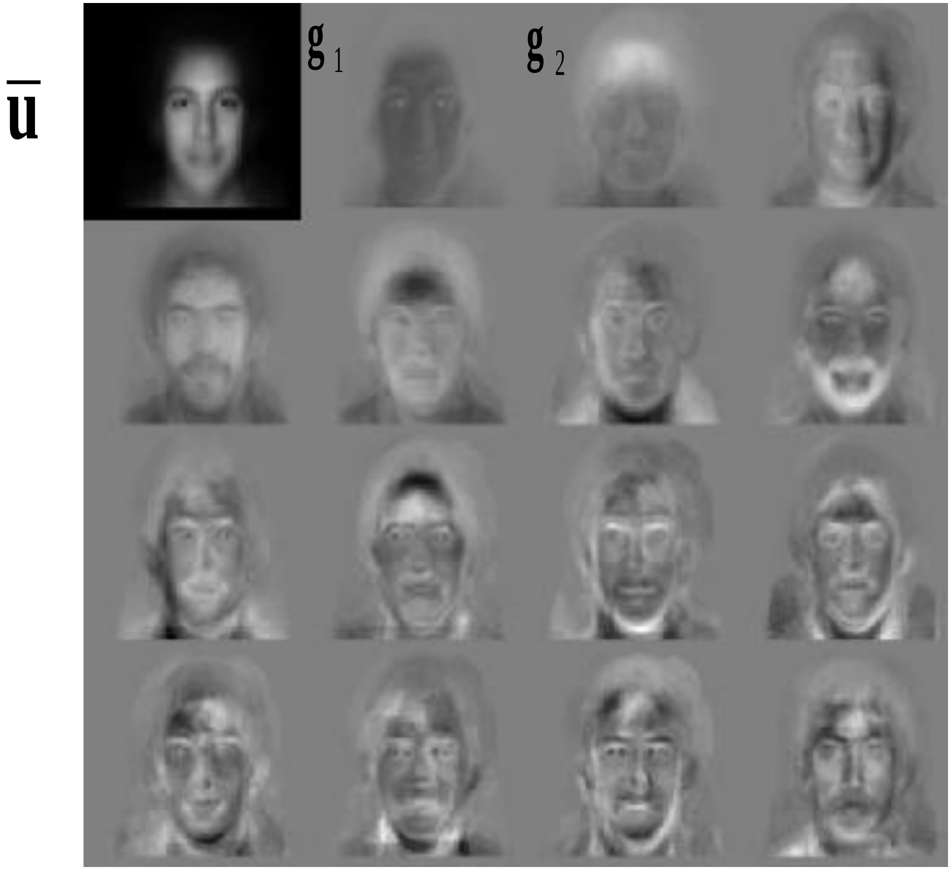
Linear Generative Model

- Suppose input **u** is represented by linear superposition of causes v_1, v_2, …, v_k and basis vectors (or "features") $\mathbf{g}_i$:

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

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

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

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

Linear Generative Model

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

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

- 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:

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

Probabilistic Interpretation

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

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

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

Next Class: More on Learning

- Unsupervised learning with linear models: Sparse coding, Predictive coding
- Supervised Learning
- Things to do:
  - Finish reading Chapters 8 and 10
  - Finish Homework #3 (due next Friday)
  - Work on mini-project

Have a great weekend!