PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 14-09

Recovering Historical Climate Records using Artificial Neural Networks in GPU

Juan Pablo Balarini, Sergio Nesmachnow

2014

Recovering historical climate records using
Artifical neural networks in GPU
ISSN 0797-6410
Reporte Técnico RT 14-09
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, 2014

Recovering Historical Climate Records using Artificial Neural Networks in GPU

Juan Pablo Balarini, Sergio Nesmachnow

Centro de Cálculo, Facultad de Ingeniería, Universidad de la República, Uruguay
{jpbalarini,sergion}@fing.edu.uy

Abstract. This article presents a parallel implementation of Artificial Neural Networks over Graphic Processing Units, and its application for recovering historical climate records from the Digi-Clima project. Several strategies are introduced to handle large volumes of historical pluviometer records, and the parallel deployment is described. The experimental evaluation demonstrates that the proposed approach is useful for recovering the climate information, achieving classification rates up to 76% for a set of real images from the Digi-Clima project. The parallel algorithm allows reducing the execution times, with an acceleration factor of up to 2.15×.

Keywords: artificial neural networks, image processing, climate records, GPU

1 Introduction

Studying the behavior of climate variables through time is crucial for science, industry, disaster prediction, and many other applications. Climate prediction is very important in short-term decision making, i.e. in agriculture, but also for long-term situations, i.e. to know sea levels in the next hundred years. Specific applications, such as bombing control on sewer systems, or water level prediction in a flood warning system can benefit of knowing long-term series (30+ years) of climate variables.

The scientific community is interested on recovering climate data stored through the years. When the search looks back in time, climate records are very scarce and difficult to recover. Furthermore, the preservation of records gathered in the pre-digital era is in danger of destruction. In Uruguay, we keep a systematic recording of climate variables from the early 1900s, most of them stored in paper. This data is of great value for science, but it has limited utilization in nowadays computerized systems, mainly due to the paper storage, and of course its preservation is in danger.

A transcription is needed for the historical information to be properly used on nowadays systems. In some cases, just like for pluviometer records, the transcription can be automatized by using digitalization and data recovering. The Digi-Clima project [1] proposes developing an efficient application for digitalizing the scanned output of pluviometers, originally collected in graphic paper bands, recovering the climate data, and storing it in a database of historical climate records.

Previous articles have presented the application of parallel scientific computing techniques to solve the problem tackled in the Digi-Clima project. In [2, 3] the solutions developed by applying parallel computing techniques in cluster and grid computing platforms were described. Both approaches were appropriate to solve the problem, achieving accurate speedup values when executing in dedicated computing infrastructures. The approach described in [4] is based on using volunteer-computing platforms. All the previous works were based on using specific image-processing techniques for recovering the information, which use several features of advanced MATLAB routines, including interpolation, counting, noise reduction, and others. In this article, we introduce a novel approach to tackle the problem using soft computing techniques: we apply neural networks (ANN) trained on Graphic Processing Units (GPU), to recover the rain information stored in paper data bands.

The main contributions of this article are: i) introducing an ANN approach for recovering historical climate information, ii) a deployment over GPUs that allows reducing the execution times significantly, and iii) the experimental analysis performed using a set of representative images from the Digi-Clima project, which demonstrates the efficacy of the proposed approach, achieving classification rates up to 76%.

The rest of the article is organized as follows. Section 2 describes the Digi-Clima project and reviews related works about historical climate data processing. Section 3 introduces the main concepts about ANNs and GPUs for parallel computing. After that, the strategies for climate data classification using ANN are described in Section 4. Section 5 reports the experimental evaluation of the proposed approach, studying the solution quality and computational efficiency. Finally, Section 6 summarizes the conclusions and formulates the main lines for future work.

2 Digi-Clima: Recovering Historical Climate Records

This section describes the Digi-Clima project and related works about recovering historical climate data.

2.1 The Digi-Clima Project

In Uruguay, the National Direction of Meteorology systematically stored climate data (manuscript reports, graphic records of pluviometers and other devices) from the early 1900s. They are very useful for studying climate phenomena and for weather forecasting using numerical models. However, analyzing and using the historical climate data is very difficult due to the paper-format in which they are stored, and their preservation is in danger, because of the paper deterioration when the records are handled.

The Digi-Clima project proposes developing a semi-automatic method for digitalizing the rain intensity records from weather stations across the country, in order to guarantee their preservation. The resulting data is stored in a database of historical climate records, making possible the utilization of the data in numerical models.

The rain records are stored in millimeter paper bands, where a pluviometer recorded the amount of rain accumulated in a certain period for a given zone of the country.

Each band contains the pluviometer records, and on the reverse, manuscript annotations indicating the beginning/end of the register, and the scale for the maximum value of the rain records for the band, among other significant information. The pluviometer draw a continuous line, reasonably smooth and (piecewise) monotonically increasing, indicating the amount of rain accumulated on the device. The value grows until the pluviometer capacity is reached. Then, a vertical fall to zero indicates that the device is emptied, and the measuring process starts again.

Fig. 1 describes the recording process using pluviometers and presents an example of the historical rain records. The sample band shows several of the troubles that make the recovering problem difficult: a discontinuous ink line, ink stains, different intensity levels, and other defects due to measuring using an analogic device. Furthermore, manuscript annotations are also present in the band to account for additional data, usually written by operators when removing the band from the pluviometer.

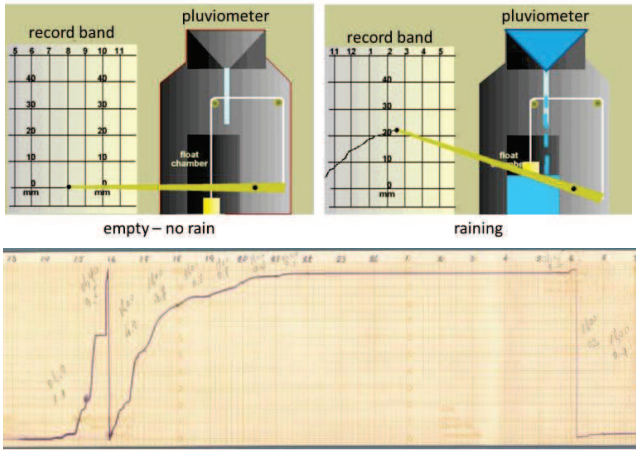

Fig. 1. Recording process and historical record band.

2.2 Related work

Some previous works have tackled similar problems, but using different approaches than the semi-automatic recovering proposed by Digi-Clima. A number previous works were focused on analyzing specific short-time climate phenomena, thus they analyzed significantly fewer volumes of data than the ones handled by Digi-Clima.

One of the first initiatives for systematic recovering historical climate data was the CLIWOC project [5], which built a database of climate observations based on thousands records (date, location, weather, wind speed/direction, sea, ice, air temperature and pressure) stored manually and in natural language from vessels logbooks between 1750 and 1850. CLIWOC also developed a dictionary that allowed unifying several criteria and terminology used in the logbooks in more than 100 years.

The RECLAIM project [6] continues the work started in CLIWOC, by processing data that were not included in the original process. At the long-term, RECLAIM proposes processing records from the instrumental era (1853–nowadays).

The Old Weather Project [7] focuses on extracting data from digital images by using volunteer human operators. In order to make more attractive the task and gathering more collaborators, the project developed an online game: a player subscribes and chooses a real ship with climate data logbooks to digitalize, earning experience and points by transcribing data, which allows him to be promoted to higher ranks.

The previous projects are related to Digi-Clima, since they propose transcribing historical climate records, and demonstrate that the research community is interested in the problem of recovering historical climate data stored in non-digital format.

Other works [2,3,4] applied parallel computing techniques in cluster, grid, and cloud computing platforms to speed up the digitalization process in Digi-Clima.

The review of the related work allowed our research group to understand the state-of-the-art about recovering historical data. However, none the similar projects have developed efficient methods for automatic transcription of data, because all of them are based on human operators. Furthermore, there are no antecedents of applying soft computing/parallel computing techniques to the recovering process. Thus, the approach applied in this article is a contribution in this line of research, by proposing using an efficient ANN approach to efficiently solve the digitalization problem.

3 Artificial Neural Networks and GPU computing

ANNs provide a practical method for learning real, discrete, and/or vector-valued functions from examples, by fitting a training set of input-output pairs. ANNs are robust to errors in the training data and has been successfully applied to problems such as image and speech recognition, and learning robot control strategies [8].

Fig. 2(a) presents the general schema of an ANN. There is a set of *neurons* connected with each other. Each neuron receives several input data, perform a linear combination and produces a result, which evaluates some function $f(x)$ for the value x.

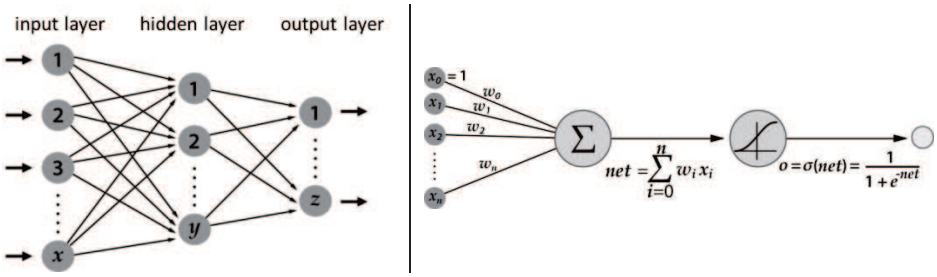

Fig. 2. (a) Schema of an ANN and (b) A single neuron.

The neurons are grouped in several layers: i) *input layer*: receives the problem input; ii) *hidden layer(s)*: receives data from the input layer or from another hidden layer, and forwards the processed data to the next layer (there may be multiple hidden layers with multiple neurons each); iii) *output layer*: determines the output of the processing for a certain problem instance (may have multiple neurons).

Fig. 2(b) shows a schema for a neuron: first, a linear combination of the neuron input data x_i, weights w_i, and an independent coefficient w_0 is made; then, the output is evaluated at some well-known *activation function*, to produce the neuron output.

In this article, the ANN used to recover historical climate data is trained with the *backpropagation* algorithm. Backpropagation learns the weights for a multilayer ANN with a fixed set of units and interconnections, by applying the gradient descent method (Algorithm 1) to minimize the squared error between the output values and the target values. The learning problem faced by backpropagation implies searching in a large space defined by all possible weight values for all neurons.

Backpropagation (training_examples, n, n_{in}, n_{out}, n_{hidden})
Each training example is a pair $\langle \vec{x}, \vec{t} \rangle$, $\vec{x}$ is the vector of ANN input values, and $\vec{t}$ is the vector of target ANN output values; n is the learning rate, n_{in} is the number of ANN inputs, n_{hid} the number of units in the hidden layer, and n_{out} the number of output neurons.
The input from neuron *i* into neuron *j* is x_{ji}, and the weight from neuron *i* to neuron *j* is w_{ji}.

1. **create** a feedforward ANN with n_{in} inputs, n_{hidden} hidden units, and n_{out} output neurons.
2. **initialize** all w_{ji} to small random numbers
3. **while** the termination condition is not met **do**
   for each $\langle \vec{x}, \vec{t} \rangle$ in training_examples **do**
   input instance $\vec{x}$, compute output o_u for every neuron u {propagate input forward}
   for each output neuron k, calculate the error term δ_k {propagate errors backward}
   $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
   for each hidden neuron h, calculate the error term δ_h {propagate errors backward}
   $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh} \delta_k$
   $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$, where $\Delta w_{ji} = n \delta_j x_{ji}$ {update each w_{ji}}

Algorithm 1. Stochastic gradient descent backpropagation algorithm for feedforward ANNs.

Backpropagation starts by constructing an ANN with the desired number of hidden/output neurons and initializing all weights to random numbers. Given this fixed ANN structure, the main loop of the algorithm iterates applying the ANN to each training example, computing the gradient with respect to the error on the example, and then updating all weights in the ANN using the *learning_rate* constant. The gradient step is iterated (using the same training examples multiple times) until the ANN performs acceptably well [9]. The propagation of the input data through the ANN is applied for evaluating a single instance. The presented ANN uses neurons of sigmoid type with activation function $F(x) = 1/(1 + e^{-x})$.

GPUs provide a large computing power by using hundreds of processing units with reasonable clock frequencies. In the last ten years, GPUs have been used as powerful parallel hardware to execute applications efficiently. High-level languages were developed to exploit the GPU capabilities. NVIDIA CUDA [10] extends the C language, providing three software layers: a low-level driver for CPU-GPU data transfers, a high-level API, and a set of libraries for scientific computing. GPUs are able to create, manage, and execute a large number of light processes (*threads*) in parallel, with reduced overhead. Threads are grouped in *blocks* (with up to 512 threads), executed in a single multiprocessor on the GPU. Three memory spaces are available: the *local memory* of each thread, a *shared memory* for threads in a block; and the *global memory* of the GPU. Two important features to achieve efficiency are avoiding thread creation (reducing the thread creation overhead), and minimizing the access to slow local and global memories, preferably using the shared memory which is placed within the GPU chip, providing a faster way to store the data.

In this work, a specific version of the backpropagation algorithm was implemented in GPU. We adapted Algorithm 1 to take into account the communication between the GPU processing units. GPU threads are assigned to execute over certain neurons on the ANN. Before executing a on the GPU, a function-dependent domain decomposition is applied to maximize the parallel execution, (i.e. each GPU thread works independently from each other), and to avoid serializations in the memory access.

4 Historical Climate Record Classification using ANN

This section describes the proposed strategies for recovering climate records using ANNs. All approaches are based on using the data of a *training set* of images to train the ANN to be used later to process other images in the Digi-Clima database.

4.1 Proposed strategies

We studied three strategies for solving the problem, which are described next.

Full image, one neuron per pixel. Our first idea was applying a common approach for image processing with ANNs, using all the image data (one neuron per pixel) as input (Fig. 3). This approach has been previously used for face recognition [10, 11], and we adapted it for pluviometer image processing by including a second hidden layer and the parallel GPU version. The output of the ANN is the time series of rain records.

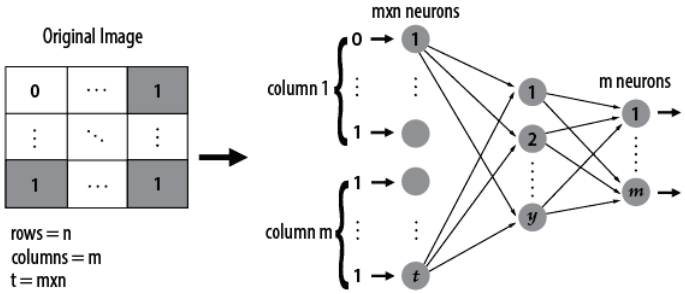

Fig. 3. Using the full image information for an $M \times N$ (pixels) image.

After performing validation experiments, this approach was discarded mainly because the ANN gave much importance to noise (background, annotations), failing to learn the correct information. Furthermore, for Digi-Clima images, we also detected other drawbacks: i) both ANN/GPU did not properly handle full information from large uncompressed images, ii) the GPU memory limit is easily reached, and iii) the processing times increase significantly, making more difficult to validate the solution.

Swapping strategies: by zones and by rows/columns. Swapping strategies are based on reducing the input information for the ANN. Initially, we divided the image in zones, taking into account the relative position of each zone for the classification (Fig. 4(a)). This approach was unable to get a good learning behavior because the zones that do not provide useful information confuse the ANN.

After that, we proposed swapping horizontally/vertically, using rows/columns as the ANN input (Fig. 4(b)). After initial validation experiments performed on random black and white images with size 32×30 pixels, we concluded that using columns is a more useful approach, due to the continuity of the rain records and the reduced influence of noise. Swapping by rows confuses the ANN, mainly due to large discontinuities on row values and due to existing more than one rain value per row.

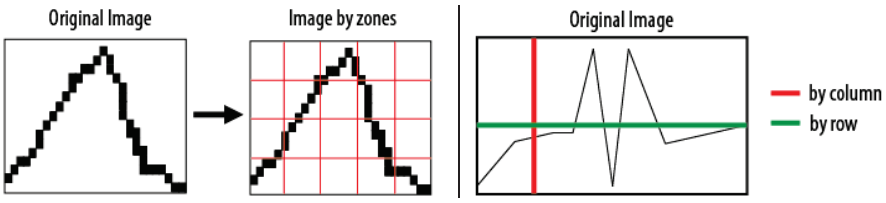

Fig. 4. Swapping strategies: (a) by zones and (b) by rows/columns.

An improved technique was then included in the column swapping approach, using information from nearby columns for the classification. This information is useful to process regions with discontinuities and breaks in the ink line. We followed an incremental strategy, by studying the ANN architecture and learning behavior for small images (32×30 pixels) and then increasing the size and complexity of the images, until finding a correct method for classifying representative Digi-Clima images.

4.2 ANN architecture and implementation details

Input and output. In order to reduce the information handled, the ANN input is encoded as real-valued numbers. Three approaches were studied for the ANN output: i) using one (real) output neuron for the functional value for each column; ii) encoding the value for a given column, having as many outputs as rows in the image: the ANN output for a certain column is the index of the activated output, or the one with the largest value if multiple neurons are active; and iii) encoding the value for a given column in binary, thus only $\log_2 n$ outputs are needed for an image with n rows, and those neurons corresponding to the output in "1" are activated. Preliminary experiments demonstrated that the first and the second approach obtained the best results.

Activation function and weights. We use an activation function that provides a trade-off between the activation/gradient variations, to enable the correct data flow in both ways of the backpropagation and reducing discrepancies between layers (n_{in} is the number of input neurons, and n_{hid} is the number of hidden neurons) [9]:

$$uniform[-4 * \frac{6}{\sqrt{n_{in} + n_{hid}}}, 4 * \frac{6}{\sqrt{n_{in} + n_{hid}}}] \quad (1)$$

We applied initialization policies to have small weights centered at zero, for the sigmoid activation function to operate in linear regime and maximize the gradient.

Learning process. Each column from each image on the training set is used to feed the ANN for training. The input for each iteration is the processed column and a given number (*total_neighbours*) of neighboring columns. The ANN input layer is loaded to

the GPU, using *total_neighbours* blocks, and as many threads per block as the number of rows in the input image; each block will load a column from the image.

To decide whether a column was correctly classified or not, an error threshold is used when contrasting the ANN output with the expected one. The classification rate for a set of images is the average of correctly classified columns for the images in the set. The training error for an image is the sum of the errors over all its columns.

4.3 GPU implementation

ANN training is a key component of the algorithm, which is performed entirely on the GPU. Fig. 5 shows the steps to train the ANN, for the entire training data set.

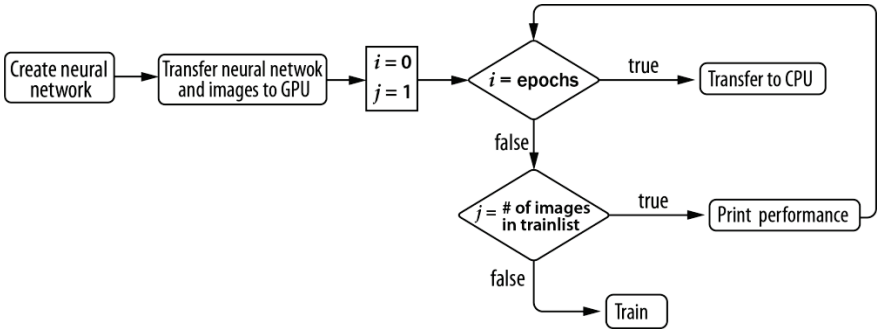

Fig. 5. Diagram of ANN training in GPU.

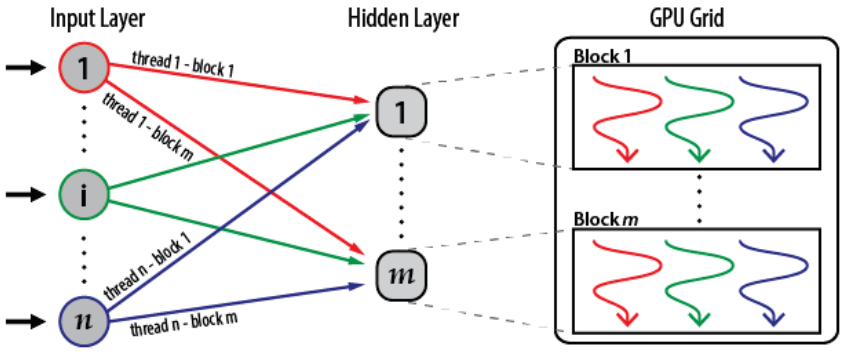

Fig. 6. Parallel ANN in GPU: forward from input layer to hidden layer.

The GPU architecture is mostly exploited when performing training and classification. For example (see Fig. 6), when *forward* is performed from the input layer to the hidden layer, the parallel algorithm creates as many blocks as hidden neurons exists. Each block will work with one neuron from the hidden layer and each thread in a block will compute the linear combination of the weights that go to that neuron by the data that comes from the input layer (the algorithm works using as many threads per block as input neurons are). Similar levels of parallelism are achieved on the rest of the used functions, changing the role of each block and each thread on a block.

5 Experimental Analysis

This section reports the experimental evaluation of the proposed ANN approach for recovering historical climate records, studying the classification rate and computational efficiency of the proposed GPU implementation.

5.1 Evaluation platform and problem instances

Evaluation platform. The parallel algorithm was developed in C++, using CUDA, Python (for auxiliary programs), and Fedora 15 (64 bits) OS. The experimental evaluation was performed on an Intel Core i7-2600 3.4 GHz processor with 16 GB RAM DDR3 and Fedora 15 (64 bits), using a GeForce GTX 480 GPU, with 1.5 GB RAM.

Problem instances. Three sets of images were used in the experimental analysis: i) 30 small images (32×30 pixels) randomly generated for debug and validation: 10 images (320 training samples) in the *training list*, and 10 images (320 samples) in both validation sets (*test1* and *test2*); ii) 30 medium-size images (2096×456 pixels), without noise and one active row for each column: 10 images (20960 training samples) in the *training list* and 5 images (10480 validation samples) in both test1 and test2; and iii) 58 pre-processed Digi-Clima images, built by applying filters to eliminate irrelevant data on the original images (size 7086×1496 pixels, JPEG, 400 DPI, 3 color channels, 24 bits) and scaled down to 2096×456 pixels: 28 images (58688 training samples) in the *training list*, and 15 images (31440 validation samples) in both test1 and test2.

5.2 Results and discussion

In order to validate the algorithm, we considered that the rate of correctly classified records for instances never seen by the ANN should be greater than 75%, meaning that we can rebuild the pluviometer graph without losing relevant information, according to our empirical experience from the Digi-Clima project.

The ANN training is performed using all the images on the training set, and then the performance is evaluated using the images from sets test1 and test2 (concluding a cycle). Depending on image size (due to execution times), training and evaluation over all images is performed n times (n epochs) to conclude an execution cycle. The reported values are the average of 5 execution cycles.

Solution quality. In the classification experiments, we first studied the classification rates for small images, in order to later scale the solution. We report the results using the swapping by column strategy, as it computed the best results overall.

The results for 32×30 images are presented in Table 1, using one hidden layer with 5 and 20 neurons, and using two hidden layers, with 50 and 10 hidden neurons each. The results correspond to 100 epochs and using 30 outputs, one for each row in the image. The considered accepted error rate was 0%, meaning that the classification for some column is correct if it exactly matches its expected value.

It is observed that classification rates up to 93.90% are obtained when using 20 neurons in the hidden layer. However, increasing the number of hidden neurons does

not allow achieving better results; despite of representing a more complex function, it is more difficult to learn the correct information. The differences on the classification rates between the parallel and the sequential algorithm is due to the different representation for floating point numbers in CPU and GPU.

Table 1. Classification rates, small images (32×30 pixels)

neurons			*classification rate*	*learning parameters*	
input	hidden	output		learning rate	momentum
			sequential		
90	5	30	74.93%	0.90	0.30
90	**20**	**30**	**93.92%**	**0.80**	**0.30**
90	50+10	30	83.34%	0.90	0.30
			parallel		
90	5	30	68.62%	0.90	0.30
90	**20**	**30**	**93.90%**	**0.80**	**0.30**
90	50+10	30	78.27%	0.70	0.30

We studied different architectures using several numbers of neighboring columns (0, 2, 4, 6, 8). The best classification results (93.43% for the sequential and 92.49% for the parallel implementation) were obtained when using two neighbours, learning rate 0.3, 90 neurons in the input layer, 20 in the hidden layer, and 30 output neurons.

Table 2 reports the results for 2096×456 images (raw and pre-processed) using: one hidden layer with 30 and 90 neurons; two hidden layers, with 90 and 30 neurons each; and 456 (one output per row) and one neuron. The results correspond to 100 and 500 epochs (for the best network architecture). The accepted error rate is 5%.

Table 2. Classification rates, medium-size images (2096×456 pixels)

neurons			*epochs*	*image type*	*classification rate*	*learning parameters*	
input	hidden	output				learning rate	neighbours
				sequential			
1368	30	456	100	raw	36.68%	0.04	3
1368	30	1	100	raw	49.90%	0.05	3
1368	90	456	100	pre-processed	52.91%	0.10	3
1368	90+30	1	100	pre-processed	59.63%	0.20	3
1368	**90+30**	**1**	**500**	**pre-processed**	**74.26%**	**0.20**	**3**
				parallel			
1368	30	456	100	raw	34.98%	0.50	3
1368	30	1	100	raw	50.01%	0.05	3
1368	90	456	100	pre-processed	52.78%	0.10	3
1368	90+30	1	100	pre-processed	61.05%	0.20	3
1368	**90+30**	**1**	**500**	**pre-processed**	**76.23%**	**0.20**	**3**

The results in Table 2 indicate that the best classification rates are obtained when using two hidden layers, with 90 and 30 neurons. When increasing the number of epochs, the classification rate for the algorithm improves. Using a *learning_rate* 0.20, classification rates of 76.23% are achieved using 500 epochs.

We conclude that the proposed approach is valid to address the historical records recovering problem. We found an ANN architecture that uses a single output neuron and obtains classification rates greater than 75% for new instances.

Execution times. Table 3 reports the execution time evaluation of the proposed ANN implementation on GPU, when using different architectures and parameters (number of neighboring columns and neurons in the input/hidden/output layer). The execution times correspond to average and standard deviation computed in 10 independent executions performed for each scenario. The *speedup* indicates the acceleration when using the parallel implementation with respect to the sequential one.

Table 3. Performance evaluation of the proposed ANN on GPU

image size	*input*	*hidden*	*output*	*time sequential(s)*	*time parallel(s)*	*speedup*
2096×456	1368	30	456	4933.01± 12.91	4613.69±7.29	1.06
2096×456	1368	60	456	10523.91±28.95	6285.34±11.14	1.67
2096×456	1368	90	456	12037.62±74.37	7856.80±7.49	1.53
2096×456	1368	90+30	456	16907.92±27.90	11455.66±14.77	1.47
2096×456	1368	30	1	3330.55±7.09	2998.31±9.29	1.11
2096×456	1368	60	1	9081.93±11.46	4214.83±20.72	2.15
2096×456	1368	90	1	10358.51±22.38	5545.01±18.86	1.86
2096×456	1368	90+30	1	10598.11±27.37	5928.30±21.83	1.78

Table 3 shows that the proposed parallel implementation in GPU reduces the training times significantly when compared against a sequential implementation. The best speedup is obtained for 60 neurons in the hidden layer: 1.67 when using one output neuron per row, and 2.15, the best overall, when using only one output neuron. For the architecture that obtained the best classification values (one output neuron, 90+30 hidden neurons), the speedup value is 1.78. These results suggest that the proposed parallel ANN implementation in GPU is an efficient option to speed up the resolution of the historical climate records classification problem.

6 Conclusions and future work

ANNs have proved to be effective for solving many real-world problems. However, ANN are not popular when handling large data sets and/or complex problems, due to the large training times required. Actually, parallel computing using GPUs achieves significant performance improvements over traditional CPU implementations. This article has presented a parallel GPU implementation of an ANN approach to solve the historical climate records classification problem. The proposed approach provides a general method for solving classification/recognition/image processing problems.

The main contributions of this work include the design and implementation of an ANN-based algorithm that obtains accurate classification rates (over 75%) on reasonable execution times. This method provides a new alternative for recovering historical climate data and it allows comparing against the image-processing algorithm developed in the Digi-Clima project. The proposed parallel strategy is based on executing multiple threads on a GPU, each one working with multiple neurons, trying to keep threads independent from each other. Each kernel function was designed to take advantage of the GPU platform (e.g., certain kernels are assigned to compute over more than one neuron, to avoid the thread creation overhead). Furthermore, the GPU shared memory was used in order to avoid the latency on the global memory access.

The experimental analysis shows that reasonable accurate classification results are obtained, and that the GPU implementation allows reducing the execution times when compared to a sequential implementation. A speedup of 2.15 is obtained when using an ANN with 1368 inputs, 60 hidden neurons and a single output. The architecture that obtained the best classification rates has a speedup of 1.78.

The main lines for future work are focused on improving the efficacy and efficiency of the presented algorithm and also tackle other classification/image processing problems using parallel ANN over GPUs. Regarding the first line of work, better execution times could be achieved if kernel invocation parameters are better adjusted to avoid problems such as thread divergence, or if a better GPU resource management is made (in the use of shared memory for example) for instances of considerable size. In addition, certain constants (such as *momentum* or *learning_rate*) could be auto-tuned by the algorithm to obtain the best classification rates. Regarding the second line of work, the developed algorithm could be used for classification of similar types of images such as images produced by electrocardiograms, seismometers, etc.

References

1. Nesmachnow, S., Usera, G., Brasileiro, F.: Digi-Clima Grid: procesamiento de imágenes y computación distribuida para la recuperación de datos del acervo climático. IEEE Computing Latin America (2014)
2. Usera, G., Nesmachnow, S., Brasileiro, F., Da Silva, M., García S.: Recovering historical climate records through grid computing. In: Latin American eScience Workshop 2013, São Paulo, Brazil (2013)
3. Nesmachnow, S., García, S., Usera, G., Brasileiro, F.: Grid computing para la recuperación de datos climáticos. In: VI Conferencia Latinoamericana de Computación de Alto Rendimiento, San José, Costa Rica (2013)
4. Nesmachnow, S., Da Silva, M.: Semi-automatic historical climate data recovering using a distributed volunteer grid infrastructure. In: 5th International Supercomputing Conference, Ensenada, México (2014)
5. García-Herrera, R., Können, G., Wheeler, D., Prieto, M., Jones, P., Koek, F.: CLIWOC: A climatological database for the World's oceans 1750–1854. Climatic Change, 73, pp. 1–12 (2005)
6. Wilkinson, C., Woodruff, S., Brohan, P., Claesson, S., Freeman, E., Lubker, S., Marzin, C., Wheeler, D.: Recovery of logbooks and international marine data: the RECLAIM Project. Int. J. Climatology 31(7), 968–979 (2011)
7. Old weather project, `http://www.oldweather.org`, accessed May 2014.
8. Mitchell, T.: Machine Learning. McGraw Hill, New York (1997)
9. Glorot, X., Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. Journal of Machine Learning Research 9, 249–256 (2010)
10. Wilt, N.: The CUDA Handbook: A Comprehensive Guide to GPU Programming. Addison-Wesley Professional, Boston (2013)
11. Balarini, J., Rodríguez, M., Nesmachnow, S.: Facial Recognition Using Neural Networks over GPGPU. CLEI Electronic Journal 15(3), 1-12 (2012)
12. Bishop, C.: Neural Networks for Pattern Recognition.Clarendon Press, Oxford. (1995).