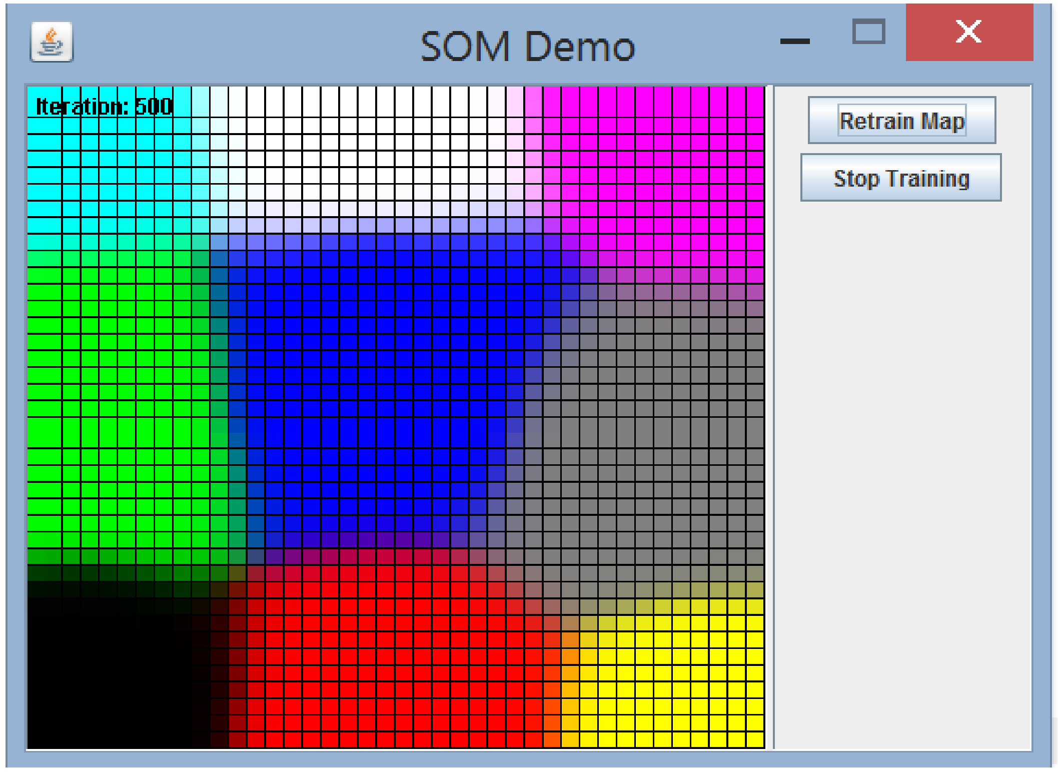
LAB 7 – SELF ORGANIZING MAPS

DATA MINING, SPRING 2014 | ANDERS HARTZEN (ANDERSHH@ITU.DK) & JENS ANDERSSON GRØN (JANG@ITU.DK)

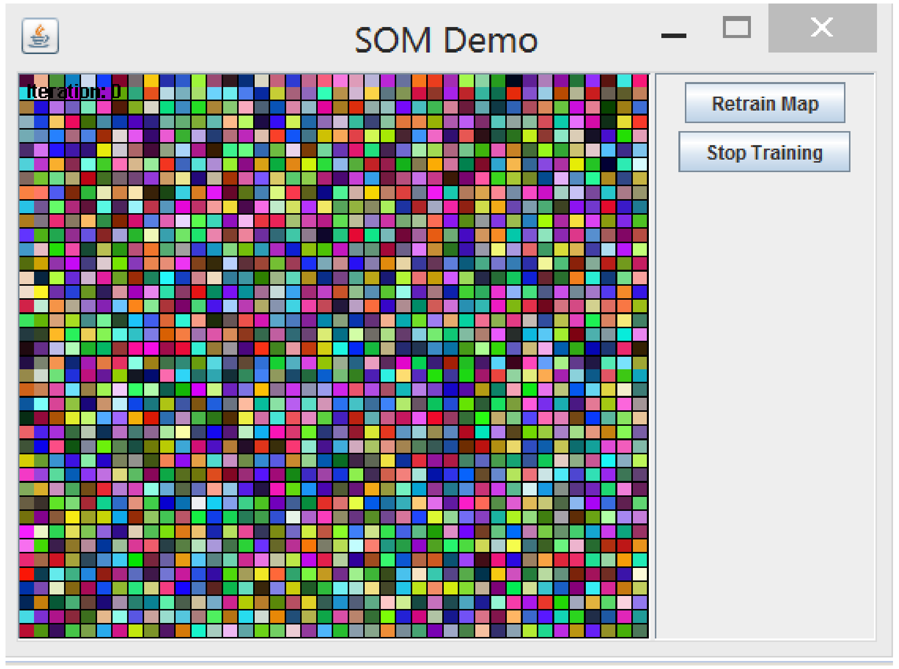
TODAY'S LAB

- In today's lab you will be working with self organizing maps to cluster colors.
  - You will implement a self organizing map that can do this.
- Code provided
  - Support structures included
  - Visualization
  - Based on this excellent tutorial:
    - http://www.ai-junkie.com/ann/som/som1.html

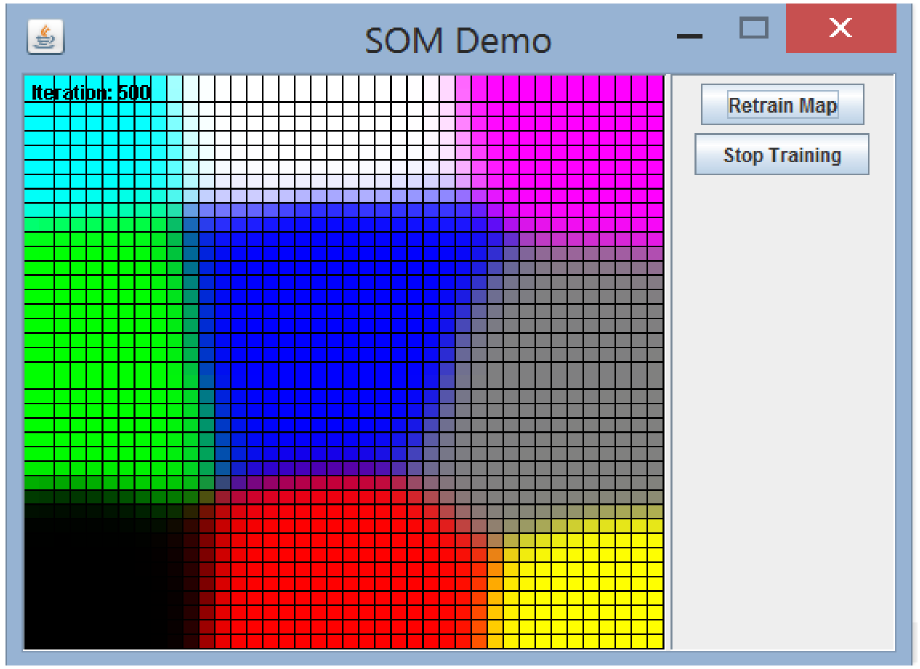


CODE PROVIDED - UI

Before

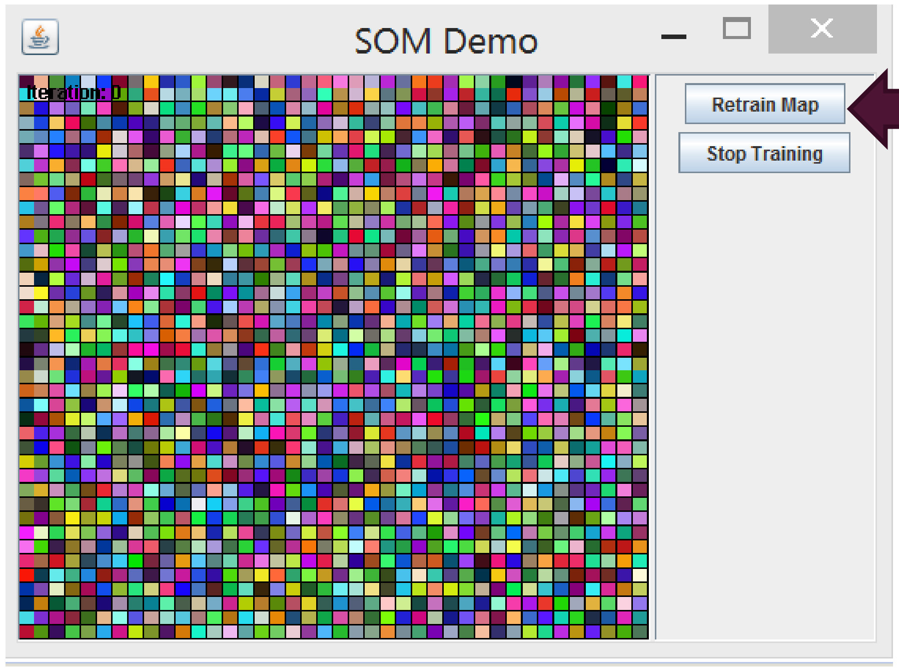


After

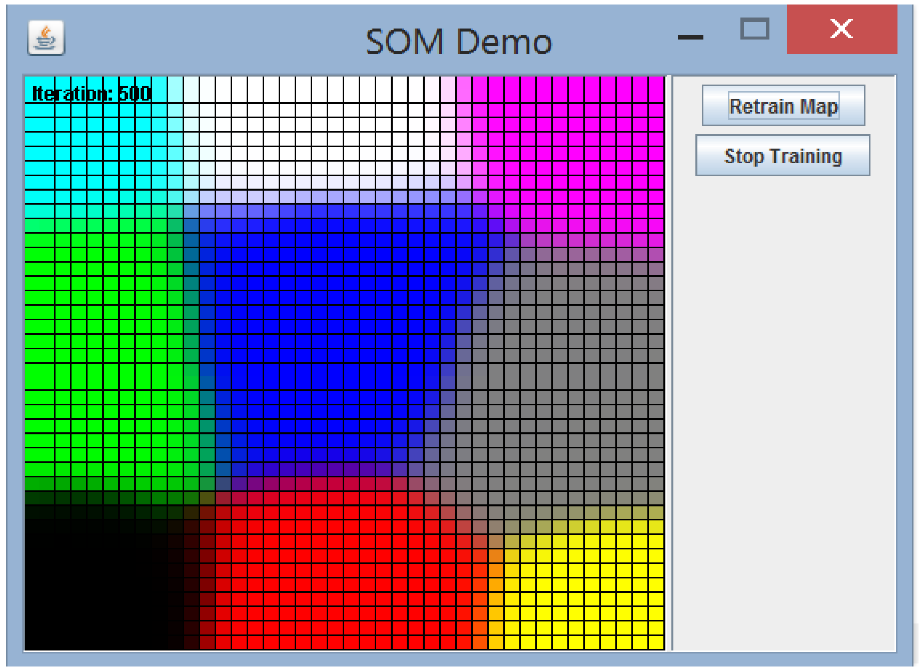


CODE PROVIDED - UI

Before



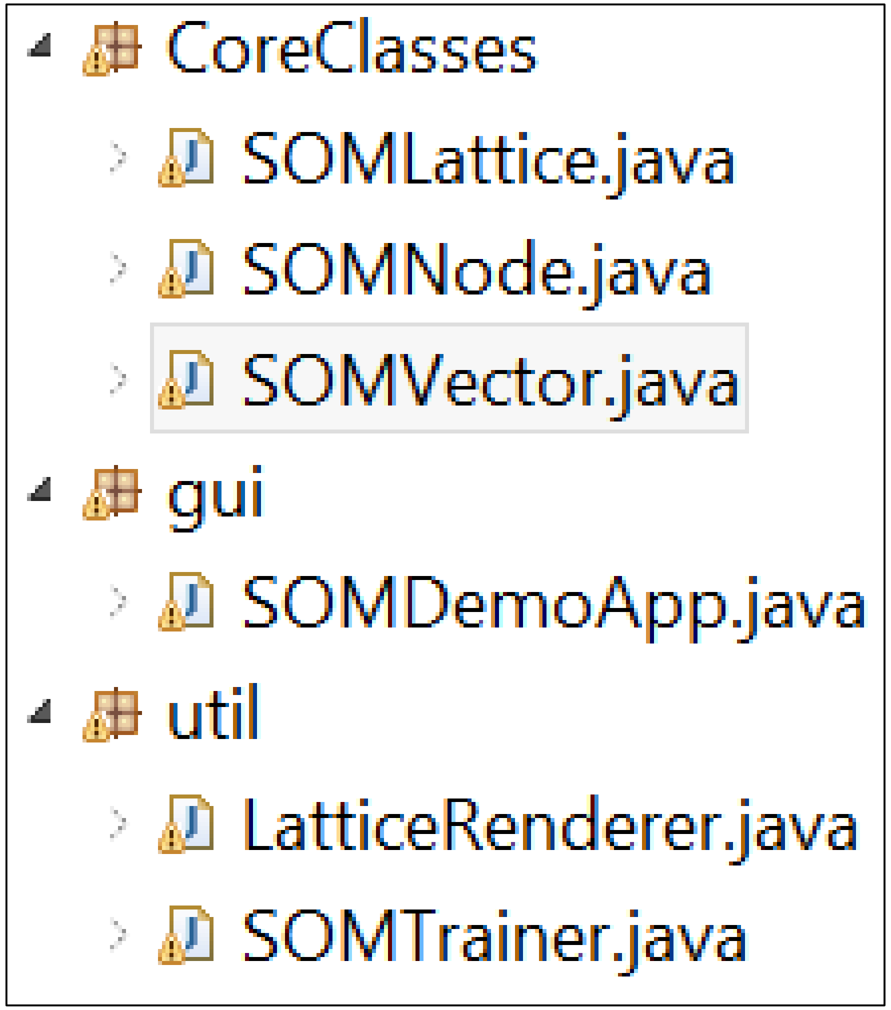
After



Start the training by clicking the "Retrain Map" button.

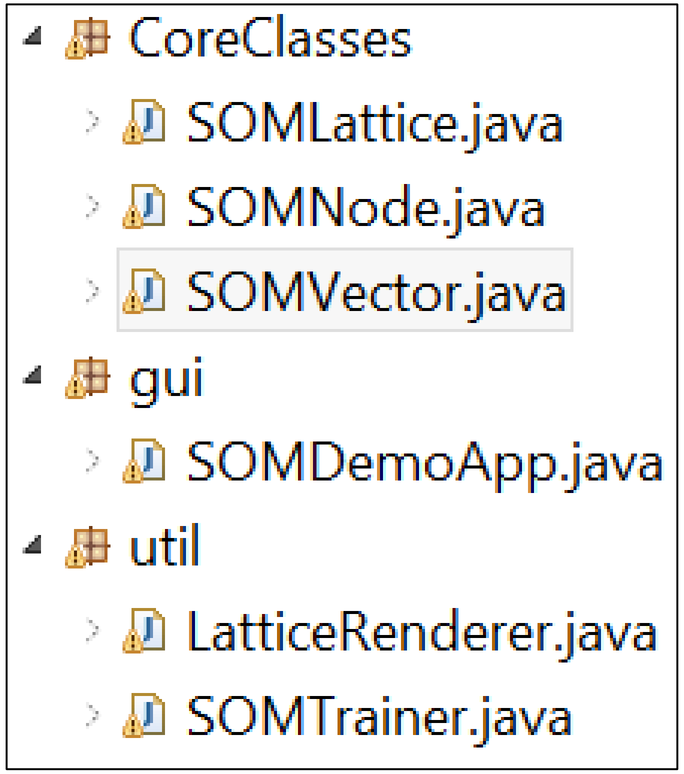
CODE PROVIDED - OVERVIEW

- CoreClasses
  - SOMLattice
    - Is where the SOM is
  - SOMNode
    - The nodes that the SOM is made up of
  - SOMVector
    - Used to contain weights and inputs
- Gui
  - SOMDemoApp
    - Contains Main method
- Util
  - LatticeRenderer
  - SOMTrainer
    - Has the SOM algorithm


CoreClasses
SOMLattice.java
SOMNode.java
SOMVector.java
gui
SOMDemoApp.java
util
LatticeRenderer.java
SOMTrainer.java

CODE PROVIDED – THINGS TO IMPLEMENT

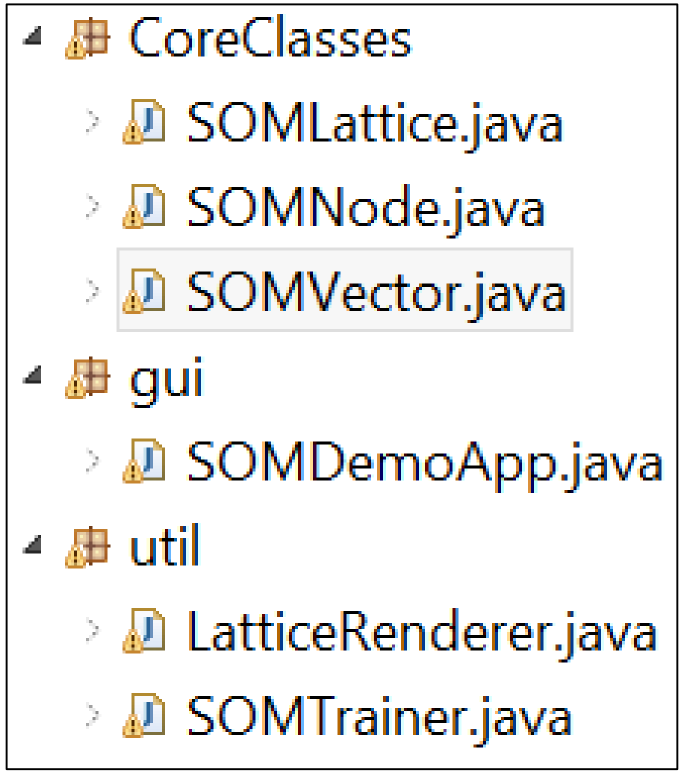
- SOMNode
  - adjustWeights
- SOMTrainer
  - run
    - Where you should implement the SOM training algorithm



```
CoreClasses
    SOMLattice.java
    SOMNode.java
    SOMVector.java
gui
    SOMDemoApp.java
util
    LatticeRenderer.java
    SOMTrainer.java
```

CODE PROVIDED – HELPFUL METHODS

- SOMLattice
  - getBMU
    - Already implemented using Euclidian distance
- SOMNode
  - distanceTo
    - Returns the squared distance between two SOMNodes
- SOMVector
  - euclideanDist
    - Returns the squared distance between two SOMVectors



```
CoreClasses
    SOMLattice.java
    SOMNode.java
    SOMVector.java
gui
    SOMDemoApp.java
util
    LatticeRenderer.java
    SOMTrainer.java
```

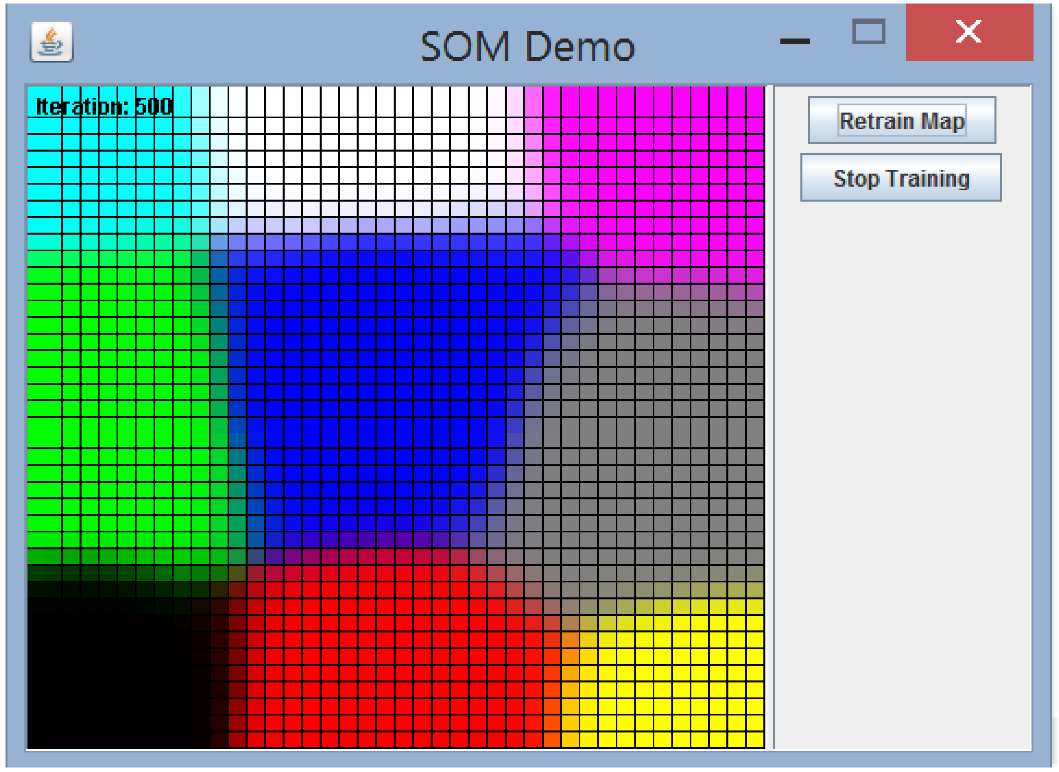
SOM ALGORITHM

1. Each node's weights are initialized.
2. A vector is chosen at random from the set of training data and presented to the lattice.
3. Every node is examined to calculate which one's weights are most like the input vector. The winning node is commonly known as the Best Matching Unit (BMU).
4. The radius of the neighbourhood of the BMU is now calculated. This is a value that starts large, typically set to the 'radius' of the lattice, but diminishes each time-step. Any nodes found within this radius are deemed to be inside the BMU's neighbourhood.
5. Each neighbouring node's (the nodes found in step 4) weights are adjusted to make them more like the input vector. The closer a node is to the BMU, the more its weights get altered.
6. Repeat step 2 for N iterations.

- From: http://www.ai-junkie.com/ann/som/som2.html
- Steps 1-3 already taken care of in the code

THE DATA

- Small dataset containing 9 different colors as RGB values
- Load in of data and set up of lattice is already done in the provided code.



PLAN OF ATTACK

- Download/set up the provided code and get an overview of it
- Start implementing the run method of the SOMTrainer class and the adjustWeights method of the SOMNode class.
- If you want a good step-for-step guide to the SOM algorithm check out http://www.ai-junkie.com/ann/som/som1.html
  - C++ examples – but don't blindly copy the guides implementation!

GOOD LUCK!