B The TarzaNN Neural Network Simulator

A number of computational models of visual attention exist, but making comparisons is difficult due to the incompatible implementations and levels at which the simulations are conducted. To address this issue, we have developed a general-purpose neural network simulator that allows all of these models to be implemented in a unified framework. The simulator allows for the distributed execution of models, in a heterogeneous environment. Graphical tools are provided for the development of models by non-programmers and a common model description format facilitates the exchange of models. In this paper we will present the design of the simulator and results that demonstrate its generality.

B.1 Introduction

Even though attention is a pervasive phenomenon in primate vision, surprisingly little agreement exists on its definition, role and mechanisms, due at least in part to the wide variety of investigative methods. As elsewhere in neuroscience, computational modelling has an important role to play by being the only technique that can bridge the gap between these methods [Wilson, 1999a] and provide answers to questions that are beyond the reach of current direct investigative methods.

A number of computational models of primate visual attention have appeared over the past two decades (see [Rothenstein and Tsotsos, 2008a] for a review). While all models share several fundamental assumptions, each is based on a unique hypothesis and method. Each seems to provide a satisfactory explanation for several experimental observations. However, a detailed comparative analysis of the existing models, that is, a comparison with each of the models subjected to the same input data set in order to both verify the published performance and to push the models to their limits, has never been undertaken. Such an analysis would be invaluable: comparative, computational testing procedures would be established for the first time, successful modelling ideas would be confirmed,

weaknesses identified, and new directions for development discovered. The goal would be to validate the models by testing them against existing knowledge of the primate attentional system; the experimental stimuli and task definitions which led to that knowledge would form the basis for the development of the test data sets.

In order to facilitate this analysis and to provide the research community with a common software platform, we have developed a general purpose, extensible, neural network simulator geared towards the computational modelling of visual attention. The simulator allows for the distributed execution of models in a heterogeneous environment. Its associated tools allow non-programmers to develop and test computational models, and a common model description format facilitates the exchange of models between research groups. The simulation results can be presented in a variety of formats, from activation maps to the equivalent of single-unit recordings and fMRI.

This paper starts with a discussion of the design of the simulator, from the perspective of the requirements imposed by existing computational models of visual attention. This is followed by a review of some of the hardware and performance issues that were taken into account. The paper continues with a description of the simulator user interface and the modelling tools that accompany it and concludes with a discussion of the system and the current state of the project, including preliminary results, items considered for future development, and a comparison with other neural network simulators.

B.2 Simulator requirements and design

The wide variety of computational models of visual attention creates unique challenges for a unified framework. For example, computer science inspired models use simple, generic McCulloch-Pitts [McCulloch and Pitts, 1943] type neurons (e.g. [Tsotsos et al., 1995, Itti et al., 1998]), sometimes in combination with specialized neurons (e.g. gating neurons in [Tsotsos et al., 1995]), in large-scale simulations; others use neurons modeled by realistic differential equations in minimalist, "proof of concept" networks (e.g. [Reynolds et al., 1999]) or large-scale systems with spike density functions (e.g. [Rolls and Deco, 2002]) or spiking (e.g. [Lee et al., 2003]).

Object-oriented techniques have been applied to the specification of the various categories of neurons in the system in a very flexible and easily extendable framework. Currently, the system provides simple McCulloch-Pitts type neurons

and two categories modeled by differential equations: Wilson-Cowan for spike density functions and Hodgkin-Huxley for spiking neurons [Wilson, 1999b]. New types of neurons can be added by simply subclassing the existing, high-level models and specializing the behaviour as desired. In particular, new kinds of neurons modeled by differential equations can be added by creating a class that describes their equations.

All large-scale computational models of visual attention share the notion of a field of neurons with identical properties and connectivity. For the rest of the paper we will refer to these fields as "feature planes." Thus, a model consists of a number of feature planes, interconnected by filters that define the receptive field properties of the neurons in the plane. This division of the models into feature planes and filters allows for very efficient implementation of the core of the simulator as a convolution engine. Different models and levels of simulation require distinct methods of communication and synchronization between feature planes. Three such methods have been implemented: lock step, asynchronous and synchronous. The lock step method corresponds to traditional neural network behaviour, where the various components perform computational steps according to an external clock determined by the slowest component. The asynchronous method allows each computation to be performed at its own pace, without any coordination except, of course, for the locking needed to ensure that data transferred between feature planes is in a consistent state. This method is the closest we can come to a biological, decentralized system, and it is the most appropriate for the more realistic simulations that use neurons defined by differential equations. If there are significant speed differences between feature planes, this method means that the fastest one will perform several steps on the same data. To handle this scenario, we have introduced the synchronous communication method, which is identical to the previous one, except for the fact that feature planes notify their dependents when new results are available and the dependents will not perform computations unless notified.

Selective Tuning is implemented in a seamless way, by simply specifying an attribute in the feature plane definition. Simple attentional tasks can be defined by specifying the relative weighting with which the different feature planes participate in the WTA competition. It is important to note that while previous implementations of ST were tightly integrated into specialized networks, this is the first generic implementation, that can be applied to any model network by simply specifying which feature planes should participate and with what thresholds.

For the purposes of this dissertation, the simulator was enhanced to perform

supervised learning. Learning occurs on Datasets, which contain lists of input images and the corresponding desired network output. When the learning algorithm is stopped after reaching the desired output error threshold, the network structure is saved for later use. In testing mode, the network structure is loaded from the saved file, and the system functions normally, with all the regular features.

Due to the wide variety of computing platforms available, portability was a major concern in the design and implementation of the simulator. While portability alone would seem to suggest Java as the programming language of choice, performance considerations make it impractical. The code is written in ANSI C++, and the only external requirement is the highly portable Qt package from Trolltech. The simulator was developed and tested on Mac OS X, Windows, Linux and Solaris, and in a heterogeneous cluster composed of computers running various versions of these operating systems.

Given the stated goal of creating a system to be used by many research groups, it was natural to adopt an open source development process. The whole project is available on the Internet (`http://www.tarzaNN.org`), and researchers can participate in the development process either directly, by submitting code through CVS, which is used for source control (`http://www.cvshome.org/`), or by requesting features and reporting bugs through Bugzilla (`http://www.bugzilla.org/`) and its web interface. Documentation, also available on the project web site, is generated automatically using doxygen (`http://www.doxygen.org/`).

B.3 Performance considerations

Performance is key to the success of any simulator, and in this case performance has two aspects. First and most obvious, model execution has to be as close as possible to real time, especially in the computationally very expensive case of differential equation models. A second, and by no means less important consideration is the speed and ease with which models can be developed and modified (see Section B.4).

The structuring of the models in feature planes connected through filters made convolutions the main computation in the system, and this allowed us to apply classical techniques from image processing to speed up the calculations. Here we will mention only two of these techniques. Filters are analyzed using linear algebra and, if possible, separated using singular value decomposition (SVD), which transforms the two dimensional convolution into a sequence of two one dimensional convolutions. In cases where decomposition is not possible, the designer

has the option of defining the filter as a linear composition of simpler filters that are applied in parallel, increasing the chances that the simpler filters are candidates for the SVD algorithm. An example of this is the difference-of-Gaussians filter, which is not decomposable, but is a linear combination of decomposable filters. The system also exploits the vector arithmetic units built into PowerPC chips.

All complex object-oriented software systems have to address the performance problems related to the use of virtual functions and the fact that they impose late binding (i.e. the code to be executed has to be determined at run time). Due to the fact that we require particular flexibility in the combination of neuron types and synchronization strategies, this issue has been addressed early in the design of our simulator. To alleviate this problem, policy-based design relying on C++ templates [Alexandrescu, 2001] was used extensively, allowing us to assemble complex behaviour in the feature plane classes out of many simple, specialized classes without the overhead of the classical object oriented approach.

One important observation is that any visual attention model comprises of a number of clusters of high connectivity with relatively sparse connectivity between clusters. Generally, the clusters correspond to the visual areas in the primate brain with dense local interconnections in the form of inhibitory pools, winner-take-all and center-surround circuits. This structure makes it possible to distribute the computation across a group of computers. Feature planes are represented on remote machines by proxy objects that handle the communication in its entirety, making the whole process completely transparent.

B.4 Tooling and interfaces

Two of the key requirements for the simulator are accessibility to non-programmers and support for collaborative research, by allowing easy exchange of models between groups. To facilitate this, models are described by using a common description format based on the XML language. A tool is being developed to graphically describe the models, allowing researchers who are not familiar with programming to use the system. XML files are automatically generated. The graphical designer has a look and feel that is very similar to that of existing tools for drawing diagrams, with typical drag-and-drop toolbars for the definition of feature planes and their interconnections. Double-clicking on the feature planes opens the properties dialog, where users can specify their size, type of neurons they contain and their parameters, and other characteristics.

Similarly, in the link properties dialog users can determine the size, type, and parameters of filters. The simulator user interface presents the results of the computations in three formats. The default view presents each feature plane as an image, with shades of gray corresponding to local activations. The main window of the application is presented in Fig. B.1, with the input image and a number of activation maps being displayed. The time-course view presents the temporal evolution of the output of individual neurons within a feature plane. Depending on the nature of the model neuron, these can correspond to spike density functions or action potentials (for Wilson-Cowan or Hodgkin-Huxley neurons, respectively [Wilson, 1999b]). In Fig. B.2, we present two spike density functions, corresponding to two neurons in a competitive network. The neuron represented in red corresponds to the winning neuron, with a characteristic response profile, while the green trace corresponds to a neuron that is inhibited by the proximity of the winner. Finally, the fMRI view presents a comparison between the current state of one or more feature planes and a snapshot taken during a different run. This comparison can be performed off-line, between two saved snapshots of activity, or in real time, between a saved snapshot and the currently executing network. The fMRI view integrating activations across all feature planes corresponding to a brain area should closely match observations made in human subjects. Fig. B.3 presents the static comparison between two activations (left hand side of the image), corresponding to the response of model non-Fourier (second order) V4 concentric units [Wilson, 1999a] to a Glass pattern [Glass, 1969] vs. a random dot pattern, with the characteristic colour scheme. A threshold slider is available at the bottom of the window, this can be used to eliminate statistically insignificant image components. Note that the feature planes communicate with the various views through an observer/notification scheme, so it is very easy to add custom views if a specific application needs them. For an example of a large-scale system that was implemented in TarzaNN, see [Zaharescu et al., 2005]. The simulator accepts input images in a wide variety of file formats, or directly from a camera, and includes mechanisms to control the motion of the camera, if applicable.

B.5 Discussion and conclusions

A number of computational models of primate visual attention have appeared over the past two decades, but due to the different initial assumptions and requirements, each modelling effort has started from scratch, and the designs reflect these requirements and the particular software and hardware present in that par-

ticular lab.

In this paper we presented the design principles behind a general purpose neural network simulator specifically aimed at computational modelling of visual attention and we discussed hardware and performance issues and how they influenced design and implementation. We also presented the user interface, with the many ways in which simulation results can be presented to the user. The tools that make the simulator accessible to non-programmers and allow for collaborative research, by facilitating the exchange of models between groups, were also introduced.

The flexibility of the simulator and the XML-based model description have allowed us to very quickly obtain implementations of the three major computational models of visual attention ([Rolls and Deco, 2002, Tsotsos et al., 1995, Itti et al., 1998]). Fig. B.4 (details extracted from Fig. B.1) corresponds to part of Fig. 3 in [Itti et al., 1998]. Fig. B.4-left represents the input image, while Fig. B.4- right is the activation of the saliency map feature plane. The largest of these models is a subset of the Selective Tuning Model for motion [Tsotsos et al., 2005], which consists of 210 feature planes organized in 4 areas (the full model will include 620 feature planes for the feedforward network, plus at least that number for the winner-take all circuits). Note that the applicability of the simulator is not limited to modelling visual attention, see for example Fig. B.5, where we have reproduced the ability of model non-Fourier (second order) V4 concentric units proposed in [Wilson, 1999a] to detect Glass patterns [Glass, 1969]. In terms of performance, at the current level of optimization, on a dual processor 2.0Ghz PowerMac G5, compiled using gcc version 3.3, a sub-network composed of 3 feature planes of Wilson-Cowan neurons described by three differential equations each, connected through 5 filters, performs one step in 40ms - two 128 by 128 and one 20 by 20 feature planes, or 33,168 neurons, which means roughly 800,000 neuron updates per second.

TarzaNN distinguishes itself from the other neural network simulators available by uniquely bringing together a series of characteristics. Portability is extremely important for collaborative research, and many simulators are limited to either Windows or one or more Unix platforms – e.g. SNNS (`http://www-ra.informatik.uni-tuebingen.de/SNNS`), Genesis (`http://www.genesis-sim.org/GENESIS`), iNVT (`http://ilab.usc.edu/toolkit/`), etc. Also, many simulators have built-in limitations that make them inappropriate for the simulation of some visual attention models, by limiting neuron types (e.g. Amygdala (`http://amygdala.sourceforge.net/`), PDP++ (`http://www.cnbc.cmu.edu/Resources/PDP++`) – or network config-

urations – e.g. iNVT (http://ilab.usc.edu/toolkit/), NeuralWorks (http://www.neuralware.com/). General-purpose neural network simulators – e.g. Neural Network Toolbox for Matlab (http://www.mathworks.com/products/neuralnet/) require programming skills for anything but the most common architectures, and performance is limited by the lack of facilities for distributed computations. Usually there is a trade-off between flexibility and the ability to design large-scale networks.

The main original contributions of the work presented in this paper are the novel object oriented framework for defining neuron properties, the configurable methods of communication and synchronization between layers and the standard-based method of describing networks. The neuron framework allows users to define model neurons at any level of complexity desired, and even test the behaviour of the same network with different models. The synchronization and communication infrastructure makes it possible to simulate both traditional computer science neural networks and biologically plausible models. The XML based network description is an important contribution in at least two ways: it allows for easy interchange of models between researchers and it makes the automatic generation of regular structures possible without a need for programming skills.

Figure B.1: The TarzaNN main window, presenting activation maps for the feature planes used in simulating the results presented by Itti et al. [Itti et al., 1998]. In their model, feature map activity is combined at each location, giving rise to a topographic saliency map. A winner-take-all network detects the most salient location. The top-left corner is the input image and the bottom-right the saliency map. The other sub-windows represent (left to right, top to bottom) activation maps for center-surround feature planes for: the image luminance, the red, the green and the blue colour channels, vertical, 45 degrees left, horizontal, and 45 degrees right orientations. The saliency map is a linear combination of the center-surround feature maps.

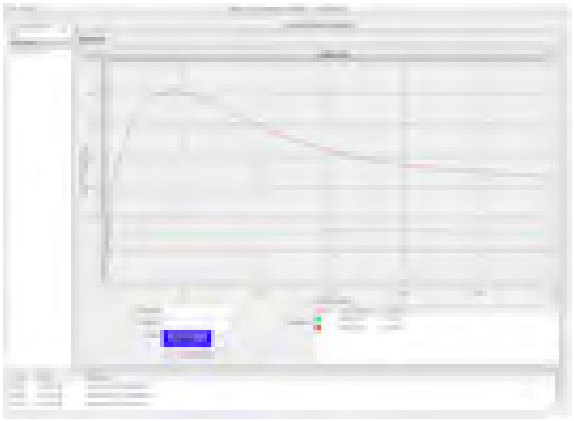

Figure B.2: Spike density functions for two Wilson-Cowan neurons [Wilson, 1999b] involved in a competitive interaction (mutual inhibition through an interneuron, not represented here). The red trace corresponds to the winning neuron, and shows the characteristic response of a neuron, with the high stimulus onset response, followed by the longer adaptation period. The green trace shows the response of a neuron that was inhibited as a result of the competition.

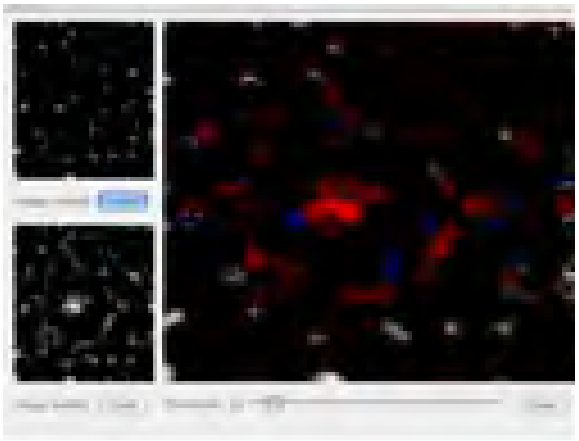

Figure B.3: Static functional MRI view corresponding to the difference between the two activation maps on the left. The bottom activation map is the response of non-Fourier V4 concentric units [Wilson, 1999a] to a Glass pattern (see Figure B.5), while the top is the response of the same feature map to a random dot pattern. The central red area in the fMRI view represents the higher activation corresponding to the detected concentric pattern, while the other red and blue areas represent stronger responses to the Glass pattern and the random pattern, respectively. These weaker secondary areas can be eliminated by adjusting the threshold at the bottom of the figure.

Figure B.4: Left – input image, Right – saliency map. This model reproduces the results presented by Itti et al. (Figure 3 in [Itti et al., 1998]). The model identifies the phone pole as the most salient area of the image, followed by the traffic sign.

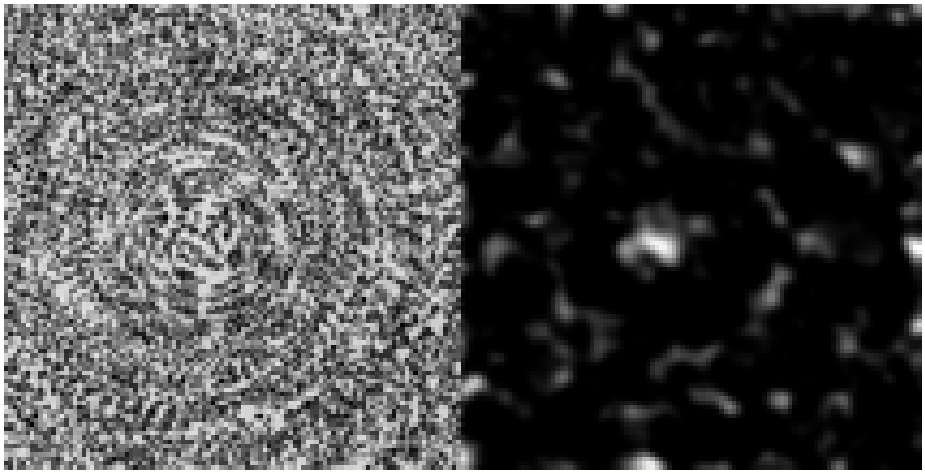

Figure B.5: Left – input Glass pattern, Right – activation map of V4 concentric unit feature plane. This network implements the non-Fourier (second order) processing presented by Wilson (Figure 6 in [Wilson, 1999a]). As can be observed in the activation map, the highest activation corresponds to the V4 concentric units corresponding to the center of the illusory circles. See also Figure 2, for a comparison between the network's response to the Glass pattern vs. a random dot pattern.