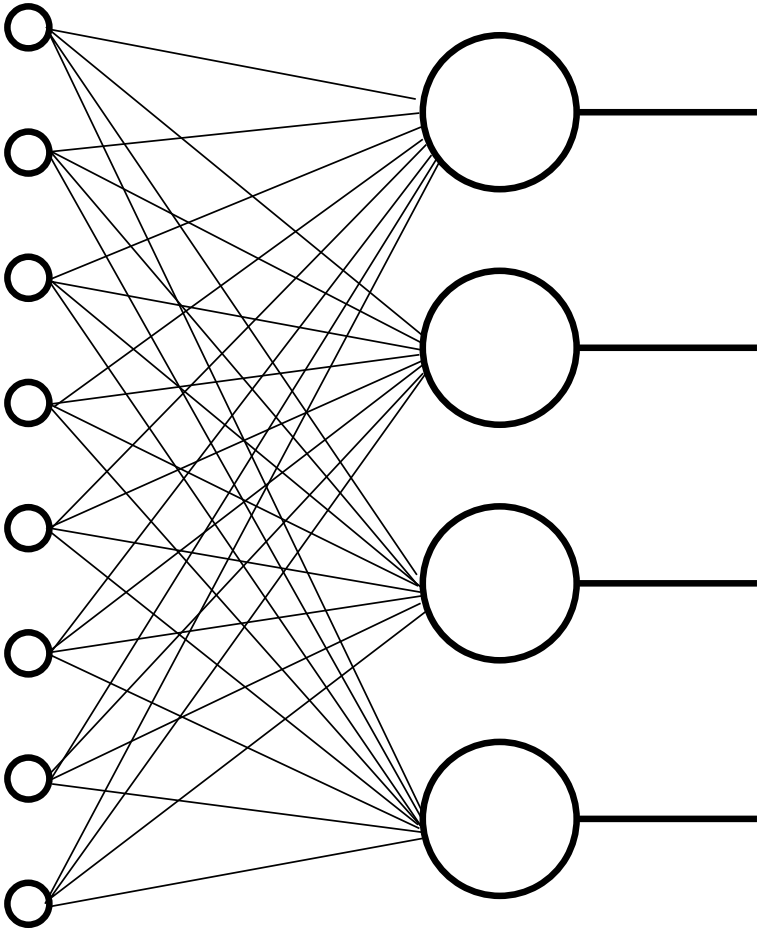
ANNEX 1: implementing a single-layer network

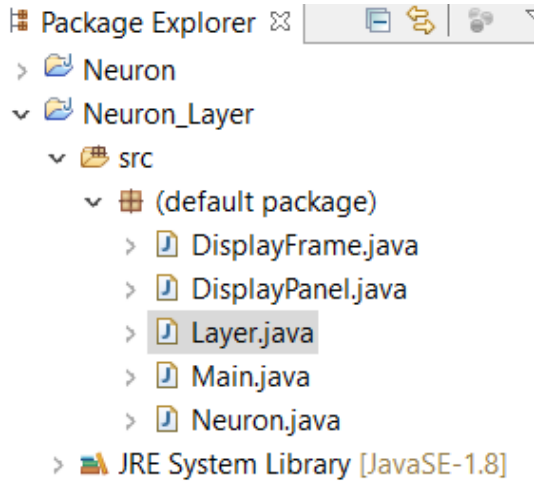


ANNEX 1: implementing a single-layer network

- **We will implement a single-layer network to recognize numbers**

- A layer is just a vector of neurons, with a vector collecting neurons' outputs
  - Several function to perform and reinforce each neuron

- Duplicate your project and rename it 'Neuron_Layer'
  - Right-click on Neuron project → copy, then right-click → paste
- Add a new class names 'Layer'
  - Right-click on 'default package' → New → class



- Close all classes in the middle part and open main.java from project Neuron_Layer

ANNEX 1: implementing a single-layer network

- The layer class must contain a vector of neurons and an output vector
  - We also add a parameter to measure performances
- Then, a constructor, using the number of neurons and input size as parameters

```
2  public class Layer {
3
4      public Neuron[] layer;
5      public float[] output;
6
7      public float delta=0;
8
9      public Layer(int nb, int size){
10
11         layer=new Neuron[nb];   // declare neuron vector
12         for (int i=0;i<nb;i++){ // initialize each neuron
13             layer[i]=new Neuron(size);
14         }
15
16         output=new float[nb];   // declare the output vector
17     }
18 }
```

ANNEX 1: implementing a single-layer network

- We implement a function to compute neurons' outputs and store results in output vector

```
18
19      public float[] compute(float[] img){
20
21          for (int i=0;i<layer.length;i++){
22              output[i]=layer[i].compute(img);
23          }
24
25          return output;
26      }
```

- And a function to perform the neurons' reinforcement
  - Note that expected result is a vector with a result for each neuron

```
29      public void learn(float[] img, int[] results){
30
31          delta=0;
32
33          for (int i=0;i<layer.length;i++){
34              layer[i].learn(img, results[i]);
35
36              delta+=Math.abs(layer[i].delta);
37          }
38      }
```

ANNEX 1: implementing a single-layer network

- The layer is complete !
  - We will create a layer of 10 neurons, one for each digit to recognize

- In Main class, replace the neuron with a layer

```
30    public Neuron neuron;    // neuron
```

→

```
30    public Layer layer;      // layer
```

- Then, in main function, replace the neuron initialization with a layer initialization
  - We construct a layer of 10 neurons

```
//////////////////////////////////////////////////
// initialization
//////////////////////////////////////////////////
neuron=new Neuron(size_x*size_y);
```

```
//////////////////////////////////////////////////
// initialization
//////////////////////////////////////////////////
layer=new Layer(10,size_x*size_y);
```

ANNEX 1: implementing a single-layer network

- The reinforcement of the layer brings very few changes
  - The expected result is defined as a vector : 1 for the digit of the image, 0 for the others :

```
58          // for each test image
59          for (test=0;test<matrixImages.length;test++){
60
61              // set output value
62              int[] expected=new int[10];
63              expected[matrixLabels[test]] = 1;
```

  - We then use layer's function instead of neuron's

```
65              // process neurons
66              layer.compute(matrixImages[test]); // get result
67
68              // reinforce neurons
69              layer.learn(matrixImages[test], expected);
70
71              // add delta to the error sum
72              sumdelta+=Math.abs(layer.delta);
73
74              display.repaint();
```

ANNEX 1: implementing a single-layer network

- For the tests, we change the interpretation of résults : the neuron with the strongest output define the ‘answer’ of the network
  - First, we compute neurons’ output

```
99          // test each image of the dataset
100         for (test=0;test<matrixImages.length;test++){
101
102             // process neurons
103             layer.compute(matrixImages[test]);
104
```

  - Then, we get the network output and count errors (you can copy this code part)

```
float max=0;
int imax=0;
for (int i=0;i<10;i++){
    if (layer.output[i]>max){
        max=layer.output[i];
        imax=i;
    }
}

System.out.println(matrixLabels[test]+" -> "+imax);

// count errors
if (matrixLabels[test]!=imax) errors++;
```

ANNEX 1: implementing a single-layer network

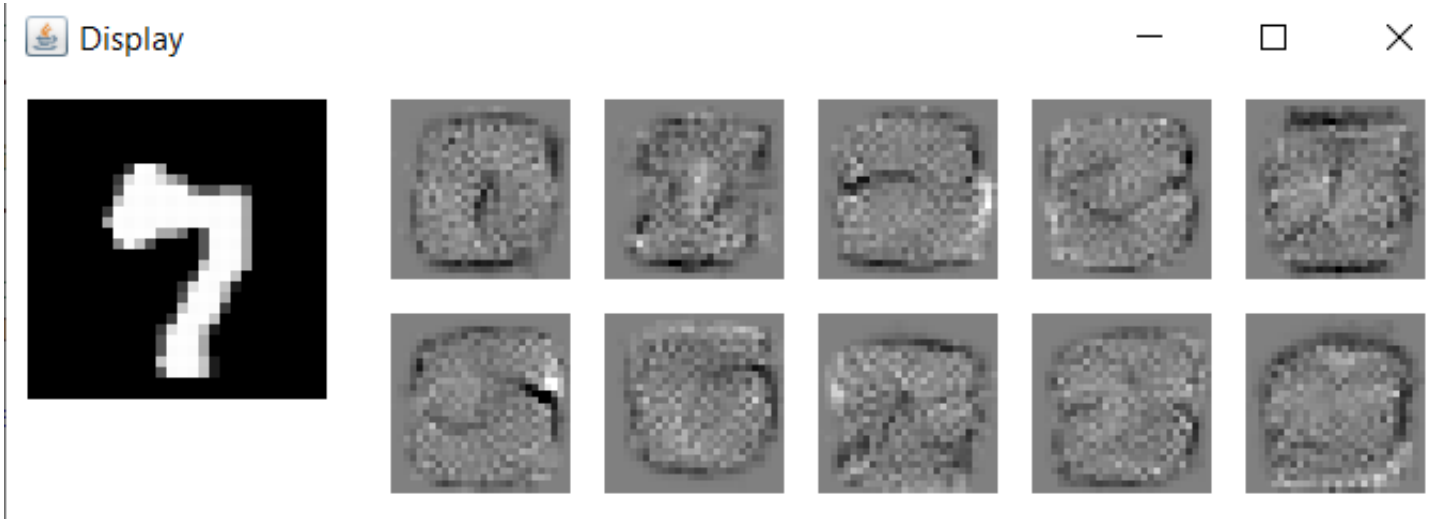
- Finally, in DisplayPanel, deactivate the second part of paintComponent function (add '/*' on line 34 to comment this part) and uncomment the third part (remove '/*' on line 44) :

```
33
34          /*for (int i=0;i<Main.size_x;i++){
35              for (int j=0;j<Main.size_y;j++){
36                  val=(int)(main.neuron.synaps[i+Main.size_x*j]*50)+128;
37                  if (val<0) val=0;
38                  if (val>255) val=255;
39                  g.setColor(new Color(val,val,val));
40                  g.fillRect(180+3*i, 10+3*j, 3, 3);
41              }
42          }/**/
43
44          for (int n=0;n<10;n++){
45              for (int i=0;i<Main.size_x;i++){
46                  for (int j=0;j<Main.size_y;j++){
47                      val=(int)(main.layer.layer[n].synaps[i+Main.size_x*j]*50)+128;
48                      if (val<0) val=0;
49                      if (val>255) val=255;
50                      g.setColor(new Color(val,val,val));
51                      g.fillRect(180+3*i+100*(n%5), 10+3*j+100*(n/5), 3, 3);
52                  }
53              }
54          }/**/
55      }
```

- Now, test your application !

ANNEX 1: implementing a single-layer network

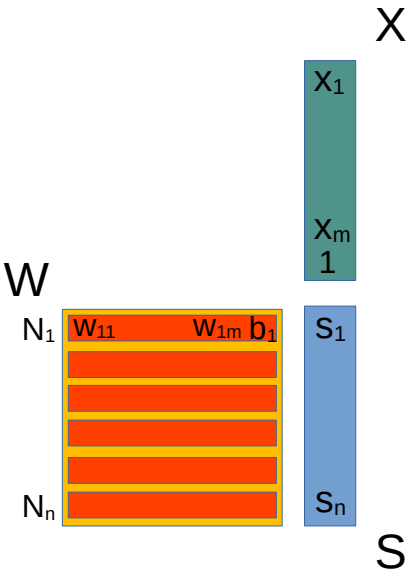
- **Results**




```
errors : 904
```

ANNEX 1: implementing a single-layer network

- **Algorithmic optimization**
- **The layer class calls instances of neurons (not optimal)**
- **For each neuron *n* : $s_n = W_n . X$**
- **From layer perspective, it is possible to gather weight vectors of neurons in a single matrix X and results in an output vector S :**
  - **$S = W . X$**



- **Some libraries have optimized function for matrix manipulation**