Learning using Dynamical Regime Identification and Synchronization

Nicolas Brodu

Abstract—**This study proposes to generalize Hebbian learning by identifying and synchronizing the dynamical regimes of individual nodes in a recurrent network. The connection weights are updated according to the closeness in the observed local dynamical regimes. Demonstration of the viability of this method is provided on spiking recurrent neural networks. Experiments are made with both artificial and real continuous data, using a frequency population coding.**

I. Introduction

This study proposes to investigate learning mechanisms in recurrent spiking neural networks in the light of dynamical regime synchronization of local interactions.

An algorithm for applying Hebbian learning to spiking recurrent neural networks is presented in [1]. This algorithm relies on the assumption that individual nodes modify their connections strength so as to synchronize their activity. Amongst other interesting results, that article discusses the benefits of a higher degree of synchronization in terms of a faster global processing capacity.

In the present study, this idea is extended to the more general notion of synchronization between the dynamical regimes of the individual nodes. A dynamical regime is meant to represent some sufficiently stable pattern in system state. The terminology of an attractor would imply some idea of finitude. A dynamical regime may be transient, or even not associated to particular underlying equations. In an open and dissipative system, this would describe well some sustained pattern, that is not stable in itself, but which is sufficiently persistent so it can be identified, like a whirlpool. This is precisely one of the main properties of the Liquid State Machine: computation without stable state [2]. In such a setup, the fading memory property [3] assumes the role of the dissipative part of the system. The openness comes from the assumption external energy is available to emit spikes, which by definition are short impulses of energy higher than the rest state.

Hence, a Liquid State Machine is in a permanently sustained mode, where energy influx constantly shifts the trajectories of this otherwise converging dynamical system. In [4] and [5] an argument is presented to further analyze the global behavior of a similar system in terms of self-organized criticality. The boundary between ordered and disordered global regimes is shown to correspond to a maximum in terms of processing power. Given the aforementioned remarks on openness and dissipation, an hypothesis would be this boundary corresponds to the case where information is neither destroyed by dissipation, nor submerged by external influences. An informational approach to this problem was provided in [3], both in the boolean framework considered by [4] and for the Liquid State Machine setup.

These global properties of the system are obtained through local interactions between neurons only. One of the learning process task, as will be detailed in the next subsection, is to modify the local interactions so as to push the system toward these desirable global properties.

This study proposes to monitor the dynamical regime properties of the individual nodes in addition to the global properties of the system. The Hebbian learning rule for spiking neurons presented in [1] is analyzed in the light of its dynamical effects. Then, these principles are generalized to derive a whole class of learning rules, based on synchronization of local dynamical regime properties. The hypothesis to test is that it is not the Hebbian rule specific choice of observable that leads to learning, but that any reasonable dynamical regime identifier will lead to similar results.

One proposal is made for a new rule using a regime identifier based on multifractal analysis as example. The results of both the new and the Hebbian rules are monitored globally. The the performance of the network is measured on a classification task, and compared to the basic Liquid State Machine performance.

The next section considers Hebbian learning in the light of individual nodes dynamical regimes, and introduces the proposed generalization. Section III describes how the new learning algorithm is tested in practice. Section IV discusses the results and section V concludes on the issues encountered in this study.

II. Learning

The Hebbian learning rule for spiking neural networks proposed in [1] monitors the time difference between the spikes emitted by an afferent and the current node.

Consider one neuron, together with its dendrite and afferent neuron synapses. When a spike is received, it contributes to the change (excitatory or inhibitory) in the membrane potential. In turn, this may trigger the current neuron spike emission. Based on this simple causality relationship, the idea is to favor the afferent neuron that provokes the changes. When monitoring the synapses activity from outside, without knowledge of their internal state, the connection strengths are increased whenever spikes are observed in this timely pattern. Conversely, whenever an afferent neuron produces a spike after the efferent neuron has itself produced one, then this spike contributes to nothing, especially when arriving during the efferent neuron refactory period. The learning function proposed in [1] consists in exponentially increasing or decreasing the connection strength, with respect to the spike time difference Δt between the efferent and afferent neurons. The connection strength is

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

This work was financed in part by the EADS Corporate Research Center, with the support of the French Ministry of Foreign Affairs.

then updated according to the following equation, with F(Δt) the increment to apply to the weight:

$$F(\Delta t)=\begin{cases} A_p \exp(-\Delta t/\tau_p), \Delta t \geq 0 \\ -A_n \exp(\Delta t/\tau_n), \Delta t < 0 \end{cases}$$

The function implemented for the experiments presented in the next section instead uses a proportional gain G for the new updated weight compared to the initial weight:

$$G(\Delta t)=\begin{cases} 1+R\exp(-\Delta t/\tau_p), \Delta t \geq 0 \\ 1/(1+R\exp(\Delta t/\tau_n)), \Delta t < 0 \end{cases} \quad (1)$$

The Hebbian rule is based on the dynamical properties of the afferent and efferent neurons spike time series. This learning rule effectively has a maximum of the F(Δt) value whenever the spikes are perfectly synchronized.

But the nodes in a real or artificial network cannot be all synchronized, because in that case there would be little left for the emergent global behavior. Re-using terminology from the introduction, this would correspond to an extreme example of ordered state, therefore far from the critical line appropriate for maximal processing capabilities.

At an individual neuron level, only one afferent neuron will produce the spike that will trigger the efferent node to also spike. In [1] this idea is expressed as competitive learning: since only one afferent node can possibly be perfectly synchronized, the other nodes will have suboptimal Δt values in the learning process. This can be seen as a form of frustration, to re-use another concept from chaotic recurrent network systems [6]. The Hebbian rule intrinsically carries its own source of frustration.

A. Motivation for Dynamical Regime Identification

This Hebbian learning rule short analysis can be generalized. The main idea is to base the learning algorithm on the synchronization between the behaviors of the nodes. The Hebbian rule considers that the spiking times difference between afferent and efferent nodes is the relevant parameter to synchronize. However, the dynamical properties of neuronal activity are possibly not restricted to only this observable. In particular it seems also plausible that connection strengths are updated on a larger time scale, considering the neuron average behavior instead of instant time differences.

What is proposed here is to study the dynamical properties of each neuron. The synchronization would occur between nodes having the same dynamical regime. The problem is then to find a relevant identifier for these dynamical regimes, which is in itself an active research domain. Similarity in dynamical regimes assumes the same function, and extend the notion, as closeness in spike time for the Hebbian Learning rule.

No claim is made on biological significance. It is possible that real neurons synchronize their dynamical properties, but in that case the method used to identify these dynamical regimes need not correspond to the method used by real neurons. This is the same issue as the correlation vs causality pitfall, and no biological significance is required for the artificial learning rule to be effective.

The methodology is then:

- Choose a dynamical regime identifier. It should be expressed in terms of local observables only, as opposed to a global property. In the Hebbian rule case, this identifier is Δt.
- Choose a significant target property for this identifier, related to synchronization. In the previous case, the goal is $0 \leq \Delta t < \varepsilon$, with ε as small as possible.
- Derive a learning rule. Care should be taken to ensure the rule does not lead to an ordered (or completely random) state, for example using frustration.

B. A Proposal Using Multifractal Analysis

As a practical implementation of this methodology, consider as a working hypothesis that the multifractal spectrum[1] of the inter-spike time series provides enough information to be used as a dynamical regime identifier.

The justification is as follow. Consider the inter-spike time of a single neuron as an observable. On average, this could be seen as the activation frequency inverse. However, averaging looses all dynamical information, which is precisely not desirable for the definition of a regime identifier. Consequently, a way must be found to preserve at least some of the dynamical information present in the time series. Frequency could complement the regime identification in a second step, but cannot be used alone.

The multi-fractal spectrum of the inter-spike time series may be chosen instead. Unlike the time average, this spectrum gives some information on the relationships between a series last value and the past values. Hence this captures part of the observable series dynamics. This is probably not sufficient to serve as a complete regime identifier, but this will provide a different enough approach from the Hebbian learning rule so as to be able to validate the methodology on a practical example: The hypothesis to test is that it is not the Hebbian rule specific choice of observable that leads to learning, but that any reasonable dynamical regime identifier will lead to similar results.

The synchronization idea provides a way to define a significant target for that spectrum observable. Two neurons will be declared having a similar dynamical regime, according to the chosen identifier definition, whenever they have the same spectrum.

The learning rule will therefore be based on the closeness of the spectra $h_a(q)$ and $h_e(q)$, for a neuron *e* and an afferent neuron *a*, whenever these spectra estimation is reliable. The correlation coefficient for the exponential fitting in the spectrum estimation [7] serves as an indicator for that spectrum reliability. This defines a stability condition. In addition to spectra closeness for stable nodes, unstable neurons are also taken into account in the following rule:

- $G(s)=1+R\cdot\exp(-C\cdot s)$ (2)
  with $s=\sum_q (h_a(q)-h_e(q))^2$
  when both the $h_a(q)$ and $h_e(q)$ spectra are reliable,
- $G(s)=\frac{1}{1+R}$ when the neuron *e* is stable but the afferent *a* is not,

1 Multifractal Analysis is concerned by the scaling properties of the fluctuations in the time series. This is a form of statistics, related to the smoothness and regularity of the data, as well as to the time series self-similar properties. This framework has been applied to many domains, including physics, biology, finance, geology, internet traffic analysis... More information, together with complete references, can be found in [8], [9] for a discrete version, and [7] for a practical and efficient algorithm.

- No change when the neuron *e* considered is itself unstable.

R is the learning rate, C a constant for the spectra closeness sensitivity. These equations are similar to the ones for the Hebbian learning.

This learning rule takes into account that some nodes may not have reached a sufficiently stable sustained regime yet, or at least one for which the multifractal estimation failed. In that case, a stable node will reduce its connections with the offending afferent neuron. Unstable nodes do not change their afferent weights. As for the Hebbian learning rule, a difference of s=0 gives the maximal G(s) value.

III. Validation and Practical Experiments

Validation of the approach described in the preceding section is made with the following experimental setup:

1. Use a recurrent spiking neural network. Liquid State Machines are well suited for the purpose.
2. Study the intrinsic capabilities of the recurrent network to process a simple classification experiment. No learning is applied to the recurrent layer.
3. Study how applying the Hebbian rule improves these capabilities.
4. Compare with how applying the dynamical regime based rule improves the capabilities.

Each of these points will be detailed in the next subsections.

A. Using a Recurrent Spiking Neural Network

The Liquid State Machine setup described in [10] consists of 3 parts: Some input nodes, responsible for generating or transmitting spike trains. A large reservoir of spiking neurons randomly interconnected, responsible for combining the input signals non-linearly and for providing a form or memory. Readout neurons, which are actually equivalent to a linear classifier for the interconnected spiking neurons signals.

Only the last part, the linear classifier, is subject to learning in the Liquid State Machine setup. The non-linearity reservoir is assumed to hold enough basic transformations on the inputs so that the output classifier will be able to find an appropriate combination for the task considered. Example tasks mentioned in [10] include computing a polynomial combination of the inputs, spike coincidences, or a sum of rates.

For the purpose of this study, all three parts were explicitly separated. The next subsection discusses the inputs. The subsections C and D cover the recurrent spiking neurons.

[10] specifies that output neurons receive a low-pass filtered signal from the recurrent neurons. This effectively corresponds to averaging the spike trains. In practice, simply feeding the neuronal activities (spike counts per time unit) to the classifier has an equivalent effect.

The global recognizer thus receives an activity signal from each neuron in the reservoir. Its task is to find a weighted combination of these activities that corresponds to the input classification. This is the subject of subsection B2.

B. Intrinsic Capabilities of the Recurrent Network

1) Input Representation

The capabilities of the network are quantified by its success in classifying simple inputs. A proposed benchmark for classification experiments [11] consists in generating two Poisson spike trains, and jittered versions are produced to build a data set. The task is to classify the noisy spike trains into the two original categories.

In this setup, the input neurons just repeat the spike trains, all processing is done by the recurrent layer. Unfortunately, the number of spikes over time is not enough to accommodate for the multifractal spectrum estimation (see next section). Population coding could be used to shift the trains in each neuron, thus producing effectively more spikes in the recurrent layer. But then, why would each neuron in the population produce exactly the same spike train? Moreover, real data is most often available in the form of continuous values instead of spike trains. Another setup is thus needed for this study to take continuous inputs in consideration.

A data instance consists in different channels of communication, each corresponding to some particular data parameter. For example, the Proben1 benchmark [12] proposes amongst other tasks to classify cancerous tumors as malignant or benign based on nine different continuous parameters, like the frequency of bare nuclei observed in the tumor. For this kind of inputs, spike trains are clearly not available and must be generated from the continuous data values.

A frequency-based population scheme is introduced. A group of input neurons is dedicated to each data channel. Within a group, each input has slightly different parameters, leading to slightly different responses encoding for the same information.

The model chosen is simple: the neuron accumulates the value it receives over time, then when a threshold is reached, it sends a spike. A possible improvement would be to use full alpha neurons, with membrane potentials, refactory periods, etc. Yet, this simple model has proved effective for encoding the channel values for the needs of this study.

At each time increment δt, a neuron accumulates $A(\delta t) = \alpha v+\beta$, with v the channel value, α a coefficient, and β a constant that will eventually provoke a spike even when the channel value is 0. Then, when a threshold T is reached, A is reset to 0 and a spike is generated. Note that in the case of constant values in a channel, the neuron responds with a fixed frequency linearly dependent on the input value. Therefore, by fixing an arbitrary threshold, it is possible to specify the neuron reaction in terms of a minimum and maximum frequency response.

This is the first part of the population coding. The second part corresponds to introducing variations in the α and β parameters for neurons within the same group. When this is done, each channel effectively generates G different spike trains, with G the size of the group associated to this channel. Each of these spike trains uniquely identifies a given input by its frequency response. Close inputs lead to close responses, thanks to the linear frequency relation. However the group behavior is more elaborated, since the variations in α and β prevent individual members of the group from being synchronized. This in turns gives the recurrent neuron layer more possibilities, with different spike trains conveying the

same information to choose from.

2) Training the Global Recognizer

The chosen input method, direct spike train feeding or population frequency coding, is completely independent from the output classifier task. The recognizer does not have access to the input neurons, only to the recurrent ones. It is provided a classification value (+1 or -1), that has to be matched based only on the recurrent neurons activity.

As for the basic Liquid State Machine framework, this learning task requires no training on the recurrent neuron connections. Only the classifier weights are modified.

This is done in this project by a simple gradient descent rule. Given the neuron activities a_n, the recognizer is a weighted linear combination:

$$R = \sum_{n=1}^{N} w_n \cdot a_n$$

The task it is given is to minimize the classification error:

$$E = \frac{1}{2}(R - C)^2, \text{ with } C = \pm 1 \text{ the data class} \quad (3)$$

This is simply done by updating the weights according to a gradient descent rule:

$$\Delta w_n = -r \frac{\partial E}{\partial w_n}, \text{ with } r \text{ the learning rate.}$$

When given an unknown network state to classify, R is computed. If R>0 then class 1 is returned, else class -1 is returned.

3) Monitoring the Network Performance

Each data instance is presented to the input nodes for a fixed duration, then another data instance is chosen at random from the training data set. Once all instances are presented, the current epoch completes and the whole process is repeated again. Given a sufficient number of epochs, this method averages out all spurious relations that would occur between the end of a spike train and the beginning of another if the instances were always presented in the same order.

When a maximum number of epochs is reached, or possibly also when reaching a stopping criterion for $E<\varepsilon$, the training is stopped. The performance of the network is then asserted on the test set.

The same method applies for testing: each unknown test instance is presented in turn, for a fixed duration, in random order. At the end of that duration, the recognizer is asked to classify the instance according to the activities observed during that time interval. A sample classification error is collected over the whole epoch by counting the proportion of instances that were incorrectly classified. The process is then repeated for another epoch, an so on, so as to smooth down the potential spurious relations aforementioned.

This gives the average of the classification error of the network in its basic form, without applying a learning rule to the recurrent neurons. The corresponding variance reflects the influence of changing the order the instances are presented to the network.

The choice of monitoring the algorithms performance with respect to the number of epochs makes the results independent of the computational time used for the experiments. On the one hand this provides a fair comparison in terms of internal network simulated time, but on the other nothing can be deduced about the real time necessary to run the experiments. It turns out that for the computations described in the next section the Hebbian rule was slightly faster than the multifractal one.

However, the exact timing strongly depends on the implementation of these algorithms. An history of past spikes is necessary for the Hebbian rule so as to track back cases where the afferent node spike comes after the efferent spike (negative learning case). In turn, each update causes a search through that list for updating the last spike information, at least for cases where Δt is below some cutoff threshold for the exponential tails of Eq. 1. The multifractal rule does not need to consider node relations at each update, and should scale up better with respect to the network connectivity. But for each spike it needs to update the corresponding time series spectrum, an O(L) algorithm, with L the number of wavelet decomposition levels.

Depending on the connectivity number, the exponential cutoff threshold, the multifractal spectrum estimation precision, and the implementation details, it may be that one or the other of the rules is the fastest.

C. Applying the Hebbian Learning Rule

The Hebbian rule as given by the definition in section II can be applied on-line, continuously, each time a spike is generated.

Unfortunately this quickly leads in practice to a predominance of one afferent neuron over all the others (for a given efferent neuron). Moreover, this specialization may occur faster than the exposition duration of one data instance. In that case, not only is the over-specialization difficult to reverse for the next instance, but spurious effects may be amplified as well.

Of course, at the internal neuron level there is no notion of current data instance, classification, or epoch. Such notions are global, and a given neuron should not be aware of them. Nevertheless, a way must be found to ensure the effect of the Hebbian learning happens on a significant time scale.

One such way could be using a very low learning rate. However, this does not solve the problem of inter-instance spurious relations. An artificial cooldown period could be introduced, during which neurons are forbidden training, at the beginning of each exposition period. But this does not prevent recurrent loops to sustain spikes anyway.

In practice for this study, better results have been observed by maintaining statistics about Δt over one exposition period, then train using the average Δt, rather than using on-line training. Not only does this solve the time scale problem, but this is also plausible as aforementioned: In that setup, neurons do synchronize with other neurons that give consistent spike information on average, which is more reliable than reacting to each individual spike.

The last remaining problem is the predominance of only one neuron amongst competing afferent neurons. This problem was also acknowledged in [1]. The original motivation for competitive learning was that synapses do not have access to other synapses data, hence the rule should not count on restricting a synapse variation according to what happens to the other synapses. The local competition inherent to the rule is a way to solve that inter-synapse non-communication problem, but it is unfortunately not sufficient to ensure a

proper balance between afferent neurons.

A common trick is to fix the sum of all afferent connection weights, and impose bounds on the weights. This does not in itself prevent one neuron from dominating all others, it just makes that event less likely to happen. Whether this is biologically motivated or not is outside the scope of this study, perhaps some bounding mechanism at the dendrite level assumes such a regulation role, perhaps not. For the current purpose, it is sufficient that such a regulation is done (or not) exactly the same way for both the Hebbian and the multifractal based rules. Since the regulation prevents premature specialization in the Hebbian case, it was used.

The network is then run with both the Hebbian rule and the recognizer regression active at the same time, using the same epoch setup as previously described for the recognizer training.

D. Applying the Dynamical Regime Based Rule

For each neuron in the network, each time the neuron spikes, the multifractal analyzer for that neuron is fed with the inter-spike time since the last spike. Thanks to an algorithm developed in [7], this can be done incrementally, with each new data updating the spectrum estimate on-line. Old data are discarded, so the spectrum is computed over the recent history. Provided there are enough spikes, this history is smaller than an instance exposition duration. Precisely, the number of input spikes to the network is a direct consequence of the population coding scheme introduced in section III.B.1. The internal nodes spiking frequency is harder to estimate, but can be measured experimentally.

The main problem for the multifractal estimation in real-time is the delay in capturing the low frequencies. Generally speaking, it is not possible to capture the frequency decomposition of a signal instantaneously. The lower the frequency to capture, the larger the acquisition delay. The multifractal estimation used in this study is based on wavelet decompositions; It is subject to this problem in the form of an acquisition delay to get the highest decomposition wavelet coefficients.

The same setup as for the Hebbian rule provides the solution: learning occurs only at the end of the exposition duration. So long as this exposition is long enough it covers the acquisition delay and the discarding of too old values: the spike time differences captured by the analyzer effectively correspond to the last instance.

IV. Results

A. Artificial Data

Experiments were run with the following parameters: 10 channels, a population of 5 nodes per channel, a minimum input frequency of 1.0 Hz, a maximum input frequency of 50 Hz, and a variance for the α and β ratios of 0.1. Data in each channel is drawn from a uniform random distribution in 0..1. Jittered versions of these data are produced by adding normally distributed random noise with mean 0 and variance 4.10^{-3} so as to build the training and test sets.

Each run consists in 20 training instances, 20 test instances, 50 training epochs, 10 test epochs. 3 experiments are run with the same random seed: One for the basic network capacity, one for the Hebbian rule, and one for the multifractal based training. The global linear recognizer learning rate is set to 0.01: a too large value makes the gradient descent unstable. The R parameters in Eq. (1) and (2) are set to 0.1. The reference parameters $\tau_p=\tau_n=20$ ms in [1] are reused for Eq. 1. The C parameter, Eq. 2, was set to 0.1 according to exploratory preliminary experiments. Individual data instances were exposed to the network for a duration of 1s simulated time. The Liquid State Machine parameters are set to the values in [10] and a cube of 6x6x6 nodes was created for the recurrent layer.

In many simulations, the recognizer correctly classified all test instances, even without training the recurrent neurons. The classification error cannot be used to study the influence of the learning algorithms, so the recognizer training error as defined by Eq. 3 was monitored.

Results are provided in Figures 1 and 2 for particular runs with a marked difference between the multifractal and the Hebbian rule effects. In each run, what differs is the speed of convergence of the global recognizer: the training process is boosted when the recurrent neurons weights are updated in addition to the global recognizer. In Figure 1, the multifractal learning rule gives the best results before convergence, in Figure 2 this is the Hebbian rule. Other runs show the case where both rules performances overlap: the error is successively lower for one rule, then the other, then the first rule again, etc.

Visual inspection suggests multifractal learning usually tends to produce the best improvements at the beginning of the training, Hebbian learning at the end. To assert this effect quantitatively, the difference between the base version and each of the Hebbian and multifractal versions was measured for each training epoch, and these differences were averaged over all runs (see Figure 3).

One possible hypothesis for this effect is that the synchronization criterion for the Hebbian rule can be

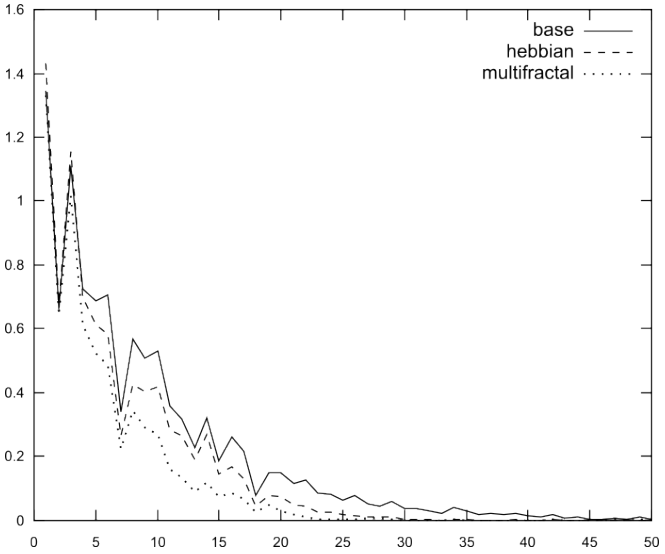

Fig. 1. The error as computed in Eq. 3 is plotted versus the epoch number. In the base experiment, only the global recognizer is active in the setup, the recurrent neuron weights are not modified. In the other experiments, the recurrent neuron weights are modified using the corresponding learning rule. This is the result for the random seed 16. In this particular example the base version gives the worse error, then the Hebbian rule, then the multifractal one.

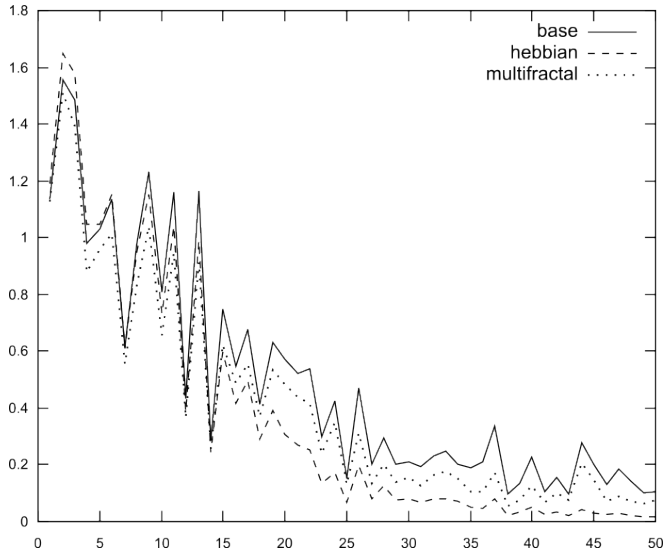

Fig. 2. This is exactly the same setup as in Figure 1, but with a different random seed (5). In this experiment the base version still gives the worse error, but this time the Hebbian learning rule gives a better result than the multifractal one.

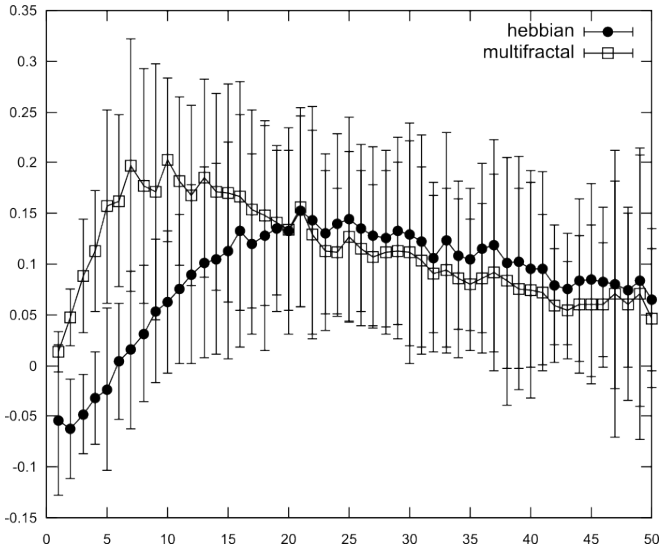

Fig. 3. Experimental measurement that on the artificial data, the multifractal rule exhibits the best improvements at the beginning of the training, whereas the Hebbian rule exhibits the best improvements at the end. The difference between each rule error and the base version error is plotted for each epoch. A larger difference means a lower training error. On this graph, the base version would be a straight line at y=0. The standard deviation over all runs is plotted as vertical bars.

achieved perfectly in practice for one (but only one) of the afferent neurons. Whereas for the multifractal case, the synchronization criterion is related to properties of the inter-spike time series, which depends on all the afferent neuron contributions. This difference could perhaps explain that the multifractal rule is less "stable" at the end of the training. Another noteworthy distinction is that the Hebbian rule seems to have a negative influence on the early stages of the training, though no hypothesis is given for this effect. In any case, both Hebbian and multifractal rules boost the learning capabilities.

The algorithms are quite sensitive to the choice of parameters. Higher learning ratios (ex with R=0.2) tend to prevent the learning process from converging. An extensive statistical testing taking in account a wide range of parameters for both algorithms, together with an explicit quantifier, would be needed to assert a statistically significant difference between the 2 rules. What was observed is that both multifractal and Hebbian learning improve the recognizer global capabilities in a similar way.

B. Real Data

Another study was conducted on the Proben1 cancer1 dataset [12]. This dataset originally contains 525 training instances (as compared to 20 in the previous experiments), and 174 test instances. Due to the lack of computational resources, the data set was reduced to only 100 training and 100 testing instances.

The setup for the Liquid State Machine, exposition duration, and all aforementioned parameters, are identical to the artificial data set study. Thirty batches of runs were executed for all three experiments, using the same random seed for each batch of three. Both training and classification errors were monitored. These were then averaged over all thirty batches to assess the performance of each learning algorithm.

The results are classification errors of respectively 6.20% (dev. 1.71%), 5.70% (dev. 1.34%) and 5.73% (dev. 1.58%) for the base, Hebbian and multifractal experiments. On some particular runs the Hebbian or the multifractal experiment produces worse results than the base experiment. Given the observed relatively high deviation compared to the mean values, and given the fact that each experiment result is within the deviation range of the others, caution is necessary.

However, two remarks may be noted:

- These numbers compare quantitatively with the results in the Proben1 article [12] using sigmoidal networks. In itself, this validates the viability of the spiking neural network approach for this kind of classification problems.
- These experiments were not set up to demonstrate which rule is better, but to demonstrate the viability of basing a learning rule on dynamical regime identification. Given that results for both Hebbian and multifractal learning are very similar, and below the base version, the methodology presented in section II.A. was applied successfully.

Concerning the training process, the difference with the base version was measured as in the artificial data case. Result is provided in Figure 4. On the real data, the Hebbian learning rule has a better influence on the training process than the multifractal one. The first part of the training is not as clear as before, but both rules quickly stabilize to some final state where no more improvement is observed compared to the base version. What's surprising is the fact that despite this training error difference, both rules give the same classification rate.

V. Conclusion

In the Liquid State Machine setup [10], the recurrent neural layer is used only as a nonlinearity reservoir; no learning mechanism is applied to it. The experiments

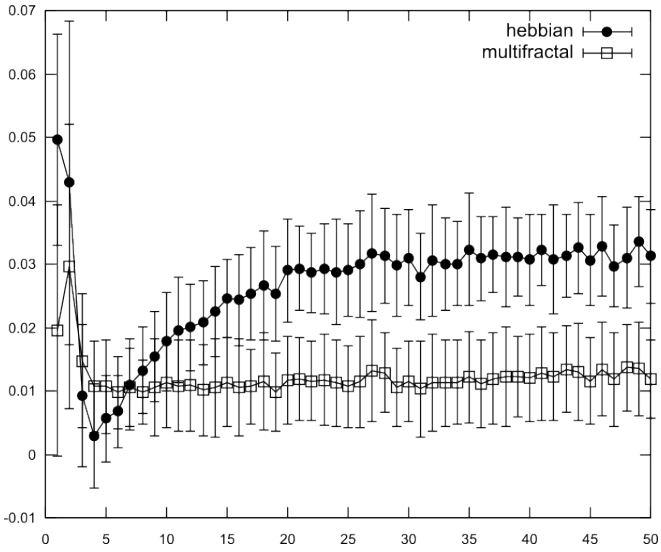

Fig. 4. This is exactly the same setup as in Figure 3, but on real data from the Proben1 data set. Unlike the situation in Figure 3, there is a clear difference between the two learning rules. Moreover, Figure 3 indicates a decrease in the gain on the base version, due to the fact the learning process is converging (see Figures 1 and 2). On the real data case, no such gain reduction is observed.

presented in this study show that such learning can improve the original setup processing capabilities.

The Hebbian learning rule was abstracted to isolate the main elements:

- Consideration of the dynamical spiking behavior of the neurons
- Learning using a synchronization mechanism
- Frustration (competitive learning) preventing the apparition of a global ordered state

These considerations are extended so as to generalize the Hebbian learning rule to a whole class of algorithms based on the neurons dynamical regimes. The main problem then is to find a relevant identifier for these dynamical regimes. Once this is done, a learning rule can be derived thanks to synchronization and frustration, with the methodology given in section II.A. Ideally, the learning rule would take in account not only the instantaneous time properties of neurons, but their whole dynamical behaviors.

Experiments were conducted considering the multifractal spectrum of the neuron inter-spike time series as a rough dynamical regime identifier candidate. The learning rule derived from this hypothesis gives results comparable to the Hebbian learning rule in practice.

A possible extension to this study would be to find a better regime identifier, potentially including some inter-neuron time information like the Hebbian rule does. The multifractal analysis captures a completely different kind of information than afferent-efferent spike time delta. Each neuron is in this case concerned only by its own dynamics. Given that results are on par with the Hebbian learning rule while using different information, a promising extension would be to mix both approach to define a better identifier taking more information into account.

In any case, the notion of identifying and synchronizing the neuron behaviors that was inherent to the Hebbian learning rule can be extended to consider more aspects of the neuron dynamical regimes.

Acknowledgment

The spiking neural network simulation framework was provided by the Amygdala project, http://amygdala.sourceforge.net/.

The random number generator is the Mersenne Twister by Makoto Matsumoto, http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html.

All other algorithms and routines are my personal creation. This project source code is available under the GNU General Public License, v2 or above. Links can be found on the author web page http://nicolas.brodu.free.fr. All random seeds and parameters used in the experiments are provided with the source code.

References

[1] 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.

[2] 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.

[3] 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.

[4] S. Bornholdt and T. Röhl, "Self-organized critical neural networks", *Physical Review E*, vol. 67, 066118, Jun. 2003.

[5] T. Natschläger, N. Bertschinger, and R. Legenstein, "At the edge of chaos: real-time computations and self-organized criticality in recurrent neural networks", in *Proc. Advances in Neural Information Processing Systems*, Dec. 2004.

[6] H. Bersini, "The frustrated and compositional nature of chaos in small Hopfield networks". *Neural Networks*, vol. 11, num. 6, pp. 1017-1025, Aug. 1998.

[7] N. Brodu, "Real-time update of multi-fractal analysis on dynamic time series using incremental discrete wavelet transforms", submitted for publication, Nov. 2005.

[8] J. F. Muzy, E. Bacry, and A. Arneodo, "Multifractal formalism for fractal signals: The structure-function approach versus the wavelet-transform modulus-maxima method", *Physical Review E*, vol. 47, num. 2, pp 875-884, Feb. 1993.

[9] P. Manimaran, P. K. Panigrahi, and J. C. Parikh, "Wavelet analysis and scaling properties of time series", *Physical Review E*, vol. 72, 046120, Oct. 2005

[10] W. Maass, T. Natschläger, and H. Markram "Computational models for generic cortical microcircuits" in *Computational Neuroscience: A Comprehensive Approach*, J. Feng, Ed. ch. 18, pp 575-605, 2003.

[11] T. Natschläger "Benchmark Tasks for Evaluating the Computational Power of NMCs", unpublished, available online at http://www.lsm.tugraz.at/bmt.html, 2003.

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