



Proposal full title:

INSIGHT: Darwinian Neurodynamics

Proposal acronym:

INSIGHT

Type of funding scheme:

Collaborative project

FP7-ICT-2011-C FET Open

D 1.2

Models of Neuronal Replicators

Name of the coordinating person:

prof. Eörs Szathmáry

Coordinator email: **szathmary.eors@gmail.com**

Coordinator phone: **+49 89 45209 35-30**

Coordinator fax: **+49 89 45209 35-31**

Revisions Tables

Due delivered date	28/02/2014	Actual delivered date	31/03/2014
Lead beneficiary	PARMENIDES		
Beneficiaries involved	QMUL, SUSSEX		
Authors	Chrisantha Fernando, Jeff Shim, James Thorniley, Phil Husbands, Andy Philippides, Eörs Szathmary.		
Dissemination level	PU	Nature	R

REV	Work performed	Reviewers	Beneficiary
1	Production of the deliverable	Chrisantha Fernando	QMUL
2	Contribution and revision	Eörs Szathmary	PARMENIDES
3	Formatting	Roberta Modolo	IN

Table of contents

1	Executive summary	3
2	From Blickeys to Synapses: Inferring Temporal Causal Networks by Observation.....	3
3	STDP based Population Expectation-Maximization Algorithm Results.....	4
3.1	Implementation of a Spiking EM (SEM) Neural Network	4
3.2	Mechanisms for Replication of SEM networks	9
3.3	ITDP controlled SEM network ensemble learning	20
4	LSM based Path Evolution Algorithm Results	23
5	Chaotic Exploration and Darwinian Neural Dynamics	24
6	Related Work on Neural Dynamics	32
7	Simulation of WP2 Neurophysiology Experiment	33
7.1	GPGPU Simulation evaluation	34
7.2	STDP causal inference results.....	36
8	SORN based Path Evolution Algorithm Results.....	41
8.1	Re-implementation and analysis of the SORN model	41
8.2	Reproducing pattern of a SORN model	44
8.3	Neuronal genetic algorithm in the SORN network.....	46
9	References.....	48

1 Executive summary

This work-package focuses on computational neuroscience models of a variety of neuronal replicators. Several models of neuronal replicators have been investigated during the past 12 months. The most complete was produced by Chrisantha Fernando at QMUL and published in the Journal Cognitive Science in December 2013, and is described in Part 2. It extends considerably the work of Szathmáry and Fernando our first PLoS article on Darwinian neurodynamics in 2008 by removing the need for specific error correction neurons, and introducing 3 plasticity rules that together work to undertake causal inference.

The second line of research comes from Sussex where Dr. Shim has been developing models of spiking neural networks capable of carrying out the expectation maximization algorithm. He has added populations to this, by developing an unsupervised ensemble method for these spiking networks, to investigate whether population based EM can be more efficient and has investigated various (neural) mechanisms for replicating such networks. Shim has also worked on extending the path evolution algorithm to Maas' new liquid state machine design which contains coupled winner take all circuits. These results are described in sections 3 and 4. Work at Sussex has also involved an investigation of the relationship between embodied chaotic exploration and evolutionary neural dynamics and preliminary work on other forms of neural dynamics that may be useful in developing neurally implemented 'macro mutation operators' in replicator systems (effecting connectivity etc). These investigations are described in sections 5 and 6. Work by James Thorniley for this deliverable has concentrated on a simulation model of the neurophysiology experiments being worked on in WP 2. This work is described in section 7. It should be noted that work *at Sussex started 6 months late due to an unavoidable delay in staff recruitment (due to a delay in the signing of the Grant Agreement which was out of the control of the project members)*. However, progress has been good and a number of publications are expected over the coming months.

The third line of research comes from the Parmenides Foundation where Prof. Szathmáry et al have attempted "domesticating" the SORN model in order to apply it for embedded path evolution and related neuronal solutions in the coming two years of the project.

2 From Bliickets to Synapses: Inferring Temporal Causal Networks by Observation.

<http://onlinelibrary.wiley.com/doi/10.1111/cogs.12073/abstract>

Please refer to the pdf of the paper attached in appendix A.

3 STDP based Population Expectation-Maximization Algorithm Results

This section describes investigations into the use of populations and replication in a spiking neural model capable of performing Bayesian computations.

3.1 Implementation of a Spiking EM (SEM) Neural Network

A body of experimental data proposes that the brain uses principles of Bayesian inference for processing sensory processing in order to solve cognitive tasks such as reasoning and for producing adequate motor outputs (Rao et al. 2002, Doya et al 2007). Bayesian inference is suggested as an important mechanism for probabilistic perception (Fiser et al 2010) in which hidden causes (e.g. the categories of objects) underlying noisy and potentially ambiguous sensory inputs have to be inferred. Learning using Bayesian inference updates the probability estimate for a hypothesis (a posterior probability distribution for hidden causes) as additional evidence is acquired.

Recently, a spike-based neuronal implementation of Bayesian computation has been proposed by Nessler et. al. (2013). Their feedforward network architecture implements Bayesian computations using population-coded input neurons and a soft winner takes all (WTA) output layer, in which internal generative models are represented implicitly through the synaptic weights to be learnt, and the inference for the probability of hidden causes is carried out by integrating such weighted inputs and competing for firing in a WTA circuit. The synaptic learning uses a spike-timing dependent plasticity (STDP) rule which has been shown to effectively emulate the Expectation Maximization (EM) algorithm. Refer to (Nessler et al 2013) for a more detailed description of the current neural model and the formulations which explain its relation to EM algorithm. The Nessler model forms the basis of the work described in this section so our interpretation of it, used in our reimplementation and extensions of the model, it is briefly described below.

A WTA circuit consists of k stochastically firing neurons. The firing of each neuron z_k is modelled as an inhomogeneous Poisson process of instantaneous firing rate $r_k(t)$.

$$r_k(t) = e^{u_k(t)}$$

$$u_k(t) = w_{k0} + \sum_{i=1}^n w_{ki} y_i(t) - I(t)$$

$$I(t) = O_{inh} - (A_{inh} + O_{inh}) \cdot e^{(t_f - t) / \tau_{inh}}$$

Where $u_k(t)$ is a membrane potential which sums up the EPSPs from all presynaptic inputs neurons (y_i ($i=1, \dots, n$)) multiplied by the respective synaptic weight w_{ki} . The variable w_{k0} is neuronal excitability, and $I(t)$ is the input from the global inhibition neuron to the WTA circuit. The EPSP evoked by the i th input neuron is modelled as a double exponential curve which has both fast

rising (τ_f) and slow decaying (τ_s) time constants. At each time instance, EPSP amplitudes are summed over all presynaptic spike times (t_p) to become $y_i(t)$ for i 'th input at time t .

$$y_i(t) = A_{EPSP} \sum_{t_p} e^{\frac{t_p-t}{\tau_s}} - e^{\frac{t_p-t}{\tau_f}}$$

The scaling factor A_{EPSP} is set as a function of the two time constants in order to ensure that the peak value of an EPSP is 1. Whenever one of the neurons in the WTA circuit fires at t_f , $I(t)$ gives a strong negative pulse (amplitude of A_{inh}) to the membrane potential of all z neurons, which exponentially decays back to its resting value (O_{inh}) with a certain time constant (τ_{inh}). Therefore, $I(t)$ determines the overall firing rate of WTA circuit as well as assessing refractory period to the fired neuron. The combined firing activity of all z neurons in WTA circuit can be expressed by the sum of K independent Poisson processes.

$$R(t) = \sum_{k=1}^K r_k(t)$$

In an infinitesimal time interval $[t, t+dt]$, we assume only one neuron z_k fires. Thus a firing event of z_k can be thought as a sample drawn from the conditional probability $q_k(t)$ which is written as a fraction of its firing rate,

$$q_k(t) = \frac{r_k(t)}{R(t)} = \frac{e^{u_k(t)}}{\sum_{j=1}^K e^{u_j(t)}}$$

The relative firing probability $q_k(t)$ of an output neuron z_k is equivalent to the posterior distribution of hidden cause k , given the evidence represented by input activation vector $\mathbf{y}(t)$. By following Baye's rule, we can identify the network dynamics as a posterior probability which is expressed using prior and likelihood as

$$p(k | \mathbf{y}(t), \mathbf{w}) = q_k(t) = \frac{e^{w_{k0}} \cdot e^{\sum w_{ki} y_i}}{\sum_{j=1}^K e^{w_{j0} + \sum w_{ji} y_i}} \Rightarrow \frac{p(k | \mathbf{w}) p(\mathbf{y} | k, \mathbf{w})}{p(\mathbf{y} | \mathbf{w})}$$

In order to follow Bayes model, this formulation holds under the normalisation constraint,

$$\sum_{k=1}^K e^{w_{k0}} = 1, \quad \forall k : \sum_{i \in G_j} e^{w_{ki}} = 1, \quad j = (1, 2, \dots, N)$$

where G_j represents a set of all possible values that an instantaneous input evidence x_j can have, which is equivalent to the discretised value of each input variable. This means that an input evidence x_j for a feature j of observed data is encoded as a group of neuronal activations y_i . If the

set of possible value of x_j consists of m values $G_j=[v_1, v_2, .. v_m]$, the input x_j is encoded using m input neurons. Therefore, if input data is given as a N ($j=1, \dots, N$) dimensional vector, the total number of input neurons is mN .

Synaptic plasticity by STDP rule used in this model captures both the biological plausibility and the computational requirement for Bayesian inference. The STDP curve of LTP part follows the shape of EPSP at the synapses, which is similar to real biological plasticity, in that the backpropagating action potential from postsynaptic neuron interacts with the presynaptic EPSP arrived at the synapse. The magnitude of weight update depends on the inverse exponential of the synaptic weight itself to prevent unbounded weight growth. The synaptic update is formulated by,

$$\Delta w_{ki} = y_i(t) \cdot c \cdot e^{-w_{ki}} - 1$$

where $y_i(t)$ is the sum of EPSPs evoked by all presynaptic spikes, and c (which is set to 5 throughout the experiment) is a constant which determines the upper bound of synaptic weight. The LTD part was set to decrease by a constant amount of 1. Given the EPSP caused by presynaptic spikes, the synaptic update occurs only at the moment of postsynaptic neuron firing with the learning rate η . This plasticity rule can exhibit the stimulus frequency dependent behaviour of biological synapses which has been observed in biological STDP experiments, that the shape of plasticity curve (including the traditional hyperbolic curve of phenomenological model) depends on the repetition frequency of the delayed pairing of pre and postsynaptic stimulations.

In analogy to the plasticity of the synaptic weights, the self-excitability of the output neurons is used. The excitability w_{k0} the neuron z_k is increased whenever it fires ($z_k(t)=1$) as a function of inverse exponential of w_{k0} and is decreased by 1 if not firing ($z_k(t)=0$).

$$\Delta w_{k0} = z_k(t) \cdot e^{-w_{k0}} - 1$$

The update of w_{k0} is circuit-spike triggered, which means that the excitabilities of all WTA neurons are updated if one of the neurons fires. Therefore the value of w_{k0} converges to satisfy the normalization constraint as a prior probability which is necessary for the above mentioned Bayesian computation. The learning rates (η) of STDP and neuronal excitability are adaptively changed by variance tracking rule as in (Nessler et al 2013, Nessler et al 2008).

Unsupervised Image Classification using SEM

As in the original work (Nessler et al 2013), the reimplemented SEM neural network was tested on an unsupervised classification task for 2D images. In order to confirm the reproduction of previous results, we conducted an experiment using four classes of images whose pixels are generated by 2D Gaussian distributions with different mean locations. The circuit architecture is a single layer feedforward network having N input neurons and a WTA circuit which has four ($K=4$) output neurons. Actually the system is not completely unsupervised because the number of classes is provided. However blinding the class label makes the task challenging for the system to discriminate distinct hidden causes (Gaussian means) in a self-organised manner. This detailed

testing of the reimplementation was necessary as the model has a variety of complexities which are not always fully explained in the original papers.

As the first experiment to check the reimplementation, we used a training set of 28x28 binary images total having 784 features. The image was further reduced by abandoning less occupied pixels by preprocessing over the entire images in the training set (pixels being 'on' in less than 3% of the total image presentation were disabled). For images generated by Gaussian spread, the preprocessing reduced the number of pixels to $m=425$, hence $N=2m=850$ input neurons. The input was fed to the network by successively presenting an image from one class for a certain duration (T_{present}), followed by the resting period (T_{rest}) where none of the input neurons fire. During input presentation, one of the two input neurons that encode a pixel state fires with a constant rate. The network architecture and its parameters are shown in Figure 1A and Table 1.

Input firing rate	40Hz
STDP	$c=5$
Global inhibition	$A_{\text{inh}}=3100, O_{\text{inh}}=-650, \tau_{\text{inh}}=0.05$
EPSP	$\tau_s=0.015, \tau_f=0.001, A_{\text{EPSP}}=1.3$

Table 1. Network parameters

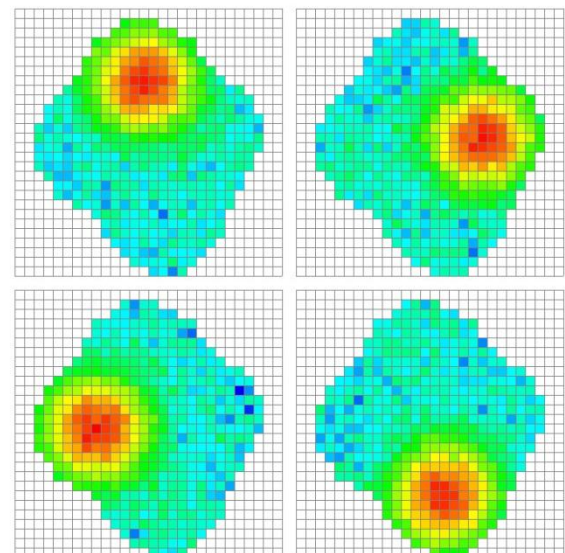
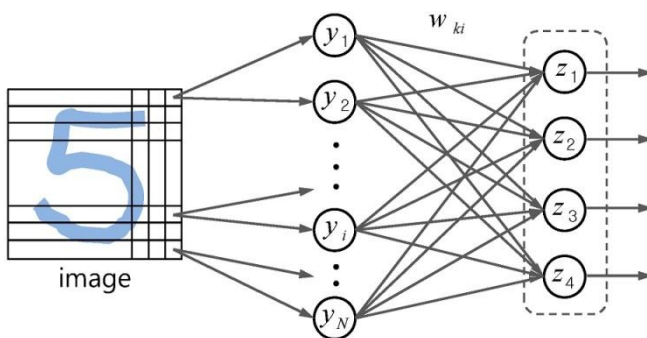


Figure 1A. SEM network architecture. Dashed box indicates the neurons comprise a WTA circuit with reciprocal inhibition.

Figure 1B. Weight maps of afferent connections for each output neuron.

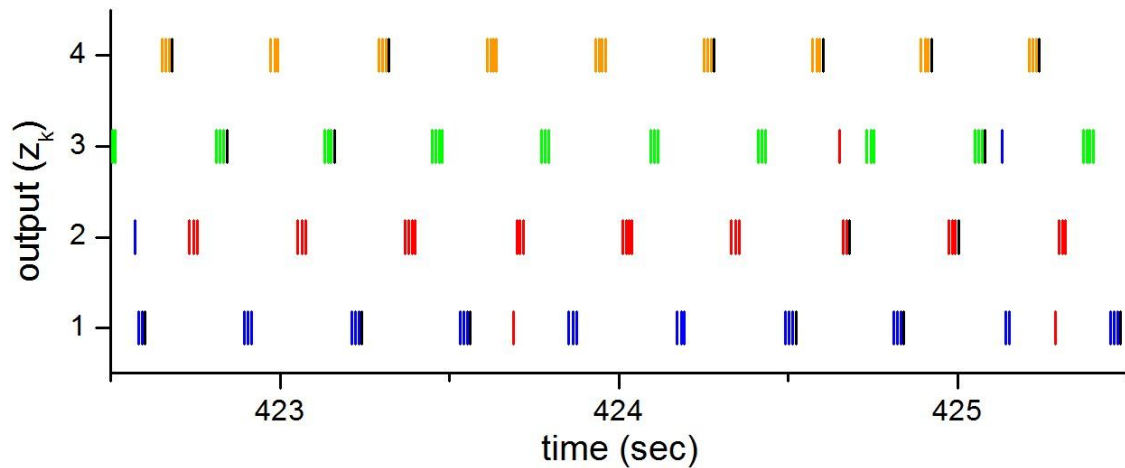


Figure 1C. Output spikes of learnt network. Different colours (red, green, blue, orange) represent the moment of the presentation of each correct class. In the experiment, each class is presented in repeating order. Spikes that fire during the resting period (no pattern presentation) were shown in black.

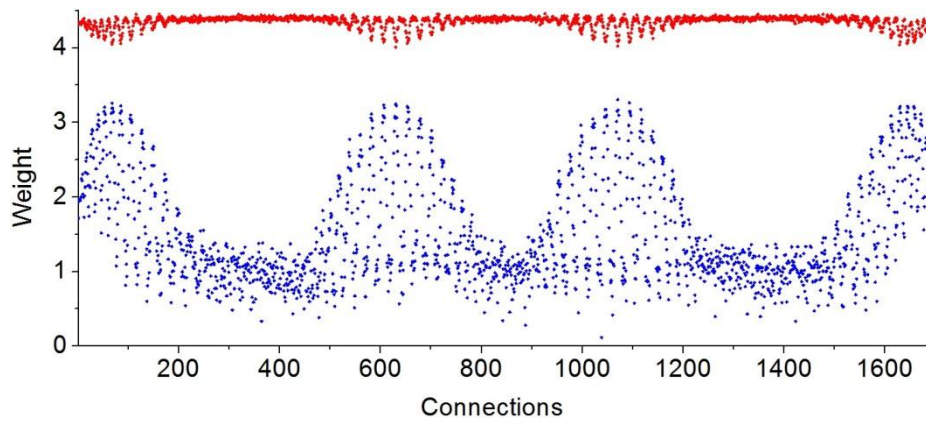


Figure 1D. Final weights after learning. Total 3400 connections are divided into two groups (ON: blue and OFF: red) and are superposed on the graph. Four ridges emerged by four hidden causes are clearly shown.

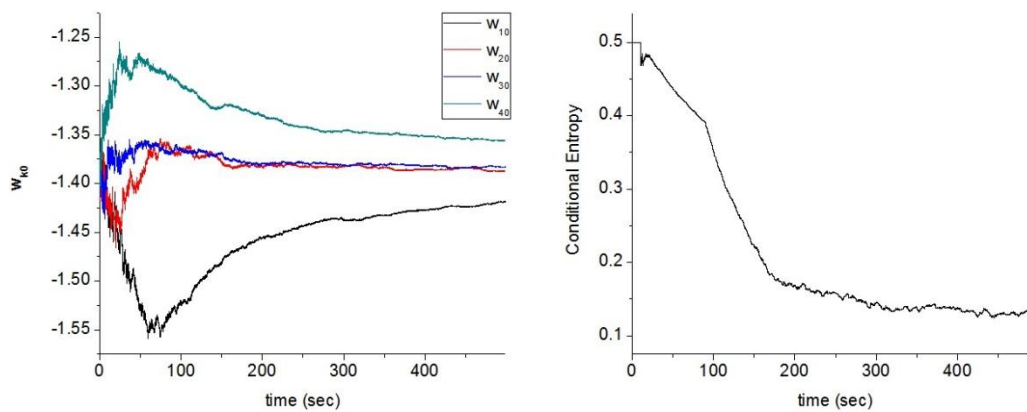


Figure 1E. Evolution of self-excitabilities and conditional entropy.

An example of the learning classification task is shown in Figure 1B-1E. One of the images in a class

is presented for 40ms followed by another 40ms of resting period. Different images generated from four classes are presented successively in a repeating order. Approximately 200 seconds after starting the simulation, the output neurons began to fire a series of ordered bursts almost exclusively to the hidden classes of each presented image as shown in Figure 1C. The weight maps for each output neuron (difference between ON and OFF weights for a pixel) shown in Figure 1B clearly represents four different hidden causes. A typical result illustrating emerged weights by learning is as shown in Figure 1D. For each input group G_j ($N_j=425$ pixels), the two weights in the group encode the probabilities of on and off of that pixel. In the case of the Gaussian image with standard deviation as used in this work, the probability that a pixel is on is less than a half, which means the ON weights are subjected to more synaptic depression than the OFF weights. Thus the whole set of weights tend to be segregated into two groups as shown in the graph. Self-excitabilities of output neurons also adapt according to the presentation probability of each class, encoding priors for hidden causes. The classification performance of the network is represented by calculating the normalised conditional entropy; $H(k|z)/H(k,z)$, where k is the correct class being presented and z represents the discrete random variable defined by the firing probabilities of output neurons. Low conditional entropy indicates that each output neuron fires predominantly for inputs from one class. In order to see momentary change of network performance, we calculated the conditional entropy within a moving time window of 1000 image presentations which is approximately 80 seconds in simulation time. Figure 1E shows the convergence of excitability and the decrease of conditional entropy over time. The experimental results successfully replicated those of the original papers and validated our reimplementation of the model.

3.2 Mechanisms for Replication of SEM networks

Using the successfully implemented SEM network model, we conducted a series of experiments on copying one SEM network to another (network replication in the NRH sense) and investigating possible benefits of employing a population of networks. Network copying primarily focuses on the replication of synaptic weights and/or producing a child network at least whose performance is comparable to the parent's. First, we will describe the test result of two possible mechanisms for neuronally copying weight distribution of a parent network to that of a child network. Secondly we will present a preliminary architecture for incorporating an ensemble of SEM networks that can perform cooperative learning.

Replication by Topographic Drive

The first experiment involved adding unidirectional topographic connections from parent WTA to child WTA neurons which is reminiscent to the previous work on NRH (Fernando et al 2008, 2012). Topographic connectivity is prevalent throughout the vertebrate brain and is thought to play an important role in the anatomical and functional organisation of different brain area (Thivierge and Marcus 2007, Sporns et al 2002). Topography is often found in projections from sensory organs to brain area but it also exists between higher brain centres, although the functional role of topographic connections between deep brain regions is poorly understood. In the framework of

the SEM network model we exploit topography in producing an instructive signal between two neuronal circuits which may contribute to transferring a chunk of organised information from one brain region to another by neuronal replication.

The overall architecture for topographic neuronal replication is shown in Figure 2A. There are two identical WTA circuits sharing the same input neurons, and the parent WTA circuit projects its output topographically to the child circuit with the same fixed strength. We negatively shifted the resting value of the global inhibition signal for the child network by the same amount as the topographic weight, so that the overall firing rates of the parent and child network outputs are similar to each other. The rest of the network parameters were set as in the previous experiment. This time we used MNIST handwritten digits (digits 0, 1, 2, and 3 were used) for input, which is a more challenging task than Gaussian image and allows more sophisticated comparison between the two network behaviours. The strength of topographic connections was set to 10, and the global inhibition pulse was shifted by the same amount. The simulation was run for 500 simulation seconds.

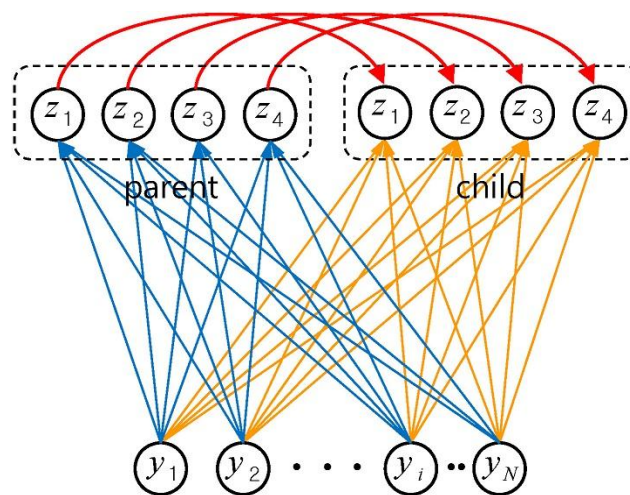


Figure 2A. A network architecture for topographic copying.

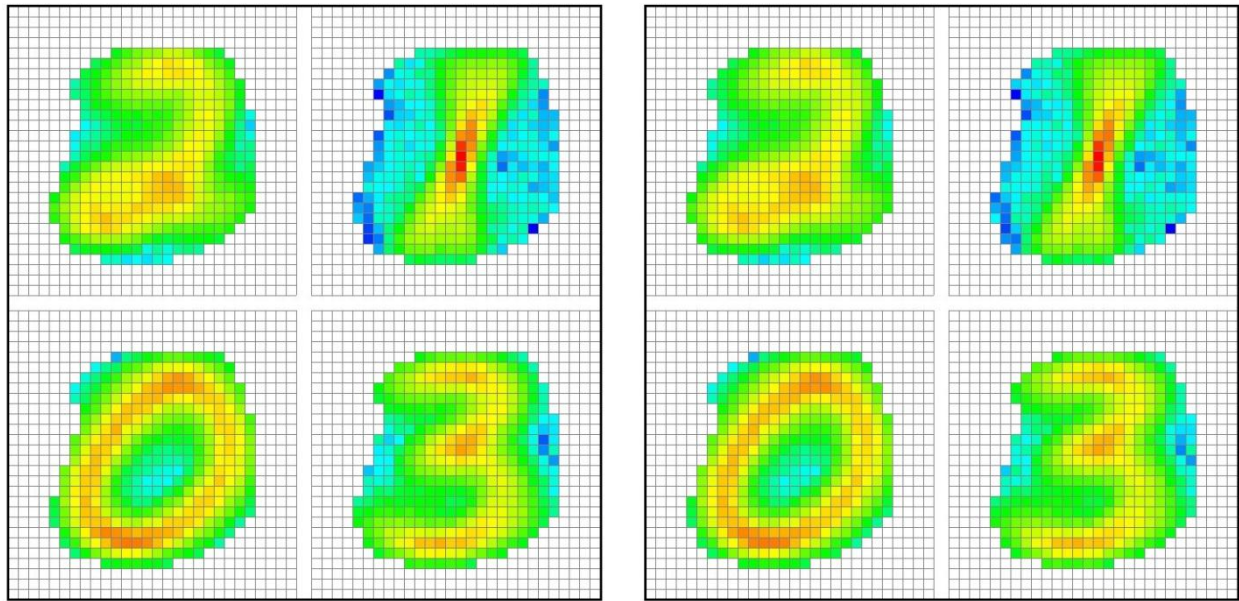


Figure 2B. Weight maps of parent (left) and child (right) networks after learning.

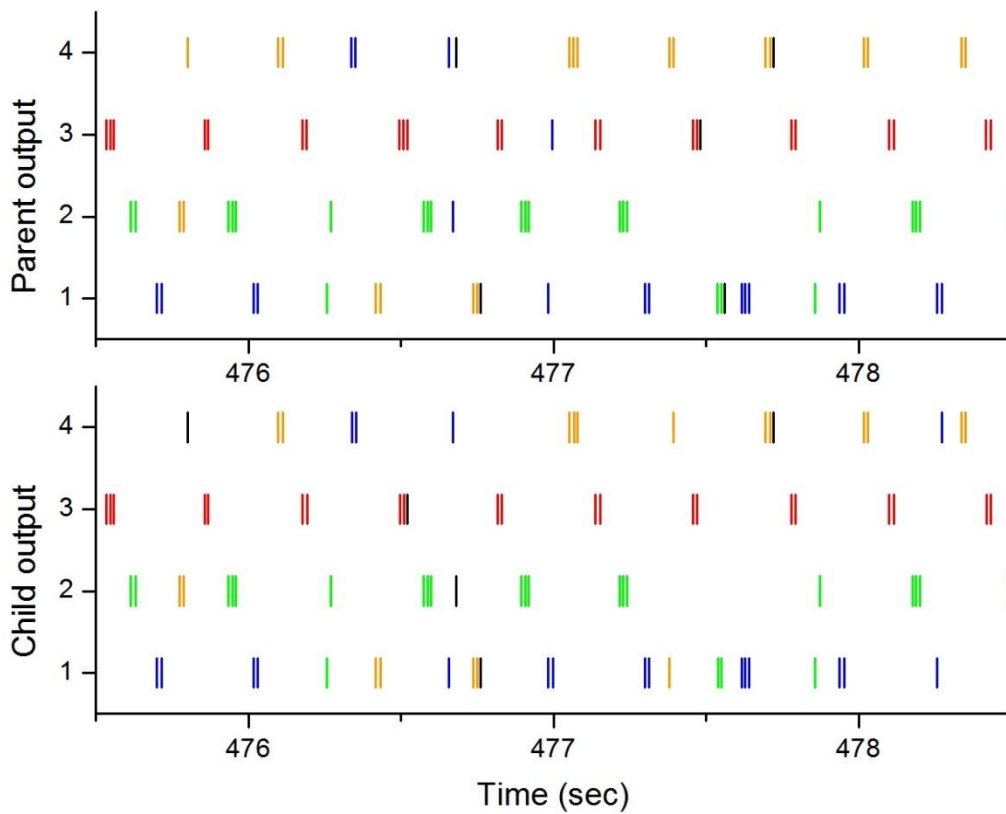


Figure 2C. Output of two networks after copying.

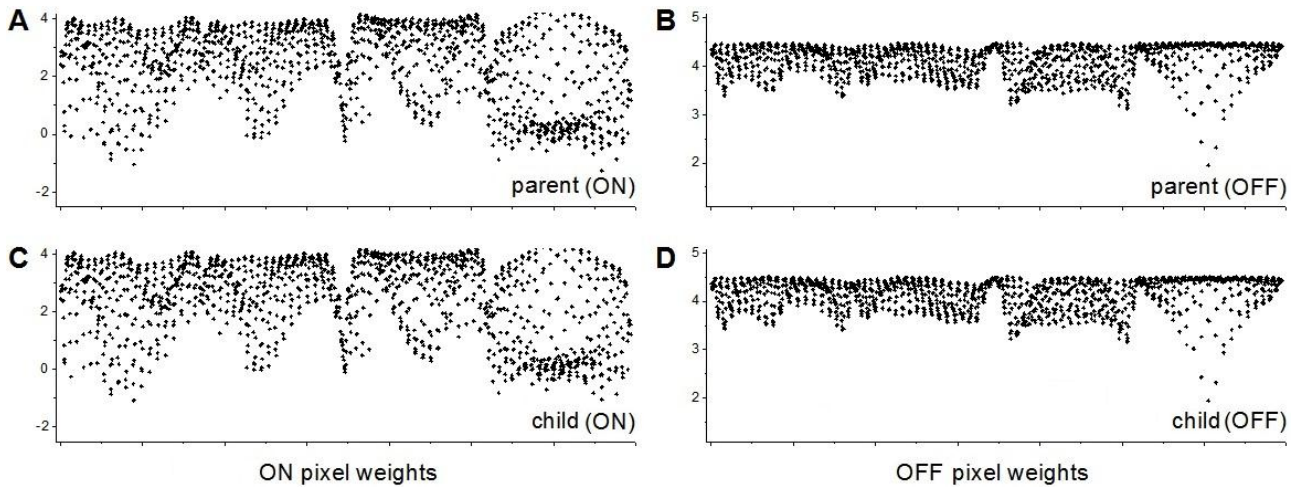


Figure 2D. All weight values of parent and child network. Weights for ON (A,C) and OFF (B,D) pixels are shown separately.

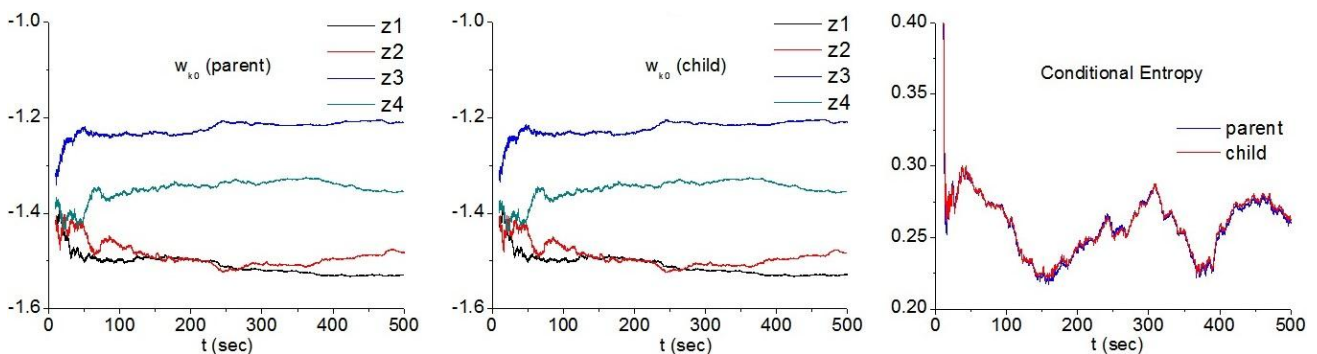


Figure 2E. Time evolution of network behaviours. (Left two graphs) Self-excitability of the two networks. (Rightmost) Conditional entropy of the two networks are superposed.

Figures 2B-2D, illustrating a typical case, show that the child network copied the parent network almost identically. This is because, even though the basic copying mechanism involves variation, the SEM network dynamics themselves performs a neuronal expectation maximisation algorithm which is *guaranteed* to converge to one of the local minima. So although the parent and child network started from different initial conditions the topographic input drives the child network to converge to the same local minimum as the parent by influencing the firing probabilities of the output neurons.

However, we observed that in some cases the firing probabilities of the output neurons of the child network sometimes converge to different classes (digits) from the parent network. For example, Figure 3A shows that the child network failed to produce identical inference to the parent's in that the output neurons fire for different digits. For the sake of the unsupervised classification task the

two networks perform equally well, however in terms of precision in the network replication task the two networks have low similarity to each other. Also, with weaker topographic strengths ($w < 10$) in some cases the two networks converged to show different performances while responding to the same digits (as shown in Figure 3B).

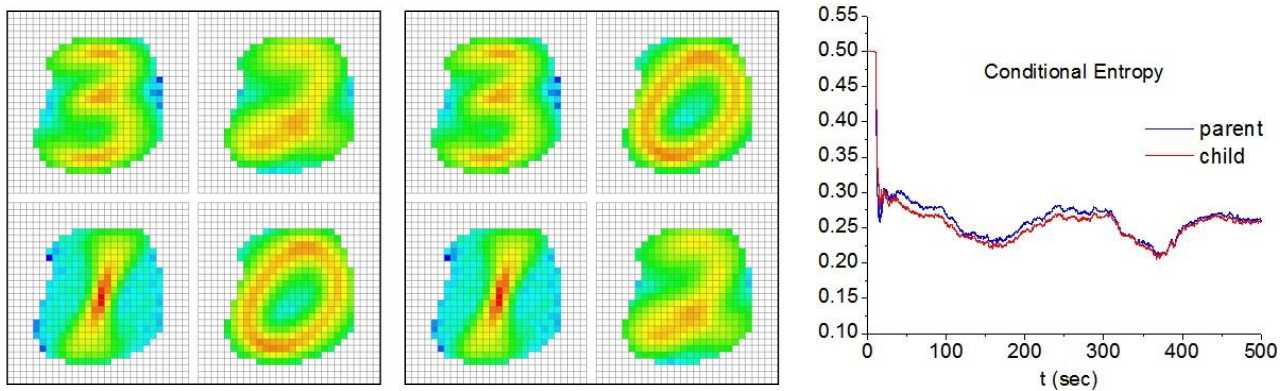


Figure 3A. A part of the child output neurons (for digits 0 and 2) failed to encode same classes of parent's while showing equal performances.

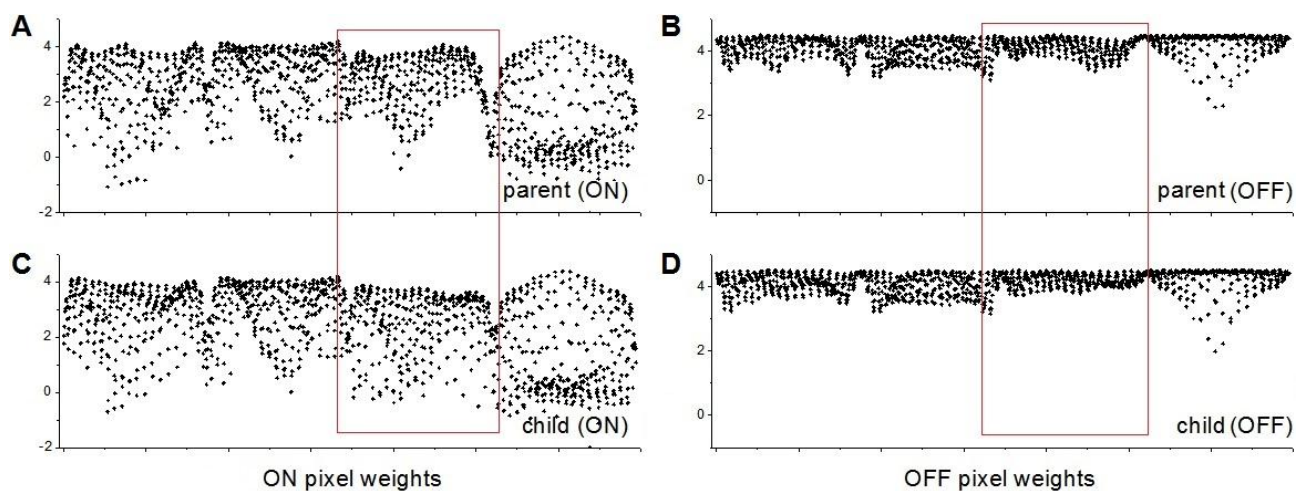
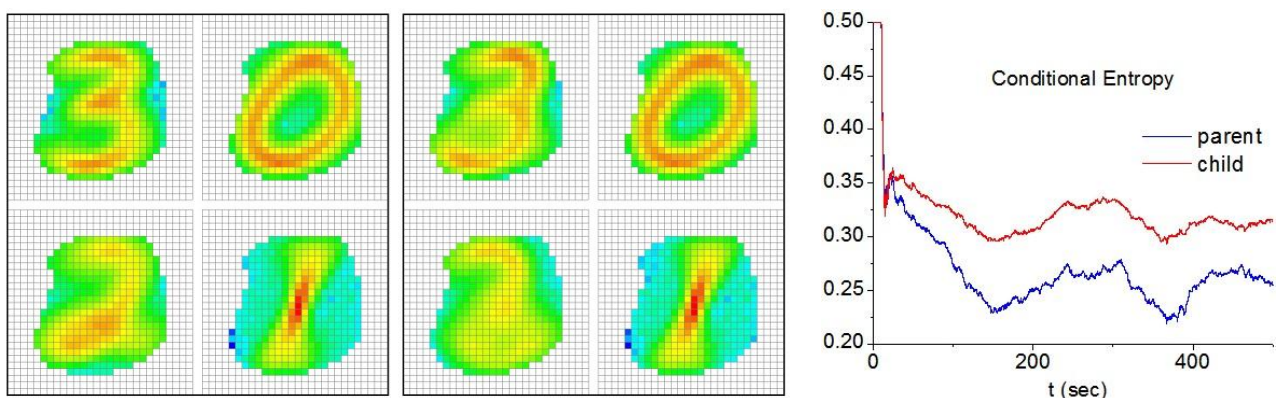


Figure 3B. The child network poorly learned to discriminate between digits 2 and 3. Weight values in red boxes show slightly more salient differences between the two networks. The topographic strength was set to 5.

This 'poorer' replication increased with decreasing topographic weight strength. Multiple trials resulted in the number of poor replication = 15 out of 20 trials for $w=2$, 7/20 for $w=5$, and 2/20 for $w=10$. It can be inferred that if the class mismatch (as in Figure 3A) between parent and child networks occurs when using a strong topographic weight, then the child network's learning performance will be strongly interfered with by the input from the parent, which will lead to even poorer performance of the child network.

While this investigation showed that such a replication mechanisms is feasible for SEM networks, it also highlighted the fact that the learning dynamics of the networks mean that under a wide range of conditions the child will converge to the parent state. This will resulting in a lack of variation within a wider framework of evolutionary neural dynamics. An interesting avenue for future work will be to conduct a rigorous analysis on the statistics of erroneous replication in relation to various topographic strength and initial conditions.

Replication using ITDP

Another method for copying SEM networks was developed based on a different form of synaptic plasticity inspired by the experimental data from vertebrate hippocampal formation. Specifically we employed and developed a model of input timing dependent synaptic plasticity (ITDP) as a possible mechanism for neuronal replication and instructive synaptic learning for cooperative computation by a population of networks. The term ITDP was first proposed by Dudman et al (2007) who discovered a particular form of heterosynaptic plasticity in the young adult rat hippocampus.

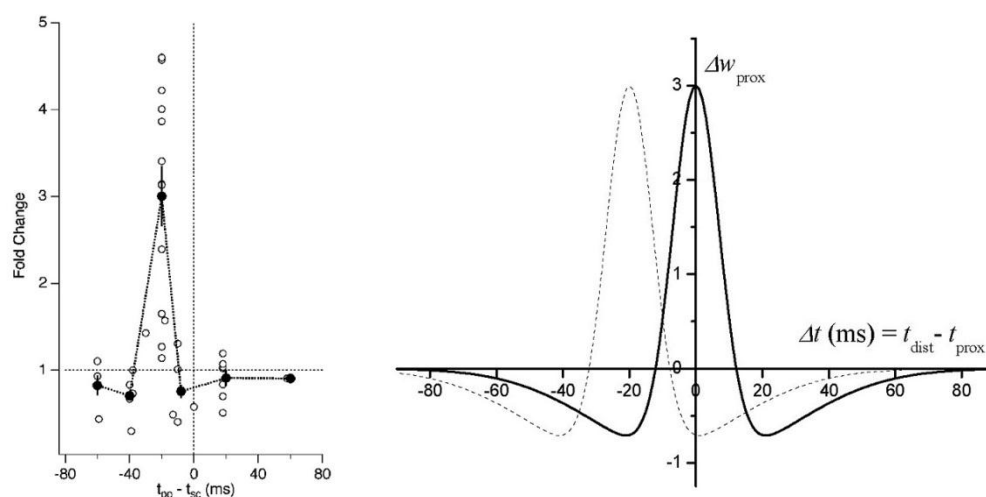


Figure 4. ITDP curves from experimental data (Dudman et al 2007) (left), and the model curve (right). The model curve used in this work was shifted to the origin by assuming the pre-existence of conduction delays in distal connections.

They reported that the subthreshold stimulation of distal performant path inputs from the entorhinal cortex (EC) to hippocampal CA1 pyramidal neurons induces long-term potentiation at

the proximal Schaffer collateral (SC) synapses from the CA3 region when these two inputs (PP-SC) are paired at a precise interval. A more recent experiment (Cho et al 2011) has also confirmed the existence of ITDP when precisely pairing the thalamic and cortical inputs (TS-CS) converging on the lateral nucleus of amygdala (LAn), which might participate in the auditory fear conditioning of the rat. Another study (Menzies et al 2010) also predicted the function of the vestibulo-ocular reflex gain adaptation by modeling heterosynaptic spike-timing dependent depression from the interaction between vestibular and floccular inputs converging on medial vestibular nucleus (MVN) in the cerebellum. Therefore, ITDP seems to be quite prevalent among different brain regions and is thought to play a considerable role in brain function. Hence it is an interesting candidate mechanism to be involved in replication.

Experimental data suggests that ITDP acts as a form of heterosynaptic hebbian learning which depends on the time difference between two input stimuli. We developed a simple ITDP model by defining a plasticity curve as a function of input time difference using a Mexican hat function as shown in Figure 4. In biological ITDP, the peak LTD occurs at the input time delay of -20ms, where distal input precedes proximal input by 20ms. For computational convenience, we assumed the axon converging on the proximal synapse already has 20ms of conduction delay, thus the plasticity curve was shifted to have its peak value at zero delay. The proximal synaptic change by ITDP can be written as

$$\frac{dw_{prox}}{dt} = \eta \sum_{t_{dist}} \sum_{t_{prox}} H(t_{dist} - t_{prox})$$

where $H(t)$ is the function shown in Figure 4, and t_{dist} and t_{prox} are the spike times of the presynaptic neurons of distal and proximal synapses respectively. The proximal weight is updated whenever either of two presynaptic neurons spikes. The learning rate was 0.01 throughout the experiment.

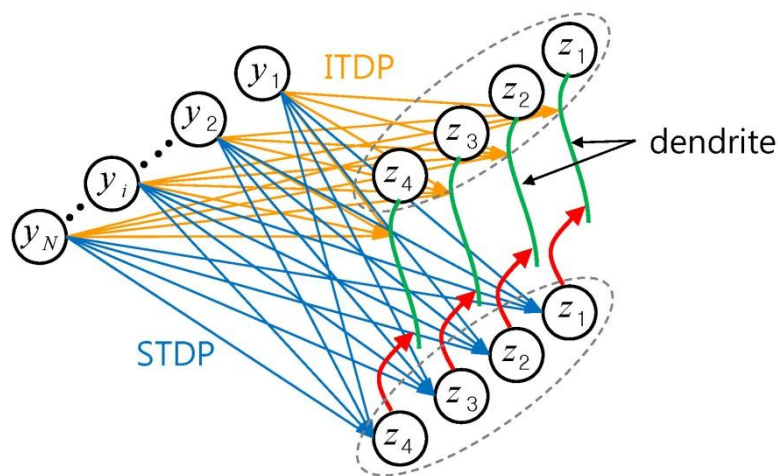


Figure 5. Network architecture for ITDP copying. Parent output neurons project as distal input, and child input connections end at proximal synapses.

Using our ITDP model we again performed an experiment on copying SEM networks. The network architecture is the same as for the previous topographic copying, except the topographic projections are considered as distal synapses as conceptually shown in Figure 5. The parent network learns using STDP in the same way as before, whereas the child network is adapted using ITDP. The basic idea is that the input weights of the child network (which become proximal input to the child WTA) are ‘instructed’ by the parent, in which a child input connection is potentiated only when its activity is correlated with the corresponding output neuron of the parent. Biological experiments (Dudman et al 2007, Cho et al 2011) have observed that synaptic potentials originating at distal dendritic locations are severely attenuated when they reach the soma, thus there is little chance of them causing somatic spikes. Hence, for simplicity, we ignored the EPSP caused by the distal input from parents by setting topographic weights to zero.

Since the above ITDP model may result in unbounded growth of some synaptic weights that are positively correlated with distal input, the child weights are subjected to a heterosynaptic competition rule in order to normalise the total afferent strengths for an output neuron k to be at the same constant level as the parent’s. After every ITDP update, the weight is again updated as

$$w_{ki} \leftarrow w_{ki} \frac{N(3 - w_{k0})}{2W_k}, \quad W_k = \sum_{i=1}^N w_{ki}$$

where N is the total number of input (presynaptic) neurons. The rule also depends on the intrinsic neuronal excitability w_{k0} so that the sum of all excitatory factors (afferent weights and self-excitability) for a postsynaptic neuron is kept constant.

The SEM replication task was performed exactly as in the previous experiment. After about 500-1000 simulation seconds of learning, the child network learnt to perform the same classification task. Due to the different nature of the two plasticity mechanisms in the parent and child network, the result exhibits less precision of replication (Figure 6A-C) than in the earlier (topographic drive) case. Typically the child network converged to almost the same weight distribution (relative ratio between each value) as the parent’s, while the precise values of weights are distributed differently. This difference stems from the different synaptic normalisation mechanisms caused by two plasticity rules, in which the ITDP expresses LTP predominantly while STDP can maintain a bounded level of weight values both by LTP and LTD. A more sophisticated formulation of the ITDP rule and synaptic normalisation can be investigated as future work.

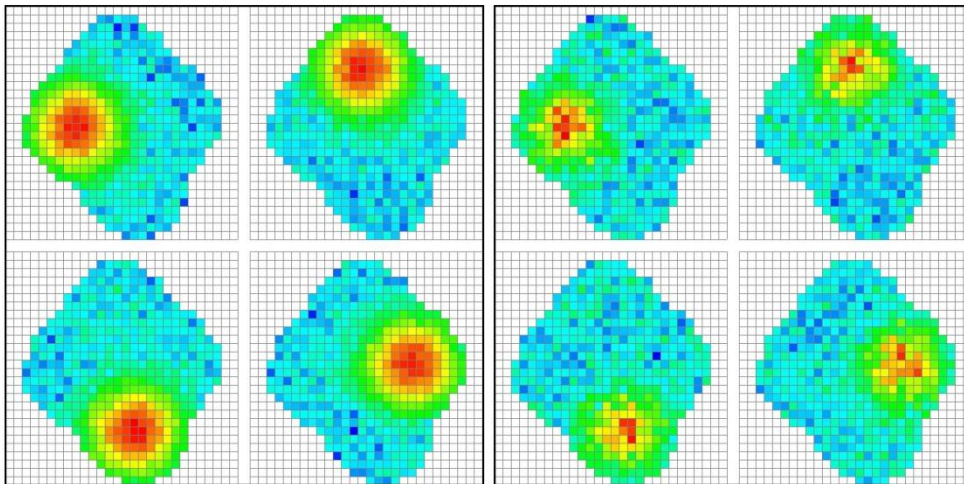


Figure 6A. Weight maps of parent (left) and child (right) network after learning Gaussian images.

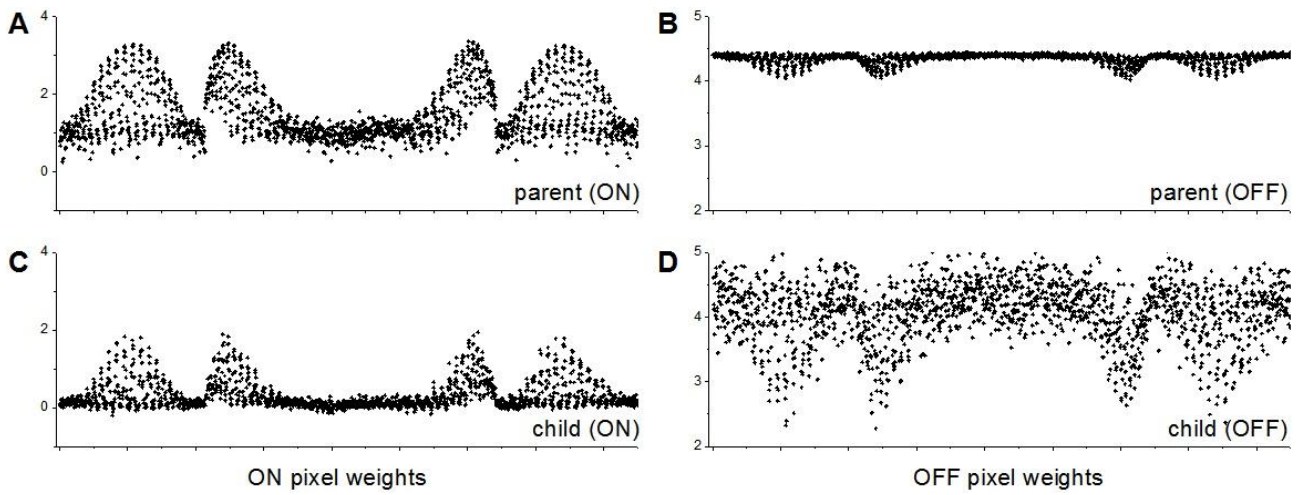


Figure 6B. Weight Values after Gaussian image learning.

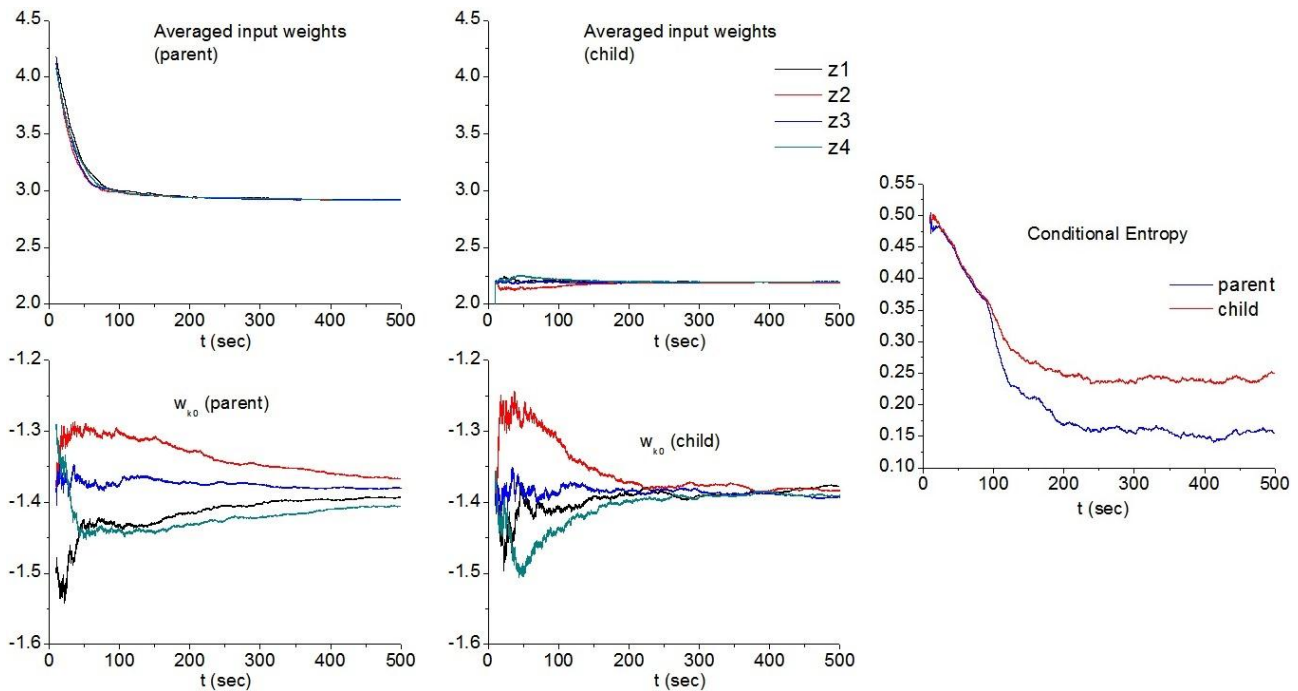


Figure 6C. Time course of ITDP learning for Gaussian image. Averaged input weights converging on each output neuron and self-excitabilities are shown for each network.

The Gaussian image learning task resulted in poorer child performances than the parent's, whereas MNIST data has shown similar performances for parent and child networks. Although the reason for this difference is not yet fully understood, one possible inference is that the ITDP plasticity curve with its Mexican hat shape includes depression (negative) regions around the potentiation centre so the learnt weight strengths of the potentiated connections and depressed connections develop sharper borders (as seen in the child weight maps in Figure 6A and 6D) than for the parents, which might conflict with the smooth weight distribution demanded in Gaussian images. On the other hand, MNIST digit images have sharp borders between on and off pixels, and hence are less influenced by this property.

Although the replicated network has captured an essential capabilities for the classification tasks, the ITDP based copying mechanism for SEM networks in the current experiment has shown imperfect replication in terms of network similarity. Nevertheless, biologically inspired ITDP replication may be able to expand the list of possible mechanisms for neuronal replication. Also ITDP may act as a controllable trigger for inducing variability of target neural circuits in terms of neuroevolution, in which a small shift of a single distal input timing can influence a number of proximal synapses. In terms of the role of neural replication in a wider Darwinian neural dynamics context, this variability may be a good thing, especially if we learn to control it a bit more.

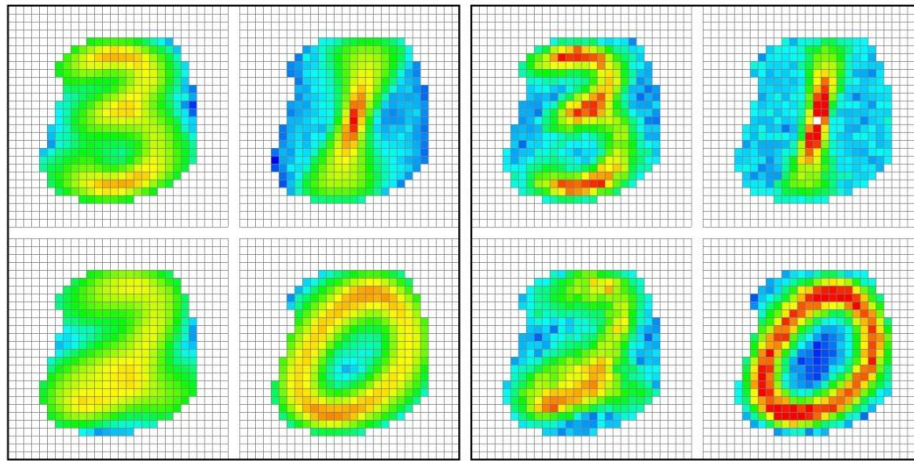


Figure 6D. Weight maps after learning MNIST dataset.

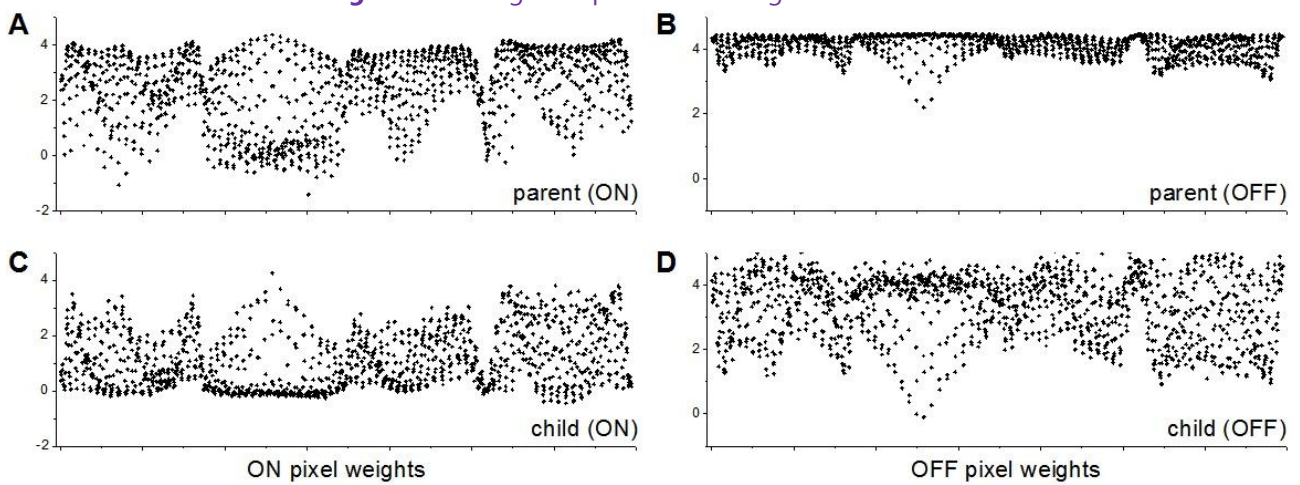


Figure 6E. Final weight values of MNIST dataset.

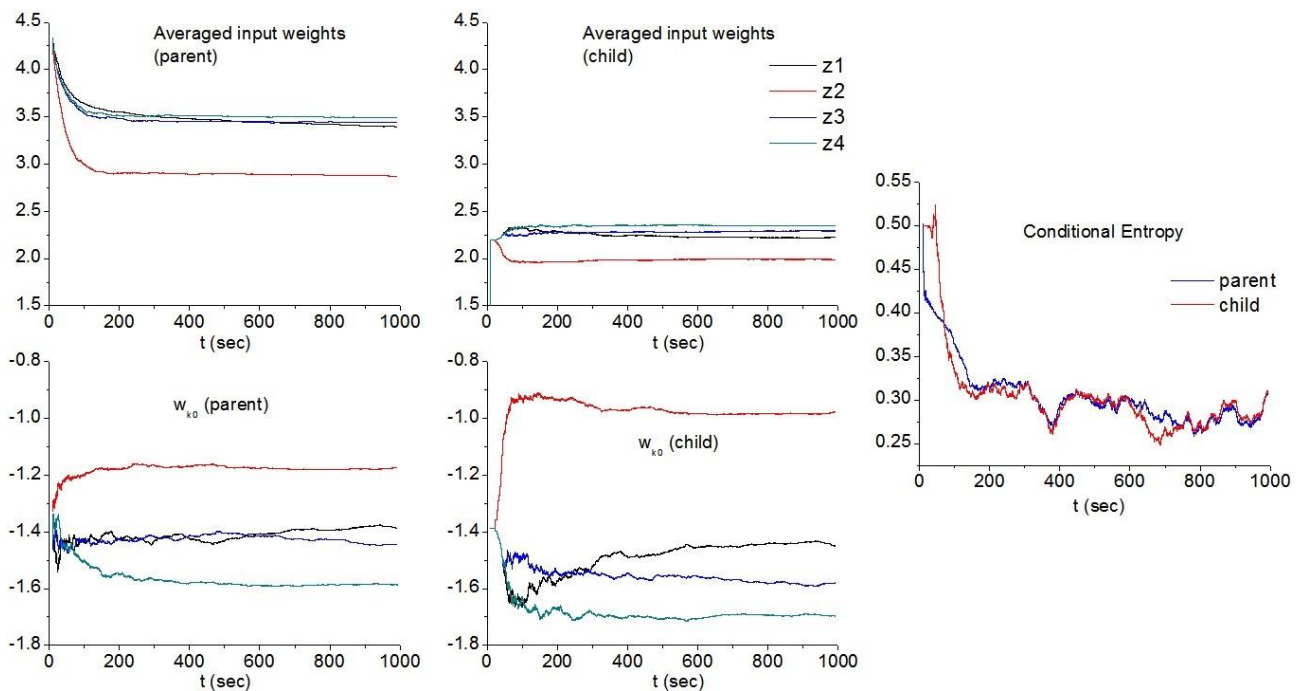


Figure 6F. Time course of ITDP learning for MNIST dataset.

3.3 ITDP controlled SEM network ensemble learning

Viewing the SEM model as a unit cortical microcircuit for solving classification tasks naturally leads to a question of how a population of such microcircuits can work in parallel toward better performance. We report a preliminary experiment of a possible neuronal architecture which implements an ensemble of SEM networks that can achieve cooperative learning of classification tasks. Ensemble learning is a population based process by which multiple models (regressors, classifiers or experts) are combined to solve a single problem. Combined decisions from an ensemble of classifiers often leads to more accurate result than the individual classifiers. The individual learners are often referred to as “weak learners” because they can be boosted by combination to become a strong learner as long as the individual weak learners are at least slightly better than random guessers.

Since ensemble learning (in the machine learning context) typically allows flexible structures by admitting any model/learning method for the individual learners, the combination of individual solutions is a crucial problem. Ensemble learning can be either supervised or unsupervised, where supervised learning is provided with the number of classes and the class labels for each input data (classifier ensemble) while this information is not available in unsupervised learning (clustering ensemble) (Hong 2008). Although the number of classes is available for the SEM model in this work, the lack of label information for each output requires solving the consensus problem of combining multiple non-labeled decisions. This problem can be addressed if there is some instruction mechanism for the SEM ensemble that operates selectively on the same class.

The experiment reported here presents a neuronally implemented model for a population of SEMs which follows a process similar to the mixture of experts method (MoE) (Jacobs et al 1991), a kind of ensemble learning which uses a neural network like architecture. We model a two layered feedforward network in which a population of WTA circuits receives a set of inputs and gives output to the final layer comprising a single WTA circuit (Figure 7A). A “gating” WTA gives topographic distal projections to the final layer WTA, controlling the efferent synaptic weights of the WTA ensemble by ITDP. The input weights of the WTA ensemble are subjected to STDP as in the previous model. When each WTA in the ensemble learns to fire for one of the classes, their outputs have to be combined such that each neuron in the final layer receives input from the neurons in the ensemble representing the same class. This is achieved by ITDP, instructed by the gating WTA, in which the gating WTA’s output selectively enhances the synaptic connections from the ensemble neurons encoding the same class as the gating WTA output.

The simulation was run using MNIST digits for 1000 seconds with an ensemble of 14 WTA circuits. The ensemble WTAs received input from random feature subspace in which a half of the pixels were randomly selected, while the gating WTA received the full input. Due to the smaller number of afferent connections to the output WTA, its global inhibition level was shifted by 500 ($A_{inh}=2600$, $O_{inh}=-150$). The rest of the parameters were the same as in the earlier work outlined in the previous section.

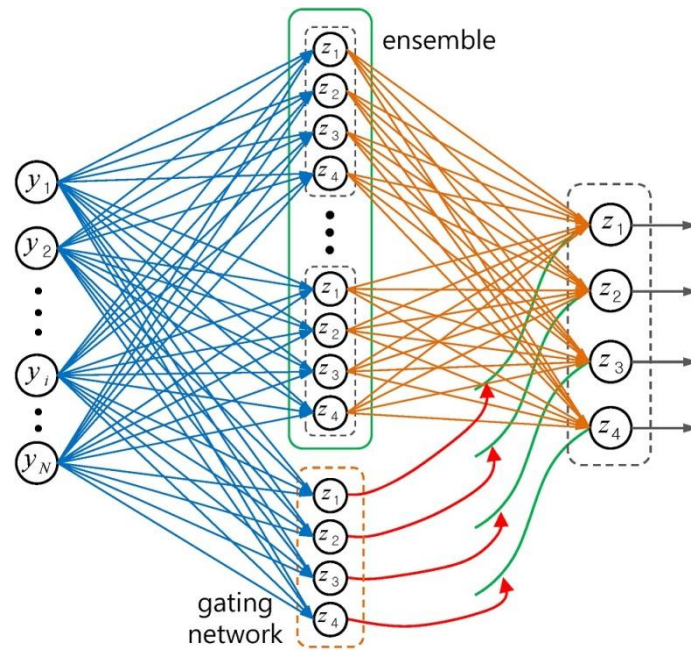


Figure 7A. Network architecture for SEM ensemble

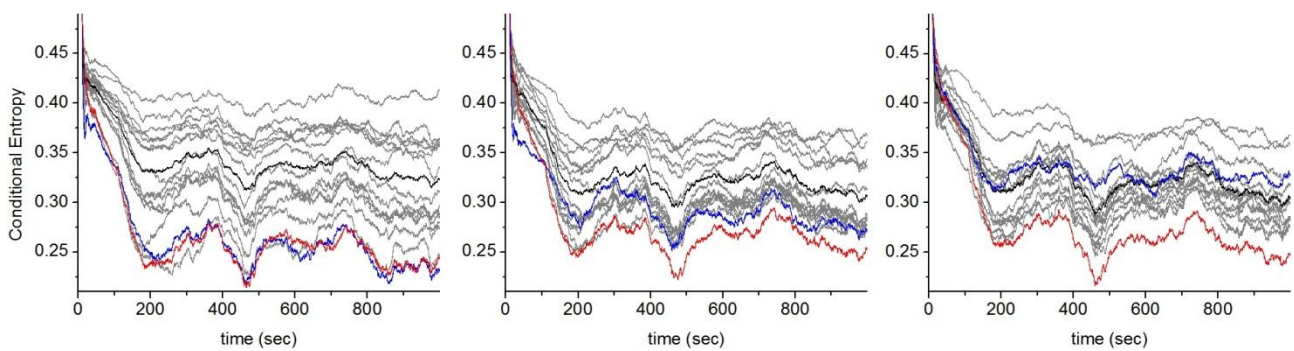


Figure 7B. Conditional entropies of SEM ensemble from different trials. Each colour indicates; [Red] Output WTA, [Blue] Gating WTA, [Black] Average of ensemble WTAs (Grey).

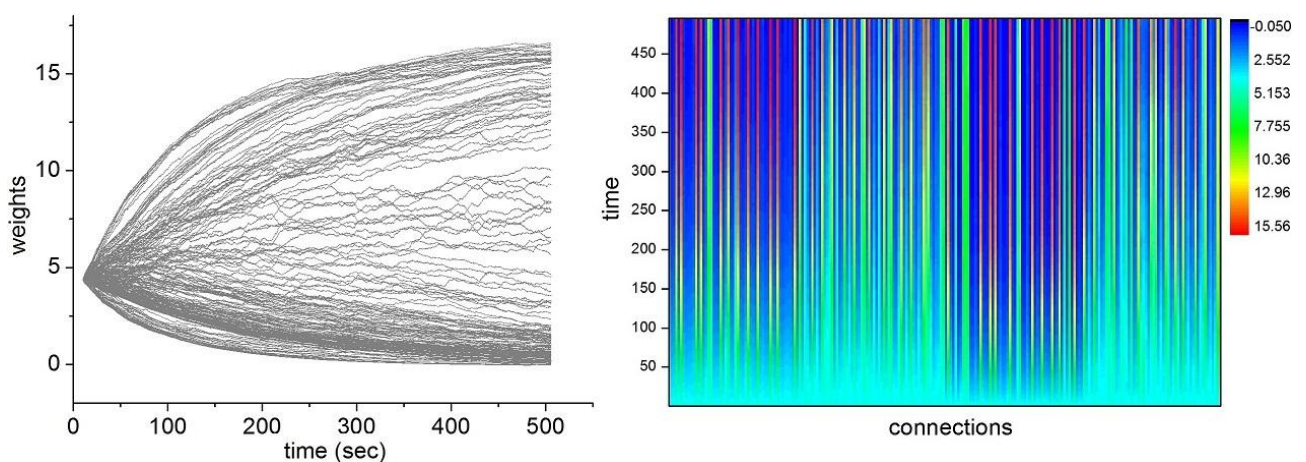


Figure 7C. Weight change of ITDP layer. Right image shows time course of all connection weights using colour map. The initial value of all ITDP weights was set to 4.4.

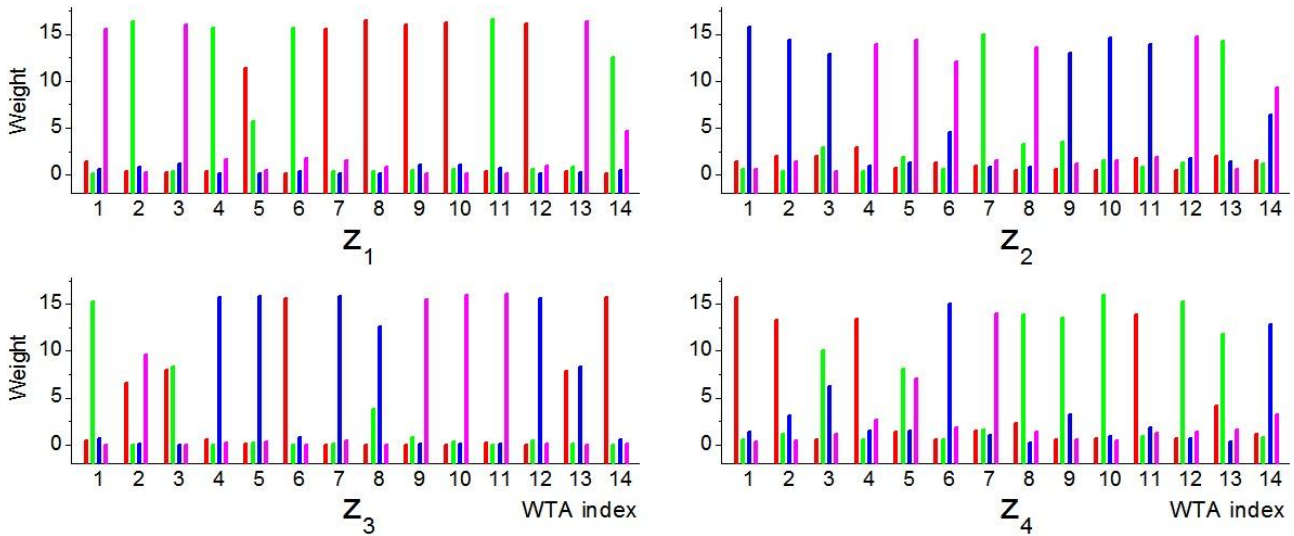


Figure 7D. An example of learnt weight to the final layer from all WTAs in ensemble. Each graph shows connection weights from ensemble members to each output neuron of the final layer (z_1 - z_4). The four colours represent the index of output neurons of each ensemble WTA, z_1 : red, z_2 : green, z_3 : blue, z_4 : magenta.

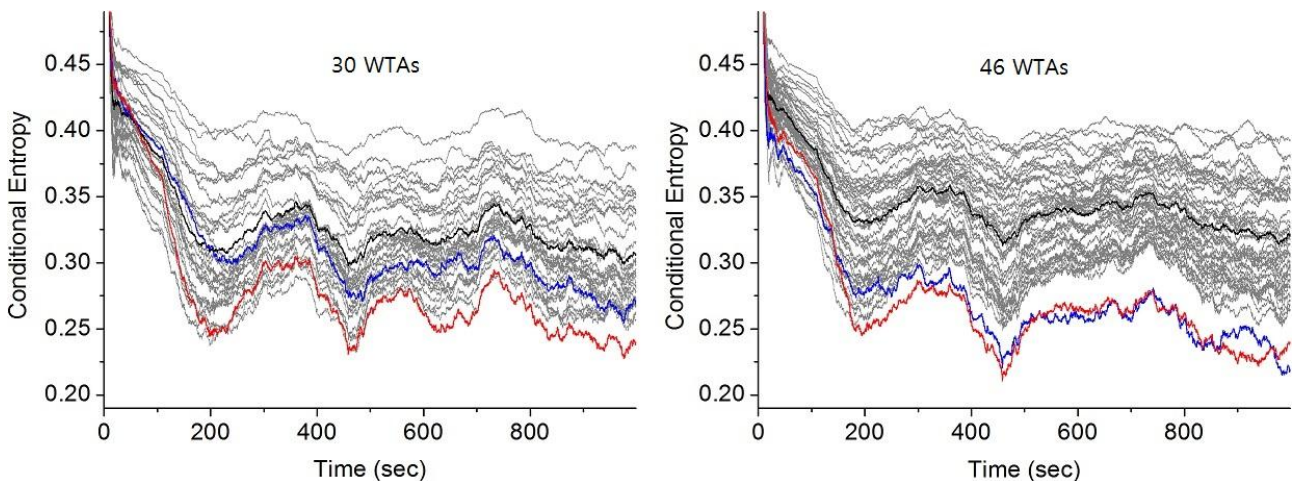


Figure 7E. Simulation with larger ensemble sizes, 30 and 46 WTAs.

The visual observation of spike bursting in the output WTA after learning seemed to show less salient difference than expected, however the traces of conditional entropy have shown that the output WTA outperformed the rest of WTAs in most cases (Figure 7B). It is quite interesting to see that the performance of the gating WTA (the only one having input from the full set of features), which actually guides the whole ensemble, does not have the best performance. Figure 7B shows three examples of different gating WTA performances of: (left) matching to the output WTA, (middle) between output WTA and the average of ensemble, (right) worse than the ensemble average. Figure 7C and 7D shows the change of ensemble efferent weights during and after learning, which clearly shows that the connections carrying the signal for the same class are enhanced and converged to the corresponding output WTA neurons. Despite the system having no

information about the class labels in ensemble WTA neurons, the gating WTA (which is also unsupervised) can selectively recruit and assign the ensemble output to converge to one of the final layer neurons.

While our SEM ensemble model mimics the MoE architecture, the process is different from MoE, in that the operation of the SEM gating WTA on the ensemble output is not based on immediate training input but is accumulated by slow additive synaptic plasticity over long time scale, whereas MoE gating network instantaneously changes the strength of ensemble efferents for each input vector. Therefore the system is unable to filter out the wrong output from ensemble WTAs when an output neuron in the ensemble fires for multiple classes, in which the false spikes are also passed to the final layer through the enhanced connections. However, the higher probability of wrong firing of an ensemble neuron dissipates the synaptic resource over multiple efferent connections, resulting in lower synaptic weights than the case of the neuron which fires predominantly for one class, hence less chance to win at the final layer WTA. Again, the false spikes of the gating WTA will result in less chance of enhancing the corresponding target set of ITDP weights because of timing mismatch. However, if a large number of ensemble WTAs fire equally wrongly for the same class, the final output develops a high chance of generating the wrong output. This often happens when the two input image are hard to discriminate (such as digits 3 and 8), even if the subfeatures are randomly selected. Therefore the system is not entirely free from the feature selection problem as experienced in other ensemble learning methods. This limitation has meant that the simulation using larger ensemble sizes did not significantly improve the performance (Figure 7E), indicating a lack of ensemble diversity. A form of evolutionary computation might be able to participate at this point, by being neurally integrated into the current architecture.

However, the important point at this stage is that this novel architecture demonstrates for the first time how ensemble learning can be neurally implemented with populations of SEM networks. The preliminary results show that the ensemble performs a little better than individual networks. The lack of diversity within the population will be tackled in the next phase of work to try and improve performance. An all STDP implementation will also be investigated. It is also possible that the relative strength of the ensemble method, in terms of efficiency of learning, might become more apparent when reducing the time spent on learning in the SEM networks (i.e. it might allow a good level of performance with fewer learning iterations). These issues will also be investigated.

4 LSM based Path Evolution Algorithm Results

Work at Sussex has also involved the reimplementing of the new spiking liquid state machine (LSM) design which contains coupled winner take all circuits developed in Maass's group (Klampfl and Maass 2013). The model has been successfully reimplemented and verified (not straightforward as there are unintentional inaccuracies in the original paper). Preliminary investigations have shown that the intrinsic dynamics of pathways in this model are so stable that it is very hard to produce any variation in behavior through mutation. The work is on-going and

more radical mutation operations will be considered. The code developed was passed on to Parmenides who are also making use of it.

5 Chaotic Exploration and Darwinian Neural Dynamics

This section describes a complementary direction of research into another kind of possible candidate for Darwinian neural dynamics where such dynamics occur at a slightly higher, more abstract level. Crucially the (chaotic) neuro dynamics are embodied and it is the whole neuro-body-environment system that must be considered, although the changes occur at the neural level.

There is a growing body of observations of intrinsic chaotic dynamics in the nervous system (Guevara, Glass, Mackey, & Shrier, 1983; Rapp, Zimmerman, Albano, Deguzman, & Greenbaun, 1985; Freeman & Viana Di Prisco, 1986; Wright & Liley, 1996; Terman & Rubin, 2007). Some studies indicate intrinsic chaotic dynamics in animal motor behaviors at both the neural level (Rapp et al., 1985; Terman & Rubin, 2007) and the level of body and limb movement (Riley & Turvey, 2002). These seem particularly prevalent during developmental and learning phases (e.g., when learning to coordinate limbs) (Ohgi, Morita, Loo, & Mizuike, 2008). The existence of such dynamics in both normal and pathological brain states, at both global and microscopic scales (Wright & Liley, 1996), and in a variety of animals, supports the idea that chaos plays a fundamental role in neural mechanisms (Skarda & Freeman, 1987; Kuniyoshi & Sangawa, 2006). Although the functional roles of chaotic dynamics in the nervous system are far from understood, a number of intriguing proposals have been put forward. Freeman and colleagues have hypothesized that chaotic background states in the rabbit olfactory system provide the system with “continued open-endedness and readiness to respond to completely novel as well as familiar input, without the requirements for an exhaustive memory search” (Skarda & Freeman, 1987). Kuniyoshi and Sangawa (2006) made the important suggestion that chaotic dynamics underpin crucial periods in animal development when brain-body-environment dynamics are explored in a spontaneous way as part of the process of acquiring motor skills. Recent robotics studies have demonstrated that chaotic neural networks can indeed power the self-exploration of brain-body-environment dynamics in an embodied system, discovering stable patterns that can be incorporated into motor behaviors (Kuniyoshi & Suzuki, 2004; Kinjo, Nabeshima, Sangawa, & Kuniyoshi, 2008; Pitti et al., 2010). Very recently this work has been fundamentally extended at Sussex (Shim 2013, Shim and Husbands 2012) to allow goal-directed (fitness-directed) search. This research developed a general and fully dynamic embodied neural system, which exploits chaotic search through adaptive bifurcation, for the real-time goal-directed exploration and learning of the possible locomotion patterns of an articulated robot of an arbitrary morphology in an unknown environment.

We have been examining the relationship between fitness-directed embodied chaotic exploration (CE) and Darwinian neural dynamics and believe there is a strong direct analogy between CE and estimation of distribution algorithms (EDA), a recognised kind of evolutionary algorithm. Hence we

believe CE can be viewed as a kind of specialised Darwinian neural dynamics. This idea is explained in the following paragraphs.

Estimation of Distribution Algorithms. Instead of using an explicit population of solutions and the traditional machinery of evolutionary algorithms, EDAs (Pelikan et al. 2000, Larrañaga and Lozano 2002, Pelikan et al. 2002) employ a (often Bayesian) probabilistic model of the distribution of solutions which can be sampled by generating possible solutions from it. Search proceeds through a series of incremental updates of the probabilistic model, usually starting with the model encoding a uniform distribution over admissible solutions and ending with a restricted model that generates only a very good solution (either the global optima or a good local optima). EDAs have long been acknowledged as a form of evolutionary algorithm (Pelikan et al. 2000, Larrañaga and Lozano 2002, Pelikan et al. 2002, Pelikan 2005). The general scheme is as follows:

1. initialize model $M(0)$ to represent uniform distribution over admissible solutions
2. generate candidate solutions by sampling $M(t)$
3. evaluate all candidate solutions
4. update model; $M(t+1) = \text{update_function}(\text{sampled solutions}(t), \text{sampled fitnesses}(t), M(t))$
5. if stopping criteria not met go to 2

The key point is that an EDA is a more abstract kind of evolutionary algorithm employing a generative model instead of an explicit population plus genetic operators.

Embodied Chaotic Exploration. Conventional optimization and search strategies generally use random perturbations of the system variables to search the space of possible solutions. We have developed a method which uses the intrinsic chaotic dynamics of the system to naturally power a search process without the need for external sources of noise (Shim and Husbands, 2010, 2012, Shim 2013). We employ the concept of Chaotic Mode Transition with external feedback (Davis, 1990), which exploits the intrinsic chaoticity of a system orbit as a perturbation force to explore multiple synchronised states of the system, and stabilises the orbit by decreasing its chaoticity according to a feedback signal that evaluates the behaviour. An evaluation signal which measures how well the locomotion behavior of the system matches the desired criteria (e.g. locomote as fast as possible) is used to control a bifurcation parameter which alters the chaoticity of the system. During exploration, the bifurcation parameter continuously drives the system between stable and chaotic regimes. If the performance reaches the desired level, the bifurcation parameter decreases to zero and the system stabilizes. A learning process that acts in tandem with the chaotic exploration captures and memorizes these high performing motor patterns. *Because the idea is a little complex and not yet well know, the overall system is now explained in some detail before we go on to explain how it relates to EDAs and Darwinian neural dynamics.*

The overall architecture of the system is illustrated in Figure 10, the neural architecture generalizes and extends that presented in (Kuniyoshi and Sangawa 2006) which is inspired by the organization of spinobulbar units in the vertebrate spinal cord and *Medulla Oblongata* (the lower part of the

brainstem which deals with autonomic, rhythmic, involuntary functions). Each joint in an articulated robot is connected to a motor unit comprising a pair of central pattern generator (CPG) neurons which receive sensory input. The neurons produce motor outputs for an antagonistic muscle pair that control the movement of the joint. Each CPG neuron is modeled as an extended Bonhöffer van der Pol (BVP) oscillator (FitzHugh 1961, Asai et al. 2003), which can be viewed as a simplification of the full Hodgkin Huxley neural model (Hodgkin and Huxley 1952), employing continuous (rate based) signals. The CPG neurons are all connected to each other but these connections are initially made inactive. The CPG neurons receive sensory signals that integrate information from the body-environment interaction dynamics experienced by the system (e.g. from force and position/angle sensors). Hence, while the direct connections are inactive, any coupling between the oscillators will be indirect via bodily and environmental interactions. The network of oscillators, coupled through physical embodiment, has multiple synchronized states (modes) that reflect the body schema and its interactions with the environment, each of which can be regarded as a potential candidate for 'meaningful' motor behavior. The exploration process, powered by adaptive bifurcation through the feedback evaluation signal, allows the system to become entrained in these modes, sampling them until one is found that is sufficiently stable and high performing for the bifurcation parameter to reduce to zero and the system to fully stabilize. As the system stabilizes, the connections between oscillators are dynamically activated using an adaptive synchronization learning scheme. In this way the wiring between the oscillators is changed in order to capture and maintain the high performing motor pattern. The learning rule is also controlled by the bifurcation parameter and is set up such that the connections between the oscillators are effectively zero (inactive) during the exploration process but gradually increase (become active) as the system nears stability (see Shim and Husbands 2012 for full details). Thus, exploration and learning are merged as a continuous dynamical process such that the desired motor behavior is spontaneously explored, discovered, and memorized in a coherent way. If the performance drops, for instance following a change in the environment or damage to the body, the system will automatically return to the exploration phase until a new stable high performing motor pattern is discovered. The method can be interpreted as a continuous and deterministic version of stochastic trial-and-error search which exploits the intrinsic chaotic behaviour of the system.

The pair of CPG neurons in each motor unit (labeled l and r) operate according to the following coupled differential equations:

$$\tau \dot{x}_l = c \left(x_l - \frac{x_l^3}{3} - y_l + z_1 \right) + \delta(H_l(s_l) - x_l) + F_l^x \quad (10)$$

$$\tau \dot{y}_l = \frac{1}{c} (x_l - b y_l + a) + \varepsilon(H_l(s_l)) + F_l^y \quad (11)$$

$$\tau \dot{x}_r = c \left(x_r - \frac{x_r^3}{3} - y_r + z_2 \right) + \delta(H_r(s_r) - x_r) + F_r^x \quad (12)$$

$$\tau \dot{y}_r = \frac{1}{c} (x_r - b y_r + a) + \varepsilon(H_r(s_r)) + F_r^y \quad (13)$$

Where τ is a time constant, and $a=0.7$, $b=0.675$, $c=1.75$ are the fixed parameters of the oscillator. Each consecutive pair in the set of $2N$ oscillators are sequentially allocated to each motor unit as $l = 2m-1$ and $r = 2m$. $\delta=0.013$ and $\varepsilon=0.022$ are the coupling strengths for afferent input $H(s)$ which is a function of raw sensor output s , processed by the sensor adaptation module (SAM) – see Figure 10. F_i^j is a coupling term between oscillators and is subject to the learning process. z_1 and z_2 are the control parameters for adjusting the chaoticity of the motor unit. Their difference ($\mu = z_2 - z_1$) changes identically in all motor units, and acts as the global bifurcation parameter. In the stable regime where the two control parameters are symmetric, it had been found (Asai et al. 2003) that the two coupled BVP equations exhibit bistable phase locking of their oscillations in a parameter range of $0.6 < z_1 = z_2 < 0.88$. From the observation of a number of experiments on the oscillator dynamics, we chose to fix $z_2 = 0.73$ and to vary z_1 in order to ensure a higher probability of multistability of the system in its stable regime. For further details see (Shim and Husbands 2012).

The evaluation signal is determined by a ratio of the actual performance (e.g. forward speed) to the desired performance. Where the desired behavior is forward locomotion, the evaluation signal, E , is measured according to the following equations, where v is the robot velocity vector and τ_E is a time constant. Using this leaky integrator equation means the velocity is continuously averaged over a time window thus eradicating gyrations and oscillations.

$$E(t) = \|\bar{v}\|, \quad \tau_E \frac{d\bar{v}}{dt} = -\bar{v} + v \quad (14)$$

The time course of the bifurcation parameter, μ , is tied to the evaluation signal using the following equations.

$$\tau_\mu \frac{d\mu}{dt} = -\mu + \mu_c G\left(\frac{E}{E_d}\right), \quad G(x) = \frac{1}{(1+e^{16x-8})} \quad (15)$$

Where τ_μ and μ_c are constants and E_d is the desired performance; $G(x)$ implements a decreasing sigmoid function which maps monotonically from (0,1) to (1,0) shaped so that the boundary value at $x=1$ and its derivative become almost 0 so as to make the function smoothly vanish to zero to facilitate gradual stabilisation. Since the method is intended for use in the most general case, where the robotic system is arbitrary, we do not have prior knowledge of what level of performance it can achieve. Using the concept of a goal setting strategy (Barlas and Yasarcan 2006), the dynamics of the desired performance are modeled as a temporal average of the actual performance, such that the expectation of a desired goal is influenced by the history of the actual performance experienced as described by the following equation.

$$\tau_d \frac{dE_d}{dt} = -E_d + E \quad (16)$$

The sensor signal fed to a CPG neuron undergoes homeostatic adaptation as it passes through a sensor adaptation module (SAM) before reaching the neuron (see Figure 10). The SAMs were

introduced because by adjusting the waveforms of input signals to be close to those of the neural activities, the synchronicity between the neural and physical system was enhanced thus allowing the neural system to cope with an arbitrary robotic system. This regulation also results in a diversification of output behaviors, increasing the scale of the search process.

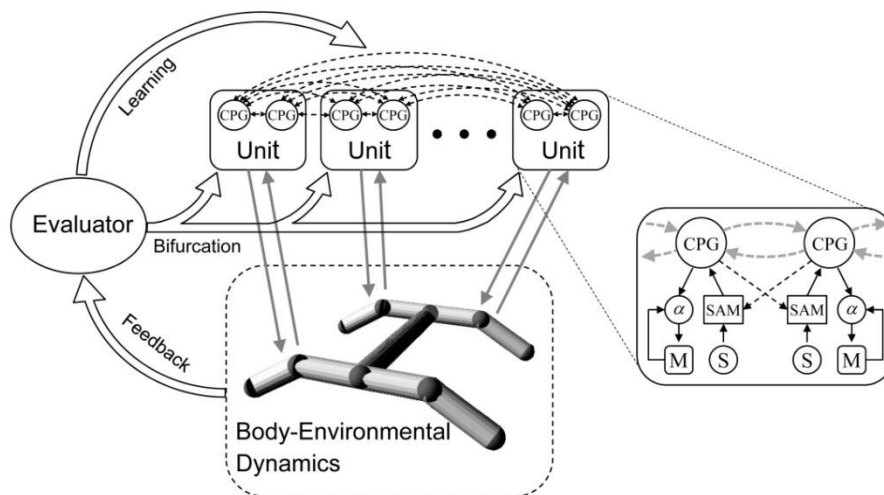


Figure 10: An overview of the integrated exploration and learning scheme. Each robot joint has a dedicated motor unit comprising oscillator- based central pattern generator neurons (CPG) with sensory input (S) and motor output (M). Connections between the oscillators are initially inactive but they are weakly coupled through the body and environment. An evaluation feedback signal controls a global bifurcation parameter that alters the chaoticity of the CPGs. The chaotic dynamics of the neuron-body-environment system drive a search process that finds motor patterns that perform well according to the evaluation criteria. As the system stabilizes on a high performing pattern the bifurcation parameter reduces to zero and the connections between the oscillators become active, their weights being set by a learning procedure that is smoothly linked to the chaotic exploration process. The learning process further stabilizes, captures and memorizes the motor patterns. Sensory input undergoes homeostatic adaptation as it passes through a

The chaotic exploration and learning system was evaluated by using it to control a range of realistically simulated articulated robots that were required to locomote in an effective way. Successful experiments with a swimming robot and a variety of walking robots of differing morphologies demonstrated that the framework is highly general (Shim and Husbands 2012). In each case a range of stable locomotion behaviors were discovered and learnt. It was also shown that the robots can readily readapt after damage or other changes.

Figure 11 shows two of the robot simulations used to demonstrate the system. Figure 12 shows a typical swimming motion discovered and captured for the 4-Fin swimmer. It is an efficient frog-like motion. Figure 13 illustrates how the method allows real-time adaptation. The robot suffers damage where the length of the fin on arm 4 was reduced by 90%. Its performance reduced almost immediately to zero so the exploration process automatically kicked-in. Very quickly it discovers

and stabilises one of the high performing transient patterns, needing only a few trials for the oscillator learning mechanism to completely stabilise the system once more into a new high performance swimming motion that is able to compensate for the damage. For details of applications of the method to other robots and body morphologies, including walking robots, see Shim and Husbands (2012).

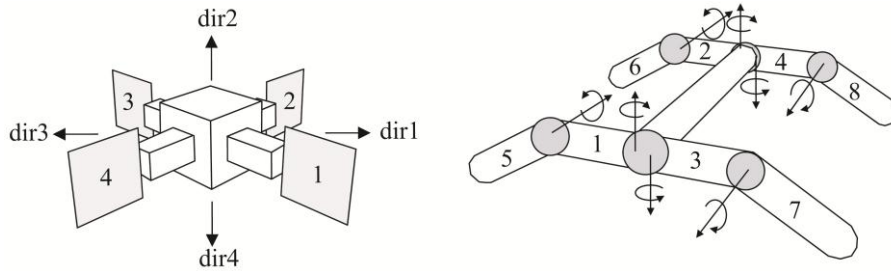


Figure 11: Robotic simulation models of a 4-Fin Swimmer (2D movement) and a Quadraped (3D movement) used to evaluate the method.

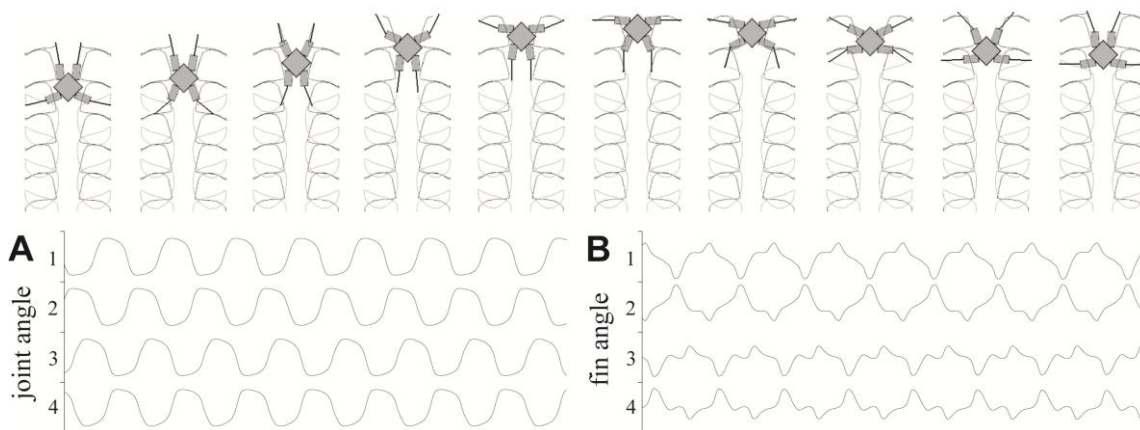


Figure 12: (Upper) Snapshots of a straight swimming (ST dir3) behaviour of the 4-Fin Swimmer developed by the exploration and capture method. Images were taken every 1/10 gait cycle. The tip trajectories of the fore (fin 3,4:black) and rear (fin 1,2:grey) fins are shown. (Lower) (A) Joint angles and (B) fin bending angles of the behaviour. Each segment along the vertical axis indicates the range [-1,1] rad.

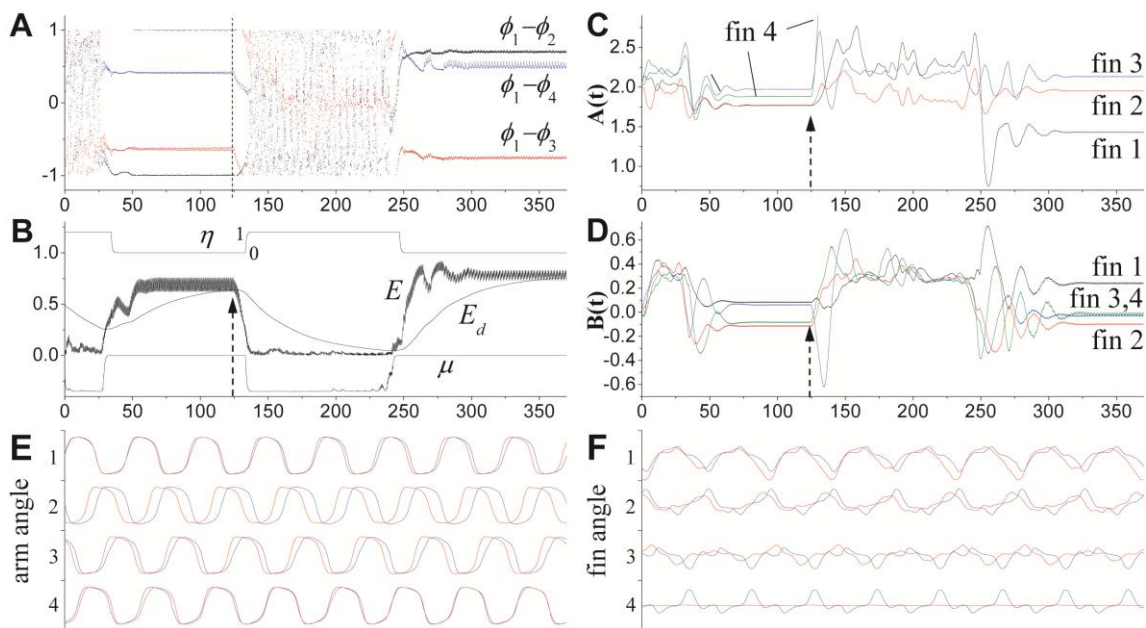


Figure 13: Realtime recovery after a radical change to the body of the swimmer (damage). Dashed lines and arrows indicates the time of damage, when the length of fin 4 is decreased to 1/10 of its original length. The sensor gain of (damaged) fin 4 ($A(t) \approx 5.0$) in (C) was truncated for a better view of the other gain plots. (E) and (F) show the joint angles and the fin angles respectively, where the undamaged motion (blue) and the readapted motion (red) are superposed. The fiducial point for the superposed plots was set to the starting point of arm angle 1 in (E).

Relationship between Embodied Chaotic Exploration and Evolutionary Neural Dynamics.

Embodied chaotic exploration (as exemplified by the neural-body-environment CE system outlined above) is a kind of generative search process in that the intrinsic dynamics of the system drive it to search through state space, this is analogous to using the generative statistical models of EDAs to sample the search space. The CE system implicitly embodies a population of points in the search space that are sampled through the intrinsic dynamics of the system (which are inherently unstable and chaotic: it cannot help but move between attractors, sampling the space). In EDAs the model is altered according to (fitness) feedback from the sampling. In generative chaotic search the intrinsic dynamics (analogous to the model in EDAs) are themselves altered according to fitness feedback (by changing the bifurcation parameter). Hence a new (mutated) state space that inherits most of its properties from the last one is produced, in the same way that the new model in EDAs is a kind of mutated copy of the previous one (see Figure 14). These relations between CE and EDA are illustrated in the table below.

	EDAs	Embodied CE
Generative system	Probabilistic model (M)	Neuro-body-environment (n-b-e)
Search space (population) sampling	Generate from M	Intrinsic n-b-e dynamics cause sampling
Updating step	Adjust M according to fitness feedback from sampled points	Adjust n-b-e dynamics according to fitness feedback from points in state space sampled

Table1: Relationships between EDAs and Embodied chaotic exploration.

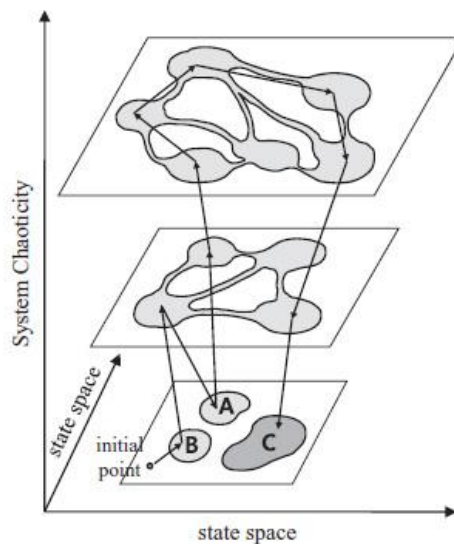


Figure 14: A conceptual illustration of the state-space of a neural-body-environment system coupled through physical embodiment, which consists of three basins of attraction (A,B,C) with different performances. An exploration process to find the desired attractor, C, by varying the complexity of the state-space landscape through changing the system chaoticity. Lump spaces and narrow passages in the landscapes of higher complexities represent quasi-attractors and itinerant pathways, respectively.

The neural oscillator learning mechanism employed in the CE system described above also plays a very important part in exploiting the brain-body-environment dynamics. The search space formed by the neuro-body-environment system tends to have no truly completely stable state for systems of the degree of complexity we are interested in (e.g. robotic system with fairly high number of DOFs). Instead, the state space will look more like several semi-stable states; the system orbit will eventually escape from each of these, moving on to another state after staying there for a while. So the whole state space might consist of a number of shallow basins connected to each other. By incorporating oscillator learning into the system, we can stabilise the orbit somewhere (in some 'shape' of limit cycle) in one of the shallow basins, thus what we actually obtain from the brain-body-environment + learning system is a state space composed of completely stable states.

If we consider a completely stabilised dynamical state (periodic behaviour), its location as a point in state space is almost always 'near' the shallow basins of the previously mentioned system without learning. We can think of the final learnt-stable-dynamics (stable limit cycle) of the system orbit as inheriting its dynamics from the unlearned dynamics (slowly changing limit cycle-like trajectory of the CE system before oscillator learning is activated), in that the learnt-stable dynamics directly reflects the characteristics of unlearned (semi-stable) dynamics. Just imagine that at some time instance we stop (learn-and-stabilise) a tiny glass bead which is slowly moving (wandering) in a shallow bowl. In our CE work, the oscillator learning starts to work only when the system chaoticity returns back to a stable regime. This means the location of the orbit right before the onset of learning is already inside a semi-stable basin of the unlearned state space. From there, the learning process morphs the state space into a set of stable basins. So the final stable dynamics are mutated forms of the semi-stable (parent) dynamics. This can be regarded as a kind of local search interacting with the quasi Darwinian dynamics of the core CE system, or possibly even as a second level of evolutionary dynamics.

The strong direct analogies between EDAs and embodied CE described earlier, suggest the latter can be viewed as a kind of abstracted evolutionary algorithm in the same way EDAs are. Hence the kind of complex neuro-body-environment dynamics analysed in the model describe earlier (Shim and Husbands 2012) points to another, more abstract, more specialised kind of candidate for Darwinian neural dynamics. The embodied CE model developed so far has strong biological underpinnings although it uses a rate based model of neuro-dynamics. It will be interesting to try and develop a spiking model with similar dynamics, to show the applicability at a more detailed level of biological realism. Further analysis of the relation between CE and evolutionary dynamics is likely to be fruitful. Other future work will demonstrate the system on a physical robot.

6 Related Work on Neural Dynamics

The issues described in the previous section, along with a more comprehensive discussion of the relationships between neuroscience and evolutionary approaches to robotics can found in the following output:

- P. Husbands, R. Moiola, Y. Shim, A. Philippides, P. Vargas, M. O'Shea (2014) Evolutionary Robotics and Neuroscience. In P. Vargas, E. Di Paolo, I. Harvey and P. Husbands (Eds), *The Horizons of Evolutionary Robotics*, MIT Press, 17-64. (March 2014)

Related work on neural assembly dynamics (formation, reconfiguration and reformation of clusters of neurons within larger ensembles) in networks of phase coupled oscillator neurons, which will inform later work on possible higher level mutation operators in Darwinian neural dynamics, as well as possible cognitive architectures, produced the following output:

- Molioli, R. and Husbands, P. (2013) Neuronal Assembly Dynamics in Supervised and Unsupervised Learning Scenarios, *Neural Computation* **25**(11) : 2934–2975. (Dec. 2013)

The latter publication is an investigation of the hypothesis that the dynamic formation of groups of neurons—neuronal assemblies— can mediate cognitive phenomena at many levels (Buzsaki 2010), although their detailed operation and mechanisms of interaction are still to be uncovered (Kopell, Whittington, & Kramer, 2011).). One hypothesis suggests that synchronized oscillations underpin their formation and functioning, with a focus on the temporal structure of neuronal signals (Varela et al. 2001). A key result from this publication is the demonstration that dynamic assembly reorganization alone (mediated by synchronisation dynamics), without the need for other plasticity mechanisms, can be exploited to solve various minimally cognitive tasks.

7 Simulation of WP2 Neurophysiology Experiment

Another strand of work at Sussex has been on developing a model of the MEA experiments discussed in D2.3 to aid with analysis and refinement of those experiments. Initial implementations are complete and preliminary results of using this simulation to explore causal learning (of the type to be attempted with the MEA) are discussed below. This section is best read in conjunction with D2.3 and outlines the achievement of Milestone 1 (Fit realistic neuronal models to experimental setup in WP2).

Fernando (2013) proposes that a spiking neural network can be made to implement causal inference along the Bayesian lines proposed by Griffiths and Tenenbaum (2005) (see discussion of causality in D2.3). The network implementation there is a network of stochastic binary units. We aim to implement a more biologically realistic spiking neural network model. A common choice for such tasks is based on the model developed by Izhikevich (2003). This is an integrate-and-fire model, where parameters can be chosen to model a number of different spiking patterns. For example, with only small changes to parameters, Izhikevich demonstrates that numerous distinct neuron types can be modelled. This flexibility may prove beneficial as we intend to use the simulation for a number of slightly different purposes. It will also be possible to ‘swap in’ different network models into the simulation.

The Izhikevich neuron model has been used as a basis for simulation of neural cultures before. Maheswaranathan et al (2012) investigated the role of neural density in emergent synchrony in networks of Izhikevich neurons. Their work incorporates a model of a neural culture in a two-dimensional plane, mimicking an MEA experiment. We are following this general approach as we develop the detailed model the MEA experiment. Furthermore, the model is suited to implementation of spike-timing dependent plasticity learning (STDP). This has been demonstrated for example by Izhikevich (2006) and subsequently used in numerous other studies.

7.1 GPGPU Simulation evaluation

A number of approaches were investigated for suitability as implementations of our requirements. Initially we looked at implementations of the 1000 neuron spiking network described by Izhikevich (2003). This can be implemented in only 26 lines of MATLAB code¹ and so is a simple example of the type of model we are interested in. Recently several authors have reported that significant speedups can be obtained by using GPGPU computing (general purpose computing with graphics processors), e.g. Nageswaran et al (2009), Nowotny (2011), Fidgeland et al (2013). We have therefore investigated the possibility of using GPGPU computing for faster simulation times. In particular, we have investigated the use of the NVIDIA CUDA framework for GPGPU computing. There are a number of neural simulation frameworks available. A popular tool suitable for this task is Brian (Goodman and Brette, 2008). This allows specification and simulation of networks in Python. However, its support for GPGPU implementations is limited at the moment. There are specific GPGPU frameworks such as Nemo² and GeNN³. These offer the ability to convert complex neural network models into GPGPU optimized code. However, for the moment a complete framework such as one of these seems to be unnecessary and indeed offers little above directly coding the solution ourselves, given the extra time needed to implement the network according to the specifications of the framework. However the use of such a framework may be worth revisiting if we decide to use more complex neural models in the future.

As an initial test, the network described by Izhikevich (2003) was first coded in Python by translating the original MATLAB code using equivalent functions provided by Python:

```
Ne = 800
Ni = 200

re = rand(Ne)
ri = rand(Ni)

a = hstack([0.02*ones(Ne), 0.02+0.08*ri])
b = hstack([0.2*ones(Ne), 0.25-0.05*ri])
c = hstack([-65+15*re**2, -65*ones(Ni)])
d = hstack([8-6*re**2, 2*ones(Ni)])

S=rand(Ne+Ni,Ne+Ni)
S[:,Ne:] *= 0.5
S[:,Ne:] *= -1.0

v = -65*ones(Ne+Ni)
```

¹ The implementation used by Izhikevich can be found here: <http://www.izhikevich.org/publications/net.m>

² <http://nemosim.sourceforge.net/>

³ <http://genn.sourceforge.net/>

$u = b \cdot v$

`firings = []`

`for t in arange(0,1000):`

`I=hstack([5*randn(Ne),2*randn(Ni)])`

`fired = v >= 30`

`firings.append(arange(Ne+Ni)[fired])`

`v[fired] = c[fired]`

`u[fired] += d[fired]`

`I += sum(S[:,fired],1)`

`v+= 0.5*(0.04*v**2+5*v+140-u+I)`

`v+= 0.5*(0.04*v**2+5*v+140-u+I)`

`u+= a*(b*v-u)`

The same network was implemented with GPU acceleration using the PyCUDA⁴ library. This library allows the programmer to call GPU kernels from Python with a modest overhead, and also implements tools for transferring data to and from the graphics device. Both implementations were timed on the following workstation. Note that the CUDA graphics card is not currently state of the art:

- Processor: Quad core 3.2GHz AMD Phenom II X4 840
- Memory: 4GB RAM
- Hard disk: SATA III 6Gbps (gigabits per second) Western digital WDC WD10EALX-009BA0. Quoted maximum transfer rate 126MBps (megabytes per second)⁵.
- Graphics: NVIDIA GeForce GTX 570 (480 CUDA cores, 1280MB Memory, 152GBps memory bandwidth)
- Operating system: Ubuntu 13.04 64-bit

The results (below) show that the PyCUDA implementation of the network simulation is drastically faster than the native Python equivalent.

⁴ <http://mathemat.tician.de/software/pycuda/>

⁵ <http://www.wdc.com/global/products/specs/?driveID=894&language=1>

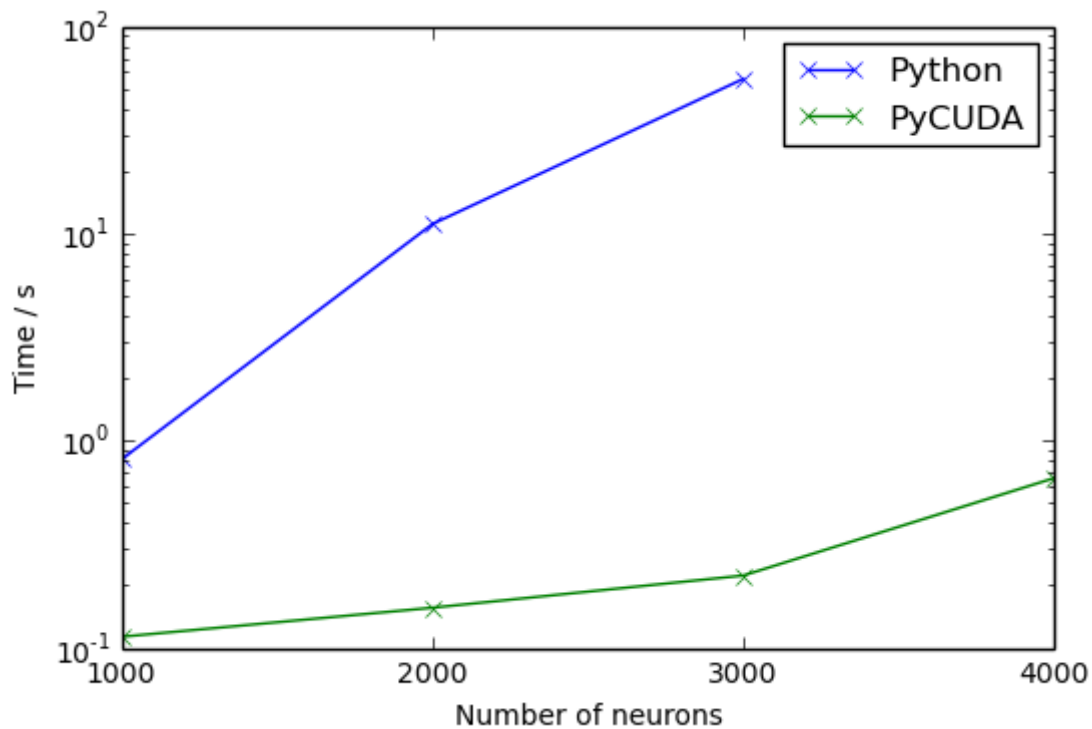


Figure 15: Speed of one second simulation of neural network with varying numbers of neurons. For the 4000 neuron case, Python took too long to produce any results

7.2 STDP causal inference results

This section describes preliminary results with causal learning in spiking networks. The full MEA spatial simulation was not used in this experiment, just the neural network part, but the full MEA setup will be modelled in the immediate future in the next iteration of experiments with this model.

We constructed a more complex Izhikevich neuron-based network, implementing the STDP strategy described by Izhikevich (2006). In our model, 1024 neurons are used, 800 of which are excitatory, and 224 are inhibitory⁶. For each pair of neurons, a synapse is created with 10% probability. Synapses leaving excitatory neurons start with a weight of +6, meaning that on a pre-synaptic firing, the post synaptic neuron will receive an additional 6mA of input current. Synapses leaving inhibitory neurons start with a weight of -5. Unlike the simpler network, the current model contains synaptic delays, randomly chosen between 1 and 20ms following Izhikevich (2006). Excitatory synapses (those leaving excitatory neurons) are adjusted by the STDP rule:

$$\Delta w = \begin{cases} -0.12e^{-\frac{t_{pre}-t_{post}}{20}} & t_{pre} > t_{post} \\ 0.1e^{-\frac{t_{pre}-t_{post}}{20}} & t_{pre} \leq t_{post} \end{cases}$$

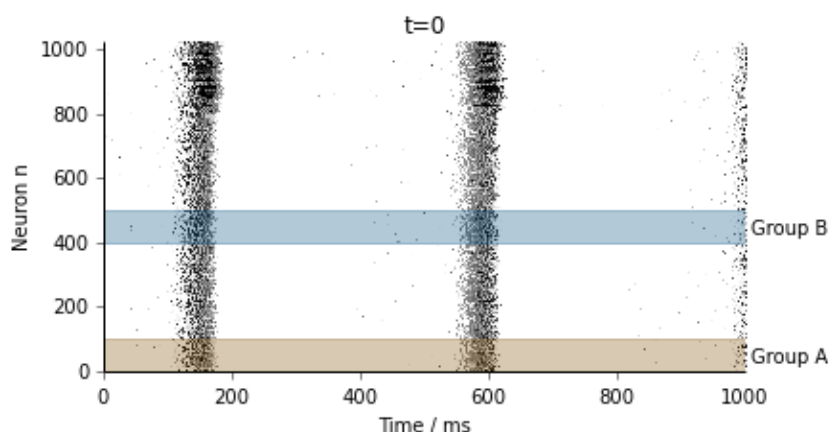
⁶ The choice of 1024 neurons speeds up the simulation by the GPU, as it ensures that several two dimensional arrays used in the algorithm are aligned in GPU memory for optimal access patterns.

Where Δw is the change in weight for a given synapse, and is determined at most twice for each post-synaptic spike – once for the difference between the post-synaptic spike time and the most recent prior pre-synaptic spike ($t_{pre} \leq t_{post}$, potentiation), and once for the difference between the post-synaptic spike time and the nearest future pre-synaptic spike ($t_{pre} > t_{post}$, depression). All these parameters are tunable and different combinations will be thoroughly investigated in the next phase of work.

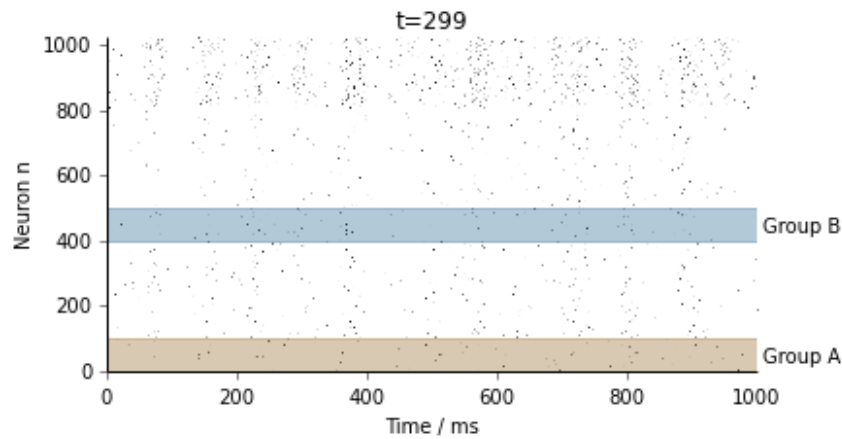
Our aim is to show that the network can perform causal learning. An initial requirement is that it must be able to distinguish directional inputs. That is, we must be able to encode “A causes B” as distinct from “B causes A”. Thus the first experiment demonstrates that a directional relationship between two event types can be encoded in the network.

In order to test this, we will assign one set of 100 neurons as group A, and another set of 100 neurons as group B. We train the network by stimulating group A then (very shortly later) group B. The STDP learning should allow the network to store the directional relationship between the two groups. Then we can recall the information by stimulating only one group – stimulating group A should yield activity in both groups A and B, since A is encoded as a cause of B, whereas stimulating group B should not generate activity in group A.

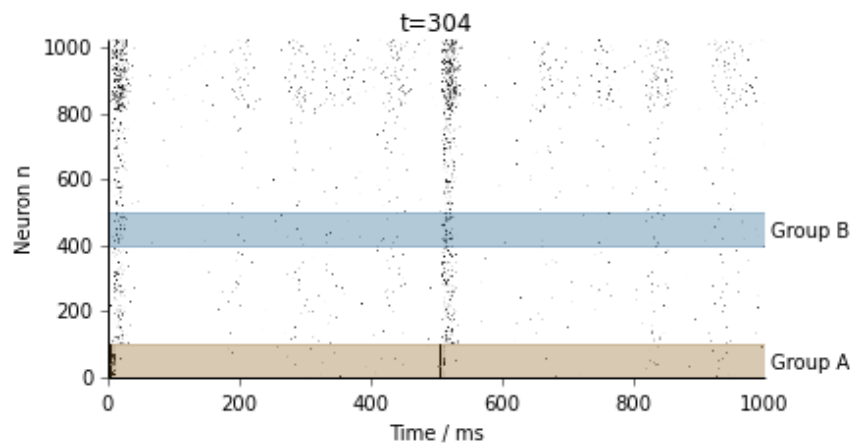
To begin with, the network is initialised with uniform weights as described above, and allowed to evolve over time. A small 20mA perturbation is added to the input of one randomly chosen neuron at each time step. In the first second of activation, we see a 2-3Hz oscillation in the entire network, as shown by the raster plot below.



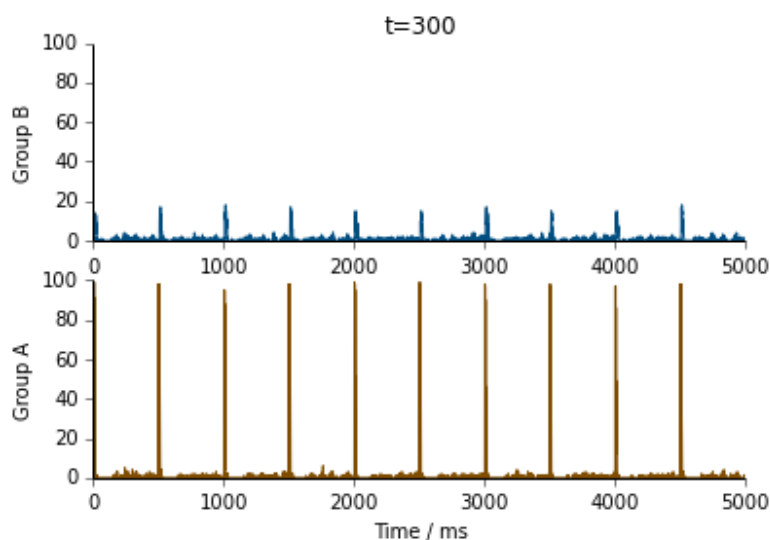
We allow the network to settle for 5 minutes of simulated time, with STDP updates being applied to synaptic weights after each second of simulated time. This allows the network to reach a quiescent state with a low level of spontaneous activity, as shown by the raster plot below. The 2-3Hz oscillation has died away, though there is a very weak oscillation at around 15Hz.



We now test the effect of stimulating group A *before* training the network. This determines what the un-trained response of the network is to a stimulus. We apply every 0.5s a stimulus of 50mA lasting 10ms to every neuron in group A, for 5 seconds in total. This raster plot shows the last second of this stimulation:

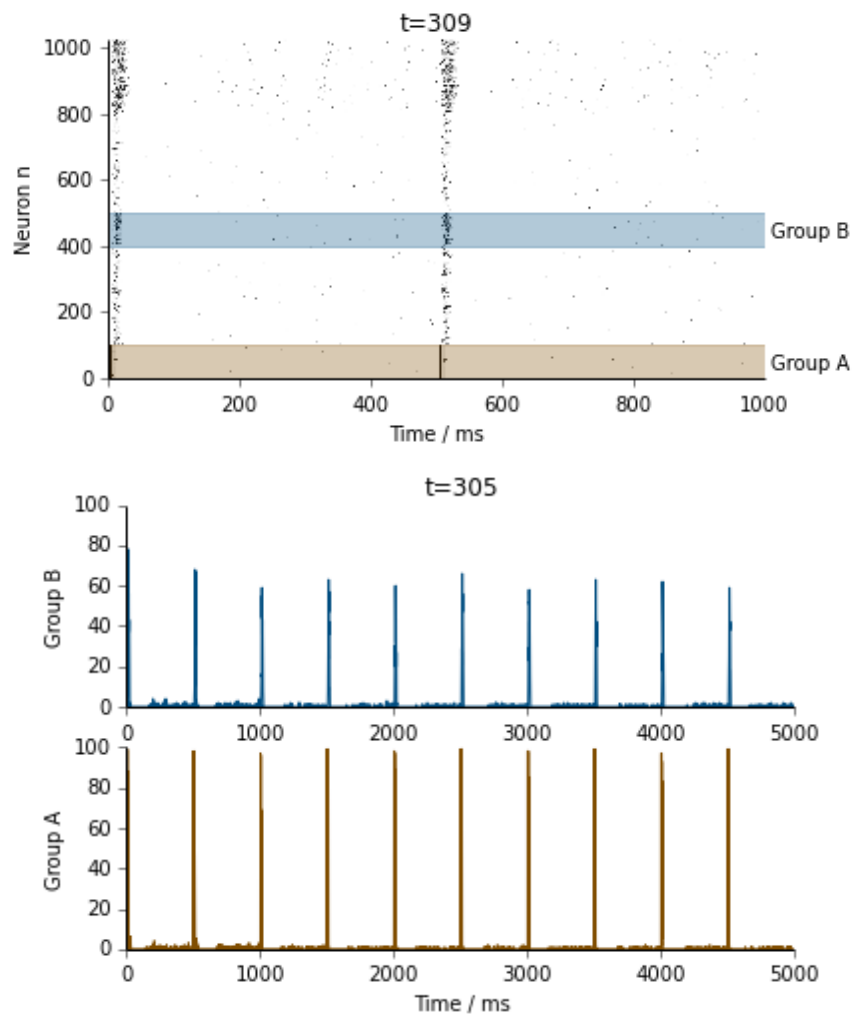


The number of firings in each group over the entire 5s test period is shown below:



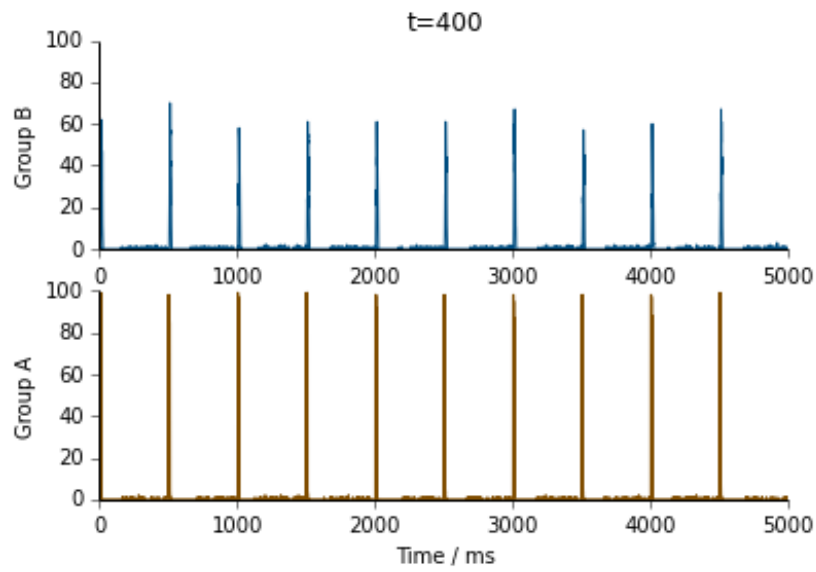
Stimulating group A therefore initially results in a large amount of activity in the group A neurons, as is to be expected, and a small amount of activity also in group B, since the network is untrained and connections will exist between the two groups.

We now train the network by stimulating group A for 10ms, then group B immediately after for another 10ms. Again the stimulation pattern is applied repeatedly every 0.5s, and STDP updates are performed every 1s of simulation time. These plots show the activity after the first 5s of training:



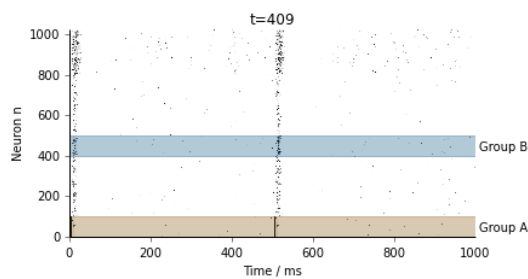
There is considerable activity in both groups since they are both being stimulated. Note that since group B is stimulated later, it has lower activation, since the activity in group A will excite inhibitory neurons, which in turn will be connected to group B.

Training continues for a further 95s (giving 100s of training time in total). This plots show the activity in the final 5 seconds of the training period:

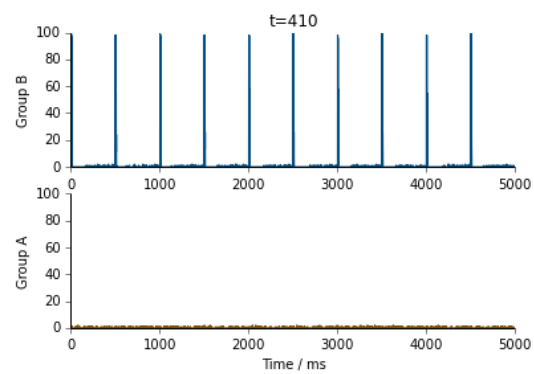
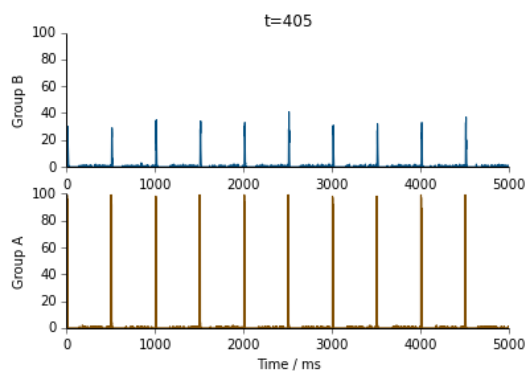
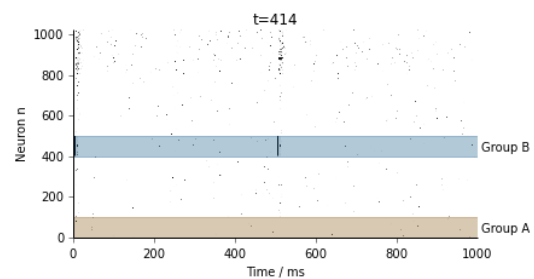


Now tests are performed by first stimulating group A only, then stimulating group B only. The STDP update rule is not applied during this “recall” phase:

Stimulate A



Stimulate B



The above plots show that when A alone is stimulated, activity occurs in B, but when B is stimulated no activity appears in group A. This is a promising result, as it shows that the directionality of the relationship can be encoded in the network using only STDP.

There are a number of extensions needed to demonstrate broader causal learning in the STDP network. To begin with, the model can be extended to probabilistic relationships as is typical in the elementary causal judgement scenarios discussed in the section on causality in D2.3. For example, if A causes B only some of the time, can the probability of this relationship be encoded? Furthermore, causal inference often makes use of the concept of *screening*, whereby correlations are not considered causal if there is a background factor that “conditions out” the correlation. For example, adopting the Granger approach, if activity in group B was more strongly correlated with some other activity in the network, or prior activity in group B, than it is with group A, we would expect not to infer the “A causes B” relationship. It remains to be seen whether this kind of screening off can be successfully encoded. The next phase of work will involve such investigations along with tuning the parameters of the simulation to match those of the physical cultures and to repeat the above experiment with the full spatial MEA simulation.

8 SORN based Path Evolution Algorithm Results

8.1 Re-implementation and analysis of the SORN model

The cortical organization is a nonrandom organization produced by *self-organization* mechanisms, where neural activity patterns are determined by network structure, and activity patterns shape network structure via plasticity mechanisms. This self-organization needs positive feedback (reinforcing) mechanisms and factors limiting the effects of the positive feedback. The major self-reinforcing factor in neuroscience is the synaptic strengthening of two correlated firing (groups) of neuron (the Hebb’s Rule). The limiting factor, being essential for proper self-organization, in this case is synaptic scaling (normalization of the neuron synaptic efficacies). The proper combination of these two factors can result in the formation of structural patterns and can describe the distribution of synaptic strength of local excitatory connections in the cortex.

The SORN model (self-organizing recurrent neural network) (Lazar et al. 2009, Zheng et al. 2013) consists of N^E excitatory neurons and $N^I = 0.2 \cdot N^E$ inhibitory neurons connected through weighted synaptic connections. Self-connections of excitatory neurons and connections between inhibitory neurons are forbidden. The excitatory to inhibitory (W^{EI}) connections have an all-to-all topology and the strengths remain fixed at their uniformly distributed random initial values. The initial connectivity between excitatory to excitatory (W^{EE}) and inhibitory to excitatory (W^{EI}) neurons are sparse and random.

The dynamics of the system uses five different plasticity mechanisms. (i.) Spike-time dependent plasticity (STDP) changes the strength of excitatory-excitatory connections in a temporally asymmetric casual fashion. (ii.) Inhibitory-excitatory connections are also subject to spike-time dependent plasticity (iSTDP) that balances the amount of excitatory and inhibitory drive a neuron is

receiving. (iii.) An intrinsic plasticity mechanism (IP) changes the firing threshold of excitatory neurons to maintain a low average firing rate: a neuron that has been just fired increases its threshold while an inactive neuron's threshold decreases with a small amount. (iv.) The synaptic normalization mechanism normalizes all the excitatory weights of a neuron. (v.) New connections between excitatory cells are generated at a small rate to maintain the overall structural plasticity and balance the synaptic elimination caused by the STDP.

According to the above description of the model, we can formulate all plasticity mechanisms, respectively:

$$\Delta W_{ij}^{EE}(t) = \eta_{STDP} (x_i(t)x_j(t-1) - x_i(t-1)x_j(t))$$

$$\Delta W_{ij}^{EI}(t) = -\eta_{inhib} y_j(t-1)(1 - x_i(t)(1 + 1/\mu_{IP}))$$

$$T_i^E(t+1) = T_i^E(t) + \eta_{IP} (x_i(t) - \mu_{IP})$$

$$W_{ij}^{EE}(t) \xleftarrow{norm} W_{ij}^{EE}(t) / \sum_j W_{ij}^{EE}(t)$$

where upper indices E and I denote excitatory and inhibitory neurons, $W_{ij}(t)$ is the strength of the synaptic connection between neurons i and j at time t . η -s refer to the strength of the different plasticity mechanisms, while $x_i(t)$ and $y_j(t)$ are the (binary) activity values of excitatory and inhibitory neurons, respectively, at time t . With these plasticity mechanisms the evolution of the network is governed by the following two equations:

$$x_i(t+1) = \Theta \left(\sum_{j=1}^{N^E} W_{ij}^{EE}(t)x_j(t) - \sum_{k=1}^{N^I} W_{ik}^{EI}(t)y_k(t) - T_i^E(t) + \xi_i^E(t) \right)$$

$$y_i(t+1) = \Theta \left(\sum_{j=1}^{N^E} W_{ij}^{IE}x_j(t) - T_i^I(t) + \xi_i^I(t) \right)$$

where $\Theta(\cdot)$ is the Heaviside function, and $\xi_i^{E,I}(t)$ represents the input to the i^{th} excitatory or inhibitory neuron at time t .

As a first step, we've tried to reproduce the distribution of the synaptic weights. According to recent investigations (Song et al., 2005, Sarid et al. 2007, Loewenstein et al. 2011) the overall distribution of synaptic strengths has approximately lognormal distribution: a small number of connections are responsible for the majority of all synaptic weights. (Only about 20% of synapses cover the 50% of the total synaptic efficacy).

To test the distribution of excitatory-to-excitatory connections, we have supplied both excitatory and inhibitory neurons with a Gaussian random noise as input with zero mean and $\sigma_\xi = 0.04$ variance: $\xi_i^E(t), \xi_i^I(t) \in N(0, \sigma_\xi)$. With the parameterization of the original model (Zheng et al. 2013). ($N^E = 200; \eta_{STDP} = 0.004; \eta_{inhib} = 0.001; \mu_{IP} = 0.1; T_{max}^E = 1; T_{max}^I = 0.5$) we could not reproduce the results of the original article (nor Figures 1., 2., 3.). To get results that are at least similar to those reported, we had to change four parameters of the system from their values described in the published paper. In case of $\eta_{STDP} = 0.003(0.004); \eta_{inhib} = 0.001(0.01); \sigma_\xi = 0.05; T_{max}^E = 0.5(1)$

(original values in brackets) and by assuming an all-to-all initial topology of the inhibitory to excitatory connections were we able to reproduce the firing pattern and weight distribution of the system. For the similarity of weight distribution, see the following figure.

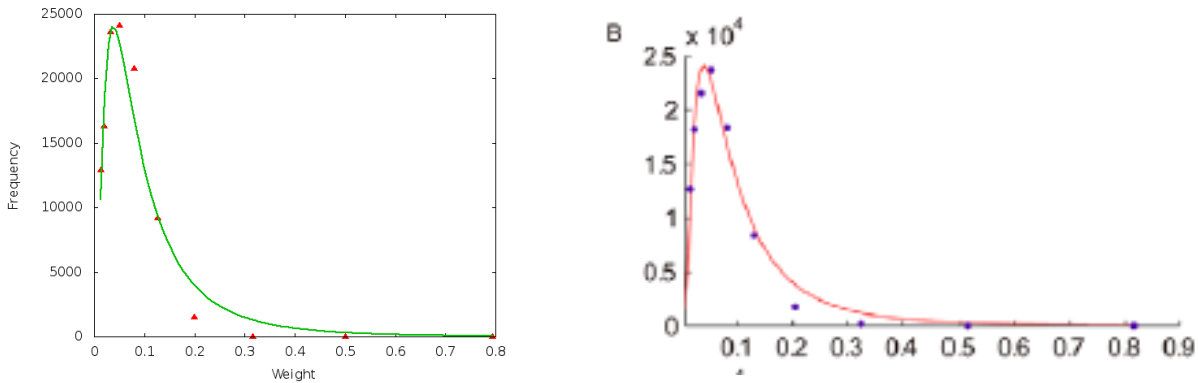


Figure 16: Distribution of synaptic strengths of excitatory to excitatory connections and lognormal fit at time $t = 104$. Results of our simulation with the modified parameter set (left) and Fig 2B of the article. Results are clearly similar.

We used both the original (published) and our modified parameter sets (only with which we could reproduce the results of the paper) to try to reproduce the long term dynamics of excitatory connections (cf. Fig. 2 of the original paper). Neither the original nor the modified parameters were able to reproduce the long term dynamics of the excitatory to excitatory connections published in the paper.

Comparing to what the authors have reported, the decay phase does not descend as low in our simulation as in theirs and the recovery stage also does not reach the same percentage in equilibrium as in their simulation. In the next figure, the dynamics of our modified parameter set can be compared by that of the original simulation published.

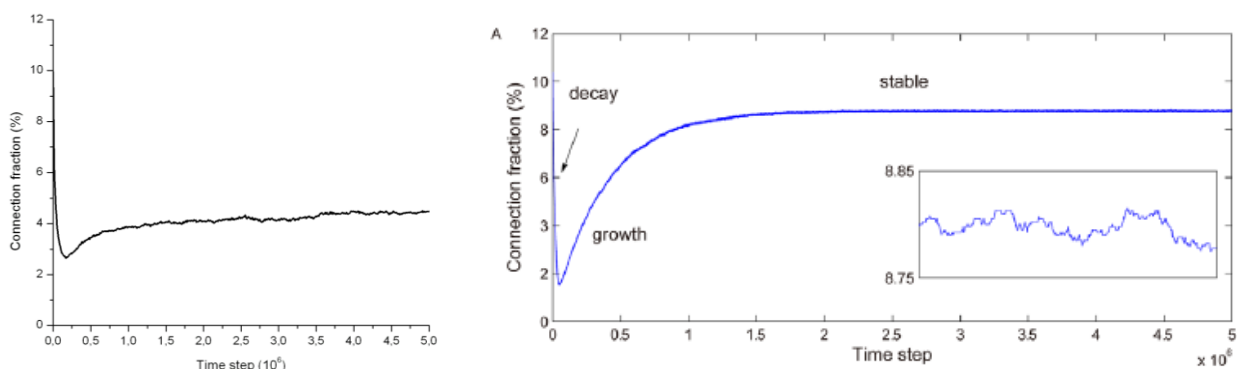


Figure 17: Fraction of existing excitatory to excitatory connections over five million time steps. The behavior is only qualitatively similar (decay, recovery and “steady state” phase), differences are discussed in the main text. Left panel: our simulation, right panel: Fig. 3A of the original article.

After the results of the SORN model were partially reproduced we've tested its response to constant input (instead of random) as we were interested how the plasticities behave when the input does not change. We fed 10 randomly chosen excitatory neurons (as input neurons) with a constant input of 1: $\xi_i^{E,I}(t) = 1, (i = 1...10)$. We have found that the response of the system is unexpected: after a transient period, there is an oscillating behavior with two phases repeating in a relatively periodic way: a silent phase (where almost all neurons are silent) is followed by a short activation phase where almost all neurons fire (see the next figure), Analysis has revealed that this is mainly due to the threshold plasticity mechanism: because of the (relatively high) constant input, firing thresholds of both input and hidden neurons are increasing continuously, until a point where the input signal cannot activate the input neurons. As a consequence, the system will ultimately reach a silent phase as no neuron can be activated due to the high threshold values. This causes all the neurons in the system to gradually decrease their firing thresholds (via the plasticity mechanism) until an input signal could finally pass over the threshold of an input neuron, and by entering the system consisting of low-threshold neurons, it causes a mass activation, returning the system to where it was started. This activation-deactivation phase is repeated in a relatively periodic way.

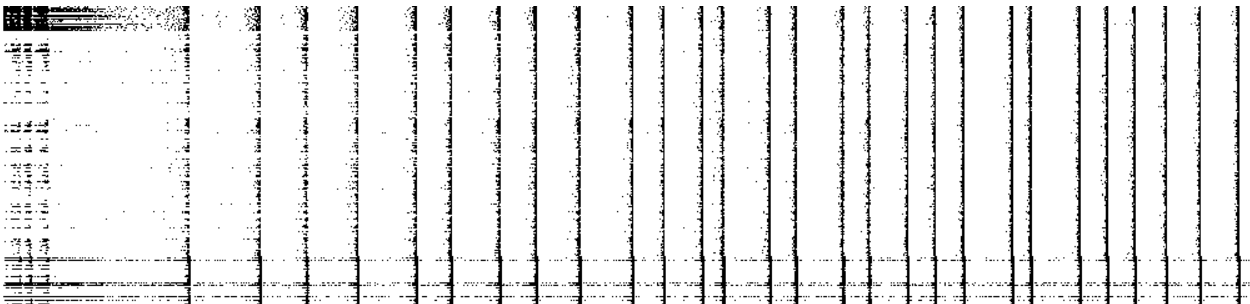


Figure 18: Time series of activities of excitatory and inhibitory neurons. The first 10 lines correspond to the input neurons, the next 190 are the other excitatory neurons and the last 20 correspond to the inhibitory neurons. Black cells denote active, white cells denote silent neurons (1000 time steps).

8.2 Reproducing pattern of a SORN model

Purpose

In this experiment, we trained the SORN model by feeding the same pattern to the inputs repeatedly. We want to see if the network keeps on reproducing the pattern by itself after a long time of training without the influence of the external inputs

Procedure

- The network used 100 excitatory neurons and 20 inhibitory neurons. Then I divided the 100 excitatory neurons into 4 groups, each groups contained 25 neurons. We named the group A,

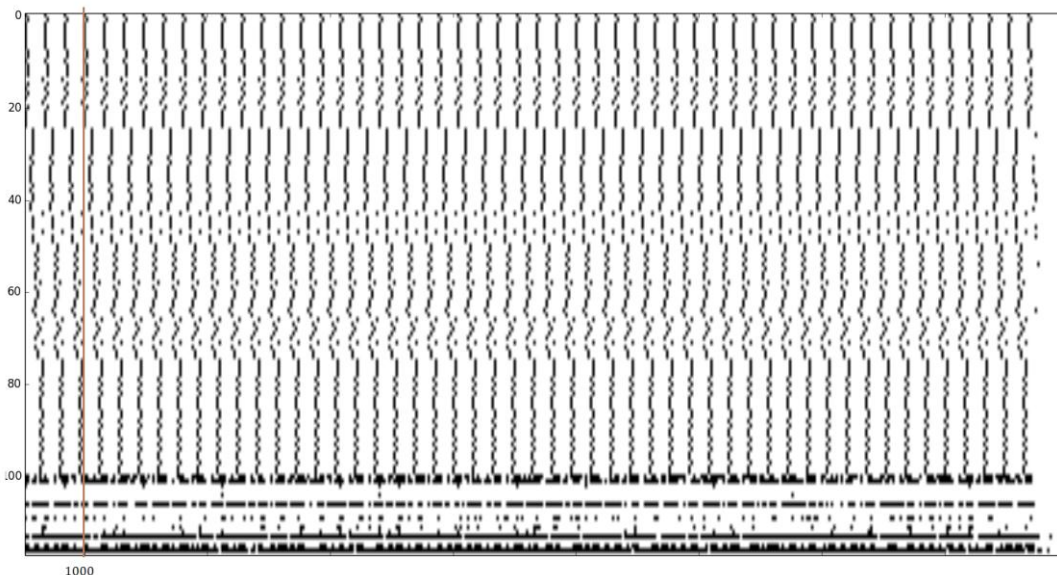
B, C and D respectively. At the training stage of the program, we activate the four groups of neurons alternatively for 2 time steps each. Activate neurons in group A at time=0~1, neurons in group B at time = 2~3, neurons in group C at time = 4~5 and neurons in group D at time = 6~7 and repeat the procedure 500 times. Finally at the test stage, I give all the neurons a small input to drive the network working by itself for 1000 time steps

Data

Number of excitatory neurons, $N^E = 100$
Number of inhibitory neurons, $N^I = 20$
Max Threshold of excitatory neurons, $T_{MAX}^E = 0.5$
Max Threshold of inhibitory neurons, $T_{MAX}^I = 1.0$
Probability of initial E-E connection, $P_{EE} = 0.1$
Probability of initial I-E connection, $P_{IE} = 1.0$
Probability of adding new E-E connection, $P_{ADDEE} = 0.1$

Discussion

The following figure shows the spikes behavior in the last 1000 time steps. The x-axis represents the time and y-axis represents the neuron behaviors which a black dot indicate a spike made by the corresponding neuron. The red line denoted the start point of the testing stage and we can obviously observed that the pattern is keep on reproducing even though we have stopped feeding patterns to the input.



Preliminary Lab

See attachment *sorn_experiment1.py*

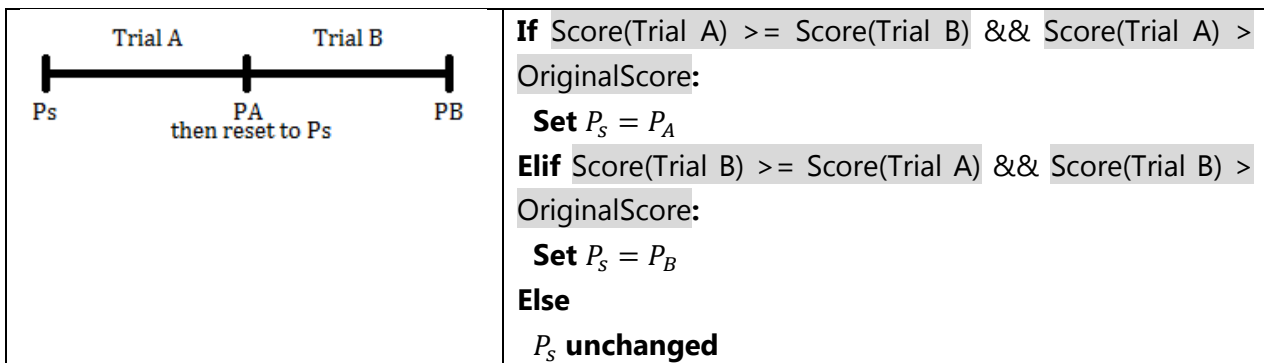
8.3 Neuronal genetic algorithm in the SORN network

Purpose

In this experiment, we tried to implement the neuronal genetic algorithm in a SORN network and see if it is possible to solve the all ones problem.

Procedure

- The simulation conducted in a series of trials. In each trial, we activate neuron 0 at time steps = 0, and then activate neuron 1 at time steps = 1 and up to activate neuron 9 at time steps = 9. The above is repeated for five times so that each trial consisted of 50 time steps. Neuron 80~100 are the outputs. After the trial, we calculate the number of firing of each single output neuron in the trial. If the number of firing is greater than a threshold. We switch the neuron "ON", else otherwise, switch it "OFF". Then finally we get obtain the fitness according to the number of "ON" neurons with a bit mask applied.
- Before each trial, we save all the parameters (weights and thresholds) to P_s , perform a mutation and obtain the fitness by Trial A. Then the parameters are saved to P_A and parameters will be reset to P_s . After, perform another mutation and obtain the fitness by Trial B. Then the parameters are saved to P_B . Finally, we use the following policy to reset the parameters to P_s and a new pair of trials will be continued.

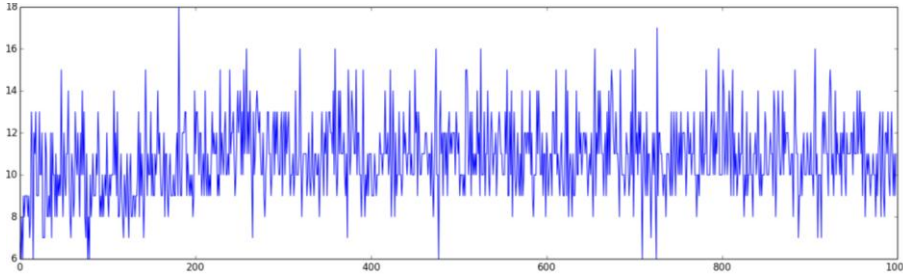


Data

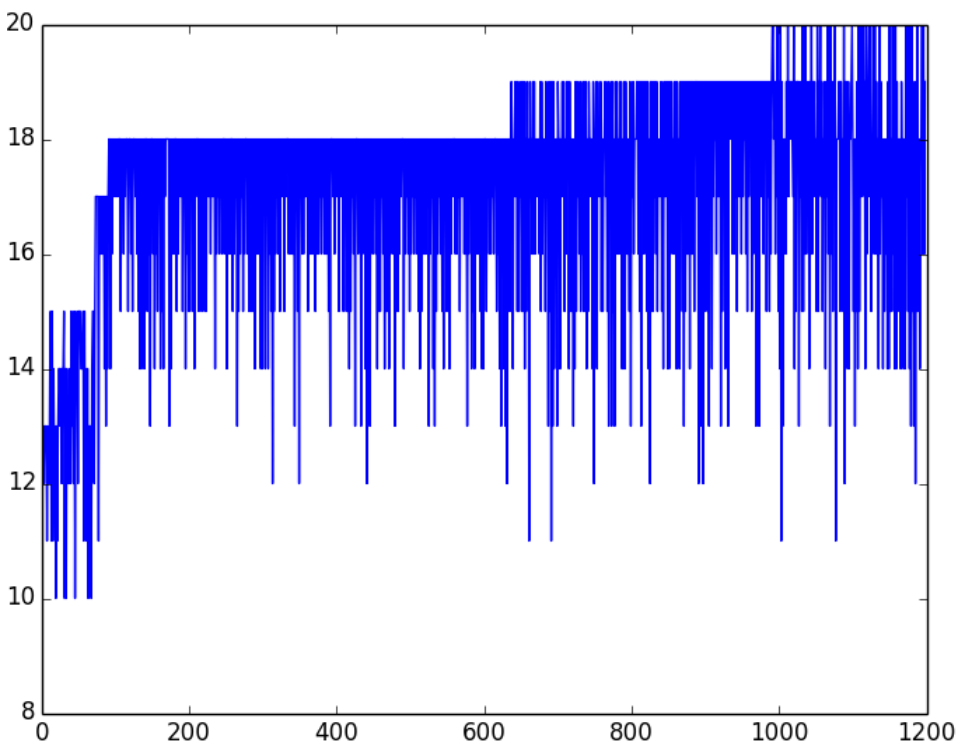
Number of excitatory neurons, $N^E = 100$
Number of inhibitory neurons, $N^I = 20$
Max Threshold of excitatory neurons, $T_{MAX}^E = 0.5$
Max Threshold of inhibitory neurons, $T_{MAX}^I = 1.0$
Probability of initial E-E connection, $P_{EE} = 0.1$
Probability of initial I-E connection, $P_{IE} = 1.0$
Probability of adding new E-E connection, $P_{ADDEE} = 0.1$
Threshold of firing neurons = 8
Probability of mutation = 0.3
Strength of mutation = 0.5

Discussion

The following shows the result of running the procedure for 1000 iterations, the x-axis represents the time steps and the y-axis represents the fitness. As we can see in the diagram, the fitness keeps oscillating along the time and the noise made by the SORN model affected the fitness from going up.



The following is another diagram showing that what will happen if we switch off all the plasticity in the SORN model. STDP, iSTDP and IP is switched OFF in this experiment and we can observe that the fitness goes up and finally reaches 20 which is the optimal solution we expected.



In conclusion, we found from these experiments that the SORN model seems not to be helpful in solving the all ones problem as the noise will affect the behavior of the neurons so that a high score cannot be maintained in trials.

Preliminary Lab

See attachment *sorn_experiment2.py*

9 References

- Asai, Y., Nomura, T., Sato, S., Tamaki, A., Matsuo, Y., Mizukura, I., et al. (2003). A coupled oscillator model of disordered interlimb coordination in patients with Parkinson's disease. *Biological Cybernetics* **88**:152–162.
- Barlas, Y., & Yasarcan, H. (2006). Goal setting, evaluation, learning and revision: A dynamic modeling approach. *Evaluation and Program Planning* **29**(1):79–87.
- Buzsaki, G. (2010). Neural syntax: Cell assemblies, synapse ensembles, and readers. *Neuron* **68**:362–385.
- Cho JH, Bayazitov I, Meloni EG, Myers KM, Carlezon Jr WA, Zakharenko SS, Bolshakov VY (2011) Coactivation of thalamic and cortical pathways induces input timing-dependent plasticity in amygdala. *Nature Neuroscience* **15**(1): 113-122.
- Davis, P. (1990). Application of optical chaos to temporal pattern search in a nonlinear optical resonator. *Japanese Journal of Applied Physics* **29**:L1238–L1240.
- Doya K, Ishii S, Pouget A, Rao RPN (2007) *Bayesian Brain: Probabilistic Approaches to Neural Coding*. MIT-Press.
- Dudman JT, Tsay D, Siegelbaum A (2007) A role for synaptic inputs at distal dendrites: Instructive signals for hippocampal long-term plasticity. *Neuron* **56**: 866-879.
- Fernando C, Karishma KK, Szathmáry E (2008) Copying and evolution of neuronal topology. *PLoS ONE* **3**(11): e3775.
- Fernando C, Szathmáry E, Husbands P (2012) Selectionist and evolutionary approaches to brain function: a critical appraisal. *Frontiers in Computational Neuroscience* **6**(24).
- Fernando, C. (2013). From Blickets to Synapses: Inferring Temporal Causal Networks by Observation. *Cognitive Science*, 1–45. doi:10.1111/cogs.12073
- Fidjeland, A. K., Gamez, D., Shanahan, M. P., & Lazdins, E. (2013). Three tools for the real-time simulation of embodied spiking neural networks using GPUs. *Neuroinformatics*, *11*(3), 267–90. doi:10.1007/s12021-012-9174-x
- Fiser J, Berkes P, Orban G, Lengyel M (2010) Statistically optimal perception and learning: from behavior to neural representation. *Trends in Cognitive Sciences* **14**: 119–130.

- FitzHugh R. (1961) Impulses and physiological states in theoretical models of nerve membrane. *Biophysical J.* **1**:445-466
- Freeman, W. J., & Viana Di Prisco, G. (1986). EEG spatial pattern differences with discriminated odors manifest chaotic and limit cycle attractors in olfactory bulb of rabbits. In G. Palm & A. Aertsen (Eds.), *Brain theory* (pp. 97–119). New York:Springer-Verlag.
- Goodman, D., & Brette, R. (2008). Brian: a simulator for spiking neural networks in python. *Frontiers in Neuroinformatics*, 2(November), 5. doi:10.3389/neuro.11.005.2008
- Griffiths, T. L., & Tenenbaum, J. B. (2005). Structure and strength in causal induction. *Cognitive Psychology*, 51(4), 334–84. doi:10.1016/j.cogpsych.2005.05.004
- Guevara, M. R., Glass, L., Mackey, M. C., & Shrier, A. (1983). Chaos in neurobiology, *IEEE Transactions on Systems, Man, and Cybernetics, SMC-13*: 790–798.
- Hodgkin, A., and Huxley, A. (1952): A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.* **117**:500–544.
- Hong Y, Kwong S, Chang Y, Ren Q (2008) Unsupervised feature selection using clustering ensembles and population based incremental learning algorithm. *Pattern Recognition* **41**: 2742-2756.
- Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6), 1569–72. doi:10.1109/TNN.2003.820440
- Izhikevich, E. M. (2006). Polychronization: computation with spikes. *Neural Computation*, 18(2), 245–82. doi:10.1162/089976606775093882
- Jacobs RA, Jordan MI, Nowlan SJ, Hinton GE (1991) Adaptive mixtures of local experts. *Neural Computation*, **3**(1):125-130.
- Kinjo, K., Nabeshima, C., Sangawa, S., & Kuniyoshi, Y. (2008). A neural model for exploration and learning of embodied movement patterns. *Journal of Robotics and Mechatronics*, 20(3), 358–366.
- Klampfl, S. and Maass, W. (2013) Emergence of Dynamic Memory Traces in Cortical Microcircuit Models through STDP, *Journal of Neuroscience* **33**(28):11515–11529
- Kopell, N., Whittington, M. A., & Kramer, M. A. (2011). Neuronal assembly dynamics in the beta1 frequency range permits short-term memory. *PNAS* **108**(9):3779–3784.

Kuniyoshi, Y., & Suzuki, S. (2004). Dynamic emergence and adaptation of behaviour through embodiment as coupled chaotic field. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems* (pp. 2042–2049). Piscataway, NJ: IEEE.

Kuniyoshi, Y., & Sangawa, S. (2006). Early motor development from partially ordered neural-body dynamics: Experiments with a cortico-spinal-musculoskeletal model. *Biological Cybernetics* **95**:589–605.

Larrañaga, P. and Lozano, J. (Eds.) (2002) *Estimation of distribution algorithms: A new tool for evolutionary computation*. Boston: Kluwer Academic Publishers.

Lazar, A., Pipa, G., & Triesch, J. (2009) SORN: a self-organizing recurrent neural network. *Frontiers in Computational Neuroscience*, **23**:1-9

Loewenstein, Y., Kuras, A., & Rumpel, S. (2011) Multiplicative dynamics underline the emergence of lognormal distribution of spine sizes in the neocortex in vivo, *The Journal of Neuroscience*, **31**:9481-9488

Maheswaranathan, N., Ferrari, S., Vandongen, A. M. J., & Henriquez, C. S. (2012). Emergent bursting and synchrony in computer simulations of neuronal cultures. *Frontiers in Computational Neuroscience*, 6(April), 15. doi:10.3389/fncom.2012.00015

Menzies JRW, Porrill J, Dutia M, Dean P (2010) Synaptic plasticity in medial vestibular nucleus neurons: Comparison with computational requirements of VOR adaptation. *PLoS ONE* **5**(10): e13182.

Nageswaran, J. M., Dutt, N., Krichmar, J. L., Nicolau, A., & Veidenbaum, A. (2009). Efficient simulation of large-scale Spiking Neural Networks using CUDA graphics processors. *2009 International Joint Conference on Neural Networks*, 2145–2152. doi:10.1109/IJCNN.2009.5179043

Nessler B, Pfeiffer M, Buesing L, Maass W (2013) Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLoS Computational Biology* **9**(4): e1003037.

Nessler B, Pfeiffer M, Maass W (2008) Hebbian learning of Bayes optimal decisions. Proceedings of NIPS 2008.

Nowotny, T. (2011). Flexible neuronal network simulation framework using code generation for NVidia® CUDA™. *BMC Neuroscience*, **12**(Suppl 1), P239. doi:10.1186/1471-2202-12-S1-P239

Ohgi, S., Morita, S., Loo, K., & Mizuike, C. (2008). Time series analysis of spontaneous upper-extremity movements of premature infants with brain injuries. *Physical Therapy* **88**(9):1022–1033.

Pelikan, M., Goldberg, D. and Cantu-paz, E. (2000) Linkage Problem, Distribution Estimation and Bayesian Networks, *Evolutionary Computation* **8**(3): 311-340.

Pelikan, M., Goldberg, D., and Lobo, F. (2002), A Survey of Optimization by Building and Using Probabilistic Models, *Computational Optimization and Applications* **21**: 5–20.

Pelikan, M. (2005), *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*. Berlin:Springer.

Pitti, A., Niiyama, R., & Kuniyoshi, Y. (2010). Creating and modulating rhythms by controlling the physics of the body. *Autonomous Robots*, **28**(3):317–329.

Rao RPN, Olshausen BA, Lewicki MS (2002) *Probabilistic Models of the Brain*. MIT Press.

Rapp, P., Zimmerman, I., Albano, A., Deguzman, G., & Greenbaun, N. (1985). Dynamics of spontaneous neural activity in the simian motor cortex: The dimension of chaotic neurons. *Physics Letters A* **110**(6):335–338.

Riley, M., & Turvey, M. (2002). Variability and determinism in motor behaviour. *Journal of Motor Behavior*, **34**(2), 99–125.

Sarid, L., Bruno, R., Sakmann, B., Segev I., & Feldmeyer D. (2007): Modeling a layer 4-to-2/3 module of a single column in rat neocortex: interweaving of in vitro and in vivo experimental observations PNAS, **104**:16353-16358

Shim, Y. (2013) *Chaotic Exploration and Learning of Locomotor Behaviours*. PhD Thesis, Dept. Informatics, University of Sussex.

Shim, Y. S., & Husbands, P. (2010). Chaotic search of emergent locomotion patterns for a bodily coupled robotic system. In *Proceedings of 12th International Conference on the Synthesis and Simulation of Living Systems (ALIFE XII)* (pp. 757–764). Cambridge, MA: MIT Press.

Shim, Y., & Husbands, P. (2012). Chaotic exploration and learning of locomotion behaviours. *Neural Computation* **24**(8):2185–2222.

Skarda, C., & Freeman, W. (1987). How brains make chaos in order to make sense of the world. *Behavioral and Brain Sciences*, **10**:161–195.

Song, S., Sjöström, P., Reihl, M., Nelson, S., & Chklovskii, D. (2005) Highly non-random features of synaptic connectivity in local cortical circuits, *PLoS Biology*, **3**:e68

Sporns O, Tononi G, Edelman GM (2002) Theoretical neuroanatomy and the connectivity of the cerebral cortex. *Behavioural Brain Research* **135**(1-2): 69–74.

Terman,D.,&Rubin, J. (2007).Neuronal dynamics and the basal ganglia. *SIAM News*, **4**(2). Available online at <http://siam.tekdevelopment.com/old-issues/2007/march-2007/>.

Thivierge JP, Marcus GF (2007) The topographic brain: from neural connectivity to cognition. *Trends in Neurosciences* **30**(6): 251-259.

Varela, F., Lachaux, J., Rodriguez, E., & Martinerie, J. (2001). The brainweb: Phase synchronization and large-scale integration. *Nat. Rev. Neurosci.*, **2**(4):229–239.

Wright, J., & Liley, D. (1996). Dynamics of the brain at global and microscopic scales: Neural networks and the EEG. *Behavioral and Brain Sciences* **19**:285–320.

Zheng, P., Dimitrakakis, C., & Triesch, J. (2013) Network self-organization explains the statistics and dynamics of synaptic connection strength in cortex. *PLoS Computational Biology*, **9**:e1002848

```

# ===== IMPLEMENTATION OF THE SORN MODEL===== #

import aux01
import numpy as np
import matplotlib.pyplot as plt
import math

# ===== INITIALIZING CONSTANTS ===== #
thresholdOn = 8 # THRESHOLD OF DEFINING ON OF FIRING NEURONS
N = 20 # PROBLEM DIMENSIONALITY
pMut = 0.3 # PROBABILITY OF MUTATION
etaMut = 0.5 # NOISE STRENGTH OF MUTATION
NN_E = 100; # EXCITORY NEURONS
NN_I = 20; # 0.2*NN_E INHIBITORY NEURONS
NUM_I = 100; # NUMBER OF THE INPUT NEURONS
# THE INPUT NEURONS ARE THE FIRST NUM_I EXCITATORY
NEURONS
pEE = 0.1; # PROBABILITY OF INITIAL E --> E CONNECTIONS
pEI = 1.0; # PROBABILITY OF INITIAL I --> E CONNECTIONS
TE_MAX = 1.0; # MAXIMUM OF EXCITATORY THRESHOLD
TI_MAX = 0.5; # MAXIMUM OF INHIBITORY THRESHOLD
sig2 = 0.001; # SD OF RANDOM NOISE ON EXCITATORY NEURONS
mulP = 0.1;
etaIP = 0.01; # COEFFICIENT OF EXCITATORY THRESHOLD PLASTICITY
etaSTDP = 0.004; # COEFFICIENT OF STDP
etaINHIB = 0.001; # COEFFICIENT OF INHIBITORY STDP
paddEE = 0; # PROBABILITY OF ADDING A NEW E --> E CONNECTION

SAVE_STEPS = 2000; # NUMBER OF TIME STEPS TO SAVE

strength = 0.8;
duration = 1; # NUMBER OF TIME STEPS OF EACH ACTIVATION
stateliteration = 50; # NUMBER OF STATES IN EACH ITERATION
iteration = 100; # NUMBER OF ITERATION

scoreFig = []; # PLOTTING THE SCORES
STEPS = duration * stateliteration * iteration; # NUMBER OF TIME STEPS

```

```

VISUAL = 1;                                # VISUALISATION
MASK = np.random.binomial(1,0.5,N) # INITIALZING THE MASK

# ===== FUNCTION TO DECLARE 2D ARRAYS ===== #

def declareArray(m, n):
    A = [];
    for i in range(m):
        A.append([0.0] * n);
    return A;

# ===== INITIALIZING ARRAYS ===== #
te = [0.0] * NN_E; # EXCITATORY NEURONS
ti = [0.0] * NN_I; # INHIBITORY NEURONS
inp = [0.0] * NUM_I; # INPUT
x = declareArray(2, NN_E); # ACTIVITY STATE OF EXCITATORY NEURONS
y = declareArray(2, NN_I); # ACTIVITY STATE OF INHIBITORY NEURONS
wee = declareArray(NN_E, NN_E);
wei = declareArray(NN_E, NN_I);
wie = declareArray(NN_I, NN_E);
img = declareArray((NN_E+NN_I), SAVE_STEPS);
stringPattern = [0.0] * N;
score = [0.0] * 3;

fix = 0;

tempx = declareArray(2, NN_E);
tempy = declareArray(2, NN_I);
tempwee1 = declareArray(NN_E, NN_E);
tempwee2 = declareArray(NN_E, NN_E);
tempwee3 = declareArray(NN_E, NN_E);
tempwei1 = declareArray(NN_E, NN_I);
tempwei2 = declareArray(NN_E, NN_I);
tempwei3 = declareArray(NN_E, NN_I);
tempwie1 = declareArray(NN_I, NN_E);
tempwie2 = declareArray(NN_I, NN_E);
tempwie3 = declareArray(NN_I, NN_E);
numberSpike = declareArray(N, iteration);

```

```

tempte1 = [0.0] * NN_E;
tempte2 = [0.0] * NN_E;
tempte3 = [0.0] * NN_E;
tempti1 = [0.0] * NN_I;
tempti2 = [0.0] * NN_I;
tempti3 = [0.0] * NN_I;

# ===== INITIALIZATION OF THE MODEL ===== #
def init():
    for i in range(0, NN_E):
        te[i] = aux01.randd() * TE_MAX;

    for i in range(0, NN_I):
        ti[i] = aux01.randd() * TI_MAX;

    for i in range(0, NN_I): # W^IE INIT (ALL-TO-ALL TOPOLOGY WITH
RANDOM[0,1] WEIGHTS)
        sum = 0;
        for j in range(0, NN_E):
            wie[i][j] = aux01.randd();
            sum = sum + wie[i][j];

        for j in range(0, NN_E):
            if (sum <> 0):
                wie[i][j] = wie[i][j] / sum;

    for i in range(0, NN_E): # W^EE INIT (SPARSE RANDOM WITH pEE PROBABILITY)
        sum = 0;
        for j in range(0, NN_E):
            if ((i <> j) and (aux01.randd() <= pEE)): # >>NO SELF-CONN
                wee[i][j] = aux01.randd();
                sum = sum + wee[i][j];
            else:
                wee[i][j] = -99.0; # NOT CONNECTED

        for j in range(0, NN_E):
            if ((sum <> 0.0) and (wee[i][j] >= 0.0)):
                wee[i][j] = wee[i][j] / sum;

```

```

for i in range(0, NN_E): # W^EI INIT (RANDOM WITH pEI PROBABILITY)
    sum = 0;
    for j in range(0, NN_I):
        if (aux01.randd() <= pEI):
            wei[i][j] = aux01.randd();
            sum = sum + wei[i][j];
        else:
            wei[i][j] = -99.0;    # NOT CONNECTED

    for j in range(0, NN_I):
        if ((sum <> 0) and (wei[i][j] >= 0)):
            wei[i][j] = wei[i][j] / sum;

for i in range(0, NN_E):
    x[0][i] = 0;

for i in range(0, NN_I):
    y[0][i] = 0;

# ===== IMPLEMENTING MODEL STEPS ===== #
def step(t):

    if (t % 100 == 0):
        print "Running step " + str(t) + " / " + str(STEPS);

    t0 = (t-1) % 2;
    t1 = t % 2;

    for i in range(0, NN_E): # EXCITATORY UPDATE (EQ.1)
        sum1 = 0.0;
        sum2 = 0.0;

        for j in range(0, NN_E):
            if (wee[i][j] >= 0.0):
                sum1 += wee[i][j] * x[t0][j];

        for j in range(0, NN_I):

```

```

        if (wei[i][j] >= 0.0):
            sum2 += wei[i][j] * y[t0][j];

sum = sum1 - sum2 + math.sqrt(sig2) * aux01.gasdev();

if (i < NUM_I):
    sum = sum + inp[i];

if (sum > te[i]):
    x[t1][i] = 1;
else:
    x[t1][i] = 0;

if fix == 0:
    for i in range(0, NN_I): # INHIBITORY UPDATE (EQ.2)
        sum1 = 0;

        for j in range(0, NN_E):
            sum1 += wie[i][j] * x[t0][j];

        sum1 += math.sqrt(sig2) * aux01.gasdev();

        if (sum1 > ti[i]):
            y[t1][i] = 1;
        else:
            y[t1][i] = 0;

    for i in range(0, NN_E): # PLASTICITIES (EE, EI) AND WEIGHTS UPDATES
        sum = 0; # E-E STDP (EQ.3)
        for j in range(0, NN_E):
            if ((wee[i][j] > 0.0) and (i <> j)):
                tt = etaSTDP * (float(x[t1][i]) * float(x[t0][j]) - float(x[t0][i]) *
float(x[t1][j]));

                if (wee[i][j] + tt <= 0):
                    wee[i][j] = -99;
                else:
                    wee[i][j] = wee[i][j] + tt;

```

```

sum = sum + wee[i][j];

for j in range(0, NN_E): # NORMALISATION (EQ.4)
    if ((sum <> 0) and (wee[i][j] > 0.0)):
        wee[i][j] = wee[i][j] / sum;

sum = 0; # EI ISTDP (EQ.7)

for j in range(0, NN_I):
    if (wei[i][j] > 0):
        tt = -1.0 * etaINHIB * y[t0][j] * (1-float(x[t1][i])) *
(1.0+1.0/float(muIP));
        if (wei[i][j] + tt <= 0):
            wei[i][j] = -99;
        else:
            wei[i][j] = wei[i][j] + tt;
            sum = sum + wei[i][j];

for j in range(0, NN_I):
    if ((sum <> 0) and (wei[i][j] > 0)):
        wei[i][j] = wei[i][j] / sum;

te[i] += etaIP * (float(x[t0][i]) - muIP); # THRESHOLD PLASTICITY
(EQ.5)

if (te[i] < 0):
    te[i] = 0;

if (aux01.randd() < paddee):# ADD A NEW E --> E CONNECTION WITH
PROBABILITY paddee AND STRENGTH 0.001
    ind = 0;
    sum = 0;

while (ind == 0):
    r1 = aux01.randl(NN_E);
    r2 = aux01.randl(NN_E);

    if ((wee[r1][r2] == -99) and (r1 <> r2)):

```

```

wee[r1][r2] = 0.001;
ind = 1;

for j in range(0, NN_E): # (RE) NORMALISATION
    if (wee[r1][j] > 0):
        sum = sum + wee[r1][j];
for j in range(0, NN_E):
    if ((sum <> 0) and (wee[r1][j] >= 0)):
        wee[r1][j] = wee[r1][j] / sum;

# ===== VISUALIZING RESULTS ===== #
def printImg(t):
    t1 = t % 2;
    for i in range(0, NN_E):
        if (x[t1][i] <> 0):
            img[i][t] = 0;
        else:
            img[i][t] = 255;
    for i in range(0, NN_I):
        if (y[t1][i] <> 0):
            img[i+NN_E][t] = 0;
        else:
            img[i+NN_E][t] = 255;

def clearStringPattern():
    for i in range(0, N):
        stringPattern[i] = 0;

def updateStringPattern(t):
    t1 = t % 2;
    for i in range(0, N):
        if (x[t1][i+100-N] == 1):
            stringPattern[i] = stringPattern[i] + 1;

def calculateStringScore(t):
    for i in range(0, N):
        numberSprike[i][t] = stringPattern[i] / 50;

```

```
if (stringPattern[i] >= thresholdOn):
    stringPattern[i] = 1;
else:
    stringPattern[i] = 0;
```

```
def savePara(pair):
    if pair == 0:
        for i in range(0, NN_E):
            for j in range(0, NN_E):
                tempwee1[i][j] = wee[i][j];

        for i in range(0, NN_E):
            for j in range(0, NN_I):
                tempwei1[i][j] = wei[i][j];

        for i in range(0, NN_I):
            for j in range(0, NN_E):
                tempwie1[i][j] = wie[i][j];

        for i in range(0, NN_E):
            tempte1[i] = te[i];

        for i in range(0, NN_I):
            tempti1[i] = ti[i];

    elif pair == 1:
        for i in range(0, NN_E):
            for j in range(0, NN_E):
                tempwee2[i][j] = wee[i][j];

        for i in range(0, NN_E):
            for j in range(0, NN_I):
                tempwei2[i][j] = wei[i][j];

        for i in range(0, NN_I):
            for j in range(0, NN_E):
                tempwie2[i][j] = wie[i][j];
```

```

    for i in range(0, NN_E):
        tempte2[i] = te[i];

    for i in range(0, NN_I):
        tempti2[i] = ti[i];
elif pair == 2:
    for i in range(0, NN_E):
        for j in range(0, NN_E):
            tempwee3[i][j] = wee[i][j];

    for i in range(0, NN_E):
        for j in range(0, NN_I):
            tempwei3[i][j] = wei[i][j];

    for i in range(0, NN_I):
        for j in range(0, NN_E):
            tempwie3[i][j] = wie[i][j];

    for i in range(0, NN_E):
        tempte3[i] = te[i];

    for i in range(0, NN_I):
        tempti3[i] = ti[i];

def clearActivity():
    for i in range(0, 2):
        for j in range(0, NN_E):
            x[i][j] = 0;

    for i in range(0, 2):
        for j in range(0, NN_I):
            y[i][j] = 0;

def loadPara(pair):
    if pair == 0:
        for i in range(0, NN_E):
            for j in range(0, NN_E):
                wee[i][j] = tempwee1[i][j];

```

```

for i in range(0, NN_E):
    for j in range(0, NN_I):
        wei[i][j] = tempwei1[i][j];

for i in range(0, NN_I):
    for j in range(0, NN_E):
        wie[i][j] = tempwie1[i][j];

for i in range(0, NN_E):
    te[i] = tempte1[i];

for i in range(0, NN_I):
    ti[i] = tempti1[i];

elif pair == 1:
    for i in range(0, NN_E):
        for j in range(0, NN_E):
            wee[i][j] = tempwee2[i][j];

    for i in range(0, NN_E):
        for j in range(0, NN_I):
            wei[i][j] = tempwei2[i][j];

    for i in range(0, NN_I):
        for j in range(0, NN_E):
            wie[i][j] = tempwie2[i][j];

    for i in range(0, NN_E):
        te[i] = tempte2[i];

    for i in range(0, NN_I):
        ti[i] = tempti2[i];
elif pair == 2:
    for i in range(0, NN_E):
        for j in range(0, NN_E):
            wee[i][j] = tempwee3[i][j];

```

```
for i in range(0, NN_E):
    for j in range(0, NN_I):
        wei[i][j] = tempwei3[i][j];
```

```
for i in range(0, NN_I):
    for j in range(0, NN_E):
        wie[i][j] = tempwie3[i][j];
```

```
for i in range(0, NN_E):
    te[i] = tempte3[i];
```

```
for i in range(0, NN_I):
    ti[i] = tempti3[i];
```

```
def performMutation():
```

```
    # ===== MUTATING WEIGHT OF E-E CONNECTIONS ===== #
```

```
    for i in range(0, NN_E):
```

```
        for j in range(0, NN_E):
```

```
            if (wee[i][j] <> -99 and np.random.random() < pMut):
```

```
                wee[i][j] = wee[i][j] + np.random.normal(0, etaMut);
```

```
                if wee[i][j] < 0.0:
```

```
                    wee[i][j] = 0.0;
```

```
                if wee[i][j] > 1.0:
```

```
                    wee[i][j] = 1.0;
```

```
    # ===== RENORMALIZATION OF THE E-E CONNECTIONS ===== #
```

```
    for i in range(0, NN_E):
```

```
        sum = 0;
```

```
        for j in range(0, NN_E):
```

```
            if (wee[i][j] <> -99):
```

```
                sum = sum + wee[i][j];
```

```
        for j in range(0, NN_E):
```

```
            if (wee[i][j] <> -99 and sum <> 0):
```

```
                wee[i][j] = wee[i][j] / sum
```

```
    # ===== #
```

```

for i in range(0, NN_E):
    for j in range(0, NN_I):
        if (wei[i][j] <> -99 and np.random.random() < pMut):
            wei[i][j] = wei[i][j] + np.random.normal(0, etaMut);
        if wei[i][j] < 0.0:
            wei[i][j] = 0
        if wei[i][j] > 1.0:
            wei[i][j] = 1.0

for i in range(0, NN_E):
    sum = 0;
    for j in range(0, NN_I):
        if (wei[i][j] <> -99):
            sum = sum + wei[i][j];
    for j in range(0, NN_I):
        if (wei[i][j] <> -99 and sum <> 0):
            wei[i][j] = wei[i][j] / sum

# ===== #

for i in range(0, NN_I):
    for j in range(0, NN_E):
        if (wie[i][j] <> -99 and np.random.random() < pMut):
            wie[i][j] = wie[i][j] + np.random.normal(0, etaMut);
        if wie[i][j] < 0:
            wie[i][j] = 0;
        if wie[i][j] > 1.0:
            wie[i][j] = 1.0;

for i in range(0, NN_I):
    sum = 0;
    for j in range(0, NN_E):
        if (wie[i][j] <> -99):
            sum = sum + wie[i][j];
    for j in range(0, NN_E):
        if (wie[i][j] <> -99 and sum <> 0):
            wie[i][j] = wie[i][j] / sum

```

```

# ===== MUTATING THE THRESHOLD ===== #

for i in range(0, NN_E):
    if (np.random.random() < pMut):
        te[i] = te[i] + np.random.normal(0, etaMut);
        if te[i] < 0:
            te[i] = 0;
        if te[i] > TE_MAX:
            te[i] = TE_MAX;

for i in range(0, NN_I):
    if (np.random.random() < pMut):
        ti[i] = ti[i] + np.random.normal(0, etaMut);

        if ti[i] < 0:
            ti[i] = 0;
        if ti[i] > TI_MAX:
            ti[i] = TI_MAX;

# ===== MAIN PROGRAM STARTS HERE ===== #

aux01.seed(500);
init();

pair = 0;
st = 0;
savePara(0);
tmp = 0;
# ===== ALGORITHM START ===== #

for time in range(1, STEPS):

    for i in range(0, 100):
        inp[i] = 0;

    inp[st%10] = strength;

```

```

if (st == 0 and pair == 0):
    clearActivity();
    savePara(2);

if (st == 0 and pair == 1):
    performMutation();
    clearActivity();

step(time);      # RUNNING SORN STEPS

if (st >= 0):
    updateStringPattern(time);

if time % duration == 0:
    st = st + 1;

    if st == stateIteration:
        st = 0;
        calculateStringScore(tmp);
        tmp = tmp + 1;
        #score[pair] = sum(stringPattern);
        #scoreFig.append(sum(stringPattern));
        score[pair] = sum(np.array(stringPattern,"b")^MASK);
        scoreFig.append(sum(np.array(stringPattern,"b")^MASK));
        clearStringPattern();

        savePara(pair);
        pair = pair + 1;

        if (pair == 1):
            loadPara(2);

        if (pair == 2):
            pair = 0;
            if (score[0] >= score[1] and score[0] >= score[2]):
                loadPara(0);

```

```

        score[2] = score[0]
    elif (score[1] >= score[0] and score[1] >= score[2]):
        loadPara(1);
        score[2] = score[1]
    else:
        loadPara(2);

    if (VISUAL and time >= STEPS - SAVE_STEPS):    # VISUAL THE RESULT ON THE
DIAGRAM
        printImg(time - STEPS + SAVE_STEPS);

plt.figure(1);
plt.imshow(img, aspect="auto");
plt.gray();

plt.figure(2);
plt.plot(scoreFig)

plt.figure(3);
plt.imshow(numberSpike, cmap="jet");
plt.show();
# ===== END ===== #

```