Annales UMCS Informatica AI XIV, 3 (2014) 33–42
DOI: 10.2478/umcsinfo-2014-0023

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Liquid computing and analysis of sound signals

Rafał Cebryk[1]*, Grzegorz M. Wójcik[1]†

[1] *Institute of Computer Science, Maria Curie-Sklodowska University, Akademicka 9, 20-033 Lublin, Poland*

Abstract – Liquid Computing Theory is a proposal of modelling the behaviour of neural microcircuits. It focuses on creating a group of neurons, known as a liquid layer, responsible for preprocessing of the signal that is being analysed. Specific information is achieved by the readout layers, task oriented groups of neurons, taught to extract particular information from the state of liquid layer. The LSMs have been used to analyse sound signals. The liquid layer was implemented in the PCSIM Simulator, and the readout layer has been prepared in the JNNS simulator. It could successfully recognise certain sounds despite noises. Those results encourage further research of the computational potential of Liquid State Machines including working in parallel with many readout layers.

1 Introduction

The knowledge regarding modelling the activity of brain has been expanding for several decades. Computational approach to investigation of brain or its parts became one of the most important fields of neuroscience [**1**]. Experiments *in computo* often lead to explanation of cognitive and psychological phenomena [**2, 3**]. When strenghtened by commonly known methods of artificial intelligence they may show new quality of results helping to understand central nervous system functionality [**2, 3**].

The discovery of existence of neural microcircuits with potential universal real-time computational power led to many attempts of modelling and simulation. One of them is the Liquid State Machine. The model, described in this article, has been used to simulate the neural microcircuit receiving sounds and trying to recognise them despite noises. One of many attempts of using the LSM to recognize certain words has been

*rafal.cebryk@umcs.lublin.pl
†gmwojcik@umcs.pl

presented in [**4**]. The authors used the Matlab LSM toolbox to perform the simulation and recognize spoken digits.

We expect that using the LSM in attempts to recognize the voice will allow to achieve clear results despite high noises. The nature of the work of spiking neurons should lead to a high performance of the calculations. What is more the structure of the LSM allows to analyze the same input at the same time in many directions, extracting more information from the spoken words, like the mood of the speaker.

The Liquid State Machine (LSM) described in [**5**] can be methaphorically explained by a lake surface reaction to different disturbances. If some stones fall into the lake, a couple of waves will propagate on its surface. The wave patterns will differ depending on stone weights, shapes, velocities, places where they hit the surface and the angle at which they fall into the water. The surface of the lake represents both current disturbance and past disturbances that have just happened. After some time they vanish. Such a plane keeps a temporary history of past events. The LSM theory assumes that the neural microcircuit is modelled as such a methaphoric liquid layer that allows the input to spread and produce different liquid states covering both current and past changes. In order to analyse the input one or more observers, trained to extract certain parts of information, analyse the surface. One can specialise in identifying the concentration of stone hits, another one can check their velocity etc. Similarly, the LSM consists of one liquid layer and at least one readout layer.

The liquid layer is built of spiking neurons known as the 3^{rd} generation neurons. They have been introduced in order to simulate the brain dynamics. Older models could not deliver and process the information as quick as spiking neuron models.

The main advantage of spiking neurons is that they encode the information in the exact time of spike. A lack of spike in a certain time is also information. As a result, not only the number of spikes but also the exact times when they occur provide a piece of information. If the information is stored only in the frequency of signals, there is no way to process the information as quickly as the brain does (it takes about 100 ms to analyse and classify the visual templates by brain).

In [**6**] it is shown that the spiking neurons can simulate multi-layer 1^{st} generation neural networks by enforcing the synchronization on all the models in the network and treating an occurrence of spike in a current step as value 1, a lack of spike equals 0. What is more it has been emphasized, that such a synchronization significantly decreases its computational power, which lies in an asynchronous mode where subtle differences in spike transmission times provide a lot of information, that would be lost in the synchronous mode.

The 3^{rd} generation neural network was used in [**7**] by Wolfgang Mass to build a computational model for generic cortical microcircuit - Liquid State Machine. Mammalian brains process a continuous stream of data all the time. That process can not be split into separate computational steps. That is why the main part of LSM - the

liquid layer is built of spiking neurons, which theoretically work on continuous values of time[1]. In addition, that layer is not a task-oriented neuron circuit, but a high dimensional dynamic system which takes on different states depending on the input. Apart from the liquid layer the machine consists of also an input layer and at least one readout layer. The input layer is responsible for sending the input stream to the liquid layer. The liquid layer spreads the signal coming from the input layer and each readout layer analyses current liquid state and extracts certain information from it. In contradiction to the liquid layer, the readout layer is a definitely task-oriented network. Moreover, connecting many different readout layers to the same liquid layer allows to read different information from the input. Wolfgang Mass in [8] describes work did by brain during walking as an argument in favour of using the LSM to model the brain activity. The input stream coming from the eyes is being analysed in parallel to extract different information for different purposes. One piece of the information extracted can be looking for a place, where the foot will be placed if it is at the same level, higher or lower than before. At the same moment the brain is checking if there are no edges that we can hit with our leg, hand or head. In the meantime another "process" can compute important data to make another step, to keep the balance etc. As one can see such a stream of data contains a lot of information that should be extracted in different directions. That is why the LSM suits that situation as we can connect many readout layers, that will extract certain information coded in the layer state in parallel.

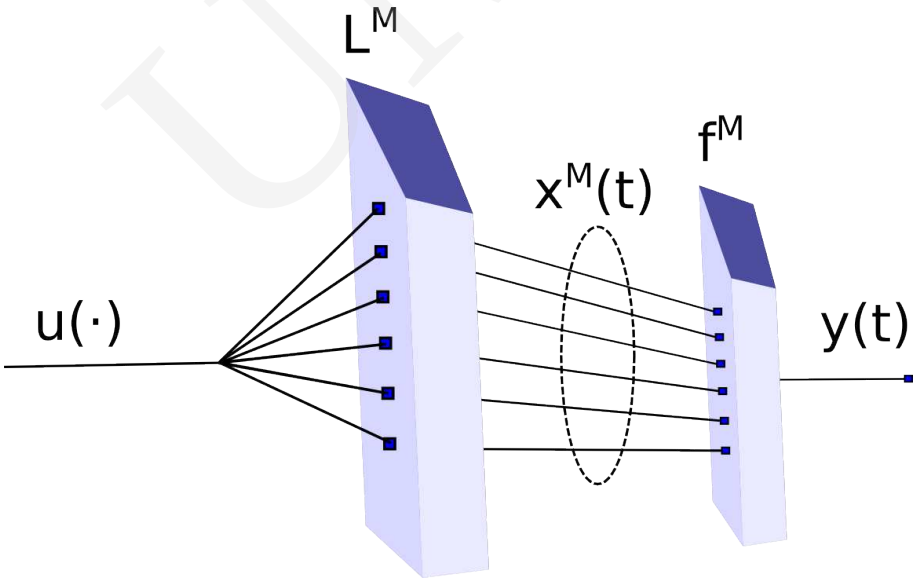

Fig. 1. The structure of the LSM

Fig. 1 shows the structure of the LSM. The input signal is represented by the function $u(\cdot)$. It reaches the liquid layer, which works as filter L^M, that all the time returns the

[1]During a simulation the continuous values are of course stored and used as discrete values.

current liquid state $x^M(t)$. The liquid state is a result of past disturbances $u(s)$, where $s \leq t$. It can be expressed as:

$$x^M(t) = (L^M\, u)(t) \tag{1}$$

The value of $x^M(t)$ is analysed all the time by the readout layer f^M, which produces the output state $y(t)$:

$$y(t) = f^M(x^M(t)). \tag{2}$$

2 Experimental set-up and simulations

In order to simulate the LSM we used the Parallel neural Circuit SIMulator (PCSIM [**9**]). The readout layer was achieved with Java Neural Network Simulator (JNNS), the successor of SNNS [**10**].

PCSIM is a parallel version of CSIM [**9**]. We have experience in PCSIM-based modelling [**13**]. It allows to perform simulations of heterogeneous networks consisting of different neuron and synapse models. The core of this tool is built in C++ which guarantees high performance of simulations. The main interface allowing to program the simulations is written in Python. The main advantage of such a solution is high efficiency in writing Python scripts and the above average flexibility of the code created in that language. What is more, Python [**11**] is full of various modules allowing to effectively analyse and visualize the effects of simulations. The PCSIM supports Message Passing Interface (MPI), which allows to split the simulation across many processes and computers grouped in clusters.

We used the LSM structure proposed by Mass assumptions in [**12**]. The input layer was connected to 30% neurons in the liquid layer. The liquid was a column of neurons of the dimensions 16 x 16 x 32 (8192 neurons). As the probability of connection between the input neuron and the columnar neuron was set to 30%, the input layer was connected to approximately 2458 neurons. The probability of connections inside the liquid layer depended on the position in the column and it was calculated from the euclidean distance between neurons. 80% of them were excitatory, and the rest was inhibitory. The parameters of both kinds of neurons are in Appendix A.

A set of 1017 neurons was randomly chosen from the excitatory models in order to record their activity and analyse it by the readout layer. In order to keep the fading memory property, the final value of each neuron was just the number of spikes sent by them during last 20% of time of their work. Those values were treated as input values for a readout layer, that was simulated with JNNS.

The readout layer was in fact a separate non-linear[2] neural network created in JNNS in order to analyse the output of the liquid layer. As the number of output neurons in the liquid was 1017, the number of the input neurons in the readout network was the same. The network consisted also of four hidden layers of 100 neurons each and the output four neurons representing the NATO words: Alfa, Bravo, Charlie and Delta.

[2]The neurons in the hidden layers used an activation function based on hyperbolic tangents.

The neurons are connected consecutively, all the input neurons are connected to all the neurons in the first hidden layer, all the neurons from the first hidden layer are connected to the second hidden layer etc. The network has been trained with the results of simulations on noised input files. For details of the neural network learning process see Appendix A.

In order to correctly train the readout module a script was generating input files with different noises. The degree of noise was described with three parameters: the probability of noise of each sample and the minimal and maximal degree of disturbance. The algorithm of adding noises to the samples is the following:

```
disturbance_range=max_disturbance-min_disturbance
noised_samples=[]
for sample_value in samples:
    if random.random()<disturbance_probability:
        noise=min_disturbance+disturbance_range*random.random()
        sample_value+=sample_value*noise
        noised_samples.add(sample_value)
```

Our LSM was used to recognise the first four words of NATO phonetic alphabet recorded on our own. This choice has been made due to its resistance to noises, as each word is built from different syllables.

The words have been recorded with the frequency 8000 Hz. The best results have been achieved when the input layer was built with one Leaky Integrate-and-Fire neuron[3], encoding the times of spikes on the base of the value of the sound. Since the frequency of the recorded words was 8000 Hz, the audio samples were provided with $1.25 \cdot 10^{-4} s = 0.125$ ms interval. The value of the signal was encoded in the number of spikes sent during the interval between samples. The minimal interval between spikes was 10^{-3} ms. 80% of time of the interval between samples has been used to store the value of the sound - it gives 0.1 ms. During that time an appropriate number of spikes was sent and still there was at least a 0.025 ms break before encoding the next sample. The maximum number of spikes that could be sent within that period is 100. It means that the values of the sounds were encoded in 100 different levels. In other words the quantization of the sound made in order to transfer it to the liquid layer split the range of possible sound values into 100 levels. The lowest level was encoded with one spike, the highest with 100.

[3]Other attempts have been made with 100-512 input neurons that split possible sound values into 100-512 levels and each neuron send a spike when the sound reached its level. It did not give the expected results, as the dynamics of the liquid was too low to produce states that could be recognised by the readout layer.

3 Results

Our simulation scenario contained almost 100 different runs that have been scheduled and executed with different parameters and architecture[4]. In order to repeat those simulations on different input we needed to set a constant value in the PCSIM seeds of random engines to make sure that connections between neurons are the same. Otherwise at each simulation PCSIM would create a new network with new connections between neurons based on random values accordingly to the probability of connections. A Python script executed a simulation with the same architecture on different input files. What is more, in order to correctly train the readout module, the script was generating new files based on the input files with appropriate disturbances (defined in the configuration file). As a result, for each input file a group of simulation results was achieved. The number of simulations executed from one input file equalled the number of different disturbances used. The disturbance impact was defined with two factors: the probability of noise occurence and the range of random change of input value. We have tested it on five different probabilities of noise occurrence: 5%, 10%, 15%, 20% and 25%. The ranges of disturbances were symmetric with respect to zero: $(-0.05 - 0.05)$, $(-0.1, 0.1)$, $(-0.15, 0.15)$, $(-0.2, 0.2)$ and asymmetric: $(0.1 - 0.2)$, $(0.2 - 0.3)$, $(0.3 - 0.4)$ and $(0.4 - 0.5)$.

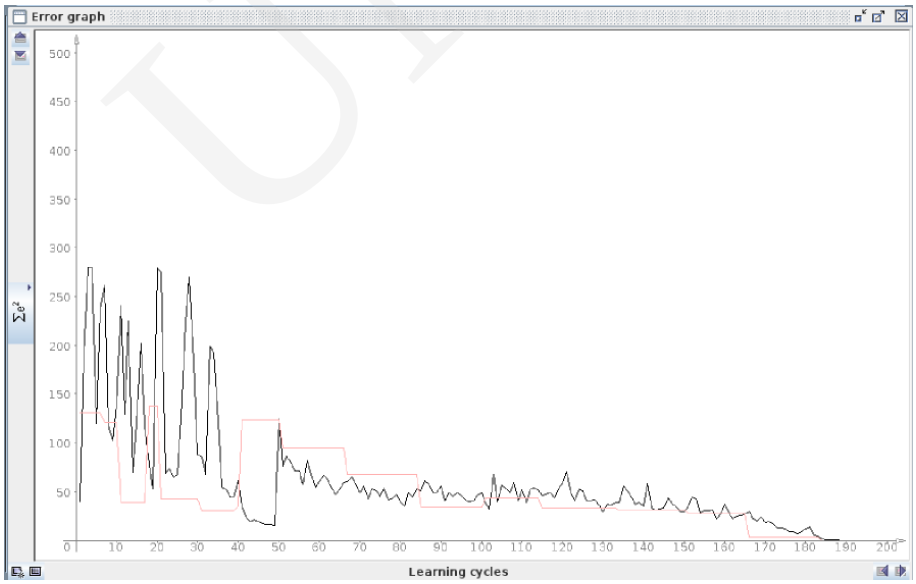

Fig. 2. Training squared error plot.

[4]We tried different combinations of a number of input neurons used to code the information and the number of output neurons whose spikes were recorded and analysed with the readout module.

The table in Appendix B shows the process of training the readout network. It contains the information regarding used algorithm, the number of learning cycles, the probability of noise for a sample and the ranges of disturbances.

The network was not trained on clean data but on the input with noises. The plot in Fig. 2 shows the value of the squared error during training. The black line shows the squared error of the training set and the red line shows the squared error of the verification set.

The results and performance of such training are shown in Fig. 3 and in the Table 1. The verification has been made on the input where the probability of disturbance was 25% and the level of disturbance equalled 0.4 - 0.5. The network was not trained on that set of inputs either, it was used only for verification. Note that accuracy of well trained network in all four cases exceeded 90%.

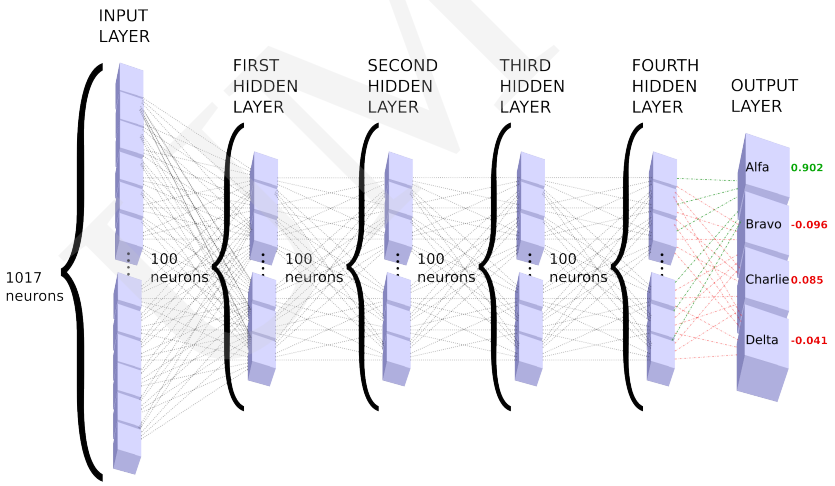

Fig. 3. JNNS network recognizing theAlfa pattern with 0.902 accuracy.

As it has been shown on the squared error plot, the network has learned to recognise all the four patterns. What's more it was even capable of recognizing a highly noised input. Similar results were achieved in [**4**].

The readout network could successfully recognise the patterns despite strong disturbances.

Table 1. JNNS network accuracy. Each column represents different inputs on the network, each row represents the value calculated by each output neuron.

	Alfa input	**Bravo input**	**Charlie input**	**Delta input**
Alfa neuron	0.902	-0.074	0.003	-0.078
Bravo neuron	-0.096	0.916	0.057	0.083
Charlie neuron	0.085	-0.059	0.901	-0.027
Delta neuron	-0.041	0.092	0.046	0.926

4 Conclusions

The results of performed simulations showed that the LSM can be used to recognise even strongly disturbed noisy patterns with the accuracy better than 90%.

However, all three modules: the input, liquid and readout layer must satisfy certain conditions in order to perform a given task. The input must produce enough spikes to obtain dynamic responses in the liquid layer. The liquid layer should be complex enough and diverse in order to let the input spread on the layer. In addition, it should produce different states for different inputs and avoid chaotic reactions. Finally, the readout layer should be able to recognise certain liquid states.

The possibility of using many readout layers to analyse the same liquid layer containing preprocessed input signal is very promising. It should be possible to use the LSM straight in the voice recognition problems and determine not only the exact spoken words, but also the sex or mood of the speaker without the need of copying the input signal.

It would be also interesting to check the performance of LSMs built of biologically realistic Hodgkin-Huxley neurons [**14, 15**] when performing the tasks discussed in this paper.

In fact, the simulated models recognised patterns, not words. The approach presented herein is then a prototype of real voice recognition. We will soon repeat those simulations with different attempts to code the sound samples not in linear values but in frequencies, so the final model would be able to recognise the word spoken by different people, with different rate, intonation etc. The continuous way of simulations should allow to perform real-time computation on voice, allowing to extract the information on the fly.

Appendix A - Parameters of JNNS Neural Network, Excitatory and Inhibitory neurons used in simulations

Excitatory model of neuron: The cell membrane capacity 2e-10 Fahrads;The cell membrane resistance 1e8 Ohms;The initial potential -60e-3 millivolts;The threshold potential -50e-3 millivolts; The resting potential -49e-3 millivolts;The reset potential -60e-3 millivolts;The refraction period 5e-3 milliseconds

Inhibitory model of neuron: The cell membrane capacity 3e-10 Fahrads;The cell membrane resistance 1e8 Ohms;The initial potential -60e-3 millivolts;The threshold potential -50e-3 millivolts; The resting potential -49e-3 millivolts;The reset potential -60e-3 millivolts;The refraction period 5e-3 milliseconds

: JNNS Neural Network Training The network was trained using the JE Backpropagation method with learning parameters set as follows $\eta = 0.2$, $d_{max} = 0.1$, $forceT = 0.5$ and 150 cycles for 1 step.

Appendix B - Readout training details

The neurons in the hidden layers used the Act_Act_TanH_Xdiv2 activation function.

No.	Training algorithm	Learning cycles	Disturbance probability	Minimal disturbance	Maximal disturbance
1	Rpropagation	10	0.05	-0.05	0.05
2	Rpropagation	10	0.10	-0.05	0.05
3	Rpropagation	10	0.15	-0.05	0.05
4	Rpropagation	10	0.20	-0.05	0.05
5	Rpropagation	10	0.25	0.10	0.20
6	Backpropagation	150	0.25	0.10	0.20

References

[1] R. Tadeusiewicz, "Modelowanie elementów systemu nerwowego z wykorzystaniem technik informatycznych, a zwłaszcza sztucznych sieci neuronowych" in "Na ścieżkach neuronauk pod redakcją naukową Piotra Fracuza", pages 13-34, Wydawnictwo KUL, 2010.

[2] R. Tadeusiewicz, "Modele elementów układu nerwowego w postaci sztucznych sieci neuronowych" in "Neurocybernetyka Teoretyczna", pages 109-127, Wydawnictwa Uniwersytetu Warszawskiego, 2009.

[3] R. Tadeusiewicz, "Using Neural Networks for Simplified Discovery of Some Psychological Phenomena" in "Artificial Intelligence and Soft Computing", Lecture Notes in Artificial Intelligence, L. Rutkowski et al., eds., editor, pages 104-123, vol. 6114, Springer-Verlag, Berlin – Heidelberg – New York, 2010.

[4] D. Verstraeten, B. Schrauwen and D. Stroobandt, "Isolated word recognition using a Liquid State Machine" in ESSANN'2005 proceedings - European Symposium on Artificial Neural Networks, Bruges (Belgium).

[5] W. Maass, T. Natschlaeger, and H. Markram. Computational models for generic cortical microcircuits. In Computational Neuroscience: A Comprehensive Approach, J. Feng, editor, chapter 18, pages 575-605, Boca Raton, 2004.

[6] W. Maass, "Computation with spiking neurons," in The Handbook of Brain Theory and Neural Networks (M. A. Arbib, ed.), pp. 1080–1083, 2 ed., 2003.

[7] W. Maass and H. Markram, "On the computational power of recurrent circuits of spiking neurons," Journal of Computer and System Sciences, vol. 69, no. 4, pp. 593–616, 2004.

[8] W. Maass, "Liquid computing," in Computability in Europe 2007 - CiE'07, Springer (Berlin), 2007.

[9] "PCSIM: A Parallel neural Circuit SIMulator." http://www.lsm.tugraz.at/pcsim/.

[10] "What is SNNS?" http://www.ra.cs.uni-tuebingen.de/SNNS/announce.html.

[11] "Python Programming Language – Official Website." http://www.python.org/.

[12] W. Maass, T. Natschlaeger, and H. Markram, "Real-time Computing without stable states: A New Framework for Neural Computation Based on Perturbations," Neural Computation, vol. 14, no. 11, pp. 2531–2560, 2002.

[13] G. M. Wojcik and J. A. Garcia-Lazaro, "Analysis of the neural hypercolumn in parallel pcsim simulations," Procedia Computer Science, vol. 1, no. 1, pp. 845-854, 2010.

[14] G. M. Wojcik, "Self-organising criticality in the simulated models of the rat cortical microcircuits," Neurocomputing, no. 79, pp. 61-67, 2012.

[15] G. M. Wojcik, "Electrical parameters influence on the dynamics of the hodgkin-huxley liquid state machine," Neurocomputing, no. 79, pp. 68-78, 2012.