

Demetz Clément.
902 964 5658

Professor Yu Hen Hu

ECE 539

Introduction to Artificial Neural Network and Fuzzy Systems

Individual Class Project

Lip-recognition Software using a Kohonen
Algorithm for Image Compression

Table of contents

Introduction	3
Work performed	4
Data Creation and Setup	4
Pre-processing: Kohonen self organization map	6
1. Initialization	6
2. A way to avoid the crossing problems	10
3. Application to a real-life problem: Selection of parameters and results	11
Multi-Layer Perceptron: Final results	13
1. Generation of Data sets	13
2. Perceptron design and final results	13
Discussion and Conclusion	6
Matlab files and used files	17
References	18

Introduction:

Processing recognition softwares always needs some incredible amount of computation power, whereas embedded system, such as cellular phone or PDA, generally has very little available power. Several firms are still trying to find the best way to implement voice recognition capabilities to allow people to “talk” with their phone to command it or to make their PDA type a letter. The new generations of these systems sometimes have a digital camera or a web cam.

In addition to taking pictures, these cameras can be used to aid in sound recognition. A lot of studies have showed that in the domain of recognizing human inputs, a combined approach is often the best way to obtain accurate classification rates. I was especially inspired by the first two papers ([1], [2]), and by the book by *Collica, R.S., Card, J.P., and Martin, W.* [8]. My idea was to focus on three sounds and to implement a lip-recognition algorithm that can be seen as a benchmark for recognition software overall. If we can recognize a few particular sounds very quickly using limited computational resources, then it may be possible to combine that sort of algorithm with “pure” voice recognition to make it faster and more efficient.

I will first apply a Kohonen Self Organization Map to a large number of pictures, corresponding to different lip configuration (one shape for every sound). The outputs of the Kohonen Map will then become the input of a Multi Layer Perceptron

In this project, the computation time will always be the key limit. To approximate a real-life problem, I have decided to work on real pictures of low quality: the type of picture that can be taken by an actual cellular phone camera. The results are shown for each chronological step of the process.

Work performed

Data creation and setup

The idea of trying to solve a real-life problem forced me to create my own database. In future systems, such as PDAs, firms hope that a miniaturized camera will be implemented and will take a picture every millisecond. Because I could not afford such a camera for my project, I have decided to start with a few samples of pictures and keep the “real-time” computational constraints by using the lowest computation time possible. In order to be implemented into such restrained systems; my algorithm needs to compute in less than 1/10 of a second (approximation of the speed at which lips move).

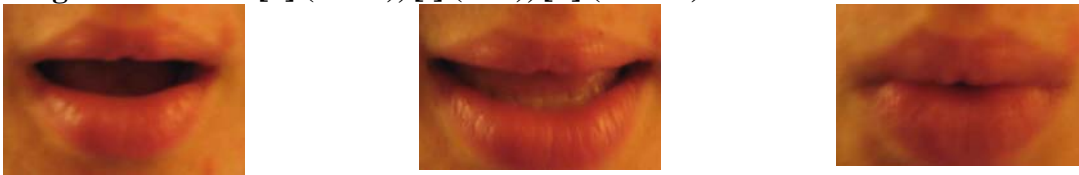
I have first worked on the “contour” of the lips, but it was too hard to differentiate the slight differences between the skin color and the lip color. The algorithm was too sensible to every parameter and to each person. But, by reading other publications on this subject, we can see that some people are studying different aspects of the problem. In our case, we will focus on the interior limit of the lips, instead of the external limits which were our first focus. According to some scientists, it makes more sense because it does not depend on individuals: whereas each person has his own physiology, the interior of the mouth must be exactly the same in order to pronounce each sound.

The pictures’ database was thus created by taking several pictures of lips while the person pronounces several sounds (such as ‘a’, ‘e’, ‘o’). We only focus on a few distinct sounds because we know that the algorithm will only be used as a support for voice recognition. Starting from a few pictures in JPEG format (.jpg), we first apply a few pre-processing operations to these files. Every JPG file is converted into a matrix format in Matlab. This format has more information than needed: we only want to work on the contour of the mouth, so we first apply a gradient-like filter to find the contours of the lips. The parameters of this filter are fixed by default but can be changed by the user, whereas the fixed value guarantees an optimal number of points (defined with respect to the ratio of the number of points to size of the image). After this filter is applied every

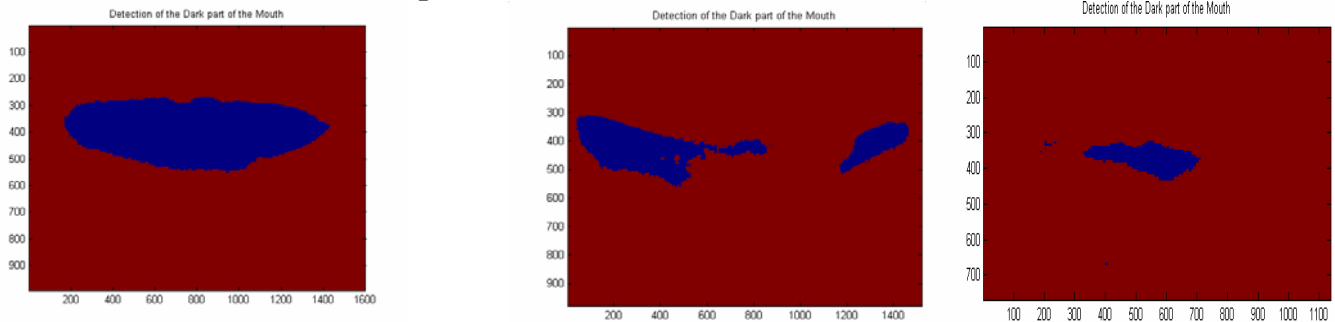
pixel is either black or white (no “grey” pixels). Here are examples of the pictures taken and the result after applying the filter algorithm.

We note that our work is performed on low-quality pictures to decrease the computation time (I have set the limit to 50KB for each picture).

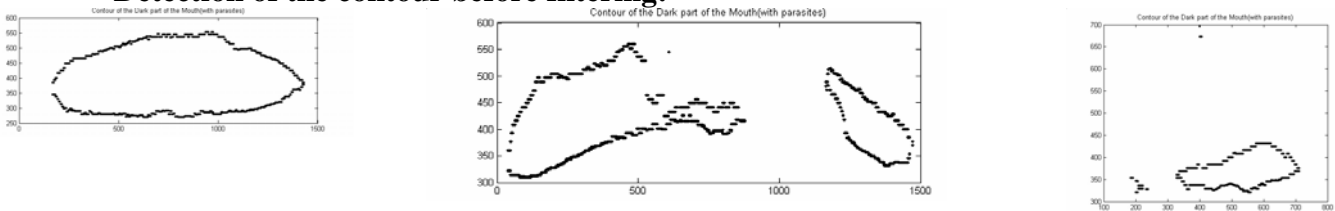
Original Pictures: [a] (“Ah”), [i] (“E”), [u] (“Ooh”):



Detection of the darker part:



Detection of the contour before filtering:



We can see that some parasites still remain; thanks to the initialization of the self-organization map around 95% of the information (explained in the next section), we will be able to work without them.

Finally, in order to fit with the self-organization map input, the matrix image form (which contains either a 255 for a white pixel or a 0 value for a black pixel) is transformed into a database, which contains every black pixel and its two coordinates. This new database, which we can describe as a black pixel coordinates database, will be used as the inputs of the Kohonen SOM. All these computations are made by *preprocessing.m*

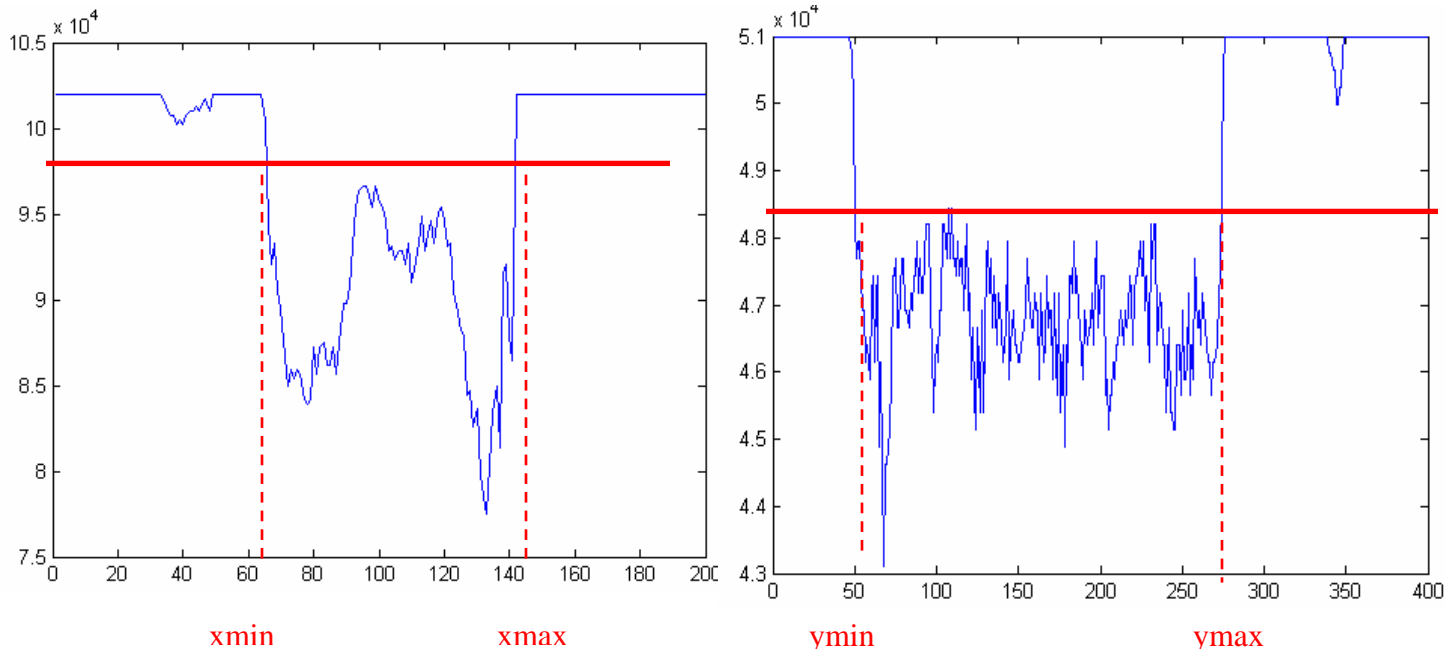
Pre-processing: Kohonen self organization map

1. Initialization.

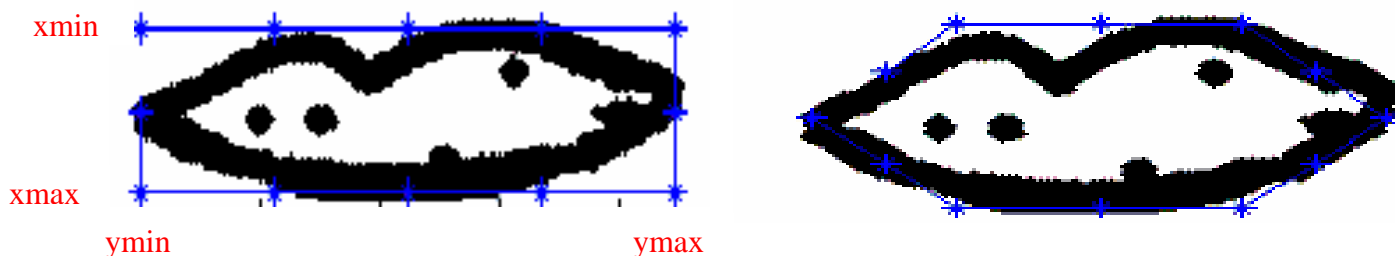
We now have a collection of points that hopefully describe the contours of the mouth in the best way possible. The idea of the Kohonen processing is to reduce the dimension of this new database to be able to insert the data into a multi-layer perceptron, We notice that all our pre-processing has remarkably reduced our database: by choosing a black and white image and keeping only the coordinates of the black pixel, the data sizes have been reduced by at least a factor of ten. But it is not enough: we still have hundreds more points; our goal is to keep not more than a dozen.

We are mostly concerned about the computation time; in order to gain precious seconds in our process, we need to initialize the neurons' centers properly. So we sum every columns and lines of the matrix image form to obtain a projection on the two axes. These two projections will inform us where most of the pixel (and also the contours of the lips) are located.

Here are two graphs representing the value of the sum for the matrix columns and lines. The max values correspond to 255 (which is the value of the white pixels) times the lines or columns where there is no information. By default, we initialize in order to keep 95% of the information (but this parameter can be changed). We will so initialize around this zone and delete all the information that is too far (I have set the limit distance to 50 pixels away: we keep only the information that is in that range). This method avoids all undesired information as shown in the previous section. Here we show how we obtain coordinates that will be used for initialization.



Using this information, we will initialize our neurons in this area: my first thought was to initialize a rectangle around the lips. Then I realized by using this technique that some points would be then a few pixels too far from the lips' contour whereas some would be just on the contour; my algorithm will then lose time getting farthest neurons closer to the contour. So I have decided to make my initialization follow the general pattern of the contour of the lips, which is more ellipsoidal than rectangular: but because an ellipsoid will require too much computation time, I have coded a "pseudo-ellipsoid" pattern. The two different initializations are shown in the next two graphs:

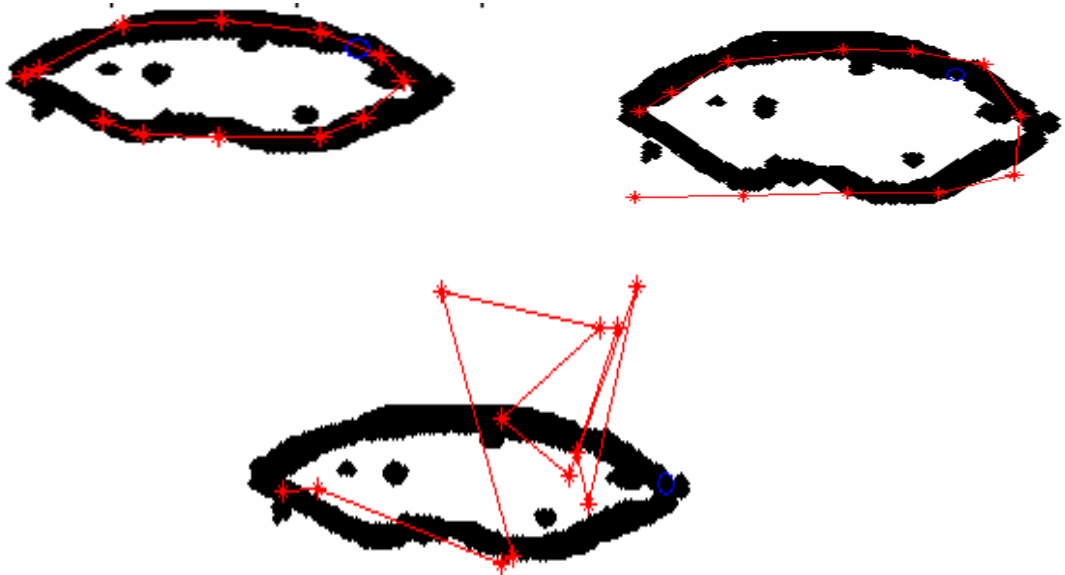


By getting closer to the lips' general patterns, the "pseudo-ellipsoid" initialization can also allow us to decrease the number of iterations that are necessary to attain a good estimate of the lips' patterns. On the next table, I show the number of iterations needed to obtain a reasonably good result.

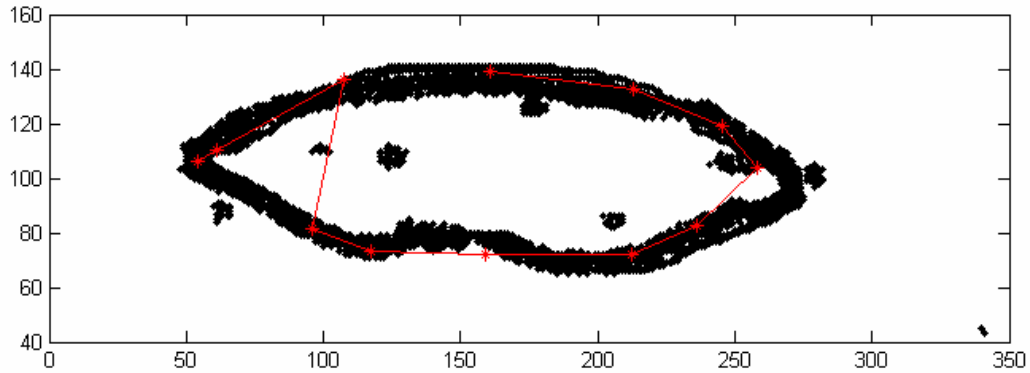
These numbers are experimental numbers: the algorithm has been applied to a large number of pictures, with a fixed eta ($\eta=0.1$).

Initialization	Random	Rectangle	“Pseudo ellipsoid”
Number of iterations to obtain convergence.	>500 (if convergence)	>70	>45

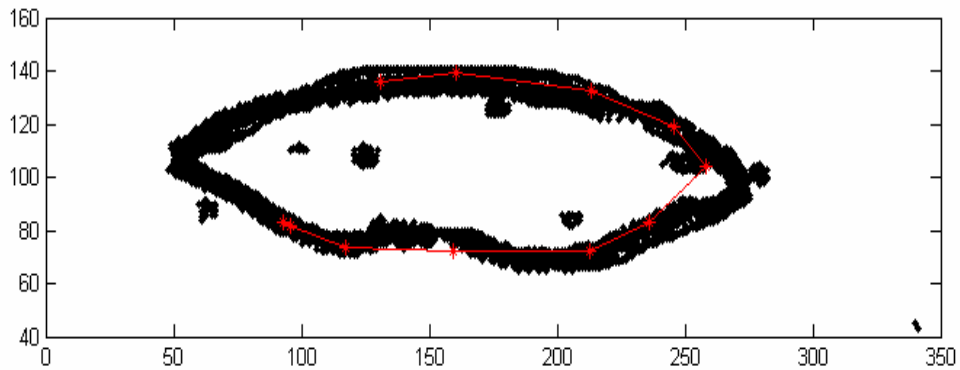
Here are some configurations after 50 iterations for pseudo-ellipsoid, rectangle and random initializations:



More than simply decreasing the number of iterations, the initialization is also very pertinent to avoid the crossing of the neuron branches and the way the branches “turn”. Following is one example where the neuron branches are crossing, due to bad initialization. Realistically we will notice that these problems of “branch crossing” sometimes still occur with a good initialization, but we will see how to avoid it in the next part.



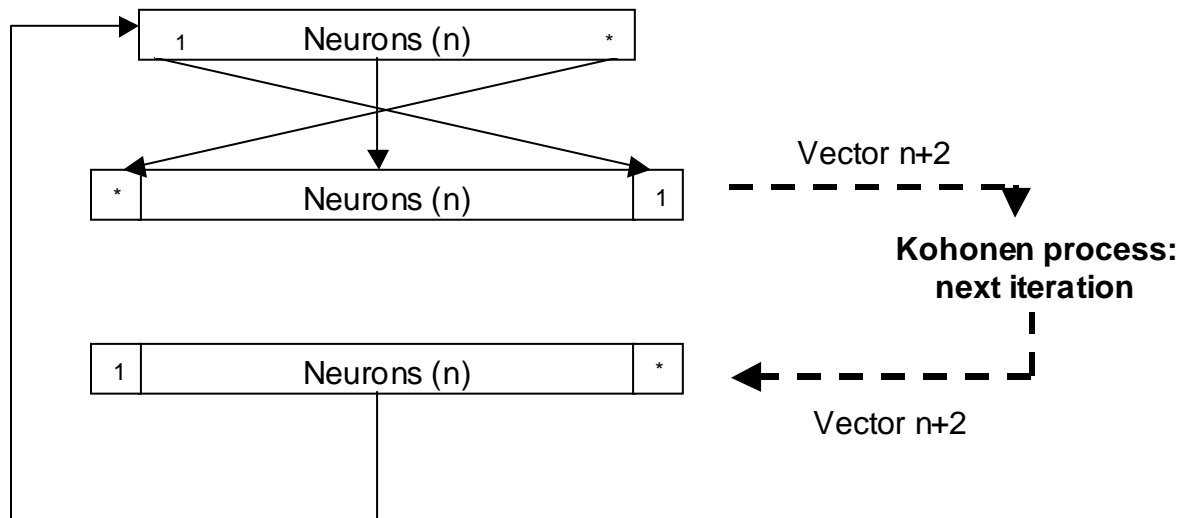
Our goal is to reduce the size of the information to a dozen neurons (in fact, the real size of the information is the number of the neurons time two, because we have two coordinates for each neuron). But an important fact is that we do not want to lose any information; a self-organization map does not stretch in a good way, if it turns and does not complete the entire circumference, we will lose a lot of information. Here is one example where the neurons are turning and they ended only on the right side.



These problems appear very often if the initialization is randomly chosen, but even if we properly initialize our neurons' positions, it can happen, so we need to find a new processing to avoid these problems.

2. A way to avoid the crossing problems.

We know that in a Kohonen self-organization map, every neuron is linked to its nearest neighbors. Actually, a neuron is linked to all the other neurons but it is more sensible to the position of the nearest one. Each time we modify the position of a neuron we also change the positions of its neighbors using a neighborhood function (usually a gaussian distribution function, with the parameters η). But the extreme neurons of the self-organization map do not have neighbors on both sides. They are only linked to one neighbor, instead of two as the other neurons are. So, I have decided to make what I called a “loop on the neuron branch” in order to link the two extreme neurons together. Because if the two extreme neurons are linked together, we can be sure that the general form of the self-organization map will be an ellipsoidal form and would not be able to be twisted and become a half ellipsoid as shown above. Thus we proceed as follows:



We copy the first and the last neurons in a new set of neurons (which now contains $n+2$ neurons) so that the first and the last neurons (1 and * in the previous graph) are neighbors. Then we apply the Kohonen process to the new set of neurons. In this way, we have virtually placed the first and the last neurons as neighbors. For example, every time we modify 1, we also modify 2 (as usual) and the last neuron *.

Thanks to this method, the branch now keeps its aspect of a “loop”, and so does not turn anymore. We have also avoided the branch crossing which now practically never

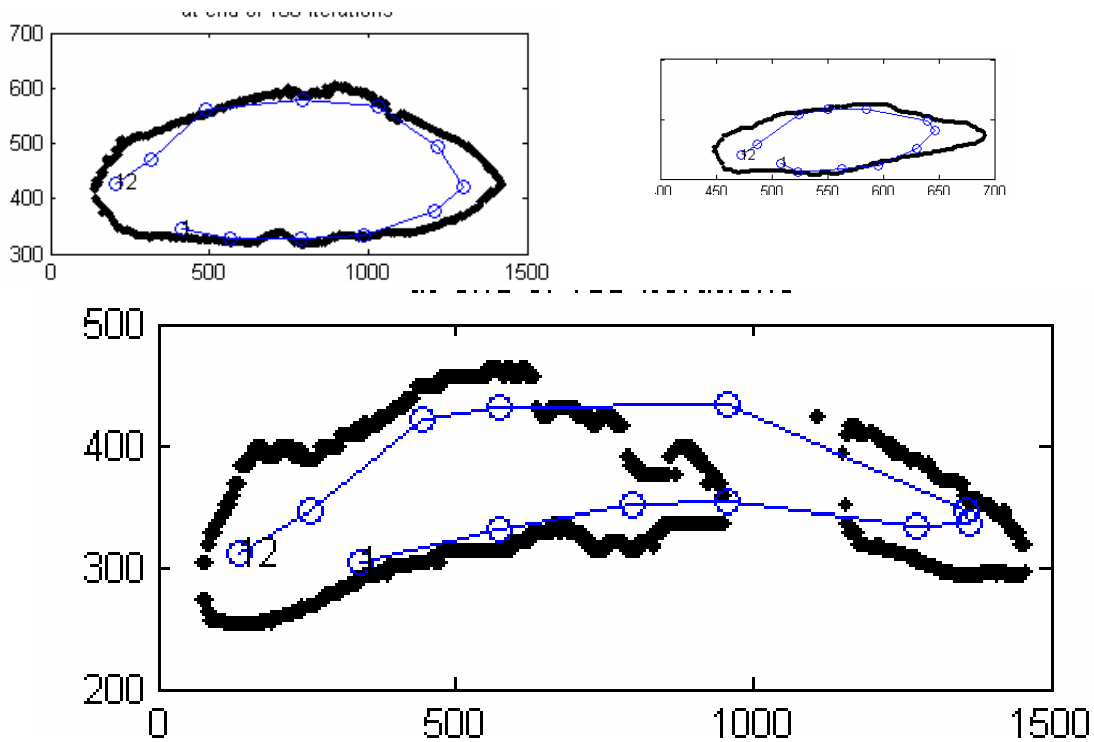
occurs anymore. And so we have solved our previous problems, and are sure that we keep a reasonable overview on our contour.

3. Application to a real-life problem: Selection of parameters and results.

There are several parameters for the Kohonen self-organization map (SOM). The first one is the number of neurons in the SOM. Because we want to limit the dimension of our problem, we need to keep a reasonable number of neurons. After several tries, we can see that 12 neurons is a good number: not too big so the computation time will not be too long and big enough so we can represent the contour. It seems fair to represent a contour with only 12 points: we need to get the extreme left and right points and the upper and lower lips can be well-described by 5 points each. We notice that this choice may seem arbitrary (as usual in the neural networks domain) but it came from several tests and depends on the size of the pictures. With a bigger picture, or another problem, we can find another value.

Other parameters are eta (parameters of the gaussian neighborhood function) and the number of generations. We will briefly review that the parameter eta described how each neurons is sensible to its neighbors: if eta is large, a neuron will move rapidly and be sensible to every points and its entire neighborhood will move with it, but after a few iterations, as eta decreases, each neuron will then be sensible to its closest neighborhood. We can describe these two phases as the ordering phase and the convergence phase. Because of the particularities of our initialization, the ordering phase is almost finished before starting the algorithm; all the neurons are ordered in a “pseudo-ellipsoidal” pattern around the lips and because of that, we can choose a small eta so we almost start directly to the second phase, the convergence phase. The idea is to make the neurons converge very fast because we are preoccupied by the computation time. The parameter eta and the number of iterations are linked. I have launched the algorithm for my entire picture database with several values of eta and I have observed how many generations we need to have a fair result.

Here, one can set $\eta=0.1$ and 65 iterations (for our three cases, 50 is generally enough but we prefer to be sure that the algorithm is very close to the contour).



By observing the results from the three previous cases, we can see that the algorithm is very robust: it even finds a very good shape for the “E” case (which is the most difficult because it is spread in two different parts). These results encourage us to apply the next step: the multilayer perceptron.

All the filters, computations and plots of this part are done by *SomKohonen.m*.

Multi-Layer Perceptron: Final result.

1. Generation of the Data sets.

We have three sets of pictures: each set contains six different pictures for every sound (each picture is name with the letter of one of the three sounds (U, E, A) and a number, for example *U4.jpg*). We apply the *main.m* program which runs the two programs of the previous part (*SomKohonen.m* and *preprocessing.m*) for every picture. This run is simplified by *TestingDataCreation.m* and *TrainingDataCreation.m*. We use the five first pictures of every sound to generate the training data and we test on the sixth. We can set up the size of each set. The two algorithms have been run and results have been recorded: it is possible to directly obtain two data sets with *wtrain.mat* and *wtest.mat*. By default, these sets are loaded by the multilayer perceptron.

The three sounds are coded with a binary set of 3 vectors [0 0 1], [0 1 0] and [1 0 0] for each sound. We use the outputs of SomKohonen: 12 outputs of two coordinates. Our multi layer perceptron input is a 24 vector. In order to avoid having the position of the mouth in the picture being a parameter in the recognition we subtract the smallest value to every coordinates: two neurons coordinates are so equal to zero. We can see this process as a normalization process. This normalization increase the recognition rate by 5 to 10%.

2. Perceptron design and final results.

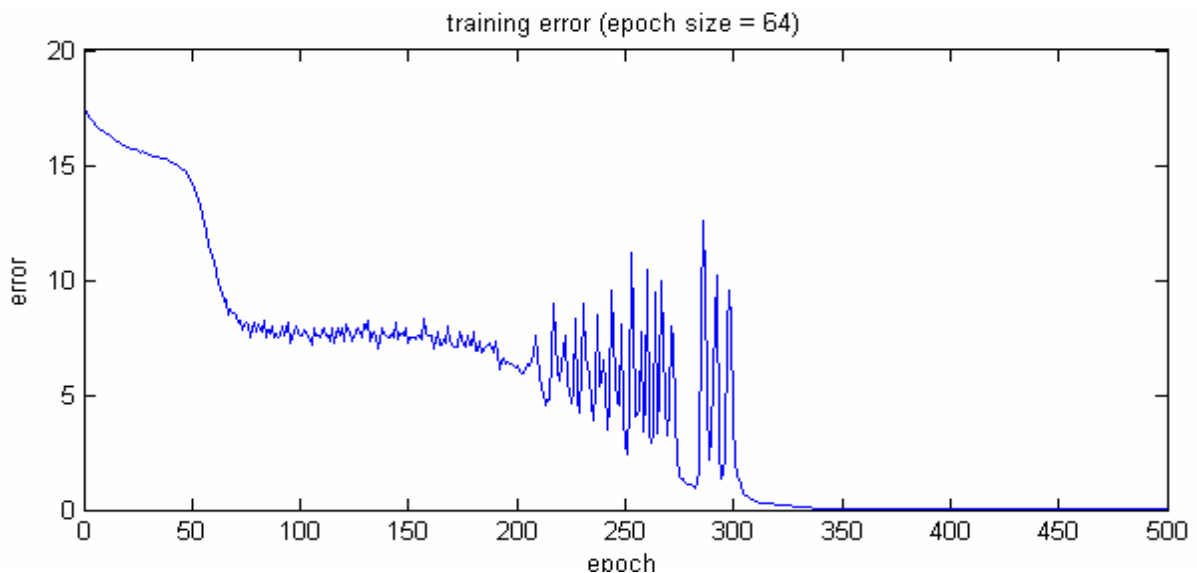
The data sets used here are *wtrain.mat* and *wtest.mat*. *wtrain.mat* contains 225 training vectors, and *wtest.mat* contains 75 testing vectors. Because of the high dimension of the problems (the dimension is 27, with 24 inputs), we need a reasonable number of neurons in the hidden layers to find a good classification rate. We notice that even if adding a few neurons will affect the training computation time, the final test on the data sets is still very fast because once the multi layer perceptron is trained, it just needs some basic matrix operation: it is still accurate for a low power system.

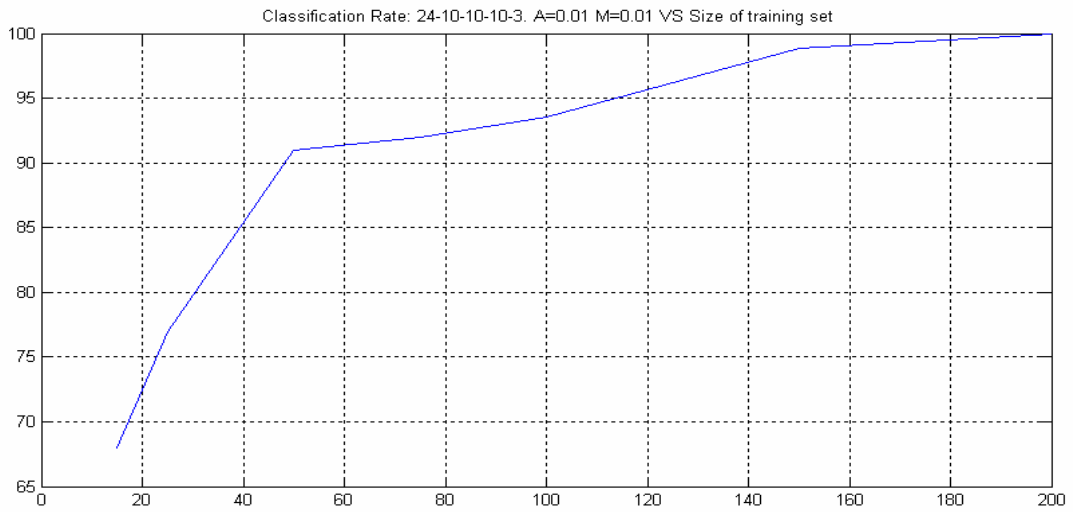
The two files are then implemented to the multi-layer perceptron: *Mconfig* and *MultiLayerPerceptron.m* are the modified version of the Professor Hu's codes: *bpconfig.m* and *bp.m* in order to fit the particularities of the problem. Using default parameters and data sets, we just need to run *MultiLayerPerceptron.m*.

In the following table, I use the default parameters for all the previous programs and I use all the data set provided by *wtest.mat* and *wtraining.mat*.

Layers (Input layer+hidden)	alpha	momentum	configuration	Testing classification rate(%)	Training classification rate(%)
2	0.1	0.8	10	27	33
2	0.05	0.05	10	73.33	93
2	0.01	0.01	10	92	100
3	0.1	0.8	10 10	52	76
3	0.01	0.01	10 10	100	100

The algorithm needs a lot of neurons to provide good results (10 seems to be a reasonable choice). Here I present some test which I find sum up the idea of the configuration: a lot of tests prove that alpha and the momentum need to be small to observe a good rate, and if we can get a good training classification rate, the key point is to obtain a good final classification rate. My final choice will be a 3-layer perceptron with Alpha = 0.01 and Momentum=0.01. This configuration always leads to a 100% classification rate if we work on the entire set of the data and it converges very quickly in less than 500 iterations:





Size of the training set	15	25	50	75	100	150	200
% Classification	68	77	91	92	93.555	98.8677	100

Above is the graph of the classification rate with respect to the size of the training set: we can observe that we need at least 150 vectors to train the multi-layer perceptron, and 250 vectors seem to be appropriate to be sure to have a 100% classification rate.

Discussion and Conclusion

The first idea of using a self organization map to reduce the dimension of the information before applying a MLP, seems to be very efficient. We have obtained a perfect recognition rate while working on a real life problem with decent material.

But, to be applied to a low power system (cellular phone, or PDA), our algorithm needs to be very fast. The multi layer perceptron is not the problem here: it only makes some basic matrix multiplications and converges very fast (less than 500 iterations). And we can imagine a firm training or pre-training the algorithm before selling the system to a customer, it also seems reasonable to spend one hour configuring the system if we can be sure that it will then always work. The Kohonen algorithm is very fast too. With a few loops and a few matrix computations, the slow part of the overall algorithm is the preprocessing data: the function *imread.m* called in *preprocessing.m* uses 80% of the computation time. If we can find a way to code this function in a fast way or if we implement the algorithm in a picture-oriented language or in a more basic language than Matlab, we can hope to reach very good results.

Matlab files and used files

Matlab files:

- Programs:

preprocessing.m
SomKohonen.m

Fit a image (.jpg by default) to the input of the SOM.
Self Organizing Map program: take coordinates of a set of points and gives back coordinates of 12 neurons which is a representation of the initial set.

Main.m

Apply *preprocessing.m* and *SomKohonen.m* to a picture jpg. Give the SOM representation of the pictures.

TestingDataCreation.m

Create a testing set: main.m applied on U6,E6 and A6.jpg

TrainingDataCreation.m

Create a training set: use the jpg training pictures.

MultiLayerPerceptron.m

MLP backpropagation driver program for classification tasks.

Mconfig.m

Initial configurations of a MLP

- utility routines (from Professor Hub *bp.m* code)

cvgtest.m

Convergence test routine.

bptest.m

Test classification (*bptest.m*) and approximation (*bptestap.m*) results

bptestap.m

randomize.m

randomize the row order of a matrix

rsample.m

random sample K out of Kr rows from matrix x

scale.m

called by *bp.m* to linearly scale the input to a specified range

actfun.m

Compute activation function and derivative.

actfunp.m

Compute activation function and derivative.

- Data:

wtrain.mat

Example of training and testing set.

wtest.m

Default sets for *MultiLayerperceptron.m*

-JPG Files:

sound [**u**] "U": U1, U2, U3, U4, U5 training set. U6 testing set.

sound [**a**] "A": A1, A2, A3, A4, A5 training set. A6 testing set.

sound [**i**] "E": E1, E2, E3, E4, E5 training set. E6 testing set.

References:

Papers and articles:

[1]-**Image compression by Self-Organized kohonen Map**

Christophe Amerijckx, Philippe Thissen..IEE Transition on Neural Networks 1998.

<http://www.dice.ucl.ac.be/~verleyse/papers/ieeetnn98ca.pdf>

[2]-**SRAM bitmap shape recognition and sorting Using Neural Networks.**

Randall S. Collica. IEEE.

http://www.ibexprocess.com/solutions/wp_SRAM.pdf

[3]-**From your lips to your printer.**

James Fallow.

<http://www.theatlantic.com/issues/2000/12/fallows.htm>

[4]-**Intel Researchers Teach Computers To 'Read Lips' To Improve Accuracy Of Speech Recognition Software**

<http://www.intel.com/pressroom/archive/releases/20030428tech.htm>

<http://www.intel.com/labs/features/sw04034.htm>

[5]-**LAFTER: A Real-time Face and Lips Tracker with Facial Expression Recognition.**

Nuria Oliver, Alex Pentland, François Bérard

<http://citeseer.nj.nec.com/362903.html>

[6]-**Computers learn to read lips.**

Montana Associated Technology Roundtables.

<http://www.matr.net/article-8026.html>

[7]-**A kohonen Neural Network Controlled All-optical router system.**

E.E.E Frietman, M.T. Hill, G.D. Khoe.

<http://www.ph.tn.tudelft.nl/~ed/pdfs/IJCR.pdf>

Books:

[8]- **SRAM bitmap shape recognition and sorting using neural networks.**

Collica, R.S., Card, J.P., and Martin.

W. ISBN 0894-6507

[9] **Neural Networks: A Comprehensive Foundation.**

Simon Haykin,

Prentice Hall, New Jersey, second edition, 1999