1 One-dimensional Kohonen network

1.1 a)

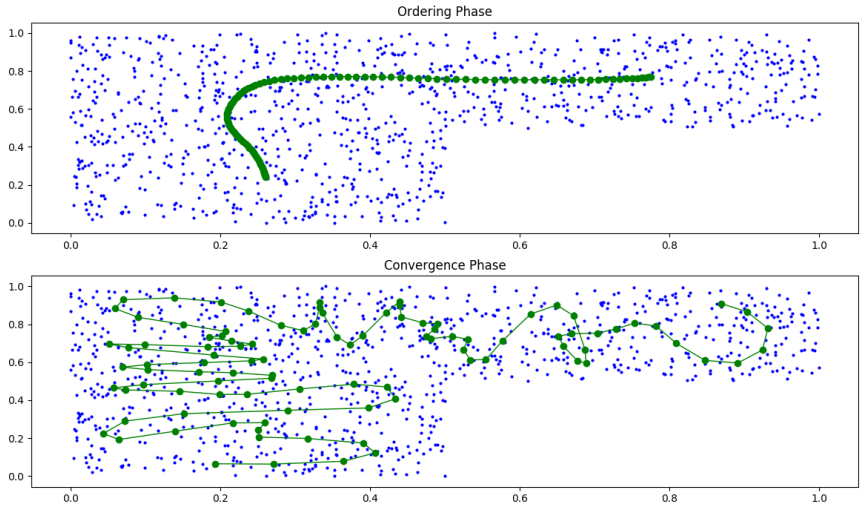

Figure 1: Plot of the weights (green) and the generated patterns (blue) after the different phases with $\sigma_0 = 100$

The network does learn the properties of the distribution and adjusts the output weights accordingly as seen in the picture above. In the ordering phase the output weights updates in a "smooth" way even though it's learning from the patterns. The convergence phase is more of a process of fine-tuning the output weights to the input data as seen in the last picture, even though this process is not as smooth as the ordering phase.

1.2 b)

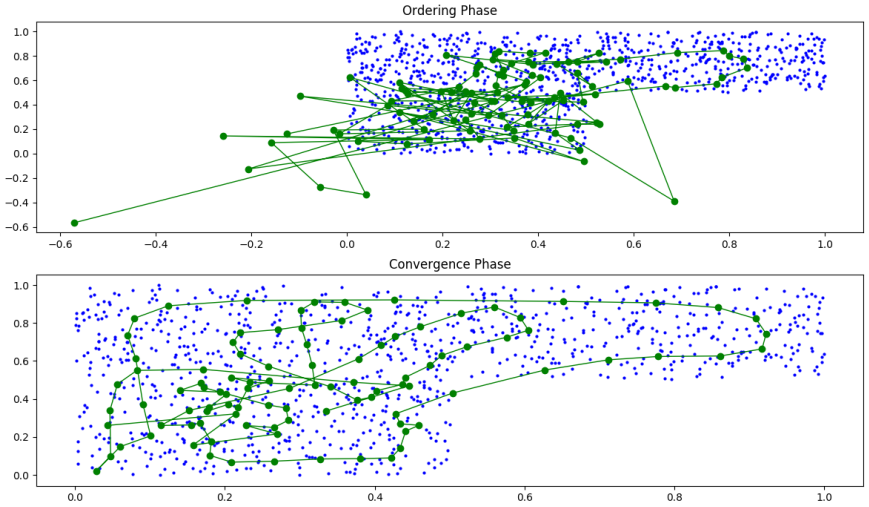

Figure 2: Plot of the weights (green) and the generated patterns (blue) after the different phases with $\sigma_0 = 5$

There is no clear order obtained in this case as seen in the picture above. Despite this, the network manages to learn and adapt to the input pattern and gets quite close to the result seen in the previous case, especially after the convergence phase. The way we choose our learning parameter and sigma is especially important and makes a big difference in our two output cases. In our case we are given the parameters but in real-life scenarios there is much trial and error to find values that result in a well-performing network.

2 Unsupervised Oja's learning with one linear unit

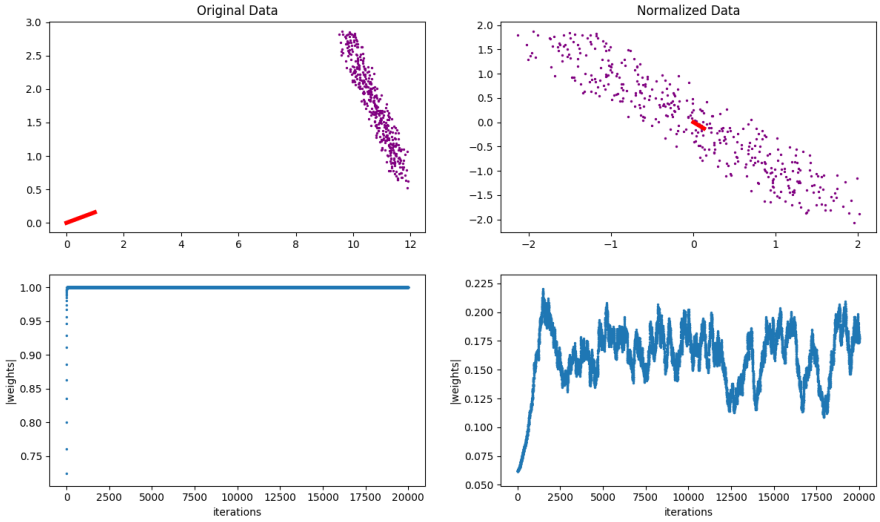

Figure 3: Plot of the weights (red), the input data (purple) and the modulus of the weight vector (blue)

We can see by comparing our results that the modulus of the weight vector converges to 1 when running on the original data, which doesn't happen when running on normalized data. Since this is the first of three properties of a weight vector to which the network is expected to converge, we do know that we don't converge in the case with normalized data.

3 Unsupervised simple competitive learning combined with supervised simple perceptron

After some discussion, we decided to not split our data set into training- and testing data. We took this decision after we plotted the given data for this task and realized that it was quite obvious that the data were generated from mathematical distributions and not real data. Training with all our data will therefore only make our model more accurate, and we will not experience the disadvantages with overfitting.

3.1 a

Our best experiment with 4 neurons gave us an average classification error of 0.032375. After the unsupervised simple competitive learning, the weights of our Gaussian neurons were following:

$$\begin{pmatrix} -6.41740939 & 5.36536793 \\ 16.73233576 & -0.23593904 \\ 4.61508451 & 7.27280455 \\ 6.70788162 & -2.49057948 \end{pmatrix}$$

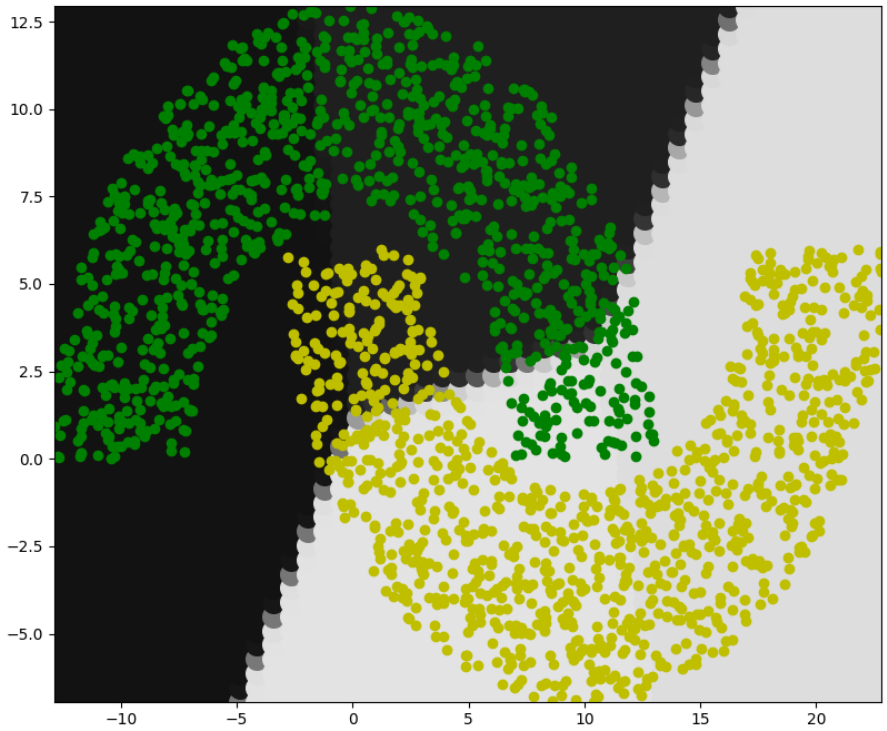

Figure 4: Plot of the input data and decision boundary with 4 Gaussian neurons

With 4 neurons our model are not able to classify the data especially good. Even though we get an average error of around 3.2%, which usually is not that bad, we clearly see on

our decision-boundary that our model doesn't perform good enough. Our input data is too complex to represent with this amount of Gaussian neurons.
The unsupervised network in our model clusters the data and sends the output to the supervised network, which then acts perceptron and does the classification of the clustered data. Since we don't get a especially detailed cluster of our input with only 4 Gaussian neurons, our classification in the supervised network isn't able to perform as good as we expect.

3.2 b

Our best experiment with 10 neurons gave us an average classification error of 0.0003. After the unsupervised simple competitive learning, the weights of our Gaussian neurons were following:

$$\begin{pmatrix} 9.80250726 & -3.68018599 \\ -7.31883889 & 7.16076731 \\ 19.75246521 & 2.96505811 \\ 9.57427408 & 3.44711739 \\ 4.39754371 & -1.93107159 \\ 15.02716904 & -2.37800023 \\ 4.75191332 & 8.86723937 \\ -9.86685303 & 2.41246487 \\ -2.03306545 & 10.10086259 \\ 0.18621978 & 2.91234529 \end{pmatrix}$$

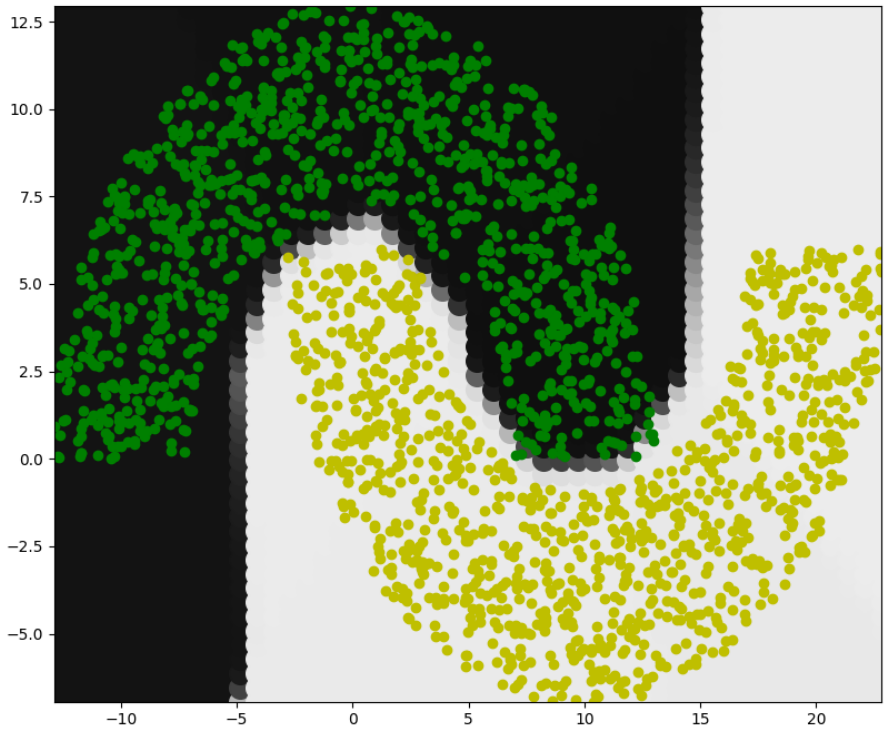

Figure 5: Plot of the input data and decision boundary with 10 Gaussian neurons

Training our model with 10 Gaussian neurons instead makes it possible for the network to classify more complex data, as the data in this task. This is possible because our unsupervised model can generate a more detailed cluster which makes it easier for the supervised classification to perform well.

3.3 c

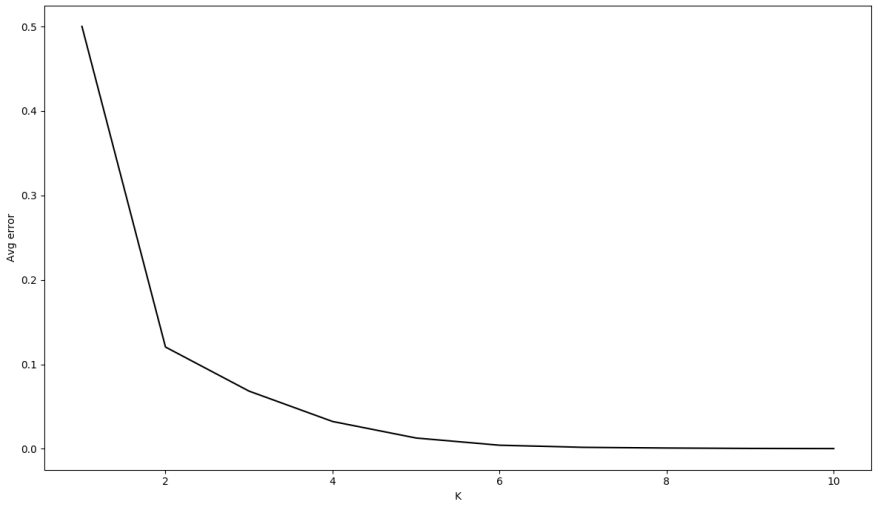

Figure 6: Plot of best avg error for different amount of neurons

The more Gaussian neurons our weights contain, the better the model performs. Since each neuron uses a nonlinear activation function in the simple perceptron network, it's possible for a model with more neurons to adapt to complex data. Since the detailness of the clustered data we send from our unsupervised network to our supervised network depends on the size of our Gaussian neurons, our model performs better for each neuron we add.