Exploration of Self-Organising Maps on Handwritten Digits

Bertrand Champenois, Ciprian I. Tomoiagă

***Abstract*—This project explores Kohonen's Self-Organising Maps [1] (SOM) as detailed in the lectures. We will perform different experiments on a subset of the MNIST database of handwritten digits, observing the effect of the algorithm's parameters. We start with fixed recommended values and gradually improve on them, at each step detailing the advantages and weaknesses of different settings and justifying our choices.**

I. Introduction

Being an exploratory task, we work with a subset consisting of 2000 samples distributed equally among the digits $\{2, 3, 5, 7\}$. This allows for smaller maps, with faster convergence, which are easier to inspect visually.

While the complete Kohonen algorithm uses a dynamic learning rate and neighbourhood function in order to ensure convergence, in the first parts we fixed these parameters to discrete values in order to better understand their influence on the map. This is addressed in the last part of this experiment, where both decrease as a function of time.

Finally, since the SOM algorithm is sensible to scaling of the features, we standardise the data before training the network to ensure all features are give equal importance. For better viewing, we undo this transformation for the images displayed in this report.

II. Learning rate and convergence

With the Gaussian neighbourhood width fixed at $\sigma = 3$, we vary the learning rate η and the number of steps $tmax$ to perform. We propose several heuristics for automatically detecting convergence of the map:

a) *Minimal movement* – At each step, compute the total movement m generated by that example over all neurons j: $m = \Sigma_j \|g(x, j) * \eta * (x - j)\|$ and take a moving average over the past w steps. Since, in expectation, the neurons should lie at the center of their cloud, we would expect this average to move towards 0. However, this does not work well due to the online nature of the update where, after the initial setup of the neurons, each example brings only a *small* improvement to the solution.

b) *Map stabilisation* – Evolving from the previous measure, we track map evolution but only at every epoch ($t = 1000$ iterations). After convergence, we expect that absolute differences between consecutive epochs will not change much. This seems to work for a correct choice of parameters σ and η, although it still oscillates.

c) *Distance from data* – Define score of a map $s = \Sigma_i (x_i - w_i)^2$ where x_i is a data point and w_i the winning neuron assigned to it. Note this is still unsupervised since no labels are used in the process. Also, being more expensive, we only evaluate it at epochs $t = 50$ steps.

d) *Probabilistic population test* – by treating the map and the data as two population samples we can use probabilistic tests to detect convergence [2]. For the imposed restrictions, the 6×6 map size is too small for the t-test to work well.

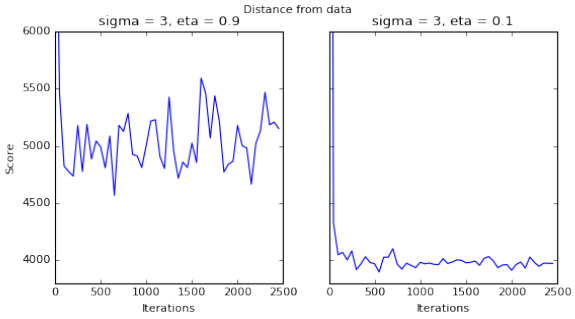


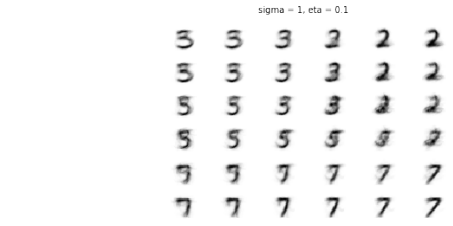
Figure 1. The *Distance to data* is a good measure for inspecting convergence.

Finally, we settled on criterion *c)* since it copes well with different sizes of the map and is less sensitive to correct choice of parameters. The steep descent in the first few hundred iterations is expected as it corresponds to the initial transition of the neurons from random values to examples of their classes. We chose to present all data samples before the true convergence test which, for a chosen set of parameters, indicates a good $tmax$ to stop the learning.

We note from figure 1 that bigger learning rates make our score function wildly oscillate and, upon inspection of the results (figures 2b and 2c), we see the map forgets quickly and only captures some recent examples. A small learning rate allows the map to converge but not always to a good representation, depending on the value of σ.

III. The neighbourhood function

In our implementation the neighbours of the winning neuron are also updated with a gaussian discounting factor parametrised by the width σ of this function. We observe that upon a good convergence (figure 2a), neurons of the map capture generalised models of the data and their variations. However, having a big σ results in a too general map that fails to capture the variance in the data, as in figure 2c.



(a)

(b) (c)

Figure 2. The neighbourhood function and the learning rate influence the quality of the map.

Alternatively, we could have used a difference of gaussians as neighbourhood function which would have enforced only small patches of resembling neurons.

IV. Digit assignment

Assuming the training resulted in a well converged map, we now try to assign labels to each neuron. The method we chose is a simple and efficient majority vote over all inputs. As such, each neuron is assigned a counter for each possible label and every time it is the winner of an input digit, the counter corresponding to that digit's label is incremented. Figure 3b shows the counters associated with each neuron from the map in figure 3a. The final label is given by the counter with the highest value.

V. Parameters estimation

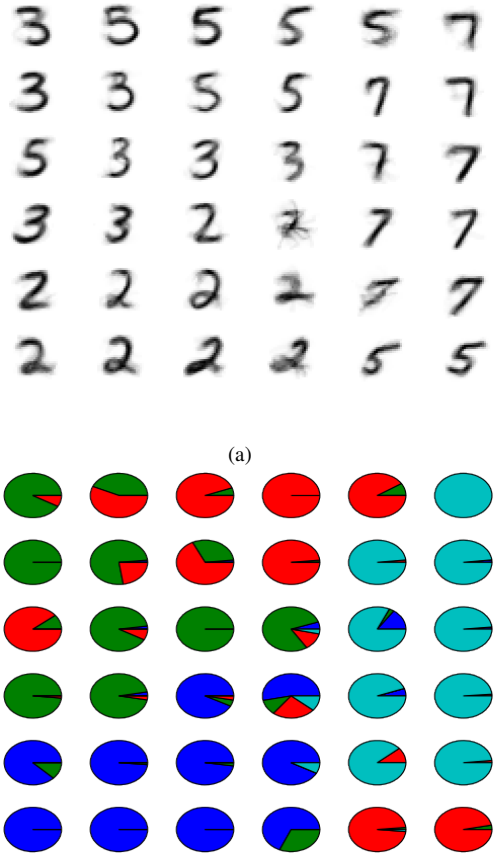
In this section we explore different sizes of the Kohonen map and then different widths of the neighborhood function.

A. Kohonen map size

First, we fix $\sigma = 3$ and explore different map sizes. Figure 4 shows a 6×6 map. The result is very blurry since the Gaussian variance is very large compared to the inter-distance between neurons. Thus, during an update, it will have a high impact on the winner's neighbors. Consequently, it is difficult to predict the digits.

Then, while keeping the same σ, we build a 8×8 map and observe the effect (figure 5). The result is far better on the map's corners, since they have less neighbours, thus less external influences. The center presents the same problem as earlier.

Finally, we build a 10×10 map to confirm our observations (figure 6).



(a)

(b)

Figure 3. The *Digit assignment*

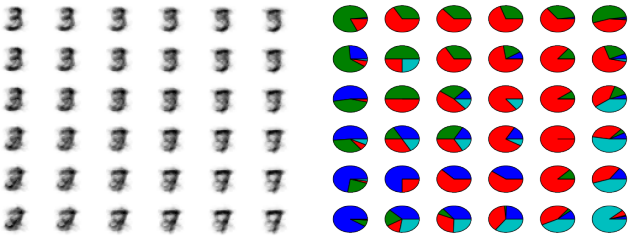


Figure 4. A 6×6 map and its prediction

To conclude this subsection, we see that the bigger the map size is for a fixed σ, the better it will converge as we move away from the center. So, we are positive that the optimal width depends on the size of the Kohonen map.

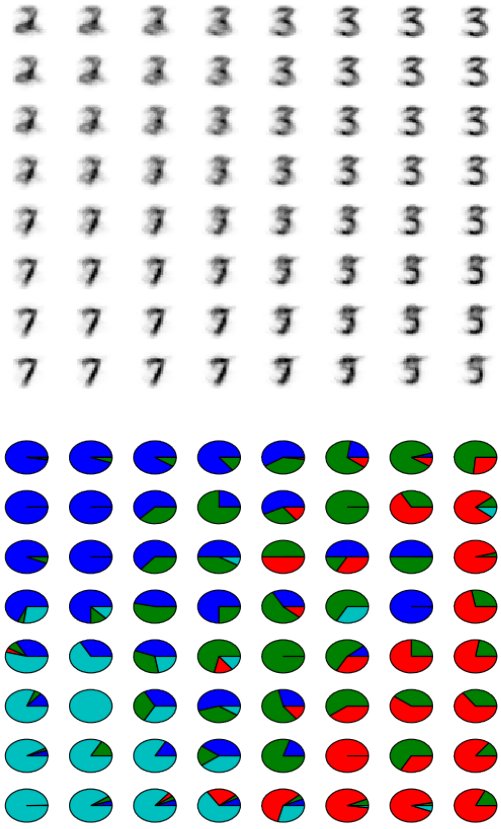


Figure 5. A 8×8 map and its prediction

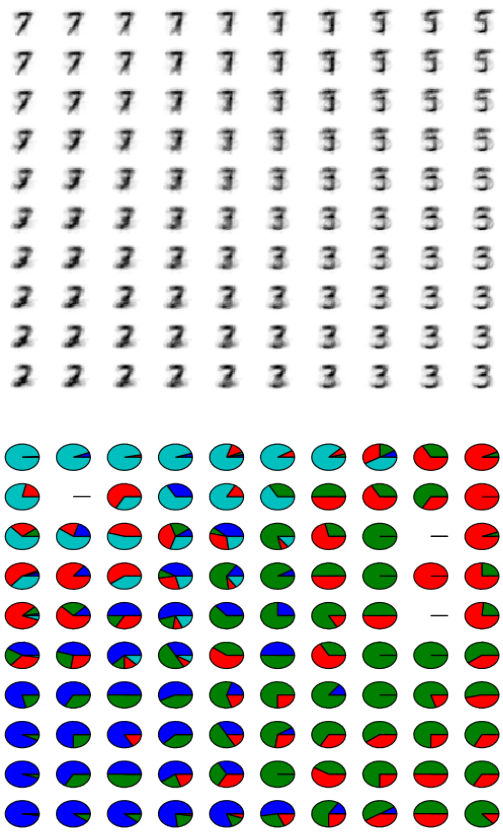


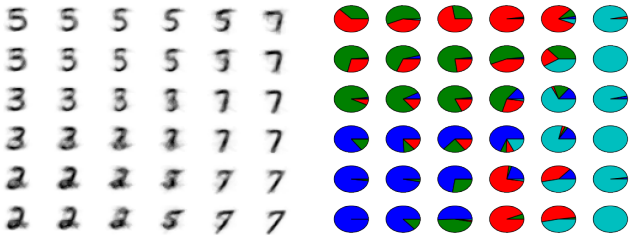
Figure 6. A 10×10 map and its prediction

B. *Width of neighbourhood function*

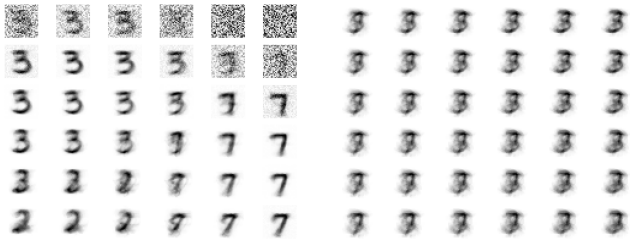
Now, the aim is to improve the weights accuracy by finding a better σ. Note that for the rest of the analysis we will work on a 6×6 map as it is big enough for predicting four digits and their variants.

Setting σ to 5, we are not surprised to see a map blurrier than before (figure 7d). The neurons seem to be almost the same and it incites us rather to decrease σ. As such, taking a unit σ and observing also the counters (figure 7a), we note that we get a better accuracy than for previous, bigger σ as well as higher precision. However, there is a price to pay which is the convergence speed. Figure 7c shows the map at the 600^{th} training iteration, where some neurons haven't been trained at all.

In conclusion, a smaller σ allows a neuron to be more accurate since it is not influenced by farther situated neighbours, only by the local ones which act like cousins. The only advantage of a bigger σ is a faster convergence to a blurry neuron.



(a) $\sigma = 1$ (b) Counters for $\sigma = 1$



(c) Situation at $t = 600$ (d) $\sigma = 5$

Figure 7. Analysing different values of the neighbourhood function

VI. A Dynamic approach

We have observed and learned useful results which enable us to understand better Kohonen maps and which give us ideas to improve our previous implementation.

Knowing that a small σ provides better accuracy will allows us to make a good digit prediction at the cost of longer convergence times. We can overcome this by using a two-phase training, the first one using a bigger σ to quickly converge to a blurry but exploitable map, and the second one with a smaller σ to refine the weights.

An even better solution is to decrease σ continuously as we progress in the training. A good heuristic for that is the linear function

$$\sigma(t) = \frac{N}{2(t+1)},$$

where t is the iteration number and N the number of samples that will be presented. This begins with a very high value but reaches quickly to sensible values, finalising with $\sigma = 0.5$ at the end of the training.

Figure 8 shows the results of this approach. We observe clear and specialised neurons, with good variation between the forms they capture. These are confirmed by the big counters of prediction.

References

[1] T. Kohonen and P. Somervuo, "Self-organizing maps of symbol strings," *Neurocomputing*, vol. 21, no. 1, pp. 19–30, 1998.

[2] B. H. Ott, "A convergence criterion for self-organizing maps," 2012.

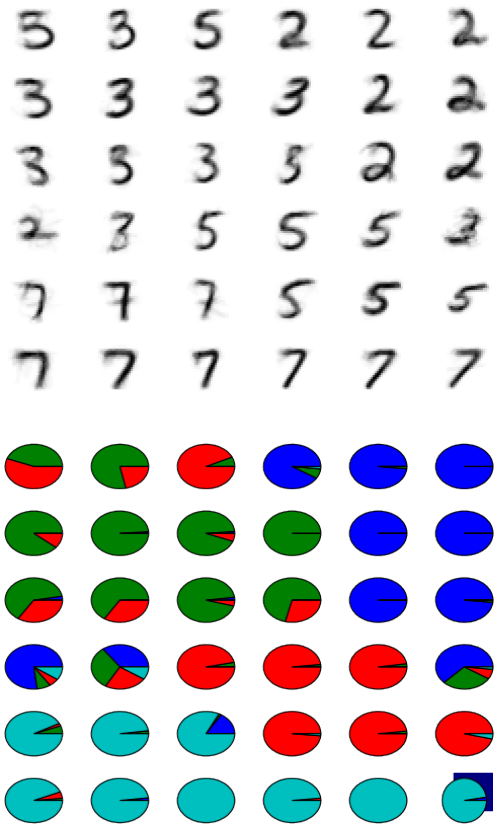


Figure 8. A 6×6 map with a dynamic σ