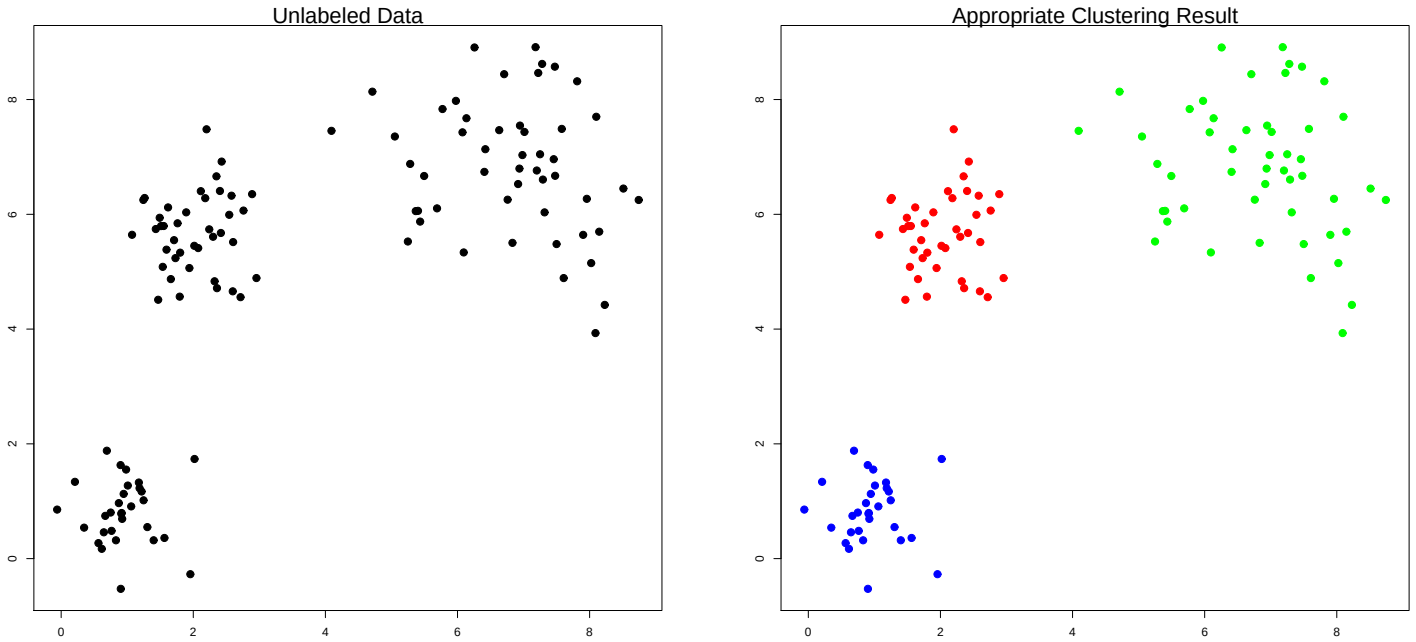
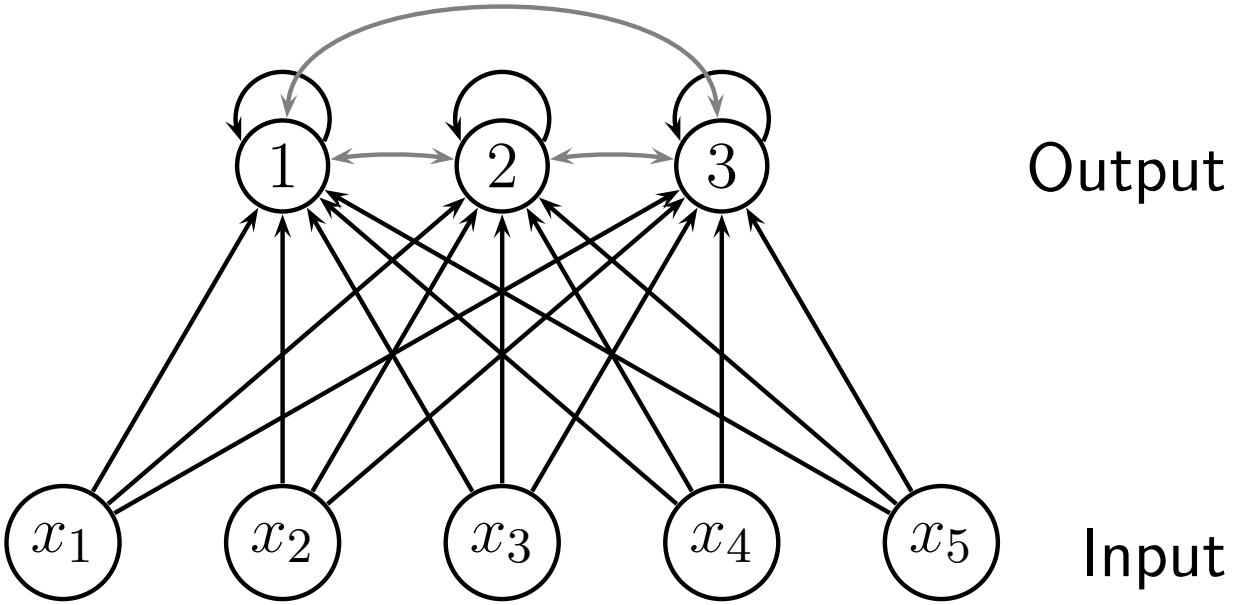
Clustering

Clustering is an *unsupervised* classification method, i.e. unlabeled data is partitioned into subsets (clusters), according to a similarity measure, such that "similar" data is grouped into the same cluster.



Objective: small inter-cluster distance and large distance between clusters.

Competetive Learning Network for Clustering



Gray colored connections are inhibitory; rest are excitatory. Only one of the output units, called the *winner*, can fire at a time. The output units compete for being the one to fire, and are therefore often called *winner-take-all* units.

Competetive Learning Network (cont.)

- Binary outputs, that is, winning unit i^* has output $O_{i^*} = 1$, rest zero
- Winner is unit with the largest net input

$$h_i = \sum_j w_{ij} x_j = \mathbf{w}_i^T x$$

for current input vector $\mathbf{x}$, hence,

$$\mathbf{w}_{i^*}^T \mathbf{x} \geq \mathbf{w}_i^T \mathbf{x} \quad \text{for all } i \tag{5}$$

- If weights for each unit are normalized ($\|\mathbf{w}_i\| = 1$) for all i, then (5) is equivalent to

$$\|\mathbf{w}_{i^*} - \mathbf{x}\| \leq \|\mathbf{w}_i - \mathbf{x}\| \quad \text{for all } i,$$

that is, winner is unit with normalized weight vector $\mathbf{w}$ closest to input vector $\mathbf{x}$

Competetive Learning Network (cont.)

- How to get it to find clusters in the input data and choose the weight vectors $\mathbf{w}_i$ accordingly?
- Start with small random values for the weights
- Present input patterns $\mathbf{x}^{(n)}$ in turn or in random order to the network
- For each input find the winner i^* among the outputs and then update weights w_{i^*j} for the winning unit only
- As a consequence $\mathbf{w}_{i^*}$ vector gets closer to current input vector $\mathbf{x}$ and makes the winning unit more likely to win on that input in the future

Obvious way to do this would be

$$\Delta w_{i^*j} = \eta x_j \qquad \text{problematic, why?}$$

Competitive Learning Rule

- Introduce normalization step: $w'_{i^*j} = \alpha w_{i^*j}$, choosing α so that $\sum_j w'_{i^*j} = 1$ or $\sum_j (w'_{i^*j})^2 = 1$
- Other approach (*standard competitive learning rule*)

$$\Delta w_{i^*j} = \eta(x_j - w_{i^*j})$$

  rule has the overall effect of moving the weight vector $\mathbf{w}_{i^*}$ of the winning unit toward the input pattern $\mathbf{x}$
- Because $O_{i^*} = 1$ and $O_i = 0$ for $i \neq i^*$ one can summarize the rule as follows:

$$\Delta w_{ij} = \eta O_i(x_j - w_{ij})$$

Competitive Learning Rule and Dead Units

Units with $\mathbf{w}_i$ which are far from any input vector may never win, and therefore never learn (dead units). There are different techniques to prevent the occurrence of dead units.

- Initialize weights to samples from the input itself (weights are all in the right domain)
- Update weights of all the losers as well as those of the winner but with a smaller learning rate η
- Subtract a threshold term μ_i from $h_i = \mathbf{w}_i^T \mathbf{x}$ and adjust the threshold to make it easier for frequently losing units to win. Units that win often should raise their μ_i's, while losers should lower them.

Cost Functions and Convergence

It would satisfiable to prove that competitive learning convergences to the "best" solution

- What is the best solution of a general clustering problem?

For the standard competitive learning rule $\Delta w_{i^*j} = \eta(x_j - w_{i^*j})$ there is an associated cost (Lyapunov) function:

$$E = \frac{1}{2}\sum_{i,j,n} M_i^{(n)}(x_j^{(n)} - w_{ij})^2 = \frac{1}{2}\sum_n \|\mathbf{x}^{(n)} - \mathbf{w}_{i^*}\|^2$$

$M_i^{(n)}$ is the *cluster membership matrix* which is specifies whether or not input pattern $\mathbf{x}^{(n)}$ activates unit i as winner:

$$M_i^{(n)} = \begin{cases} 1 & \text{if } i = i^*(n) \\ 0 & \text{otherwise} \end{cases}$$

Cost Functions and Convergence (cont.)

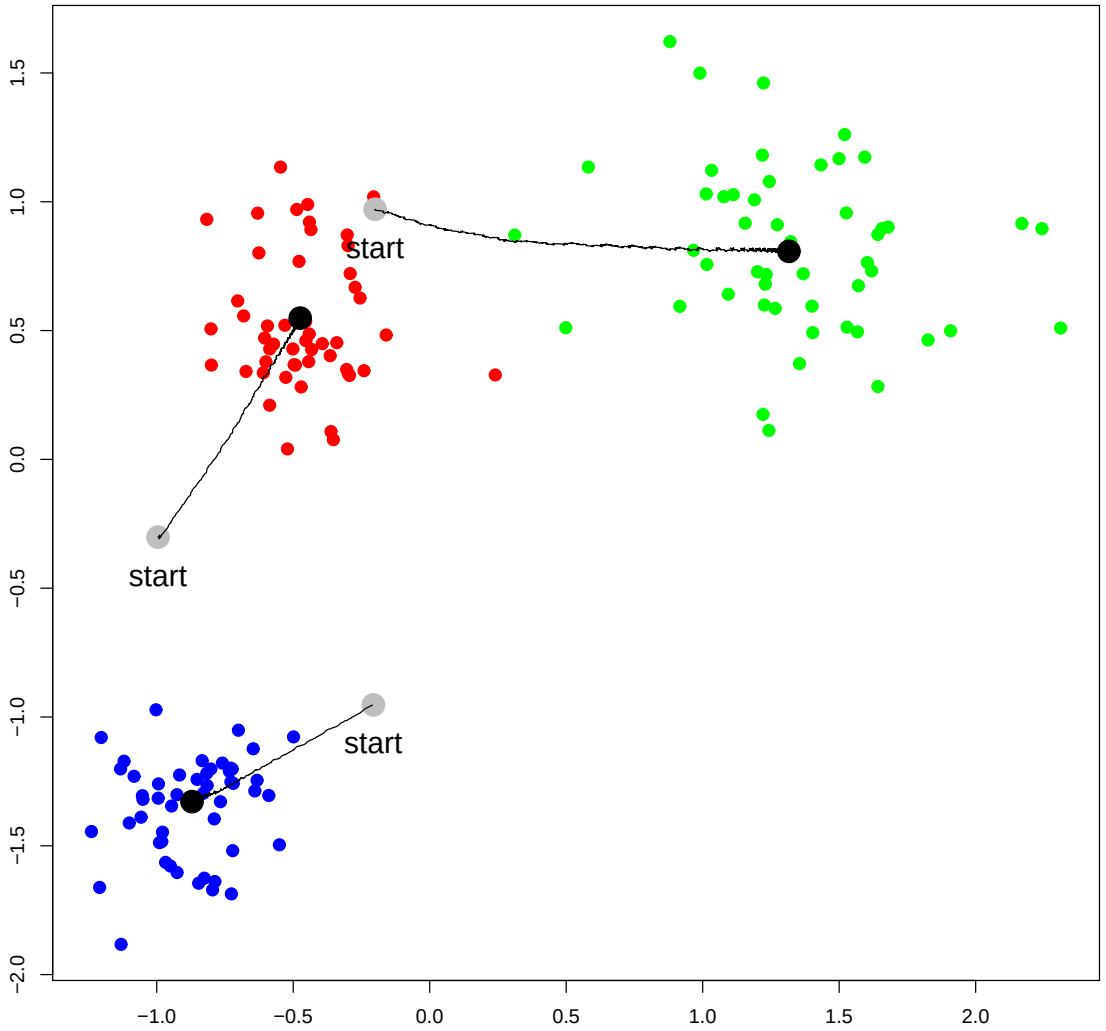
Gradient descent on the cost function yields

$$-\eta \frac{\partial E}{\partial w_{ij}} = \eta \sum_n M_i^{(n)} (x_j^{(n)} - w_{ij})$$

which is the sum of the standard rule over all the patterns n for which i is the winner.

- On average (for small enough η) the standard rule decreases the cost function until we reach a local minimum
- Update in batch mode by accumulating the changes in Δw_{ij}. This corresponds to K-Means clustering

Winner-Take-All Network Example



K-Means Clustering

- *Goal:* Partition data set $\{\mathbf{x}_t\}_{t=1}^N \in \mathbb{R}^d$ into some number K of clusters.
- *Objective function:* Distances within a cluster are small compared with distances to points outside of the cluster.

Let $\boldsymbol{\mu}_k \in \mathbb{R}^d$ where $k = 1, 2, \ldots, K$ represents a prototype which is associated with the k^{th} cluster. For each data point $\mathbf{x}_t$ exists a corresponding set of indicator variabes $r_{tk} \in \{0, 1\}$. If $\mathbf{x}_t$ is assigned to cluster k then $r_{tk} = 1$, otherwise $r_{tj} = 0$ for $j \neq k$.

- *Goal more formally:* Find values for the $\{r_{tk}\}$ and the $\{\boldsymbol{\mu}_k\}$ so as to minimize

$$J = \sum_{t=1}^{N} \sum_{k=1}^{K} r_{tk} \|\mathbf{x}_t - \boldsymbol{\mu}_k\|^2$$

K-Means Clustering (cont.)

J can be minimized in a two-step approach.

- Step 1: Determine responsibilities

$$r_{tk} = \begin{cases} 1 & \text{if } k = \text{argmin}_j \|\mathbf{x}_t - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise} \end{cases}$$

  in other words, assign the t^{th} data point to the closest cluster center $\boldsymbol{\mu}_j$.

- Step 2: Recompute (update) the cluster means $\boldsymbol{\mu}_j$

$$\boldsymbol{\mu}_j = \frac{\sum_t r_{tk}\mathbf{x}_t}{\sum_t r_{tk}}$$

Repeat step 1 and 2 until there is no further change in responsibilities or max. number of iterations is reached.

K-Means Clustering (cont.)

In step 1, we minimize J with respect to the r_{tk}, keeping $\boldsymbol{\mu}_k$ fixed.

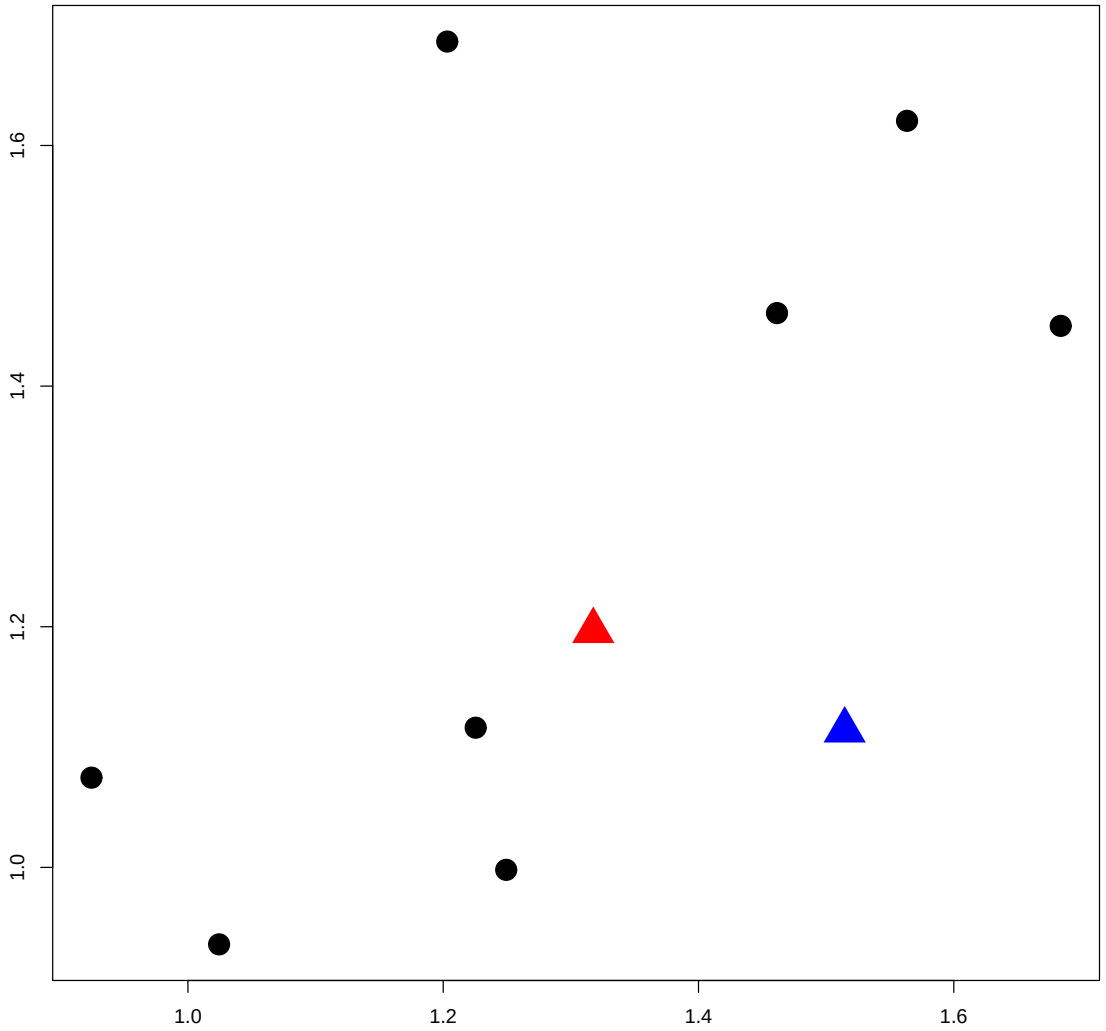
In step 2, we minimize J with respect to the $\boldsymbol{\mu}_k$, keeping the r_{tk} fixed.

Let's look closer at step 2. J is a quadratic function of $\boldsymbol{\mu}_k$ and it can be minimized by setting its derivative with respect to $\boldsymbol{\mu}_k$ to zero.

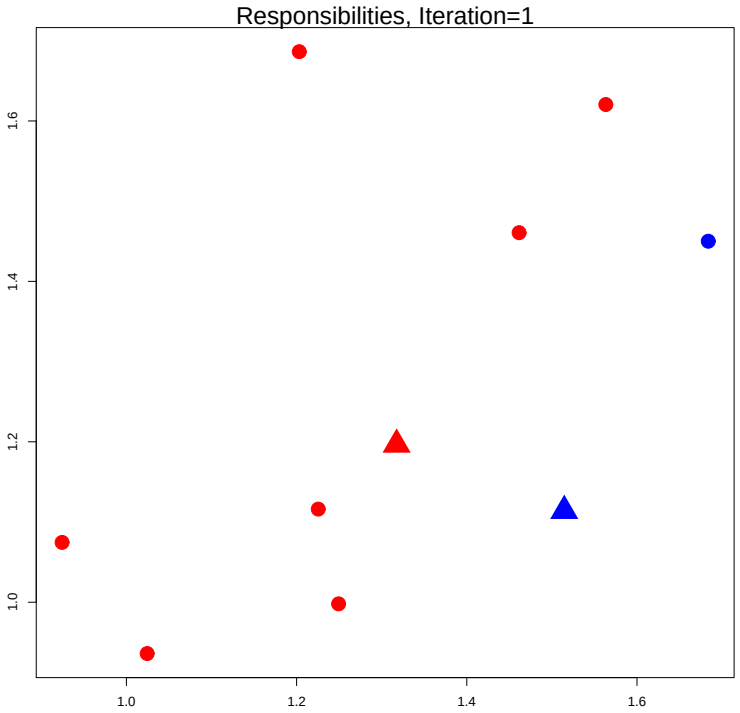
$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \sum_{t=1}^{N} \sum_{k=1}^{K} r_{tk} \|\mathbf{x}_t - \boldsymbol{\mu}_k\|^2 = 2 \sum_{t=1}^{N} r_{tk} (\mathbf{x}_t - \boldsymbol{\mu}_k)$$

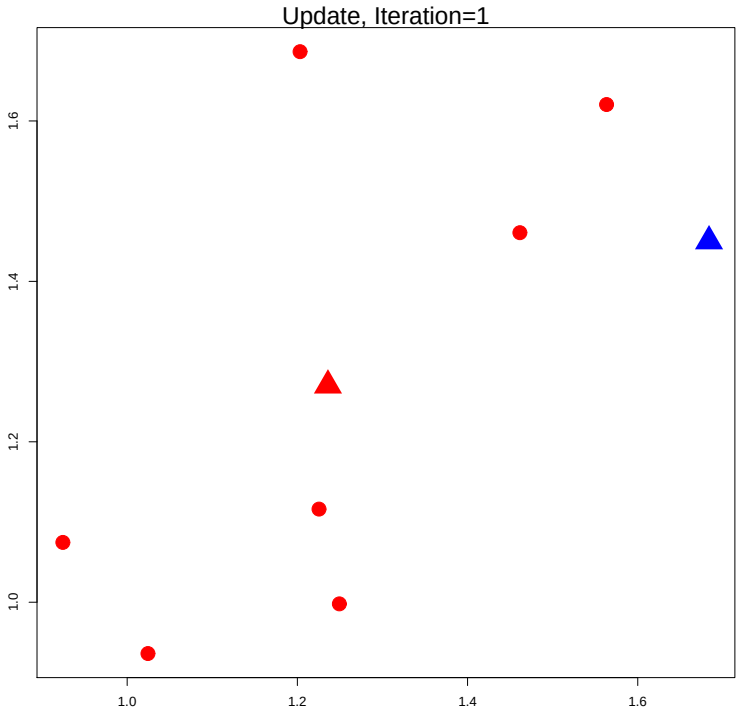
$$0 = 2 \sum_{t=1}^{N} r_{tk} (\mathbf{x}_t - \boldsymbol{\mu}_k) \Leftrightarrow \boldsymbol{\mu}_k = \frac{\sum_t r_{tk} \mathbf{x}_t}{\sum_t r_{tk}}$$

K-Means Clustering Example



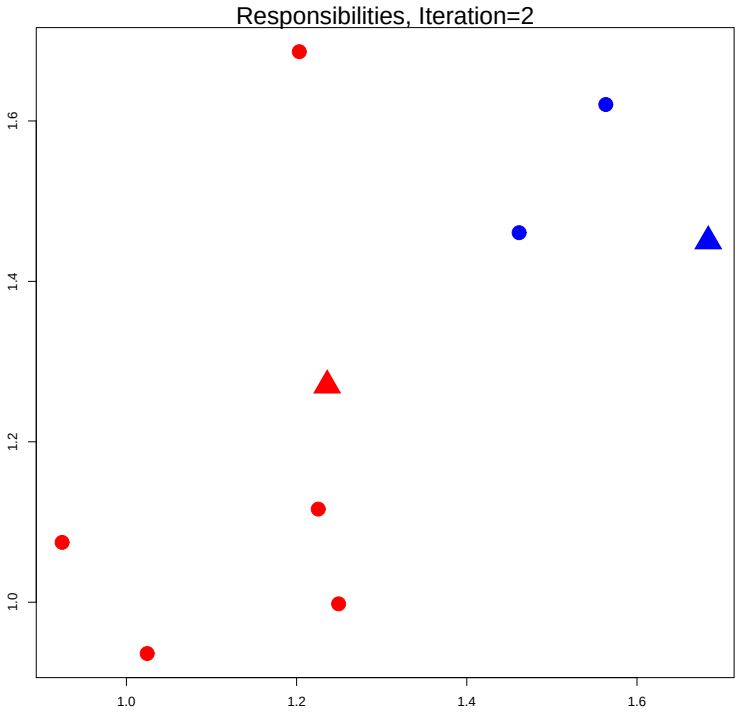
K-Means Clustering Example (cont.)

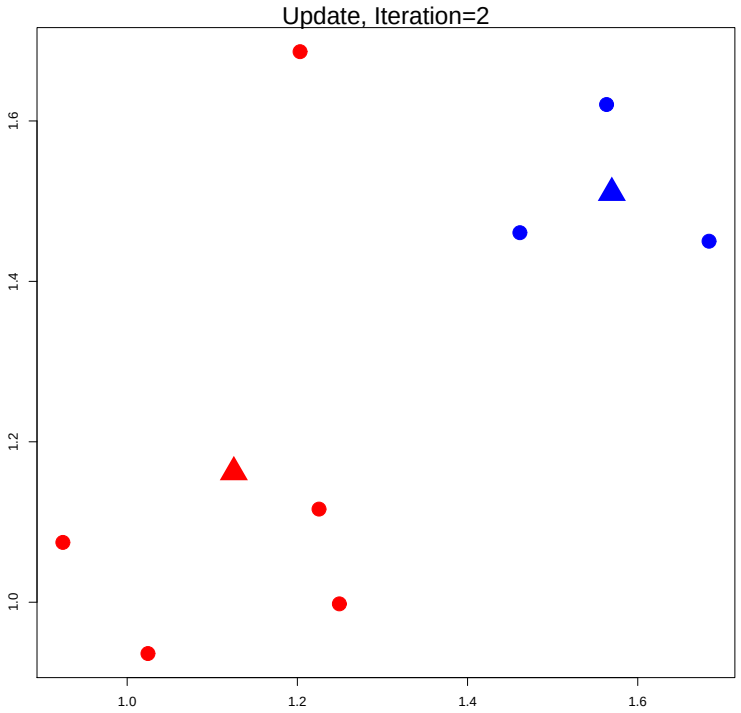




$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}^T = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ \vdots & \vdots \\ r_{81} & r_{82} \end{bmatrix}$$

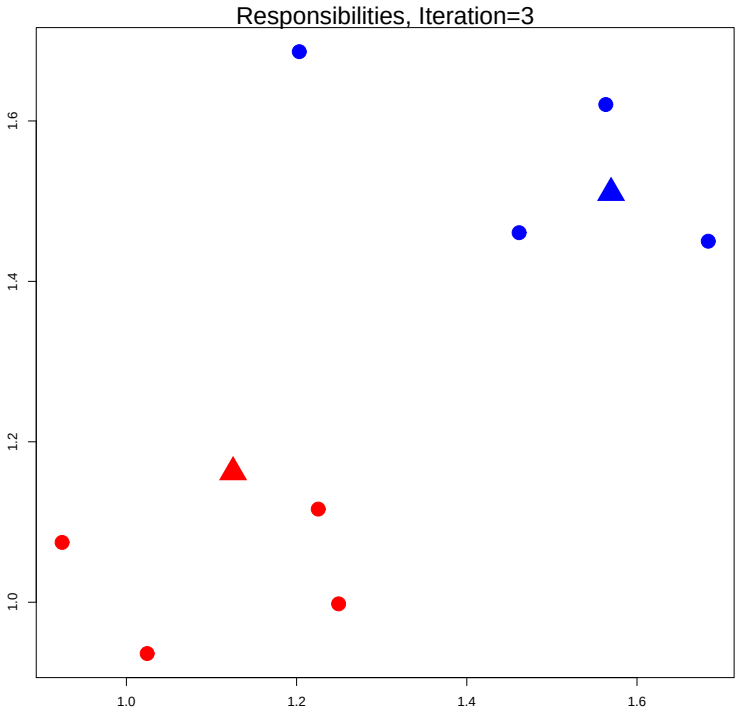
K-Means Clustering Example (cont.)

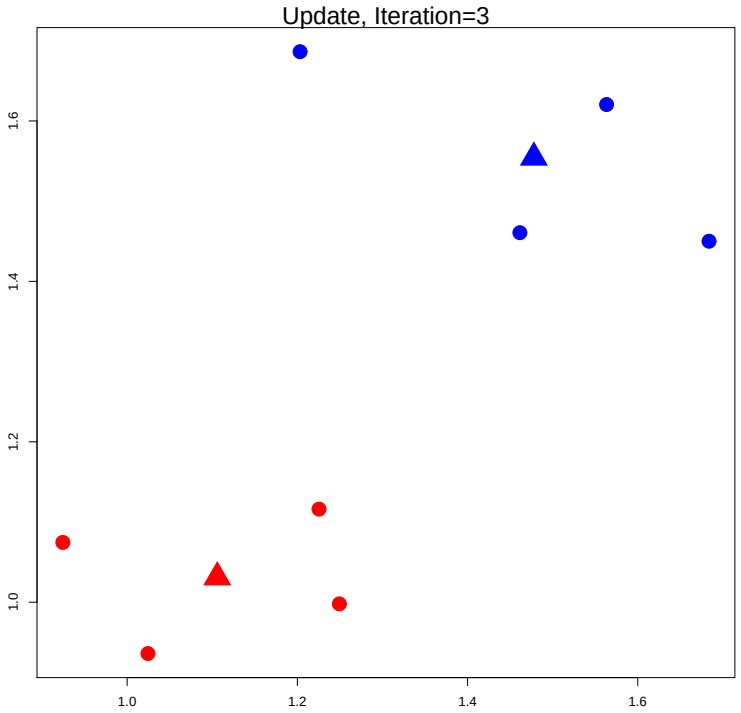




$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}^T$$

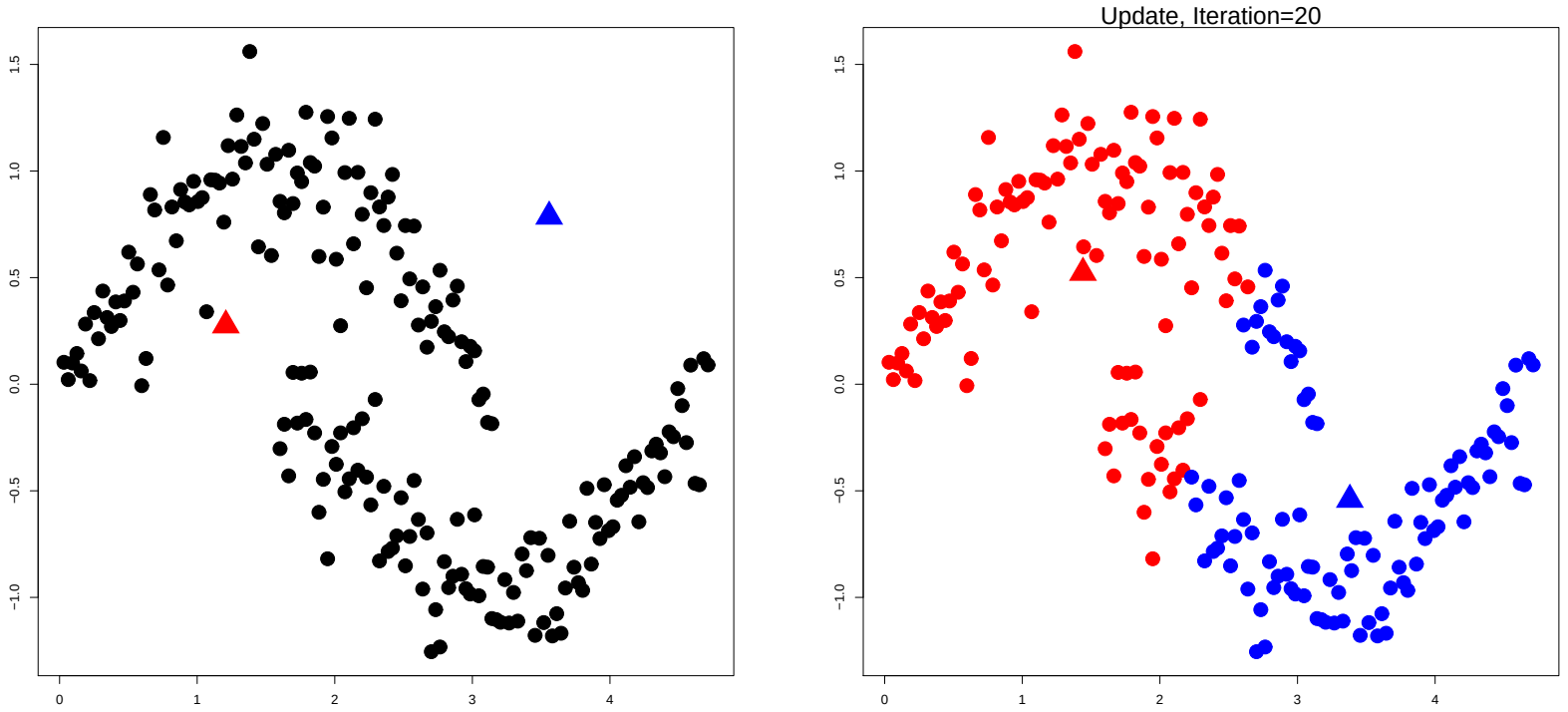
K-Means Clustering Example (cont.)





$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

K-Means Clustering Example (cont.)



K-Means final solution depends largely on the initialized starting values and is not guaranteed to return a global optimum.

K-Means Clustering in R

```
library(cclust)

## cluster 1 ##
x1 <- rnorm(30,1,0.5);
y1 <- rnorm(30,1,0.5);

## cluster 2 ##
x2 <- rnorm(40,2,0.5);
y2 <- rnorm(40,6,0.7);

## cluster 3 ##
x3 <- rnorm(50,7,1);
y3 <- rnorm(50,7,1);

d <- rbind(cbind(x1,y1),cbind(x2,y2),cbind(x3,y3));
typ <- c(rep("4",30),rep("2",40),rep("3",50));
data <- data.frame(d,typ);

# lets viz. it
plot(data$x1, data$y1, col=as.vector(data$typ));
```

K-Means Clustering in R

```
# perform k-means clustering
k <- 3;
iter <- 100;
which.distance <- "euclidean";
# which.distance <- "manhattan";
kmclust <- cclust(d,k,iter.max=iter,method="kmeans",dist=which.distance);

# print coord. of init. cluster centers
print(kmclust$initcenters);

# print coord. of final cluster centers
print(kmclust$centers);

# lets vis. it; kmclust$cluster gives assigned cluster class of each point
# e.g. [1,1,2,2,3,1,3,3]
plot(data$x1, data$y1, col=(kmclust$cluster+1));
points(kmclust$centers, col=seq(1:kmclust$ncenters)+1, cex=3.5, pch=17);
```

Kohonen's Self-Organized Map (SOM)

Goal: discover underlying structure of the data

- Winner-Take-All neural network ignored the geometrical arrangements of output units
- Idea: output units that are close together are going to interact differently than output units that are far apart

Output units O_i are arranged in an array (generally one- or two-dimensional), and are fully connected via w_{ij} to the input units.

- Similar to the Winner-Take-All rule, the winner i^* is chosen as the output unit with weight vector closest to current input $\mathbf{x}$

$$\|\mathbf{w}_{i^*} - \mathbf{x}\| \leq \|\mathbf{w}_i - \mathbf{x}\| \qquad \text{for all } i$$

Note, this cannot not be done by a linear network unless the weights are normalized

Kohonen's Self-Organized Map (SOM) (cont.)

Learning rule:

$$\Delta w_{ij} = \eta \Lambda(i, i^*)(x_j - w_{i^* j}) \qquad \text{for all } i, j$$

The *neighborhood function* $\Lambda(i, i^*)$ is 1 for $i = i^*$ and falls off with distance $\|r_i - r_{i^*}\|$ between units i and i^* in the output array.

- Typical choice for $\Lambda(i, i^*)$ is

$$\Lambda(i, i^*) = \exp(-\|r_i - r_{i^*}\|^2 / 2\sigma^2)$$

Nearby units receive similar updates and thus end up responding to nearby input patterns.

The update rule drags the weight vector $\mathbf{w}_{i^*}$ belonging to the winner towards $\mathbf{x}$. However, it also drags the $\mathbf{w}_i$'s of the closest units along with it.

SOM Example