Unsupervised Learning and Neural Networks
Mini-Project 1 Report

Yao-Chieh (Jeff) Hu, Seth Vanderwilt

04 December 2016

Mini-project overview

Our task was to implement the Kohonen algorithm (or self-organizing map) and apply it to a dataset of hand-written digits. Please see `kohonen.py` for our implementation.

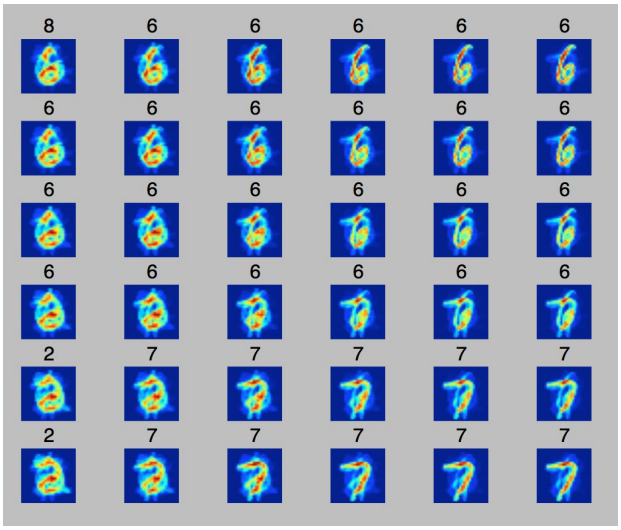
Part 1: Learning rate and convergence

We chose the learning rate (eta) to be a constant 0.5. With a higher learning rate (e.g. 1), we would be more sensitive to the latest data (with a poorer "memory" of the past examples) and if the learning rate was very low, we would not learn anything. So we arbitrarily chose a learning rate of 0.5.

Since there is no accepted definition for when a Kohonen map has "converged" we decided that simply limiting the number of times it had seen a datapoint was close enough. We experimented with stopping the algorithm when the assignments no longer changed but found that many datapoints would keep changing their "winner" unit, meaning this definition of convergence would likely not terminate in practice. So we arbitrarily limit the number of iterations to 10,000, which means we expect to see each datapoint approximately 5 times (there are 2000 training examples containing our target digits).

Part 2: Visualization and description of the learnt prototypes

(Note that the number above each prototype is the digit we assigned it in part 3)



Part 3: Assigning a digit to each prototype

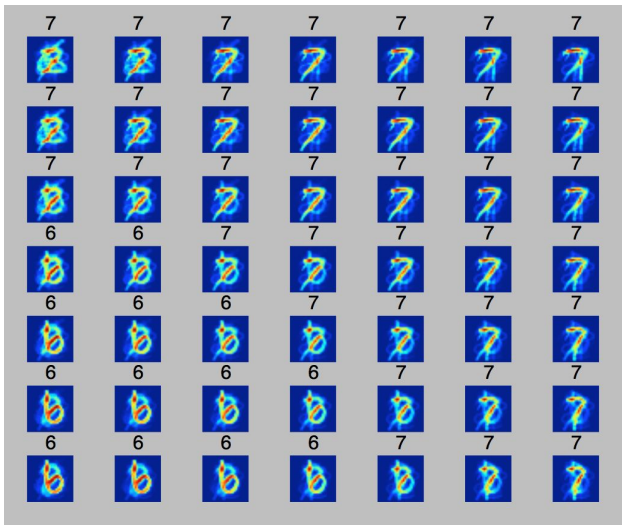
In a Kohonen map, each neuron/unit does not necessarily represent a certain digit. However, we decided that since the "winner" in the Kohonen algorithm is the unit with minimal distance to the datapoint, we could more or less go the opposite direction. Thus we assign each unit the digit of the datapoint that was closest to it according to Euclidean distance.

If you view the screenshot above, you can see that each unit has been assigned a digit. Some of them clearly resemble their assigned digit, but others look more like some combination of multiple digits, which is to be expected.

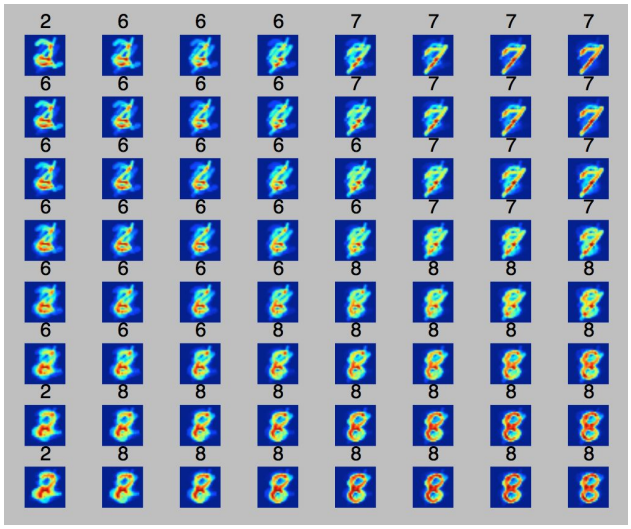
Part 4: Exploring network sizes and widths of neighborhood function

First we tried increasing our network size to 49, 64 and 100 units.

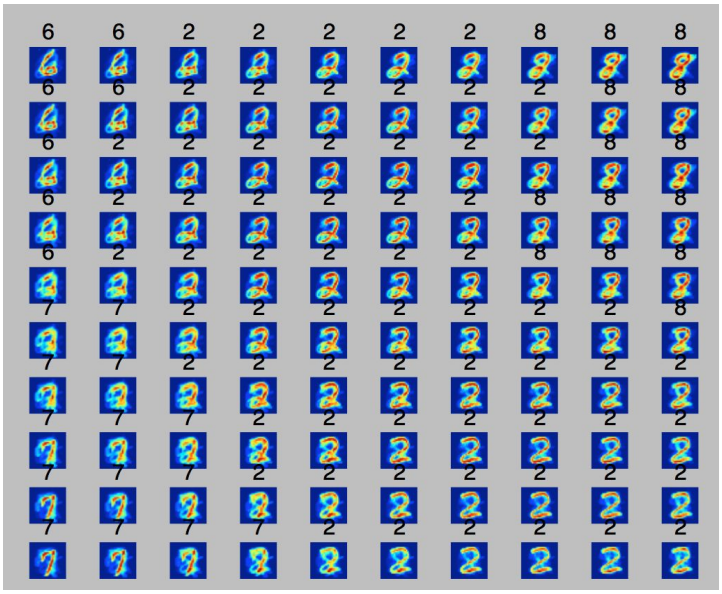
With 49 units:



With 64 units:

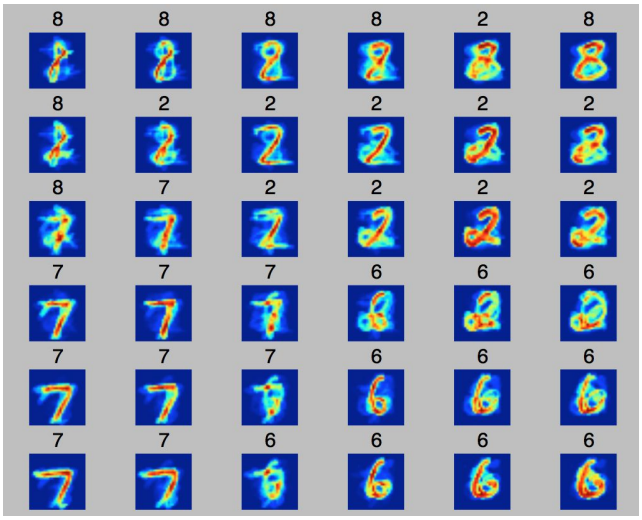


With 100 units:

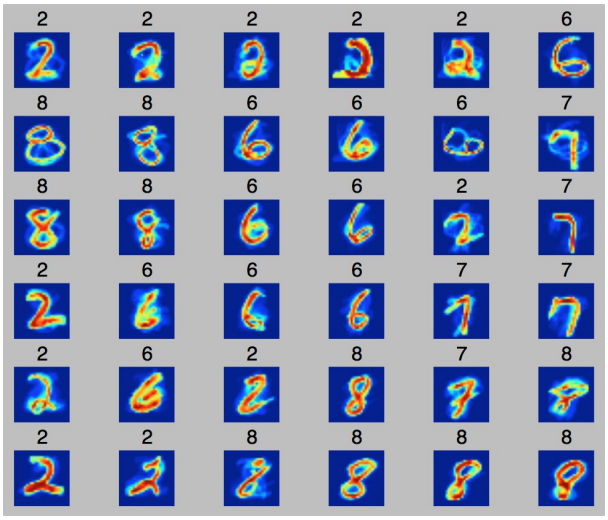


We explored four widths of the neighborhood function ($\sigma = 0.5$, 1, 3, and 5).

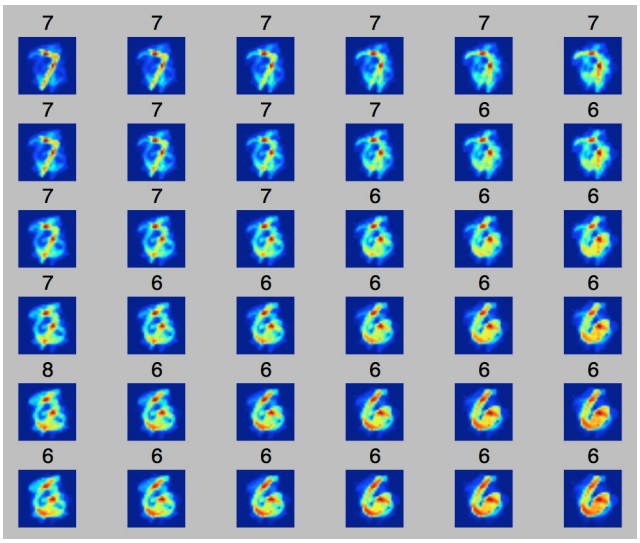
$\sigma = 0.5$: "winner-take all" behavior observed, width is too small



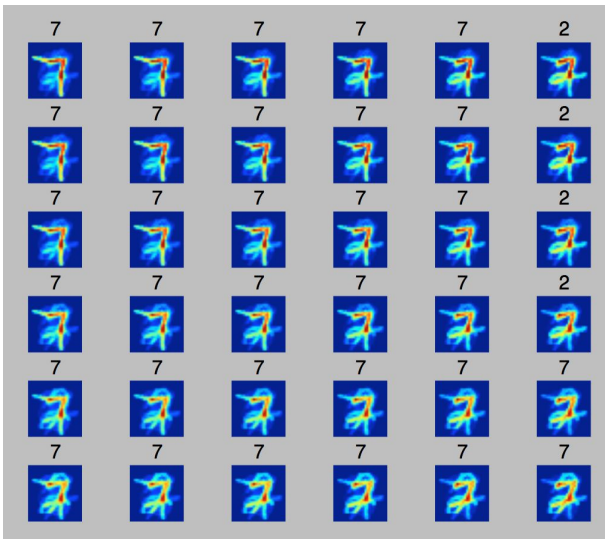
$\sigma = 1$: looks better, some overlap between neighbors



$\sigma = 3$: many prototypes end up with very similar values



$\sigma = 5$: all prototypes look almost identical

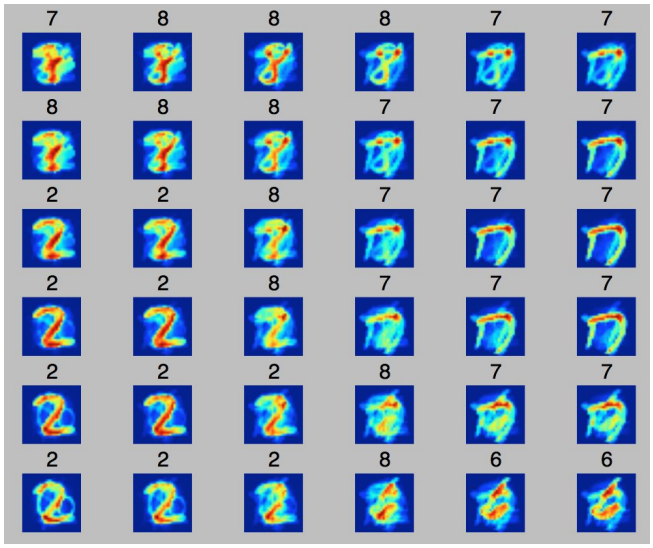


The width of the neighborhood function determines how updates spread beyond the winning unit. A small width means that updates are more localized whereas a large width influences other units much further away from the winner. This means that if the width is too big we end up updating all of the units no matter who won and thus end up losing some of the locality effect in our map's layout. Therefore we must choose a width that is appropriate for the size of the Kohonen map.

We tried one more time with $\sigma = 2$ and size=6x6, to see if that looked better than the other width choices (see below). It looks like there is more overlaps between neighboring neurons than with the smaller sigma but not so much as the default $\sigma = 3$. We don't have a good way to evaluate the

“goodness” of our map, but clearly selecting the optimal width must include information about the size and configuration of the Kohonen map.

$\sigma = 2$ and size=6x6



Part 5: Results of varying width of neighborhood function over time

We decrease the neighborhood width by 0.01% (compounding) each iteration, so that after 10,000 iterations the width is only roughly 37% of its original value. This seems to give some interesting results (see below). The result looks somewhere in between our experiments with width=3.0 and width=1.0, but with some clear prototypes and some more clustering effects among neighboring neurons.

Figure: width starts at 3.0, decreases to 1.1 by the end of the iterations.