Mini project 1: Clustering with Kohonen Maps Assignment

Vidit Vidit
Dunai Fuentes Hitos
CS-434, EPFL, Switzerland

***Abstract*—Our goal is to implement the Kohonen algorithm and apply it to the well-known data set of hand-written digits, MNIST, reduced to 4 digits. In the process we define a convergence criteria for a multi-dimensional network, find the optimal learning rate for our problem, compute the optimal width of the neighborhood function for different network sizes, and experiment with dynamical update of the same. When the units (or centers, or prototypes) are assigned a class by using K-neighbors (K=3) from the data set, the resulting accuracy on a test set that the network hasn't seen before is of 95%. Outstanding results for an unsupervised learning technique.**

I. Introduction

In this multi-class clustering and classification project, to ease the task, we have reduced the complexity of the popular dataset MNIST to just 4 digits. These had been decided pseudo-randomly from a combination of both author's names, more precisely the string "Dunai Vidit", that resulted in the digits:$1, 4, 5, 6$.

We further divide the subset of MNIST in two block: a train set, that will be used to give shape to the network and ultimately assign a label to the units, and a test set, that is labeled by the Kohonen map to check it's accuracy against the ground truth.

The code and some extra notes can be found in the complementary ipython notebook.

II. Learning Rate & Convergence

In any stochastic or online optimization technique the idea that lies behind it is that the generalization of the model will increase with each new data point, getting us a step closer to the optimum configuration. First it is worth noticing that even though a huge learning rate would allow us to quickly move in the direction that has being pointed out as optimal by the sample (or samples in batch method's), it can and will also turn destructive once our generalization is good enough, as "learning" what seems to be the new trend on the data will make us"forget" what was already learned.

So how fast should we learn and when to stop learning? The most common answer to this question in state-of-the-art optimization techniques is to learn quick at first, and progressively slow down the learning rate until you touch the convergence criteria or a limit to the number of iterations, avoiding the problem of destructing what has been learned during the latest stages of the process, while also granting a good speed towards the optimum (or local optimums in non-convex problems such as our own) in the early stages.

However, we are not gonna implement these adaptive techniques to the learning rate. Instead, a few different constant values are explored, and how they affect the evolution of the weights. This is what is presented in Figure 1.

How to define the convergence is a little bit more tricky. Wishfully we would like the algorithm to stop once there is nothing more to be learned, this is, when the model explains the data to the best of its capacity. However, because not all the data is contemplated when making an update, but only one point at a time, we can never reach stability. Every new data point will be tried to be explain best by the closest center and his neighbors, slightly destroying what they knew from the previous data point. The centers will always be perturbed, but within these oscillations there is an overall trend towards an optimal configuration. When this trend disappears, we are left with the back-and-forth movement of the centers, we consider that we have reached convergence.

In terms of implementation, because the expected center update is higher when the trend is present, we decided to count the number of times that the averaged (among centers) squared center update, rectified by the learning rate, reaches a lower-threshold value that has been empirically determined to be mostly present during the steady regime. We claim convergence once our count reaches yet another threshold that acts a a safety buffer. It is a simplified criteria that gives as good results as looking for a moving average close to zero.

An important note it's that due to the non-convexity of our problem, a smaller learning rate is not strictly superior for an infinite amount of time, as it can get more easily stuck in a shallower local minimum. On the other hand an unreasonably big learning rate can be jumping around the solution without never getting to it. Once more, the online nature of our updates makes this matter even more critical. A good compromise for the learning rate must be reached. In Figure 2 we present the first results for the learning rate $\eta = 0.03$. Comments on other results for different learning rates can be found along with the source code notebook.

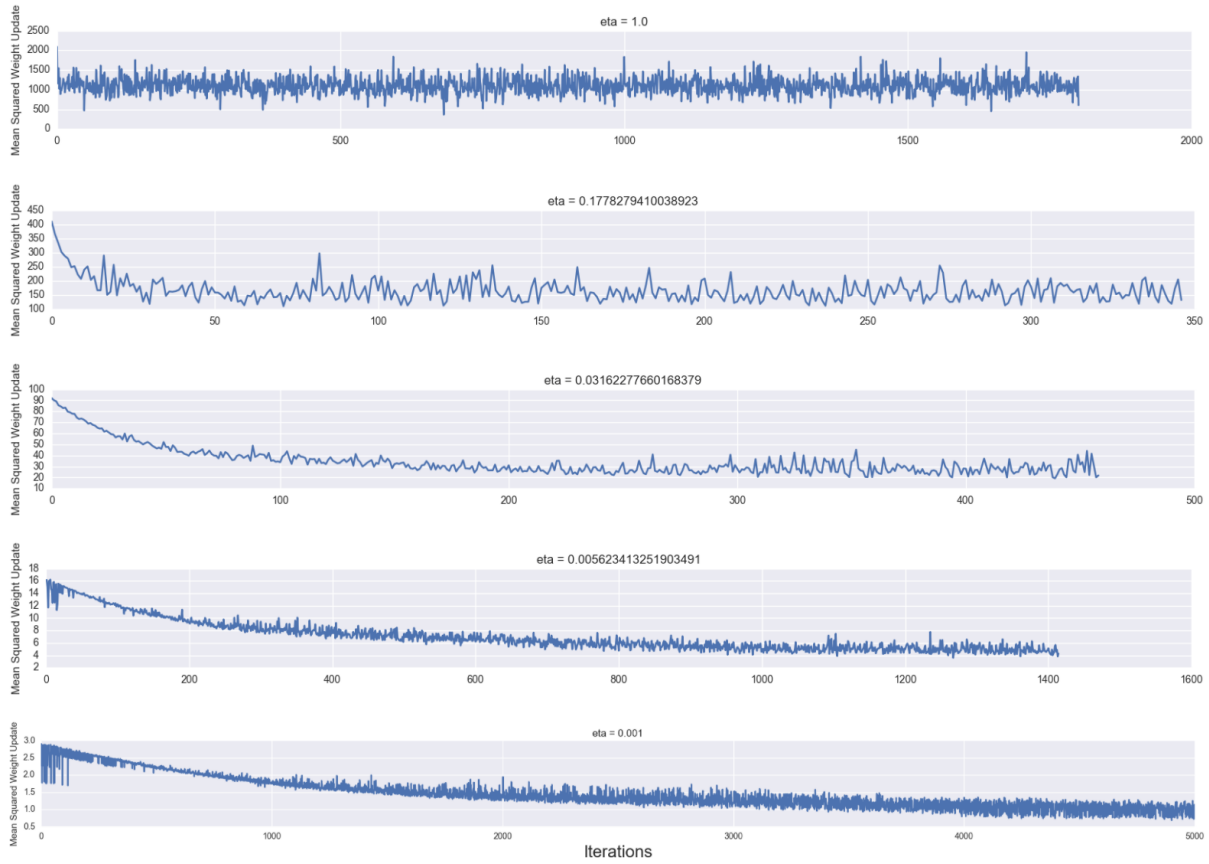

Figure 1. The image presents the average squared center's update over iterations (or "seen samples"). We can see that the amplitude of the updates is smaller when the learning rate is smaller, and that it also implies longer transients.

Figure 2. The image shows the 36 prototypes trained until convergence for $\eta = 0.03$ and a $\sigma = 3$. We can see how the different sections of the resemble some digits more strongly than others, like the 6 on the top right. However, everything is still very mixed and clusters not well defined. This is because the neighborhood function's width σ is too high and it is forcing the prototypes to pull themselves towards each other and (on average) towards the center of our hyper-dimensional space.

III. Assigning a digit to a prototype

Each prototype is a point in the 28×28 image space. When we visualize it, it may look like on of our digits if it is very close to it, or like a rater blurred mixture of digits if it is somewhere in between them in the image space. This is what we have just seen in Figure 2.

When it comes to deciding which digit should we assign to it, what to us can mean "which number does it look like", in a mathematical expression it can be rephrased as "which numbers are close to it".

There are many different metrics that can be used to compute this distance. In our case we will stick to the good-old Euclidean distance, but it is worth noticing that for images and in general for high dimensional spaces, due to the curse of dimensionality, the Euclidean distance can be a questionable choice.

Finally we turn to a K-neighbors voting scheme to decide the digit that should be assigned to the prototype. All throughout the project with have worked with a modest $K = 3$ as we shouldn't be in great danger of overfitting. Results for the previous configuration are shown in Figure 3.

```
[[ 4.  4.  4.  6.  6.  6.]
 [ 4.  4.  4.  6.  6.  6.]
 [ 4.  4.  4.  4.  1.  1.]
 [ 4.  4.  4.  1.  1.  1.]
 [ 4.  4.  4.  1.  1.  1.]
 [ 4.  4.  4.  1.  1.  1.]]
```

Figure 3. Here we present the K-neighbors classification of the prototypes of Figure 2. We notice how we are not getting any fives despite their silhouette being recognizable in the image above. Our interpretation is that the cluster of fives is less "central" than the others (further away from the center of mass).

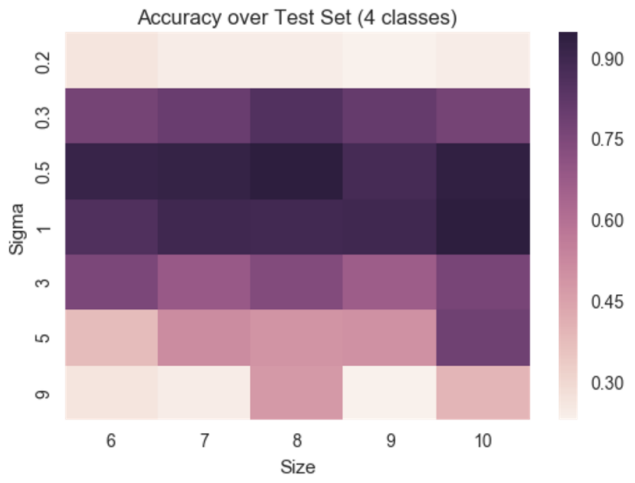

Figure 4. Heatmap of the accuracy over test set for different values of σ and $Size_k$. The network is composed of a total of $(Size_k)^2$ units. For $\sigma = 0.5$ & $Size_k = 8$ we get 95% accuracy

IV. Network Size & Neighborhood

We have already mentioned how $\sigma = 3$ was not appropriate for our problem as it amounts to a very wide neighborhood that it's preventing the formation of isolated clusters, pulling everything towards the center of mass. Now we would like to explore how different values of σ in our Gaussian function affect our results as well as different network sizes simultaneously.

To test their optimality explaining the underlying structure of the data, we compute the *accuracy* of the networks predicting the labels for unseen data points in the test set. This is done with K-neighbors again ($K = 3$), starting from the assigned digits to the prototypes. Results can be seen in a heatmap in Figure 4. It is worth reminding that in a 4-class classification problem, a random prediction will achieve an expected 0.25 accuracy.

We note that the optimal width indeed depends on the size of the network, with a higher value of σ needed for bigger networks. A big network is more prone to overfitting as each prototype has to account for a smaller region of the space (it will be easier to find some of them displaced to regions of the space where only outliers appear), and this is why the prototypes need more neighbors to regularize the overall behavior.

V. Neighborhood Function's width decay

We have argued before how the parameter σ controls the width of the neighborhood function and the number of prototypes that get significantly affected with each new sample seen. The development of the network will be strongly tied to this parameter. In our first experiment a big value of σ forced everything towards the center of mass, and in the heatmap of Figure 4 we observed how a very low value of sigma also fails to deliver decent results as we lose the concept of a network for that of many prototypes all behaving independently, overfitting the data.

A way to deal with this tradeoff is to establish a decay on the sigma value, starting with high values of sigma that control the correct development of the network (first towards the center of mass, then periodically spreading into more independent clusters) and ending with low values of sigma, affecting only individual prototypes so that there is not much perturbation of the overall network which fortifies the concept of convergence.

We have opted for a linear decay with each iteration, the evolution of the accuracy is shown in Figure 5, the final prototypes in Figure 6, and their respective assigned labels in Figure 7. We have also used a smaller value of the learning rate to that suggested by the heatmap to avoid excessive overfitting during the very low sigma steps (by forcing them to be small) and to better control the development during the high sigma steps, again by limiting the capacity of the whole network to jump straightaway to every new data point that it sees.

VI. Final Comments

The convergence criteria that we defined wasn't friendly towards changes in the value of σ. Smaller values of σ will generate a smaller mean displacement of the network and thus would require a lower low-threshold than the one we took experimentally for the case of $\sigma = 3$. Rectifying the value of the threshold for σ it is not trivial as it was for η. For instance, for $\sigma = 0.5$ a threshold 10 times smaller than the previous one works best. In any case, we have checked that 5000 iterations are always enough for convergence with our (σ, η) pairs, and we have used this limit for the

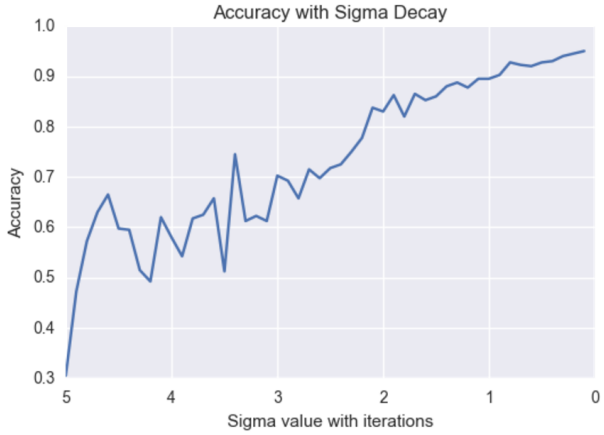

Figure 5. Here we show the evolution of the performance of our network classifying unseen samples from the test data in term of accuracy. We can see a rapid improve at first, when the network easily learns the basic structure, then there are some oscillations until σ gets small enough to allow specialized clusters, and finally we see progressive improvement as these clusters become independent of each other.

Figure 6. Prototypes after sigma-decay training. We notices each number seems to be clustered even in the two dimensional display, which doesn't necessary mean that stray fives on the right have lost their place. They would probably be easily separated from the rest by a hyperplane and put together with the other fives.

```
[[ 1.  1.  1.  1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.]
 [ 5.  1.  6.  6.  6.  5.  5.  4.]
 [ 5.  5.  6.  6.  6.  4.  4.  4.]
 [ 5.  5.  6.  6.  6.  4.  4.  4.]
 [ 5.  5.  6.  6.  6.  4.  4.  4.]
 [ 5.  5.  6.  6.  6.  4.  4.  4.]
 [ 5.  5.  6.  6.  6.  6.  4.  4.]]
```

Figure 7. Labels assigned to prototypes from Figure 6.

final experiments.

Our final results are of 95% accuracy, both in the case with decay and for static values $\sigma = 0.5, Size_k = 8$. These are outstanding performances for an unsupervised learning algorithm.

ACKNOWLEDGEMENTS

We want to thank our professor, Marc-Oliver Gewaltig, and the team of assistants, for putting together the material and functions that allowed us to make this project.