Prediction metrics for Liquid State Machine and Klinokinesis on Intel's Loihi

DDP Stage-II Report

submitted in partial fulfillment
of the requirements for the degree of

Bachelor & Master of Technology

by

Apoorv Kishore
Roll no. : 16D070018

under the guidance of

Prof. Udayan Ganguly

Department of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
2021

Contents

Abstract **3**

Acknowledgements **4**

1 Liquid State Machines **5**
1.1 Introduction 5
1.2 Structure and Properties of the LSM 5
1.2.1 Modelling a Biological Neuron 5
1.2.2 Leaky Integrate and Fire Neuron 6
1.2.3 LSM architecture 7
1.2.4 Separation and Approximation 8

2 Performance Prediction Metrics **9**
2.1 Introduction 9
2.2 Memory Metric 9
2.3 Lyapunov Exponent 10
2.3.1 Edge of Chaos 10
2.3.2 Lyapunov Exponent: expression of separation ability and chaos 11
2.4 Comparison : Memory metric and Lyapunov Exponent 11
2.5 A detailed study of Prediction metrics : Kernel Quality 13

3 Memory Metric and Self Organized Criticality **15**
3.1 Introduction 15
3.2 Replication of Gorad et al. 15
3.3 Averaging Window Dependence 18
3.4 Comparison among metrics 20
3.4.1 Kernel Quality calculation 20
3.4.2 Results 21
3.5 Self Organised Criticality 22

4 Loihi: Analysis and Application **24**
4.1 Intel's Loihi for SNNs 24
4.2 Comparison Between MATLAB and Loihi Simulations 24
4.3 Comparing Loihi and MATLAB : Accuracy, τ_M and μ 27
4.4 C-elegans: Klinotaxis and Klinokinesis 27
4.5 One-dimensional Navigation 28
4.5.1 Circuit Diagram 28
4.5.2 Concentration Encoding 29
4.5.3 Subtraction 30
4.5.4 Implementation 31

5 Simplified Klinokinesis for Resource-Constrained Navigation on Loihi 33
5.1 Introduction . . . 33
5.2 Network Architecture . . . 33
5.2.1 Single concentration sensor . . . 33
5.2.2 Neuron Model . . . 34
5.2.3 Working Principle . . . 34
5.2.4 Gradient Detection . . . 36
5.2.5 Navigation . . . 37
5.2.6 Network Response . . . 38
5.3 Simulations and Results . . . 39

6 Future Prospects 44

Bibliography 44

Abstract

Spiking Neural Networks (SNN) which are the 3rd generation of Neural Networks (NN) are more closer to brain than any other generation of NN. SNN are highly power efficient as they operate only using spikes. However, till now, they neither have the mathematical backing nor the results that are offered by 2nd generation of NNs, the Artificial Neural Networks which have completely changed the field of Artificial Intelligence

Liquid State Machines (LSM) which come under broader field of Reservoir Computing are also particular kind of SNN. They provide a very general computing framework along with a biological motivation as they are partially based on visual and auditory cortex of brain. LSM have found an application in speech recognition, video classification, reinforcement learning, biomimetics, etc. with achieving near state of art results in speech recognition task. Even With advantages like no tuning of weights in reservoir, parallel computation, no specific modelling for any given task, etc., they have not been explored significantly due to hard task of designing reservoirs and time taken for training on a general CPU or GPU even for a moderate sized reservoir (few 100 neurons). We aim to develop rigid and accurate performance prediction metrics for LSM and do a detailed comparative analysis on the metrics available today. We also study the variation of the metric over variation of different model parameters.

Loihi is a neuromorphic research chip designed by Intel Labs that uses an asynchronous spiking neural network (SNN) to implement adaptive self-modifying event-driven parallel computations used to implement learning and inference. A single Loihi chips contains 0.13 Million neurons across 128 cores giving a significant advantage for implementing large size LSMs. We aim to utilize this power for implementing such LSMs and performing a comparative analysis between the implementations of the LSM in MATLAB and in Loihi. We also study how the performance of the LSM changes with the small changes in neuronal spike timings.

Acknowledgements

I would like to express my sincere gratitude towards **Prof. Udayan Ganguly** for his constant guidance, motivation and inputs. I would like to thank my mentor **Rajat Patel** for his constant support in the project. I would also like to thank **Vivek Saraswat** for his valuable ideas and inputs throughout the project.

I would like to thank **Ajinkya Gorad** and **Shashwat Shukla** for providing the underlying simulation material required for the project. I would also like to thank my colleagues working in our group for discussing the various possibilities and ideas and thus helping me with the difficulties.

Apoorv Kishore
IIT Bombay
June, 2021

Chapter 1

1 Liquid State Machines

1.1 Introduction

Like the Turing machine, the model of a liquid state machine (LSM) is based on a rigorous mathematical framework that guarantees, under idealized conditions, universal computational power. In contrast to Turing machines, which have universal computational power for off-line computation on (static) discrete inputs, LSMs have in a very specific sense universal computational power for real-time computing with fading memory on analog functions in continuous time.

This real-time universal computation power is what makes LSMs a biologically plausible model of the cerebral cortex wherein neural microcircuits perform diverse real-time information processing tasks. Furthermore biological data prove that cortical microcircuits can support several real-time computational tasks in parallel, which is not possible in most modelling approaches.The highly diverse and complex dynamics on several temporal scales observed makes them very different than computational models used in computer science or artificial intelligence which provides motivation to develop such a biologically realistic model.

In this chapter, we briefly discuss the structure of Liquid State Machines, universal computational power for real time computing on continuous input streams, effects of synaptic connectivity on performance, learning rules and the applications of LSM.

1.2 Structure and Properties of the LSM

The structure of the LSM is divided into three sub-parts which are the neural reservoir, Input network and the classifier. To discuss each of them we must first understand how a biological neuron is modelled as a spiking neuron in computer simulations.

1.2.1 Modelling a Biological Neuron

A typical biological neuron can be divided into three functionally distinct parts, called dendrites, soma, and axon; see figure 1.1. The dendrites are like the 'input device' that collect signals from other neurons and then transmit them to the soma. The soma is more or less like the 'central processing unit' that performs a crucial non-linear processing step: If the total input arriving at the soma exceeds a certain threshold, then an output signal is generated. The output signal is taken over by the 'output device,' the axon, which delivers the message to other neurons. Thus a chain is formed, and the signal is passed on.

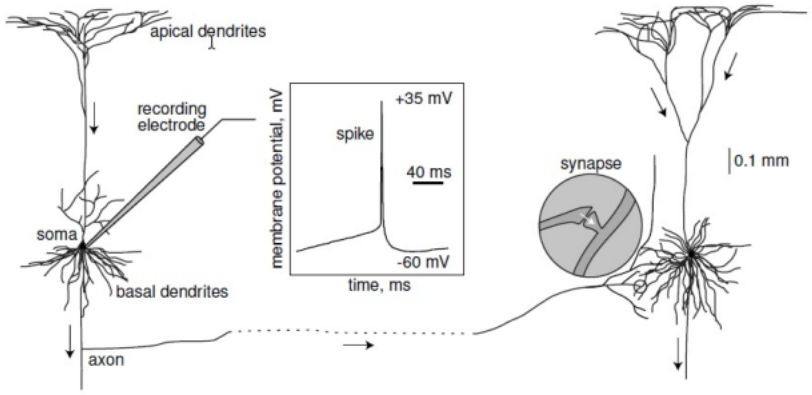

Figure 1: Biological Neuron and its signal transmission

These biological Neurons carry information through the transmission of these spikes in the membrane potential. The spikes in themselves are not the source of information, rather, their time of arrival and number of spikes constitute the actual information. A neuron after spiking the first time cant spike till certain amount of time. This duration is called refractory period of neuron.

The site where the axon of a one neuron makes contact with the dendrite of another neuron is the synapse. When a spike arrives at a synapse, it triggers a complex chain of biochemical processing steps that lead to a release of neurotransmitter from the presynaptic terminal. As soon as transmitter molecules have reached the postsynaptic side, they will be detected by specialized receptors in the postsynaptic cell membrane and lead to an opening of specific ion channels. In all, it changes the membrane potential at the postsynaptic site, with the chemical signal is translated into an electrical response. The voltage response of the postsynaptic neuron to a presynaptic spike is called the postsynaptic potential. This biological neuron varies highly in terms of properties exhibited across species, across areas of a brain of a one particular species and even in a one micro-cortical circuit. Various mathematical models have been proposed for this biological neuron which at best models only some properties of neuron at best. All these models are different from the very simple models used in ANNs. Integrate and Fire, Leaky Integrate and Fire, Hodgkin Huxley model, Exponential integrate-and-fire, Morris Lecar model are some of widely known models.

1.2.2 Leaky Integrate and Fire Neuron

Leaky Integrate and Fire models are simplest of the models described in previous subsection. They are also the most widely used models in SNNs implemented at software or hardware level. The dynamics of membrane potential V(t) in LIF neuron is described by following equation

$$\tau \frac{dV}{dt} = -(V(t) - V_{rest}) + R.I(t) \tag{1}$$

where I(t) is the current generated as postsynaptic response from input spikes, τ is the membrane time constant and R is the membrane resistance. The resting potential V_{rest} is generally set as per the values obtained from the biological neuron which is around -65mV (the value used for our simulations is $V_{rest} = 0$. The neuron is said to have spiked when the membrane potential rises beyond a threshold potential V_{th} which according to the biological values is around 20mV. The synaptic current which enters a particular neuron is calculated through the following equation

$$I(t) = \sum_{i=1}^{N} w_i H(t, i) \tag{2}$$

where, w_i is synaptic weight of connection from preneuron i to current neuron and $H(t, i)$ is postsynaptic response to spikes from preneuron i. $H(t, i)$ is depends on synaptic delay d_i, pre neuron spike times t_i and time constant τ. Various forms of $H(t, i)$ have been used in literature and we will specify where ever applicable.

1.2.3 LSM architecture

The Liquid State machine is divided into three major parts which is the Input network, liquid reservoir and classifier. The Liquid or the reservoir is a recurrent spiking neural network in which all the neurons are connected randomly and each of these connections are fixed for a particular LSM irrespective of the task performed which is similar to the biological neural networks.

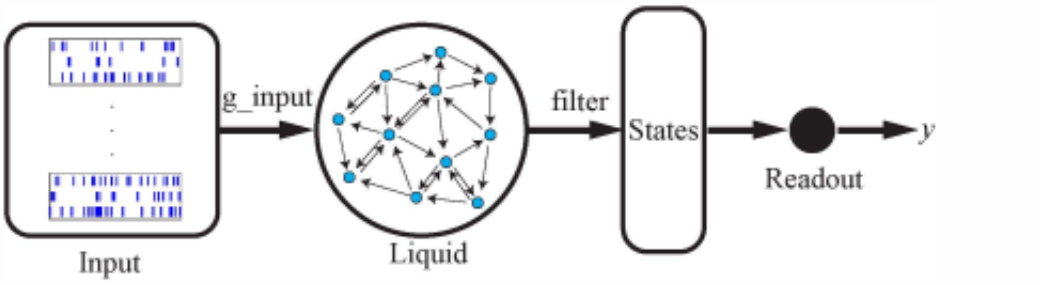

Figure 2: Liquid State Machine

The input layer of neurons are connected to some or all of the reservoir neurons and these input connections are also fixed for a particular LSM. The input layer of neurons transmit the input spike signals into the Neuron Liquid.

From the reservoir Liquid, the final states are calculated which generally are the spike pattern over time of each reservoir Neuron. These final states are then used for the classification task which is performed by one or many readout neurons. These neurons are connected to all reservoir neurons through some plastic weights and during classification they train their weights to perform the task optimally

1.2.4 Separation and Approximation

LSMs possess two properties; separation and approximation property; by virtue of which they have Universal Computational Power and are good biologically inspired models for mammalian neocortex as described in Maass et. al [1]. The reservoir can be seen as filter $L^M(t)$ which maps input $u(t$ to reservoir state $x^M(t)$ and the readout synapses as $f^M(t)$ projecting $x^M(t)$ to output $y(t)$. As long as the the class of filters from which L^M is drawn satisfies point-wise separation property and f^M satisfy the approximation property, the complete model can approximate any time invariant function with fading memory to any arbitrary precision.

The separation property deals with the ability of the reservoir to separate two signals which are very close to each other. The approximation property deals with the ability of the readout to approximate any function.

Chapter 2

2 Performance Prediction Metrics

2.1 Introduction

Given a reservoir, what will be a good metric that can predict the performance of a network with high confidence? We briefly discuss two such ideas here: memory metric through state space approximation [5] and Lyapunov Exponent [4]. There exists more ways like using STDP, genetic evolution, etc. for designing LSMs but we will not focus on those methods.

2.2 Memory Metric

The Memory Metric or more commonly known as τ_M is a prediction metric which was first introduced in Gorad et al. This metric is derived from the State Space approximation of the Liquid State Machine. The LSM is a highly non-linear system and its functionality can be given by the following equations

$$X^{k+1} = f(X^k, U^k) \quad and \quad R_0^k = f_w(X^k) \tag{3}$$

where X^k is the spike rate of the Reservoir at time k calculated by averaging the spikes over a small averaging window, U^k is the input at time k and R_o^k is the output at time k. This system is linearly approximated by the state space representation which is calculated using the Moore-Penrose inverse. The state space representation is given by the following set of equations.

$$X^{k+1} = AX^k + BU^k \quad and \quad R_0^k = WX^k \tag{4}$$

From the above state space equation the τ_M is calculated using the following set of equations.

$$\tau_M = \frac{1}{N}\sum_{i=1}^{N}\frac{h}{1-|a_i|} \quad where \quad a = diag(A) \tag{5}$$

τ_M is the embodiment of the ability of the reservoir to remember its previous spikes. Mathematically speaking, it is a time constant of the dependency of the new spike rates on the old spike rates which if large, means that this dependency takes long time to decay and hence the memory of the system is more.

The state space or linear approximation of a highly non-linear LSM gives us accurate results due to the observations from the plot shown in Fig.4. It can be noticed that the

correlation between the states calculated using state space and the original states have high correlation in the low error region. Hence in the high accuracy region, The State Space approximation is accurate for the LSM.

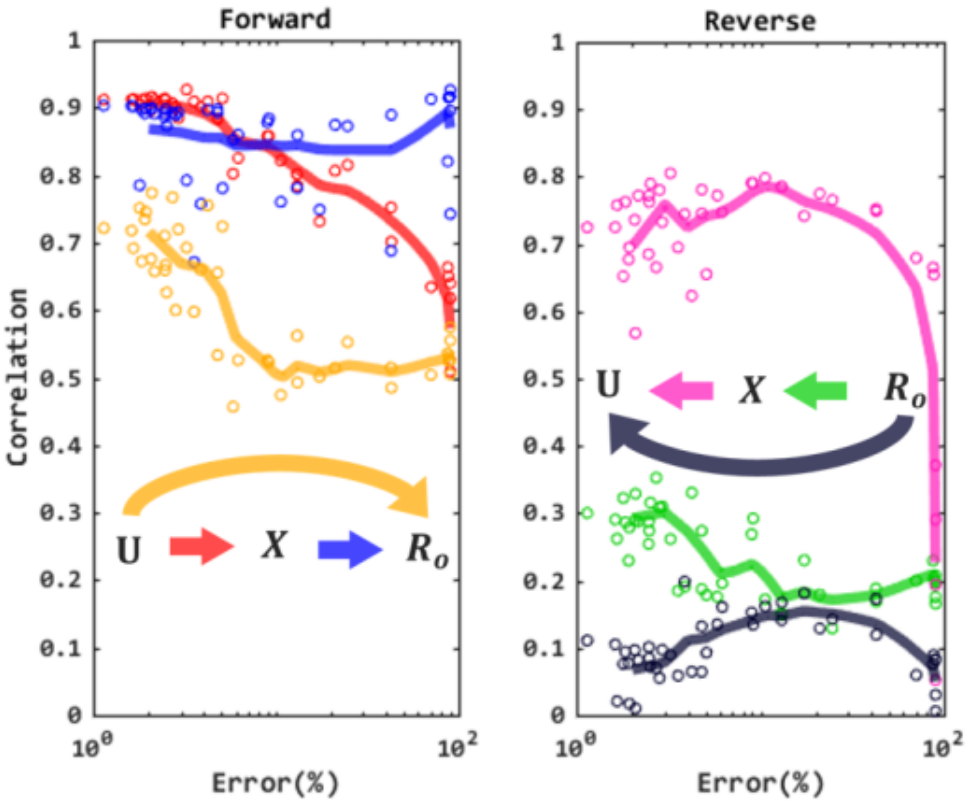

Figure 3: Average Correlation between LSM model and its state space approximation on TI46 dataset vs the error rate

2.3 Lyapunov Exponent

To introduce Lyapunov Exponent or μ we must first understand what is the meaning of a chaotic system and what defines the edge of chaos

2.3.1 Edge of Chaos

A recurrent neural circuit is a special case of a dynamical system. By changing some global parameters of the system, e.g. connectivity structure or the functional dependence of the output of an element on the output of other elements, one can change the dynamics of the system from ordered (low to medium spike rates in LSM reservoir with some history being remembered) to chaotic (too many neurons spiking resulting in no valuable information being stored or generated). The transition boundary is called

edge of chaos. Studies have suggested that systems performance are best when they are operating at edge of chaos

2.3.2 Lyapunov Exponent: expression of separation ability and chaos

According to the standard defintion used in [4], we look at μ or the lyapunov exponent through formula

$$\delta_{\Delta T} = \delta_0 e^{\mu \Delta T} \tag{6}$$

where δ_0 is the small difference or disturbance introduced in the state of the reservoir at time t_0 and $\delta_{\Delta T}$ is the resulting state difference at time $t_0 + \Delta T$. From the equations we can gather that when the system is chaotic μ will be positive and if the system is ordered μ will be negative, hence the edge of chaos is defined by the line $\mu = 0$. As discussed in the previous section, the performance of the LSM is optimal around this region. This result can be seen from Fig.5. It is also important to note that around the edge of chaos, μ is monotonic whereas the performance has a peak. μ is hence said to be an expression of both the chaos and the separation ability of the reservoir.

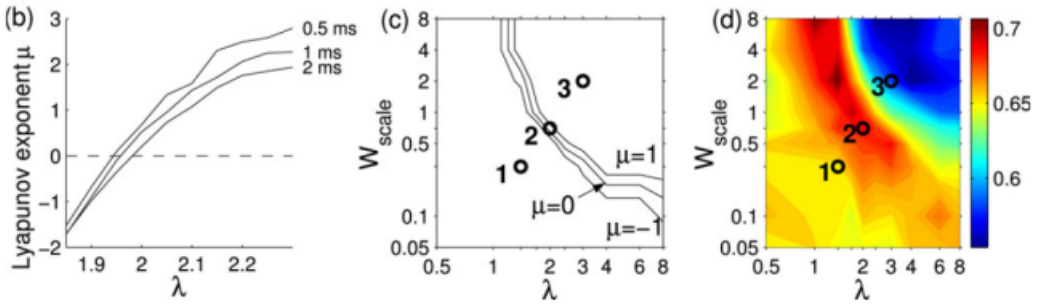

Figure 4: Lyapunov exponent variation over parameters and relation with performance

2.4 Comparison : Memory metric and Lyapunov Exponent

For designing a reservoir in LSM both lyapunov exponent and memory metric provide an easier setup. [5] compares both of them on TI46 speech dataset. Some of the results from their work are shown in fig6. We can observe that at low error rates τ_M increases almost linearly. Moreover, PCC values show that low error rates are more correlated with τ_M than μ. Also, for both higher and lower values for μ the error rate will increase which was also observed in previous sub section, while τ_M increment corresponds to higher accuracy. With increase in no. of epochs a clear reduction in error rate is observed along with increase in τ_M which adds to last point

This behaviour in μ is observed because as we discussed previously, near the edge of chaos which is the low error regime, the Lyapunov Exponent is a monotone function whereas the performance has a peak or the error has a dip hence the curved plot. This tells us that Lyapunov Exponent is not a good predictor at low error regime.

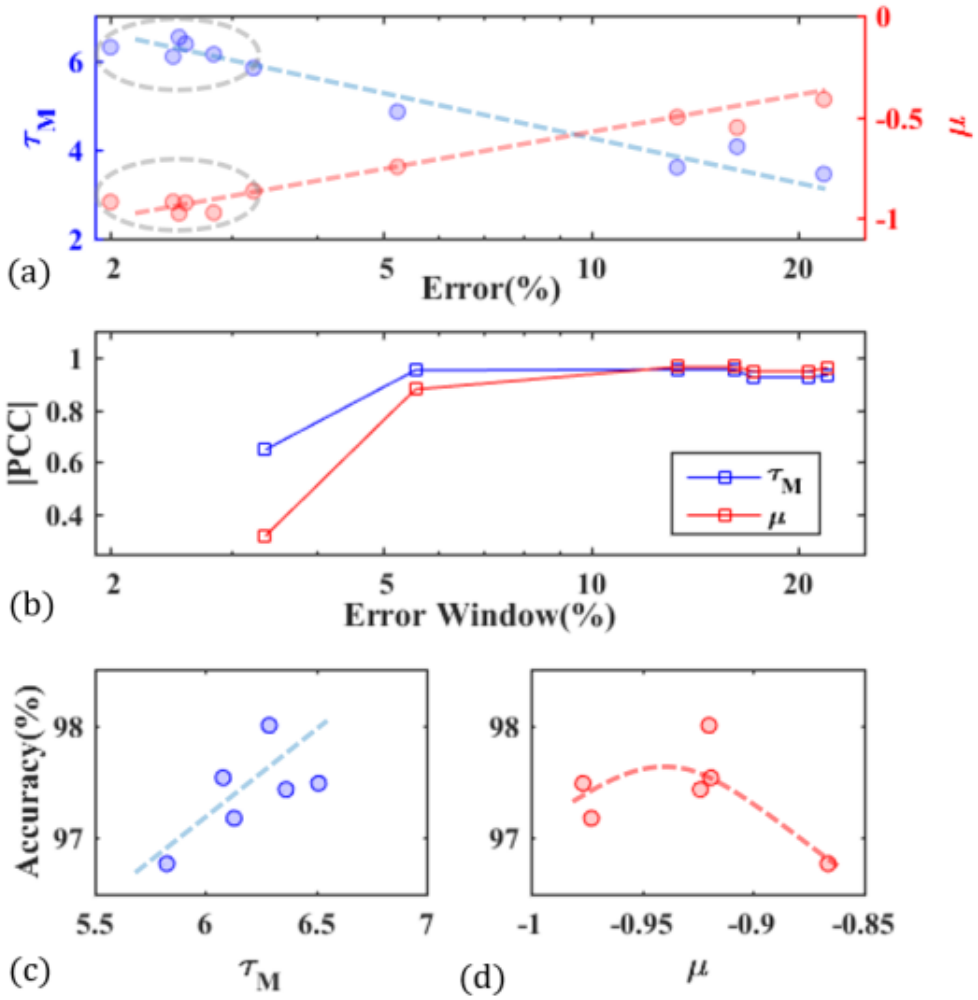

Figure 5: Comparison of μ and τ_M. PCC is pearson correlation coefficient calculated between accuracy and metric

2.5 A detailed study of Prediction metrics : Kernel Quality

In this section, we discuss about the results shown in Legenstein et al. [**?**]. This paper performs n-bit parity calculation task on real-time data and then tries to compare various prediction metrics on the basis of how they good they are at predicting the performance. From this paper and the previous discussion about Lyapunov Exponent we gather

- Lyapunov Exponent is a representation of the separation ability of the reservoir
- It is not a good predictor at low error regime
- It does predict the performance acceptably at relatively higher error regimes

These factors lead us to believe that at higher error regimes, the contribution of the generalization ability is very less and separation ability is what defines the performance however as we move towards lower error regimes, the contribution of generalization starts to increase and Hence only separation is not enough to do a good prediction. Hence a metric that captures both separation and generalization ability of the reservoir is needed.

The paper defines another Prediction metric called the Kernel Quality which is given by the separation rank - generalization rank; calculated from the final state matrix.

$$F = [S(1)^T \;\; S(2)^T \;\ldots\; S(N)^T]^T \tag{7}$$

where $S(m)$ is the final state vector resulting from the input m. The rank of the final state matrix for very different N inputs gives us the separation rank whereas the rank for very close N inputs gives us the generalization rank. The Metric Kernel Quality has been shown to predict the performance acceptably for low error regimes as shown in the following Figures.

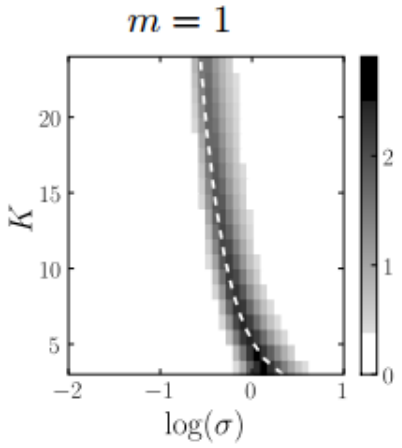

Figure 6: Actual Performance of the LSM

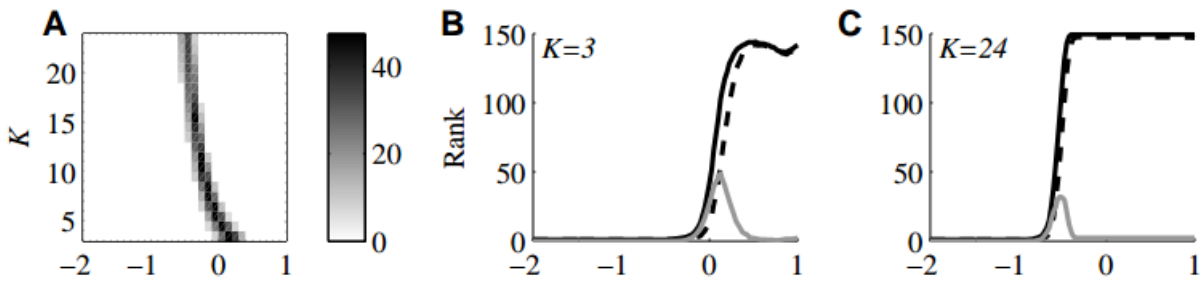

Figure 7: Prediction of Performance by Kernel Quality

Chapter 3

3 Memory Metric and Self Organized Criticality

3.1 Introduction

We tried to study many questions that are raised from our previous analysis such as why a good Correlation Coefficient is produced for high error values for τ_M when we have seen that Linear State Space representation is not accurate at high error values? What are the implications of the form of State Matrix A, the averaging window for state calculation on τ_M? Is there any underlying mathematical relation between memory metric and Lyapunov Exponent that can be found using a state space representation of μ. Since Kernel Quality is also a good predictor of performance as seen from previous studies shown in the last section, a natural question that arises is that which metric performs the best among these. To dwell into these questions we first reproduce the results presented in Gorad et al. [5]

3.2 Replication of Gorad et al.

We use the same Network and Neuron Parameters as used in Gorad et al. to compute performance of a 125 Neuron reservoir on TI46 dataset and compare memory metric and Lyapunov Exponent on their predictive power. We compute performance and the metrics averaged over 3 different reservoirs and computed for 17 different weight scaling values. The results we obtained show similar trends in accuracy with the weight scaling as observed in [5]. We see that accuracy decreases as weights are increased and so does the memory metric. We were also able to replicate the linear behaviour of τ_M in low error regime and the curved μ vs accuracy plot for low error. Similar to [5] we were able to show that PCC of τ_M for low error regime is almost twice that of μ

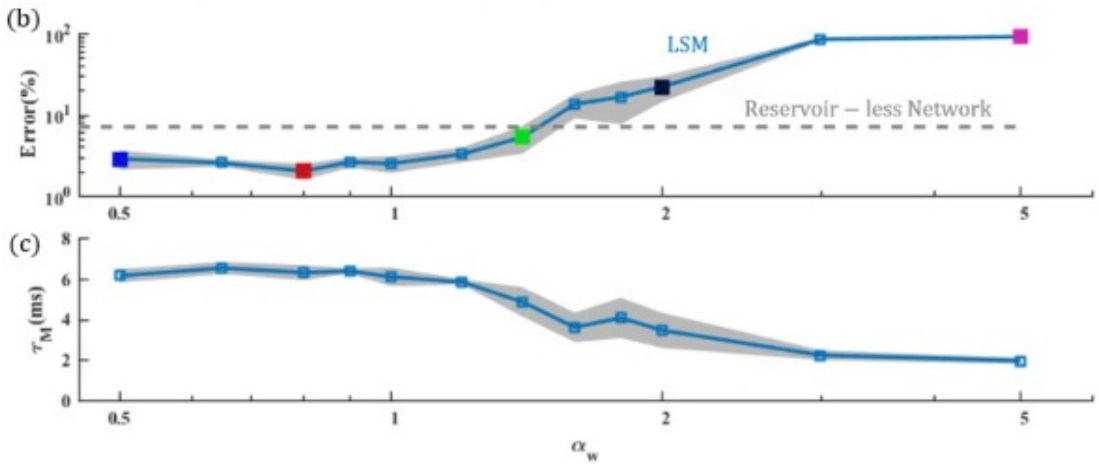

Figure 8: Error and τ_M vs weight scaling : Gorad et. al

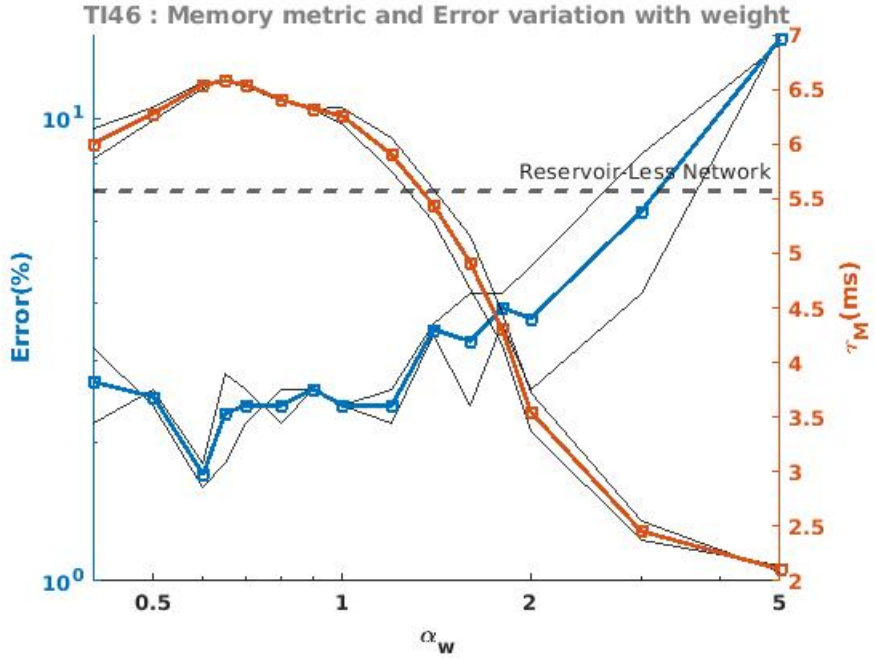

Figure 9: Error and τ_M vs weight scaling : Present Simulation

	τ_M	μ
Gorad et.al	64%	31%
Present Simulation	77.49%	36.86%

Table 1: PCC value comparison at low error values

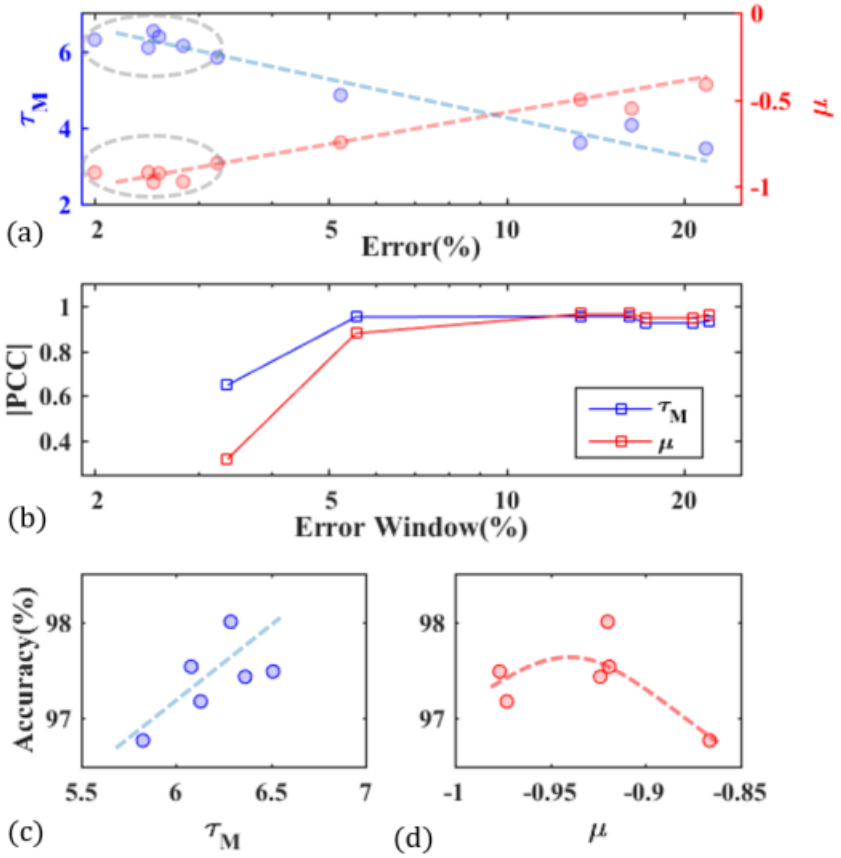

Figure 10: Correlation of τ_M and μ with accuracy : Gorad et. al

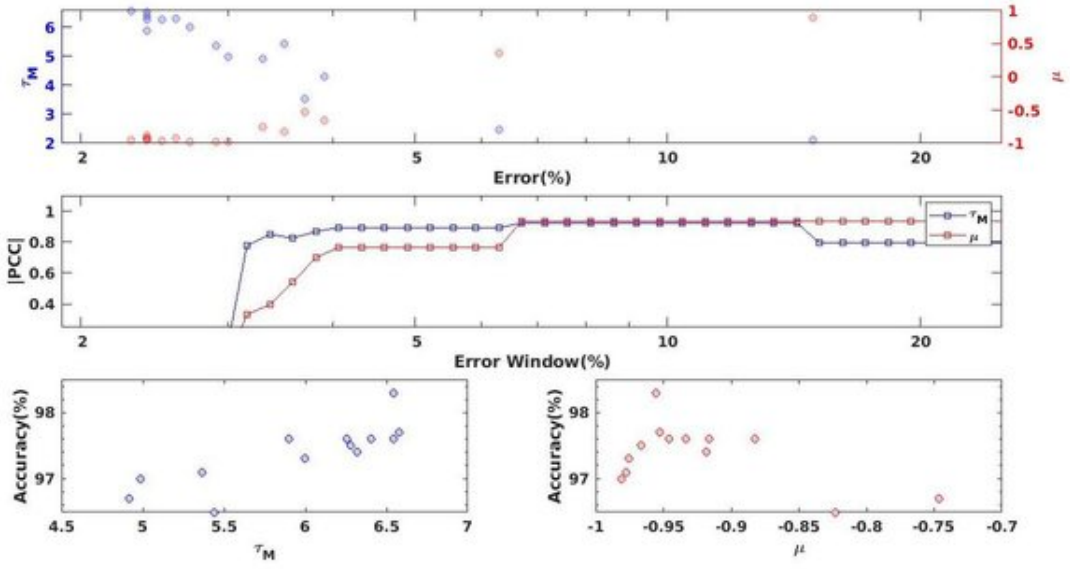

Figure 11: Correlation of τ_M and μ with accuracy : Present Simulation

3.3 Averaging Window Dependence

One of the aspects of memory metric that we wanted to study was how the averaging window size that is used to calculate the spike rate X of the reservoir, affects the memory metric. It was expected that as the window size increases the PCC of τ_M would increase because on increasing of the window, the spike rate X holds more information about the reservoir spikes at each time instant and thus the state space becomes somewhat more accurate of a representation for the LSM in low error regimes, which then leads to a good prediction by τ_M. This leads to the expectation of the increase in PCC with increasing window.

As seen in the observation table below, such a trend is indeed observed however the increase in PCC is not by drastic amount. We also observe that for very low value of the averaging window size, the PCC of τ_M becomes very less and its profile also does not hold any significant information, this might be the case because when window is size is 1ms, the state X represents the instantaneous spikes of the reservoir. We know that instantaneous spikes are highly non-linearly dependent on each other which is why the state space representation is not at all accurate and hence τ_M also does not hold any useful information about the reservoir. We also observe that the profile of τ_M is rigid for very high values of averaging window also.

Averaging window	tauM	LE
1ms	32.25%	36.86%
2ms	76.22%	36.86%
4ms	77.15%	36.86%
5ms	77.45%	36.86%
20ms	77.48%	36.86%
50ms	77.49%	36.86%
100ms	77.37%	36.86%
500ms	77.54%	36.86%
1000ms	77.87%	36.86%
2000ms	77.54%	36.86%
5000ms	77.30%	36.86%
10000ms	77.89%	36.86%
11996ms	77.69%	36.86%

Table 2: PCC of metrics with varying averaging window

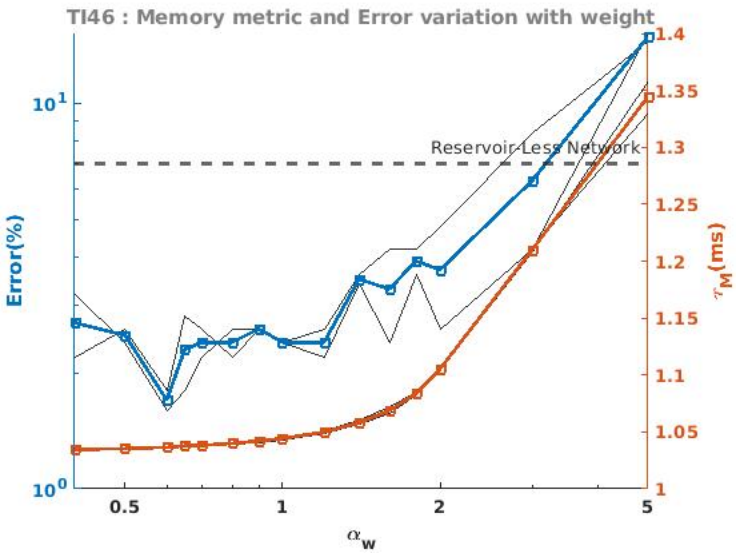

Figure 12: Error and τ_M vs weight scaling for window size = 1ms

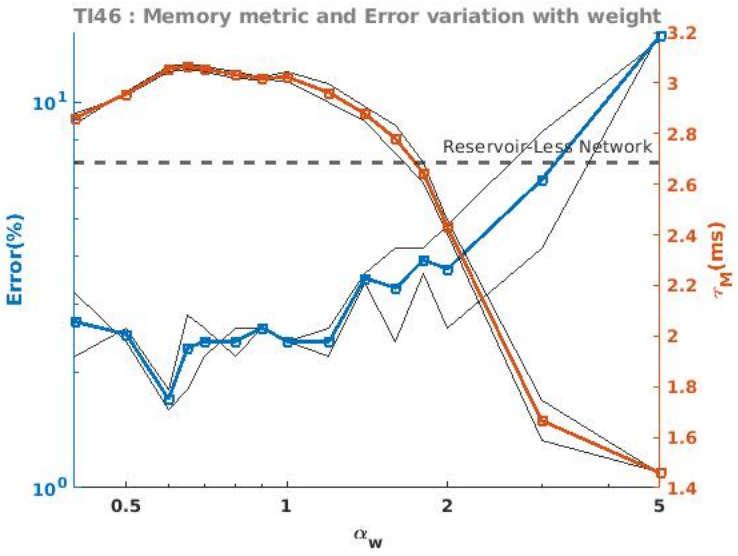

Figure 13: Error and τ_M vs weight scaling for window size = 4ms

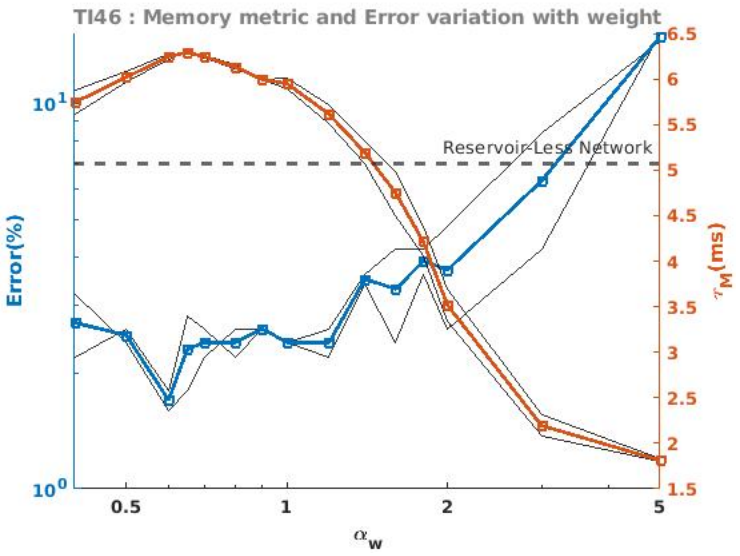

Figure 14: Error and τ_M vs weight scaling for window size = 5000ms

3.4 Comparison among metrics

The paper Gorad et al. [5] compares the performance of memory metric and Lyapunov Exponent. In previous sections it has been shown that Lyapunov Exponent captures only the separation ability of the system which is why it might not be the best prediction metric present in current literature. For this reason, Kernel Quality holds importance in our study, as it captures both separation effect and the generalisation effect of the system both of which are essential deciding factors for neural network performance. Hence we compare the correlation of each of these metrics with Liquid State Machine classification performance on TI46-speech dataset.

3.4.1 Kernel Quality calculation

The calculation of Kernel Quality is performed using the concept of separation rank and generalisation rank. These ranks represent the separation and generalisation ability of the system respectively. Let $S(m)$ be the vector defining the state of the network for stimulus m, then separation and generalisation ranks are calculated from the Final State Matrix given by

$$F = \begin{bmatrix} S(1)^T \\ S(2)^T \\ .. \\ S(N)^T \end{bmatrix} \tag{8}$$

for total N stimuli given to the network as inputs. Here each vector $S(m)$ is given by

$$S(m) = \begin{bmatrix} s_m^f(1) \\ s_m^f(2) \\ .. \\ s_m^f(n) \end{bmatrix} \tag{9}$$

where n is the total neurons in the reservoir. $s_m^f(i)$ is the final state of neuron i corresponding to input stimulus m. It is given by

$$s_m^f(i) = \sum_k exp(-\frac{t_{sim} - t_k}{\tau}) \tag{10}$$

where t_{sim} is the total time duration and t_k is the time of the k_{th} spike of i_{th} neuron. τ is some time constant. The rank of matrix F gives the separation rank when the inputs are very close in nature (it speaks to the separating power of very close signals), and gives the generalisation rank when the inputs are further apart (it speaks to the generalising power of far apart inputs into one group). The Kernel Quality of the network is then given by

$$Kernel\ \ Quality = Sep\ \ rank - Gen\ \ rank \tag{11}$$

3.4.2 Results

The Pearson Correlation Coefficients of memory metric, Lyapunv exponent and Kernel Quality are determined for TI46-speech classification task. The results obtained clearly display that memory metric has a higher correlation in the low error regime as compared to Kernel Quality or Lyapunov Exponent, which establishes the consistency and importance of memory metric.

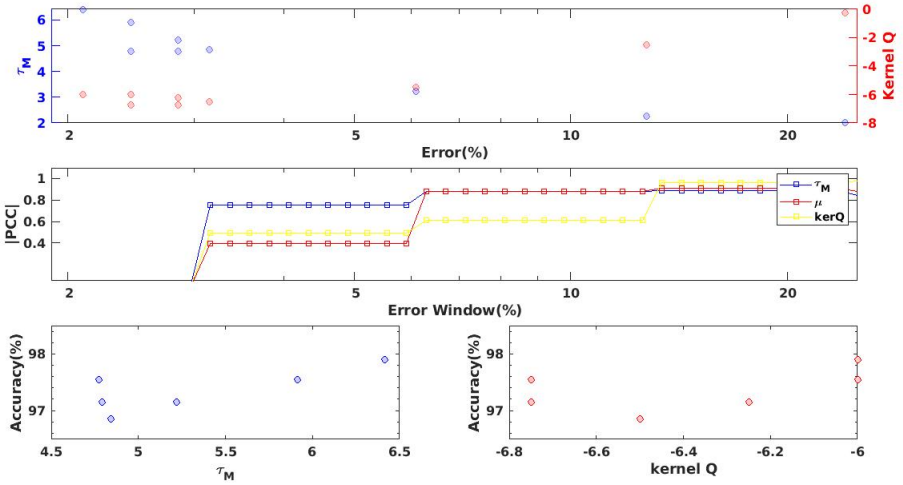

Figure 15: Correlation of τ_m, μ and $kerQ$

Metric	PCC at low error regime
τ_m	75.15%
μ	39.77%
$KerQ$	56.84%

Table 3: PCC of different metrics

3.5 Self Organised Criticality

In this section we summarise the results obtained in [9]. Critical state is a biologically observed state in which most naturally occurring substances are present in, such as the flowing river, the glowing star, the falling sand in an hourglass, etc. a self organised critical system, spontaneously reaches a state of dynamic equilibrium in which it's sensitivity is enhanced and it becomes robust to small noise in the system even though it is receptive of it. The state of dynamic equilibrium is the critical state. In several neuro-cortical experiments, signatures of self organised critical system has been found, which leads us to believe that brain also functions like a self organised critical system. Breaking down the functionality of a self organised critical Liquid State Machine could provide us with a lot of insight in the dynamics of real cortical networks. This study could further be explored in the perspective of memory and its behaviour for critical/non-critical systems.

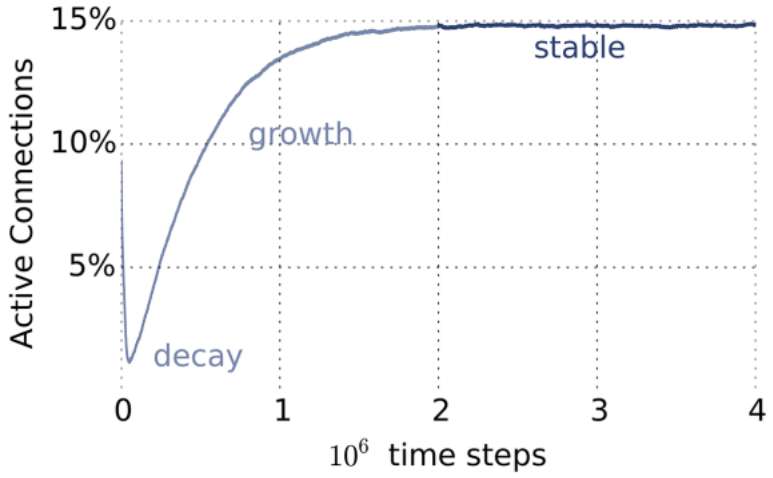

Figure 16: Evolution of the network activity over time

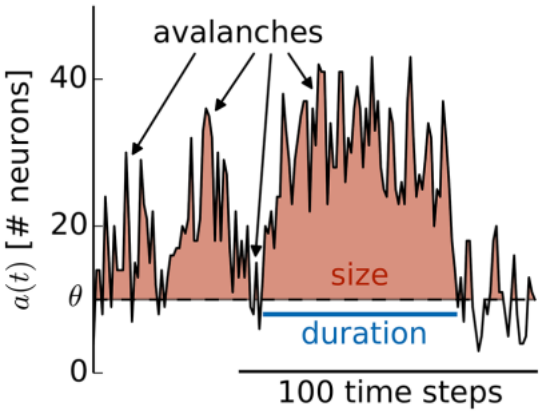

Figure 17: Occurrence of avalanches at the stable/critical state

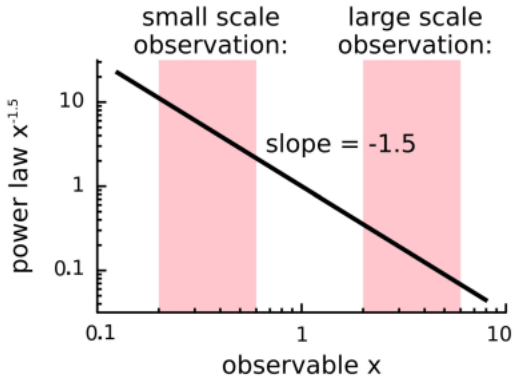

Figure 18: Power law followed by distribution of both avalanche size and duration

As shown in figure 17, once the system evolves into the stable state, we can see occurrence of avalanches in the network activity. This happens due to noise being present or a feeble input being fed into the system. The fingerprint of a self-organised critical system is that these avalanches follow some power law both in its size and duration. This fingerprint is observed in real biological experiments involving cortical networks. Simulating such networks could open up domains of further interesting research.

Chapter 4

4 Loihi: Analysis and Application

4.1 Intel's Loihi for SNNs

Artificial neural networks (ANNs) have very developed architecture for present day simulations that enable them to perform computations over huge networks in comparably lesser time than LSMs or other SNN networks. Since we know that Brain has more than 100 Billion Neurons, we must also have an architecture that enables us to simulate huge SNN Networks. Intel in 2018 announced a manycore neuromorphic chip called Loihi. This chip enables us to achieve faster computations on huge SNN Networks.

The neuromorphic computing field of research spans a range of different neuron models and levels of abstraction. Loihi is motivated by a particular class of algorithmic results and perspectives from computational neuroscience and recent neuromorphic advances. One loihi chip contains 0.13 million neurons distributed across 128 neuromorphic cores thus, giving a significant computation power for simulating SNNs with LIF neuron as building block.

4.2 Comparison Between MATLAB and Loihi Simulations

In this part we will discuss about the spike time comparison for each neuron for same network models. This comparison will tell us the difference between computer simulations and hardware implementations. This comparison is also very important because once we start performing various tasks in Loihi which have already been done in the previous state of the art studies, we will be able to discuss how the difference in the spike-times for each Neuron contributes towards the difference in the hardware and software simulations. We compared the spike timings for four different Networks ; single neuron , two neuron reservoir, 10 Neuron reservoir and 150 Neuron reservoir. We show the results for the spike time comparison so obtained in the figures below:

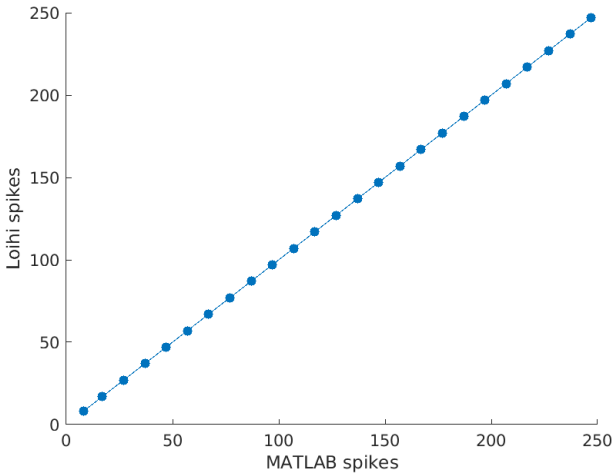

Figure 19: spike time comparison for Reservoir Neurons = 1

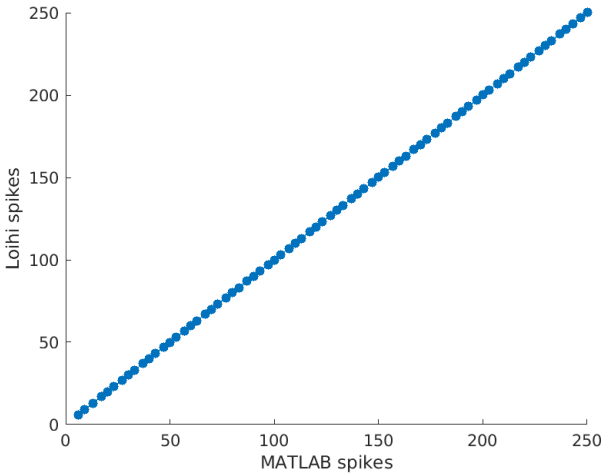

Figure 20: spike time comparison for Reservoir Neurons = 2

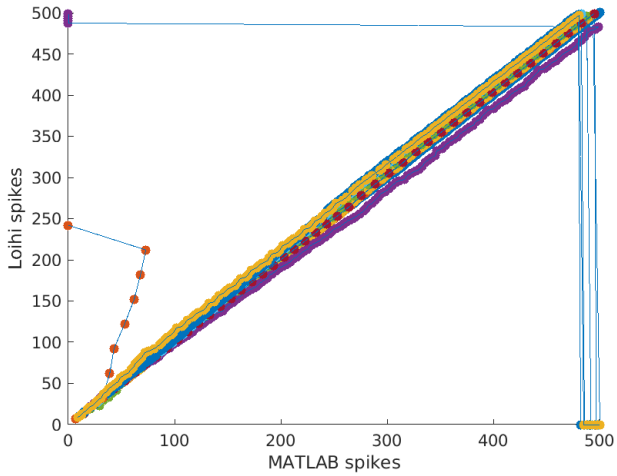

Figure 21: spike time comparison for Reservoir Neurons = 10

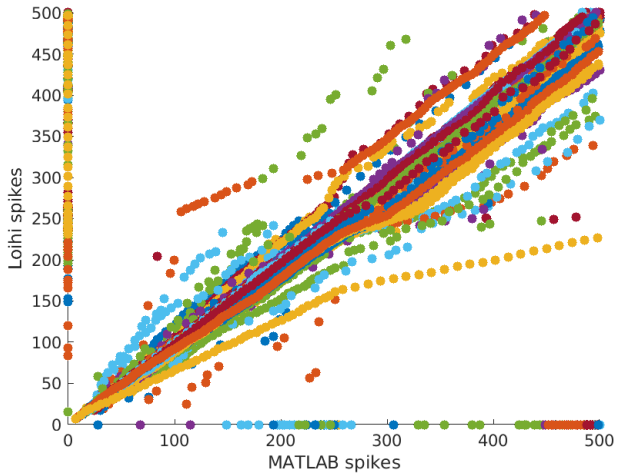

Figure 22: spike time comparison for Reservoir Neurons = 150

For low number of Reservoir we notice that the spike times are completely synchronized. For higher values we start to see some deviation from the ideal line. These deviations in spike timings occur because of the accumulation of bit precision error in Loihi when compared to MATLAB. When we look at the data for the 150 reservoir neurons network we observe that the avg difference in the total number of spikes produced per neuron is 3.84 whereas the difference in spike timings is almost 49.48%. We can hence say that when we look at the broader spiking pattern, both Loihi and MATLAB will be somewhat similar however individual spike timings will be very much different. The next question we asked was how this difference in spike timings reflect in the accuracy produced in both of these systems using the same system configurations.

4.3 Comparing Loihi and MATLAB : Accuracy, τ_M and μ

The Performance on TI46 dataset is calculated for the same LSM Network in both Loihi and MATLAB. The Prediction metrics τ_M and μ are also compared. It is observed that the average pointwise difference in accuracy is 0.93%, which tells us that even though the spike times are hugely shifted between MATLAB and Loihi, the accuracy produced by both the systems are strikingly similar. It is also observed that average pointwise difference between Lyapunov exponent for these two systems is 1.21% and the average difference in τ_M is 14.75%.

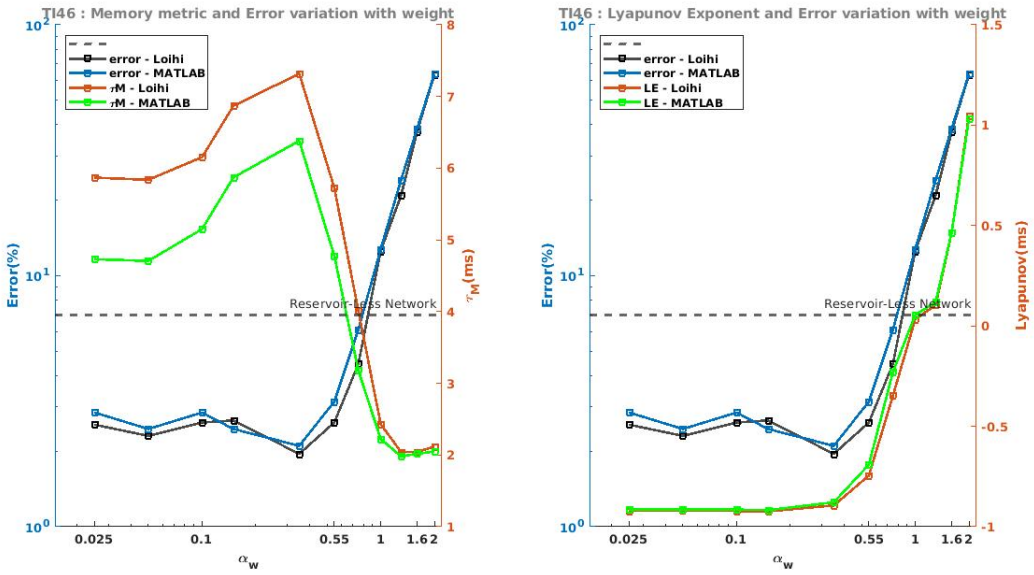

Figure 23: Comparison between MATLAB and Loihi on the basis of performance : TI46 dataset

4.4 C-elegans: Klinotaxis and Klinokinesis

Klinotaxis and Klinokinesis are the processes using which simple biological organisms like C-elegans, use basic NaCl concentration gradient in their surroundings to navigate. They either gravitate towards higher concentration or lower as per their bodily requirements. The paper [10] presents a Spiking Neural Network model for autonomous navigation on a given NaCl concentration space. In particular, the focus of this paper is on the problem of contour tracking, wherein the bot must reach and subsequently follow a desired concentration setpoint. In Klinokinesis mechanism, the bot constantly checks the gradient of concentration around it and moves in the positive gradient direction if it is below the setpoint and negative gradient direction if it is above the setpoint. Though Klinokinesis gives the correct result, the path taken by the bot is not the shortest path as it only employs the direction of the gradient and not the magnitude. Here, Klinotaxis

comes into the picture, it uses the magnitude of the gradient to enable the bot to reach the setpoint using the shortest contour.

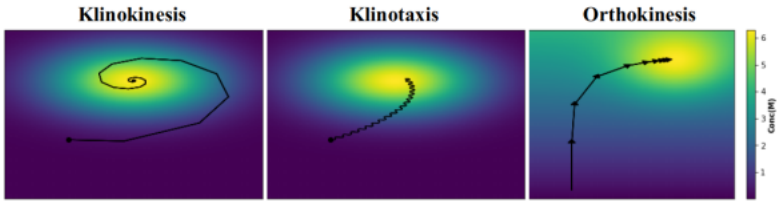

Figure 24: Klinotaxis, Klinokinesis and Orthokinesis

Orthokinesis is just used to alter the speed of the worm as it gets closer to the setpoint. The following block diagram is used in the paper to achieve the above results.

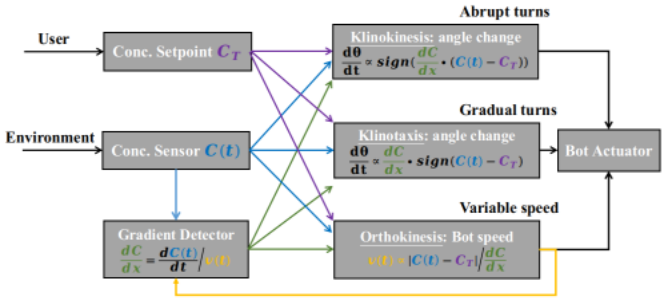

Figure 25: Block Diagram of the network

4.5 One-dimensional Navigation

Our aim is to implement some form of navigation in Intel's Loihi. Klinokinesis being the simplest approach, we start with the objective to implement a one-dimensional form of it. The differentiating factor of this implementation from the implementation in [10] is that here, we aim to achieve fully spike based decision-making and operational tasks. This might not only be a promising Neuromorphic hardware application but might also potentially provide insights on how these operations actually take place in cortical networks.

4.5.1 Circuit Diagram

In Loihi, the following network is used to model the simplified klinokinesis navigation. The setpoint concentration block and the present concentration block constantly provide the spike encoded signal of setpoint concentration and present concentration respectively. The spike response of the neurons are dependent on whether the positive weighted signal has higher frequency. The output of these neurons are used to determine the decision, which is to chose between the left and right direction. Once a choice

is made, the decision block signals the motor network to make the worm move and send back the newly sensed concentration to the present concentration block.

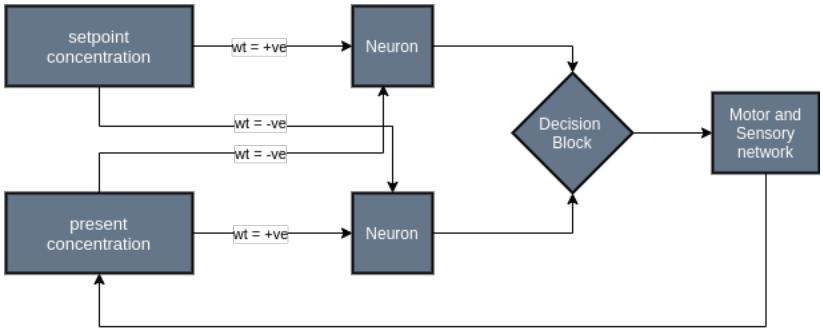

Figure 26: Block Diagram of the all spike Klinokinesis network

4.5.2 Concentration Encoding

The concentration space around the worm has to be encoded in spikes in order to use them in the above network. This task is performed as shown in the figure 28. The point in 1-D space with higher concentration has higher frequency and the one with lower concentration has lower frequency. In our simulation we are working with a concentration space consisting of 5 points f1, f2, f3, f4 and f5 having spike encoded frequencies as 1Hz, 1.25Hz, 1.67Hz, 2.5Hz and 5Hz respectively.

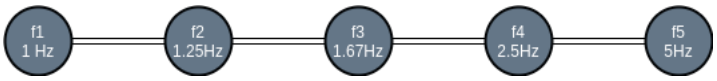

Figure 27: The concentration space encoded in frequency

The frequency encoding of each point is shown in the figures below

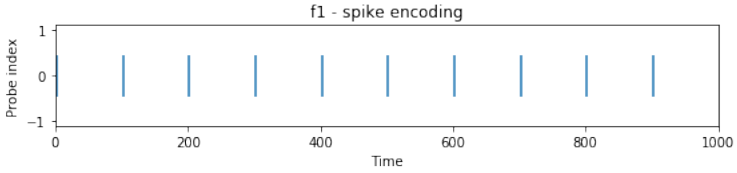

Figure 28: spike encoded f1 = 1Hz

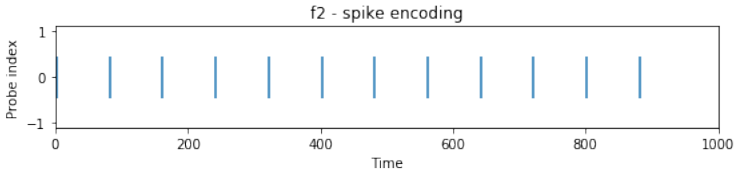

Figure 29: spike encoded f2 = 1.25Hz

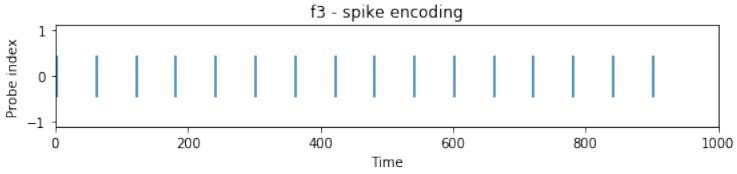

Figure 30: spike encoded f3 = 1.67Hz

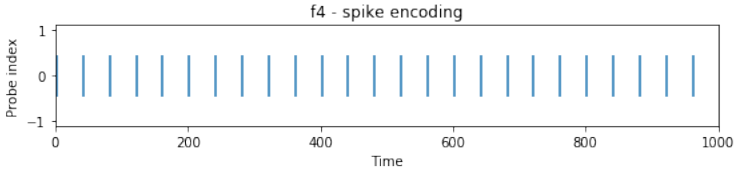

Figure 31: spike encoded f4 = 2.5Hz

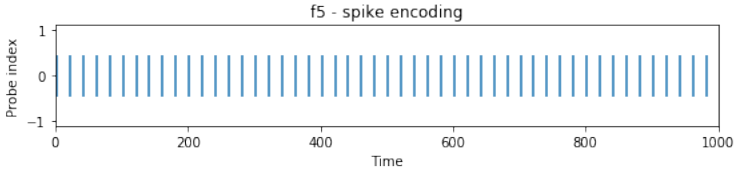

Figure 32: spike encoded f5 = 5Hz

4.5.3 Subtraction

The Decision making block works on purely spikes because at a particular time window, either only one of the neurons are firing or both of them are not firing. In one neuron is firing and the other is not, then this gives us a particular direction depending on which of the two neurons is firing. If both of them are not firing, then this simply implies that the worm has reached the setpoint concentration and it has to stop moving. Once these decisions are made, they are passed on to the motor neurons which move the worm and then give back the next concentration value that it moves on to. This block is not spike based in our implementation, but mathematically implemented on python. The subtraction of two spike based signals help in decision making as follows

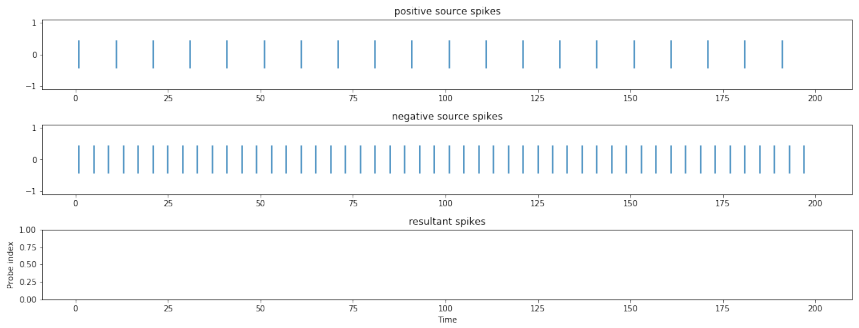

Figure 33: Subtraction of a high freq signal from a low freq signal

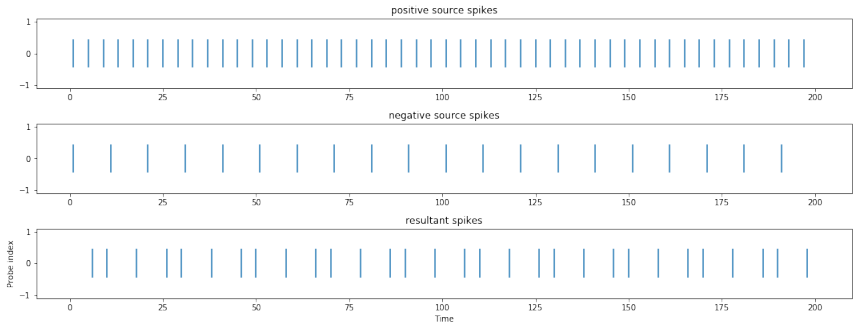

Figure 34: Subtraction of a low freq signal from a high freq signal

This tells us that if setpoint (C_T) is of higher frequency than the present concentration ($C(t)$), then their subtraction $C_T - C(t)$ in bottom neuron would give resulting spikes like in Figure 35 and the top neuron would not spike like in Figure 34. This would tell the decision block to move upwards in the concentration space, and similarly if the frequencies were the other way round. Once both the frequencies become same, as a result both the neurons would stop spiking.

4.5.4 Implementation

For the Implementation, a time window is selected in which the network would operate on the present concentration, after this time window the decision would be passed on to the motor neurons who would then move and give back the new concentration value to be used for the next time window. In our experiment, the concentration space is as shown in the previous sections i.e. a 1-D space with 5 points f1, f2, f3, f4 and f5. Each point has a unique frequency representation as shown in the subsection 4.5.2. We perform various experiments providing different setpoint and initial concentrations to the neuron. It can be observed from the plots below that at the end of the simulation the worm reaches its setpoint concentration, i.e. matches the setpoint frequency.

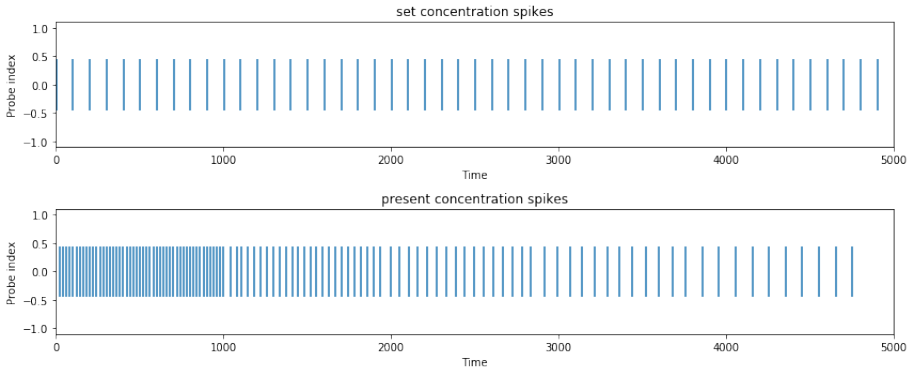

Figure 35: setpoint frequency = f1 = 1Hz, initial frequency = f5 = 5Hz

The operating window chosen for each simulation is approximately 1000 time steps which is close to 10s.

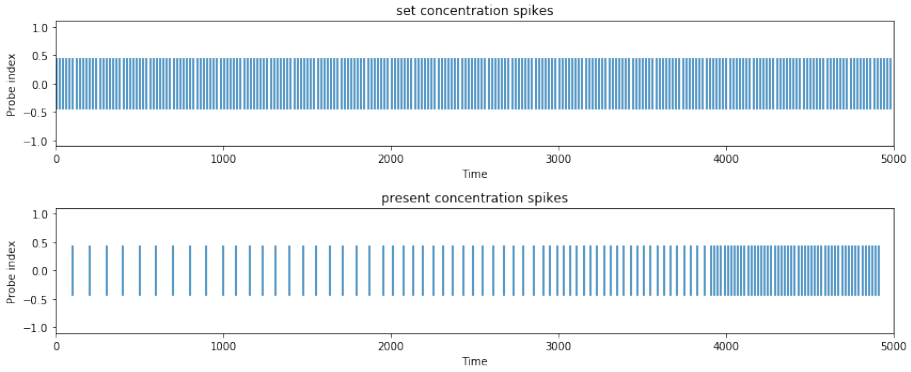

Figure 36: setpoint frequency = f5 = 5Hz, initial frequency = f1 = 1Hz

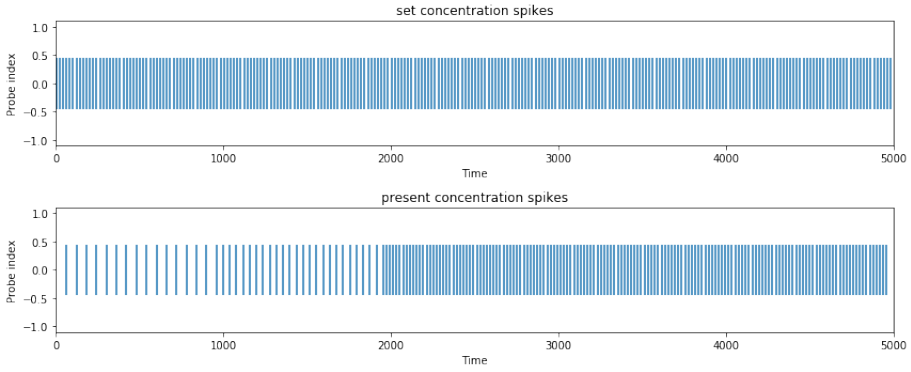

Figure 37: setpoint frequency = f5 = 5Hz, initial frequency = f3 = 1.67Hz

Chapter 5

5 Simplified Klinokinesis for Resource-Constrained Navigation on Loihi

5.1 Introduction

C. elegans shows chemotaxis using klinokinesis where the worm senses the concentration based on a single concentration sensor to compute the concentration gradient to perform foraging through gradient ascent/descent towards the target concentration followed by contour tracking. The biomimetic implementation requires complex neurons with multiple ion channel dynamics as well as interneurons for control. While this is a key capability of autonomous robots, its implementation on energy-efficient neuromorphic hardware like Intel's Loihi requires adaptation of the network to hardware-specific constraints, which has not been achieved. In this paper, we demonstrate the adaptation of chemotaxis based on klinokinesis to Loihi by implementing necessary neuronal dynamics with only LIF neurons as well as a complete spike-based implementation of all functions e.g. Heaviside function and subtractions. Our results show that Loihi implementation is equivalent to the software counterpart on Python in terms of performance - both during foraging and contour tracking. The Loihi results are also resilient in noisy environments. Thus, we demonstrate a successful adaptation of chemotaxis on Loihi - which can now be combined with the rich array of SNN blocks for SNN based complex robotic control.

5.2 Network Architecture

5.2.1 Single concentration sensor

C. Elegans employs the information of its immediate previous position from memory to steer itself in the desired direction. The end goal could be to reach a preferable concentration of NaCl in the environment which is correlated with its starvation. This desired end concentration is called the setpoint. The navigation process can be modeled through mechanisms like Klinokinesis, Klinotaxis, and Orthokinesis. Klinokinesis uses the single concentration sensor in the worm to regularly sense the current concentration (C) and utilize its memory of the past concentration to estimate the concentration gradient in its path and navigate towards the desired setpoint concentration (C_T). The essential aspects of the algorithm are as follows: If the worm is far away from C_T and cannot detect any surrounding gradient, it traverses its neighborhood randomly in search of food until it encounters a non-zero gradient. If the worm senses the gradient to be positive(negative), and the present $C < C_T (C > C_T)$, then it assumes the present direction of motion as favorable otherwise, it changes the course of its motion to subsequently align itself with the correct gradient. It is important to note that the

klinokinesis uses only the sign of the gradient. In the following subsections, we discuss the network used to implement this algorithm in Loihi.

5.2.2 Neuron Model

The algorithm is implemented using the Leaky-Integrate and Fire (LIF) neuron model which is the neuron model supported by Loihi. The membrane current and potential update equations of a typical LIF model used in Loihi are as follows.

$$u_i(t) = u_i(t-1) \cdot (2^{12} - \delta_i^{(u)}) \cdot 2^{-12} + 2^6 \sum_j w_{ij} \cdot s_j(t) \tag{12}$$

$$v_i(t) = v_i(t-1) \cdot (2^{12} - \delta_i^{(v)}) \cdot 2^{-12} + u_i(t) + u_{bias}(t) \tag{13}$$

where $u_i(t)$ and $v_i(t)$ denote the membrane current and potential of the i^{th} neuron at time t respectively. $\delta_i^{(u)}$ and $\delta_i^{(v)}$ represent the current and voltage decay. w_{ij} is the weight of the synapse originating from the j^{th} neuron and connecting the i^{th} neuron. $s_j(t)$ holds value 1 if the j^{th} neuron has spiked at time t and holds zero otherwise. $u_{bias}(t)$ is the bias current or external input current to the neuron at time t. The i^{th} neuron produces a spike when its potential $v_i(t)$ exceeds a certain potential threshold v_{TH}, post which, its potential is reset back to zero. The various parameter and their corresponding values used in our simulations are tabulated below

Parameter	Value (unit-less)
$\delta_i^{(u)}$	2^{12}
$\delta_i^{(v)}$	1
$w_{excitatory}$	128
$w_{inhibitory}$	-128
u_{bias}	0
v_{TH}	6400

Figure 38: LIF neuron model parameters for simulations

5.2.3 Working Principle

A favorable direction of further motion for the worm is determined by finding out whether the worm is currently moving towards an increasing or decreasing concentration value. Hence, determining whether concentration is currently increasing(gradient ¿ 0) or decreasing(gradient¡0) is of primary concern in klinokinesis. As mentioned in Section I, [3] achieves this functionality of estimating the sign of the gradient by employing ASE neurons. ASER neurons take as input $H(C - C_R)$ or $H(C_R - C)$, where C_R($C_L$used for ASEL neurons) is an internal neuron parameter representing the previous concentration and $H(.)$ is the heaviside or unit step function. In this algorithm, however, we use the heaviside function implemented in spikes to determine the sign of the gradient. Consider

$C(t)$ as the present concentration of the worm and $C(t-1)$ as the concentration of the immediate previous position. If

$$H(C(t) - C(t-1)) = 1 \tag{14}$$
$$H(C(t-1) - C(t)) = 0 \tag{15}$$

Then

$$C(t) - C(t-1) > 0 \tag{16}$$

which implies that gradient > 0. If

$$H(C(t) - C(t-1)) = 0 \tag{17}$$
$$H(C(t-1) - C(t)) = 1 \tag{18}$$

Then

$$C(t-1) - C(t) > 0 \tag{19}$$

which implies that the gradient < 0. However, If both the Heaviside functions are zero, then it simply means that $C(t) = C(t-1)$ and the gradient is zero as well. Figure below demonstrates how we implement the Heaviside function using spiking neurons. The concentration values are frequency encoded in spikes such that the higher the concentration, the higher is the spike rate. Due to the spike encoding of concentration values, temporal spike patterns cannot reveal instantaneous results but require a time frame to perform subtraction during which the concentrations remain fixed as can be observed from the figure. This pre-defined time frame is referred to in this paper as the time window demonstrated by the vertical black dashed lines in the figure and has a fixed value of 4000 time steps for all the simulations. If the resultant neuron spikes more than once in the corresponding time window, then the result of the Heaviside is said to be 1, else it is 0. In the figure, $C(t)$ is represented as C_1, and $C(t-1$ is represented as C_2. It can be observed from the figure that in the first time window, $C_1 < C_2$, which means that $H(C_1 - C_2)$ [represented by N_3] should be 0 and $H(C_2 - C_1)$ [represented by N_4] should be 1 which agrees with the spiking approximation of result. It can also be observed from the response of N_3 that the subtraction operation is capable of showing the correct result only if operated over a long enough time window. Another important thing to note is that at the beginning of the third time window, N_3 spikes once due to residual potential but the Heaviside still shows the correct result which demonstrates the robustness of the algorithm.

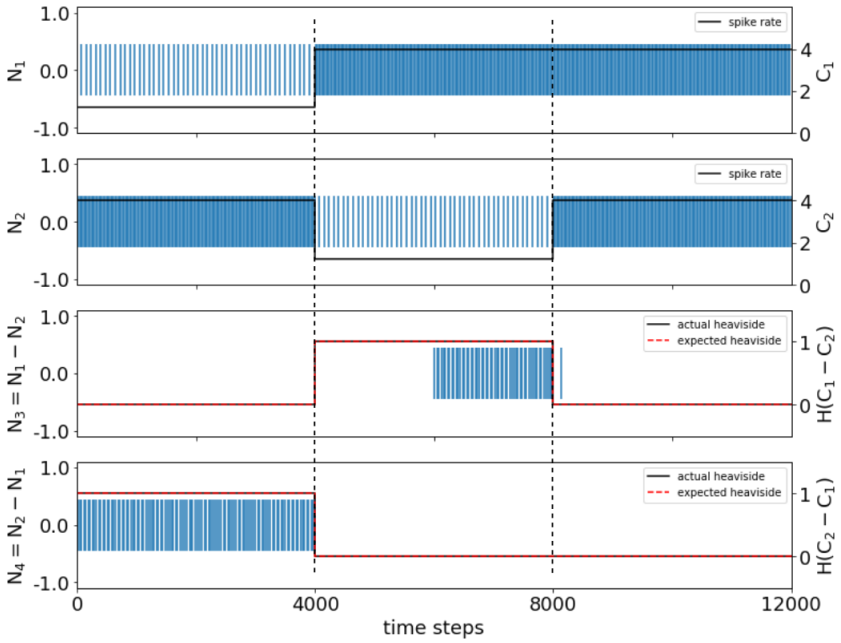

Figure 39: Spiking implementation of Heaviside function

5.2.4 Gradient Detection

The block diagram of the proposed network is shown in the figure below. The present concentration sensed from the interactive spike sender acts as the response of the neuron $N_{present}$ which when delayed by a time window length becomes the spike train encoding of the previous concentration and hence the response of neuron N_{prev}. Similarly, the setpoint concentration is generated from a basic spike sender for neuron N_{set}. Just as demonstrated in subsection B, the responses of the neurons $N_{gradpos}$ and $N_{gradneg}$ are the approximation to $H(C(t) - C(t-1))$ and $H(C(t-1) - C(t))$ which in turn determine the sign of the gradient in the present time window as positive and negative respectively.

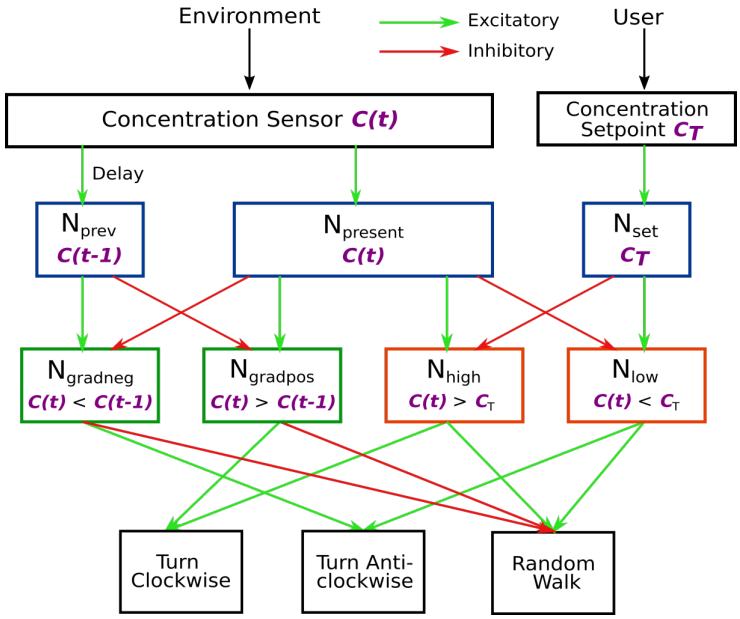

Figure 40: The Block Diagram

5.2.5 Navigation

From the block diagram shown in the block diagram, it can be observed that the neurons N_{high} and N_{low} respond when the present concentration $C(t)$ is higher and lower than the setpoint C_T respectively. If $C(t) > C_T$ then it implies that the worm must align itself towards a negative gradient or If $C(t) < C_T$ then it must align itself towards a positive gradient to reach the setpoint. If both $C(t)$ and C_T hold the same value, then it must mean that the worm has reached its destination successfully and hence must stay around it. In our algorithm, there is no concept of velocity since the concentration space is quantized and hence the worm changes its position by 1 element of the 2D space matrix after each time window. Since the concentration space is also quantized, there might occur instances where a plateau (zero gradient region) is encountered near the setpoint concentration. In such cases, the worm is tuned to perform a random walk after which it would reach the edge of that plateau and realign itself towards the correct direction. The navigation control layer works as follows: If $C(t) < C_T$ (N_{low} is spiking) and the worm is moving on a negative gradient ($N_{gradneg}$ is spiking), then this direction of motion is undesirable as the worm must rise in terms of surrounding concentration, in this case, the worm is tuned to turn anti-clockwise by 45^o. Similarly, If $C(t) > C_T$ (N_{high} is spiking) and the worm is moving on a positive gradient ($N_{gradpos}$ is spiking), then the direction is undesirable and it is tuned to turn clockwise by 45^o. Finally, if neither $N_{gradpos}$ nor $N_{gradneg}$ are spiking then the worm is tuned to explore randomly by choosing an angle of turning with equal probability from $-45^o, 0^o, 45^o$. This can be modeled by providing a bias current to the random walk neuron so that it also captures the case in which it encounters a plateau at the setpoint concentration,

i.e. none of the 4 neurons from the second layer are spiking, and hence, in this case, as well, the worm will perform random exploration. For the rest of the possible cases, in which the present direction is desirable, the direction of the motion of the worm is kept unaltered. The magnitude of the turning angle is not continuous and fixed to 45^o because the concentration space is not continuous and hence rotation is possible only in multiples of 45^o. For the subsequent simulations in this paper, the final layer neurons have not been explicitly implemented as neurons in Loihi, rather, only the functionality has been captured for the sake of time.

5.2.6 Network Response

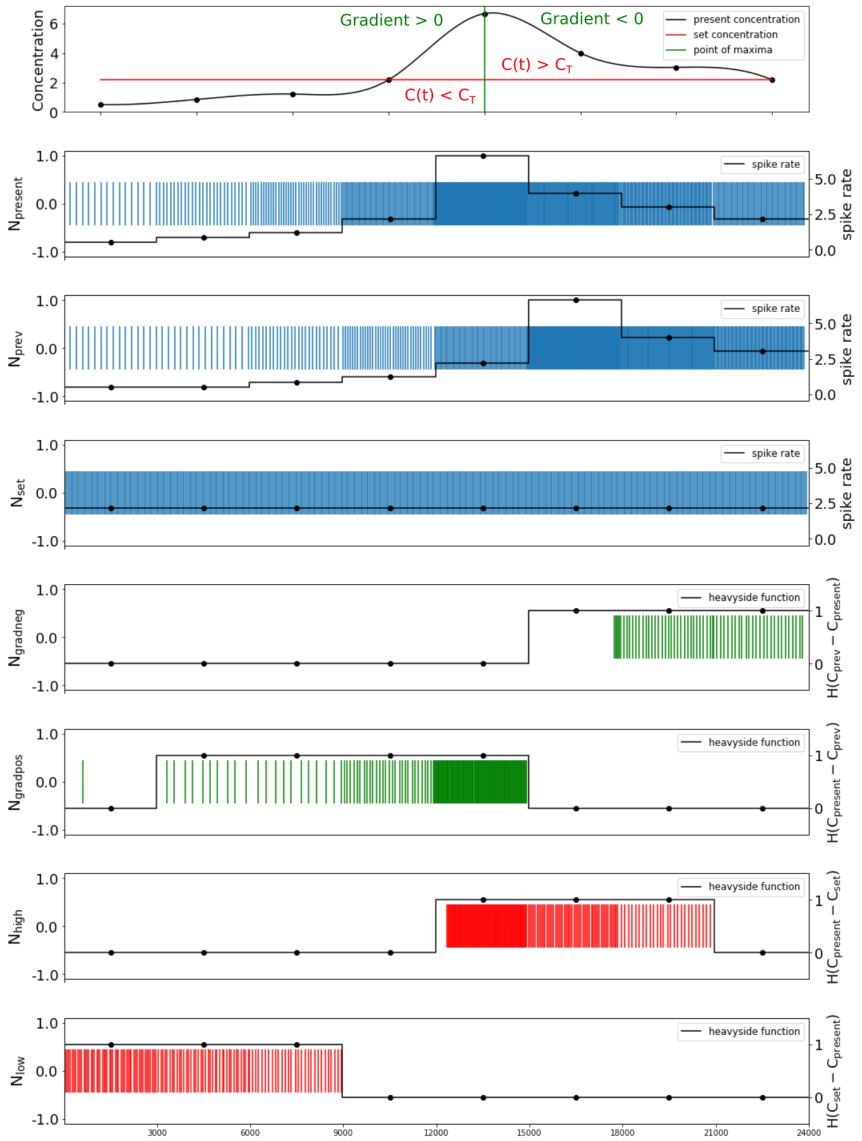

Figure 41: A sample concentration profile and the corresponding neuron responses

The figure above demonstrates a sample temporal concentration profile of the worm and the corresponding responses of each neuron in the network. The first plot shows the temporal concentration profile, the setpoint concentration, and the point of maxima. It also divides the concentration space into 4 overlapping regions where Gradient > 0 (region before the maxima), Gradient < 0(region after the maxima), $C(t) < C_T$ (region before the point of intersection of C C_T), and $C(t) > C_T$ (region after the point of intersection of C C_T) are pointed out. These regions are color-coded the same as the neurons whose spiking behavior determines whether the condition of that region is true. The second plot shows the encoding of chosen concentration values by $N_{present}$ for each time window. The third and fourth plots show the delayed response of N_{prev} and the constant setpoint response of N_{set}. The fifth and sixth plots validate the coinciding of the spiking regions of $N_{gradneg}$ and $N_{gradpos}$ with the Gradient < 0 and Gradient > 0 regions (green) marked in the first plot respectively. The final two plots also validate the coinciding of spiking regions of N_{high} and N_{low} with the $C(t) > C_T$ and $C(t) < C_T$ regions (red) marked in the first plot respectively. It can be noticed that there are time windows in which neither N_{high} nor N_{low} are spiking. In these time windows, the present concentration is equal to the setpoint. It should be noted that this is just a sample profile meant only for studying the neuron responses and hence navigational rules are not enforced here.

5.3 Simulations and Results

The simulations are set in a concentration space of size 101 x 101 where each dimension spans a range [0,100]. The concentration space is quantized using 20 concentration values given by 0.25Hz, 0.37Hz, 0.5Hz, 0.67Hz, 0.85Hz, 1Hz, 1.1Hz, 1.22Hz, 1.39Hz, 1.61Hz, 1.89Hz, 2.17Hz, 2.56Hz, 3.03Hz, 3.45Hz, 4Hz, 4.76Hz, 5.56Hz, 6.67Hz, 8.33Hz carefully chosen such that each concentration is easily distinguishable from another in spiking domain and the subtraction operation yields a result in an affordable time window of duration 4000 time steps. The setpoint concentration for all the simulations is fixed to $C_T = 4.0$Hz. Figure below is a typical simulation demonstrating the ability of the worm-bot to perform a random exploration when it is unable to sense any gradient. The worm is launched at the point (10,50) [purple dot] with concentration value C = 0.25Hz. Towards the end of the simulation, it starts to experience a change in the gradient sign and the trajectory hence starts to become more aligned.

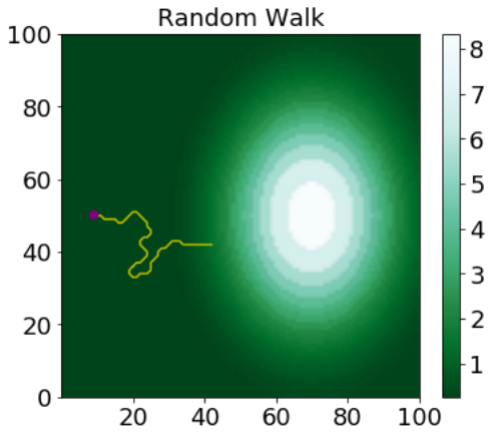

Figure 42: Random Walking on a gradient-less concentration surrounding

In Figure below, the initiating point of the simulation is at the purple dot with concentration value $C = 2.56$Hz and C_T being the fixed 4.0Hz. The bot starts to ascend the gradient in a more defined trajectory than that of figure above, although it is not exactly a straight line due to the plateaus formed by quantization.

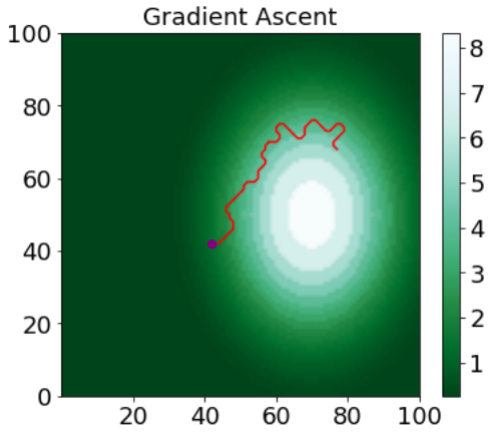

Figure 43: Gradient Ascent

Figure below displays a typical contour tracking trajectory traced by the worm.

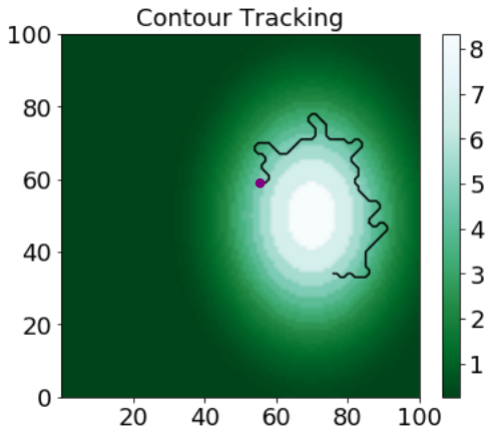

Figure 44: Contour Tracking

The complete Contour Tracking trajectory of the worm from the initial point (10,50) is plotted in Figure below and the trajectory of the worm in the vicinity of a noisy concentration space with White Gaussian Noise of $= 0$ and $= 0.05$ is plotted in Figure below that. The average absolute deviation during the tracking of the contour for a noisy trajectory was found to be 7.08 units in the 101x101 space which underlines the robustness of the algorithm.

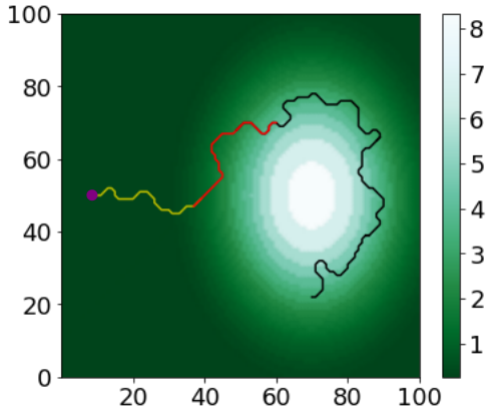

Figure 45: Complete Trajectory of the worm

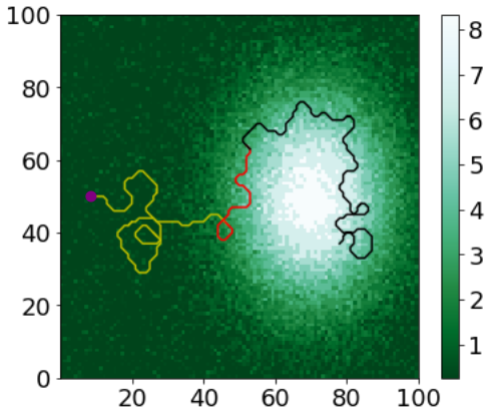

Figure 46: Complete Trajectory of the worm in a noisy environment

The absolute tracking deviation values are calculated by determining the maximum physical distance of the trajectory from the ideal contour of setpoint concentration 4Hz. We chose to calculate the deviation in this manner because deviation calculation solely based on concentration values will also capture the effects of non-linear encoding and quantization which is undesirable in this case. We further implemented the same algorithm on software using Python, capturing the functionality of Interactive Spike Sender as well, to validate the correctness of operation and comparability of time consumption with Loihi. We ran a total of 40 simulations with the same initial conditions, random seed(different for subsequent simulations), and concentration setpoint in each simulation for Python and Loihi. The trajectories for both Python and Loihi are plotted for one of the simulations in Figure below.

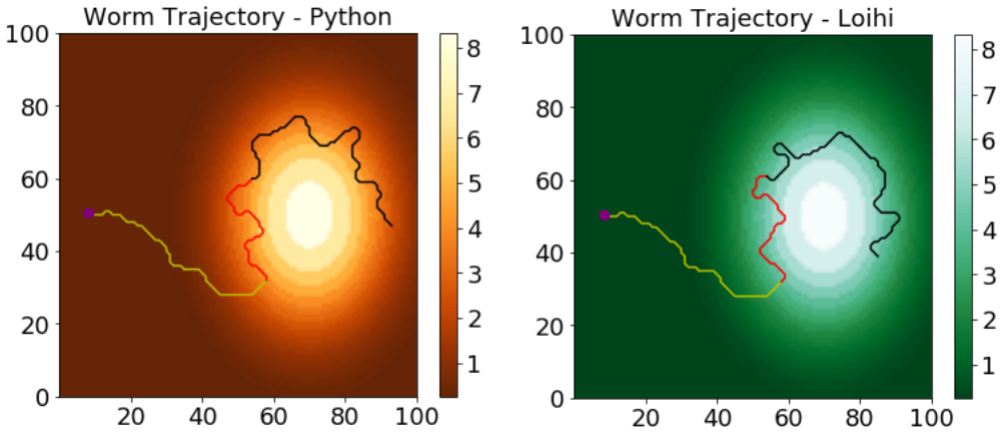

Figure 47: Python trajectory vs Loihi trajectory

The simulation results for all three implementations are tabulated in the Table below. Setpoint foraging time is determined by measuring the time it takes for the worm to reach the setpoint for the first time during the simulation. The foraging time

of Loihi's worm on average was found to be 1.92% greater than the foraging time of Python's worm which exhibits the time comparability of the two implementations. The table also shows the average contour tracking deviation for all the cases relative to the deviation observed in python. The difference in trajectory illustrates the closeness of the trajectories traced by the worm under the same conditions and parameters in Loihi w.r.t the trajectory traced by python. This difference arises because Loihi and Interactive Spike Sender have no common notion of time, which leads to small spike-time irregularities. These irregularities accumulate over time to produce more differences as the worm travels further. Hence we find that the normalized distance between the final points in the trajectories of Loihi and python exhibits the cumulative irregularities and is a good metric to represent the difference in their trajectories. This average difference of trajectories between Loihi and python is 6.73% which establishes the congruency in their trajectories.

Simulation	Avg setpoint foraging time	Avg relative tracking deviation w.r.t Python	Avg difference in trajectory w.r.t Python
Loihi	3906.13s	1.216	6.73%
Loihi (Noisy)	5689.36s	1.414	-
Python	3832.42s	1	-

Figure 48: Table comparing Python and Loihi simulations

Research has shown that Neuromorphic hardware reduces the simulation time (by 2% in CPUs) and power consumption(by up to 100 times) [9][10] as compared to their software counterparts. We believe that the timings here are comparable and not definitive of the fastness/slowness of Loihi as compared to software because of the very small network used. These results hence highlight the fidelity, correctness, and time comparability of the hardware implementation of this algorithm with respect to the software.

Chapter 6

6 Future Prospects

The Future Prospects of the current work are huge. Some of them are listed below

- Implementation of Self-Organised criticality in Liquid state Machines and determining the implications of memory for that system
- Implementation of Stochastic Oscillations in Liquid State Machines and determining the memory parallel to biological observations
- Studying the effect of Noise on the performance and dynamics of Liquid State Machines
- Developing the simulation to perform adaptive klinokinesis i.e. klinotaxis using the magnitude of the gradient
- Developing a closed system form for Liquid State Machines implementing its feature characteristics

References

[1] Maass, Wolfgang, Thomas Natschl¨ager, and Henry Markram. *Real-time computing without stable states: A new framework for neural computation based on perturbations.*. Neural computation 14.11 (2002): 2531-2560.

[2] Maass, Wolfgang, Thomas Natschl¨ager, and Henry Markram. *A model for real-time computation in generic neural microcircuits.* Advances in neural information processing systems. 2003.

[3] Ju, Han, et al. *Effects of synaptic connectivity on liquid state machine performance.* Neural Networks 38 (2013): 39-51.

[4] Legenstein, Robert, and Wolfgang Maass. *Edge of chaos and prediction of computational performance for neural circuit models* Neural Networks 20.3 (2007): 323-334.

[5] Gorad, Ajinkya, Vivek Saraswat, and Udayan Ganguly. *Predicting Performance using Approximate State Space Model for Liquid State Machines.* arXiv preprint arXiv:1901.06240 (2019)

[6] Verstraeten, David, et al. *Isolated word recognition with the liquid state machine* Information Processing Letters 95.6 (2005): 521-528.

[7] Wang, Qian, and Peng Li. *D-lsm: Deep liquid state machine with unsupervised recurrent reservoir tuning* 2016 23rd International Conference on Pattern Recognition (ICPR). IEEE, 2016..

[8] Lars Busing, Robert Legenstein, Benjamin Schrauwen *On Computational Power and the Order-Chaos Phase Transition in Reservoir Computing* Institute for Theoretical Computer Science Graz University of Technology A-8010 Graz

[9] Bruno Del Papa, Viola Priesemann, Jochen Triesch *Criticality meets learning: Criticality signatures in a self-organizing recurrent neural network* Public Library of Science (PLOS) One: 2017

[10] Shashwat Shukla, Rohan Pathak, Vivek Saraswat, Udayan Ganguly *Adaptive Chemotaxis for improved Contour Tracking using Spiking Neural Networks* International Conference on Artificial Neural Networks: 2020