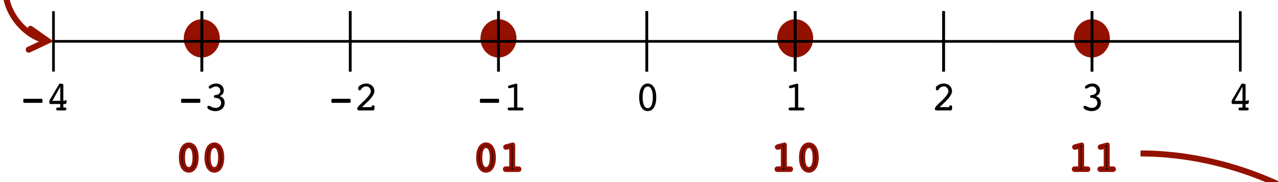
Learning Vector Quantization (LVQ) – Neural Network

Paulo Trigo Silva @ ISEL

About the “vector quantization” concept

a “vector quantization” (VQ) is nothing more than an approximator; the idea is similar to that of “rounding-off” (e.g., to the nearest integer); the original motivation is *dimensionality reduction* or *data compression*

a VQ example of a *1-dimensional* (1D) space



each number in [–4 .. –2[is approximated by –3

each number in [–2 .. 0[is approximated by –1

each number in [0 .. 2[is approximated by 1

each number in [2 .. 4[is approximated by 3

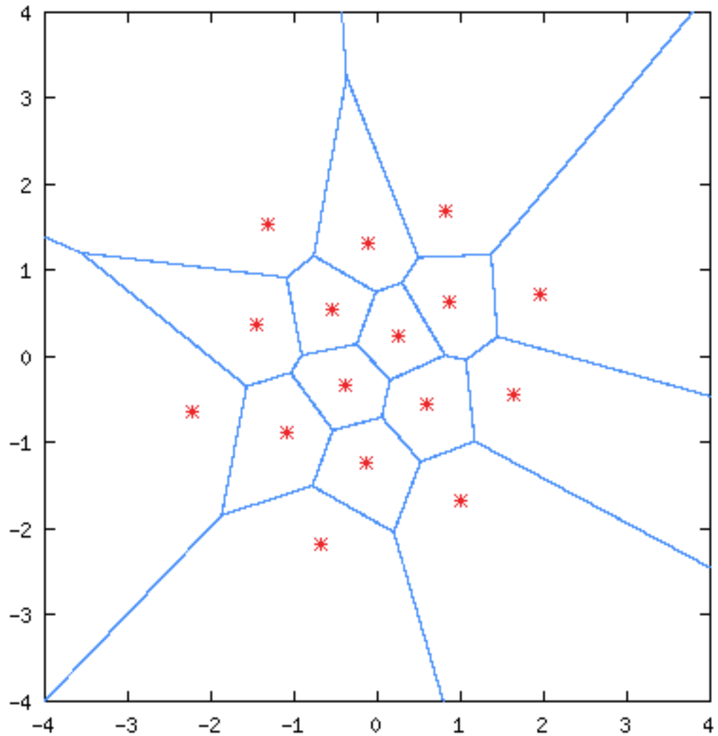
the approximate values are uniquely represented by 2 bits

`00` for `−3`, `01` for `−1`, `10` for `1`, `11` for `3`

so, the approximation is also a “compressed” way of representing the data; we use 2 bits instead of the 4 bits that would needed to represent the 9 different integers

… the "vector quantization" concept – a 2D example

a VQ example of a *2-dimensional* (2D) space



each pair of numbers (i.e. each 2D point) falling in a particular region is approximated by the red cross associated with that region;

notice that there are 16 regions and 16 red crosses each of which can be uniquely represented by 4 bits; thus, this is a 2-dimensional, 4-bit VQ

Paulo Trigo Silva @ ISEL

… "vector quantization" (VQ) and Learning VQ (LVQ)

so, the general formulation is that "vector quantization" maps *k-dimensional* vectors into a finite set of vectors $Y = \{y_i: i = 1, 2, \ldots, n\}$;

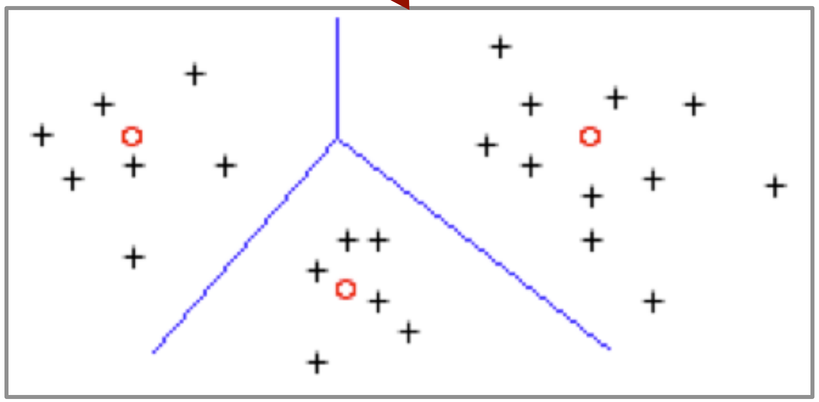
each vector y_i is called a "codebook vector" (cBv), and the set, Y, of all cBv is called a "codebook"

the "Learning Vector Quantization" (LVQ) goal is that:

given a dataset (training sequence) and given a number of "codebook vectors", find a codebook which best represents the class values in the dataset

example: each + is a training example (from the dataset) end each red-circle is a cBv

the lines represent the decision boundaries



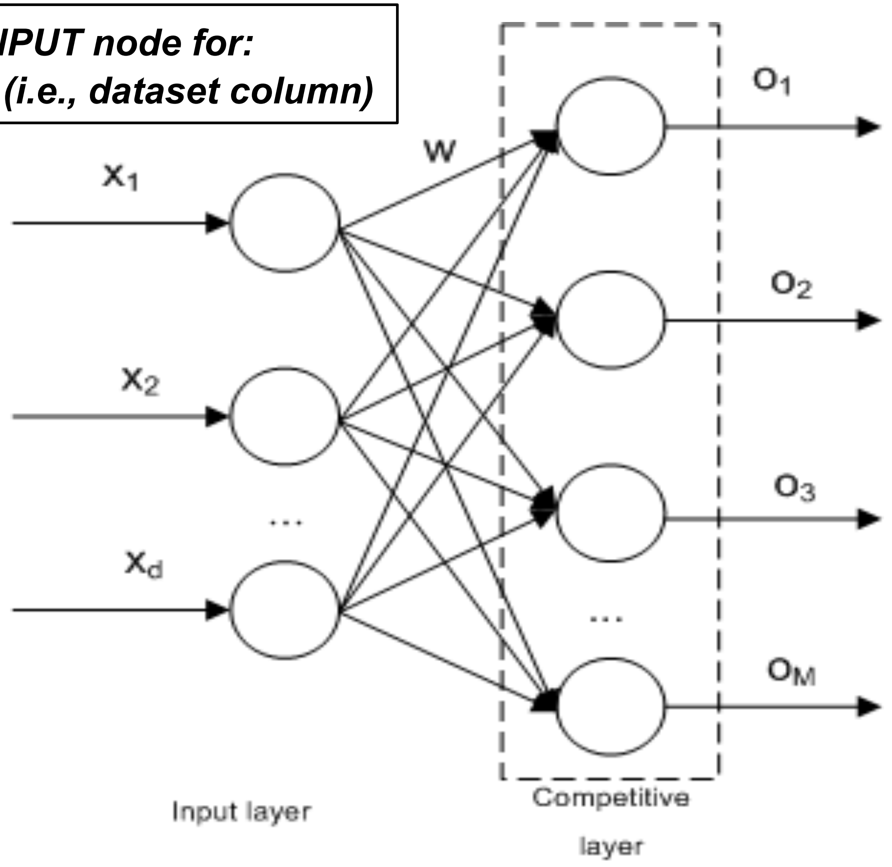
Paulo Trigo Silva @ ISEL

LVQ representation – Neural Network (no Hidden Layers)

one layer of:
INPUT nodes

one layer of:
OUTPUT nodes

one INPUT node for: each feature (i.e., dataset column)



*the OUTPUT layer is also called; a "**codebook**"*

*each OUTPUT node is also called: a "**codebook vector**"*

at least one OUTPUT node for: each class value

one weight (connection) from each INPUT node to each OUTPUT node

... there are as many **input** nodes as **features** in the dataset

… there is at least one **output** node for each **class** value

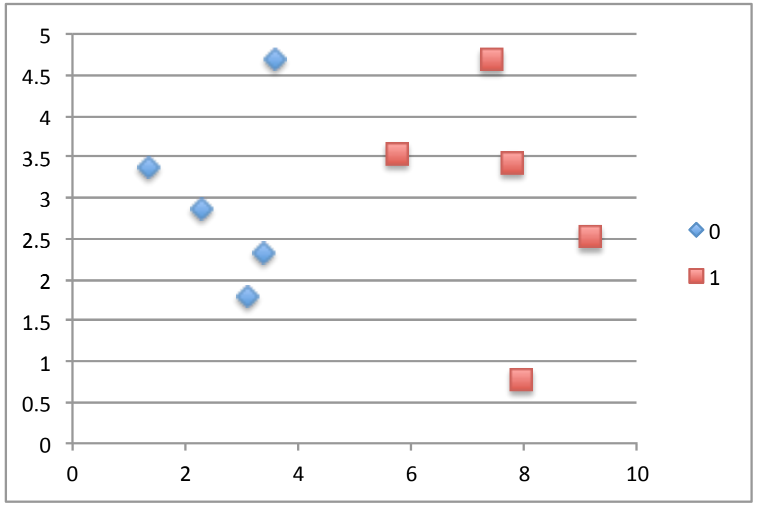
here, D input and M output nodes; so D features and if using 2 nodes for each class value we have M/2 class values

Paulo Trigo Silva @ ISEL

Example – dataset and "codebook vectors"

Dataset		
X1	X2	Y
3,39353321	2,33127338	0
3,11007348	1,78153964	0
1,34380883	3,36836095	0
3,58229404	4,67917911	0
2,28036244	2,86699026	0
7,42343694	4,69652288	1
5,745052	3,5339898	1
9,17216862	2,51110105	1
7,79278348	3,42408894	1
7,93982082	0,79163723	1

dataset



features: X1, X2
class: Y
class values: 0, 1

Codebook Vectors		
X1	X2	Y
3,58229404	0,79163723	0
7,79278348	2,33127338	0
7,93982082	2,86699026	1
3,39353321	4,67917911	1

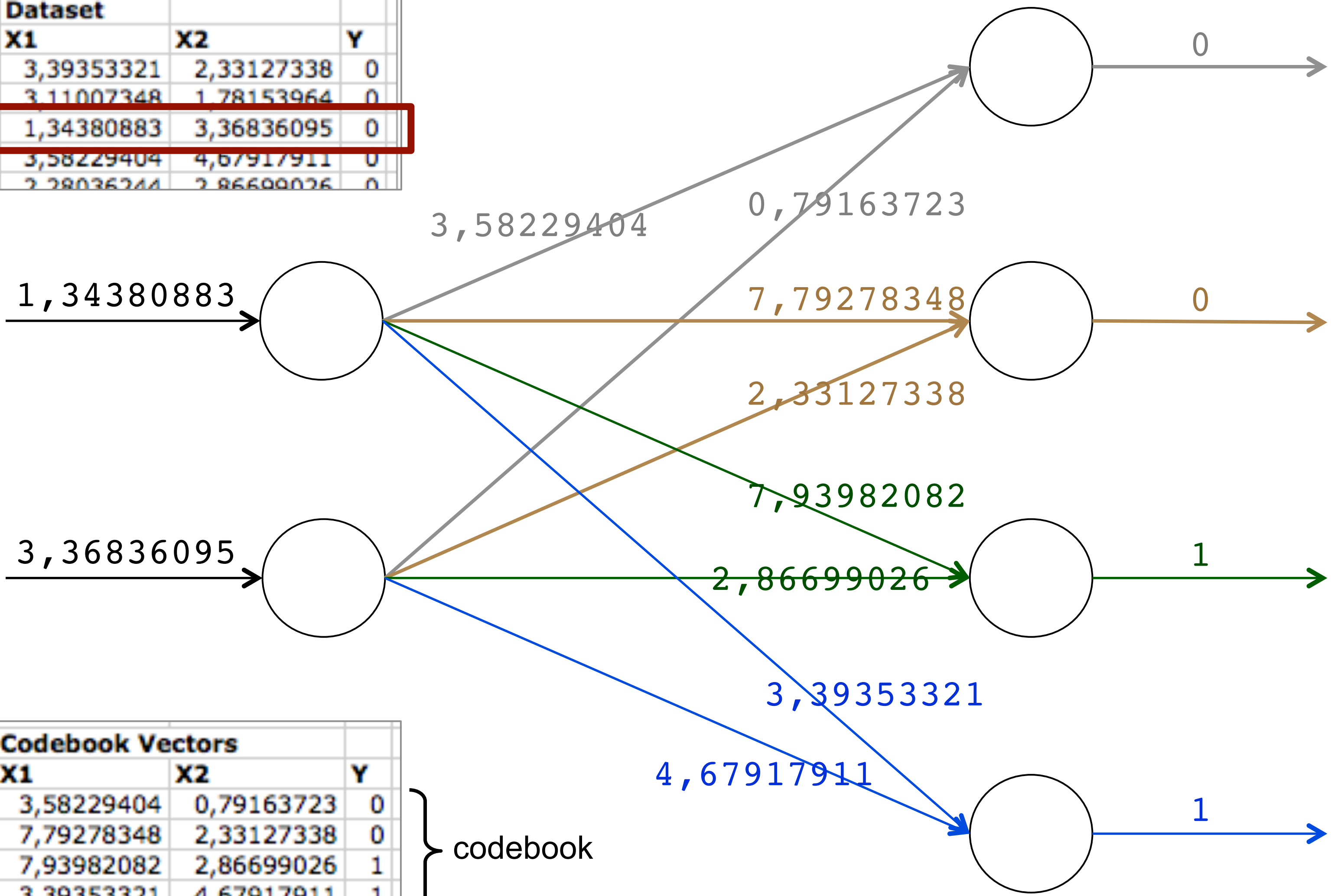
Exercise:

given these dataset and codebook vectors draw the LVQ neural network representation;

tag the connections using the weights taken from these codebook vectors and, as input, consider the third (3rd) instance of the dataset.

… example – the LVQ Neural Network representation

Dataset		
X1	X2	Y
3,39353321	2,33127338	0
3,11007348	1,78153964	0
1,34380883	3,36836095	0
3,58229404	4,67917911	0
2,28036244	2,86699026	0

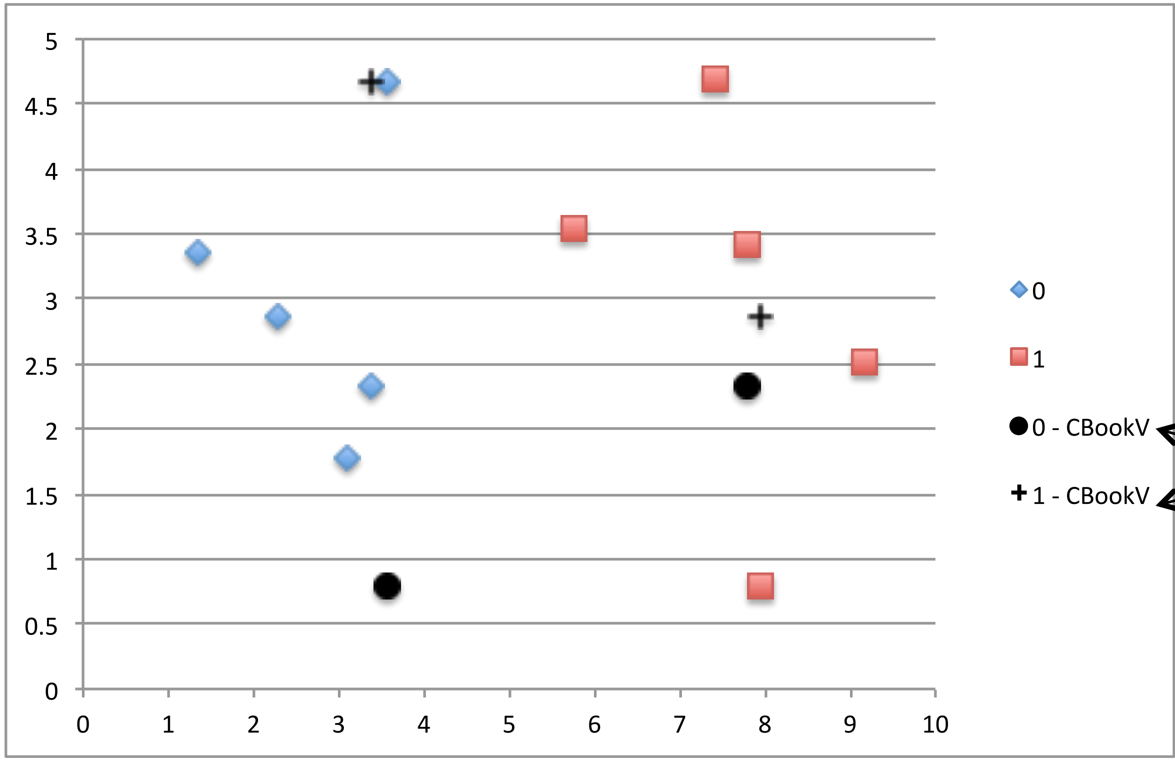


Codebook Vectors		
X1	X2	Y
3,58229404	0,79163723	0
7,79278348	2,33127338	0
7,93982082	2,86699026	1
3,39353321	4,67917911	1

codebook

Paulo Trigo Silva @ ISEL

… example – plot of dataset and the initial codebook



each codebook vector (cBV)

Codebook Vectors		
X1	X2	Y
3,58229404	0,79163723	0
7,79278348	2,33127338	0
7,93982082	2,86699026	1
3,39353321	4,67917911	1

the **initial** codebook

we defined 2 codebook vectors (cBv) for each class value;

i.e, 2 cBv for class value 0 and 2 cBv for class value 1

LVQ method – a “competitive learning” approach

… the intuition is that,
given **a training example**, each output unit (codebook vector)
“competes” with all other output units

and, the winner is,
the output unit **closer** to that training example

and, the winner gets (as a reward):
an **update of its weights** aiming to get more adapted than other units;
the adaptation is processed via **attraction** and **repulsion** rules

attraction rule – move codebook vector **closer to** the training example
repulsion rule – move codebook vector **away from** the training example

*because <u>**one**</u> **codebook vector** is selected for modification <u>for each training instance,</u>
the algorithm is referred to as a <u>winner-take-all</u> (type of <u>competitive</u> learning)*

Paulo Trigo Silva @ ISEL

The LVQ learning process – from the Data to the Model

the LVQ algorithm learns the “codebook vectors” from the training data

- **[1]** choose the number of codebook vectors (cBv) to use
  - possibly (but not necessarily) an equal number of cBv for each class
- **[2]** start the learning process with a pool of cBv
  - either randomly selected from training data,
  - or randomly generated with the same scale as the training data
  - … each cBv has the same number of attributes as data and an output class value
- **[3]** the instances in the training data are processed one-at-a-time
  - for a given training instance, select its **most similar** cBv
  - … the selected cBv is the “winner”, also called the “best matching unit” (**BMU**)
- **[4]** the BMU gets its weights updated
  - if the BMU has the same class as the training instance, apply the **attraction** rule
  - otherwise, apply the **repulsion** rule (in relation to that training instance)

Paulo Trigo Silva @ ISEL

"similar to" – how to compute this concept?

… for a given instance, select its **most similar** cBv

a common way of computing similarity is to establish a **distance** criteria
and assume such **distance as the similarity order** relation
i.e., the *closer the higher similarity*

so, we need to define&compute a **distance** between vectors

and **rank** the distance from an instance data to each cBv
the **closer** they are the **more similar** they are!

Paulo Trigo Silva @ ISEL

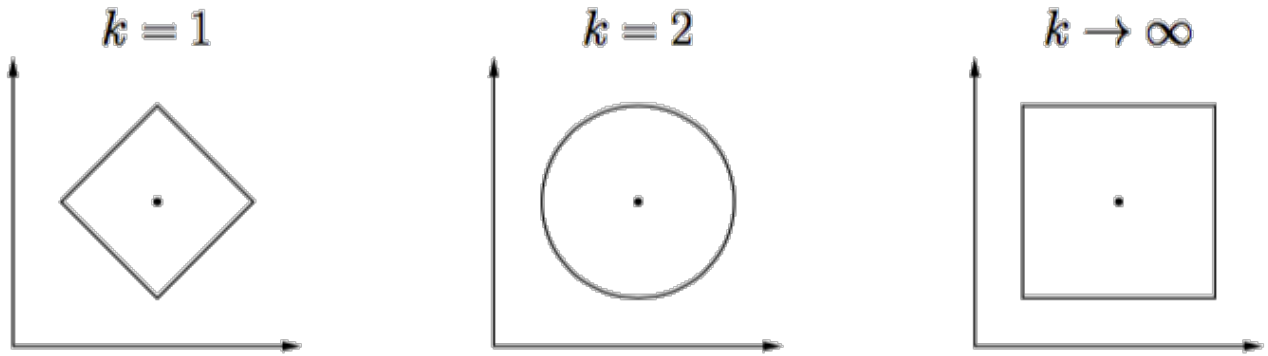
“similar to” – define&compute a distance criteria

a general formulation of several distance functions (or, distance criteria) commonly used for numeric data

$$d_k(\vec{x},\vec{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^k\right)^{\frac{1}{k}}$$

Well-known special cases from this family are:

$k = 1$: Manhattan or city block distance,
$k = 2$: Euclidean distance,
$k \to \infty$: maximum distance, i.e. $d_\infty(\vec{x},\vec{y}) = \max_{i=1}^{n} |x_i - y_i|$.



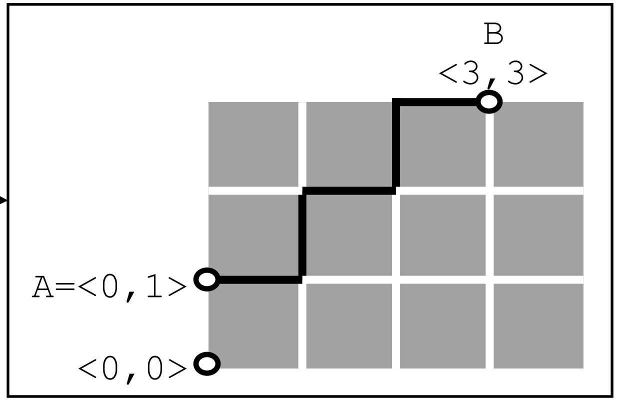
all points lying on a circle or rectangle are sharing the **same distance to the center point** according to the corresponding distance function

Paulo Trigo Silva @ ISEL

“similar to” – the Manhattan distance

the Manhattan distance, or the “city block” distance
inspired on the idea of Manhattan having a “grid format”

the Manhattan distance between
`A=<0,1>` and `B=<3,3>`
3 – 0 + 3 – 1 = 3 + 2 = 5



the Manhattan distance

$$ManhDist(X,Y) = \sum_{i=1}^{n} | x_i - y_i |$$

compute Manhattan distance between
`A=<0,3,2,1,10>` and `B=<2,7,1,0,0>`

|0-2|+|3-7|+|2-1|+|1-0|+|10-0| = 18

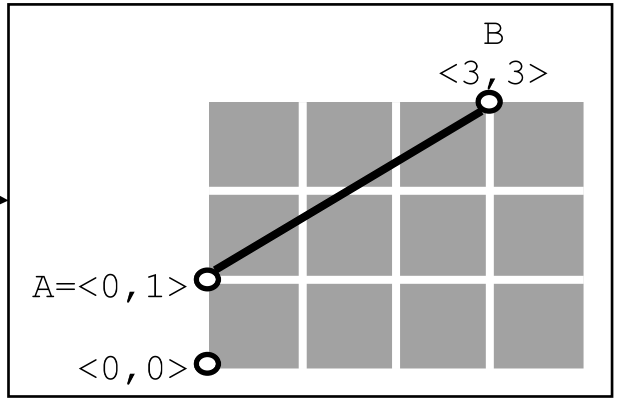
“similar to” – the Eclidean distance

straight line distance between two points

the Euclidean distance between

A=<0,1> and B=<3,3>

$[\,(3-0)^2 + (3-1)^2\,]^{1/2} = [9+4]^{1/2} = 3.6$



the Euclidean distance between vectors d_j and d_k

$$\left|d_j - d_k\right| = \sqrt{\sum_{i=1}^{n}\left(d_{i,j} - d_{i,k}\right)^2}$$

compute Euclidean distance between

d_1=<a,b,c> and d_2=<x,y,z>

$$\sqrt{\mathrm{Abs}[a-x]^2 + \mathrm{Abs}[b-y]^2 + \mathrm{Abs}[d-z]^2}$$

Paulo Trigo Silva @ ISEL

“attraction/repulsion” – how to compute these concepts?

… if the BMU has *the same class as* the training instance, **then** apply **attraction** rule; **otherwise** apply **repulsion** rule

… and an additional question is:
what is the **amount** that the vector is **moved**?
such amount is called the “**learning rate**”

attraction rule – move codebook vector, *x*, **closer to** the training example, *t*, by the amount of *LearningRate*,

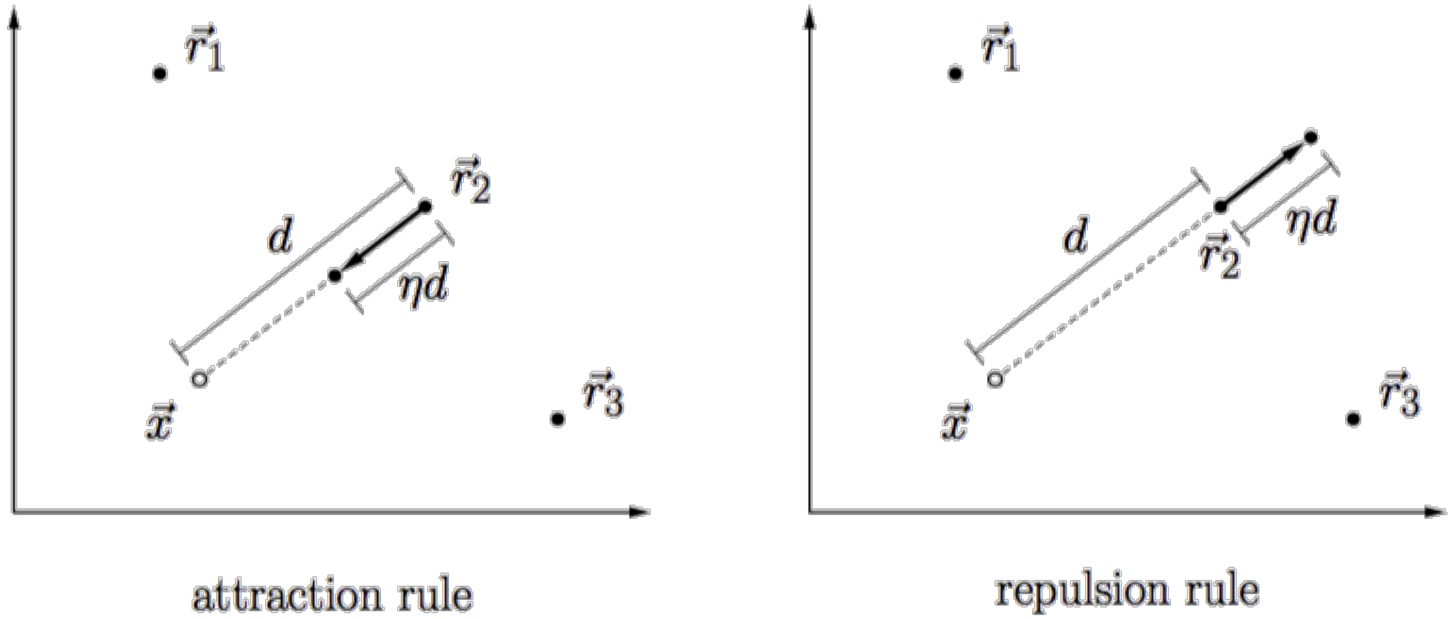
$$x = x + LearningRate \times (t - x)$$

repulsion rule – move codebook vector, *x*, **away from** the training example, *t*, by the amount of *LearningRate*,

$$x = x - LearningRate \times (t - x)$$

Paulo Trigo Silva @ ISEL

“attraction/repulsion” – an illustrative example



- $\vec{x}$: data point, $\vec{r}_i$: reference vector
- $\eta = 0.4$ (learning rate)

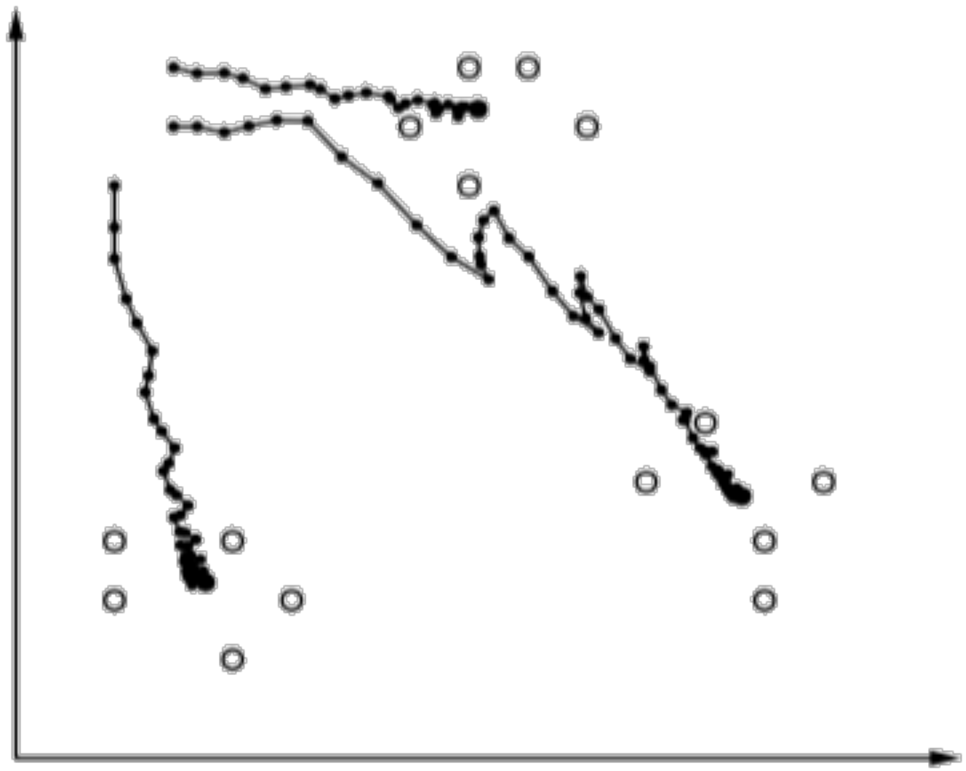
… this would be repeated for each training instance;
one iteration of the training dataset is called an **epoch**;
the process is completed for a **number of epochs** that we define, e.g., 200

“attraction/repulsion and epoch” – an illustrative example

each training instance originates the attraction/repulsion of <u>only one</u> cBv

an **epoch** has as many cBv movements as the number, N, of training instances

a **number of epochs**, MaxEpoch, originates MaxEpoch × N movements of cBv



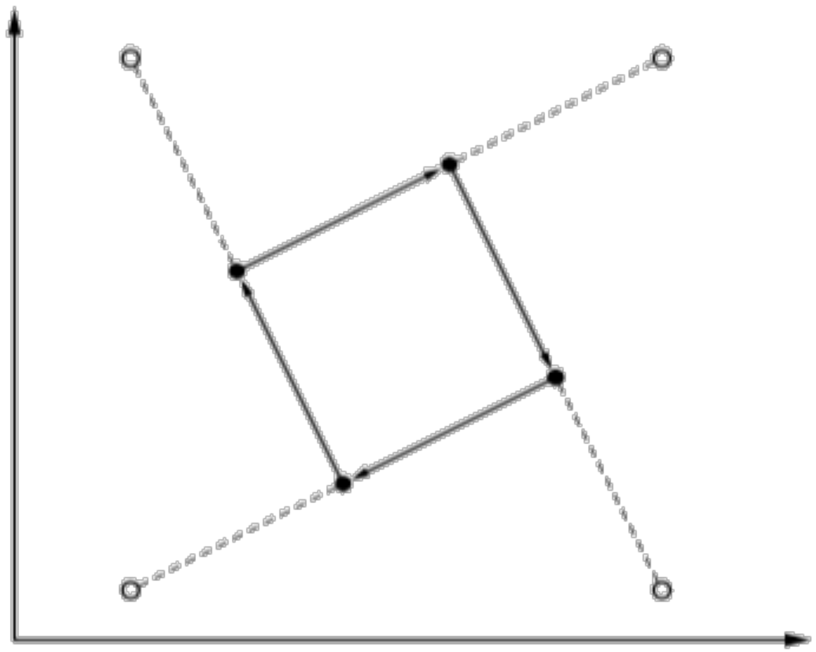
in this illustration we have one codebook vector (cBv) associated with each class value;

but we may have several cBv associated with each class value

the amount that each cBv is moved is controlled by the *LearningRate*;

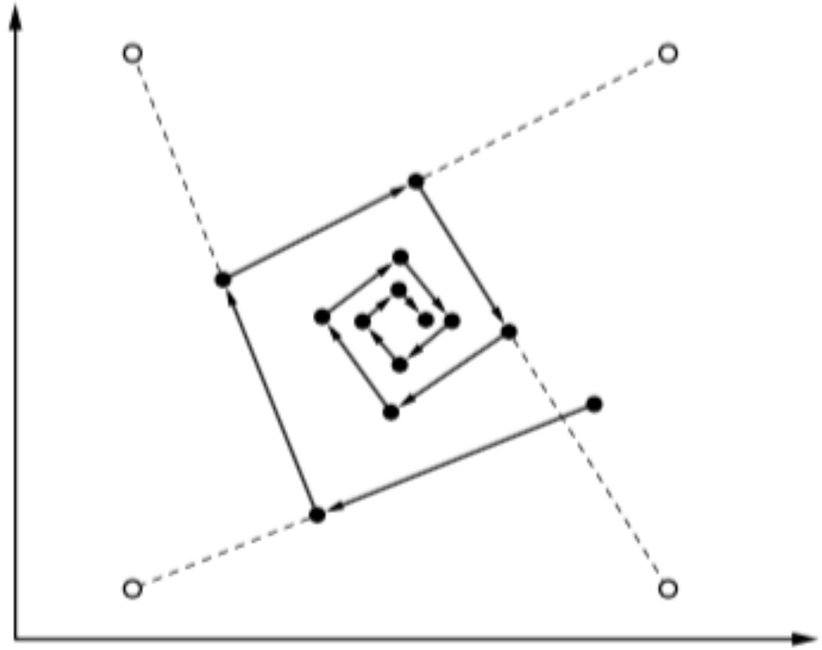
in this illustration we have a **fixed** *LearningRate* = 0.1

The *LearningRate* – fixed or time-dependent?



problem – a **fixed** learning rate

the process may get trapped into an oscillatory mode



solution – a **time-dependent** learning rate

the rate decreases with **epoch**

Paulo Trigo Silva @ ISEL

The *LearningRate* – a time-dependent approach

$$LearningRate = alpha \times (1 - \frac{Epoch}{MaxEpoch})$$

LearningRate is the learning rate for the current *Epoch* (from 0 to *MaxEpoch* – 1)

alpha is the learning rate specified to the algorithm at the start of training run

so, as *Epoch* increases (i.e., as time goes) the *LearningRate* decreases, and therefore the attraction/repulsion movements get smaller and smaller

using the above approach the *LearningRate* <u>decreases linearly</u> with the *Epoch*

but, there are other alternatives to define a time-dependent *LearningRate*, such as,

$LearningRate = alpha^{Epoch}$, with $0 \leq alpha \leq 1$

where the *LearningRate* decreases in a non-linearly with the *Epoch*

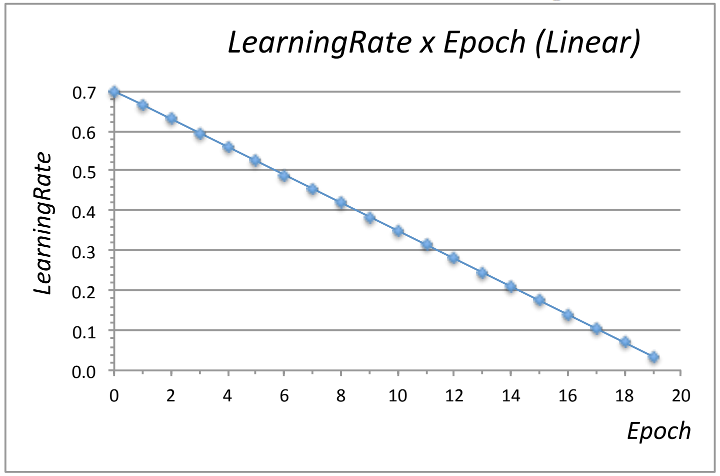
The time-dependent *LearningRate* – an example

alpha = 0,7

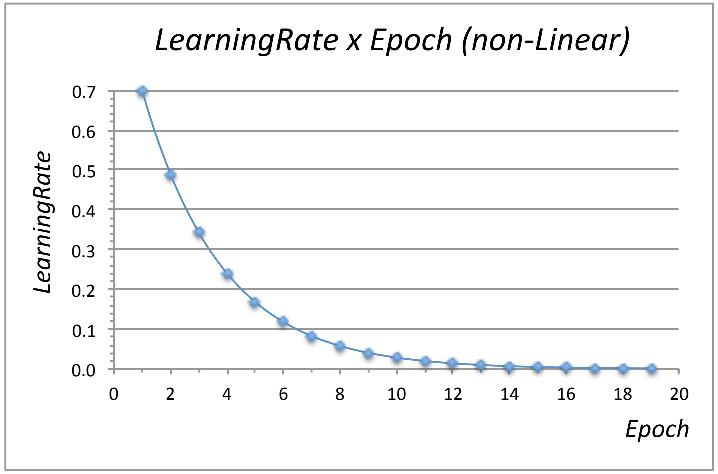
MaxEpoch = 20

	Epoch	Linear LearningRate	non-Linear LearningRate
alpha	0	0,70000	1,00000
	1	0,66500	0,70000
	2	0,63000	0,49000
	3	0,59500	0,34300
	4	0,56000	0,24010
	5	0,52500	0,16807
	6	0,49000	0,11765
	7	0,45500	0,08235
	8	0,42000	0,05765
	9	0,38500	0,04035
	10	0,35000	0,02825
	11	0,31500	0,01977
	12	0,28000	0,01384
	13	0,24500	0,00969
	14	0,21000	0,00678
	15	0,17500	0,00475
	16	0,14000	0,00332
	17	0,10500	0,00233
	18	0,07000	0,00163
	19	0,03500	0,00114
MaxEpoch	20		

$$LearningRate = alpha \times (1 - \frac{Epoch}{MaxEpoch})$$



LearningRate = *alpha* ^{Epoch}, with 0 ≤ *alpha* ≤ 1



Paulo Trigo Silva @ ISEL

[recall] The overall learning process – an example

the LVQ algorithm learns the "codebook vectors" from the training data

- **[1]** choose the number of codebook vectors (cBv) to use
  - possibly (but not necessarily) an equal number of cBv for each class
- **[2]** start the learning process with a pool of cBv
  - either randomly selected from training data,
  - or randomly generated with the same scale as the training data
  - … each cBv has the same number of attributes as data and an output class value
- **[3]** the instances in the training data are processed one-at-a-time
  - for a given training instance, select its **most similar** cBv
  - … the selected cBv is the "winner", also called the "best matching unit" (**BMU**)
- **[4]** the BMU gets its weights updated
  - if the BMU has the same class as the training instance, apply the **attraction** rule
  - otherwise, apply the **repulsion** rule (in relation to that training instance)

Dataset

X1	X2	Y
3,39353321	2,33127338	0
3,11007348	1,78153964	0
1,34380883	3,36836095	0
3,58229404	4,67917911	0
2,28036244	2,86699026	0
7,42343694	4,69652288	1
5,745052	3,5339898	1
9,17216862	2,51110105	1
7,79278348	3,42408894	1
7,93982082	0,79163723	1

Codebook Vectors

X1	X2	Y
3,58229404	0,79163723	0
7,79278348	2,33127338	0
7,93982082	2,86699026	1
3,39353321	4,67917911	1

… example – the first 4 dataset examples

Dataset

X1	X2	Y
3,39353321	2,33127338	0
3,11007348	1,78153964	0
1,34380883	3,36836095	0
3,58229404	4,67917911	0
2,28036244	2,86699026	0
7,42343694	4,69652288	1
5,745052	3,5339898	1
9,17216862	2,51110105	1
7,79278348	3,42408894	1
7,93982082	0,79163723	1

Codebook Vectors

X1	X2	Y
3,58229404	0,79163723	0
7,79278348	2,33127338	0
7,93982082	2,86699026	1
3,39353321	4,67917911	1

- …
- **[3]** the instances in the training data are processed one-at-a-time
  - for a given training instance, select its **most similar** cBv
  - … the selected cBv is the “winner”, also called the “best matching unit” (**BMU**)
- **[4]** the BMU gets its weights updated
  - if the BMU has the same class as the training instance, apply the **attraction** rule
  - otherwise, apply the **repulsion** rule (in relation to that training instance)

$$EuclideanDistance(a, b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

(fixed) LearningRate = 0,7

Training

#	Codebook vectors X1	X2	Y	Input X1	X2	Y	Distances (X1-X1)^2	(x2-X2)^	Sum	Distance	BMU?	Codebook vectors t+1 X1	X2	Y
1	3,58229404	0,79163723	0	3,39353321	2,33127338	0	0,035631	2,37048	2,40611	1,55116	BMU	3,450161	1,869383	0
2	7,79278348	2,33127338	0	3,39353321	2,33127338	0	19,3534	0	19,3534	4,39925		7,792783	2,331273	0
3	7,93982082	2,86699026	1	3,39353321	2,33127338	0	20,66873	0,28699	20,9557	4,57774		7,939821	2,86699	1
4	3,39353321	4,67917911	1	3,39353321	2,33127338	0	0	5,51266	5,51266	2,34791		3,393533	4,679179	1
1	3,45016146	1,86938254	0	3,11007348	1,78153964	0	0,11566	0,00772	0,12338	0,35125	BMU	3,2121	1,807893	0
2	7,79278348	2,33127338	0	3,11007348	1,78153964	0	21,92777	0,30221	22,23	4,71487		7,792783	2,331273	0
3	7,93982082	2,86699026	1	3,11007348	1,78153964	0	23,32646	1,1782	24,5047	4,95022		7,939821	2,86699	1
4	3,39353321	4,67917911	1	3,11007348	1,78153964	0	0,080349	8,39631	8,47666	2,91147		3,393533	4,679179	1
1	3,21209988	1,80789251	0	1,34380883	3,36836095	0	3,490511	2,43506	5,92557	2,43425		3,2121	1,807893	0
2	7,79278348	2,33127338	0	1,34380883	3,36836095	0	41,58927	1,07555	42,6648	6,53183		7,792783	2,331273	0
3	7,93982082	2,86699026	1	1,34380883	3,36836095	0	43,50737	0,25137	43,7587	6,61504		7,939821	2,86699	1
4	3,39353321	4,67917911	1	1,34380883	3,36836095	0	4,20137	1,71824	5,91961	2,43303	BMU	4,82834	5,596752	1
1	3,21209988	1,80789251	0	3,58229404	4,67917911	0	0,137044	8,24429	8,38133	2,89505		3,2121	1,807893	0
2	7,79278348	2,33127338	0	3,58229404	4,67917911	0	17,72822	5,51266	23,2409	4,82088		7,792783	2,331273	0
3	7,93982082	2,86699026	1	3,58229404	4,67917911	0	18,98804	3,28403	22,2721	4,71933		7,939821	2,86699	1
4	4,82834028	5,59675182	1	3,58229404	4,67917911	0	1,552631	0,84194	2,39457	1,54744	BMU	5,700573	6,239053	1

…

Paulo Trigo Silva @ ISEL

… example – for each BMU attraction or repulsion rule?

Codebook Vectors		
X1	X2	Y
3,58229404	0,79163723	0
7,79278348	2,33127338	0
7,93982082	2,86699026	1
3,39353321	4,67917911	1

- …
- **[4]** the BMU gets its weights updated
  - if the BMU has the same class as the training instance, apply the **attraction** rule
  - otherwise, apply the **repulsion** rule (in relation to that training instance)

Training

	Codebook vectors			Input			Distances					Codebook vectors t+1		
#	X1	X2	Y	X1	X2	Y	(X1-X1)^2	(x2-X2)^	Sum	Distance	BMU?	X1	X2	Y
1	3,58229404	0,79163723	0	3,39353321	2,33127338	0	0,035631	2,37048	2,40611	1,55116	BMU	3,450161	1,869383	0
2	7,79278348	2,33127338	0	3,39353321	2,33127338	0	19,3534	0	19,3534	4,39925		7,792783	2,331273	0
3	7,93982082	2,86699026	1	3,39353321	2,33127338	0	20,66873	0,28699	20,9557	4,57774		7,939821	2,86699	1
4	3,39353321	4,67917911	1	3,39353321	2,33127338	0	0	5,51266	5,51266	2,34791		3,393533	4,679179	1
1	3,45016146	1,86938254	0	3,11007348	1,78153964	0	0,11566	0,00772	0,12338	0,35125	BMU	3,2121	1,807893	0
2	7,79278348	2,33127338	0	3,11007348	1,78153964	0	21,92777	0,30221	22,23	4,71487		7,792783	2,331273	0
3	7,93982082	2,86699026	1	3,11007348	1,78153964	0	23,32646	1,1782	24,5047	4,95022		7,939821	2,86699	1
4	3,39353321	4,67917911	1	3,11007348	1,78153964	0	0,080349	8,39631	8,47666	2,91147		3,393533	4,679179	1
1	3,21209988	1,80789251	0	1,34380883	3,36836095	0	3,490511	2,43506	5,92557	2,43425		3,2121	1,807893	0
2	7,79278348	2,33127338	0	1,34380883	3,36836095	0	41,58927	1,07555	42,6648	6,53183		7,792783	2,331273	0
3	7,93982082	2,86699026	1	1,34380883	3,36836095	0	43,50737	0,25137	43,7587	6,61504		7,939821	2,86699	1
4	3,39353321	4,67917911	1	1,34380883	3,36836095	0	4,20137	1,71824	5,91961	2,43303	BMU	4,82834	5,596752	1
1	3,21209988	1,80789251	0	3,58229404	4,67917911	0	0,137044	8,24429	8,38133	2,89505		3,2121	1,807893	0
2	7,79278348	2,33127338	0	3,58229404	4,67917911	0	17,72822	5,51266	23,2409	4,82088		7,792783	2,331273	0
3	7,93982082	2,86699026	1	3,58229404	4,67917911	0	18,98804	3,28403	22,2721	4,71933		7,939821	2,86699	1
4	4,82834028	5,59675182	1	3,58229404	4,67917911	0	1,552631	0,84194	2,39457	1,54744	BMU	5,700573	6,239053	1

…

Paulo Trigo Silva @ ISEL

… example – initially and after 1 epoch

Dataset

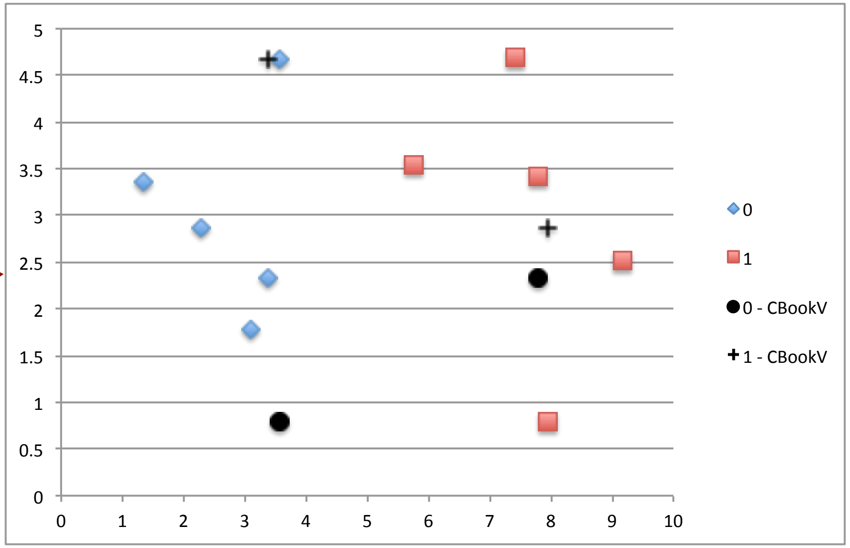
X1	X2	Y
3,39353321	2,33127338	0
3,11007348	1,78153964	0
1,34380883	3,36836095	0
3,58229404	4,67917911	0
2,28036244	2,86699026	0
7,42343694	4,69652288	1
5,745052	3,5339898	1
9,17216862	2,51110105	1
7,79278348	3,42408894	1
7,93982082	0,79163723	1

(fixed) LearningRate = 0,7

Codebook Vectors

X1	X2	Y
3,58229404	0,79163723	0
7,79278348	2,33127338	0
7,93982082	2,86699026	1
3,39353321	4,67917911	1

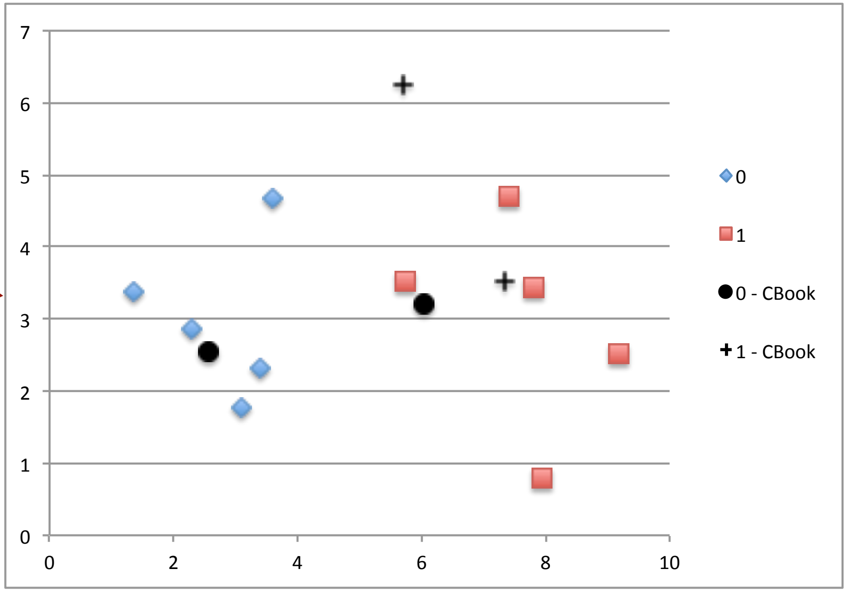
initially



after 1 epoch

Trained Codebook Vectors - after 1 epoch

X1	X2	Y
2,55988367	2,54926094	0
6,04838903	3,19502377	0
7,34346105	3,5122898	1
5,70057264	6,23905272	1



Paulo Trigo Silva @ ISEL

same example – but, with 100 and 500 *MaxEpoch*

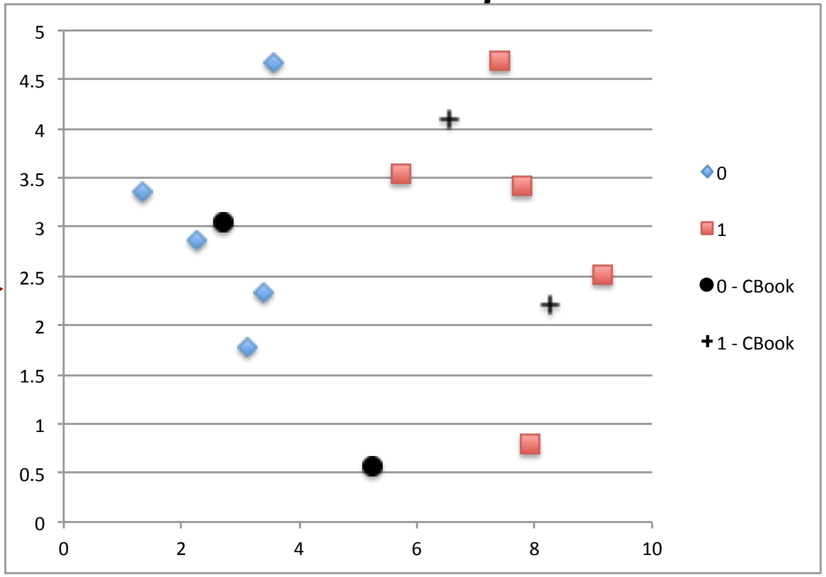
time-dependent LearningRate

$$LearningRate = alpha \times (1 - \frac{Epoch}{MaxEpoch})$$

alpha = 0,7; *MaxEpoch* = 100

Trained Codebook Vectors - after 100 epochs

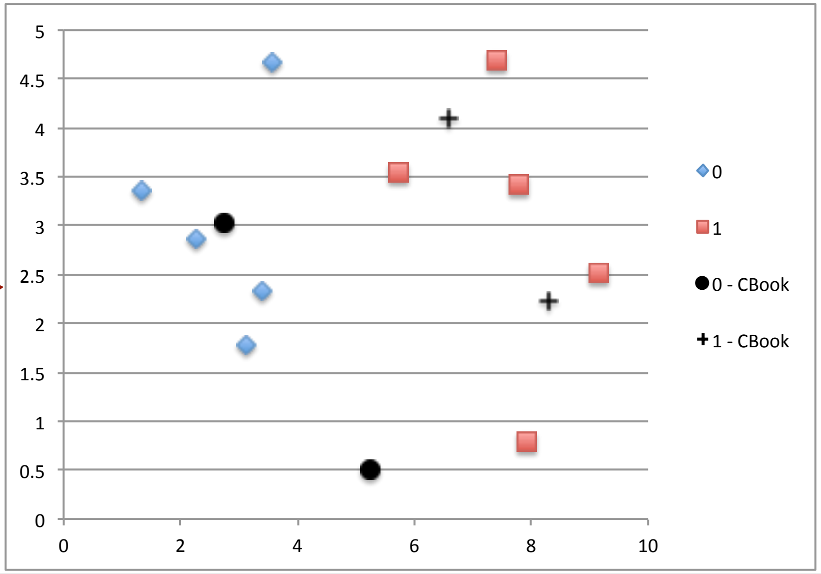
X1	X2	Y
2,72574276	3,04268663	0
5,2636527	0,55812955	0
8,27643991	2,20561477	1
6,55207296	4,09297275	1



alpha = 0,7; *MaxEpoch* = 500

Trained Codebook Vectors - after 500 epochs

X1	X2	Y
2,73469779	3,02211129	0
5,24320591	0,49988667	0
8,29041704	2,22638249	1
6,57012799	4,10547856	1



… a question!

Dataset

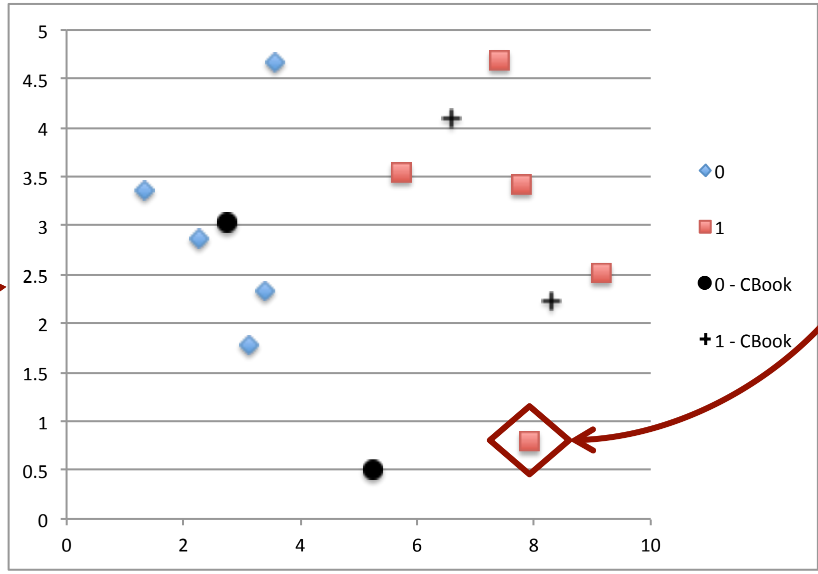
X1	X2	Y
3,39353321	2,33127338	0
3,11007348	1,78153964	0
1,34380883	3,36836095	0
3,58229404	4,67917911	0
2,28036244	2,86699026	0
7,42343694	4,69652288	1
5,745052	3,5339898	1
9,17216862	2,51110105	1
7,79278348	3,42408894	1
7,93982082	0,79163723	1

given the dataset, the "codebook vectors" and the Euclidean distance how is the point marked (with a red diamond) classified (as a zero or as a one)?

alpha = 0,7; *MaxEpoch* = 500

Trained Codebook Vectors - after 500 epochs

X1	X2	Y
2,73469779	3,02211129	0
5,24320591	0,49988667	0
8,29041704	2,22638249	1
6,57012799	4,10547856	1



notice that the point marked (with a red diamond) belongs to the dataset and is the instance (with class one) in the "lowest-and-most-right" location…

Paulo Trigo Silva @ ISEL

… a question – lets answer the question!

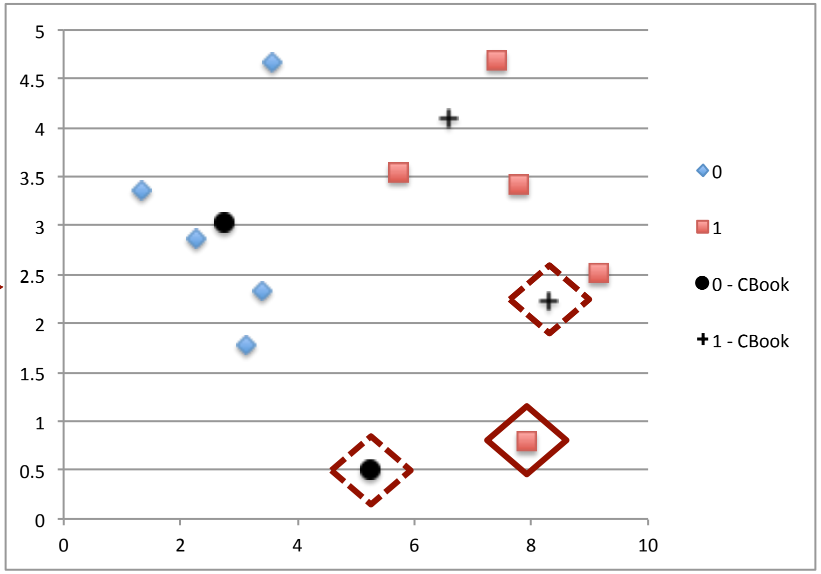
Dataset

X1	X2	Y
3,39353321	2,33127338	0
3,11007348	1,78153964	0
1,34380883	3,36836095	0
3,58229404	4,67917911	0
2,28036244	2,86699026	0
7,42343694	4,69652288	1
5,745052	3,5339898	1
9,17216862	2,51110105	1
7,79278348	3,42408894	1
7,93982082	0,79163723	1

alpha = 0,7; *MaxEpoch* = 500

Trained Codebook Vectors - after 500 epochs

X1	X2	Y
2,73469779	3,02211129	0
5,24320591	0,49988667	0
8,29041704	2,22638249	1
6,57012799	4,10547856	1



we can identify, in the dataset, the point to be classified

we can (visually) see which cBv are closer to the point to be classified

so, we **just need** to calculate the **distance** (Euclidean) from **each cBv to the point**…

Paulo Trigo Silva @ ISEL

… the classification of the marked point!

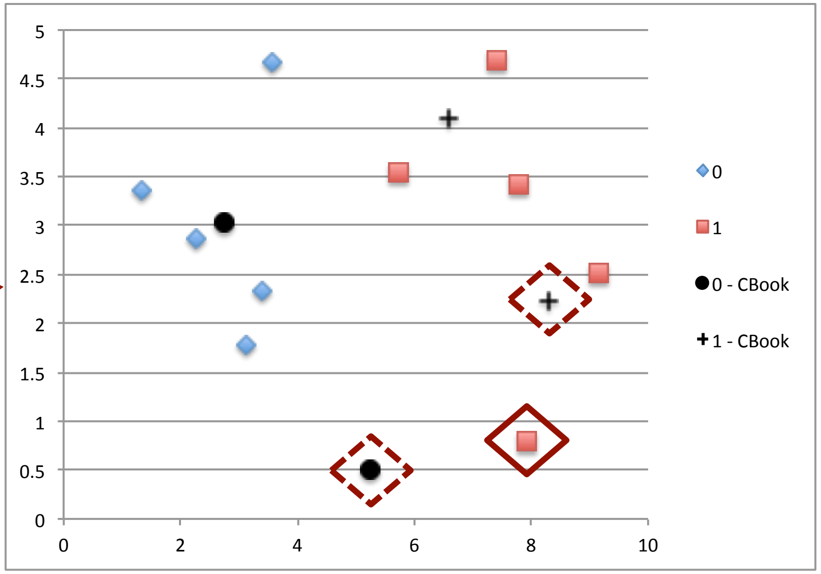
Dataset

X1	X2	Y
3,39353321	2,33127338	0
3,11007348	1,78153964	0
1,34380883	3,36836095	0
3,58229404	4,67917911	0
2,28036244	2,86699026	0
7,42343694	4,69652288	1
5,745052	3,5339898	1
9,17216862	2,51110105	1
7,79278348	3,42408894	1
7,93982082	0,79163723	1

alpha = 0,7; *MaxEpoch* = 500

Trained Codebook Vectors - after 500 epochs

X1	X2	Y
2,73469779	3,02211129	0
5,24320591	0,49988667	0
8,29041704	2,22638249	1
6,57012799	4,10547856	1



	X1	X2	class	
the point to be classified	7,93982082	0,79163723	1	
				Euclidean distance
cBv	5,24320591	0,49988667	0	2,71235144
cBv	8,29041704	2,22638249	1	1,47696028

so, the marked point is classified as 1 (one)
why?

Paulo Trigo Silva @ ISEL

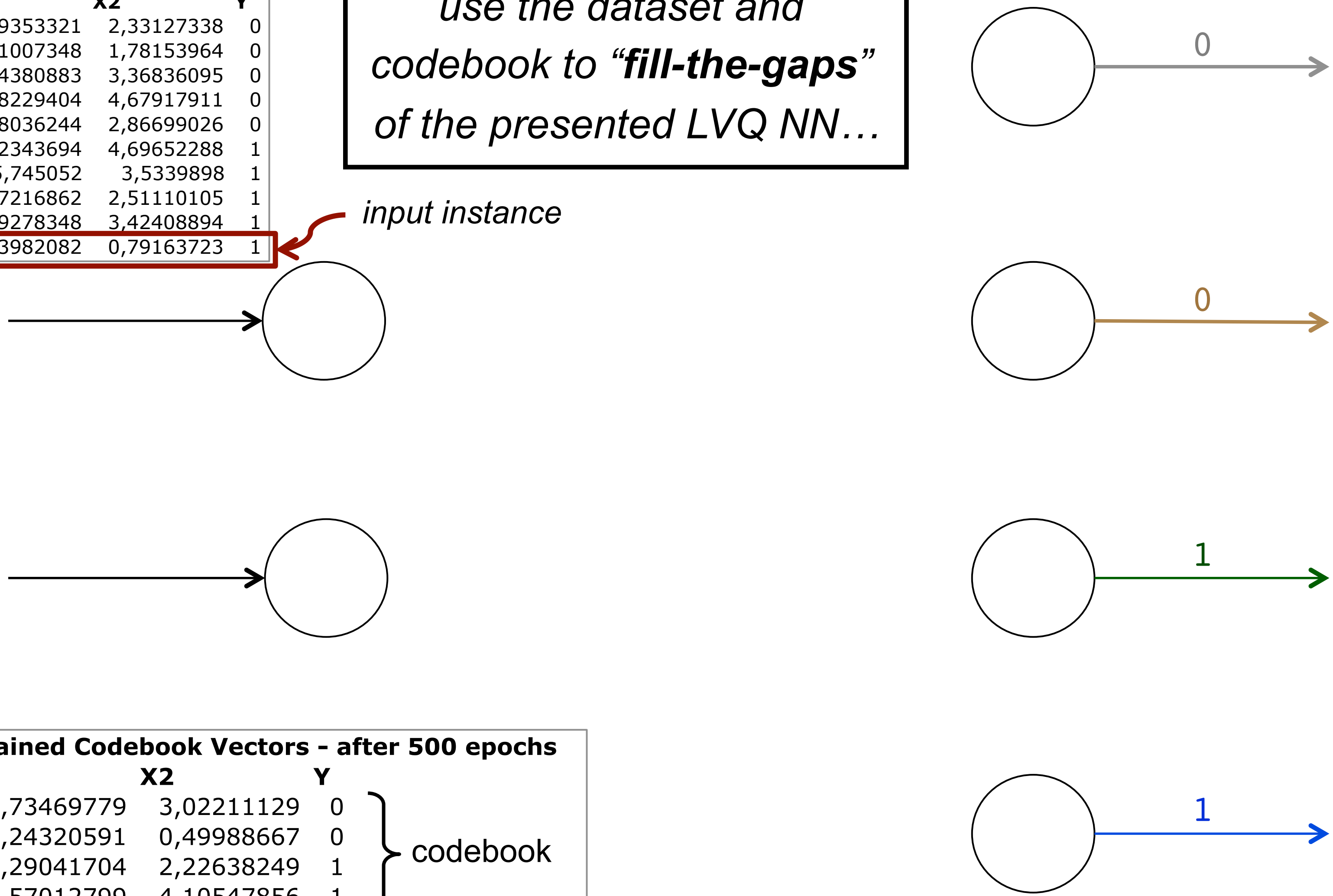
an exercise – the LVQ Neural Network (NN) structure

Dataset

X1	**X2**	**Y**
3,39353321	2,33127338	0
3,11007348	1,78153964	0
1,34380883	3,36836095	0
3,58229404	4,67917911	0
2,28036244	2,86699026	0
7,42343694	4,69652288	1
5,745052	3,5339898	1
9,17216862	2,51110105	1
7,79278348	3,42408894	1
7,93982082	0,79163723	1

*use the dataset and codebook to "**fill-the-gaps**" of the presented LVQ NN…*

input instance



Trained Codebook Vectors - after 500 epochs

X1	**X2**	**Y**
2,73469779	3,02211129	0
5,24320591	0,49988667	0
8,29041704	2,22638249	1
6,57012799	4,10547856	1

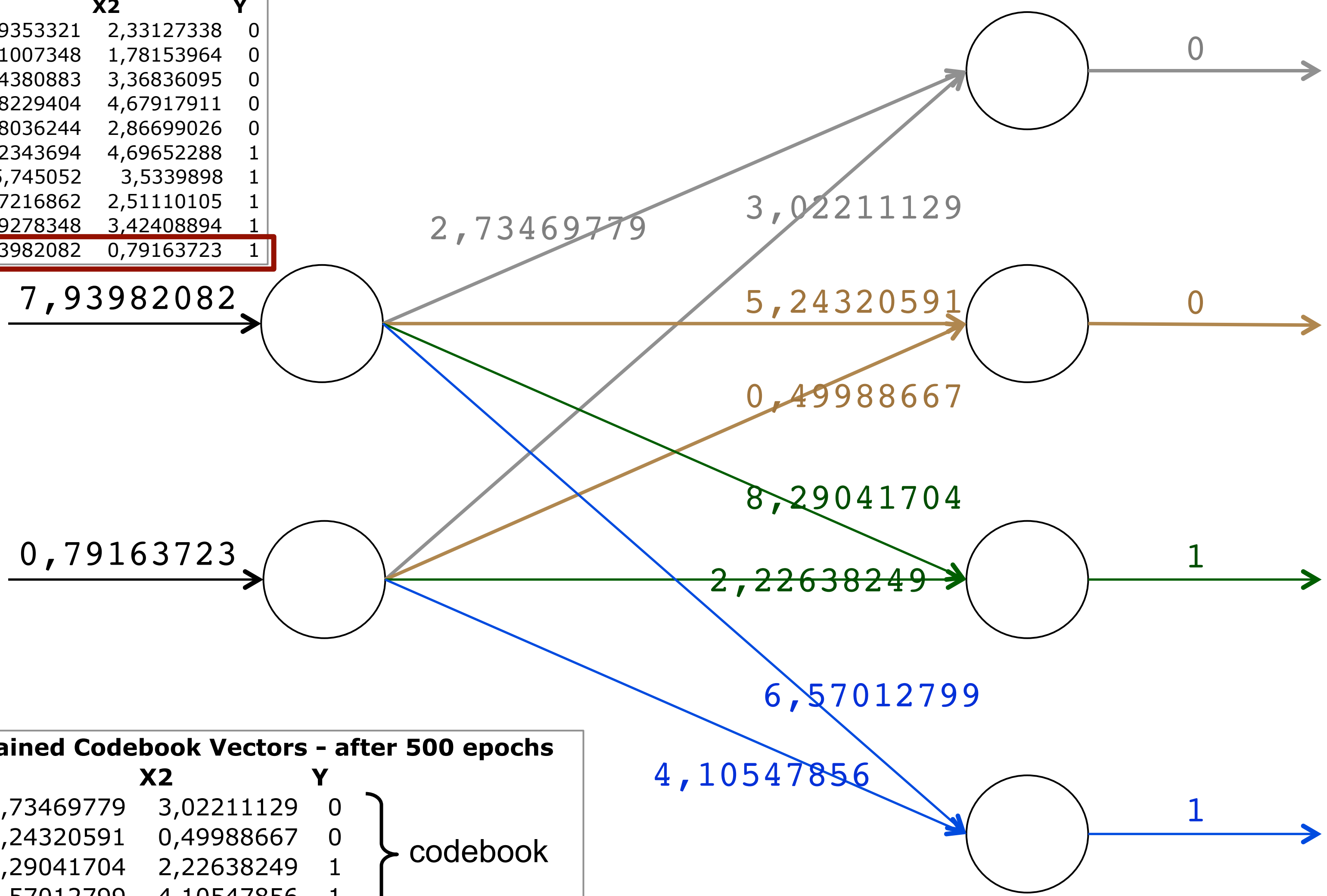
codebook

Paulo Trigo Silva @ ISEL

… the LVQ Neural Network NN structure

Dataset

X1	**X2**	**Y**
3,39353321	2,33127338	0
3,11007348	1,78153964	0
1,34380883	3,36836095	0
3,58229404	4,67917911	0
2,28036244	2,86699026	0
7,42343694	4,69652288	1
5,745052	3,5339898	1
9,17216862	2,51110105	1
7,79278348	3,42408894	1
7,93982082	0,79163723	1



Trained Codebook Vectors - after 500 epochs

X1	**X2**	**Y**
2,73469779	3,02211129	0
5,24320591	0,49988667	0
8,29041704	2,22638249	1
6,57012799	4,10547856	1

codebook

Paulo Trigo Silva @ ISEL

LVQ – some additional approaches

an alternative update-rule

Idea: update not only the BMU (closest cBv), but **update the two closest** cBv

additionally we may also impose that such update only occurs in case the **two closest cBv represent different classes**

other ideas

Frequency Sensitive Competitive Learning

the distance to a cBv is modified according to the number of data points that are assigned to this cBv

Size and Shape Parameters

associate each cVb with a cluster radius; update radius depending on how close the data points are

Fuzzy LVQ

exploits the close relationship to fuzzy clustering (an online version of it)

Extend LVQ to deal with with categorical values…

usually, a feature may be either numeric or categorical (nominal or ordinal)

nominal – values belong to a limited and set of categories without natural ordering
e.g., "arthritis", "asthma", "diabetes", "ulcers"

ordinal – values have particular order but unknown distance
e.g., "very-low", "low", "normal", "high", "very-high"

LVQ is originally designed for metric vector spaces (numeric features)

when extending LVQ to non-vector representation (i.e., to include categorical domain features) we find two main difficulties:

a. define the distance measurement, and
b. define the learning update rules

extend distance measurement – e.g., mismatch measurement on categorical features
extend update rules – is more complex and a "nice" approach is proposed in,
"*Extending Learning Vector Quantization for Classifying Data with Categorical Values*";
by Ning Chen and Nuno Marques; Communications in Computer and Information Science

Paulo Trigo Silva @ ISEL