Student names: Manu Srinath Halvagal, Ze ya Yin

1 Kohonen maps and Hebbian Learning

How many layers are there in a Kohonen Map and what do they do?
There is only one computational layer in a Kohonen map. The weights coming in from the input to this layer eventually converge to a topologically ordered representation of the input space. Thus a Kohonen map creates a similarity graph of the input space, mapping a potentially high-dimensional input space to a low-dimensional discrete output space.

The self-organising process have 4 components; Initialization, Competition, Cooperation and Adaptation. How are all these components implemented in the Kohonen Map?

1. Initialization : Each synaptic weight is initialized to a small random value. In this case, no prior order is encoded into the map.
2. Competition: For each input pattern, one neuron (which has the strongest response to this particular input pattern) is picked as the winning neuron. This winning neuron then gets updated strongly to further match the input pattern, while other neurons get weaker or no updates at all.
3. Cooperation: A neighbourhood of the winning neuron is defined based on the topological arrangement of the neurons, and all the neurons in this neighbourhood are also updated similarly to the winning neuron but to a lesser extent.
4. Adaptation: The synaptic weights of the winning neurons and it's close neighbourhood are adjusted with a Hebbian-type learning rule so that the response of the winning neuron to a similar input pattern is enhanced even further.

Describe how a Kohonen Map performs dimensionality reduction
The SOM grid is non-linear in the full dimensional space where the "grid" is warped to more-closely fit the input data during training. This is a form of dimensionality reduction because now relational information can be measured in the topological space of the grid in the lower dimensions instead of the full m-dimensions (where m is the dimension of variables). To be more precise, the active region in a Kohonen network is only in two dimensions, so it just gives 2-dimensional information of measured distance. Therefore, an SOM is a mapping of the m -dimensions onto the 2-D SOM grid.

Kohonen maps are also described as Topographic maps, what is meant by Topographic maps and give an example of their possible presence in the human brain.
A topographic map means that the relative position between the neurons within the lattice (of inter-neuron connections) will, in some way, correspond to the relative positions between the samples from the input space. Topologically ordered maps are thought to exist in sensory maps in the cerebral cortex. For example, visual input is mapped in an ordered way that codes for the location on the retinal plane and other features of visual cues such as orientation in a continuous fashion. This kind of ordering also exists for tactile and acoustic sensory inputs as well.

Which one of the Hebbian type learning rule corresponds to the weight update in the Kohonen maps (Reminder of the examples of Hebbian type rules: Have a look at the Lecture 2, slide number 9). Show how can you obtain the learning rule of the Kohonen map from the corresponding Hebbian rule.
The generalized Hebbian learning rule with a pre-synaptic threshold corresponds most closely with the weight update in the Kohonen maps. We can develop the equivalence as follows. Consider $y(n) = \nu_i$ (output or post-synaptic firing rate) and $x(n) = \nu_j$ (input or pre-synaptic firing rate). Now the

Hebbian rule with a pre-synaptic threshold gives:

$$\frac{d}{dt}\mathbf{w}_k(t) \propto c_{11}^{corr} y(n)(\mathbf{x}(n) - \vartheta)$$

Now consider the threshold ϑ to be equal to the weight that is being updated. Furthermore, by setting the term $c_{11}^{corr} y(n)$ to the neighbourhood function around the winning neuron (to take into account both competition and cooperation), we arrive at the Kohonen weight update. It is more straightforward to arrive at the Kohonen update if we start with Oja's Hebbian learning rule, which provides theoretical justification for this choice of ϑ (by constraining the weight vectors to norm 1).

2 Kohonen Maps on Hand-written Digits

In the following discussions, we use the term epoch to mean that every sample has been presented once to the network in some arbitrary order.

2.1 Learning Rate and Convergence

To study the convergence of the algorithm, we needed to decide a performance measure. A straightforward way to do this is to look at the average (over every sample in the epoch) squared error between the input samples x_i and its closest prototype (winning neuron m).

$$d(\mathbf{x}_i, m) = ||\mathbf{x}_i - \mathbf{w}_m(t)|| \tag{1}$$

While this would be sufficient, a more appropriate measure would be a weighted mean of the errors between a sample and every prototype k, weighted by the neighbourhood Gaussian function $N(m, k)$ around the winning neuron m (Equation 2). This can be thought of as a quantization error, and is in fact, simply the mean magnitude of the synaptic weight updates (Equation 3) of all neurons upto a factor of the learning rate η.

$$d(\mathbf{x}_i, m) = \sum_k N(m, k)||\mathbf{x}_i - \mathbf{w}_k(t)|| \tag{2}$$

$$\Delta\mathbf{w}_k(t) = \eta N(m, k)(\mathbf{x}_i - \mathbf{w}_k(t)) \tag{3}$$

Therefore once the epochal average of $d(\mathbf{x}_i, m)$ remains stable, we can be sure that the network weights have also settled into a metastable state.

For training Kohonen maps, one usually starts with a high learning rate (~ 0.1) and gradually reduces it. However, here we use a constant learning rate, which could be somewhat trickier to pick. We tested out logarithmically spaced values [1.0,0.1,0.01,0.001,0.0001] for training the map for around 25 epochs. In each case, we use a Kohonen network of 6x6 neurons and a (constant) standard deviation $\sigma = 3$ (as specified). The learning curves for each of these learning rates is presented in Figure 1. We see that the root mean square error converges faster to lower values within the first few epochs with a learning rate of 0.001. For larger learning rates, the mean error shows higher fluctuations and oscillates around a steady value due to the the large step size, which is not easy to notice in Figure 1, but could be observed in their individual test graphs. For this reason, we chose to go ahead with a learning rate of 0.001 (convergence would be extremely slow if we used 0.0001).

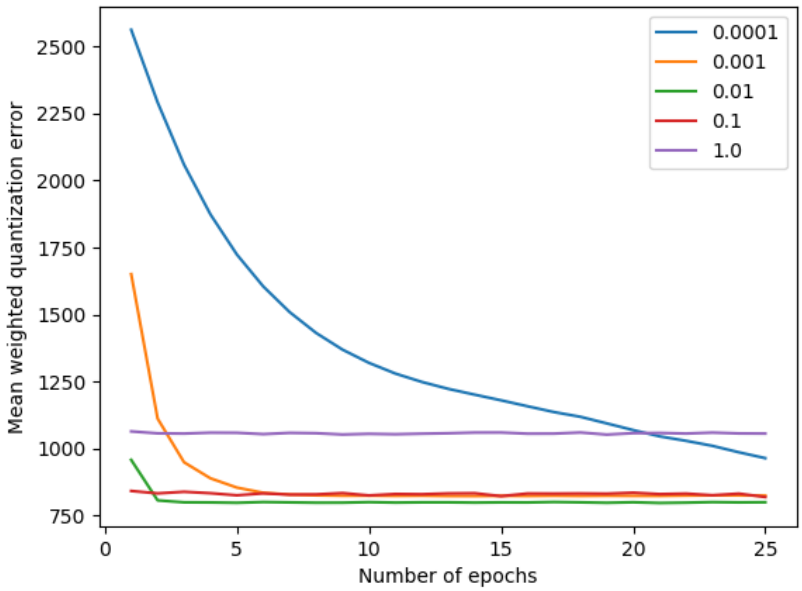


Figure 1: Comparison between learning rates

2.2 Learned Prototypes

For the specified Kohonen map of 6x6 neurons and (constant) standard deviation $\sigma = 3$, the prototypes we obtain after 100 epochs are shown in Figure 2. The four digits chosen in our case are [1,4,8,9]. Some prototypes are close to one particular digit. However, others look like a mixture of two digits (particularly for the digits 4 and 9 because they are extremely similar). A few neurons will tend to have the strongest responses to one particular digit in its pure form (maximizing the spread of the neural lattice over the input space) and will be the best representatives of that digit. The neurons in between these maximally responsive neurons will be topologically situated in between the digits, and hence it is not surprising that they look like mixtures. Another observation we can make from the learning curve was that the network had already converged after about 20 epochs. So running the algorithm for about 25 epochs is more than sufficient.

2.3 Labeling the Prototypes

To assign the "best" label to each prototype, we first clustered together all the images of each digit, i.e, we represented each digit with the centroid of all the images of that digit. Then for each prototype, we found the closest of the four centroids and assigned the corresponding labels to the prototypes. As shown in Figure 2, some neurons do not completely correspond to the digit they are labeled with. The reason for this is the same as explained above (existence of interpolating neurons).

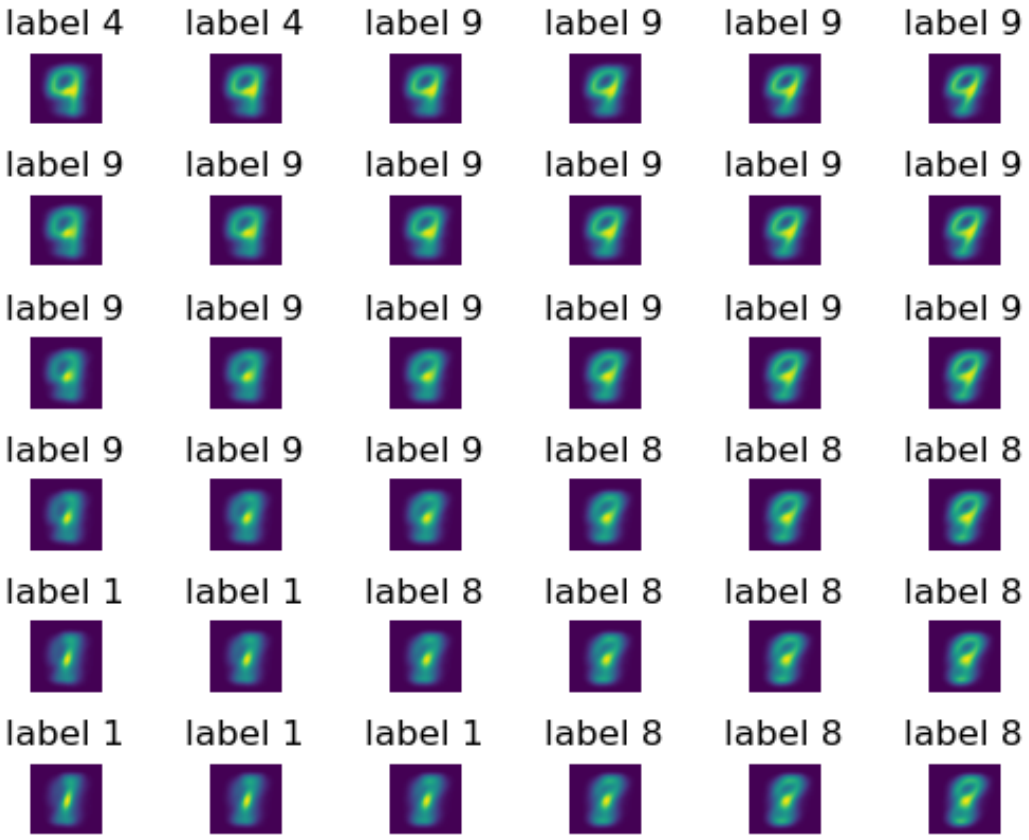


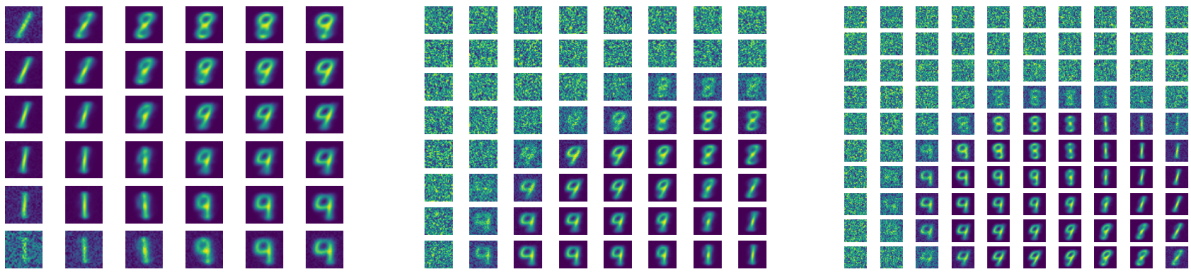
Figure 2: Prototypes learned after 100 epochs

2.4 Varying the Network Size and Neighbourhood Width

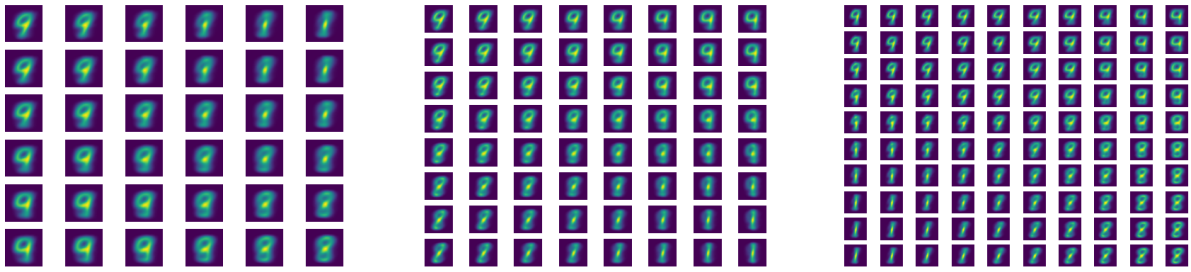
In this section, the influences of the map size and the width of the neighborhood function on the learned prototypes (Figure 3) are investigated. The map size is chosen between 6, 8 and 10 and the width from 1,3 and 7. Having a large neighbourhood width makes almost all the neurons move together especially when the map size is small (Figure 3g). As a result, all prototypes tend to look very similar and fuzzy. Therefore we also see in Figure 4a that maps with wider widths tend to have large quantization errors due to this lack of sufficiently strong competition. If we reduce the neighbourhood width, then the Kohenen map gradually moves closer to competitive learning. However, if we reduce the width too much, only the winning neuron is activated and gets its weight optimized at every iteration. This can lead to very slow learning especially when the map size is quite large as shown in Figures 3b and 3c.

In addition, larger map sizes can lead to more specialized prototypes. When considering different sizes, it is noticed that there are relatively few prototypes to represent mixtures of different digits. As the map size increases, more prototypes are available to form representations of the input. Therefore, large map sizes can also reduce the quantization error as shown in 4b as a large grid-map has more representative power. One disadvantage of a large map size is that it can take more epochs to converge especially with small neighbourhood widths.

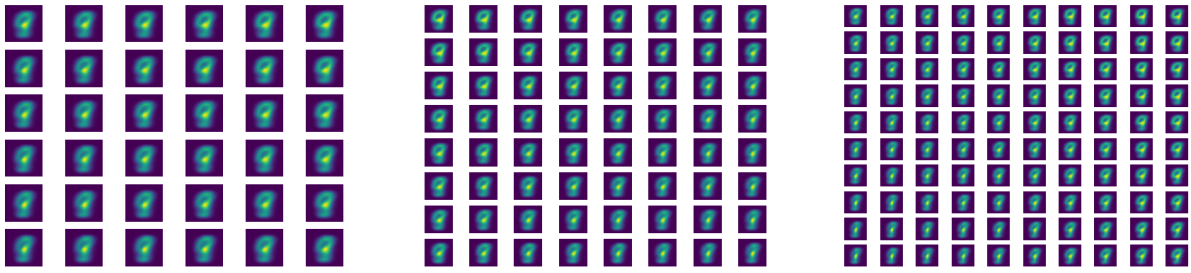
To answer the question, the optimal width definitely depends on the map size. Larger maps require larger widths for effective training.



(a) 6×6 *map with* $\sigma = 1$ *(b)* 8×8 *map with* $\sigma = 1$ *(c)* 10×10 *map with* $\sigma = 1$

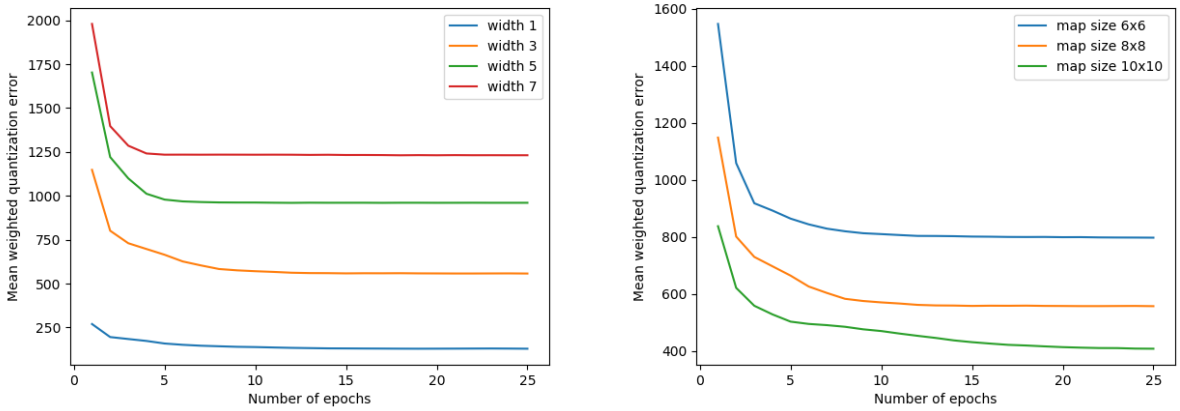


(d) 6×6 *map with* $\sigma = 3$ *(e)* 8×8 *map with* $\sigma = 3$ *(f)* 10×10 *map with* $\sigma = 3$



(g) 6×6 *map with* $\sigma = 7$ *(h)* 8×8 *map with* $\sigma = 7$ *(i)* 10×10 *map with* $\sigma = 7$

Figure 3: Prototypes learned with varying map sizes and neighbourhood widths



(a) Different neighbourhood widths for an 8×8 *map* *(b) Different map sizes with a fixed neighbourhood width of* 3

Figure 4: Learning curves for varying map sizes and neighbourhood widths

2.5 Decaying the neighbourhood width over time

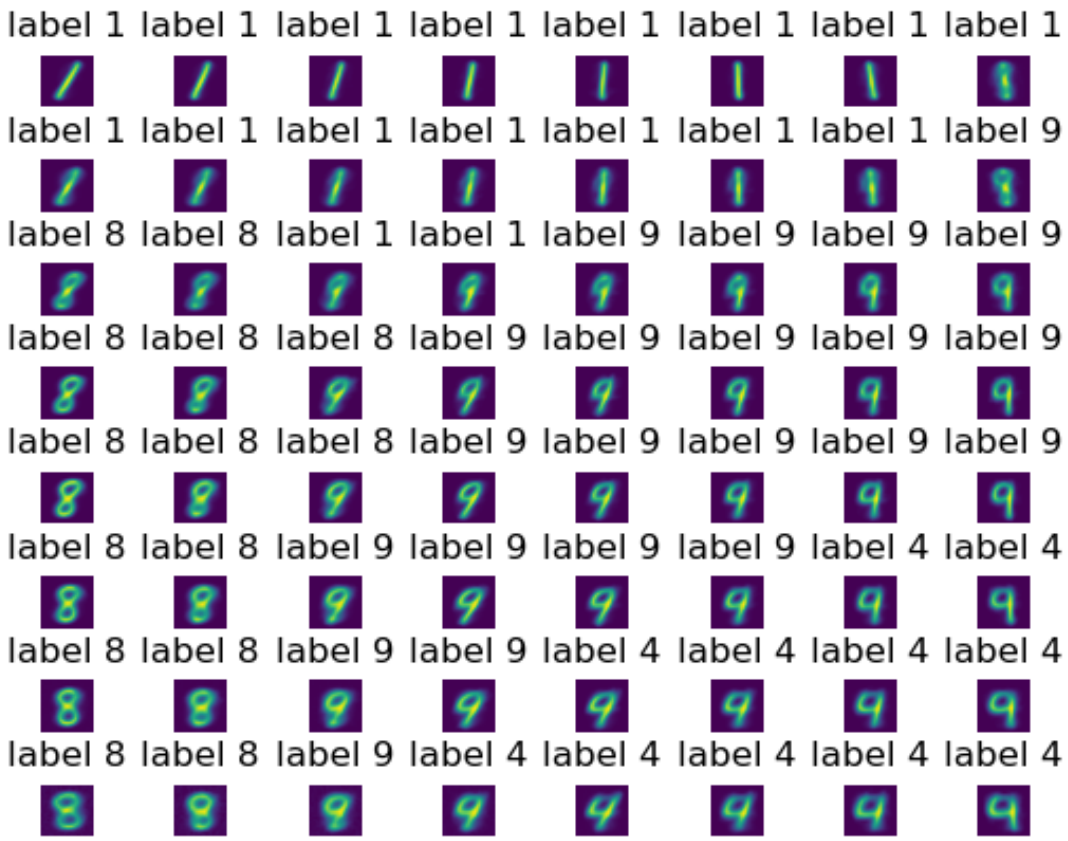


Figure 5: Prototypes learned after 25 epochs

This result is better than that obtained using a constant neighbourhood width. In general, a large width applied at the beginning activates most neurons and speeds up the initial self-organizing phase. However, this can also cause all neurons to end up with very similar weights. So decreasing the width over time is preferred to speed up training but also make more specialized prototypes in the end which improves the quantization error significantly.

In our code, we implement this with an exponential decay of the width applied at the end of every epoch.

$$\sigma = \sigma_0 \exp \frac{n}{\tau}$$

where n is the number of epochs, σ_0 is chosen as half the map width and τ is a time constant that we tuned to get optimal results.