Asymmetric Neural Networks
Training and Memory Breakdown

Dorian Bivolaru

Electrical Engineering and Computer Science
Jacobs University Bremen
Campus Ring 4
28759 Bremen
Germany

Type: Guided Research Proposal
Date: May 13, 2007
Supervisor: Dr. Mathias Bode

Executive Summary

The main difference between a computer and a human brain is the complexity of the tasks it can process. Even if a computer is unmatchable at bare mathematical operations and automated tasks, there are things which even a child can do better and faster (recognizing objects, faces, and so on) that even the most advanced AI system running on the world's faster supercomputer cannot match. The brain has a lot of other features which would be desirable in artificial systems: robustness, fault tolerance, flexibility, handling of fuzzy / probabilistic / noisy / inconsistent data, highly parallel and small, very compact and with low power dissipation.
The concept of neural networks, to come up with a different computational paradigm than the one introduced by von Neumann (used as basis for almost all machines to date), was conceived as to mimicking nature itself. As the name implies it is inspired from neuroscience but it does not try to be biologically realistic in detail. This discipline combines different areas like systems theory, statistical physics, network theory, probabilistics and dynamical systems but it's applications do not lie only in computer science / engineering but in diverse areas like sensory / cognitive psychology, military intelligence, industrial analysis and basically any area in which different elements interact with each other in a known / unknown way such that the final result of the system is modeled in a predictable way and can even, by specialized analysis, be influenced with very small interventions in it's components so as to obtain a different result.

1. Summary

The dynamics of stochastic asymmetric neural networks is simulated in a MATLAB framework. Temperature and number of patterns is varied such that different phases of the network are observed. A new state for compound ensemble spin glass states is found. Evaluation of such states shows that the compound state that superimposes over the normal spin glass state is always opposed to the former (ie they have opposite eigenvalues), resulting in oscillation-spin-glass-states (OSGS).

2. Statement and Motivation of Research

As the name implies, a neural network was first aimed at simulating brain behavior and connections, highly motivated by the possibility of making artificial computing networks. But the models used are very simplistic in nature so that we will not be lost in unimportant details which don't pertain to the total system behavior of such a network. There are two types of learning mechanisms in these networks: **supervised** and **unsupervised**. We will study the latter, as we are interested in the basic functionality.

McCulloch and Pitts [26] proposed a simple model of a neuron as a binary threshold unit which they proved that in synchronous associations of such neurons, they are capable of **universal computation** for suitably chosen weights. The model computes a weighted sum[1] of it's inputs and then passes it through a Heaviside step function:

$$n_i(t+1)=g\left(\sum_{j=1}^{N} w_{ij}\, n_j(t)-\theta_i\right) \tag{1}$$

The above example is more general, making use of a general **activation function** *g*, which in the case of McCulloch and Pitts was the Heaviside step function. The n_i's represent the neuron's state at time *t* or *t*+1 and the $\theta ' s$ are the **threshold** values.

Updates of the neuron states can either be done **synchronously** using a central clock to

1 *Unless otherwise noted we will be using normal alphabet letters for neuron indexes (going from 1 to N) and greek letters for the stored patterns (going from 1 to p) where N is the number of neurons and p the number of (stored) patterns.*

update them at the same time, or **asynchronously** by either going randomly over the neuron array or setting a probability of update for each individual neuron (both asynchronous methods being equivalent).

Later, Hopfield [13] devised, using a similar model called the **Hopfield model** (but instead of a general *g* function, he uses the *sgn* function; i.e. the outputs are either *-1* or *+1* whereas for the McCulloch-Pitts neuron, they were either *0* or *+1*), a **content-adressable** memory, insensitive to small errors in the input pattern that can, when presented with a pattern, recall the closest match of the known stored patterns. This *storage* is done off-line using the Hebb-rule [27] weights (i.e. (2) in the case of short-time connections) whereas a plus-one rule (3), in the case of long-time connections, would be used later for the asymmetric case.

$$w_{ij}^{S}=\frac{1}{N}\sum_{\mu=1}^{p}\xi_i^{\mu}\xi_j^{\mu} \tag{2}$$

$$w_{ij}^{L}=\frac{1}{N}\sum_{\mu=1}^{p}\xi_i^{\mu+1}\xi_j^{\mu} \tag{3}$$

Hopfield also introduced an energy function (similar to mechanics) and emphasized the representation of stored patterns as dynamical attractors with an associated basin of attraction. He also managed to find out that the storage **capacity** of a Hopfield network cannot be higher than **p_{max}=0.138 N** for random pattern sets. Of course, orthogonal patterns will give zero crosstalk, but using too many such patterns (~N) would defeat the purposes of an associative memory as the weights would all be $w_{ij}=\delta_{ij}$ and the network becomes stable in any state.

The normal states (called **retrieval states**) are not the only possible states. There are also **reversed states** (which occur when more than half of the input bits are in a reversed state), but also **mixture states** which are not equal to any single pattern but correspond to linear combinations of an odd number of patterns [1]. If we increase the number of stored patterns we will obtain also local minima that are not correlated with any finite number of the original patterns. These are called **spin glass states** because of the similarity with sping glass models in statistical mechanics. All of the unwanted states are called generically just **spurious states.**

There was also an effort in completing the model so that it's fully equivalent with magnetic (spin-glass) models by inserting also **thermal fluctuations**. More specifically, the activation function is replaced from a *sgn* to a fully fledged Fermi distribution (4) function (see Glauber dynamics [27]) rescaled for [-1..1].

$$g(x) = \frac{1-e^{-2\beta x}}{1+e^{-2\beta x}} = \tanh(\beta x) \tag{4}$$

Such networks are called **stochastic networks** and are usually equivalent to using random thresholds in (1). Such networks are analyzed using mean field theory [1] and it was found that the storage capacity for them is also $\alpha_c \approx 0.138$ although this is obtained in the low-temperature limit (as it should be expected). There are 4 possible phase-states for such a network called A, B, C and D. States A represent error-free retrieval states, whereas B represent all the other retrieval states. In state C (high temperature), the network shows only spin glass states, but these are not correlated with any of the stored patterns. If the temperature is raised more, then these states melt as well and the only mean field solution is zero [1].

The problem of correlated patterns which limit the network capacity even for *p<<N* was solved [14][21] by changing the Hebbian rule to use the pattern overlap matrix **Q** in what is called a **pseudo-inverse rule.**

$$\begin{gathered} w_{ij} = \frac{1}{N} \sum_{\mu\nu} \xi_i^{\mu} Q^{\mu\nu} \xi_j^{\nu} \\ where,\ Q_{\mu\nu} = \frac{1}{N} \sum_{i=1}^{N} \xi_i^{\mu} \xi_i^{\nu} \end{gathered} \tag{5}$$

Such a construct was shown [14] to have almost the same p_{max} capacity in practice, but from the biological point of view, it presents non-locality whereas the Hebbian rule would imply interaction only at the pre- and post-synaptic levels, like it would be expected from a real neuron.

Until now we have looked into generating strong attractors for stable pattern states. But it is also interesting to investigate the possibility of storing, recalling, generating and learning sequences of patterns. Thus our goal changes from stability of one pattern to closed limit cycles such that these sequences are encoded in the choice of connections weights as before. We will investigate whether the pattern sequence storage does remember common features of the stored patterns and how does the stored features relate to common elements in the patterns; is the neural network capable of extracting general information from a class of patterns.

Hopfield suggested [13] that a possibility would be using **assymetric connections**, more specifically,

$$w_{ij}=\frac{1}{N}\sum_{\mu=1}^{p}\xi_i^\mu\xi_j^\mu+\frac{\lambda}{N}\sum_{\mu=1}^{p}\xi_i^{\mu+1}\xi_j^\mu \tag{6}$$

to produce such sequences of patterns. Here λ is a **asymmetry constant** that governs the relative strength between symmetric and asymmetric terms. For the case of correlated patterns, the Hebb rule (6) may be replaced by a pseudo-inverse rule similar to (5).

$$w_{ij}=\frac{1}{N}\sum_{\mu\nu}\xi_i^\mu Q^{\mu\nu}\xi_j^\nu+\frac{\lambda}{N}\sum_{\mu\nu}\xi_i^{\mu+1}Q^{\mu\nu}\xi_j^\nu \tag{7}$$

Unfortunately these schemes proved unusable in practice. The synchronous updating tends to de-phase the system so that one obtains states that overlap several consecutive patterns, and the sequence is soon lost. Only if the length of the sequence is very small ($p<<N$), the limit cycles are embedded successfully [18]. One of the focuses of our research is finding a good method to train such an asymmetric network efficiently.

There were some attempts [2][3] at tackling this problem using sparse patterns on a stochastic network with additional terms in (6) to inhibit transitions to pattern states that were not the next ones in sequence. The transitions themselves occurred at relatively high temperatures (being driven to the next state by thermal noise), but the delays were unpredictable. Also it was investigated [24][16][20] that by using dynamic weights (more specifically just changing the λ to grow linearly as to provoke the next transition) it is possible to obtain a robust scheme but with careful selection of the constants (delay and asymmetry). In the case of using an exponential-based kernel for the moving-average of the delayed response, it was shown [22] that the system acquires chaotic behavior for a large asymmetry constant. There were reports [17] of using this for successfull modeling of the **central pattern generator** of the mollusk *Tritonia diomedea.*

In the case of delta-based kernels, the long time connections simply reduce to **delayed**

synapses that pass a given signal after a time delay. In general this amounts to using a Hebbian-type learning scheme with temporization:

$$w_{ij}^{L}=\frac{1}{N}\frac{1}{T}\int_{0}^{T}\xi_i(t+\tau_{ij})\xi_j(t)\,dt \tag{8}$$

which was shown [4][12][15] that can "lea rn" even sequences with different transition times than the synaptic delay time. This would imply a range of delay times in the network, with a broad distribution so as to cover the relevant time scales of the input sequence; short-time connections would be superfluous for they are included in the model as the zero delay term. The connections that become strong are those in **resonance** with the sequence itself as shown in [7][25]. Another focus of our research would be to analyze whether equations (6),(7)&(8) can be used in a similar analysis as in [1], for finding out an equivalent phase diagram of an asymmetric network as well as the storage capacity of such a network and any presence of spurious states. Simulations will be performed to analyze the change in system behavior when orthogonal pattern overloading occurs and see if, similar to the symmetric case, spin glass states are formed, in both stochastic and non-stochastic scenarios.

3. Work documentation

The working framework was implemented in MATLAB as depicted in the following schematic.

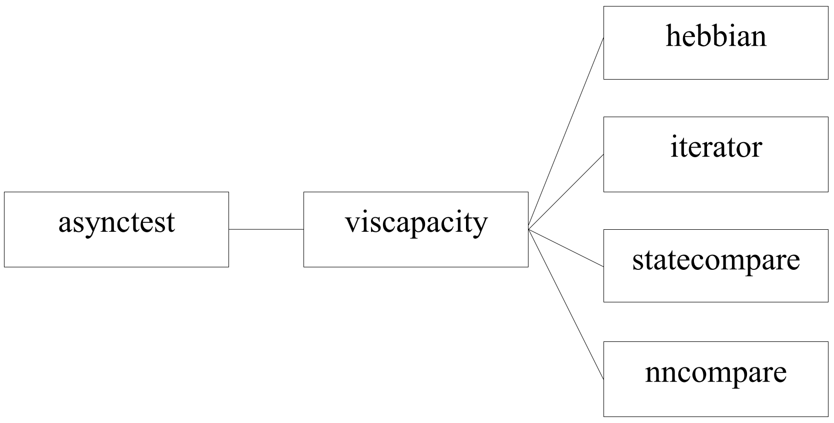

FIG. 1 – Module list for working framework

Each module is implemented in a different m-file as a separate function. Following there is a small description for each one:

hebbian	Implements the learning rule for short/long connections.
iterator	Stochastic and delayed neural network simulator.
statecompare	Calculate overlap between two given states.
nncompare	Calculate overlap between given state and memorized patterns.
viscapacity	Generate enforced random learning patterns and simulate the network.
asynctest	Call viscapacity with different parameters.

TABLE 1 – Module descriptions

3.1 System details

We will present shortly how we arrived at the current framework configuration. First of all, we needed to analyze a **stochastic** system, and for this we would need a lot of simulations to obtain the *average* values $<S_i>$ from different S_i measurements. Instead we can use **continuous valued units** with the activation function (4) which is completely equivalent as the formulas are identical $<S_i> = V_i$.

$$V_i = g(u_i) = g(\sum_j w_{ij} V_j) \tag{9}$$

The weights are trained using the Hebb learning rule as explained above [24][16] where the long connections (3) represent slow synapses that have delayed or sluggish response. More precisely, the input $h_i(t)$ to a unit is given by:

$$h_i(t) = \sum_k \left[w_{ij}^S S_j(t) + \lambda w_{ij}^L \overline{S_j}(t) \right] \tag{10}$$

Where the delayed response $\overline{S_j}(t)$ is a weighted moving average (memory trace) over past values of S_j:

$$\overline{S_j}(t) = \int_{-\infty}^{t} G(t-t')S_j(t')dt' \tag{11}$$

For simplicity of implementation we will use a delta function as the G kernel which will amount to a shifted equation similar to (8):

$$w_{ij}^L = \frac{1}{N} \sum_{t=0,\tau,2\tau,\dots} \xi_i(t+\tau)\xi_j(t) \tag{12}$$

Of course considering that we have a finite set of training patterns, and considering the wanted sequence of pattern generation we can say that $\xi_i(t)=\xi_i(\mu\tau)=\xi_i^\mu$ and thus the actual equation for the w^L_{ij}'s reduces to (3). The only other thing we need now is a nice form for (11) which is straightforward using the delta kernel $G(t)=\delta(t-\tau)$:

$$\overline{S_j}(t)=S_j(t-\tau) \tag{13}$$

And thus we obtain the final formula used in the *iterator* module considering stochastic averaging as explained above. Discretizing the system in time up to a scaling constant we obtain:

$$V_i[k+1]=\tanh\left(2\beta\sum_k\left[w^S_{ij}V_j[k]+\lambda\, w^L_{ij}V_j[k-\tau]\right]\right) \tag{14}$$

3.2 Comparison method

For analyzing the simulation results using our built neural network we use three measuring parameters: pattern overlap (m_μ), maximum overlap (m) and mean square non-retrieve overlap (r). We will explain their properties shortly.

$$m_\mu=\frac{1}{N}\sum_i \xi_i^\mu V_i \tag{15}$$

Of course m is just the biggest m_μ for all µ. Let υ be this pattern with biggest overlap as computed with the formula above. Then, by definition:

$$r=\frac{N}{p}\sum_{\mu\neq\nu} m_\mu^2 \tag{16}$$

Considering random patterns, the expected values (supposing we are retrieving pattern α) are:

m_μ, $\mu \neq \alpha$	$\sim \frac{1}{\sqrt{N}}$
$m = m_\alpha$	~1.00
r	~1.00

TABLE 2 -Pa rameter expected values for optimal retrieval

Ideally, if we get back the pattern that we want, almost error-free, then our parameters should take the values in Table 2. Glancing at (15), we can note that the overlap between a normal state and it's corresponding reversed state amounts to *-1.00* whereas to obtain a zero, we need half of the bits to match and half of them to be reversed and of course for two identical patterns the result is 1.00.

The *m*'s are computed in *statecompare*/*nncompare* whereas *r* is computed inside *viscapacity*.

4. Results

Different sets of simulations have been done using N=100, λ=2 and τ=8 for different number of patterns *p* and temperatures *T* (different *β*). One example (2a) follows with the desired operation of the system as limit cycle memory.

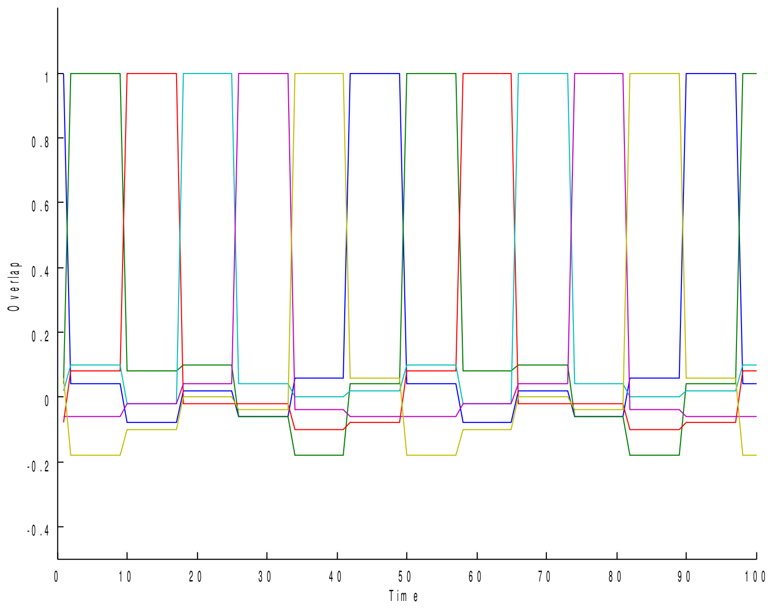

FIG. 2a – Norm al system operation; p = 6; T=0;

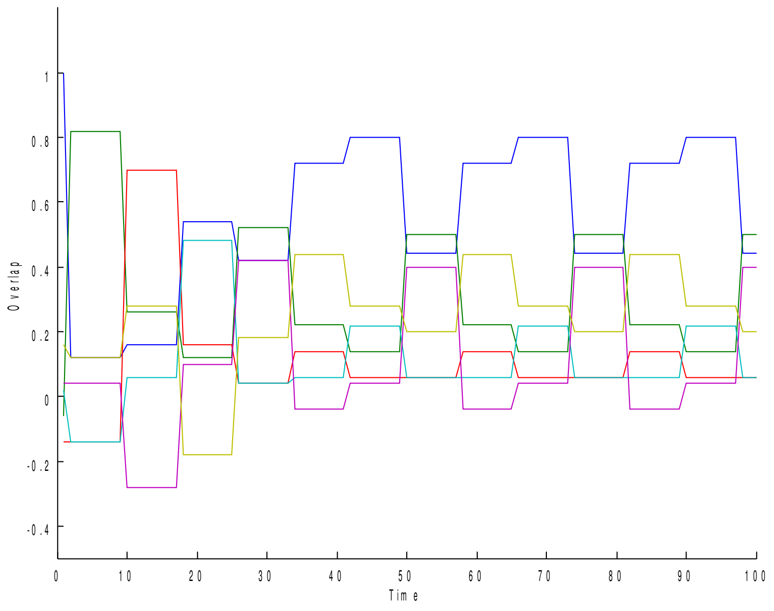

FIG. 2b – Mixed-mode operation; p = 6; T=0;

As it can be seen on Fig. 2a, the system evolves in time by switching from pattern 1 through 6 in order 1->2->3->4->5->6. On Fig. 2b, the network prefers, after a short relaxation period, a mixture-state oscillation between three such mixed states (MS).

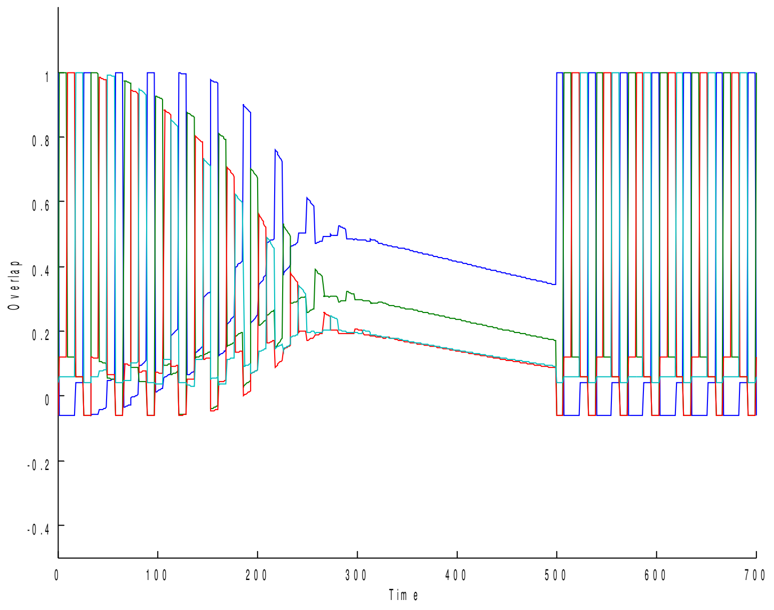

FIG. 3a -Ti me-varying temperature system;

p = 4; β=166/t for t<500 and β=200 in rest.

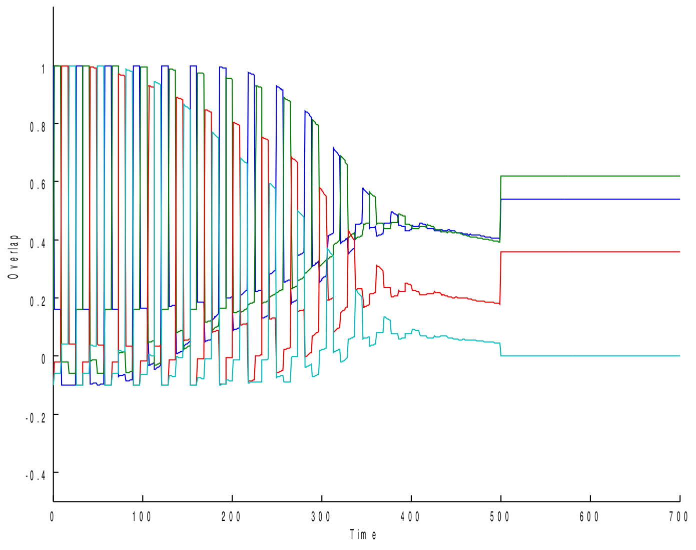

FIG. 3b – Time-varying temperature system;

p = 4; β=233/t for t<500 and β=200 in rest.

Next we have a time-varying temperature simulation. The system is heated up linearily in time with different slopes. After 500 time ticks, the system is switch to zero-freezing. Two cases are shown (Fig. 3a, 3b). In the first case, the system returns to normal cycle generation (NS), whereas in the second case, while rapidly cooled, the system switches to a non-oscillating spin-glass state (SGS). Taking measurements from the graph we notice that both systems are error-free until around t=40 (which corresponds to T=0.17-0.24) when they start decaying exponentially (EDNS – exponentially decayed normal state) (still keeping the cycle order intact though). The systems transitions eventually to a mean-field-zero (MFZ) solution decay at approximatively T=1.9 loosing all it's correlation with any of the pattern states.

We note here that temperature values compared to the symmetric case are not equivalent as this temperature is scaled depending on the normalization constant(s) present in the learning rule of the system. In our case, these temperature values depend on λ as it shows up in the weights formula and thus are valid only for λ=2.

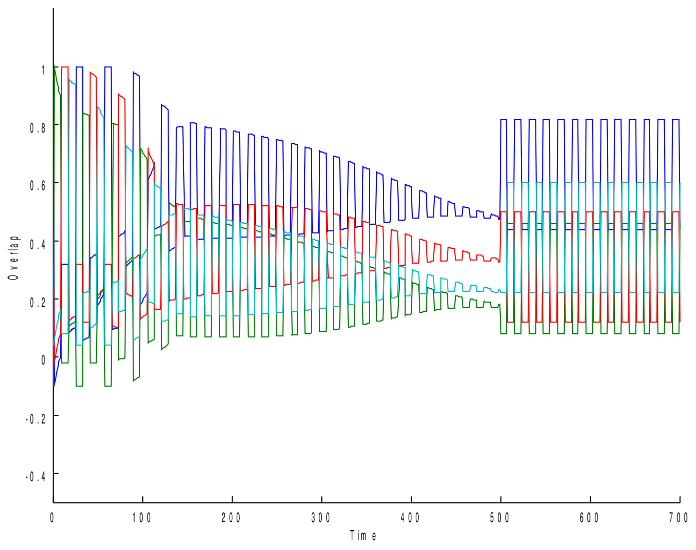

FIG. 4 -Formation of (ED)OSGS states.

p = 4; β=200/t for t<500 and β=200 in rest.

Here we can observe an interesting phenomenon. First the system is in a EDNS but t=140 (T = 0.7) the oscillation order is completely lost, going into what we will call an exponential decayed oscillating spin glass state (EDOSGS). The main difference between an EDNS and an

EDOSGS is the *loss* of cycle corellation. Of course, while temperature increases, the decay will probably switch to a MFZ state where pattern correlation is lost as well. Interestingly enough, when the temperature is switched back instantly to zero-freezing point, the system instead of going into an SGS as expected, it starts to oscillate around it. We will call these states oscillating-SGS or OSGS.

These OSGS states are a novelty, but this phenomenon is not unique to asymmetric networks. As we will show, this can occur in symmetric networks all the same, but it is very hard to separate it's effects from a normal SGS, as symmetric networks are static (not time-dependent). We will note that compared to a NS which repeats itself every $p\tau$ time ticks, an OSGS repeats always every 2τ. The reason is quite simple. Around the SGS point, a limit cycle forms. Because a pattern can be related to this SGS point only in two modes (as either *up* or *down* spin) and assuming that this point is a much stronger attractor than any of the other patterns (as explained by mean field theory for $T>T_c$), it follows that we will have our pattern correlation switch from up to down and down to up as fast as possible which for us means it is done in just one τ. An interesting question is why do these OSGSs apparently have the same amplitude. Could it be that OSGSs have a constant radius basin around them where the oscillations are possible? This could be a nice theme for future investigations.

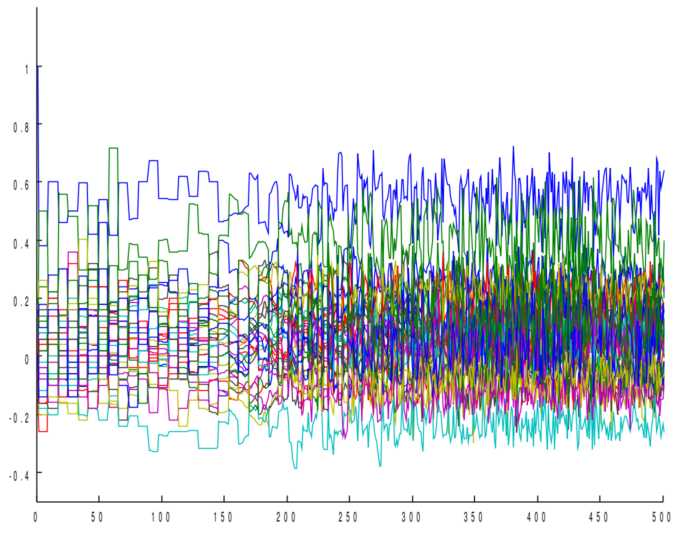

FIG. 4 -Pattern saturation chaotic evolution;

p = 30; T=0;

Another interesting development is pattern saturation. For α>>0.14 the system starts to behave eratically, full random separation of the pattern correlations around a SGS is obtained combined with relative temperature increases. If the networks is simulated at (T~0) then no chaotic behavior is observed; only a lot of different mixed states. The spectrum of such a random signal was analyzed using FFT and was found not to contain any specific frequency content distinguishable from the noise floor. We arrived at the conclusion that the system behaves as a white noise generator, although such approaches have already been analyzed more thoroughly [30].

4.1. Theoretical classification

As we have shown before, we have found a few types of states: (ED)NS, MS, (ED)OSGS, SGS, MFZ. Considering equation (14) and Fig. 2a we can infer two rules for the dynamics of the system. One for constant state, and the other for the transition:

$$\xi_i^{\mu}=g\left(\sum_j w_{ij}^S \xi_j^{\mu}+\lambda w_{ij}^L \xi_j^{\mu-1}\right) \tag{17}$$

$$\xi_i^{\mu+1}=g\left(\sum_j w_{ij}^S \xi_j^{\mu}+\lambda w_{ij}^L \xi_j^{\mu}\right)=g\left(\sum_j w_{ij}\, \xi_j^{\mu}\right) \tag{18}$$

It is easy to notice that for a cycle of length *p*, we have the identity:

$$\overline{\xi^{\mu}}=L^p \overline{\xi^{\mu}} \tag{19}$$

Where we defined the L *non-linear* operator as $L_{ij}\equiv g\, w_{ij}$. As such we will extend the definition of eigenvector/eigenvalue to non-linear operators as well. Therefore the NS are the "eigenvec tors" of L_{ij}^p that have "eigenvalu e" of 1 or more specifically the generalized eigenvectors of algebraic multiplicity p.

The SGS are those states that are constant in time (we will note them with G):

$$G_i^\mu = g\left(\sum_j w_{ij} G_j^\mu\right) \tag{20}$$

$$\overline{G^\mu} = L\overline{G^\mu} \tag{21}$$

Please note that here we used overlined vectorial notation. The SGS are the normal eigenvectors of L with eigenvalue 1. What we can observe is that any SGS Gµ also satisfies the NS equation. From here we can prove the existence of the OSGS which is an ensemble of SGS+NS that can exist (as an EDOSGS) above the critical temperature as opposed to normal NS.

$$L\overline{G^p} + \gamma L\overline{\xi^\nu} = \overline{G^p} - \gamma\overline{\xi^\nu} \tag{22}$$

$$L\overline{\xi^\nu} = -\overline{\xi^\nu} \tag{23}$$

Comparing (23) with (19) we arrive at the conclusion that for any NS present in an OSGS ensemble, we have that $(-1)^p = 1$ => p=even which is proven experimentally not to be correct; there are OSGS states for any (small) p, may it be even or odd. It follows that there should be a linear combination of patterns such that it forms a "-1" eigenvector. Note that because L is not linear, we cannot do superposition (distribute the sum) for L.

$$L\sum_\mu \gamma_\mu \overline{\xi^\mu} = -\sum_\mu \gamma_\mu \overline{\xi^\mu} \tag{24}$$

5. Conclusions

We have found 5 types of states for an asymmetric neural network:

- Normal States (NS) – representing the useful operation of the neural network as a central pattern generator; EDNS represent the stochastic version of NS where temperature affects the retrieval of states exponentially.
- Mean Field Zero (MFZ) – at high temperatures, as expected, the activation function is scaled that much, as to have the system evolve exponentially towards zero with the temperature increase.
- Mixed States (MS) – a linear combination of NS that have a stronger attractor than each individual state.
- Spin Glass States (SGS) – when the system is heated beyond critical temperature and then frozen abruptly, states appear that have no or little relation to the initial states.
- Osciallating Spin Glass States (OSGS) – sometimes, when frozen, the system does not go to a stationary SGS, but converges to a limit cycle around that SGS, the oscillations being correlated with the stored patterns, but not independently, similar to an ensemble that orients itself with opposite spin as the SGS.

References

[1] Amit, D. (1988). Neural Networks for Counting Chimes. *Proceedings of the National Academy of Sciences, USA* **85**, 2141-2145.

[2] Buhmann, J. and K. Schulten (1987). Noise-Driven Temporal Association in Neural Networks. *Europhysics Letters* **4**, 1205-1209.

[3] Buhmann, J. and K. Schulten (1988). Storing Sequences of Biased Patterns in Neural Networks with Stochastic Dynamics. In *Neural Computers* (Neuss 1987), eds. R. Eckmiller and Ch. von der Malsburg, 231-242. Berlin: Springer-Verlag

[4] Coolen, A.C.C. And C.C.A.M. Gielen (1988). Delays in Neural Networks. *Europhysics Letters* **7**, 281-285.

[5] Crisanti, A. and H. Sompolinsky (1987). Dynamics of Spin Systems with Randomly Assymetric Bonds: Langevin Dynamics and a Spherical Model. *Physical Review A* **36**, 4922-4939.

[6] Crisanti, A., D.J. Amit, and H. Gutfreund (1986). Saturation Level of Hopfield Model for Neural Network. *Europhysics Letters* **2**, 337-341.

[7] Dehaene, S., J.-P. Changeux, and J.-P. Nadal (1987). Neural Networks That Learn Temporal Sequences by Selection. *Proceedings of the National Academy of Sciences, USA* **84**, 2727-2731.

[8] Derrida, B., E. Gardner, and A. Zippelius (1987). An Exactly Soluble Asymmetric Neural Network Model. *Europhysics Letters* **4**, 167-173.

[9] Gutfreund, H. and M. Mézard (1988). Processing of Temporal Sequences in Neural Networks. *Physical Review Letters* **61**, 235-238.

[10] Hertz, J.A., G. Grinstein, and S. Solla (1986). Memory Networks with Asymmetric Bonds. In *Neural Networks for Computing* (Snowbird 1986), ed. J.S. Denker, 212-218. New York: American Institute of Physics.

[11] Hertz, J.A., G. Grinstein, and S. Solla (1987). Irreversible Spin Glasses and Neural Networks. In *Heidelberg Colloquium on Glassy Dynamics* (Heidelberg 1986), eds. J.L. van Hemmen and I. Morgenstern, 538-546. Berlin: Springer-Verlag.

[12] Hertz, A., B. Sulzer, R. Kűhn, and J.L. van Hemmen (1989). Hebbian Learning Reconsidered: Representation of Static and Dynamic Objects in Associative Neural

Nets. *Biological Cybernetics* **60**, 457-467.

[13] Hopfield, J.J. (1982). Neural Networks and Physical Systems with Emergent Colective Computational Abilities. *Proceedings of the National Academy of Sciences, USA* **81**, 3088-3092. Reprinted in Anderson and Rosenfield [1988].

[14] Kanter, I. And H. Sompolinsky (1987). Associative Recall of Memory Without Errors. *Physical Review A* **35**, 380-392.

[15] Kerszberg, M. and A. Zippelius (1990). Synchronization in Neural Assemblies. *Physica Scripta* **T33**, 54-64.

[16] Kleinfield, D. (1986). Sequential State Generation by Model Nerual Networks. *Proceedings of the National Academy of Sciences, USA* **83**, 9469-9473.

[17] Kleinfield, D. and H. Sompolinsky (1989). Associative Network Models for Central Pattern Generators. In *Methods in Neuronal Modeling: From Synapses to Networks*, eds. C. Koch and I. Segev, 195-246. Cambridge: MIT Press.

[18] Nishimori, H., T. Nakamura, and M. Shiino (1990). Retrieval of Spatio-Temporal Sequence in Asynchronous Neural Network. *Physical Review A* **41**, 3346-3354.

[19] Parisi, G. (1986). Asymmetric Neural Networks and the Process of Learning. *Journal of Physics A* **19**, L675-L680.

[20] Peretto, P. and J.J. Niez (1986). Collective Properties of Neural Networks. In *Disordered Systems and Biological Organization* (Les Houches 1985), eds. E. Bienenstock, F. Fogelman-Soulié, and G. Weisbuch, 171-185. Berlin: Springer-Verlag.

[21] Personnaz, L., I. Guyon, and G. Dreyfus (1986). Collective Computational Properties of Neural Networks: New Learning Mechanisms. *Phyiscal Review A* **34**, 4217-4228.

[22] Riedel, U., R. Kűhn, and J.L. van Hemmen (1988). Temporal Sequences and Chaos in Neural Nets. *Physical Review A* **38**, 1105-1108.

[23] Sompolinsky, H. (1987). The Theory of Neural Networks: The Hebb Rules and Beyond. In *Heidelberg Colloquium of Glassy Dynamics* (Heidelberg 1986), eds. J.L. van Hemmen and I. Morgenstern, 485-527. Berlin: Springer-Verlag.

[24] Sompolinsky, H. and I. Kanter (1986). Temproal Association in Asymmetric Neural Networks. *Physical Review Letters* **57**, 2861-2864

[25] Touluse, G., S. Dehaene, and J.-P. Changeux (1986). Sping Glass Model of Learning by Selection. *Proceedings of the National Academy of Sciences, USA* **83**, 1695-1698.

[26] McCulloch, W.S. and W. Pitts (1943). A Logical Calculus of Ideas Immanent in Nervous

Activity. *Bulletin of Mathematical Biophysics* **5**, 115-133. Reprinted in Anderson and Rosenfield [1988].

[27] Hebb, D.O. (1949). *The Organization of Behavior.* New York: Wiley. Partially reprinted in Anderson and Rosenfeld [1988].

[28] Glauber, R.J. (1963). Time-Dependent Statistics of the Ising Model. *Journal of Mathematical Physics* **4**, 294-307.

[29] Jinho , K. and T. Sato (1996). Analysis of Periodic Attractor in a Simple Hysteresis Network. *IEICE Transactions on Fundamentals*, vol. **E79-A**, N0.6 JUNE 1996.

[30] Yang, X.-S. and Y. Huang (2006). Complex Dynamics in Simple Hopfield Neural Networks, *Chaos* **16**, 033114

Appendix A – A few interesting simulations

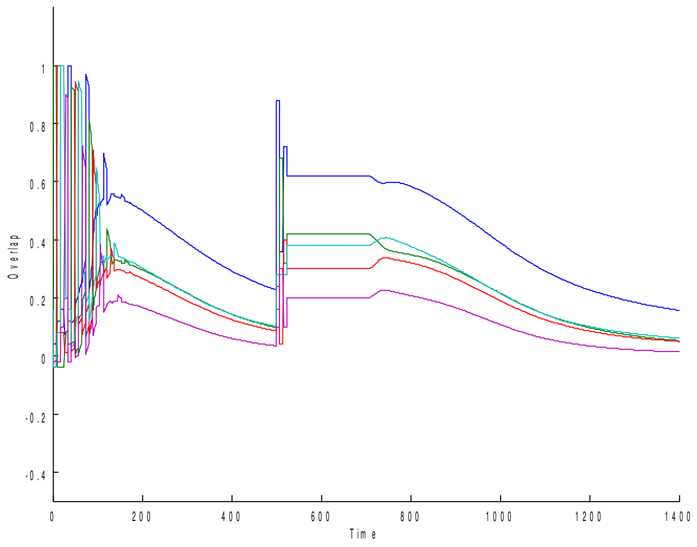

FIG. 5 –Hea ting –f reezing; Heating freezing;

p = 5; T = t/100; T=200 for [500,700]

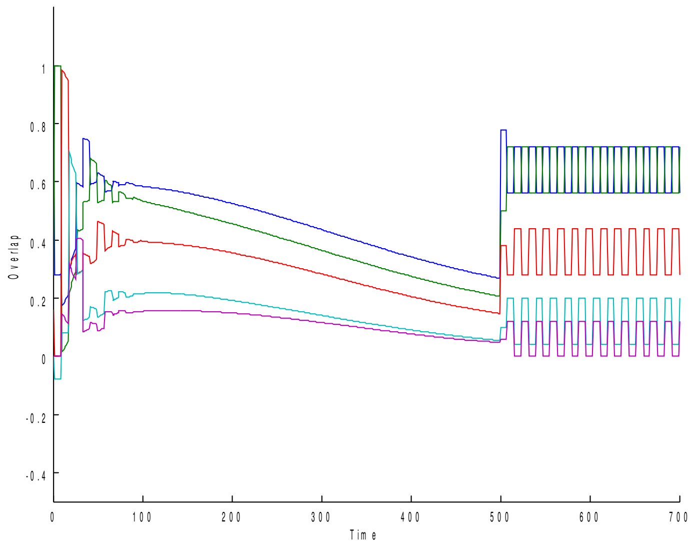

FIG. 6 – Another example of OSGS; p = 5; T = t/100;

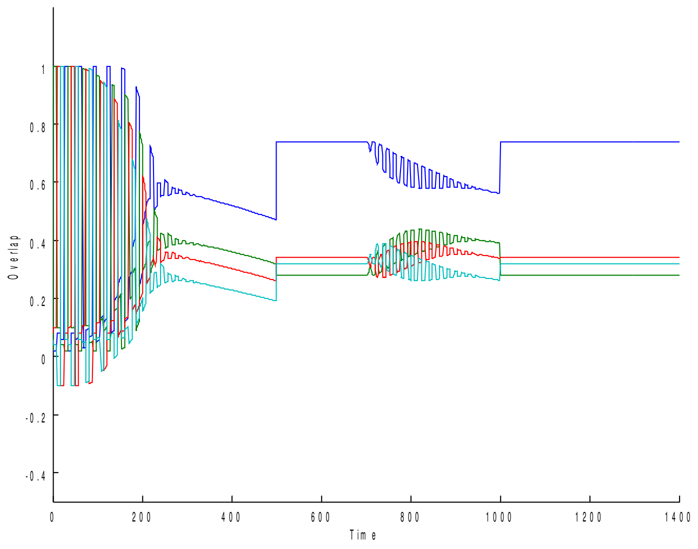

FIG. 7 – Transition SGS->EDNS->SGS; p=4; T = t / 220;

T=200 for [500,700] and [1000, 1400]

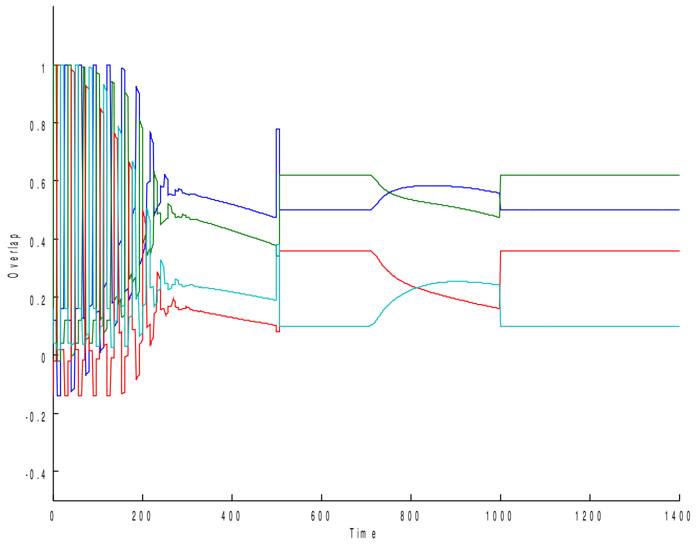

FIG. 8 – Transition SGS->MFZ->SGS; p=4; T = t / 220;

T=200 for [500,700] and [1000, 1400]

Appendix B – The code

```
%%%
%%% Hopfield Neural Network; Symmetric Hebbian Learning
%%%

% Input:  n - number of neurons; train - matrix with training patterns
%         long - long connection coefficient lambda
% Output: w - weights matrix
function [w_s, w_l] = hebbian(n, train)
    % Number of patterns p
    [p, q] = size(train);

    % Sanity check
    if (q ~= n)
        error('hebbian: Given training patterns have wrong dimensions.')
    end

    % Fill in weights
    %fprintf(1, 'hebbian: Learning... ');
    w_s = zeros(n, n);
    w_l = zeros(n, n);
    for i = 1:n
        for j = 1:n
            tmp_s = 0;
            tmp_l = 0;
            for k = 1:p
                t = mod(k, p) + 1;
                tmp_s = tmp_s + train(k, i) * train(k, j);
                tmp_l = tmp_l + train(t, i) * train(k, j);
            end
            w_s(i, j) = tmp_s / n;
            w_l(i, j) = tmp_l / n;
        end
    end
    %fprintf(1, 'DONE\n')
end

%%%
%%% Hopfield Neural Network; Iterator
%%%

% Input:  n - number of neurons; w - matrix with weights(s/l);
%         g - activation function; bgn - initial neuron states
%         beta - stochastic squashing; lambda - long coeff.
% Output: hnn - output neuron states
function hnn = iterator(n, w_s, w_l, bgn, prev, beta, lambda)
    % Sanity Check
    [x, y] = size(w_s);
    [xx, yy] = size(w_l);
    [t, q] = size(bgn);
    [t, qq] = size(prev);
    if ((x ~= n) || (y ~= n) || (q ~= n) || (qq ~= n) || (xx ~= n) || (yy ~=
n))
        error('iterator: Given data has wrong dimensions')
    end

    hnn = bgn;
    hash = randperm(n);
    for i = 1:n
        %i = hash(k);
        s = 0;
        l = 0;
        for j = 1:n
            s = s + w_s(i, j) * hnn(j);
            l = l + w_l(i, j) * prev(j);
        end
        s = s + lambda * l;
        hnn(i) = g(beta, s);
    end
    %fprintf('\niterator: Convergence steps: %d\n\n', step)
end

%Activation function
function y = g(beta, x)
    y = tanh(beta * x);
end


%%%
%%% Hopfield Neural Network; Compare states
%%%

% Input:  state - one training patterns;
%         hnn - state to compare with
% Output: difference
function dst = statecompare(state, hnn)
    % Number of neurons n
    [x, n] = size(state);
    [y, n2] = size(hnn);

    % Sanity check
    if ((x ~= 1) || (y ~= 1) || (n ~= n2))
        error('statecompare: Wrong dimensions.')
    end

    %dif = abs(state - hnn);
    %dst = sum(dif);
    %dst = n - dst / 2;

    dst = sum(state .* hnn) / n;
end

%%%
%%% Hopfield Neural Network; Compare neural networks
%%%

% Input:  n - number of neurons; train - matrix with training patterns;
%         hnn - long connection coefficient lambda
% Output: w - weights matrix
function cdst = nncompare(n, train, hnn)
    % Number of patterns p
    [p, q] = size(train);
    [x, y] = size(hnn);

    % Sanity check
    if ((q ~= n) || (y ~= n))
        error('nncompare: Given training patterns have wrong dimensions.')
    end

    %fprintf(1, 'nncompare: Comparing... ')
    cdst = zeros(1, p);
    for i = 1:p
```

```
        cdst(i) = statecompare(hnn, train(i,:));
    end

    %fprintf(1, 'PATTERN %d [%d]\n', pat, dst)
end

%%%
%%% Hopfield Neural Network; Assymmetric Simulation with Stochastics
%%%

% Input:  mode - 0/hebbian, 1/psi
% Output: m^mu vs time
function [mc, fc, ret] = viscapacity(p, n, tix, bx)

    TICKS    = tix;     %Number of iterations
    MAX_NEUR = n;    %Number of neurons
    MAX_PAT  = p;      %Number of patterns
    LAMBDA   = 2;      %Long connectivity
    TAU      = 8;      %Time shift; Delta kernel
    RAND_FIN = 1;      %Random levels
    BETA     = bx;   %Stochastic equivalency
    ERR      = 0.001 * n; %Starting error / no. of bit

    %Generate x-axis: time
    time = 1:TICKS;

    %Generate random training patterns
    train = zeros(MAX_PAT, MAX_NEUR);
    for i = 1:MAX_PAT
        for j = 1:1000000000
            tmp = randint(1, MAX_NEUR, RAND_FIN + 1);
            tmp = 2 .* (tmp ./ RAND_FIN) - 1;
            ok = 1;
            break;
            for k = 1:(i-1)
                ddd = statecompare(tmp, train(k, :));
                if (ddd > 0.5)
                    ok = 0;
                    break
                end
            end
            if (ok == 1)
                break
            end
        end
        train(i, :) = tmp;
    end

    %Train the network
    [w_s, w_l] = hebbian(MAX_NEUR, train);

    %Starting pattern
    d = ERR;
    bgn = train(1, :);
    for i = 1:MAX_NEUR
        c = d / (n - i + 1);
        if (rand < c)
            bgn(i) = -bgn(i);
            d = d - 1;
        end
    end
    %Generate data
    data = zeros(MAX_NEUR, TICKS);
    dif = zeros(MAX_PAT, TICKS);
    data(:, 1) = bgn;
    dif(:, 1) = nncompare(MAX_NEUR, train, bgn)';
    for i = 2:TICKS
        if (i < 500)
            f = 100/i;
        elseif (i < 700)
            f = 100;
        elseif (i < 1200)
            f = 100/(i - 700);
        else
            f = 100;
        end
        f = BETA;
        nex = iterator(MAX_NEUR, w_s, w_l, bgn, data(:, getprev(i, TAU))', f,
LAMBDA);
        data(:, i) = nex;
        dif(:, i) = nncompare(MAX_NEUR, train, nex)';
    end

    ret = dif;
    %Le graph
    %hold all;
    %ylim([-0.5 1.2]);
    %for i = 1:MAX_PAT
    %    plot(time, dif(i, :))
    %end
    %hold off;

    %Process; Compute 'r' no-store-overlap
    s = zeros(1, TICKS);
    for i = 1:TICKS
        m = max(dif(:, i));
        s(i) = MAX_NEUR / MAX_PAT * (sum(dif(:, i).^2) - m^2);
        mc(i) = m;
    end
    fc = s;

    %ylim([-0.3 1.2]);
    %plot(time, s)

    %fc = sum(s(800:1400)) / (1400-800);

    %if (qswer > 0.6)
    %    ret = viscapacity(mode, p);
    %end

end

function y = getprev(x, tau)
    if (x - tau < 1)
        y = 1;
    else
        y = x - tau;
    end
end

%%%
%%% Hopfield Neural Network; Assymmetric Graphical Visualization
%%%
```

```
% Input:  mode - 0/hebbian, 1/psi
% Output: m^mu vs time
iter = 10;
n = 100;
pat = n / 5;
tix = n * 2;
beta = 0.5;

s = zeros(1, pat);
qt = zeros(1, pat);
for i = 1:pat;
    a = 0;
    q = 0;
    w = 0;
    for j = 1:iter
        [m, r, b] = viscapacity(i, n, tix, beta);
        a = a + b;
        q = q + max(r);
        w = w + min(m);
        fprintf(1, '.')
        if (mod(i*iter + j, 70) == 0)
            fprintf(1, '\n')
        end
    end
    kk = a ./ iter;
    pp = q ./ iter;
    mm = w ./ iter;
    %s(i) = sum(pp) ./ tix;
    %qt(i) = sum(mm) ./ tix;
    s(i) = pp;
    qt(i) = mm;
end
hold all;
plot(smooth(s))
plot(smooth(qt))
hold off;
%ylim([-0.5 1.2]);
%hold all;
%plot(pp)
%for i = 1:pat
%    plot(kk(i, :))
%end
%hold off;

%hold all
%kkx=kk(3, :);
%m = sum(kkx) / length(kkx);
%kkx = kkx - m;
%plot(kkx)
%fs=fft(kkx(500:1400));
%plot(log(2*abs(fs(1:floor(length(fs)/2)))))
```