Massively Parallel Neural Encoding and Decoding of Visual Stimuli

Aurel A. Lazar[*], Yiyin Zhou

Department of Electrical Engineering, Columbia University, New York, NY 10027, USA

[*]Corresponding author. Tel: +1 212-854-1747; fax: +1 212-932-9421
Email addresses: `aurel@ee.columbia.edu` (Aurel A. Lazar), `yiyin@ee.columbia.edu` (Yiyin Zhou)

Massively Parallel Neural Encoding and Decoding of Visual Stimuli

Aurel A. Lazar[*], Yiyin Zhou

Department of Electrical Engineering, Columbia University, New York, NY 10027, USA

Abstract

The massively parallel nature of Video Time Encoding Machines (TEMs) calls for scalable, massively parallel decoders that are implemented with neural components. The current generation of decoding algorithms is based on computing the pseudo-inverse of a matrix and does not satisfy these requirements.

Here we consider Video TEMs with an architecture built using Gabor receptive fields and a population of Integrate-and-Fire neurons. We show how to build a scalable architecture for Video Time Decoding Machines using recurrent neural networks. Furthermore, we extend our architecture to handle the reconstruction of visual stimuli encoded with massively parallel Video TEMs having neurons with random thresholds. Finally, we discuss in detail our algorithms and demonstrate their scalability and performance on a large scale GPU cluster.

Keywords: neural encoding of visual stimuli, spiking neural models, massively parallel reconstruction of visual stimuli, recurrent neural networks, neural circuits with random thresholds, receptive fields.

1. Introduction

The increasing availability of multi-electrode recordings and functional imaging methods has led to the application of neural decoding techniques to the recovery of complex stimuli such as natural video scenes. An optimal linear decoding algorithm was applied by (Stanley et al., 1999) to the reconstruction of natural video scenes from recordings of a neural population of the cat's Lateral Geniculate Nucleus (LGN) resulting in recognizable moving objects. Visual image reconstruction from fMRI data was examined in (Miyawaki et al., 2008), whereas in (Kay et al., 2008) fMRI data was used to identify natural images.

A formal model based approach for encoding and reconstruction in the early visual system was advanced in (Lazar & Pnevmatikakis, 2011) and (Lazar et al., 2010). In this approach Time Encoding Machines (TEMs) model the representation (encoding)

[*]Corresponding author. Tel: +1 212-854-1747; fax: +1 212-932-9421
Email addresses: aurel@ee.columbia.edu (Aurel A. Lazar), yiyin@ee.columbia.edu (Yiyin Zhou)

of stimuli by sensory systems with neural circuits that communicate via spikes (action potentials). Single-input single-output TEMs asynchronously encode time-varying analog stimuli into a time sequence (Lazar & Tóth, 2004). Video Time Encoding Machines (Video TEMs) encode space-time-varying signals including visual stimuli (movies, animation) into a multidimensional time sequence (Lazar & Pnevmatikakis, 2011). Different models of neural encoding circuits have been investigated including circuits with random parameters (Lazar et al., 2010).

Hardware implementations of TEMs are also available. For example Asynchronous Sigma-Delta Modulators (ASDMs), that have been shown to be an instance of single-input single-output TEMs, can be robustly implemented in low power analog VLSI (Kinget et al., 2005). With the ever decreasing voltage and increasing clock rate, amplitude domain high precision quantizers are more and more difficult to implement. In the nanoworld, it is more cost effective to measure "time" as opposed to measuring "space" (signal amplitude). Thus, information representation in time domain matches up with the miniaturization trends of nanotechnology. The next generation silicon encoders are expected to operate in the time domain (Lazar et al., 2008).

Given Nyquist-type rate conditions, a time encoded bandlimited signal can be recovered with arbitrary accuracy by Time Decoding Machines (TDMs) (Lazar & Tóth, 2004). For stimuli encoded with single-input single-output TEMs several real-time reconstruction algorithms have been demonstrated in the past (Lazar et al., 2008; Harris et al., 2008). Although the encoding mechanism can be efficiently implemented, the reconstruction algorithms call for the pseudo-inversion of a matrix. The massively parallel nature of Video TEMs calls for scalable, massively parallel decoders that are (preferably) implemented with neural components. The current generation of decoding algorithms is based on computing the pseudo-inverse of a matrix and does not satisfy these requirements.

Here we consider Video TEMs built using Gabor receptive fields and a population of Integrate-and-Fire neurons. We seek a solution for the reconstruction of time encoded signals using neural hardware components (Lazar, 2006). Clearly, a decoding circuit built using neural components has to minimize the same cost function that leads to a solution via a matrix pseudo-inverse.

We show how to build a scalable architecture for Video TDMs using recurrent neural networks (Cichocki & Unbehauen, 1993). The recurrent neural network decoding method has two main advantages: (i) it is intrinsically parallel and thereby *scalable* for real-time decoding, and (ii) it can be implemented using simple neural hardware components. Furthermore, we extend our architecture to handle the reconstruction of visual stimuli encoded with massively parallel Video TEMs having neurons with random thresholds. Finally, we discuss in detail our algorithms and demonstrate their scalability and performance on a GPU cluster. Briefly, the simulation results show that the proposed method provides high quality reconstructions that are comparable to the ones obtained by applying the matrix pseudo-inverse method.

This paper is organized as follows. In section 2 the vector space of visual stimuli is introduced. In section 3 the massively parallel architecture of Video TEMs consisting of receptive fields in cascade with neural circuits is described. The massively parallel architecture of Video TDMs using recursive neural networks is presented in section 4. The complexity of the massively parallel encoding and decoding algorithms is discussed in section 5. In section 6 two examples of encoding of visual stimuli with TEMs with deterministic/random thresholds are given. Section 7 briefly concludes the paper.

2. Modeling the Visual Stimuli

In this paper visual stimuli are modeled as elements of the vector space of tri-variable trigonometric polynomials denoted by $\mathcal{H}$. Each element $I \in \mathcal{H}$ is of the form

$$I(x,y,t) = \sum_{m_x=-M_x}^{M_x} \sum_{m_y=-M_y}^{M_y} \sum_{m_t=-M_t}^{M_t} c_{m_x,m_y,m_t} e_{m_x,m_y,m_t}(x,y,t), \qquad (1)$$

where the $c_{m_x,m_y,m_t} \in \mathbb{R}$ are constants and

$$\begin{aligned} & e_{m_x,m_y,m_t}(x,y,t) \\ & = e_{m_x}(x) e_{m_y}(y) e_{m_t}(t) \\ & = \exp\left(jm_x \frac{\Omega_x}{M_x} x + jm_y \frac{\Omega_y}{M_y} y + jm_t \frac{\Omega_t}{M_t} t \right), \\ & m_x = -M_x, \cdots, M_x, m_y = -M_y, \cdots, M_y, \\ & m_t = -M_t, \cdots, M_t, \end{aligned}$$

constitute a basis of $\mathcal{H}$ and $(x,y,t) \in \mathbb{R}^3$. $(\Omega_x, \Omega_y, \Omega_t)$ and (M_x, M_y, M_t) are, respectively, the bandwidth and the order of the trigonometric polynomials in each variable. An element $I \in \mathcal{H}$ is also, respectively, periodic in each variable with period

$$S_x = \frac{2\pi M_x}{\Omega_x}, S_y = \frac{2\pi M_y}{\Omega_y}, S_t = \frac{2\pi M_t}{\Omega_t}.$$

By defining the inner product in $\mathcal{H}$ as

$$\langle I_1, I_2 \rangle = \frac{1}{S_x S_y S_t} \int_{-S_t/2}^{S_t/2} \int_{-S_x/2}^{S_x/2} \int_{-S_y/2}^{S_y/2} I_1(x,y,t) \overline{I_2(x,y,t)} dx dy dt,$$

the space of trigonometric polynomials is a Hilbert space. Since $\mathcal{H}$ is finite dimensional it is also a Reproducing Kernel Hilbert Space (RKHS). The reproducing kernel (RK) is given by

$$\begin{aligned} & K(x,y,t;x',y',t') = \\ & \sum_{m_x=-M_x}^{M_x} \sum_{m_y=-M_y}^{M_y} \sum_{m_t=-M_t}^{M_t} e_{m_x,m_y,m_t}(x-x', y-y', t-t'). \end{aligned}$$

Using the above Hilbert space as a model of visual stimuli is extensively discussed and justified in (Lazar et al., 2010).

3. Massively Parallel Video Time Encoding

In this section we describe the architecture of space-time Video Time Encoding Machines. For a detailed and highly intuitive treatment of the one-dimensional case we refer the reader to (Lazar & Zhou, 2011).

3.1. *Video Encoding with IAF Neurons*

The architecture of the video TEM is shown in Fig. 1. It is a massively parallel architecture, with each parallel branch consisting of two modules in cascade: a visual receptive field and a neural circuit consisting of an IAF neuron.

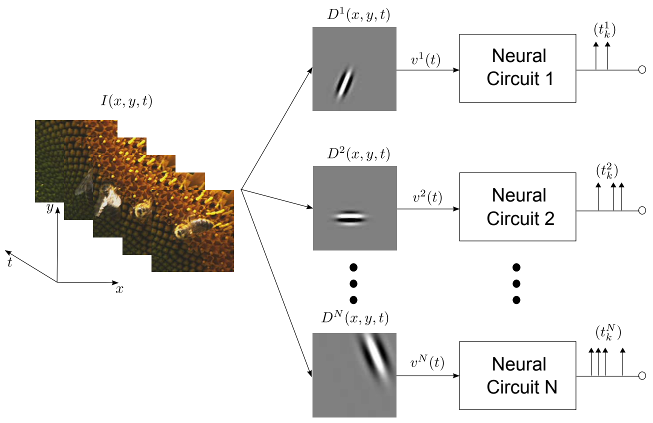

Figure 1: Diagram of Video Time Encoding Machine.

Visual receptive fields are often used to model the preference of a neuron to the spatio-temporal pattern of stimuli. More formally, the receptive fields considered here are spatio-temporal linear filters that preprocess the visual stimuli and feed them to the the IAF neurons. We denote the visual receptive field in the video TEM as $D^j(x,y,t), j = 1,\dots,N$, where N is the number of branches. In the case of spatio-temporal separable receptive fields, the jth receptive field can be separated into

$$D^j(x,y,t) = D_S^j(x,y)D_T^j(t),$$

where $D_S^j(x,y)$ is the spatial and $D_T^j(t)$ is the temporal component of the receptive field.

Often, the spatial component of the receptive field of simple cells in the primary visual cortex (V1) can be mathematically generated by the Gabor mother wavelet

$$\gamma(x,y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{4x^2+y^2}{8}\right)\left(e^{i\kappa x} - e^{-\kappa^2/2}\right). \tag{2}$$

From the Gabor mother wavelet, we can derive a family of Gabor filters that represents the spatial component of the receptive fields of different neurons. The set of all receptive fields can be obtained by performing the following three operations or combinations thereof:

- Dilation $D_\alpha, \alpha \in \mathbb{R}_+$: $D_\alpha \gamma(x,y) = |\alpha|^{-1}\gamma(\frac{x}{\alpha}, \frac{y}{\alpha})$,
- Rotation $R_\theta, \theta \in [0, 2\pi)$: $R_\theta \gamma(x,y) = \gamma(x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta)$,
- Translation $T_{(x_0,y_0)}, (x_0,y_0) \in \mathbb{R}^2$: $T_{(x_0,y_0)}\gamma(x,y) = \gamma(x-x_0, y-y_0)$.

In this paper we will primarily focus on the case of space-time separable receptive fields and use spatial Gabor filters. However, the formulation developed here can be applied to more general settings.

In order to reconstruct the video signal, we require that the set of receptive fields forms a frame and covers the entire spatial field (Lazar & Pnevmatikakis, 2011). For a finite aperture ("stimulus size"), such a condition imposes a lower bound on the number of neurons N. As the examples in section 6 demonstrate, even for small apertures, a large number of neurons is required to faithfully represent the visual stimuli. Due to the parallelism of the encoding architecture one can readily deal with a massively large number of neurons.

Mathematically, the filtering operation of the visual receptive field at the jth branch $D^j(x,y,t)$ is given by the operator ${}^{\mathbb{F}}L^j : \mathcal{H} \to \mathcal{H}_t$ by

$$ {}^{\mathbb{F}}L^j I = \int_{\mathbb{R}} \left(\int_{\mathbb{D}^2} D^j(x,y,s) I(x,y,t-s)dxdy \right) ds = v^j(t), \tag{3}$$

where $\mathbb{D}^2$ is the aperture. ${}^{\mathbb{F}}L^j$ maps the visual stimulus from the 3-D space $\mathcal{H}$ into the 1-D space $\mathcal{H}_t$, the space of univariable trigonometric polynomials with bandwidth Ω_t and order M_t.

After filtering by the receptive field, the output in each branch is fed into an IAF neuron that encodes the continuous signal $v^j \in \mathcal{H}_t$ into a spike train. Let us denote the output of the jth neuron as $(t_k^j), k = 1, 2, \cdots, n_j$. The encoding is described by the t-transform:

$$\int_{t_k^j}^{t_{k+1}^j} v^j(t)ds = \kappa^j\delta^j - b^j(t_{k+1}^j - t_k^j),$$

for all $k \in \mathbb{Z}$, where κ^j, δ^j and b^j are, respectively, the integration constant, threshold and bias of the jth IAF neuron. We now define the bounded linear functionals ${}^{\mathbb{T}}L_k^j : \mathcal{H}_t \to \mathbb{R}$ as

$$ {}^{\mathbb{T}}L_k^j v^j = \int_{t_k^j}^{t_{k+1}^j} v^j(t)ds = \kappa^j\delta^j - b^j(t_{k+1}^j - t_k^j) = q_k^j$$

with $v^j \in \mathcal{H}_t$, for all $j = 1, 2, ..., N$.

Finally, we define the bounded linear functionals $L_k^j : \mathcal{H} \to \mathbb{R}$ as the composition of the two operators above describing receptive field filtering and neuron encoding

$$L_k^j = {}^{\mathbb{T}}L_k^j \, {}^{\mathbb{F}}L^j. \tag{4}$$

Therefore

$$L_k^j I = {}^{\mathbb{T}}L_k^j \, {}^{\mathbb{F}}L^j I = \langle I, \phi_k^j \rangle = q_k^j,$$

where the second equality is due to the Riesz representation theorem and

$$\phi_k^j(x,y,t) = \langle \phi_k^j, K_{x,y,t} \rangle = L_k^j \overline{K_{x,y,t}},$$

with $K_{x,y,t}(x',y',t') = K(x,y,t;x',y',t')$.

Formulation of time encoding of stimuli in inner product form provides a simple yet very powerful insight into the encoding process itself. Since the inner products are merely projections of the visual stimulus onto the axes defined by the ϕ_k^js, encoding is interpreted as generalized sampling, and the q_k^js are the measurements given by sampling the signal. Note however that unlike in traditional sampling, the sampling functionals in time encoding are signal dependent.

3.2. Video Encoding with IAF Neurons with Random Thresholds

In this section we assume that all IAF neurons of the video TEM have thresholds that are randomly distributed. Biological neurons in the fly visual system and in the early visual system of the cat have been modeled as IAF neurons with random thresholds by (Gestri et al., 1980) and (Reich et al., 1997). Here we assume that the threshold value is distributed according to the Gaussian distribution $\mathcal{N}(\delta^j, (\sigma^j)^2)$. We also assume here that the random value of the threshold is kept constant between two consecutive spikes. Therefore, the t-transform of the IAF neuron with random threshold can be expressed as (see also (Lazar et al., 2010))

$$\int_{t_k^j}^{t_{k+1}^j} v^j(t)ds = \kappa^j \delta_k^j - b^j(t_{k+1}^j - t_k^j) = q_k^j + \epsilon_k^j, \tag{5}$$

where

$$q_k^j = \kappa^j \delta^j - b^j(t_{k+1}^j - t_k^j),$$

and $\epsilon_k^j = \kappa^j(\delta_k^j - \delta^j)$ are i.i.d. random variables with mean zero and variance $(\kappa^j \sigma^j)^2$ for all $k = 1, 2, \cdots, n_j$, $j = 1, 2, \cdots, N$. By defining the bounded linear operators $L_k^j : \mathcal{H} \to \mathbb{R}$ (see also (4)), the t-transform of the video TEM is given by

$$L_k^j I = \langle I, \phi_k^j \rangle = q_k^j + \epsilon_k^j.$$

for all $k \in \mathbb{Z}$.

4. Massively Parallel Video Time Decoding

In this section we describe the architecture of space-time Video Time Decoding Machines. For a detailed and highly intuitive treatment of the one-dimensional case we refer the reader to (Lazar & Zhou, 2011).

4.1. Video Time Decoding with Recurrent Neural Networks

We formulate the reconstruction of the encoded stimulus as a variational problem. Given the spike times and the parameters (including the receptive field) of the neuron, the reconstruction is the spline interpolation problem (Bezhaev & Vasilenko, 2001)

$$\hat{I} = \underset{I \in \mathcal{H}, \{L_k^j I = q_k^j\}_{k=1,\cdots,n_j}^{j=1,\cdots,N}}{\operatorname{argmin}} \{\|I\|_{\mathcal{H}}^2\}. \tag{6}$$

Therefore, the goal of stimulus reconstruction is to find a minimum norm solution among all the elements in the RKHS that are consistent with the measurements made in the encoding stage.

Theorem 1. *The solution to the spline interpolation problem* (6) *is*

$$\hat{I} = \sum_{j=1}^{N} \sum_{k=1}^{n_j} c_k^j \phi_k^j, \tag{7}$$

where the c_k^j's are the solution to the system of linear equations

$$\mathbf{Gc} = \mathbf{q}. \tag{8}$$

with $\mathbf{c} = \left[c_1^1, c_2^1, \cdots, c_{n_1}^1, c_1^2, c_2^2, \cdots, c_{n_2}^2, \cdots, c_1^N, \cdots, c_{n_N}^N\right]^T$,
$\mathbf{q} = \left[q_1^1, q_2^1, \cdots, q_{n_1}^1, q_1^2, q_2^2, \cdots, q_{n_2}^2, \cdots, q_1^N, \cdots, q_{n_N}^N\right]^T$*, and* $\mathbf{G} = [\mathbf{G}^{ij}]$ *is a block matrix with block entries given by*

$\left[\mathbf{G}^{ij}\right]_{kl} = \langle \phi_k^i, \phi_l^j \rangle$, *for all* $i, j = 1, 2, \cdots, N$ *and* $k = 1, 2, \cdots, n_i, l = 1, 2 \cdots, n_j$.

Proof: The form of the solution (7) is given by the Representer Theorem. Substituting the solution into equation (6), the coefficients c_k^j can be obtained by solving the constraint optimization problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\mathbf{c}^T\mathbf{Gc} \\ \text{subject to} & \mathbf{Gc} = \mathbf{q} \end{array}. \tag{9}$$

It is easy to see that the above quadratic optimization problem is equivalent to solving the system of linear equations (8). □

Note that

$$\phi_k^i = \sum_{m_x=-M_x}^{M_x} \sum_{m_y=-M_y}^{M_y} \sum_{m_t=-M_t}^{M_t} a_{k,m_x,m_y,m_t}^i e_{m_x,m_y,m_t},$$

where

$$\begin{aligned} a_{k,m_x,m_y,m_t}^i &= \langle \phi_k^i, e_{m_x,m_y,m_t} \rangle = L_k^i e_{-m_x,-m_y,-m_t} \\ &= \int_{t_k^i}^{t_{k+1}^i} \int_{\mathbb{R}} \left(\int_{\mathbb{D}^2} D^i(x,y,s) e_{-m_x,-m_y}(x,y) dxdy \right) e_{-m_t}(t-s) dsdt. \end{aligned}$$

Then, the entries of $\mathbf{G}$ can be more explicitly expressed as

$$\left[\mathbf{G}^{ij}\right]_{kl} = \sum_{m_x=-M_x}^{M_x} \sum_{m_y=-M_y}^{M_y} \sum_{m_t=-M_t}^{M_t} a^i_{k,m_x,m_y,m_t} . \overline{a^j_{l,m_x,m_y,m_t}}.$$

In the case where spatio-temporal separable receptive fields are used in the encoding, a^i_{k,m_x,m_y,m_t} can be further separated into

$$a^i_{k,m_x,m_y,m_t} = d^i_{m_x,m_y} p^i_{k,m_t},$$

where

$$d^i_{m_x,m_y} = \int_{\mathbb{D}^2} D^i_S(x,y) e_{-m_x,-m_y}(x,y) dxdy,$$
$$p^i_{k,m_t} = \int_{t^i_k}^{t^i_{k+1}} \int_{\mathbb{R}} D^i_T(s) e_{-m_t}(t-s) dsdt.$$

Thus, the spatial and the temporal components can be computed separately.

Since $\mathbf{G}$ is typically ill-conditioned, the Moore-Penrose pseudo-inverse (Penrose, 1955) is often used to obtain the solution for $\mathbf{c}$. A popular albeit computationally demanding algorithm for evaluating the pseudo-inverse is based on singular value decomposition (SVD). Recurrent neural networks have been extensively studied to solve linear equations and optimization problems (Cichocki & Unbehauen, 1993). These networks consist of neuron like operators that are simple to implement. Moreover, they provide an architecture that can be massively parallelized thereby providing a more plausible solution to the reconstruction procedure. We now describe a recurrent neural network that leads to efficient video decoding.

Theorem 2. *The solution to the spline interpolation problem* (6) *is*

$$\hat{I} = \sum_{j=1}^{N} \sum_{k=1}^{n_j} c^j_k \phi^j_k, \tag{10}$$

where $\mathbf{c}$ *is the stationary point of the system of differential equations*

$$\frac{d\mathbf{c}}{dt} = \alpha \left(\mathbf{q} - \mathbf{G}\mathbf{c}\right), \tag{11}$$

with initial condition $\mathbf{c}(0) = \mathbf{0}$ *and* $\alpha > 0$.

Proof: For the energy function $E(\mathbf{c}) = \frac{1}{2}\|\mathbf{G}\mathbf{c} - \mathbf{q}\|^2 \geq 0$, we have

$$\frac{dE}{dt} = \sum_{i=1}^{n} \frac{\partial E}{\partial c_i} \frac{dc_i}{dt} = (\nabla E)^T \frac{d\mathbf{c}}{dt} = -(\mathbf{G}\mathbf{c} - \mathbf{q})^T \mathbf{G} (\mathbf{G}\mathbf{c} - \mathbf{q}) \leq 0,$$

where $n = \sum_{j=1}^{N} n_j$. The last inequality is due to the fact that $\mathbf{G}$ is positive-semidefinite. Therefore, the time derivative of the energy function is monotonically decreasing, and the equalibrium condition, $\frac{dE}{dt} = 0$, is satisfied if and only if $\mathbf{Gc} - \mathbf{q} = 0$ or $\mathbf{G}^T(\mathbf{Gc} - \mathbf{q}) = 0$. In other words, the solution to equation (11) is guaranteed to converge to a stationary point $\mathbf{c}$ that satisfies either exactly $\mathbf{Gc} = \mathbf{q}$, or $\mathbf{c} = (\mathbf{G}^T\mathbf{G})^+\mathbf{G}^T\mathbf{q} = \mathbf{G}^+\mathbf{q}$, where $\mathbf{G}^+$ denotes the Moore-Penrose pseudo-inverse of $\mathbf{G}$. $\square$

Equation (11) can easily be implemented by a circuit consisting of neural components such as integrators and adders; the video TDM can be realized as the diagram shown in Fig. 2. Note that the above system of differential equations may converge slowly. In such cases the circuit simulation can be stopped while still guaranteeing a high quality reconstruction of the visual stimuli (see also section 6).

The recurrent neural circuit in Fig. 2 is massively parallel. It consists of $n = \sum_{j=1}^{N} n_j$ parallel branches. This number can be very large since it represents the total number of spikes generated by the encoder. In each branch, only two simple components are required - a multiply/add unit and an integrator. Such simple circuits can be effectively realized in analog VLSI or simulated on a high performance computer.

The original method of evaluating the pseudo-inverse, on the one hand, is typically based on an elegant mathematical treatment using SVD. It requires additional workspace in memory that limits the problem size that can be solved, and although scaling the algorithm to multiple computing devices is possible, it is highly nontrivial. The recurrent neural network approach, on the other hand, is rather straightforward to both scale up and scale to multiple computing devices, such as a cluster of GPUs. It does not require extra workspace so that the entire memory resource can efficiently be utilized. As we will show later in this section, only a small modification is needed in the implementation of the recurrent neural networks in order to scale the reconstruction to a large number of GPU nodes.

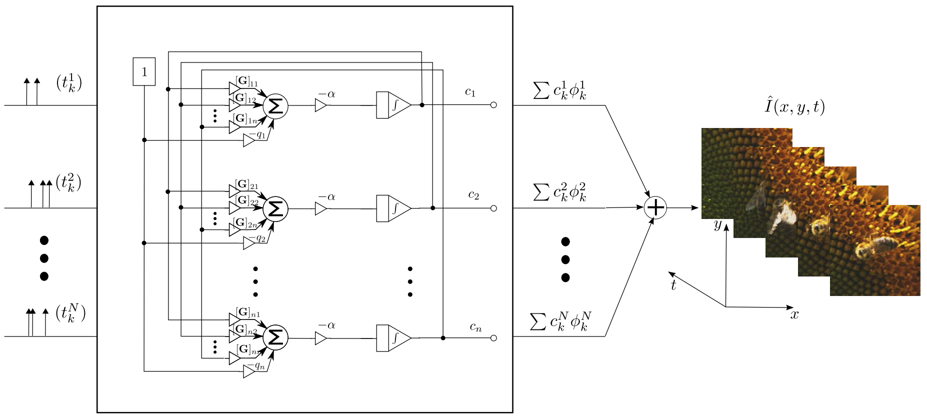

Figure 2: Block diagram of the video TDM implemented using recurrent neural networks. The recurrent neural network is shown in the square box.

4.2. Extension to Encoding with Neurons with Random Threshold

We can easily extend the existing recurrent neural network architecture to decoding of stimuli encoded with Video TEMs with random thresholds. In devising the decoding algorithm, we again take a variational approach, by considering the reconstruction as the solution to the smoothing spline problem

$$\hat{I} = \underset{I \in \mathcal{H}}{\operatorname{argmin}} \{ \sum_{j=1}^{N} \frac{1}{(\kappa^j \sigma^j)^2} \sum_{k=1}^{n_j} \left(\langle I, \phi_k^j \rangle - q_k^j \right)^2 + n\lambda \|I\|_{\mathcal{H}}^2 \}, \tag{12}$$

where $n = \sum_{j=1}^{N} n_j$. The above formulation aims to minimize the error between samples and measurements, and at the same time, it is regularized by the norm of the signal in the Hilbert space.

Theorem 3. *The solution to the smoothing spline problem* (12) *is*

$$\hat{I} = \sum_{j=1}^{N} \frac{1}{\kappa^j \sigma^j} \sum_{k=1}^{n_j} c_k^j \phi_k^j, \tag{13}$$

where the c_k^j's are given by the solution of the system of linear equations

$$\mathbf{G}^T \left(\mathbf{G} + n\lambda \mathbf{I} \right) \mathbf{c} = \mathbf{G}^T \mathbf{q}. \tag{14}$$

with $\mathbf{c} = \left[c_1^1, c_2^1, \cdots, c_{n_1}^1, c_1^2, c_2^2, \cdots, c_{n_2}^2, \cdots, c_1^N, \cdots, c_{n_N}^N \right]^T$,
$\mathbf{q} = \left[\frac{1}{\kappa^1 \sigma^1} q_1^1, \frac{1}{\kappa^1 \sigma^1} q_2^1, \cdots, \frac{1}{\kappa^1 \sigma^1} q_{n_1}^1, \frac{1}{\kappa^2 \sigma^2} q_1^2, \frac{1}{\kappa^2 \sigma^2} q_2^2, \cdots, \frac{1}{\kappa^2 \sigma^2} q_{n_2}^2, \cdots, \frac{1}{\kappa^N \sigma^N} q_1^N, \cdots, \frac{1}{\kappa^N \sigma^N} q_{n_N}^N \right]^T$,
$\mathbf{I}$ *is the $n \times n$ identity matrix and $\mathbf{G} = \left[\mathbf{G}^{ij} \right]$ is a block matrix with block entries given by*

$$\left[\mathbf{G}^{ij} \right]_{kl} = \frac{\langle \phi_k^i, \phi_l^j \rangle}{(\kappa^i \sigma^i)(\kappa^j \sigma^j)}, \text{ for all } i, j = 1, 2, \cdots, N \text{ and } k = 1, 2, \cdots, n_i, l = 1, 2, \cdots, n_j.$$

Proof: Again, due to the Representer Theorem, the solution of the problem is of the form (13). Substituting the solution into (12), the coefficients c_k^j are the solution to the unconstrained optimization problem

$$\text{minimize } E(\mathbf{c}) = \|\mathbf{G}\mathbf{c} - \mathbf{q}\|_{l_2}^2 + n\lambda \mathbf{c}^T \mathbf{G} \mathbf{c}. \tag{15}$$

Since the problem is convex, the solution is give by setting the gradient of the objective function to zero. Hence, we have

$$\nabla E(\mathbf{c}) = 2 \left(\mathbf{G}^T \mathbf{G} \mathbf{c} + n\lambda \mathbf{G} \mathbf{c} - \mathbf{G}^T \mathbf{q} \right) = \mathbf{0}, \tag{16}$$

or

$$\mathbf{G}^T \left(\mathbf{G} + n\lambda \mathbf{I} \right) \mathbf{c} = \mathbf{G}^T \mathbf{q}. \tag{17}$$

□

Using a general gradient approach we now have the following

Theorem 4. *The solution to the smoothing spline problem is*

$$\hat{I} = \sum_{j=1}^{N} \frac{1}{\kappa^j \sigma^j} \sum_{k=1}^{n_j} c_k^j \phi_k^j, \tag{18}$$

where $\mathbf{c}$ *is the stationary point of the differential equations*

$$\frac{d\mathbf{c}}{dt} = -\boldsymbol{\mu} \nabla E(\mathbf{c}), \tag{19}$$

with initial condition $\mathbf{c}(0) = \mathbf{0}$*, where*

$$E(\mathbf{c}) = \left(\|\mathbf{G}\mathbf{c} - \mathbf{q}\|_{l_2}^2 + n\lambda \mathbf{c}^T \mathbf{G}\mathbf{c} \right), \tag{20}$$

and $\boldsymbol{\mu}(\mathbf{c}, t)$ *is an* $n \times n$ *symmetric positive definite matrix, whose entries are usually dependent on the variables* $\mathbf{c}(t)$ *and time* t*.*

Proof: The gradient of the energy function E is guaranteed to vanish since

$$\frac{dE}{dt} = \sum_{i=1}^{n} \frac{\partial E}{\partial c_i} \frac{dc_i}{dt} = (\nabla E)^T \frac{d\mathbf{c}}{dt} = -(\nabla E)^T \boldsymbol{\mu} \nabla E \leq 0,$$

and the system of differential equations reaches stationary point if and only if $\nabla E = 0$. Therefore, (19) asymptotically approaches the global minimizer of the optimization problem (15). □

Consequently, we have

$$\frac{d\mathbf{c}}{dt} = -\boldsymbol{\mu} \mathbf{G}^T \left((\mathbf{G} + n\lambda \mathbf{I})\mathbf{c} - \mathbf{q} \right).$$

The above set of differential equations can also be mapped into a recurrent neural network, and thus the video TDM for visual stimuli encoded with neurons with random thresholds can be realized by the diagram shown in Figure 3. This is a three layer neural network. In the first layer, consisting of n multiply/add units as shown in the left most column, the vector $(\mathbf{G} + n\lambda \mathbf{I})\,\mathbf{c} - \mathbf{q}$ is computed. The multiplication factors are the entries of the matrix $\mathbf{G}+n\lambda\mathbf{I}$ and the vector $\mathbf{q}$. In the second layer, $\nabla E(\mathbf{c})$ is evaluated. This layer also consists of n multiply/add units, with multiplication factors provided by the entries of the matrix $\mathbf{G}$. Note that $\mathbf{G}$ is a symmetric matrix. The gradient is weighted by the learning rate $\boldsymbol{\mu}$ in the third layer, that also consists of n multiply/add units. The outputs of the third layer provide the time derivative of the vector $\mathbf{c}(t)$. The time derivatives are then integrated and the outputs are fed back into the first layer.

The circuit described above (see Figure 3) may converge at a rate even slower than the circuit described in section 4.1 (see also Figure 2). However, note that the vector of coefficients $\mathbf{c}$ satisfying $(\mathbf{G} + n\lambda\mathbf{I})\mathbf{c} = \mathbf{q}$ is also a solution to (17). The latter set of equations is essentially of the same form as (8), as can be seen by replacing $\mathbf{G}$ with $\mathbf{G} + n\lambda\mathbf{I}$. Therefore, the circuit that solves the spline interpolation problem can also be used to solve the smoothing spline problem, while providing a faster speed of convergence.

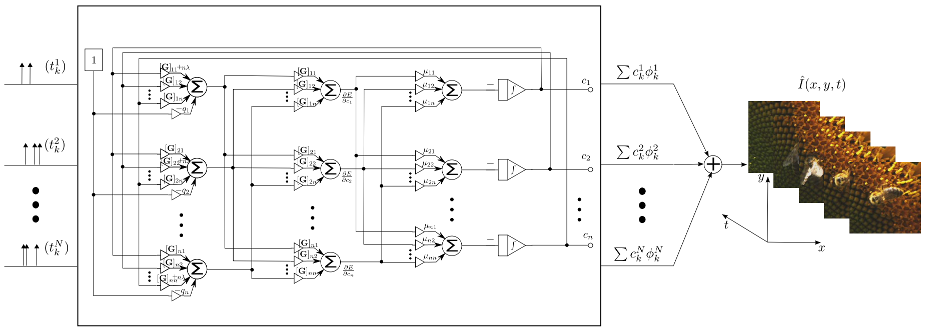

Figure 3: Block diagram of the video TDM implemented by recurrent neural networks when the visual stimulus is encoded by a video TEM with neurons with random threshold. The recurrent neural network is shown in the square box.

5. Algorithmic Considerations

5.1. Encoding on a GPU cluster

Since the architecture of the video TEM is intrinsically parallel, it is straightforward to implement on a GPU cluster. The encoding is implemented one segment of the stimulus at a time. Each segment consists of a certain number of frames of the digital representation of the video. Therefore, encoding can be performed in real-time with a delay approximately given by the duration of the segment. The two cascaded encoding modules, the visual receptive fields and the neural circuits, are treated differently in the implementation.

The double integration of the spatial filtering operation is approximated by a double finite sum. Then, filtering of multiple frames of a visual stimulus by a set of receptive fields is simply a matrix-matrix multiplication, where one of the matrices has each vectorized filter as its columns, and the other matrix has each vectorized visual stimulus frame as its columns. Although the matrix-matrix multiplication can be performed efficiently on GPUs, every receptive field has to be explicitly computed based on its parameters and stored in memory. For large size visual stimuli this requires storing a large number of receptive fields. While computing the receptive fields is extremely efficient, the limited amount of memory on GPUs can create a bottleneck. By increasing the number of GPUs that collectively encode the stimulus, the memory limitation can be relaxed. However, a more cost effective approach can be devised.

We noticed that the spatial filtering can be viewed as the convolution between the image and the same filter centered at zero evaluated at the filter's translation parameter value. Hence, all translations with the same dilation and rotation parameters can be computed simultaneously by a spatial convolution, either by using the FFT method or, for filters with impulse response that are Gabor functions, by using the Fast Gabor Filtering method of (Wang & Shi, 2010). In either case, only one receptive field has to be computed and stored per dilation and rotation, explicitly for the FFT and implicitly for the Fast Gabor Filtering method. Memory usage can be thereby substantially reduced and the performance of the encoding algorithm vastly improved for a large number of translations.

For the neural circuits module, we use one CUDA thread to perform the encoding of a single neuron. That is, each CUDA thread reads in the corresponding filtering output, integrates it and determines the spike time. To achieve a higher precision in spike timing, we nominally perform a linear interpolation between two consecutive video frames; the time of spike occurrence is exactly the time when the integrated filter output hits the threshold. In other words, the spike time is accurate under the linear interpolation assumption between two consecutive video frames. The spikes generated during the duration of the video segment are then stored to the disk.

There are multiple ways to store spikes. One way is to store the spike time relative to the beginning of the visual stimulus. However, such an approach is vulnerable to floating point overflow as the spike times become larger and larger. An alternative way is to store the spikes in the format of inter-spike intervals. This approach avoids the overflow problem, but lacks a time reference. Decoding of any part of the video would require to read and add up all the spikes from the beginning of the encoding process until the point of interest. The latter is not favorable when the time instance is large. In our algorithm, we store the spike times relative to an evolving time line, e.g., the spike times relative to the beginning of the second. This way, we can avoid accumulation operations of the spike times, since the stored time will always be in $[0, 1)$, and one can read out spikes starting from any second.

The pseudo code of the encoding algorithm is provided in Algorithm 1.

```
Input: Visual stream or Video data
Output: Spike trains
Prepare Gabor filters;
while not reaching the end of the video do
    if head node then
        Read in next $N$ frames of video;
    end
    Broadcast video segment to all nodes;
    Transfer video segment to GPU memory;
    for $i$=1:$N$ do
        Convolve filter with $i$th frame;
        Extract filtered value at designed translation points;
    end
    foreach CUDA thread $j$ do
        Compute $j$th IAF neuron spike times;
    end
    Gather all spike times at head node;
    if head node then
        Store spike time to file;
    end
end
```

Algorithm 1: The video TEM algorithm.

```
Input: Spike trains
Output: Reconstructed video
foreach GPU node $k$ do
    foreach Receptive field $i$ do
        foreach Receptive field $j$ do
            Compute $\sum_{m_x=-M_x}^{M_x} \sum_{m_y=-M_y}^{M_y} d^i_{m_x,m_y} \overline{d^j_{m_x,m_y}}$ ;
        end
    end
    foreach CUDA thread (i,j) do
        Compute $[\mathbf{G_k}]_{ij}$;
    end
    foreach CUDA thread $i$ do
        Compute $[\mathbf{q_k}]_i$;
    end
end
All GPUs collectively simulate recurrent neural network to obtain $[\mathbf{c_k}]$;
foreach GPU node $l$ on the diagonal do
    Reconstruct partial sum $\sum_i [\mathbf{c_l}]_i \phi_i$;
    Gather all partial reconstructions at head node;
end
if head node then
    Sum up all parts of the reconstructions;
end
```

Algorithm 2: The video TDM algorithm using a recurrent neural network.

5.2. Decoding on a GPU cluster

One of the main advantages of using a recurrent neural network for decoding is that it is straightforward to scale the system to multiple computing units. The GPU's intrinsically parallel architecture is a perfect fit. Here, we discuss how the decoding is realized on multiple GPUs, whose hosts are connected using a switch fabric and peer-to-peer communication is accomplished through the Message Passing Interface (MPI).

The need for scaling up the size of the decoding algorithm is driven by the size of $\mathbf{G}$, since the entries of $\mathbf{G}$ have to be stored in memory during the entire decoding process. Since the memory requirement for $\mathbf{G}$ is large, we divided $\mathbf{G}$ into blocks. Each block is mapped into a single GPU with enough memory to store all its entries (2.5 GB for the current GPU hardware). Since $\mathbf{G}$ is symmetric, only the upper diagonal blocks of $\mathbf{G}$ are used. Therefore, $\mathbf{G}$ is divided into blocks of size of about $25,000 \times 25,000$ (not necessarily square matrices) and is computed and stored in a distributed fashion on all GPUs in the cluster.

$\mathbf{G}$ can be efficiently computed when the receptive fields are separable. In this case, the computation of each entry of $\mathbf{G}$ can be separated into a spatial and a temporal component. The spatial component is completely independent of the spike times, and thus it can be computed *a priori*. The temporal component can be computed using one CUDA thread per entry. The spatial component can be viewed as weights and be applied afterwards. The vector $\mathbf{q}$ is also straightforward to compute, using one CUDA thread per entry.

After $\mathbf{G}$ and $\mathbf{q}$ are computed, the RNN system of differential equations is evaluated according to the forward Euler method. A critical step in computing the differential equations is the matrix vector multiplication of $\mathbf{G}$ and $\mathbf{c}$. This is done in two steps. First, on each GPU, a local matrix-vector multiplication is performed with a block of $\mathbf{G}$ and the corresponding segment of $\mathbf{c}$. Second, the results of the local matrix vector multiplication are gathered (using MPI) into the GPUs that store the diagonal blocks of $\mathbf{G}$, summed together, and the solution at the current time is updated. Then, the solution is broadcasted to corresponding blocks, again through MPI, before the next iteration is performed. The differential equations are simulated either for a fixed amount of computation time or until the gradient of the cost function is smaller than a certain threshold. In practice, we found that a fixed amount of time works well enough. The approximate solution $\mathbf{c}$ of the output of the circuit is then used to reconstruct the signal based on (7). The pseudocode of the decoding algorithm is given in Algorithm 2.

5.3. Volume Stitching

Even if the stimulus reconstruction is performed on a large scale GPU cluster it is still necessary, due to the massive number of neurons and massive number of spikes generated in encoding for a large aperture stimulus, to divide the stimulus into smaller volumes and focus the reconstruction on each volume. After all volumes are reconstructed, they can be stitched together using a stitching algorithm following a procedure similar to the one in (Lazar et al., 2008). We now describe how each stimulus segment is reconstructed and provide the stitching algorithm for the complete recovery of the visual stimulus.

We first divide the stimulus into pieces of fixed size, overlapping volumes, as illustrated in Fig. 4. We denote the length of each volume in x, y and t direction as J_x, J_y and J_t, respectively. The length of the overlapping part of two adjacent volumes in the x, y and t directions is denoted by O_x, O_y and O_t, respectively, with $2O_x < J_x, 2O_y < J_y, 2O_t < J_t$. We define by $(V_{k,l,m}), k, l, m \in \mathbb{Z}$, the volume segment localized in

$$(k(J_x - O_x), (k+1)(J_x - O_x) + O_x] \times (l(J_y - O_y), (l+1)(J_y - O_y) + O_y] \\ \times (m(J_t - O_t), (m+1)(J_t - O_t) + O_t],$$

and by $\hat{I}_{k,l,m}(x, y, t)$ the stimulus reconstruction based on spikes localized in the volume segment $V_{k,l,m}$. $\hat{I}_{k,l,m}$ is obtained by the decoding procedure described in Section 5.2 and only takes into account the spikes localized in the volume $V_{k,l,m}$ that satisfy the following conditions (i) they are generated by neurons whose receptive fields are centered inside the spatial domain of $V_{k,l,m}$ and (ii) the spike times are inside the temporal domain of $V_{k,l,m}$ or are exactly the closest spikes before or after the temporal segment.

The stimulus reconstructions of the individual volume segments are stitched together with a simple shifting windows algorithm. We define the windows

$$w_{k,l,m} = w^x_k \cdot w^y_l \cdot w^t_m, \tag{21}$$

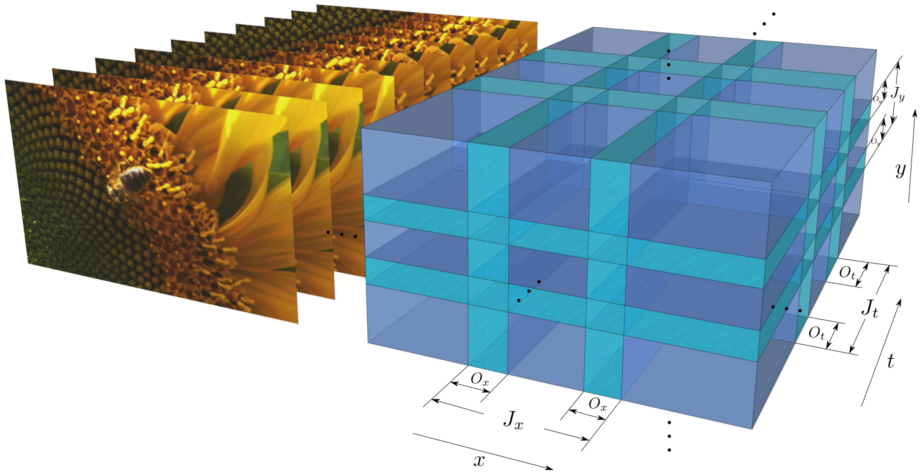

Figure 4: (left) Space-time natural video sequence. (right) Division of the space-time video sequence into fixed sized, overlapping volume segments. The brighter blue color indicates the overlapping adjacent volume segments.

where

$$w_k^x = \begin{cases} 0, & x \notin (k(J_x - O_x), k(J_x - O_x) + J_x], \\ \theta_k^x, & x \in (k(J_x - O_x), k(J_x - O_x) + O_x], \\ 1, & x \in (k(J_x - O_x) + O_x, (k+1)(J_x - O_x)], \\ 1 - \theta_{k+1}^x, & x \in ((k+1)(J_x - O_x), (k+1)(J_x - O_x) + O_x], \end{cases} \tag{22}$$

with θ_k^x is an appropriately chosen function. An example is given in equation (24) in section 6.1. The functions w_l^y and w_m^t are similarly defined. It is easy to see that the defined window functions form a partition of unity.

The overall visual stimulus reconstruction is therefore given by

$$\hat{I}(x, y, t) = \sum_{k,l,m \in \mathbb{Z}} w_{k,l,m} \hat{I}_{k,l,m}(x, y, t). \tag{23}$$

Since a fixed sized volume is used for stitching, the shape of the window functions are all the same and they can be pre-computed. Thus, the overall visual stimulus reconstruction can be obtained by reconstructing the visual stimulus in each volume segment, multiplying these with the stored window functions, and then stitching the latter in an 'overlap-add' fashion. The pseudo-code for the volume stitching is provided in Algorithm 3.

```
Input: Reconstructions in each volume $\hat{I}_{k,l,m}$
Output: Complete reconstruction $\hat{I}$
Compute the volume window function;
while not reaching the end of the video do
    Perform $\sum_{k,l,m\in\mathbb{Z}} w_{k,l,m}\hat{I}_{k,l,m}(x,y,t)$ in ‘overlap-add’ fashion;
end
```

Algorithm 3: The volume stitching algorithm.

6. Examples

In this section we provide a complete example of a stimulus encoding with a video TEM and decoding with a video TDM. The encoding with neurons with deterministic thresholds and its decoding will be illustrated first. Then, the result of a noisy case will be presented.

In our implementation we used the Python programming language; for the GPU part of the implementation we employed PyCUDA (Klöckner et al., 2009).

The visual stimulus of interest was an nHD format (640×360 pixels) color video sequence defined in the domain $\mathbb{D}^2 = [-18, 22] \times [-9.5, 13]$. We decomposed the color video into RGB components, resulting in three monochrome sequences. Each color component was preprocessed such that most of the energy in the spatial spectrum was within 4Hz in both directions. Temporally, the video was 10 second long and was stored at 25 frames per second. Each pixel was filtered with a 10 Hz lowpass filter and upsampled by a factor of 4, in order to improve the accuracy of the analog integration. The preprocessed video was viewed as the original visual stimulus that was encoded by the video TEM.

6.1. Example of Video TEM with IAF Neurons

We now describe an example of a video TEM realized with Gabor filters and IAF neurons with deterministic thresholds that encodes the aforementioned video. The following encoding procedure was repeated for all three color components of the video.

The visual receptive fields used in the encoding act spatially only, i.e., $D^j(x, y, t) = D^j_S(x, y)\delta(t)$, where $\delta(t)$ is the Dirac delta function. The spatial receptive fields were a family of Gabor filters, derived from (2). We used 5 dilations, with $\alpha = 2\left(\frac{1}{2}\right)^m, m = 0, 1, 2, 3, 4$. Translation parameters were provided by a Cartesian lattice for each dilation, with spacing between two neighboring translations 2.5, 1.625, 1, $11/16$, 0.5, respectively. In addition, 8 rotations were used, with $\theta = l\theta_0, l = 0, 1, \cdots, 7$, where $\theta_0 = 7\pi/8$. Finally, the real and imaginary parts are viewed as two receptive fields.

Each receptive field output was then fed into an IAF neuron with parameters $\kappa = 1.0, \delta = 0.03, b = 0.8$. The initial conditions of the membrane potential of the neurons were uniformly drawn from $[0, \delta)$. In all, a total of $112, 208$ neurons with Gabor receptive fields were used for each color component.

We first tested the encoding using the matrix-matrix multiplication method in simulating receptive field filtering. Encoding was performed on a cluster of 16 Tesla M2050 GPUs. The total number of spikes generated in the duration of the video were

$(30,329,137)$, $(30,221,717)$ and $(30,045,454)$ for the R, G and B components, respectively. It took approximately 110 seconds to encode one color component of the 10 seconds video. This is about 33 times slower than real-time for the whole color video. A more detailed timing revealed that the filtering consumed 95% of the total encoding time, while the distribution of the video data among processors and the operation of IAF neurons took about 2.5% each. It can be seen that the encoding bottleneck was due to the receptive field filtering.

Second, as mentioned in Section 5.1, we replaced the brute force matrix-matrix multiplication method in filtering with the Fast Gabor Filtering technique. In the process we only used 5 GPUs, each responsible for all the receptive fields with the same dilation. For dilation with parameter 2, however, we kept the matrix-matrix multiplication method, since the Fast Gabor Filtering cannot handle Gabor filters with very large support and the number of translations required for the largest dilation is small. In general, a trade-off needs to be considered between the two methods, in order to find the most efficient approach for different visual stimuli. The encoding time using the Fast Gabor Filtering method was also around 110 seconds. Therefore, we achieved the same performance using less than $1/3$ of the GPUs in the first test.

In decoding, the RGB components were reconstructed separately and then combined together. For each component, the reconstruction went as follows. We performed the volume stitching method described in Section 5.3. with $J_x = J_y = 13$, $O_x = 4$, $O_y = 3.5$, $J_t = 0.35$, $O_t = 0.05$. The function θ_k^x was given by

$$\theta_k^x = \sin^2\left(\frac{\pi}{2}\cdot\frac{x-k(J_x-O_x)}{O_x}\right), \tag{24}$$

and the functions θ_l^y, θ_m^t were similarly defined. We chose the order of the visual space as

$$M_x = 72, M_y = 72, M_t = 10,$$
$$\Omega_x = 8\pi, \Omega_y = 8\pi, \Omega_t = 20\pi.$$

Therefore, the reconstruction of each block was embedded in the domain $\mathbb{T}\times\mathbb{D}^2 = [0,1]\times[-9,9]\times[-9,9]$, and thereby, the periodicity of the stimuli in RKHS does not appear in the recovery.

Typically, there were around $220,000$ to $240,000$ spikes within each segment in the block. We employed 55 Tesla M2050 GPUs to reconstruct each segment. This corresponds to dividing the $\mathbf{G}$ matrix into 10×10 blocks, of which the 55 upper diagonal blocks were explicitly computed. The recurrent neural network described in Section 4.1 was used, with $\alpha = 20$ and $3,000$ time steps with 10^{-4}s each. The run time of reconstructing each segment was about 4 minutes, simulating the neural network for 0.3 second. Therefore, for each spatial block, the output of the neural circuit can provide close to real-time reconstruction. The total run time for the reconstruction of color video was about 57 hours.

After stitching both spatially and temporally, we obtained a high quality reconstruction for all three components. To evaluate the quality of the reconstruction, we only considered the central 620×340 region, where the boarder of 10 pixels were not taken into account due to boundary errors. The Signal-to-Noise Ratio (SNR) for R, G, B components were 31.85 [dB], 30.59 [dB], 26.81 [dB], respectively. The mean SSIM index (Wang et al., 2004) across all frames were also computed. They were 0.9963, for R, 0.9970 for G and 0.9973 for B. RGB components were combined to visualize the complete recovery. One of the frames of the original visual stimulus and the corresponding reconstruction frame is shown in Fig. 5. The complete video can be found in the supplemental material [supplementary video 1]. The high quality reconstruction shows both the effectiveness of the massively parallel decoding algorithm as well as the faithful representation of the visual stimulus by the massively parallel video time encoder.

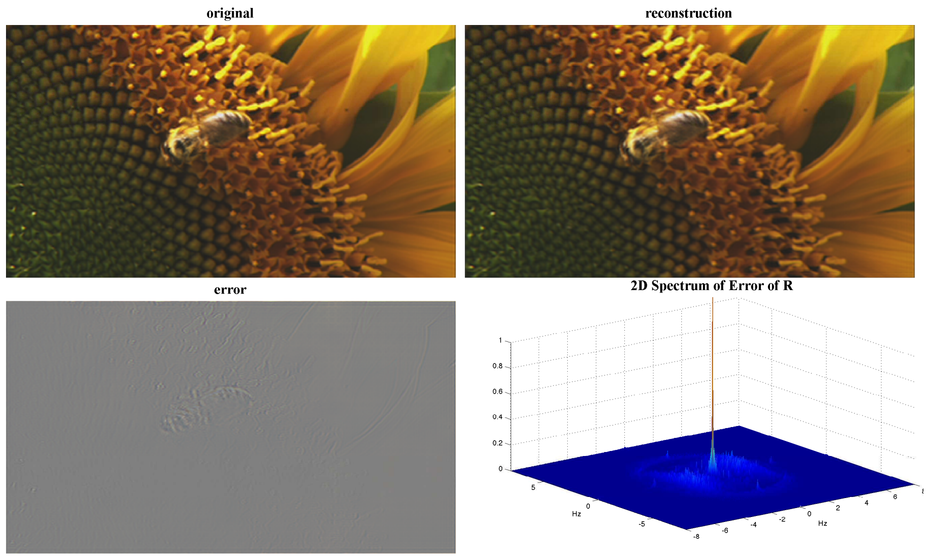

Figure 5: Reconstruction of a visual stimulus encoded with a Video TEM with IAF neurons with deterministic thresholds. Original frame (top left), its reconstruction (top right), the error (bottom left) and the 2D spectrum of the error in R component (bottom right).

Furthermore, both the video time encoding and video time decoding architectures described in the example are highly scalable due to their massive parallelism. Given more computing resources, one can either reduce the simulation time by distributing computation to more nodes, or increase the aperture of the video.

6.2. Example of a Video TEM with IAF Neurons with Random Thresholds

We proceed to an example of a video TEM with neurons with random thresholds.

The parameters of the neurons and their Gabor receptive fields were chosen to be the same as in the previous example, except for the values of the thresholds of the neurons. The thresholds of jth neuron were drawn from the Gaussian distribution $\mathcal{N}\left(\delta^j, (\sigma^j)^2\right)$, where $\delta^j = 0.03$, and σ^j was drawn from a Gaussian distribution $\mathcal{N}(10^{-4}, 10^{-10})$. The initial conditions of the membrane potential of the neurons were drawn from a uniform distribution on the interval $[0, \delta_0^j]$, where δ_0^j is the threshold of the first spike of the jth neuron.

Again, encoding was performed on a cluster of 16 Tesla M2050 GPUs. $(30, 330, 904)$, $(30, 223, 666)$ and $(30, 047, 093)$ spikes were fired in the 10 seconds for R, G and B components, respectively.

Decoding followed the same spatial stitching procedure as in the previous example. Again 55 GPUs were employed in the decoding. The smoothing parameters were set such that $n\lambda$ was fixed for each color components. They were $n\lambda = 0.05, 0.01, 1.0$ for R, G, B, respectively. The recurrent neural network used and the parameters were the same as in Section 6.1.

We obtained the reconstructed visual stimulus after about 57 hours of simulation. One of the frames of the original visual stimulus and the corresponding reconstruction frame are shown in Fig. 6. SNR for the R, G, B components were 18.21 [dB], 15.50 [dB], 9.18 [dB], respectively. The mean SSIM index were 0.851, 0.829 and 0.850 for R, G, B components, respectively. Note that the SNR of the blue component reconstruction is much smaller. This is due to the fact that the blue component of the visual stimulus has smaller overall intensity, but the amount of noise in the spikes were the same for all the three components. As a comparison, we also performed the reconstruction without regularization, *i.e.*, $n\lambda = 0$. The resulting SNR of the reconstructions were 18.18 [dB], 15.45 [dB], 8.52 [dB], and the mean SSIM index were 0.844, 0.827 and 0.793, respectively, for R, G and B components. The complete reconstructed video can be found in the supplemental material [supplementary video 2].

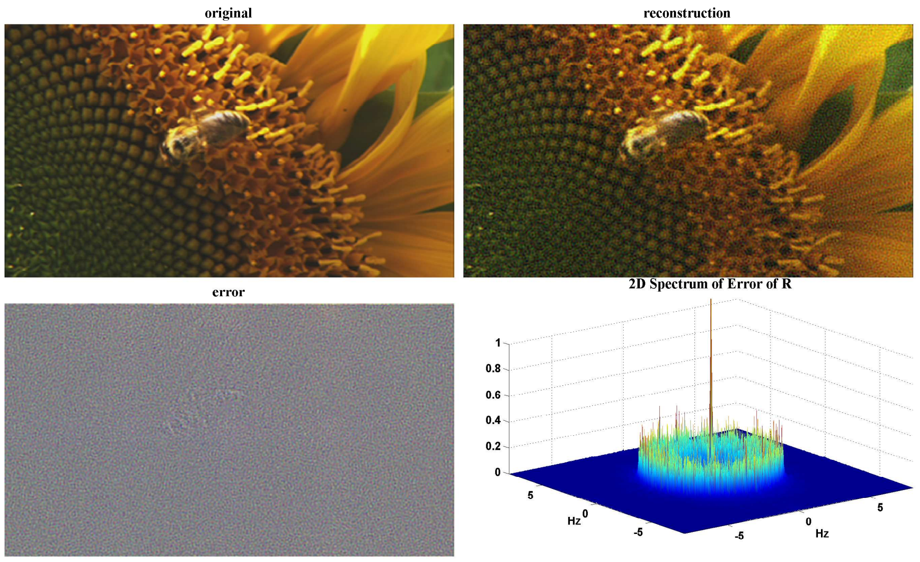

Figure 6: Reconstruction of a visual stimulus encoded with a Video TEM with IAF neurons with random thresholds. Original frame (top left), its reconstruction (top right), the error (bottom left) and the 2D spectrum of the error in the R component (bottom right).

7. Discussion and Conclusions

In the present report, a massively parallel architecture of Video TEMs and Video TDMs was described. The massive parallelism of the Video TEM was intrinsically implemented by a population of IAF neurons. The required degree of parallelism of the Video TDMs was achieved with the proposed massively parallel, highly scalable and easy to implement analog VLSI recurrent neural network circuits. An extension of the Video TDM to recover stimuli encoded with Video TEMs with neural circuits with random thresholds was also presented. We described the implementation of the massively parallel Video TEM and Video TDM on a GPU cluster in the Python language, and demonstrated their performance for large aperture visual stimuli.

The reconstruction of stimuli encoded with Video TEMs was formulated as an optimization problem. Consequently, a large variety of recurrent neural networks can be employed for devising recovery algorithms. The class of optimization problems can also be extended to include reconstruction constraints. For example, nonlinear optimization problems with equality or inequality constraints can be efficiently dealt with (Xia & Wang, 2005; Xia et al., 2008). Additional constraints can be imposed on the reconstruction problem. As an example, a sparse solution can be obtained by minimizing the l_1 norm. The associated optimization problem can be formulated as a linear program and time domain linear programming circuits can be employed for real-time implementations (Cruz-Albrecht & Petre, 2010).

The complexity of encoding of visual stimuli is given by the number of neurons used. As we have seen the size of the recurrent neural network is given by the number of spikes to be decoded rather than by the number of neurons that generate these spikes. Therefore, a massive number of neurons is required to process the information encoded by a relatively small number of neurons. This observation may explain why there is an explosively larger number of spiking neurons in V1 than in the retina.

The stitching algorithm presented here scales the original reconstruction method to large aperture visual stimuli. Noteworthy is that the algorithm employs window functions acting as weights on each volume segment. While the Gabor filters with the same dilation and translation but with different orientations may form hypercolumns in the visual cortex (Hubel & Wiesel, 1962), the spatial stitching windows can be interpreted as modeling synaptic weights between the spatially organized hypercolumns. The temporal stitching windows may be implemented by a feedback mechanism whereby the window functions are viewed as modeling synaptic weights of the neurons in the feedback loop.

The results presented here offer a number of interesting avenues for further research. How to model the computation of the entries of the matrix $\mathbf{G}$ and the vector $\mathbf{q}$ with processes native to dendritic trees will be described elsewhere.

8. Acknowledgments

The research reported here was supported by AFOSR under grant # FA9550-09-1-0350 and, in part, by a grant of computer time from the City University of New

York High Performance Computing Center under NSF Grants CNS-0855217 and CNS-0958379. We thank Prof. Bertram Shi from Hong Kong University of Science and Technology for providing us with the Fast Gabor Filtering CUDA code written in C++.

References

Bezhaev, A. Y., & Vasilenko, V. A. (2001). *Variational Theory of Splines*. New York: Kuluwer Academic / Plenum Publishers.

Cichocki, A., & Unbehauen, R. (1993). *Neural Networks for Optimization and Signal Processing*. John Wiley & Sons.

Cruz-Albrecht, J., & Petre, P. (2010). Pulse Domain Linear Programming Circuit. US Patent 7724168.

Gestri, G., Mastebroek, H., & Zaagman, W. (1980). Stochastic Constancy, Variability and Adaptation of Spike Generation: Performance of a Giant Neuron in the Visual System of the Fly. *Biological Cybernetics*, *38*, 31–40.

Harris, J., Xu, J., Rastogi, M., Singh-Alvarado, A., Garg, V., Principe, J., & Vuppamandla, K. (2008). Real Time Signal Reconstruction from Spikes on a Digital Signal Processor. In *IEEE International Symposium on Circuits and Systems* 18-21 (pp. 1060–1063).

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology*, *160*, 106–152.

Kay, K., Naselaris, T., Prenger, R., & Gallant, J. (2008). Identifying natural images from human brain activity. *Nature*, *452*, 352–356.

Kinget, P. R., Lazar, A. A., & Tóth, L. T. (2005). On the robustness of the vlsi implementation of a time encoding machine. In *IEEE International Symposium on Circuits and Systems*.

Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., & Fasih, A. (2009). PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *submitted*, .

Lazar, A., & Tóth, L. (2004). Perfect Recovery and Sensitivity Analysis of time Encoded Bandlimited Signals. *IEEE Transactions on Circuits and Systems-I: Regular Papers*, *51*, 2060–2073.

Lazar, A. A. (2006). A Simple Model of Spike Processing. *Neurocomputing*, *69*, 1081–1085.

Lazar, A. A., & Pnevmatikakis, E. A. (2011). Video time encoding machines. *IEEE Transactions on Neural Networks*, *22*, 461–473.

Lazar, A. A., Pnevmatikakis, E. A., & Zhou, Y. (2010). Encoding natural scenes with neural circuits with random thresholds. *Vision Research*, *50*, 2200–2212. Special Issue on Mathematical Models of Visual Coding.

Lazar, A. A., Simonyi, E. K., & Tóth, L. T. (2008). An overcomplete stitching algorithm for time decoding machines. *IEEE Transactions on Circuits and Systems-I: Regular Papers*, *55*, 2619–2630.

Lazar, A. A., & Zhou, Y. (2011). Realizing Video Time Decoding Machines with Recurrent Neural Networks. In *Proceedings of the International Joint Conference on Neural Networks*. San Jose, CA: IEEE.

Miyawaki, Y., Uchida, H., Yamashita, O., Sato, M., Morito, Y., Tanabe, H., Sadato, N., & Kamitani, Y. (2008). Visual Image Reconstruction from Human Brain Activity using a Combination of Multiscale Local Image Decoders. *Neuron*, *60*, 915–929.

Penrose, R. (1955). A Generalized Inverse for Matrices. *Proc. Cambridge Philos. Soc.*, *51*, 406–413.

Reich, D., Victor, J., Knight, B., Ozaki, T., & Kaplan, E. (1997). Response Variability and Timing Precision of Neuronal Spike Trains in Vivo. *Journal of neurophysiology*, *77*, 2836–2841.

Stanley, G., Li, F., & Dan, Y. (1999). Reconstruction of Natural Scenes from Ensemble Responses in the Lateral Geniculate Nucleus. *Journal of Neuroscience*, *19*, 8036–8042.

Wang, X., & Shi, B. E. (2010). GPU Implementation of Fast Gabor Filters. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (pp. 373–376). Paris.

Wang, Z., Bovik, A., Sheikh, H., & Simoncelli, E. (2004). Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, *13*, 600–612.

Xia, Y., Feng, G., & Wang, J. (2008). A Recurrent Neural Network for Solving Nonlinear Optimization Problems with Inequality Constraints. *IEEE Transactions on Neural Networks*, *19*, 1340–1353.

Xia, Y., & Wang, J. (2005). A Recurrent Neural Network for Solving Nonlinear Convex Programs Subject to Linear Constraints. *IEEE Transactions on Neural Networks*, *16*, 379–386.