Ship recognition using
Distributed Self Organizing Maps

Victor Jose Lobo\textsuperscript{1}, Nuno Bandeira\textsuperscript{2}, Fernando Moura-Pires\textsuperscript{2}

\textsuperscript{1} New University of Lisbon/Portuguese Naval Academy, Escola Naval, Alfeite, 2800 ALMADA, PORTUGAL, vsl@di.fct.unl.pt
\textsuperscript{2} Dep.Informatics, New University of Lisbon, Quinta da Torre 2825 MONTE DA CAPARICA, PORTUGAL, nb@di.fct.unl.pt, fmp@di.fct.unl.pt

Abstract

A ship recognition system by the classification of acoustic signatures is presented. Recognition is achieved by matching received signatures with clusters in a map obtained by distributed training of a Kohonen’s Self Organising Map. The system is presented as implemented using a Parallel Virtual Machine (PVM) layer running over a network of Intel based PCs running Microsoft Windows95.

Keywords: SOM, distributed algorithms, parallel algorithms, neural networks

Introduction

The authors of this article have been involved in a project that requires analysis and classification of passive sonar signals. Basically our aim is to distinguish ships using the noise they produce in the ocean (their acoustic signatures). In previous work, we showed that Kohonen’s Self-Organising Maps (SOM) can be used very effectively as classifiers [Lobo 95], with the advantage that they can also do novelty detection. These maps are also very efficient tools for exploratory data analysis [Ultsh 89], which is very important for us.

Our main objective is to develop a low-cost system that can run on an ordinary PC with a sound card, and running Microsoft Windows operating system. The system should be able to receive an audio signal from the sonar, digitise it, show it’s spectrogram on the screen, and classify it according to some pre-loaded library. Its interface should be as simple as possible so as to require minimum training by the operator. Furthermore, the result of the classification must be easy to interpret. It has been pointed out in many papers that Kohonen’s SOM is particularly suited for this task, as is produces a 2D map where the different classes can easily be distinguished, and where new samples are mapped near similar classes, although clearly identified as not belonging to them.

The main problems with our previous versions was that they could not classify sound in real time, their user interface was cumbersome, and they took a very long time to train a map.

The problem of classifying in real time almost solved itself thanks to the always increasing computing power of modern PCs, but we did optimise our code. We also re-designed our user interface, making the system much more user friendly. The final problem (the very long training times), was solved by distributing the processing load on a number of networked computers. To keep to our objective of producing a low-cost and easily available system, we chose to use networked PCs running Microsoft Windows and a “Parallel Virtual Machine” PVM layer [Geist 94], [Alves 95]. A PVM layer enables networked computers to host processes that interact in a way similar to the processes in a Unix machine.

We feel that distributing the work load over networked PCs running MS-Windows is a very powerful solution, since most organisations have a large number of these computers, that are basically idle during nights and weekends. During these periods they could be used to train very large Self-Organising Maps, possibly for data-mining purposes. The computers to be used only need to have a PVM daemon running, that could be started when they boot.

Application

An average operator of the program uses only the main screen. As it can be seen in Figure 1 this screen has a trained map on the left (which has a color legend below) and a spectrogram on the right, to visualise the evolution of the spectra through time. When in operation, the system runs as a real-time classifier: the
operator sees the spectrogram moving up (new spectra appearing at the bottom) and sees a blinking sign
on the map indicating the winning neuron for the currently accepted spectra. It is very easy for the
operator to identify the ship by matching the color of the winning neuron with the color legend below the
map.

If the operator finds a new kind of ship for which the system hadn’t been trained for (easily detectable
because the winning neuron should be in a white zone or oscillating between 2 or more clusters), he
should record it, using an option available on the menu bar. If possible the operator should try to identify
the ship using classical methods, and include that information with the recording. This recording can later
be used to build a new, more effective map, for further classifications. Sometimes, an operator may find
himself in a situation where he can’t retrain the map and needs to identify the new ship in case he finds it
again. There are some simple tools developed to assist him on this task. On one hand, the program allows
the operator to re-label the winning neurons and updates the map legend. On the other hand, the ‘Visualise
spectra’ option allows the operator to visually compare the spectra with other pre-recorded ones.
Nevertheless, even without these tools, the proximity of the winning neurons (for the incoming, unknown
signal) to the surrounding clusters should be an indication of that ship’s nature.

The training of a new map is a heavy process because there is the need to keep all the frequency vectors
(old and new) in memory plus the new map (width * height neurons) and we were dealing with 2048 bins

Figure 1 - The main screen of the application

in each frequency vector, each bin having size double. The training can be done on-site or in a computing
center. In the latter case it will, almost surely, be possible to use the distributed training option on 2 or
more computers. If the time necessary for the normal training process is available (i.e. its reasonable to
assume the system won’t be used for classification for a long while) then it is also possible to train a new
map on-site, on a single computer.

On-site training should be avoided because the choice of the parameters for the training process
shouldn’t be fixed, i.e. the intervention of a knowledgeable specialist, while not essential, would be highly
recommended. If there is more than 1 machine available then the distributed option should be used – the operator only needs to tell the system of the number of available machines to distribute the initial map. Then he should proceed to the map training dialog (see Figure 2), fill the Edit/Combo boxes with the training parameters, start the training process, and wait for completion. Once the training is done, only a simple calibration process [Kohonen 95] is needed to achieve operational status.

**Architecture**

In order to achieve the above-mentioned degree of customisability a particular implementation of SOM had to be developed. There were two separate paths of development: the interface layer and the distributed SOM layer. The interface layer was thought from the start to work exclusively in Windows95 due to the large number of Windows/Intel machines available, ease of use and also due to ease of interface development for Windows95 (many visual components were available and easily customisable in our chosen development tool).

However there were no constraints on the development of the distributed SOM layers, so our main reference goals in the development of this layer were developing portable, flexible and efficient code. Such would enable us to distribute SOM over heterogeneous architectures (trying to reach as far as PVM implementations could go) and still maintain an acceptable level of performance. Last but not least, a significant degree of flexibility was desired in order to implement a SOM layer that could be used to test different approaches, such as experimentation with different neighbourhood functions, distance functions, alpha convergence, etc. This lead to the development of a structured object oriented architecture that implements the desired level of flexibility and portability in a natural way by defining a set of classes (Neighbourhood, Alpha, Distance) which can be derived to different approaches and know how to port themselves to different architectures through PVM.

The chosen distribution algorithm is shown in Figure 3 and can be described as follows (using $N_p$ processors (PVM hosts), a coordinator process $C$ (within $N_p$), $N_t$ training patterns $x_i$ and $N_n$ neurons forming a SOM):

1. **Assign the neurons.** Assign the $N_n$ neurons to the processors, in such a way that each processor receives approximately the same number of neurons ($N_p/N_n$), and that for any given area of the SOM the neurons are evenly distributed amongst the processors.

2. **Set up the training set.** Send all the $N_t$ training patterns to all the processors, together with the initial training parameters.
3 - Train the neurons. For each pattern $x_i$ in the training set, do the following:

3.1 Calculate the local winner neuron in each processor (calculation phase)

3.2 Send all local winner neuron coordinates and distances to the pattern $x$ to the coordinator. At the coordinator, select the global winner, and send its coordinates to all processors (voting phase)

3.3 Update the neurons at each processor, according to the update function $F$ and update the network training parameters (update phase)

4 - Repeat step 3 until the stopping criteria is met

5 - Send all neurons back to the coordinator.

Conclusions

A full presentation of our results can be found in [Bandeira 98], but we will briefly discuss the most relevant findings. The overhead imposed by the distribution is quite significant, so performance will actually degrade when we try to distribute small maps over a network. Furthermore, even when we can achieve gains by distributing the map, there is an optimum number of machines that should be used, because if we have too many machines the ratio of processing provided to communication overhead imposed becomes less than 1.

However the distributed SOM can achieve very impressive gains with large maps. This improvement is due not only to the increase in processing power but also due to the increase in available RAM memory (that avoids disk swapping). By avoiding disk swapping, we can achieve a 20 fold decrease in training time. Furthermore when distributing the training over 12 computers, we managed to decrease the training time ten fold due only to the increase in computing power. The overall gain in this case can add up to a 220 fold decrease in total training time.

The program developed proves several important points:
1) Parallel computing can be achieved with low-cost and generally available hardware and software
2) Very large Self-Organising Maps can be trained in a reasonably short time using this method
3) Real-time classification using a very large SOM can be achieved.

References


