Quantized Liquid State Machines: Design and Implementation

Johannes Maurin Voshol
Faculty of Science
University of Antwerp
Antwerp, Belgium
johannes.voshol@student.uantwerpen.be

***Abstract*—paper presents a Liquid State Machine using a liquid consisting of quantized neurons that are operating on lower-bit representations and fixed point computations. It provides a next step towards the implementation of efficient accelerators that can be used in the field of neuromorphic computing. A minimal implementation of the liquid is realized by only using a neuron potential vector, weight matrix, a threshold value and a leak function. These components all make use of the lower-bit representation of the neurons. The liquid dynamics are not event-driven, but simulated using a clock function. The Liquid State Machine is tasked to predict a chaotic Mackey-Glass time series as to compare various parameters in terms of accuracy and efficiency. Parameters include the amount of bits used to represent a neuron in the liquid, the liquid's size and the influence of different encodings to represent the time series. The accuracy is measured by the minimal validation loss of the readout layer and the efficiency is measured by the total amount of generated spikes by the liquid. A liquid using 16 bit representations is used as a baseline, as fractions of its values are closer to floating point representations. The results show a 50% improvement in efficiency at the cost of a 2% increase in validation loss for liquids using only 2 bits. The best liquid in terms of accuracy is provided using only 3 bits, with an 8% improvement in efficiency and a 5% decrease in validation loss. Moreover, the simplicity of the model makes it easy to find parameters to balance the trade-off between accuracy and efficiency.**

***Index Terms*—neuromorphic engineering, liquid state machine, quantization, time series prediction**

I. Introduction

In machine learning, Recurrent Neural Networks (RNN) have been studied extensively for processing temporal problems, such as time-series prediction, system control or identification, vision and speech [1]. However, training a RNN is difficult as the recurrent property leads to exploding and vanishing gradient problems that get worse with long sequences.

Another approach to these problems is Reservoir Computing (RC), where a nonlinear dynamic random recurrent network (the reservoir) is initialized and left untrained. The state of this reservoir is a high-dimensional representation of the processed input sequence and can effectively be used for pattern analysis by doing classification or linear regression. There are two main types of reservoir computing. First, the Echo State Network (ESN) is generally implemented as a conventional recurrent network, where the dynamics operate on the edge of stability by setting the sparsity of the connections via the spectral radius [2]. The Liquid State Machine (LSM) is another type of RC model implemented as spiking networks [3]. The biologically inspired nodes in the reservoir of the LSM work with temporal signals in continuous time. The asynchronous properties of spiking neurons can improve the computational performance and efficiency of the networks [4].

Neuromorphic computing, often called the third generation of AI, is a relatively new but already major research field. Its goal is to design and evaluate low powered accelerators to broaden the applicability of new deep learning algorithms based on spiking networks. This is nontrivial, as the evaluation of brain-inspired algorithms on traditional (synchronous) computers, often called the Von Neumann architecture, require large amounts of time and power because of the memory bottleneck. Therefore, neuromorphic accelerators try to distribute the memory over the architecture keeping it in close proximity to the computing elements [4]. When embedding Spiking Neural Networks (SNN) into these architectures, there is a need to memorize the weights and other parameters of the spiking neuron. Moreover, for the network to learn a task using a learning rule like Spike-Timing-Dependent Plasticity (STDP) [5], the parameters should also be adjustable.

It was shown that Quantized Neural Networks (QNN) using lower-bit quantized weights and activations could achieve prediction accuracies comparable to their 32-bit counterparts [6]. It allows for replacing the floating point computations by fixed point computations which drastically reduces memory size, memory accesses and power consumption. Parameter quantization of spiking neurons could simplify the hardware architecture, as only the fixed point operations have to be built into the circuit. In the case of an LSM, the weights are static and left untrained, which makes quantization even more appealing if the parameters can directly be embedded into the circuits, making a highly efficient reservoir.

For this paper, a simple spiking neuron reservoir (liquid) is implemented and evaluated with Python code to get insight on its possibilities and limitations. The term 'simple' or 'simplified' is describing the question whether fancy state-of-the-art techniques for spiking neurons (e.g. refraction or delayed synaptic connections) are required to perform adequately on simple tasks. Various lower-bit representations will be compared. This is done in combination with a small range of other parameters present in the model. The task of the LSM is to do regression and predict the chaotic time series generated

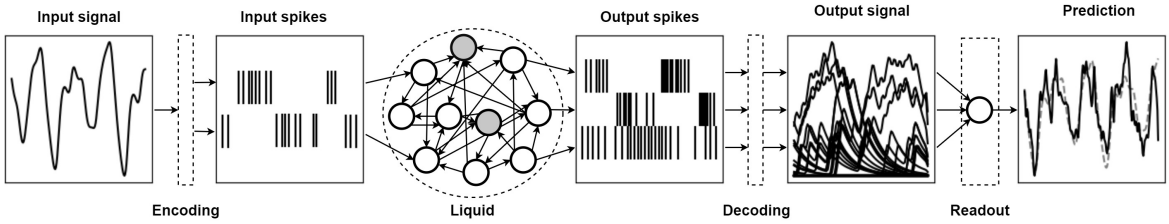

Fig. 1. Data flow of a Liquid State Machine.

from the Mackey-Glass differential equation [7].

The results of this research must provide answers (or a good starting point) to the questions: What is the minimal set of parameters required to design an LSM, and which of those parameters can be simplified while taking the accuracy of the model into account? Specifically, can some of the parameters like weights and neuron potentials be quantized while maintaining adequate accuracy for the considered task? If this last question can be answered with 'yes', then it offers potential to repeat parts of the experiments on a physical accelerator.

Section II describes the formal design choices of the simplified LSM, along with the hyper-parameters that are being used. It provides a complete description on the dynamics of the liquids used in this research. The dataset and the metrics that are being used to evaluate the liquid are described in Section III. The experiment setup and the obtained experiment results are interpreted and discussed in this section as well. Finally, the paper is concluded in Section IV, where we will look back at the main questions and answer them based on the results found in Section III.

II. Methods

The physical hardware is not required to analyze the accuracy of the simplified model. We can simulate the asynchronous neuromorphic model on a synchronous computer to draw conclusions on the properties before designing the accelerators. However, this implementation uses a clock to transition to the next state. This means that the firing of neurons happen in discrete time-steps, leading to different results compared to the more advanced event-driven implementation of other neuromorphic systems.

A. General Framework

There are many components in an LSM that must be taken into account besides the dynamics of the reservoir. Figure 1 visualizes the complete data flow of the model to be implemented. First, the input signal x^T is encoded to a neural representation u consisting of timed spikes that captures the (analog) information of the input signal. The details of the spike encoding are further discussed in section II-C. However, this encoding step in not necessarily part of the LSM itself as research is being done on neuromorphic sensors where the data already has a spike representation [8]. The input spikes are fed to the liquid in sequence while the output spikes are being recorded. The output is a history-dependent representation of the input and must first be decoded to an analog signal to work with the linear (memory-less) readout layer that makes predictions. Since the output signal is a high-dimensional representation, a single fully connected layer is sufficient to do regression.

There are multiple advantages of creating a pipeline where each step is applied on the complete dataset, instead of applying all the steps sequentially on each value in the time-series. First, the input signals and its encoded input spikes are static over all the experiments using that dataset. It is therefore efficient to calculate these spikes only once. Second, if the readout layer were to train on the data sequentially with a high learning rate, the readout layer might forget previous learned regression. This is due to the output data of the liquid being very correlated within a given time frame. Replay is implemented by collecting all the decoded output spikes prior to training and let readout layer to learn on shuffled batches. This can improve the stability and accuracy of the readout layer.

B. Liquid dynamics

$$\begin{aligned} v(0) &= \mathbf{0} \\ v(t) &= W_{in}^{(b)}\, u(t) + W^{(b)} s(t) + v'(t) \end{aligned}$$

The vector v represents the current membrane potential of all the neurons present in the liquid and starts with 0 potential. Neurons spike whenever the membrane potentials exceed the threshold δ_b. The binary vector s represent the spikes following from the current membrane potential and vector v' represents the new membrane potentials after the spikes are extracted and the spiking neurons are reset to 0 potential. Over the neurons that do not spike a leak function is applied to reduce its membrane potential.

The external input spikes of the time series are represented by u. Matrices $W_i n^{(b)}$ and $W^{(b)}$ represent the weighted connections used to add u to the liquid and direct the internal spikes of the liquid. To calculate the next membrane potential, $v(t-1)$ is split into $s(t)$ and $v'(t)$ that are then combined to calculate $v(t)$.

$$s_i(t) = \begin{cases} 1 & \text{if } v_i(t-1) > \delta_b \\ 0 & \text{otherwise} \end{cases}$$

$$v'_i(t) = \begin{cases} 0 & \text{if } v_i(t-1) > \delta_b \\ \text{leak}(v_i(t-1)) & \text{otherwise} \end{cases}$$

1) Quantized neurons: For this study, the neurons in the liquid are quantized in their membrane potential as well as their connection weights. They are represented with natural numbers from a predefined set. The reasoning is that these representations further simplify the implementation of the hardware architecture. To simplify the system even further, the quantized neurons only have access to b bits to represent their membrane potentials and weights. With this, the floating point fractions are completely removed from the liquid.

$$\mathbf{w}_b = \{0, 1, \ ... \ , 2^b - 1\} \quad \text{(for } b \text{ bits)}$$

The neurons are connected using values from a discrete set of weights $\mathbf{w}_b$, that contains all the values that b bits can represent. The values in the set are evenly distributed and the probability of selecting the a weight for each connection is the same. Therefore, the average strength of a connection is therefore 2^{b-1}. As seen in Equation II-B, a spike is generated whenever the threshold δ_b of a neuron is exceeded. Its membrane potential is then set back to 0. In this implementation, the threshold value was deliberately set to $2^b - 1$. It allows for a neuron to spike whenever an integer overflow occurs after their membrane potential is increased. From here on, a liquid constructed using b bits will be referred to as a b-liquid. As the neuron potentials cannot be represented using floating point numbers, a different representation must be implemented to simulate the exponentially decreasing leak. A simple subtraction is preferred over a division. The b-liquids in this research use a bit-wise leak based on the current neuron potential. This operation does not seem efficient, as the computation of a log and an exponent operation is required. However, this can be easily implemented. Using bit-wise operations, the function finds the highest 2-power in the liquid and divides this by 2^λ. This value can then be subtracted from the potential. Moreover, calculating the leak this way guarantees that the value will always be of the form $10..0_2$ (a 1 followed by zeros) making it an easy operation to subtract from the potential.

$$\text{leak}(x) = x - max(1, \ 2^{floor(log_2(x))-\lambda})$$

Using this approach, the leak will be the same for the potentials $2^k - 1$ and 2^{k-1}, resulting in an effective dynamic leak in the ranges between 0.75 and 0.875. This implication was not proven to be a problem during the experiments. As the liquid uses only natural numbers, the leak has to be a minimum of 1, hence the max operation. This changes the behavior of the leak per value of b. Neurons using large values for b do not leak their potential slower, but rather have more time to leak very small fractions of their potential in later time steps, whereas neurons using small values for b have relative larger step sizes in their leak and reach a 0 potential relative faster. This is illustrated in Figure 2 with the use of a log scale. The dynamic leak can also be noticed, as the potential is not decreasing in equal steps. However, the bit-wise leak does approximate the exponentially leak well as the graph looks linear from the perspective of a log scale. It

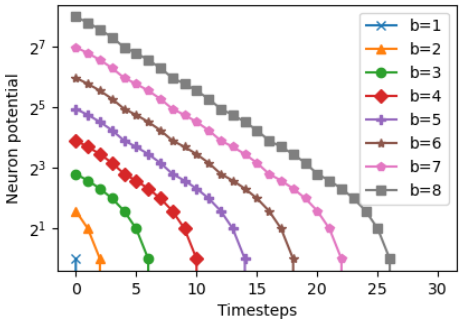

Fig. 2. Bit-wise leak of neurons using b amount of bits.

shows that for a Leaky Integrate Fire (LIF) using lower-bit representations for potential, threshold and leak can still be effectively implemented.

2) Neuron connections: Signals between neurons may be excitatory, where one neuron stimulates another neuron to also fire, or can be inhibitory, which causes the neuron to repress another neuron. This can easily be simulated by setting positive or negative weights for the excitatory and inhibitory connections respectively. $N = 1, 2..., n$ denotes the set of indices of the vectors and matrices that corresponds with all the neurons present in the liquid. The set corresponding with excitatory- and inhibitory neurons is captured in E and I respectively, where $N - E = I$. The balance of excitatory- and inhibitory neurons is important. Using too few inhibitory neurons may cause the liquid to spiral out of control as there is too much energy (potential of the membranes) in the system. This results in the loss of temporal information of recent inputs. On the other hand, temporal information is also lost when setting the amount of inhibitory neurons too high, as it leads to a liquid where neuron potentials easily fall flat. A right balance is achieved by setting the ratio of inhibitory/excitatory neurons to $1/4$.

The matrices $W_{in}^{(b)}$ and $W^{(b)}$ represent the the connections from and to the neurons in the liquid as well as define the weights of those connections. Whenever a connection between two neurons is added, its weight is sampled uniformly from $\mathbf{w}_b$. It is a static parameter but serves an essential role in the dynamics of the liquid described in Equation II-B.

The sparse matrix $W_{in}^{(b)}$ of size $|N| \times |U|$ adds the input u to the membrane potential. Here, U depicts the set of indices for the input channels. One input channel is connected to one or more neurons in the liquid. Its sparsity can be regulated by specifying α, the percentage of neurons one input channel is allowed to connect to. It is common practice to input only to excitatory neurons, hence the 0 for connections to inhibitory neurons.

$W^{(b)}$ is an $|N| \times |N|$ sparse connection matrix for internal neurons in the liquid. To visualize the separation of the incoming and outgoing connections for excitatory and inhibitory neurons, two lines have been added to the matrix. The amount of outgoing connections of a neuron and whether it should connect to an excitatory or inhibitory neuron is

described by the constant **c**, which is a tuple that consists of 4 values indicating the amount of random connections from excitatory/inhibitory to excitatory/inhibitory neurons.

$$W_{in}^{(b)} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,|U|} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,|U|} \\ \vdots & \vdots & \ddots & \vdots \\ w_{|N|,1} & w_{|N|,2} & \cdots & w_{|N|,|U|} \end{bmatrix}$$

$$\begin{array}{llll} \forall i \in E, & \forall j \in U & : & w_{ij} \in \mathbf{w}_b \quad \text{(excitatory)} \\ \forall i \in I, & \forall j \in U & : & w_{ij} = 0 \quad \text{(inhibitory)} \end{array}$$

$$W^{(b)} = \left[\begin{array}{cccc|cc} w_{1,1} & w_{1,2} & \cdots & w_{1,|E|} & \cdots & w_{1,|N|} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,|E|} & \cdots & w_{2,|N|} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{|E|,1} & w_{|E|,2} & \cdots & w_{|E|,|E|} & \cdots & w_{|E|,|N|} \\ \hline \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{|N|,1} & w_{|N|,2} & \cdots & w_{|N|,|E|} & \cdots & w_{|N|,|N|} \end{array}\right]$$

$$\begin{array}{llll} \forall i \in N, & \forall j \in E & : & w_{ij} \in \mathbf{w}_b \quad \text{(excitatory)} \\ \forall i \in N, & \forall j \in I & : & -w_{ij} \in \mathbf{w}_b \quad \text{(inhibitory)} \end{array}$$

C. Spike coding

1) Encoding: For the liquid to process our time series to a higher dimensional output, the signal must be represented as spikes. There are two types of encoding considered in this paper: temporal encoding and rate encoding. Other types of encoding like population and sparse coding are not considered due to the scope of the project. Rate encoding relies on the actual value of the signal. Spikes are generated at times steps when the value of the signal is also high. One problem with this type of encoding is that it takes many (unnecessary) spikes to represent a signal that is high for longer periods of time. However, it may be easily implemented by feeding the signal directly to a layer of spiking neurons that have a threshold adjusted to the input signal. We will call this encoding Spiking Neuron Encoding (SNE). Temporal encoding is the second type of encoding. This technique is more popular for representing signals because of their efficiency. The spikes are only generated whenever the signal changes in value, hence the term temporal. In other words, temporal encoding is effectively still a rate encoding over the derivative of the signal. The property of temporal encodings to transfer required information faster than a rate encoding can be useful. Systems that have to make quick decisions based on changes in the environment benefit most from a temporal encoding. However, a temporal encoding cannot be represented in a single stream of binary spikes. It requires either the use of negative spikes, which does not work for the liquid, or a second stream using positive spikes representing the negative spikes. The implementation of the temporal encoding used in this research is based on the Step-Forward (SF) algorithm [9]. Thresholds for the temporal- and rate encoding algorithms are chosen to yield the same average amount of spikes for a time series. Besides this, the percentage of neurons to input the spikes to are also chosen to provide good liquid dynamics and is represented with α. When using the encodings separately, this value is set to 20% of the neurons in the liquid. This value is decreased to 10% when the encodings are used in combination, as double the amount of spikes are presented to the liquid.

2) Decoding: One way to represent the state of the liquid x^M could be to take the membrane potential v, but then the spatio-temporal patterns of the liquid are not represented. To represent the liquid state, it is better to take the sum of all the spikes s over the last τ time-steps into account.

$$x_i^M(t) = \sum_{n=0}^{\tau} \gamma^n s_i(t-n) \qquad \forall i \in E$$

By definition this is an exponentially decreasing rate decoding using a sliding window. Following the literature, the state of the liquid will be only based on the output of excitatory neurons. It is important for the discount γ not to be too small, as it possibly flattens older values in the window to 0, making part of the sliding window unusable. When setting γ too high in combination with a large window size, recent spikes hardly affect the liquid state. This causes the decoder to react too late to recent information provided by the liquid and complicates the learning process of the readout layer. The parameters τ and γ are balanced to optimize the memory size of the stored data (e.g. $\tau \leq 50$) and its containment of information (e.g. $\gamma^{\tau-1} \geq 0.1$), which includes adjusting τ to the pace in which the temporal data is presented and processed. A time series that is highly volatile in a short time frame requires the decoder to react fast to the changes.

D. Readout

After initialization of the liquid, the readout is the only component of the LSM with trainable parameters. It consists of a single fully connected layer f_θ to do regression or classification. The readout does not have to be be any deeper, as the output of the liquid is already a high dimensional representation of the processed input. The task in Section III will be to predict a time series k time steps ahead. In this notation, x and y present the continuous signals of the (to-be predicted) time series T and model representation M.

$$x^T(t+k) = y^T(t) \approx \hat{y}^M(t) = f_\theta(x^M(t))$$

To train the readout, the mean squared error (MSE) is generally used as the loss function. The mean absolute error (MAE) was considered, but did not yield better results given our prediction task. If our regression task would be converted to a classification task where predicting the exact peaks of the time series is not necessary, then MAE could possibly serve useful. The network was trained using the stochastic optimizer Adam [10].

$$\mathcal{L}(y^T, \hat{y}^M) = \frac{1}{n}\sum_{i=0}^{n}(y_i^T - \hat{y}_i^M)^2 \quad \text{(MSE)}$$

E. Hyper-parameters

1) Static parameters: Up until this point, many variables and terms were introduced that can be seen as hyper-parameters of our system. Some of the parameters can be set to a specific values that have been proved to be good values according to the literature. This includes the excitatory/inhibitory ratio and the neuron connections $\mathbf{c}$. Other parameters like the neuron threshold, weights and leak have been successfully rewritten in terms of b bits or other as a small sets of integers which reduces the hyper-parameter space drastically. The one important hyper-parameter of the liquid will thus be the the amount of bits available to the neurons. It is also important to experiment how the input signal can best be represented to the readout by selecting the best encoder, input ratio and amount of neurons for the liquid. Figure 3 shows the table of static- and experimental parameters.

Static parameters		
Parameter	Value	Description
$\|E\|/\|N\|$	80%	Excitatory/inhibitory ratio
δ_b	$2^b - 1$	Neuron threshold
λ	2	Leak exponent
τ	50	Sliding window size
$\gamma^{\tau-1}$	0.02	Sliding window leak
$\mathbf{c}$	$(2, 2, 1, 1)$	Neuron connections
k	+20	Prediction task
$(\tau_{\text{MG}}, a, b,$ $n, x_0, h)$	$(17, 0.2, 0.1,$ $10, 1.2, 1)$	Mackey-Glass parameters
Experimental parameters		
Parameter	Set	Description
$\|N\|$	$\{50, .., 2000\}$	Liquid size
b	$\{1, 2, 3, 4, 6, 8, 16\}$	Bits
α	$\{10\%, 20\%, 30\%\}$	Neurons per input channel
Encoder	{SF, SNE, SF+SNE}	Signal encoding

Fig. 3. Parameters that will be kept the same across experiments and parameters that will be compared during the experiments. Note that the parameters of the readout layer are not compared due to the scope of the project.

F. Implementation details

The LSM described in the sections above was fully implemented in Python with use of the `NumPy` and `PyTorch` libraries. They allow for high level vector and matrix computations that are also highly optimized in c++ in the backend. Moreover, the `PyTorch` library makes it possible to train the readout layer on a GPU. In most cases, running the liquid took more computational time than training the readout layer, especially for simulating large liquids. Future improvements might include switching to `CuPy`, which is an GPU accelerated version of `NumPy`. As previously stated, this liquid implementation is not event-driven but works with a tick function. The classes in the repository implement each of the steps of the data flow.

III. Results and Discussion

A. Dataset

The time series used for the experiments is based on the Mackey-Glass differential equation [7]. To provide a simple overview, the derivative at time step t of the time series is based on the actual value at time step $t - \tau_{\text{MG}}$. The time series looks very chaotic, but if the liquid can manage to memorize the values from the past, the readout will be able to predict the next value. For all experiments, a time series is generated with a length of 5000. This length makes sure that it does not take too much time to run the liquid but still provide enough samples for the readout to learn from.

The encoded time series is run though the liquid and then decoded. This bulk of data is then split into three separate sets. First, a (small) test set is taken from the final k time steps of the liquid output. The remaining data is randomly split into a training (80%) and validation (20%) set. The readout layer will train on the training set while evaluating on the validation set after every epoch. The samples of the validation set are different from the training set, but are still highly correlated. Overfitting should be avoided by monitoring the training process, keeping the validation and training loss close over time. For the test set, this correlation will be less as it is captured in the time steps after the training set. It can thus be used to test our predictions on the final output of the liquid. These are the actual predictions that are most interesting.

B. Metrics

To evaluate the b-liquids in terms of accuracy and efficiency, two metrics are needed. As shown in Section II-D, the minimal loss for the validation set is used to measure the accuracy of our model. As the hyper-parameters of the readout layer are static, it measures the quality of our liquid indirectly. This loss may vary across experiments, but the relative loss within the experiments still show the best parameters for the liquid. Measuring the activity of the liquid (in combination with the accuracy) is important, as it shows efficiency of the model. Moreover, a low activity also means that the actual power consumption of the physical accelerator will be lower. The activity is calculated by summing up the total spikes s in the liquid for each time step. It can be displayed as the absolute activity representing the power consumption of the model. It can also be displayed as the relative activity compared to the size of the liquid. Another metric is the total energy of the system, which is calculated by averaging the neuron potentials of v over all the time steps. It does not necessarily display the efficiency of our model, but it can still provide insights on the dynamics of the liquid.

$$\begin{aligned} Activity^M(t) &= \tfrac{1}{|N|} \cdot s(t) \cdot \mathbf{1} \\ Energy^M(t) &= \tfrac{1}{\delta_b \cdot |N|} \cdot v(t) \cdot \mathbf{1} \end{aligned}$$

C. Experimental results

Various liquid sizes are compared over the the same input signal in Figure 4. The liquid must be large enough provide the readout layer with a rich representation of the processed data. For this specific task, increasing the liquid size from a low 50 to 500 improves the accuracy drastically. Whenever the liquid has a size larger than 700, the accuracy is getting worse, further reducing the efficiency. The 2-liquid is not under-performing, but increasing the neuron representations to 3 bits drastically improves the accuracy. On the other hand,

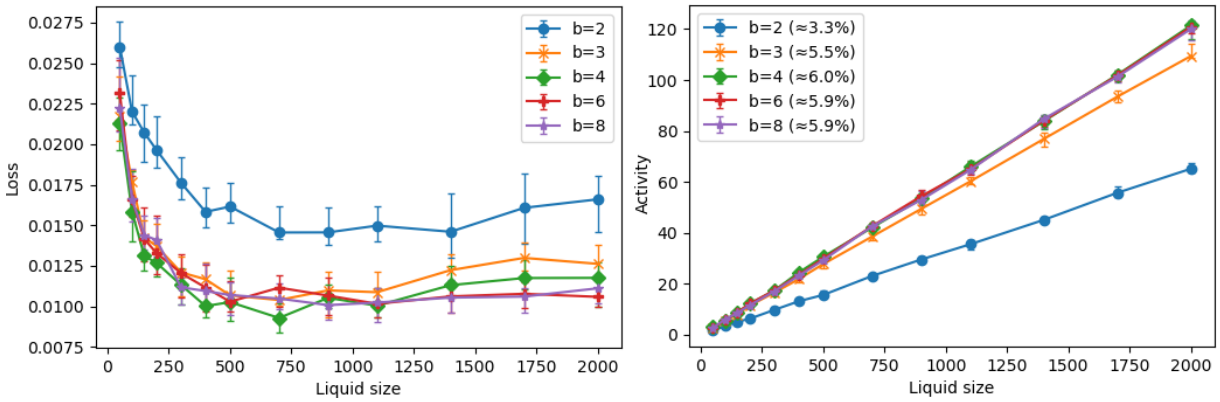

Fig. 4. Finding the optimal liquid size $|N|$ and visualizing the activity for various b-liquids tasked to predict a signal using only a temporal encoding and $\alpha = 30\%$. Each data point represents a median and quartiles over 30 samples.

Encoder	SF	SNE	SF+SNE
α	20%	20%	10%
avg u	0.356	0.354	0.710
b	**Relative Activity**		
2	0.7%	0.3%	1.4%
3	1.6%	1.4%	3.2%
4	1.3%	1.7%	3.2%
6	1.3%	1.6%	3.5%
8	1.6%	1.4%	3.7%
16	1.6%	1.5%	3.5%
b	**Accuracy ($\mathcal{L} \cdot 10^4$)**		
2	233	157	37
3	93	101	26
4	80	82	24
6	78	61	26
8	57	70	26
16	56	69	25

Fig. 5. Activity and accuracy of various b-liquids ($|N| = 600$). Here, α is chosen such that the inputted energy to the liquid is the same across experiments. The average u represents the average amount of spikes inputted to the liquid at each time step. The results are the averages over 10 randomly generated liquids.

b	**Average** $\mathcal{L} \cdot 10^4$	**Best** $\mathcal{L} \cdot 10^4$	**Absolute Activity**	**Epochs** $\mathcal{L} \cdot 10^4 < 30$
16	26.3	21.5	43.7	316
2	26.2 (-0.4%)	22.3 (+3.7%)	20.9 (-52%)	139 (-56%)
3	24.3 (-7.6%)	20.4 (-5.1%)	39.9 (-8.7%)	239 (-24%)
6	25.4 (-3.4%)	22.0 (+2.3%)	43.2 (-1.1%)	284 (-10%)

Fig. 6. Accuracy and activity of various b-liquids tasked to predict a Mackey-Glass timeseries. The results are the averages over 20 randomly generated liquids ($|N| = 500$, $\lambda = 2$, $|T| = 5000$, $\alpha = 10\%$, $y^T(t) = x^T(t + 20)$, $lr = 0.0005, epochs = 800$)

the 2-liquid is almost 2 times more efficient compared to the other b-liquids. The trade-off between accuracy and efficiency for 2 and 3 bits is clearly noticed in this experiment. Figure 5 further illustrates this trade-off. It also shows that combining a rate- and temporal encoding drastically improves the accuracy at the cost of efficiency. In the other experiments, four models are compared. First, 2/3-liquids are considered as the trade-off could already be seen in previous observations. The results for a 6-liquid are shown as this model often provided the best accuracy. Lastly, floating point weights are not possible to measure in this implemented system. Therefore, a 16-liquid is used to mimic the fractional properties of floating point values. In Figure 6, a 16-liquid is used as a baseline. All the lower-bit liquids provide better accuracies on average. Moreover, the best accuracy of all the models is provided by a 3-liquid. Its validation loss is 5% lower compared to the best 16-liquid whilst having a lower activity.

The intermediate states of the LSM are visualized in Figure 7. It shows the last 300 time steps of the time series, where the last 150 time steps are part of the test set. The following observations can be made: (6a) For 3-liquids and below, the fast leak is significantly present in the liquid. The energy levels are more volatile and even drop to 0 from time to time, which means that there is a significant loss of information (memory reset) at those time steps. (6b) All 3-liquids and above share the same density in their output spikes. (6c) Using a 2-liquid, a lot of neurons do never spike as illustrated in the (almost empty) decoded output. (6e) A low learning rate is applied during the training phase to reduce the variance. The epochs are therefore increased. This way, the plateaus can be clearly shown. The 2-liquid provides less complicated data for the readout layer and with that, makes the training much faster. However, the learning curve also plateaus faster. The loss for a 16-liquid descents slower as the decoded spikes are more complicated because the set of discrete values is much larger.

D. Discussion

Whenever a single bit is used to represent the neurons in the liquid, the binary spikes between the neurons are not strong enough to push neuron potentials above the threshold before leaking back from 1 to 0. The provided results are thus not containing measurements for $b = 1$. Running this liquid without a leak results in a high activity liquid that does under-perform compared to the other liquids. However, it has an activity that is about 20 times the activity of the other liquids.

The weight matrices of the liquid are randomly generated based on the outgoing connections specified by $\mathbf{c}$. In this paper, neurons are not verified whether they have at least one incoming connection. When using the value $(2, 2, 1, 1)$, this always results in roughly 5% of neurons not having any

incoming connections from either liquid neurons and inputted spikes. These neurons are considered dead neurons. This ought to be improved to reduce the unnecessary waste in memory.

There are a few hyper-parameters that must be chosen to construct a b-liquid. The main strategy of choosing these parameters would be to first select them at random with a large enough liquid size. Then work from front to end in the LSM selecting the best parameters in the order: encoding, liquid size, b value and then decoding. The relative difference in accuracy stays the same when testing parameters. For example, changing the b values does not change the optimal liquid size by much. This is also true for the amount of epochs required to train the readout layer. Parameters can easily be tested on a subset of the time series where the readout is trained for a short time. Moreover, when the readout is provided with a rich representation of the input signal, implicating that the liquid dynamics are well adjusted, the gradient descent will be faster.

Using multiple encodings per input signal will always provide the best results, as the information of the signal is better captured. However, this goes at the cost of efficiency as a larger liquid size is required. If the task of the model is to react to changes in the data, a single temporal encoding would suffice.

Selecting the minimal amount of bits depends heavily on the task of the model. A 2-liquid is not the best performing model in terms of accuracy, but it is the best performing model in efficiency. At the cost of one bit, the increase in accuracy using a 3-liquid is significant, although its efficiency decreases. All the 4-liquids and above share the same accuracy. The 6-liquid performed best in these experiments, but the 2 extra bits for a slight accuracy increase might not be worth it.

Changing the leak exponent λ to other values did not change the accuracy by much. This is always true for the lower-bit neurons, as the minimum leak stays 1 for all values of λ. For the higher-bit neurons, a higher leak (lower λ) resulted in less activity, but not in better accuracies. The implementation of the leak could also be further adjusted to better capture the exponential decrease of lower bit rates. Slowing down the leak for lower-bit weights could be achieved by only leaking every n time steps after the last spike was received. This would however bring more complexity to the implementation.

For simplicity and reduction of the hyper-parameter space, the threshold was expressed in terms of parameter b. This parameter was set to be equal to the maximum strength of the weights. This is unlike the values used in the literature as one spike should not always lead for another neuron to spike. This parameter could be further researched by setting it to an arbitrary value or, to keep the simple overflow model, to a value double the size ($\delta_b = 2^{b+1} - 1$).

IV. Conclusion

A liquid and its dynamics play an important role in a LSM. It can be represented and effectively utilized in a simple manner by using a neuron potential vector, weight matrix, threshold and a leak function. All without the use of other techniques like refraction or delayed synaptic connections. Moreover, it was shown that the neurons in the liquid can effectively be quantized. First, the state of a neuron can be represented only using integer values represented by b bits. Its dynamics can be implemented only using fixed-point computations. This drastically reduces memory size and power consumption. Second, the results showed that all the lower-bit liquids using these type of neurons performed well in terms of accuracy and activity compared to a 16-liquid baseline. Lastly, the simplicity of the model looks promising and provides opportunities for other downstream tasks. Its parameters can easily be considered and adjusted to balance the trade-off between accuracy and efficiency. Further research could demonstrate the implications of using an event-driven b-liquid, taking the next step towards the implementation of efficient liquid accelerators.

Supplementary materials

The Python implementation and Jupyter noteboook can both be found at: `github.com/m4urin/SimpleLSM`

Acknowledgment

I am thanking Werner Van Leekwijck and Tsang Ing Jyh for their exceptional support during the semester and the reviewing of this paper. Writing this research paper would not have been possible without the many weekly meetings, where they provided knowledge, insight and expertise on many topics in the field of neuromorphic computing.

I am also grateful for the helpful comments offered by my good friend, Matthias Tavasszy.

References

[1] Schrauwen, Benjamin, David Verstraeten, and Jan Van Campenhout. "An overview of reservoir computing: theory, applications and implementations." Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007. 2007.

[2] Jaeger, Herbert. "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note." Bonn, Germany: German National Research Center for Information Technology GMD Technical Report 148.34 (2001): 13.

[3] Maass, Wolfgang. "Liquid state machines: motivation, theory, and applications." Computability in context: computation and logic in the real world (2011): 275-296.

[4] Bouvier, Maxence, et al. "Spiking neural networks hardware implementations and challenges: A survey." ACM Journal on Emerging Technologies in Computing Systems (JETC) 15.2 (2019): 1-35.

[5] Grüning, André, and Sander M. Bohte. "Spiking neural networks: Principles and challenges." ESANN. 2014.

[6] Hubara, Itay, et al. "Quantized neural networks: Training neural networks with low precision weights and activations." The Journal of Machine Learning Research 18.1 (2017): 6869-6898.

[7] Mackey, Michael C., and Leon Glass. "Oscillation and chaos in physiological control systems." Science 197.4300 (1977): 287-289.

[8] Dupeyroux, Julien. "A toolbox for neuromorphic sensing in robotics." arXiv preprint arXiv:2103.02751 (2021).

[9] B. Petro, N. Kasabov and R. M. Kiss, "Selection and Optimization of Temporal Spike Encoding Methods for Spiking Neural Networks," in IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 2, pp. 358-370, Feb. 2020, doi: 10.1109/TNNLS.2019.2906158.

[10] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

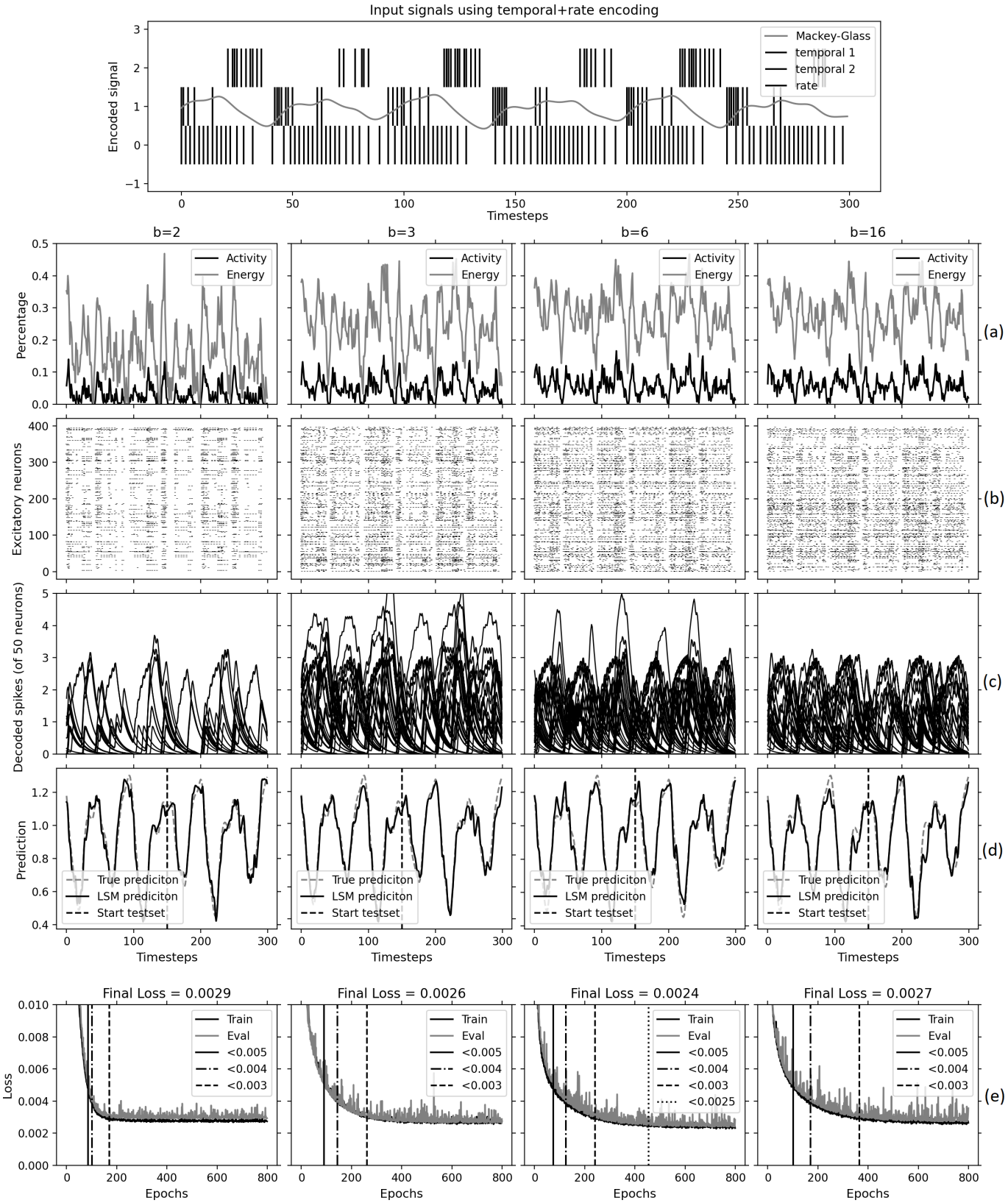

Fig. 7. Visualization of the intermediate LSM stages. Four b-liquids are tasked to predict a Mackey-Glass timeseries. In this experiment, the parameters are the same as Figure 6.