# Module V: Self-Organizing Feature Maps

Pawel Chilinski

November 28, 2014

Examples in this document can be found in the source bundle attached to this document (or on github repository: `https://github.com/pawelc/neuralnet/tree/master/src/NeuralNet/exercises/module5`) and can be run after installing torch7 machine learning library: `http://torch.ch/`.

## Exercise 1

Implement the two-inputs SOM with 1D output lattice and test it on a square grid with random samples.

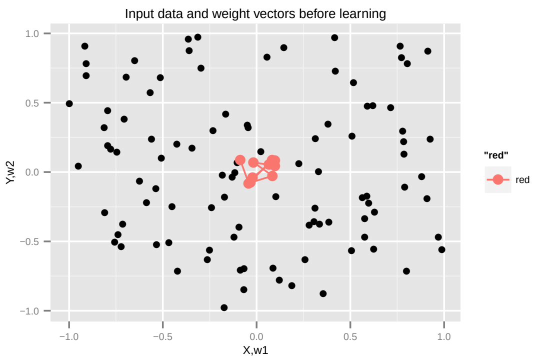
I implemented this exercise in lua script: /neuralnet/src/NeuralNet/exercises/module5/trainLattice.lua and .

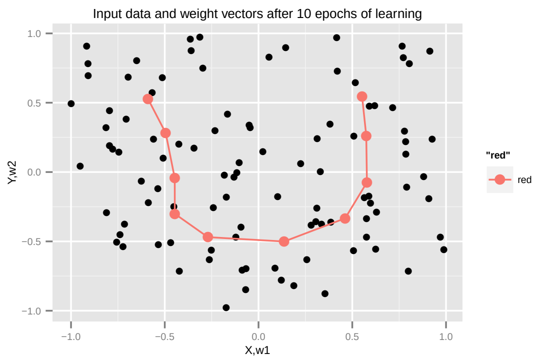
Generating input data composed of 100 random points:

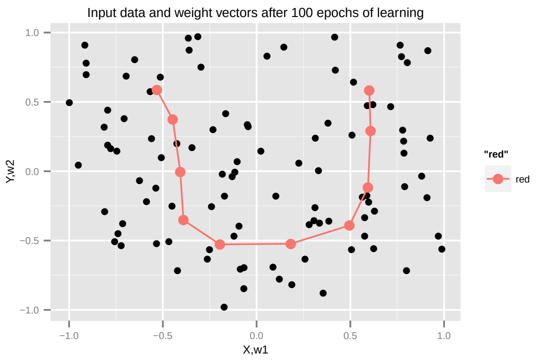
Computing weights of output neurons in 1D lattice for different number of learning iterations (epochs), effective width of the topological neighborhood of 1, learning rate of 0.1, time decay of 1000:

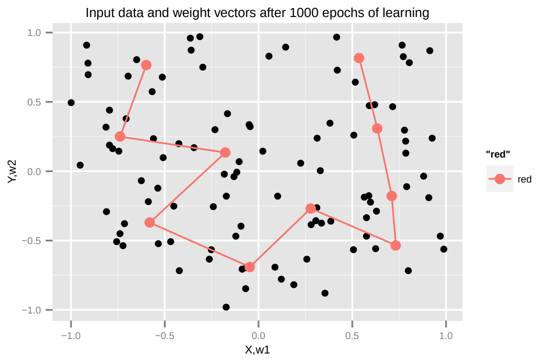
```
> set.seed(123)
> ex1Output1 <- runLua("module5/trainLattice.lua -w -e 0 -s 1 -l 0.1 -t 1000 -n '10,1'",data)
> ex1Output2 <- runLua("module5/trainLattice.lua -w -e 10 -s 1 -l 0.1 -t 1000 -n '10,1'",data)
> ex1Output3 <- runLua("module5/trainLattice.lua -w -e 100 -s 1 -l 0.1 -t 1000 -n '10,1'",data)
> ex1Output4 <- runLua("module5/trainLattice.lua -w -e 1000 -s 1 -l 0.1 -t 1000 -n '10,1'",data)
> ex1Output5 <- runLua("module5/trainLattice.lua -w -e 10000 -s 1 -l 0.1 -t 1000 -n '10,1'",data)
```

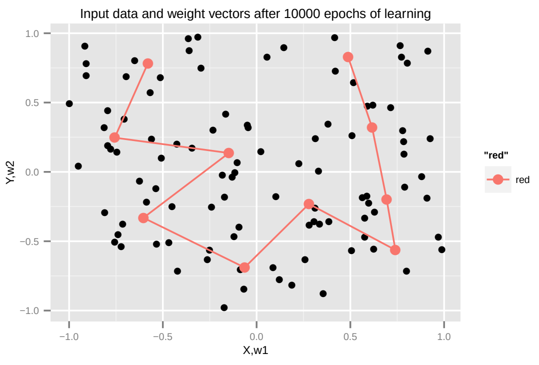
Showing learnt weights of 10 neurons after different number of learning iterations (epochs):







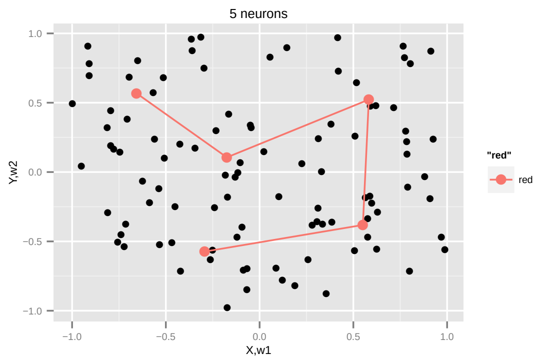


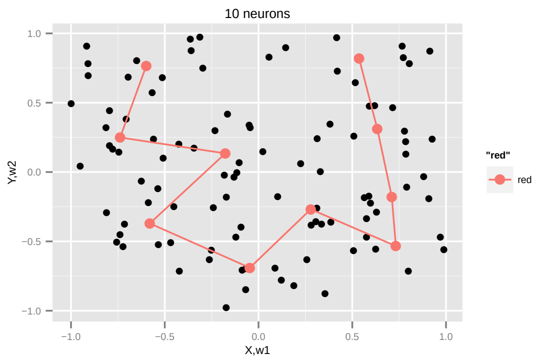


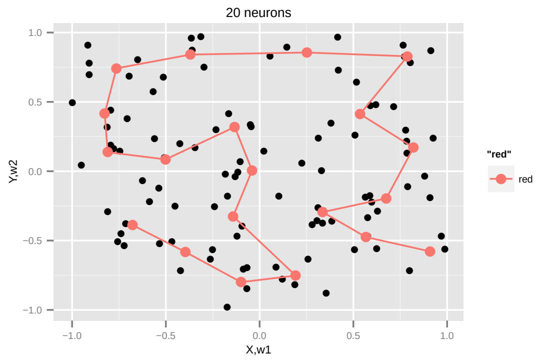
Computing weights of output neurons in 1D lattice for different number of output neurons, effective width of the topological neighborhood of 1, learning rate of 0.1, time decay of 1000, epochs of 1000:

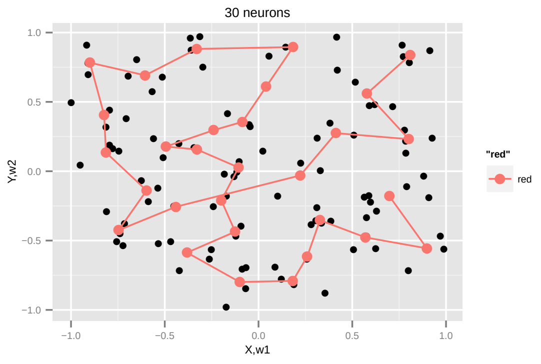
```
> set.seed(123)
> ex1OutputNeuron1 <- runLua("module5/trainLattice.lua -w -e 1000 -s 1 -l 0.1 -t 1000 -n '5,1'",data)
> ex1OutputNeuron2 <- runLua("module5/trainLattice.lua -w -e 1000 -s 1 -l 0.1 -t 1000 -n '10,1'",data)
> ex1OutputNeuron3 <- runLua("module5/trainLattice.lua -w -e 1000 -s 1 -l 0.1 -t 1000 -n '20,1'",data)
> ex1OutputNeuron4 <- runLua("module5/trainLattice.lua -w -e 1000 -s 1 -l 0.1 -t 1000 -n '30,1'",data)
> ex1OutputNeuron5 <- runLua("module5/trainLattice.lua -w -e 1000 -s 1 -l 0.1 -t 1000 -n '40,1'",data)
> ex1OutputNeuron6 <- runLua("module5/trainLattice.lua -w -e 1000 -s 1 -l 0.1 -t 1000 -n '50,1'",data)
```

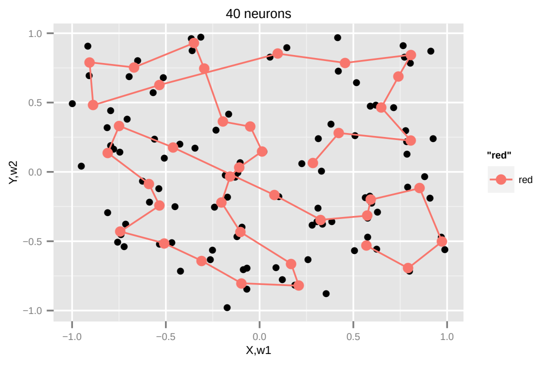
Showing learnt weights of 1D lattice with different number of neurons:

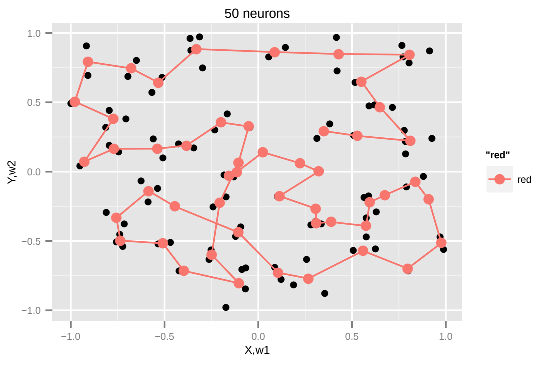












We can see on plots topological ordering of neurons' weights. The weight vectors start from small random weights and during the training they allign along the path that fits the input data.

## Exercise 2

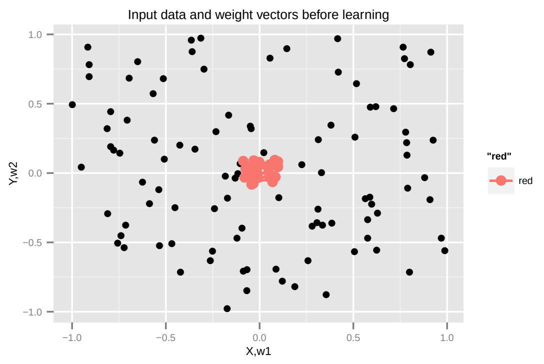
Implement the two-inputs SOM with 2D output lattice and test it on a square grid with random samples.

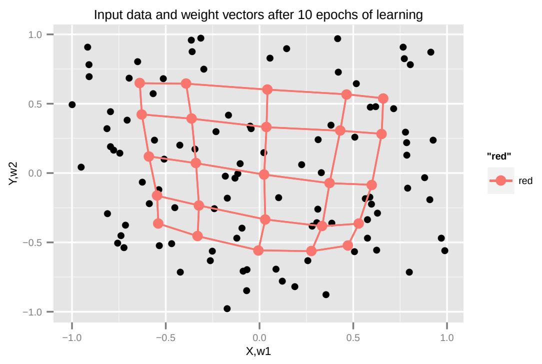
I implemented this exercise in the same script as exerice 1: /neuralnet/src/NeuralNet/exercises/module5/trainLattice.lua. It is invoked with different parameters to construct different size of 2D output lattice (parameter -n "5,5")

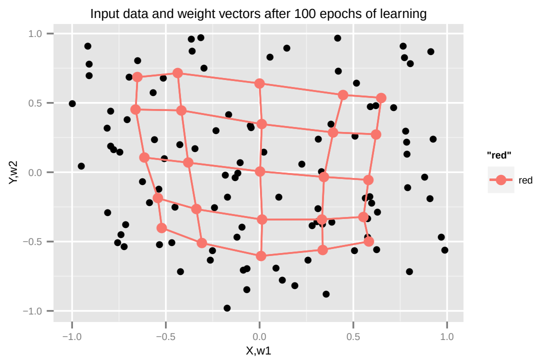
Reusing data generated in the previous example and computing weights of output neurons in 2D lattice for different number of learning iterations (epochs), effective width of the topological neighborhood of 1, learning rate of 0.1, time decay of 1000:

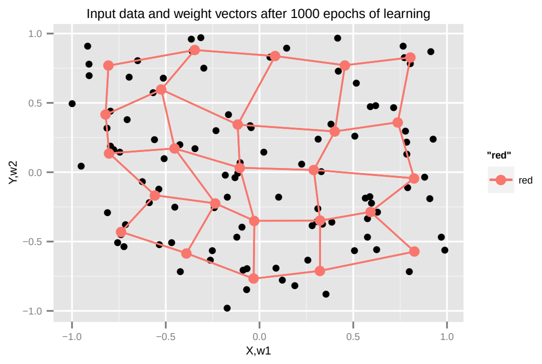
```
> set.seed(123)
> ex2Output1Epochs <- runLua("module5/trainLattice.lua -w -e 0 -s 1 -l 0.1 -t 1000 -n '5,5'",data)
> ex2Output2Epochs <- runLua("module5/trainLattice.lua -w -e 10 -s 1 -l 0.1 -t 1000 -n '5,5'",data)
> ex2Output3Epochs <- runLua("module5/trainLattice.lua -w -e 100 -s 1 -l 0.1 -t 1000 -n '5,5'",data)
> ex2Output4Epochs <- runLua("module5/trainLattice.lua -w -e 1000 -s 1 -l 0.1 -t 1000 -n '5,5'",data)
> ex2Output5Epochs <- runLua("module5/trainLattice.lua -w -e 10000 -s 1 -l 0.1 -t 1000 -n '5,5'",data)
```

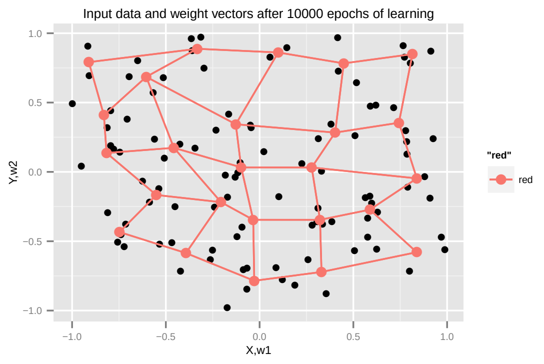
Showing input data and weight vectors with topollogy of the 2D lattice:











We can see that before learning the 2D lattice doesn't have regular structure but then starts to form the net. It looks that for 10 and 100 epochs net is evently spread but for more epochs it starts to fit data more precisely, we can suppose that the model overfits.

## Exercise 3

Basing on the previous exercise, extend the number of inputs to 4 and modify the learning rate and the forgetting factor to be exponentially decreasing. Perform cluster analysis on the standard Iris Flower dataset. Try a 3-by-3 output lattice.

Import iris data:

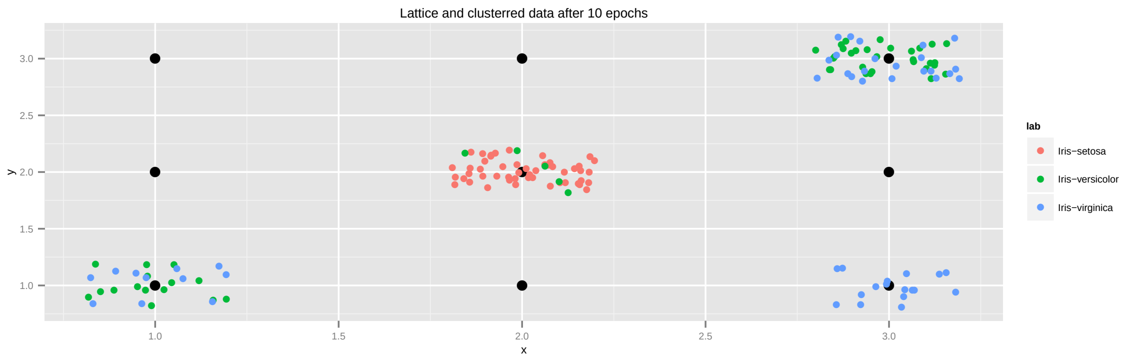
```
> iris<-read.table("iris.data.txt",header=F,sep=",",colClasses = c("numeric","numeric","numeric","numeric","fa
```

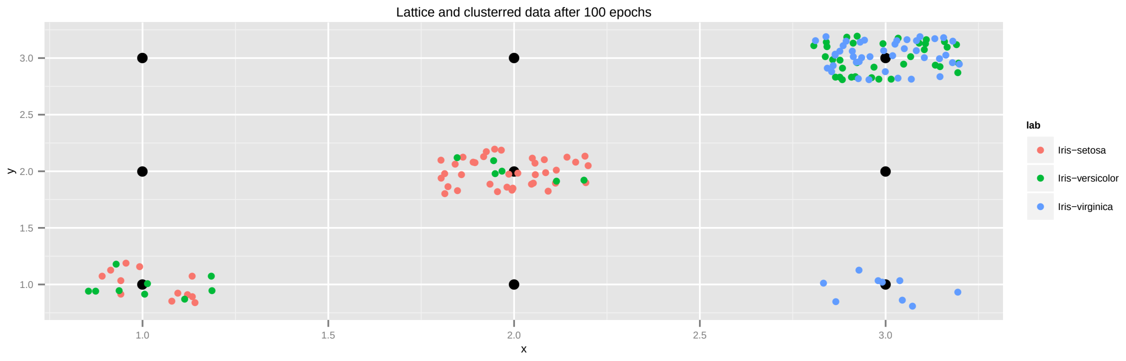
Running SOM learning algorithm with parameters:

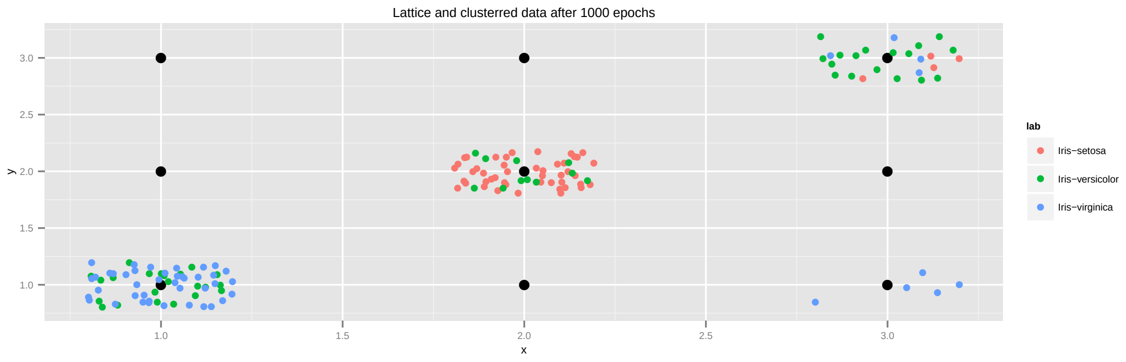
- Input dimension (-d) 4
- Return weights (-w)
- Number of epochs (-e), trying with 10,100,1000
- effective width of the topological neighborhood of 1 (-s)
- learning rate of 0.1 (-l)
- time decay of 1000 (-t)
- lattice size 3x3(-n) $\tau_2$ decay time in the learning rate and forgetting factor (-r)

```
> set.seed(123)
> ex3Output1 <- runLua("module5/trainLattice.lua -d 4 -w -e 10 -s 1 -l 0.1 -t 1000 -n '3,3' -r 100",iris[,c(1,
> ex3Output1DataClustered<-clusterPoints(ex3Output1,iris[,1:4],iris[,5])
> ex3Output2 <- runLua("module5/trainLattice.lua -d 4 -w -e 100 -s 1 -l 0.1 -t 1000 -n '3,3' -r 100",iris[,c(1
> ex3Output2DataClustered<-clusterPoints(ex3Output2,iris[,1:4],iris[,5])
> ex3Output3 <- runLua("module5/trainLattice.lua -d 4 -w -e 1000 -s 1 -l 0.1 -t 1000 -n '3,3' -r 100",iris[,c(
> ex3Output3DataClustered<-clusterPoints(ex3Output3,iris[,1:4],iris[,5])
```

Next we show topological map of the neurons and depict which data points are the closest to which neuron (in terms of Euclidian distance). For each datapoint we find the closest neuron and place this data point next to it. Also each data point is coloured using true label so we can assess the quality of clustering:







We can see that after 1000 epochs majority of the data points from each class is assigned to seperate neuron.

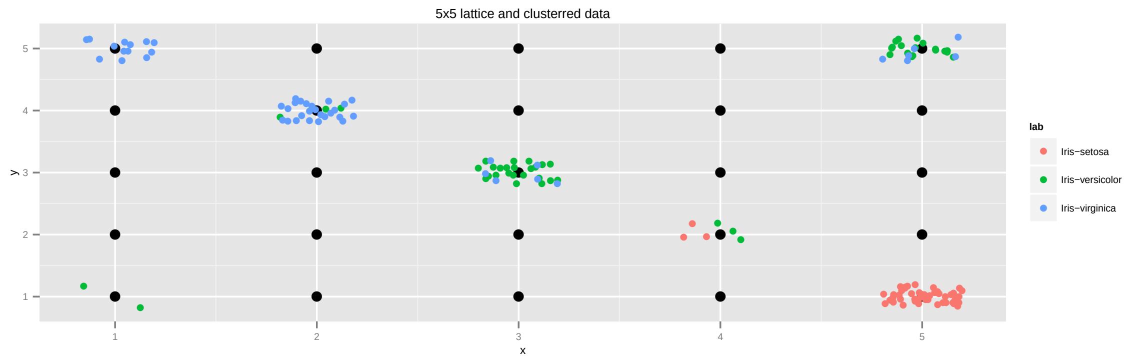
## Exercise 4

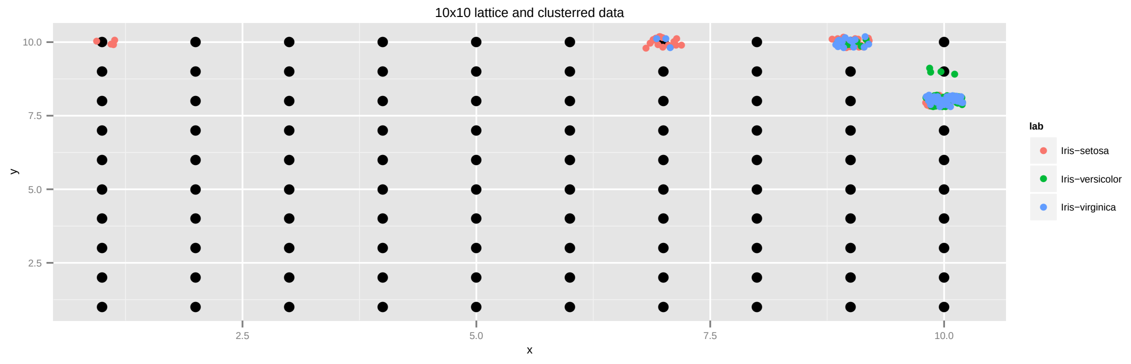
Basing on the previous exercise, use larger output lattices such that their deformations occur frequently. Try to avoid these deformations with different neighborhood sizes 1,2,3.

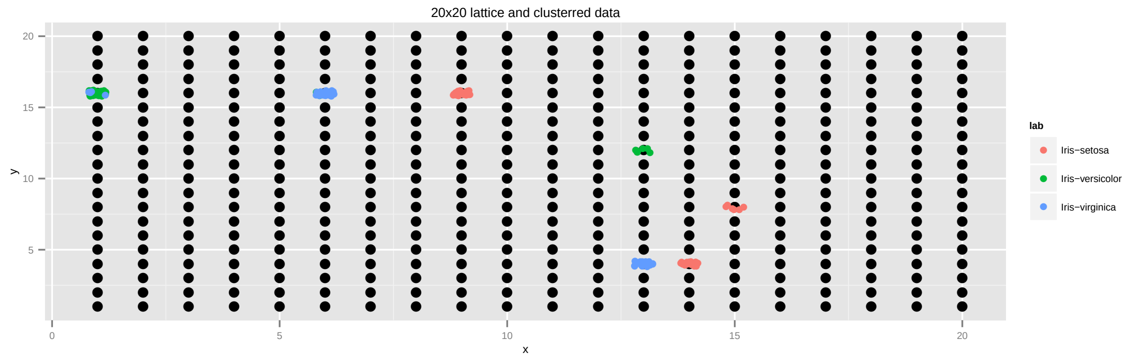
Learning SOM models for different sizes of output lattices of 5x5, 10x10 and 20x20 for 1000 epochs on iris data set:

```
> set.seed(123)
> ex4Output1 <- runLua("module5/trainLattice.lua -d 4 -w -e 1000 -s 1 -l 0.1 -t 1000 -n '5,5' -r 100",iris[,c(
> ex4Output1DataClustered<-clusterPoints(ex4Output1,iris[,1:4],iris[,5])
> ex4Output2 <- runLua("module5/trainLattice.lua -d 4 -w -e 1000 -s 1 -l 0.1 -t 1000 -n '10,10' -r 100",iris[,
> ex4Output2DataClustered<-clusterPoints(ex4Output2,iris[,1:4],iris[,5])
> ex4Output3 <- runLua("module5/trainLattice.lua -d 4 -w -e 1000 -s 1 -l 0.1 -t 1000 -n '20,20' -r 100",iris[,
> ex4Output3DataClustered<-clusterPoints(ex4Output3,iris[,1:4],iris[,5])
```

Showing the result of clusterring data to this models (the same type of plot as in Exercise 3):







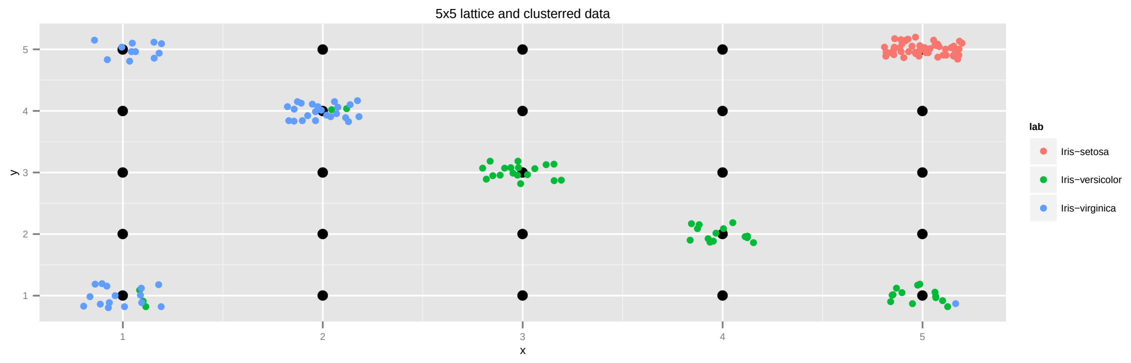
## Exercise 5

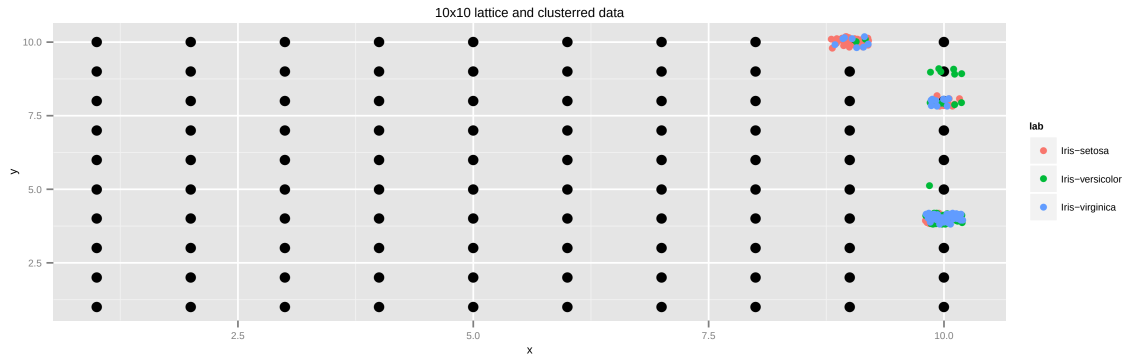
Perform comparative analysis of the Gaussian neighborhood function with the Mexican Hat function (implemented by any of the two formulae).

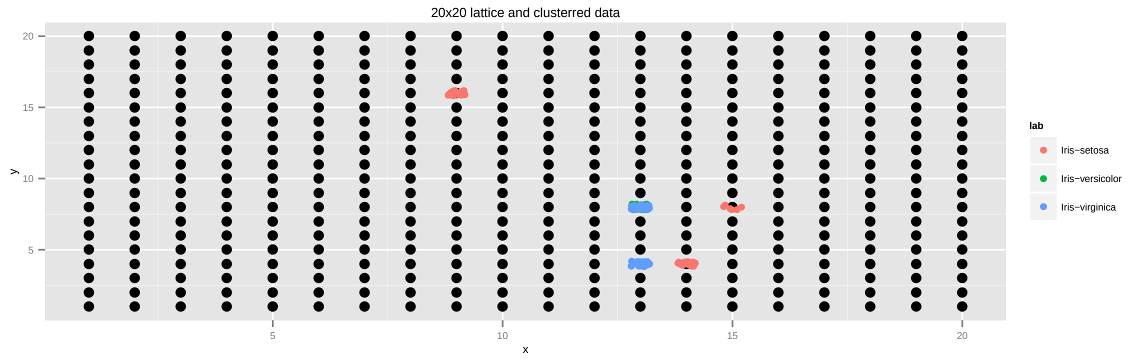
Learning SOM models for different sizes of output lattices of 5x5, 10x10 and 20x20 for 1000 epochs on iris data set, using Mexican Hat function:

```
> set.seed(123)
> ex5Output1 <- runLua("module5/trainLattice.lua -d 4 -w -e 1000 -s 1 -l 0.1 -t 1000 -n '5,5' -r 100 -o '{name
> ex5Output1DataClustered<-clusterPoints(ex5Output1,iris[,1:4],iris[,5])
> ex5Output2 <- runLua("module5/trainLattice.lua -d 4 -w -e 1000 -s 1 -l 0.1 -t 1000 -n '10,10' -r 100 -o '{na
> ex5Output2DataClustered<-clusterPoints(ex5Output2,iris[,1:4],iris[,5])
> ex5Output3 <- runLua("module5/trainLattice.lua -d 4 -w -e 1000 -s 1 -l 0.1 -t 1000 -n '20,20' -r 100 -o '{na
> ex5Output3DataClustered<-clusterPoints(ex5Output3,iris[,1:4],iris[,5])
```

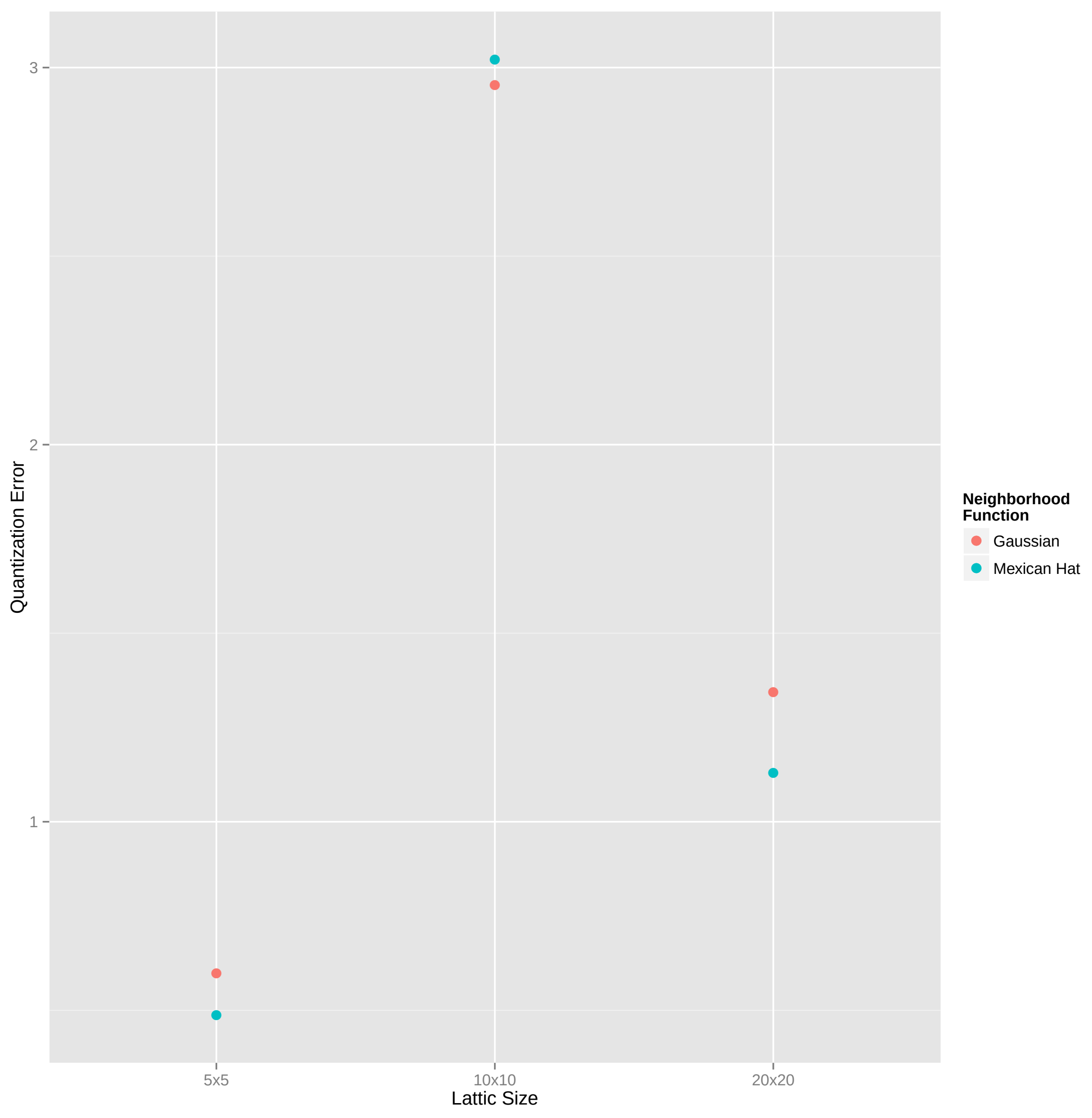
Showing the result of clusterring data to this models (the same type of plot as in Exercise 3):







Showing quantization error for models learned with Gaussian neighborhood function and with the Mexican Hat function for different sizes of lattices:


Quantization Error
3
2
1
5x5
10x10
20x20
Lattic Size
Neighborhood Function
Gaussian
Mexican Hat