Quantifying the Effect of Learning on Recurrent Spiking Neurons

Nicolas Brodu

Abstract—**This work is concerned with measuring what is the response of recurrent spiking neurons when a learning rule is applied to them, in a Liquid State Machine context. Two indicators are considered for monitoring on-line the effect of learning: the separation property, which has already been studied in previous works, and an incremental version of the statistical complexity measure that is introduced expressly for our needs. It is found that while separation increases, a neuron's average statistical complexity decreases when a learning rule is applied. This means that neurons become more predictable and their behavior is simplified as an effect of learning. A key feature of this work is to provide a quantification of this phenomenon.**

I. Introduction

In order to understand why learning is effective or not in a spiking neural network, it is important to rely on objective and quantitative indicators of the system state. Study of these indicators and their evolution during the learning phase then provides insight into the learning process.

A first indicator that was proposed together with the definition of the Liquid State Machine [1] is the separation property. The idea is to measure the capabilities of the recurrent layer (see Fig. 1) to produce discernibly different responses for different inputs. If the system is too static many inputs are mapped together in a small number of cases, their trajectories in the system state space are merged and information is lost. This corresponds to low separation. On the other hand, when the network is too random, the response it gives to different inputs is not statistically significantly different, and it is not possible to distinguish between the inputs. Therefore, separation was proposed as a way to quantify when the system has maximal processing capabilities on the inputs [2]. A study of the effect of Hebbian learning [3] on the separation indicator was presented in [4]. The current work confirms their finding independently, that separation increases while learning, and extends this result to the multifractal learning rule that was introduced in [5].

Yet, while separation is well understood, it strongly depends on the target task. Separation is defined and measured as the capacity of the recurrent layer to produce signals that can be separated by the linear classifier layer (see Fig. 1). An objective property of the recurrent neurons behavior is thus needed if we want to assert the intrinsic effect of the learning algorithm: Learning is applied on the recurrent layer and its effect shall ideally not depend on how we define the inputs and outputs.

A second indicator is therefore used in this paper: the Grassberger-Crutchfield-Young statistical complexity [7] of individual neurons. The idea is to measure the amount of information that is present in the past of a neuron, which is relevant to predict its future behavior. Thus we are measuring

Nicolas Brodu is a PhD candidate at the Department of Computer Science and Software Engineering, Concordia University, Montreal, Quebec, Canada, H3G 1M8 (e-mail: nicolas.brodu@free.fr).

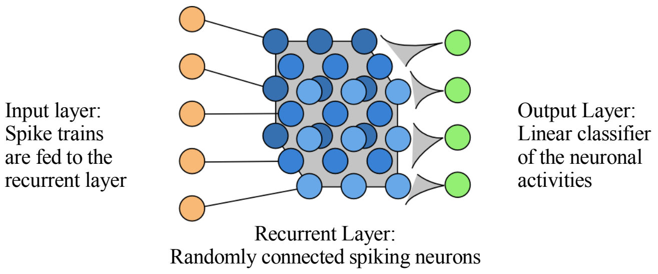

Fig. 1. Liquid State Machine schematic organization.

an intrinsic property of the recurrent neurons, the difficulty to predict their behaviors, independently of the inputs and outputs

The statistical complexity measure has a long history, tracing back to prediction processes, causal states and epsilon machines [6]. Practical algorithms for its application to discrete systems are presented in [8] and [9]. However these algorithms are adapted to fixed data sets, and what we want here is to monitor the evolution of the statistical complexity measure while learning. Therefore, a crucial step of this work is to extend the algorithm presented in [9] so as to make it incremental: The new algorithm can be applied to the neurons while they learn. Section II.E presents this step in details.

Like separation, statistical complexity is low for both highly ordered and highly disordered systems, and reaches a maximum in between [7]. However, it is found in this study that from a given random initial state, whereas separation increases, the statistical complexity decreases while learning. Therefore, the two indicators do not provide the same measure on the system. This result also indicates that while order and chaos considerations may be useful, indicators that reach a maximum between these system states must be interpreted with caution. This point is out of scope of the present document but it is extended in [10].

Part II describes the two indicators: the definition of the separation property is recalled and the way it is applied in the current setup is explained. Statistical complexity is then introduced, together with how it is applied to spiking neurons in this work. The algorithm for estimating incrementally the complexity with live data is then detailed. Part III describes the experimental protocol used to conduct the measurements and interprets the results. The conclusion comes back to the main points of this work.

II. The indicators

A. Separation

Separation seeks to quantify the ability of the network to produce distinct outputs for distinct inputs. This raises two questions: how do we measure that outputs are distinct, and similarly for the inputs. Separation is computed using measurements at the interface between the recurrent layer and the output layer. Since the inputs are not directly available at this point the first step is to represent the state of the network so this state can be related to the inputs. [11] and [4] use regular sampling for this: A neuron state is set to 1 or 0 if it has spiked or not during a given time interval. Results obtained with this definition depend on the sampling frequency. So as to remove this dependency the state of the network is measured in this work by averaging the activity signals of the neurons over time (spike counts per second). An advantage of this definition is that separation is computed in the data space used by the output linear classifier. The duration for averaging the spike counts is set to the exposition time of the inputs to the network. The state S of the network is thus now a vector of frequency values, rather than a vector of boolean values.

Inputs are presented to the network and the corresponding states are collected after the exposition period. [11] and [4] propose classifying these states into categories corresponding to the desired output targets for the inputs: States corresponding to the same expected output are set together. Then, for each output class i, all states S that are expected to match class i are averaged into a class center $C_i = \frac{1}{N}\sum_{j=1}^{N} S_j$ with N the number of states in class i.

Separation is presented in [11] as the average distance between the class centers. The formula is adapted here to respect this definition exactly: For N classes, there are N(N-1)/2 center differences: $Sep = \frac{2}{N(N-1)}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\|C_i - C_j\|$.

In the case there are only two output classes, separation is thus equivalent to the 2-norm of the centers difference.

To sum up: Separation is defined a posteriori using the expected output classes. The average network state for each output class is computed over all input instances mapped to this expected output. Then, the average difference between class centers is called the separation.

This simple definition is well adapted to the study of a particular problem: If the learning rule increases the separation, then it is easier for the output layer linear classifier to distinguish the network states corresponding to the input/output mapping under consideration. Yet when studying the effect of learning rules in general, separation would be more useful if it was applied on a significant sampling of all possible input/output mappings. This way, it would be possible to assert the global effect of the learning rule, whatever the specific problem it is applied to. This global study is however not always possible and a better method is needed to assert the intrinsic effects of a learning rule on the recurrent layer, independently of the input/output mappings. One such method is provided by computing the statistical complexity of the neurons, as is explained in the next section.

B. *Statistical complexity*

Statistical complexity is a very general measure applicable to a wide range of problems, as explained in [6]. The main idea is to quantify the amount of information that is present in the past of a system, which is relevant to predicting its future. For discrete systems in particular statistical complexity is an objective property that can be estimated from observed data [8]

In the context of spiking neurons some information present in the past spike timings can be used to predict the future spikes, while some other information does not help. Consider for example that the neuron is locked into spiking at the highest frequency (just after the refractory period). In this case observing the past of the system gives always the same cyclic pattern. Based only on these past observations the next spike may be predicted accurately. But the information contained in the cyclic pattern is low: the cycle period and the last spike timing are enough to describe the neuron behavior. At the other extreme if the neuron apparently spikes randomly then knowing when it has previously spiked does not help. We could model the occurrence of new spikes with a fixed distribution (ex: Poisson) and ignore the past timings. The information needed to model the neuron behavior is in this case contained in the distribution. If the neuron now spikes sometimes randomly, sometimes produces cyclic activities, and sometimes keeps silent for extended periods of time, it may be better modeled by a state machine with transition probabilities estimated from the observed data. The information needed to describe the neuron behavior is then encoded into this state machine, and it is greater than in the previous two cases that each correspond to one of the machine states.

This short example has introduced two notions: 1. That statistical complexity is low for systems with behaviors that are either too static or too random, while it increases for intricate behaviors. 2. The notion of state machines modeling a system behavior, estimated from the past, and used to predict the future. The first point is what gives statistical complexity its name. The second point is actually how it is defined [6].

The present work reuses the same light-cone approach as explained in [9]. When exploiting the past to predict the future then any observable bit counts, and light-cones precisely include all observable information. The past (resp. future) light cone of the system includes the states of all entities that could have a causal influence on (resp. be influenced by) the present system state. In practice there is a computational power limit as to how far in time the measurements may be carried on. As in [9], the past and future light cones are simply truncated to a maximum time. This is justified in the next section.

Figure 2 shows a schematic presentation of the relations between the past and the future light cones. Note that point P is in the past light cone of the system (the considered neuron in our case), whereas point O is outside the cone and cannot be known by monitoring the history of causally related entities. However, both points may be necessary to correctly predict point F in the future. For a given past cone c, there may thus be several observed future cones f, and the probability distribution p(f|c) captures how well we can predict the different futures when observing a given system past. Conversely, when two pasts c_1 and c_2 lead to the same probability distribution $p(f|c_1) = p(f|c_2)$, then for all purposes with respect to the future of the system it is not possible to statistically distinguish between these two pasts: they are equivalent. The equivalence classes $\varepsilon(c)=\{x: p(f|x)=p(f|c)\}$ are called the Causal States of the system [6]. In turn, the amount of information necessary to encode these causal states (for a discrete system, see [9]) reflects how difficult it is to find ε(c) from c, hence how to get

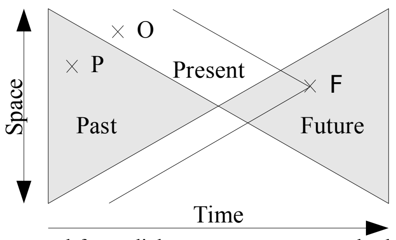

Fig. 2. The past and future light cones are represented schematically on this figure. These cones are the basis of the statistical complexity definition which is detailed in the main text.

the desired p(f|c) for predicting the system future. The total information necessary to encode all the states is the system statistical complexity, $-\Sigma_{\varepsilon}\ p(\varepsilon)\ \log2(p(\varepsilon))$. For a particular ε matching a cone c the local complexity [9] is $-\log2(p(\varepsilon))$.

A more formal approach is provided in [6], this section goal was to present the ideas behind statistical complexity in a intuitive way. The next section relies on this presentation so as to explain how statistical complexity is defined for spiking neurons for the present work. The new algorithm for estimating the causal states incrementally is detailed in Section II.E.

C. Application to spiking neurons

The question is now how to define a light cone for a spiking neuron. In the present LSM study a neuron state is fully determined by the previous spikes that neuron has received and emitted over time. Starting from the same initial point, if the neuron receives the same sequence of spikes, it will have the same final state: no other parameter interferes.

Entities of interest are in our case efferent and afferent neurons. There may be a causal relation between these entities and the considered neuron only beyond the transmission delay (in the past or in the future). Considering that the past spikes of a neuron have an effect on the membrane potential and influence the reaction for the next spike, the neuron is itself a causally linked entity with transmission delay 0. This gives a unified framework for the current neuron together with all its efferent and afferent neurons.

The time to the last (resp. next) spike is then monitored for each entity, counting from the delay point, as described in Fig. 3. The rationale for Fig. 3, A4, is that past a certain time, all spikes have a negligible influence. This approximation is justified by the decaying property of the membrane potential. Another approximation is to retain only the last spike, for all entities. This is justified by considering that previous spikes have a negligible influence compared to the last one.

Thanks to these considerations it is now possible to build the light cones. For a given neuron, consider a vector of real values. Each entry in that vector correspond to one entity related to the current neuron: afferent nodes in the case of past light cones, efferent nodes in the case of future light cones, and including the current neuron itself in both cases. The value for that vector entry is the observed time difference between the last (resp. next) spike and the point in time corresponding to the transmission delay, represented by thick lines on Fig. 3.

D. Difficulties

The algorithm that was presented in [9], and that is extended in the next section, can only process discrete values. But since the simulation relies on discrete time intervals for the numerical integration the observed time differences are already discretized anyway. However for efficiency it may be necessary to consider a coarser time scale for the construction of the light cones. This implies the assumption that close spikes in time produce the same effect, and the spike timing precision for the light cones needs not be the same as for the numerical integration. The simulations carried on for this study retain 16 levels of quantization for each observed time difference in the construction of the light cones.

In [9] all cells in a cellular automaton are produced by the same rule and therefore convey the same statistical distribution

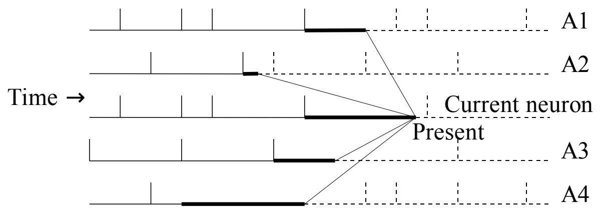

Fig. 3. Past light cone construction for a given neuron. The future light cones may be derived by symmetry. A1 to A4 are afferent neurons, the diagonal lines represent the transmission delays in time. Dashed spikes have no influence on the current neuron state, though some of them have occurred in the past. Thick lines represent the time to the last spike before the delay point. In the case of A4, the maximum time has been reached: all spikes with larger times are mapped to the saturated maximum.

of events. Observations from different cells are thus merged to produce more data and make the algorithm converge faster (and this is reproduced in Fig. 5). In the current study, each neuron is a distinct system, with its own past and future properties. For example, the light cone sizes are dependent on the afferent and efferent neuron connectivities. For this reason, it is not possible to collect and gather results from different neurons to produce the complexity estimates. Fortunately, the incremental version of the algorithm converges fast in practice (in terms of number of data required), so it can still produce reasonable values from the neuron spiking data.

Another related difficulty comes from the learning itself. It is not formally correct to continue adding observations while applying the learning rule at the same time: the rule precisely changes the connection weights, and therefore the neuron is not a stationary system. The assumption is made here that incurred changes are statistically significant only on a large time scale. This approximation allows to justify feeding more observations to a neuron complexity analyzer even while learning. However, for correctness, obsolete data must still be removed. The incremental algorithm can handle the adding and removing of observations equally well for that purpose.

E. Incremental algorithm

The incremental algorithm for estimating causal states from light cones is presented in this section. A C++ reference implementation is available on the author web site at http://nicolas.brodu.free.fr. This implementation is generic (applicable to any user-defined light cone type) and standalone.

As explained in Section II.B. the system past and future light cones are monitored, and a distribution of observed future light cones is built for each past light cone. Let c be a past cone and $d=\{(f,n_f)\}$ an estimated pseudo-distribution over future cones, defined as the number of times n_f each future cone f in this set was observed since the last time the algorithm was called. n_f may thus be negative in the case obsolete observations are removed. However since only previously observed data may be removed the updated distribution is always valid (see Fig. 4, *). An observation is for this algorithm a pair $u=(c,d_u)$ associating a past cone to a distribution update over futures. An estimated causal state is a pair $s=(U,d_s)$ associating a set $U=\{u\}$ of past cones and their observations to an average future distribution $d_s=\Sigma_U d_u$.

Figure 4 presents the algorithm pseudo-code:

```
Inputs:   New observations in a set $U_{obs}$
          A current set of states S, possibly empty
Output:   The set S updated with $U_{obs}$
```

```
for each u ∈ $U_{obs}$ in random order
    if a state s ∈ S contains v=($c_u$,$d_v$) ∈ $U_s$
        s ← ($U_s$ \ v, $d_s$ - $d_v$)              # remove v from s
        if s = Ø
            S ← S \ s
        else
            call merge({s})
        $d_u$ ← $d_u$ + $d_v$                    # see main text note *
        if $d_u$ = Ø                        # all futures removed?
            loop to the next u
    M := Ø
    for each state s ∈ S, if $\chi^2(d_u,d_s) < \alpha$
        M ← M ∪ {s}
    if M = Ø
        S ← S ∪ { ({u}, $d_u$) }  # no match, create a new state
    else
        let s ∈ M                          # choose one s from M
        s ← ( $U_s$ ∪ u, $d_s$+ $d_u$)              # insert u into s
        call merge(M)

subroutine merge(M):
    let s ∈ M
    for each state t ∈ M, t≠s
        s ← ( $U_s$ ∪ $U_t$, $d_s$+ $d_t$ )           # merge t with s
        S ← S \ t                          # remove t from S
    M ← Ø
    for each state t ∈ S, t≠s, if $\chi^2(d_t,d_s) < \alpha$
        M ← M ∪ {s}
    if M ≠ Ø call merge(M)
```

Fig. 4. Incremental algorithm for estimating statistical complexity

More details are given in [10], especially about the need for the merge step: After a call to the algorithm no two clusters may match. The risk of attributing a past to a wrong cluster decreases each time that past cone is observed again, since that past is then re-clustered with the updated observations.

The algorithm may be called with as little as one added or removed observed light cone pair, which makes it fully incremental. The algorithm can also be called at the end, with all available observations, making it fully non-incremental. Speed can be traded for computation granularity by choosing how much data to present before re-clustering. Only the clusters that are affected by the new observations are modified. Utility functions are provided to compute the local complexity for each cluster and the system global statistical complexity.

In order to validate the algorithm in practice a similar setup

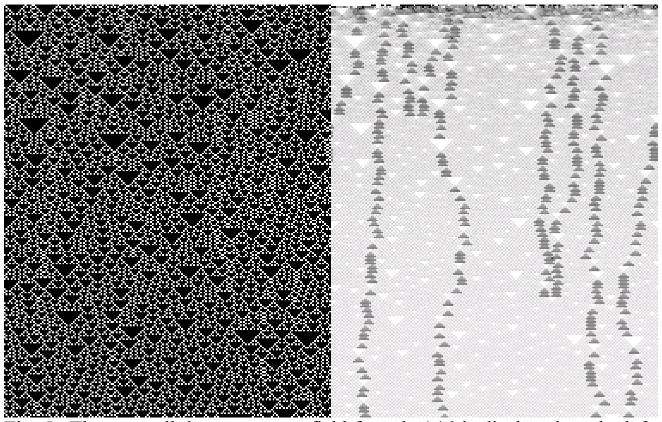

Fig. 5. The raw cellular automaton field for rule 146 is displayed on the left, the local complexity field given by the fully incremental version of the algorithm is shown on the right, starting from the same random initial conditions. Complexity values are mapped to a gray scale so black is more complex. Time goes top to down, space is a one-dimensional cyclic array.

as in [9] is tested: A cellular automaton is defined using an elementary update rule. The rule number 146 is chosen so Fig. 5 in this document is comparable to Fig 6. in [9]. Results for the other rules matching examples from [9] are not displayed here due to space limitations, but they can be generated with a program provided together with the reference implementation of the algorithm. The past (resp. future) light-cones are made of the current cell together with all interacting cells in the 3 rows before (resp. after) the current row. Observations are added one by one, left to right, top to down. Note how the algorithm can find patterns of higher complexity, and how the triangles visible in the raw cellular automaton field are mapped to a light gray background in the other regions. The incremental version is able to distinguish patterns quite fast, and then converges to the stable states given by the non-incremental version. Compared to the algorithm in [9] there is now the possibility to get complexity estimates on the fly as soon as new data become available. However the cost is a re-clustering of previously observed data matching the updated distribution.

For our main neural network scenario each neuron is a distinct system: observations are collected for each neuron separately. To make a parallel with cellular automata, each neuron would correspond to a different evolution rule. The local complexity field is restricted to the only available spatial location in each separate system. In this context, the global statistical complexity measure derived from all clusters for that neuron makes more sense. The next part makes use of the incremental algorithm to monitor how the global measure evolves while learning.

III. Experiments And Results

A. Experimental setup

With the incremental statistical complexity measure we now have a powerful tool to reach the initial goal of this work in addition to the separation property: To study the effect of applying learning rules on the recurrent spiking neurons layer.

The experimental method is expressly defined like in [5] so as to make results comparable. The Liquid State Machine (LSM) is first asked to perform a classification task without any learning rule applied to the recurrent layer. The goal of this experiment is to assert the base capabilities of the LSM, so we have something to compare with later on when applying the learning rules. Indeed, even if in the base experiment the recurrent layer is not modified, the linear classifier is still trained: This corresponds to the usual setup of the LSM [1].

The Hebbian learning rule version for spiking neurons that is proposed in [3] is then applied in a second experiment. Training is now effective for the recurrent layer, in addition to the output linear classifier that was also present in the base experiment. This is where the indicators presented in the first part of this document become useful: We can now analyze what is the effect of applying a learning rule on recurrent neurons not only on the final network performance, but also on the recurrent layer itself.

The same experiment is then repeated again using the multifractal learning rule that is proposed in [5]. The goals of this third experiment are to validate the use of the indicators in an additional context and to compare the two learning rules.

Input data are provided as vectors of continuous values between 0 and 1. The task to is classify the vectors into two output categories. The continuous inputs are first mapped to spike trains according to the frequency population coding scheme that is presented in [5]. Each input is connected to three receptor neurons. These receptors spike with their own frequency dependent on the continuous input value, with minimal and maximal spiking frequencies set to 30 and 90 Hz. The training instances are exposed for one second of simulated time. The LSM is built with a 4×4×4 cube of recurrent spiking neurons with the same parameters as [5]. The learning rate for the Hebbian and multifractal rules is set to 0.07. The network time is monitored for 50 epochs, an epoch is defined as the simulation time necessary to present all training instances in random order. Real data from the Cancer1 task of the Proben1 dataset [12] are used, but due to the computational power necessary to estimate the separation and the statistical complexity indicators that real data set is reduced to the first 40 training and 30 testing instances without duplicates.

Let's note S_j the network activity vector S for each data instance j, as defined in Section II.A. Let's also note C_j the class target for the activity vector S_j. The output layer weights W of the LSM are computed in all three experiments as the least squared error solution for minimizing $\|WS-C\|^2$, for each epoch. In the Hebbian and in the multifractal experiments the corresponding learning rule is additionally applied to the recurrent layer. All three experiments are then run with the same random seed in one batch. 30 batches of these 3 experiments are averaged so as to produce the graphs below.

B. *Monitoring the indicators*

1) Separation

The measure of separation is a straightforward application of the formula presented in Section II.A to two output classes. It is computed after each epoch and monitored throughout the training phase. Fig 6. presents the average evolution graph.

Separation measures the ability of the network to distinguish between the inputs. When no learning rule is applied to the recurrent layer the separation remains constant (neglecting variations due do presenting the spike trains in random order each epoch). Without learning the "liquid" classification properties are determined by the random initial conditions and they are not modified during the experiment.

Conversely, applying a learning rule to the recurrent layer is a complementary way to make the LSM learn. The linear global classifier acts on the data space consisting of filtered signals from the recurrent layer (average spike counts in the simplest form). Separation precisely measures how data in this space may be classified linearly into the output categories. While a LSM can learn with only a global linear classifier active [1], separation shows that applying a learning rule to the recurrent layer boosts the process. Section III.C. below shows that this boost may also be too much, with a risk of over fitting.

These experiments also serve as an independent replication of [4] regarding the effect of applying Hebbian learning on separation, as well as an extension to another learning rule.

2) Statistical Complexity

A statistical complexity analyzer is attached to each spiking neuron in the recurrent layer. Each spike is consigned and kept as long as necessary for building the light cones: the transmission delays with efferent and afferent neurons are checked against the neuron's current "present" time. As soon as a spike is received that allows to determine the future cone for that present time, this cone is built, and the present is advanced. The last spike received by a neuron is therefore part of that neuron's future cone. All neurons present time are different and lag on the simulation time, up to the maximum delay as explained in Fig 3. Since it is not certain at this point how much data can be added while respecting the stationary approximation, as explained in section II.D, it was decided to remove obsolete data as soon as possible: In each experiment the expired light cones are removed after one epoch.

As before, the base experiment serves as a sanity check for the algorithm: The recurrent layer is not modified, so it is expected that the complexity remains at the same level for all epochs. Both the multifractal and the Hebbian learning lead to a decrease of the neurons complexity (Fig. 7). But the statistical complexity measure is purely data based and just reflects the amount of information from the past of a neuron that is relevant to predicting its future. In the context of this work this means that both learning rules have modified the connections so that the recurrent neurons produce spikes that are easier to relate to their activation signals. In other words, applying the learning rules has made the recurrent neurons more predictable.

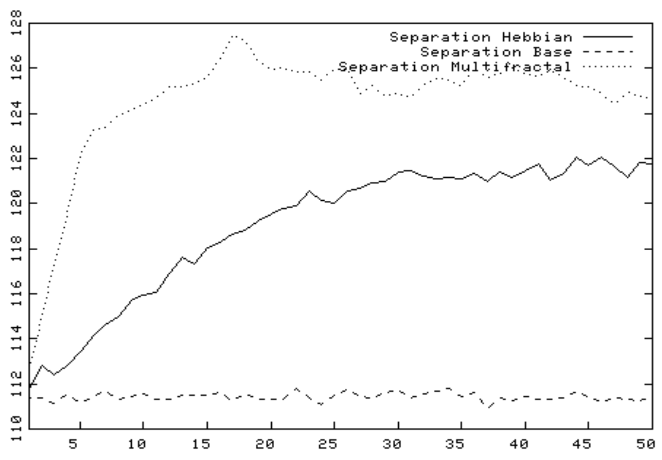

Fig. 6. Evolution of the Separation indicator during the learning phase. The separation unit is homogeneous to a number of spikes per second. Separation is plotted against the number of training epochs.

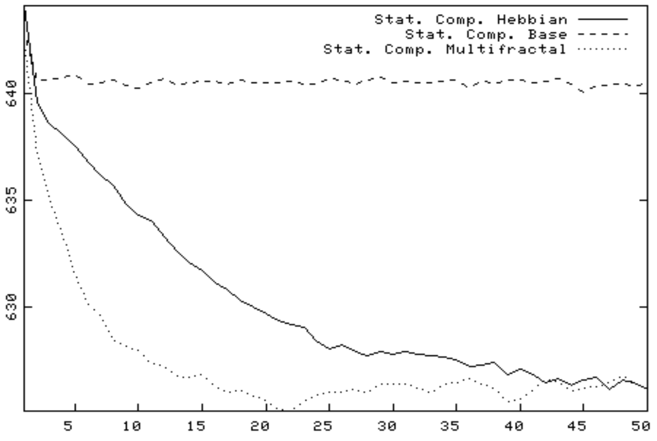

Fig. 7. Evolution of the Statistical Complexity indicator during the learning phase. The Statistical Complexity is plotted against the number of training epochs. The unit is the number of bits that would be needed to represent all the neuron causal states in the recurrent layer.

C. Classification results

The results for the classification task on the testing data set (generalization error) are given in Table I.

TABLE I: GENERALIZATION ERRORS

Experiment type	Base	Hebbian	Multifractal
Average Classif. Error	4.9 %	5.3 %	6.7 %
Standard Deviation	3.7 %	3.8 %	5.3 %

In order to interpret the figures in Table I a linear classifier was trained directly on the input/output mappings using Least Squared Error estimation. This is equivalent to connecting the output layer directly to the input data. The result is a classification error of 23.3%. Therefore, the recurrent layer indeed brings a significant "reservoir" of non-linear transforms, as expected from [1], even for as few as $4 \times 4 \times 4 = 64$ recurrent neurons. The base experiment indicates the performance of the LSM with respect to this fixed reservoir only, matching the setup from [1], and validates that LSMs may be used for classification problems. Unlike [5] however applying the learning rules in the present case results in a worse generalization error. This is simply explained by over fitting, which also matches the large separation increase on the testing set shown in Fig. 6. The two learning rules over fitted results also match their respective separation performance.

An hypothesis for explaining why applying the learning rules on the recurrent layer produces over fitting is perhaps the small size of the network. Less recurrent neurons mean less states to synchronize, and synchronization is a fundamental component of the learning rules as explained in [5]. The results in Fig. 7. showing a lesser statistical complexity mean that the neurons have more predictable behaviors than before applying the learning rules, which also corroborate to some extent the hypothesis that the recurrent neurons would get synchronized. A proper synchronization study would be needed to assert the validity of this hypothesis, possibly with techniques like these explained in [13]. In any case the larger network presented in [5] was applied the same learning rules and the results did *not* suffer from over fitting in that case. The remark from [14] that larger networks perform generally better for LSMs may thus be extended to the case where learning rules are applied to the recurrent layer.

IV. CONCLUSION

Separation was proposed in [1] and applied in [4, 2] in order to quantify the ability of a recurrent spiking neural network to be able to process the inputs. This indicator was confirmed to be a useful way to monitor the effects of applying learning rules on the recurrent neuron layer. In particular, the observed increase in separation reflects a boost of the capabilities of the output layer linear classifier in Liquid State Machines. The experiments carried in Section III also exhibit a case where this boost is too much so over fitting occurs, and the greater the observed separation the greater the over fitting. Yet similar experiments from [5] on a larger network produced better generalization performances in the cases where the learning rules are applied to the recurrent layer than in the base experiment. A future direction could be to investigate which cases lead to over fitting or not, so as to better apply learning techniques on the recurrent layer.

A complementary insight on the system is provided by the statistical complexity indicator. Unlike separation the statistical complexity definition does not depend on a specific input/ output mapping: Statistical complexity is a purely data based objective property of the recurrent layer. Statistical complexity thus quantifies the influence of the learning rules in themselves, irrespectively of the data classification task. It was found that when a learning rule is applied the complexity of the neurons decreases. This means that neurons become more predictable, that their response to their activation signals is more determined than before applying the learning rule. Another possible future work would be to connect this finding to the synchronization properties of the learning rules, and to research why learning increases the neurons predictability.

The incremental algorithm that was presented in Section II is very general and not restricted to the analysis of neural networks. In fact, the algorithm is applicable to any complex system as it does not presume anything about the kind of relations (non-linear, recurrent... [8]) that take place within a system. The new algorithm is well adapted in particular to the study of recurrent spiking neurons, but in fact it represents a contribution in itself that may very well be reused outside the context of spiking neural networks.

REFERENCES

[1] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations", *Neural Computation*, vol. 14, num. 11, pp. 2531-2560. Nov. 2002.

[2] R. Legenstein and W. Maass, "What makes a dynamical system computationally powerful?" *New Directions in Statistical Signal Processing: From Systems to Brain*, Cambridge, MA: MIT Press, 2005.

[3] S. Song, K. D. Miller, L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity", *Nature Neuroscience*. vol. 3, num. 9, pp 919-926, Sep. 2000.

[4] D. Norton and D. Ventura, "Preparing more effective Liquid State Machines using Hebbian learning", *International Joint Conference on Neural Networks*, proceedings pp 8359-8364, 2006.

[5] N. Brodu, "Learning using dynamical regime identification and synchronization", *International Joint Conference on Neural Networks*, proceedings pp 662-668, 2006.

[6] C. R. Shalizi, Causal Architecture, Complexity and Self-Organization in Time Series and Cellular Automata. PhD dissertation, 2001.

[7] C. R. Shalizi, "Methods and techniques of complex systems science: An overview", Chapter 1, pp 33-114, T. S. Deisboeck and J. Y. Kresh, Complex Systems Science in Biomedicine, 2006.

[8] C. R. Shalizi, K. L. Shalizi, "Blind construction of optimal nonlinear recursive predictors for discrete sequences", *20th conference on Uncertainty in artificial intelligence*, proc. pp 504-511, 2004.

[9] C. R. Shalizi, R. Haslinger, J-B. Rouquier, K. L. Klinkner, C. Moore, "Automatic Filters for the Detection of Coherent Structure in Spatiotemporal Systems", *Physical Review E*, 73(3) 036104, 2005.

[10] N. Brodu, "Practical investigations of complex systems", PhD dissertation, 2007, to appear.

[11] E. Goodman, D. Ventura, "Spatiotemporal pattern recognition via liquid state machines", *International Joint Conference on Neural Networks*, proceedings pp 7579-7584, 2006.

[12] Lutz Prechelt, "Proben1, a set of neural network benchmark problems and benchmarking rules". Technical Report 21/94, Fakultät für Informatik, Karlsruhe University, 1994.

[13] C. R. Shalizi, M. F. Camperi, K. L. Klinkner, "Discovering Functional Communities in Dynamical Networks", to appear in proc. *ICML workshop "Statistical Network Analysis: Models, Issues and New Directions"*, 2006.

[14] W. Maass, T. Natschläger, H. Markram. "Computational models for generic cortical microcircuits". *Computational Neuroscience: A Comprehensive Approach*, ch 18, pp 575-605. J. Feng Ed. Chapman & Hall/CRC, 2004.