TRITA-CSC-CB 2012:02 | ISRN KTH/CSC/CB--12/02--SE | ISSN 1653-5707

TECHNICAL REPORT
Event-based Sensor Interface for Supercomputer scale Neural Networks

Erik M. Rehn

Stockholm 2012
CB – Computational Biology
School of Computer Sciense and Communication
KTH Royal Institute of Technology

Erik M. Rehn
CB – Computational Biology

Event-based Sensor Interface for

Supercomputer scale Neural Networks

Publicationsdate: June 2012
E-mail to author:
TRITA-CSC-CB 2012:02
ISRN KTH/CSC/CB--12/02--SE
ISSN 1653-5707

NADA is a co-operating departement between the
Royal Institute of Technology and Stockholm University

Reports can be ordered from:

Research Group:
CB – Computational Biology
URL: www.kth.se/csc/forskning/cb

School of Computer Science and Communication
KTH (Royal Institute of Technology)
SE- 100 44 Stockhom, Sweden

URL: www.kth.se/csc

Event-based Sensor Interface for Supercomputer scale Neural Networks

Erik M. Rehn
Bernstein Center for Computational Neuroscience
Berlin, Germany

Supervisors: Simon Benjaminsson, Anders Lansner
Department of Computational Biology
KTH Royal Institute of Technology
Stockholm, Sweden

1. Introduction

One challenge in the scientific endeavor to artificially reproduce the behavior of large biological neural systems is our lack of enough computational power. How can we create an artificial neural system that is in anyway comparable in size to a real human brain? Today there exist two approaches to this problem, supercomputers and neuromorphic hardware, both showing promising future possibilities. The next generation of exascale supercomputers, prognosticated to be available in a decade or so, are by some estimated to have the capability to simulate a complete human brain down to the cellular level [1] and numerous projects exist to develop highly scalable hardware tailored for neural computation [2, 3]. However, given that these efforts produce what they promise and that we one day have sufficient computational power to simulate a neural system in the scale of the human brain a new need arise. To be able to act intelligently in the world an agent needs senses to perceive it. The action-perception loop has to be closed and some kind of physical sensory-motor system has to be developed.

Here we present an investigation of how to build a software interface between a relatively large neural network model running on a supercomputer and a so called silicon retina image sensor [4]. The goal is to construct a framework that enables neural simulations to be fed with input from event-based sensors in real time. Event-based sensors are in general well suited for generating input to artificial neural systems due to the resemblance between their event coding and spikes in biological systems.

We demonstrate how to use the event input from the image sensor to complete patterns in an autoassociative recurrent attractor neural network running in a highly parallelized environment. The recurrent connections are trained with the Bayesian Confidence Propagation Neural Networks (BCPNN) learning rule on images representing simple geometric shapes and the sensor input is then used to activate these network states one at a time.

2. Technical infrastructure

In the following sections a brief overview of the technical infrastructure upon which our implementation is built is described. The information from the image sensor has to travel through multiple steps on its way to the simulation environment and these will be described, starting from the image sensor. On the hardware level there are three steps; the sensor, a standard PC working as a USB-host for the sensor and a supercomputer. The supercomputer used is KTH's "Lindgren" Cray XE6, having 35592 2.1 GHz processor cores with a theoretical peak performance of 305 TeraFlops. In this study only a small fraction of these cores are utilized. Significant effort has been put into optimizing the simulator as much as possible to reduce the number of needed cores. Several different layers of software have also been used which will be described below.

DVS Silicon Retina

In the last 20 years several attempts have been made to construct event-based image sensors with different degree of similarity to biological retinas [5]. The main difference between these so called silicon retinas and a normal digital camera is that instead of sending whole frames at fixed frame rates, they only respond to changes in the input image represented as events. Using this address-event representation (AER), where the address codes for pixel location and illuminance change, the redundancy in transmitted image data is greatly decreased. The temporal resolution and latency achieved by an event-based image sensor can also be very high compared to traditional cameras due to the asynchronous nature of the event coding. Pixel changes are sampled and transmitted as they happen instead of at fixed frame rates. Silicon retinas have therefore been applied successfully to problems where a high speed camera would normally be needed [4].

In this project we use a Dynamic Vision Sensor (DVS) Silicon Retina developed at the Institute of Neuroinformatics at the University of Zurich [4]. It has a 128x128 pixel resolution (DVS128) and communicates with a desktop computer through a standard USB-interface. It solely relies on temporal contrast, meaning that unlike the vertebrate visual system, which employs spatial edge detection and spatial contrast adaptation, it only detects local illuminance changes at single pixels. This has the effect that a static scene will not produce any output events. The sensor has either to be moved or pointed at a moving scene to generate any output other than noise.

The events produces by the DVS128 are either “on” or ”off” which signals an increase or decrease in pixel illuminance respectively. This gives a total address-event space of size 32768 (128x128x2). The temporal resolution of the events is in the order of 10th of µs. In total an event is represented as an integer (0-32767) and a microsecond timestamp. The illuminance change sensitivity is adjustable but under normal configuration the number of generated events per second varies between about ten for a static scene (only noise) and tens of thousands for a highly dynamic scene.

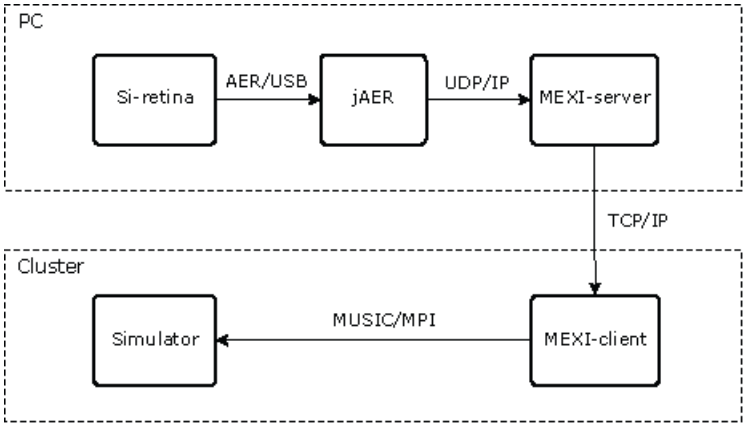

Figure 1. Overview of the technical infrastructure.

jAER

The Java Address-Event Representation software (jAER) is an open-source desktop application developed to process event-based sensor data [6]. It supports numerous hardware devices, where the DVS128 silicon retina is one of them, and provides tools for working with AER data in real-time, e.g. filters, calibration interfaces, etc. Its role in our setup is its ability to receive events from the USB-interface of the sensor and then forward them as a stream of UDP datagrams. Using jAER as a middleware relieves us from the cumbersome work of handling the USB connection directly.

Figure 2. Image taken with jAER while the DVS Silicon retina is moved in front of a photo. On-events are white while off-events are black. jAER has an maximum update rate of 120 Hz which means that the image is a sum of all events generate during a 8.33 ms interval.

MUSIC

The multi-simulation coordinator (MUSIC) is a software framework developed at the KTH Royal Institute of Technology and is today maintained by the INCF [7]. It provides an interface between different neural simulators running in a parallel environment. The standard message passing interface (MPI) is used to send events/spikes and continuous data between parallel applications in a way that enable different simulations to run at different speed and with different time step size. The timing features of MUSIC comes well in hand when events from the silicon retina is generated in real-time while the simulator consuming the events runs slower or faster than real-time. MUSIC makes sure that events sent from one end of a communication port arrive at the other end at the right local time and that event addresses are mapped correctly between processes.

MEXI

The MUSIC framework uses MPI for communication and is mainly meant to run on a single host or a computational cluster. To be able to send events to a MUSIC enabled application from a separate host the MEXI software has been developed. It can be seen as a network adapter for MUSIC and uses TCP/IP for communication. MEXI has a server-client architecture where the server sends events to the client. In our setup (see figure 1) the client runs on Lindgren and forwards incoming events to the simulator through MUSIC. The server works like an event proxy and runs on *nix systems. It receives events from jAER sent as UDP datagrams and forwards them to the MEXI client.

The reason for using MEXI and not directly sending the events from jAER to the simulator is that by using MEXI we gain the timing functionally of MUSIC. The connection scheme of MEXI, with the client as the receiver and the server as the sender, is also suitable for cluster environments where the cluster nodes are often behind NAT. This is the case for Lindgren and it is not possible to directly send UDP packets to a cluster node from the outside.

ANSCore

ANSCore is a high performance neural simulator developed by the Computational Biology group at KTH. It has previously been used to simulate a model composed by 65.5 million neural units and 393.2 billion synapses on the JUGENE 1 PetaFlops machine [8]. It is written in C++ and uses MPI for message passing. It mainly supports graded units even though some support for integrate-and-fire neurons exists.

To enable ANSCore to consume events from the silicon retina it has been integrated with MUSIC and a lot of effort has been put into optimizing its inner workings to make the model described below need as few processor cores as possible.

3. Network model

To investigate how events from the silicon retina can be used as input to a simulated neural network a proof of concept network model has been developed. It is a simple two-layer model consisting of an input layer and an autoassociative layer of binary units. Both layers have the same number of units as the silicon retina pixel resolution and the sensor input is mapped retinotopically onto layer 1, the input layer. The input layer has a one-to-one connectivity to the autoassociative layer. The autoassociative layer is trained on simple spatial patterns and learns to complete noisy or incomplete versions of these. The patterns are two dimensional binary images depicting simple geometric shapes represented as binary vectors of length equal to the silicon retina resolution. The purpose of the model is to be able to recognize and complete shapes in the noisy camera input.

One important limitation of the model is that it does not use spiking units. Instead it consists of thresholded graded units with binary output. The on- and off-events from the silicon retina is treated the same and a unit of the input layer will be activated (with output 1) as long as it receives at least one event during a simulation time step. This might seem to destroy the purpose of using event-based input but we are here mainly interested in demonstrating the possibility to use event-based sensors in combination with large neural simulations in real time and not in producing a very interesting model. The same technical infrastructure as described above could easily be used together with a model that utilized spike timings or firing rates.

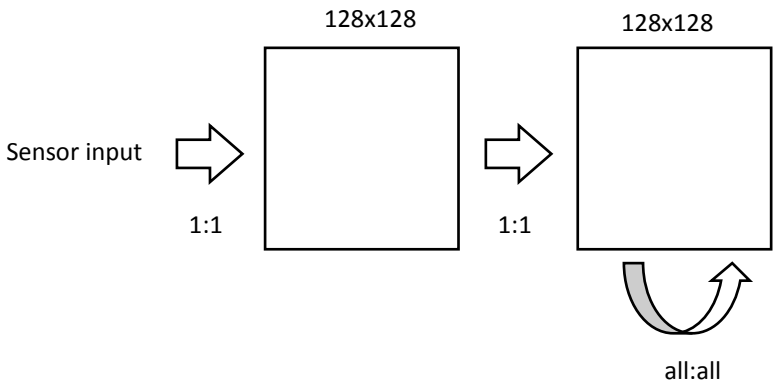

Figure 3. Schematic of the network structure. The sensor input has a retinotopic mapping onto the input layer (1:1). The mapping is preserved as the input is forwarded to the second autoassociative layer which is recurrently connected to itself. Both layers have a size of 128x128 units, same as the silicon retina resolution. The recurrent connectivity of the autoassociative layer consists of up to about $2.6 * 10^8$ synapses.

BCPNN

Incremental recurrent Bayesian Confidence Propagation Neural Networks (BCPNN) is an attractor network model which learns patterns of activity in correspondence to Hebbian cell assemblies. Our model uses a variant of the BCPNN learning rule to train the autoassociative layer. BCPNN is derived from the Naïve Bayesian Classifier and a simplified version of its formulation will be outlined here. For a derivation and deeper discussion of BCPNN see [9].

BCPNNs are normally modular with a structure inspired by the columnar organization of the neocortex. BCPNN units can be thought of as cortical minicolumns grouped in so called hypercolumns. The central idea is that the activity of a hypercolumn encodes the value of a continuous attribute as a discrete probability distribution. The attribute value is split in intervals and the activity of each child minicolumn then represents the probability that the value lies in a certain interval. Let n be the number of hypercolumns, M_i the number of minicolumns (or units) in hypercolumn i and X_i a set of attributes taking the discrete values $x_{ii'}$, then in a recurrent BCPNN the activity of a unit $h_{jj'}$ (unit j' of hypercolumn j) at time t is:

$$h_{jj'}(t) = f\left(\beta_{jj'} + \sum_{i=1}^{n} log\left[\sum_{i'=1}^{M_i} w_{ii'jj'} * h_{ii'}(t-1)\right]\right) \tag{3.1}$$

where $\beta_{jj'}$ is the bias of unit jj' and $w_{ii'jj'}$ the weight between unit jj' and unit ii'. Because the activity of a whole hypercolumn represents a probability distribution, $P_{X_i}(x_{ii'})$, the function $f(z)$ is needed to normalize the total activity to 1.

For our model we drop the modular structure and let each hypercolumn consist of only one unit which is either on or off depending on a threshold γ. The normalization function is then defined as

$$f(z) = \begin{cases} 1, & z \geq \gamma \\ 0, otherwise \end{cases} \tag{3.2}$$

This gives a simplified recurrent network of binary units without any hypercolumns but which still has a formulation consistent with BCPNN theory. We omit the double indices and end up with the following activation function for unit j at time step t:

$$h_j(t) = f\left(\beta_j + \sum_{i \in K} \log(w_{ij} * h_i(t-1))\right) \tag{3.3}$$

where K is the number of presynaptic units synapsing onto unit j. This thresholding also has another desirable effect on the network dynamics, other than making the output of the autoassociative layer binary; only units which receive substantial recurrent input are going to generate output. Because the autoassociative layer is fully recurrently connected to itself only input patterns with high enough similarity to one of the trained patterns will be completed. The threshold makes sure that the network only moves towards a learned state of attraction if input overlaps to a great extend with that state. The value chosen for γ depends on the size of the trained patterns and how easily the network should be attracted by one of the training patterns.

The BCPNN uses a Hebbian type learning rule which is based on the unit activation probabilities and can for our simplified network model be formulated as

$$w_{ij} = \frac{P(x_j, x_i)}{P(x_j) * P(x_i)} \tag{3.4}$$

$$\beta_j = \log(P(x_j)) \tag{3.5}$$

where $P(x_j, x_i)$ is the joint probability of the post- and presynaptic units being active together, $P(x_j)$ the probability of the postsynaptic unit being active and $P(x_i)$ the probability of the presynaptic units being active.

To let BCPNN:s learn incrementally all probabilities are calculated online using exponential smoothed running averages. The updates at time step t is then

$$\frac{dP(x_i)(t)}{dt} = \alpha\big([(1-\lambda_0)h_i(t) + \lambda_0] - P(x_i)(t)\big) \tag{3.6}$$

$$\frac{dP(x_j, x_i)(t)}{dt} = \alpha\left(\left[(1-\lambda_0)h_j(t)h_i(t) + \lambda_0\right] - P(x_j, x_i)(t)\right) \tag{3.7}$$

where α is the learning rate and λ_0 is a small intrinsic background activation probability needed to prevent the probabilities from becoming zero. For a pair of units which never experience any activation $P(x_i)$ and $P(x_j)$ will converge to λ_0, and $P(x_j, x_i)$ to λ_0^2, producing a weight $w_{ij} = 1$.

The process of training a BCPNN, as stated above, is a matter of estimating the activation probabilities of each unit, $P(x_i)$, and the coactivation probability between all units, $P(x_i, x_j)$. These are then used to calculate the weights, w_{ij}, and the bias, β_j, of each unit. This is done online for every time step using the running averages in equation 3.6 and 3.7. One should notice that this, the standard BCPNN learning rule, produces both exhibitory ($w_j > 1$) and inhibitory ($w_j < 1$) connections and a symmetric connection matrix. This has turned out to be problematic for the performance of our model. Units which are never activated during the training, because they are not part of any of the training patterns, will have inhibitory connections to units which are part of one or more patterns, with weight

$$w_{ij} = \frac{\lambda_0^2}{\lambda_0 * P(x_j)} < 1 \tag{3.8}$$

where $P(x_j) > \lambda_0$. These inhibitory connections will make attracted states very sensitive to input in regions which have not seen any input during the training phase, making it very hard to stay in an attracted state with the noisy silicon retina input. Noise or signals that are present in untrained parts of the visual field will suppress activation of the trained patterns, producing a model which is very sensitive to background noise and imperfect input patterns. Our wish is instead that an activated pattern should stay activated, by sustained recurrent reverberation, until the input forces the network into another trained state. Input in parts of the visual field which have not seen any activation during training should not influence the activity of units which are parts of a pattern. Fortunately this can be achieved by introducing a simple condition for when weights should be updated during the training.

$$w_{ij} = \begin{cases} \log\left(\dfrac{P(x_j, x_i)}{P(x_j) * P(x_i)}\right), & if\ P(x_j) > \lambda_0 \\ 1, & otherwise \end{cases} \tag{3.9}$$

This rule will produces a non-symmetric weight matrix where units which are part of one or more patterns inhibit all units which are not part of the same patterns, while units which are not part of any pattern will not influence other units.

Feed-forward input

Other than the recurrent connections the units of the autoassociative layer also have connections to the input layer with a one-to-one connectivity. We therefore let unit j in the input layer influence the activity of unit j in the autoassociative layer in the following way:

$$h_j(t) = \begin{cases} 1, & if\ b_j(t-1) = 1 \\ f\left(\beta_j + \sum_{i \in K} \log(w_{ij} * h_i(t-1))\right), & otherwise \end{cases} \quad (3.10)$$

where $b_j(t-1)$ is the binary input to unit j in the autoassociative layer from the corresponding unit in the input layer. This implies that the unit output is 1 if the value of the activation function is above a certain threshold γ or if it gets feed-forward input from below.

4. Experiments

Two experiments are here demonstrated to show how the developed proof of concept network model performs and how it is able to use the silicon retina as input. For each experiment the network goes through a training and a test phase. During the training phase a number of training patterns are presented to the input layer and the recurrent connections in the autoassociative layer are disabled while the incremental weight update for these are enabled. After the training the network goes into the test phase where distorted versions of the training patterns or events from the silicon retina are used as input. During the test phase the weight update is disabled.

Pattern completion

To demonstrate the pattern completion capabilities of the network an initial experiment is conducted without any input form the silicon retina. The network is trained on three different patterns depicting simple shapes. During the training phase each pattern is presented 10 times. Because the weight update is based on estimating the activation probabilities using running averages it is important the training patterns are not shown one at a time 10 times in a row. Patterns presented further back in time would then have a weaker memory imprint than those presented last. The pattern presentation is therefore interlaced, i.e. the first pattern is presented once, then the second once and then the third once. This is then repeated 10 times.

After the training all weights are locked, the recurrent connections are enabled and the simulation enters the test phase. Noisy versions of the training patterns are then produced by randomly removing about 60% of the pixels belonging to the pattern. Some background noise is also added to every test pattern by randomly enabling 1% of the pixels in the background. This background noise level is somewhat comparable to the amount of noise seen in the silicon retina data.

In figure 4 the result of experiment is presented. As one can see the network gracefully moves from one state to the next as new patterns arrive at the input layer, new patterns inhibit old patterns which lack support by the feed-forward activity.

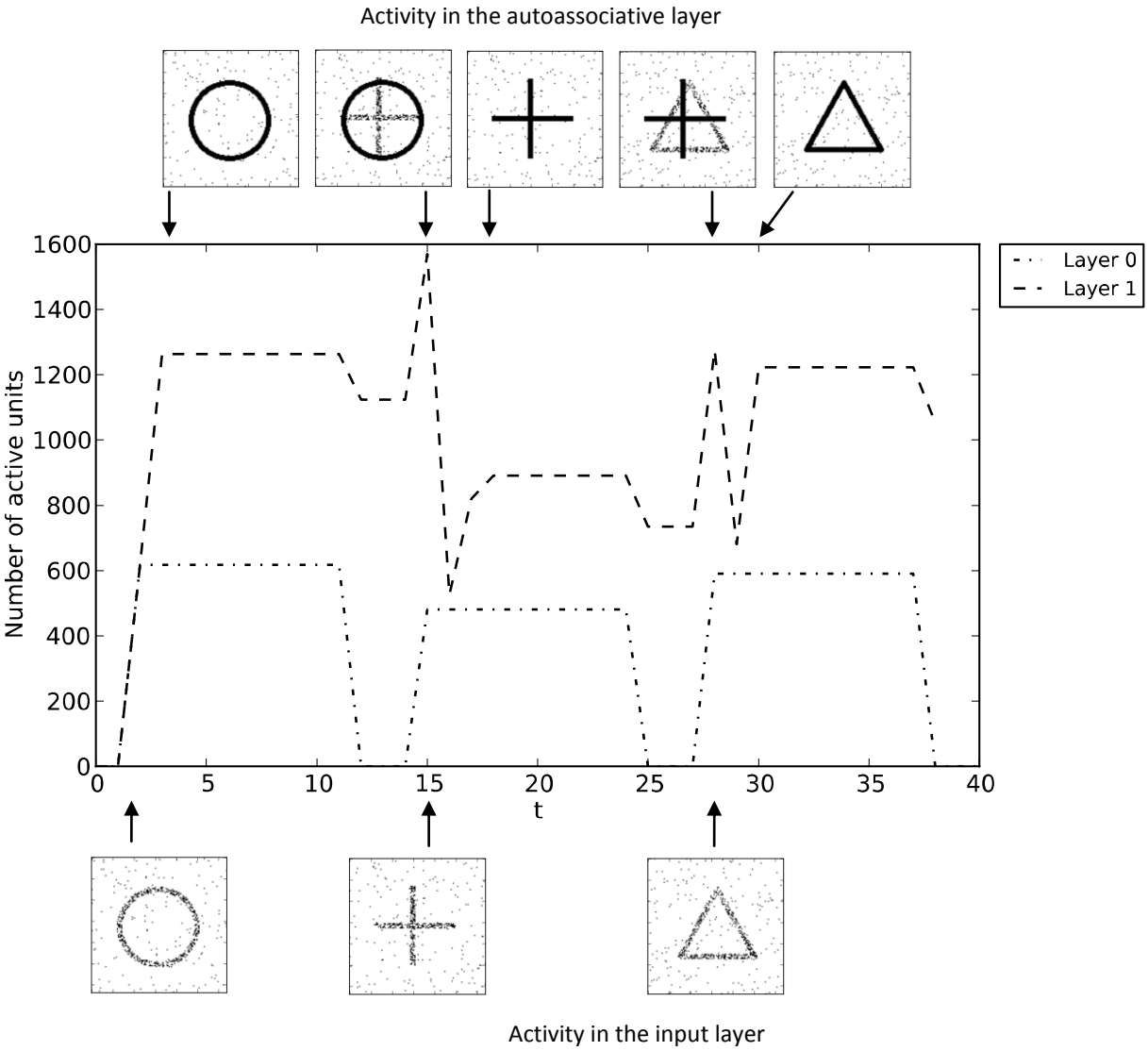

Figure 4. Demonstration of pattern completion. Number of active units for every time step during the test phase for the input layer and the autoassociative layer. The activity in both layers are shown as 2d-images for a selection of the time steps. The upper row of images shows the activity in the autoassociative layer and the lower the activity in the input layer. Three different patterns where used, a circle, a plus sign and a triangle. Model parameters: activation threshold, $\gamma = 300$, background activity, $\lambda_0 = 0.005$, learning rate, $\alpha = 0.1$.

Silicon retina input

Here event-based input from the silicon retina is used as input during the testing phase. The simulation runs simultaneously as the silicon retina is operated and the events are sent continuously to the simulator. The training is conducted as in the previous experiment but instead of a circle, a plus and a triangle, three vertical bars with different horizontal position (left, center, right) are used as training patterns, see figure 5. The reason for using bars instead of the shapes used previously is that bar-like forms are easier to produce with the silicon retina.

The simulation is updated every 100 ms and all arriving events during this time are used to set the activity of the input layer. About 40 nodes of the Lindgren cluster is used for a total of 960 processor cores. In figure 6 the network activity is shown for a selection of the time steps.

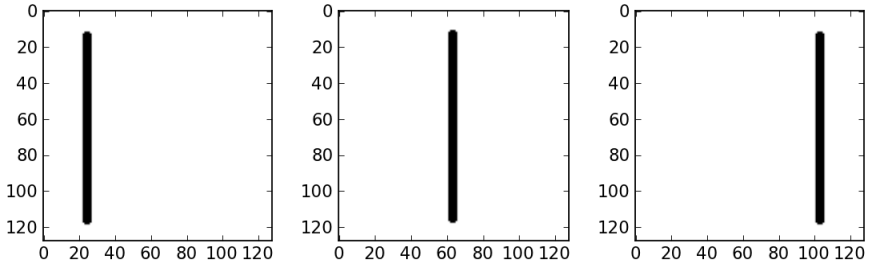

Figure 5. Training patterns used for the silicon retina experiment.

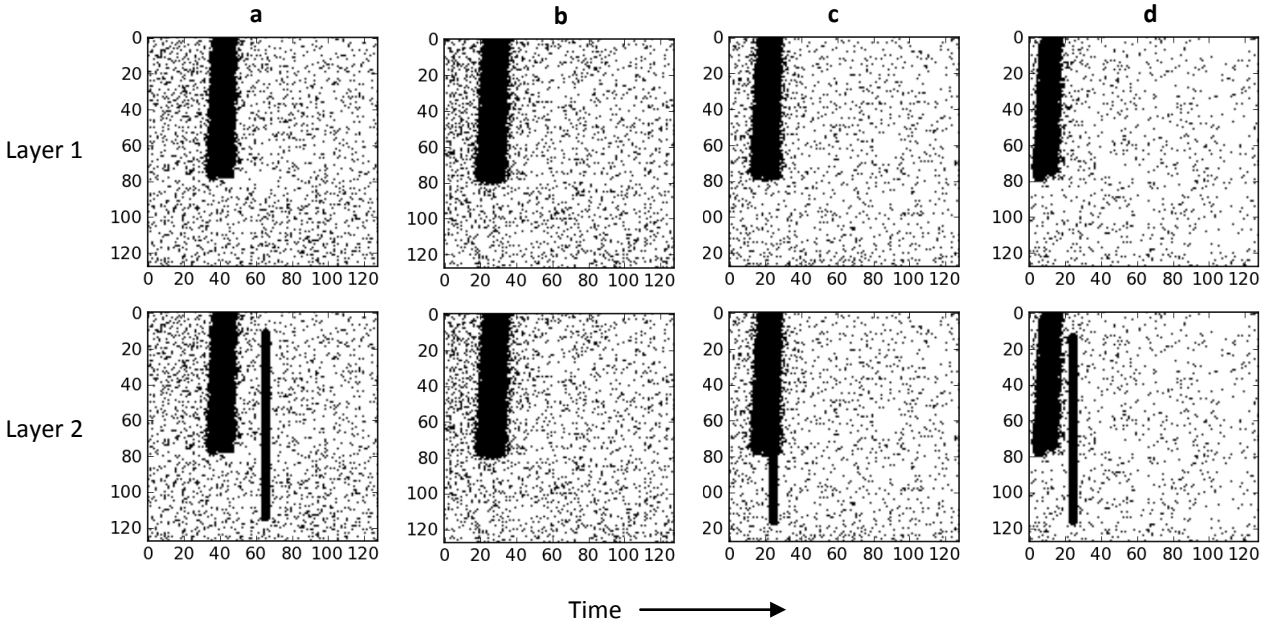

Figure 6. Four snapshots of the network activity of the two layers generated from silicon retina input. The silicon retina is pointed at a piece of black adhesive tape mounted on a white wall and moved slowly to the right. As the bar depicting the tape starts to overlap a new training pattern (**b**) the previously activated pattern (seen in **a**) dies out and the new pattern gets activated (**c**). The newly activated pattern stays active as the silicon retina is moved further to the right (**d**). Same model parameters where used as in the previous pattern completion experiment.

5. Discussion

Here we have demonstrated a first step towards using event-based sensor data as input to large scale neural networks simulated on supercomputers. The model developed for this purpose is simplistic and should only be seen as a proof of concept. It does not for instance utilize the events (or spikes) in way that one would expect from a neural system. A natural next step would be to use the silicon retina in connection with a model of the early human visual system in which at least individual events are converted to firing rates.

The DVS128 silicon retina we used here is as previously mentioned a temporal contrast sensor which means that it outputs the derivative of the image projected onto the sensor surface. This is a design choice which allows the DVS128 to have a very wide dynamic range of six decades and minimum redundancy in its response. However, the complete lack of steady state response seems to makes it hard to recreate a static image from the events. Some experiments have been conducted where the events for every pixel were summed up to investigate how well on- and off-events balance each other. If this balance was perfect the integrated image

would correspond to the static image produced by a conversional black and white camera. Unfortunately the images we have been able to produce are of a way to poor quality to be considered useful. However, more work in this direction could probably lead to better results.

A main strength of the DVS128 is its very high time resolution which makes it suitable for many applications but the need for microsecond precision is irrelevant if the purpose is to mimic the slow mammalian visual system. Moreover, in our setup we use jAER which has a maximum update rate of 120 Hz. This means that all events received from the sensor between two updates are grouped together in packets and sent as a UDP datagram. This induces a latency which further removes the purpose of using an asynchronous sensor like the DVS128. One can overcome this caveat by handling the USB event stream directly instead of going through jAER but if the event are to be sent over a network they have to be grouped in packets somehow anyway. Another option would be to use a conventional camera with a fairly high frame rate and convert the image to an event-based representation in software. This would allow for much higher spatial resolution and more control over how the event-based representation is produced.

Also worth noticing is that other attempts to build event-based retina-like image sensors exist which unlike the DVS128 combine both temporal and spatial filtering [5, 10]. This is more in line with the functioning of biological retinas and is preferable if high biological realism is of interest.

References

[1] H. Markrama, K. Meierb, T. Lippertc, S. Grillnerd, R. Frackowiake, S. Dehaenef, A. Knollg, H. Sompolinskyh, K. Verstrekeni, J. DeFelipej, S. Grantk, J.P. Changeuxl, A. Sariam, "Introducing the Human Brain Project", Procedia Computer Science, Volume 7, 2011, Pages 39–42

[2] D. Brüderle, M.A. Petrovici, B. Vogginger, M. Ehrlich and T. Pfeil, et al., "A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems", Biological Cybernetics, 2011, Volume 104, Numbers 4-5, Pages 263-296

[3] S.H. Jo, T. Chang, I. Ebong, B.B. Bhadviya, P. Mazumder, W. Lu, "Nanoscale Memristor Device as Synapse in Neuromorphic Systems", Nano Letters 2010, 10, 1297-–1301

[4] "Dynamic Vision Sensor (DVS) - asynchronous temporal contrast silicon retina" [online], Available from: http://siliconretina.ini.uzh.ch/wiki/index.php

[5] T. Delbruck, P. Lichtsteiner, "Freeing vision from frames", The Neuromorphic Engineer, Volume 3, Issue 1, May 2006

[6] "jAER Open Source Project" [online], Available from: http://sourceforge.net/apps/trac/jaer/wiki

[7] M. Djurfeldt, J. Hjorth, J. M. Eppler, N. Dudani, M. Helias, T. C. Potjans, U. S. Bhalla, M. Diesmann, J. H. Kotaleski and Ö. Ekeberg, "Run-Time Interoperability Between Neuronal Network Simulators Based on the MUSIC Framework", Neuroinformatics, Volume 8, Number 1, 43-60, DOI: 10.1007/s12021-010-9064-z, 2010.

[8] S. Benjaminsson, A. Lansner, "Extreme Scaling of Brain Simulations", Jülich Blue Gene/P Extreme Scaling Workshop 2011, Technical Report FZJ-JSC-IB-2011-02. Available from: http://www2.fz-juelich.de/jsc/docs/autoren2011/mohr1/

[9] A. Sandberg, “Bayesian Attractor Neural Network Models of Memory”, Ph.D. dissertation Stockholm University, Department of Numerical Analysis and Computer Science, TRITA-NA-0310, ISBN 91-7265-684-0, June 2003.

[10] K.A. Zaghloul, K.A. Boahen, “Optic Nerve Signals in a Neuromorphic Chip I: Outer and Inner Retina Models”, IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING, VOL. 51, NO. 4, APRIL 2004

CB – Computational Biology
TRITA-CSC-CB 2012:02
ISRN KTH/CSC/CB--12/02--SE
ISSN 1653-5707

CB – Computational Biology
School of Computer Science and Communication
KTH Royal Institute of Technology
www.kth.se/csc