Amin Karbasi
E-mail: amin,karbasi@epfl.ch
Amir Hesam Salavati
E-mail: hesam.salavati@epfl.ch

Supervisor: Prof. Amin Shokrollahi
E-mail: amin.shokrollahi@epfl.ch

Algorithmics Laboratory (ALGO)
Ecole Polytechnique Federale de Lausanne (EPFL)

1 Summary

In the past three weeks, we worked a bit on the neural network built from unreliable components. We performed more simulations to assess the performance of the algorithm from different aspects. We also did some theoretical work to have more accurate bounds on the final BER of the algorithm. The details are provided in the sequel.

We also worked on implementing a multi-layer neural network to perform Non-Linear Minor Component analysis (NLMCA). The learning algorithm is finished and working but the recall algorithm still needs some tinkering.

We also continued our simulations on the CIFAR-10 dataset in line with our ICML paper. We have tested yet some new ideas but to no veil (for now) as we still lack a neural network capable of learning and performing the recall operation to the required level both at the same time. We will continue on this direction and test some other ideas with the help of the new master students that have taken the semester projects on this topic.

2 Fault tolerant neural networks

We continued our research on neural networks built from unreliable components, i.e. from neurons with internal noise. We have performed more simulations to investigate various aspects of the algorithm in practice as well as theoretically analyzing the performance. The results for both parts will come in the sequel.

2.1 New results on the effect of noise on the performance of the algorithm

2.1.1 Effect of internal noise on the speed of the algorithm

We continued our empirical investigations on the effect of internal noise on the performance of the error correcting algorithm. To this end, we refined our simulations from last week to compute the number of iterations the algorithm performs to correct a fixed amount of external noise and variable internal noise parameters.

Figure 1 illustrates the number of iterations performed by the neural algorithm for correcting the external errors when ϵ, the fraction of external errors, was fixed to 0.125. The maximum number of iterations had been set to 40. Thus, the corresponding areas in the figure where the number of iterations equals 40 indicate decoding failure.

Figures 2a and 2b are projected version of Figure 1 on two dimensions and show the average number of decoding iterations as a function of υ and ν, respectively.

As evident from the figures, the amount of internal noise drastically affect the speed of the neural algrotihm. For one, internal noise in pattern neurons is necessary for the neural algorithm to be successful, as shown in Figure 2b. The same figure also shows that there is an optimal value for the noise at constraint neurons (around $\nu \simeq 0.25$) for which the number of iterations is minimized. Furthermore, a close inspection of Figure 2a reveal the same phenomenon and around $\upsilon = 0.4$, the number of global iterations seems to reach a minimum. Another interesting behavior uncovered by Figure 2a is that for some values of υ (e.g. $\upsilon = 0.1$) networks with less internal noise at the constraint side are faster in dealing with external errors.

More extensive results and discussions about the new results can be found in [2].

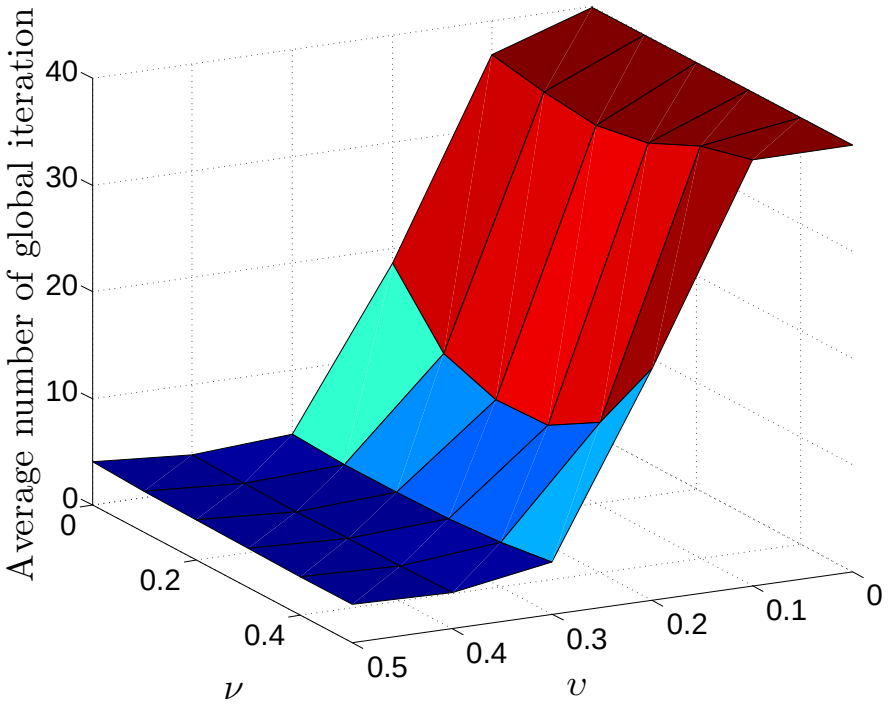
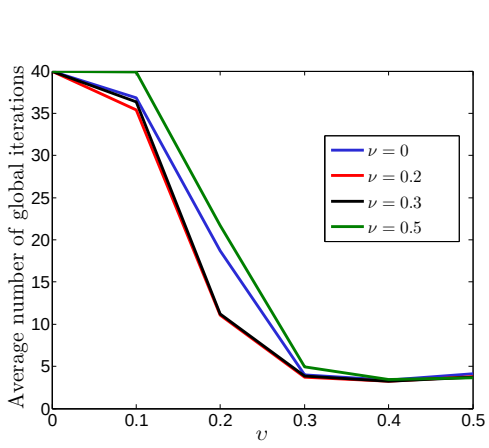
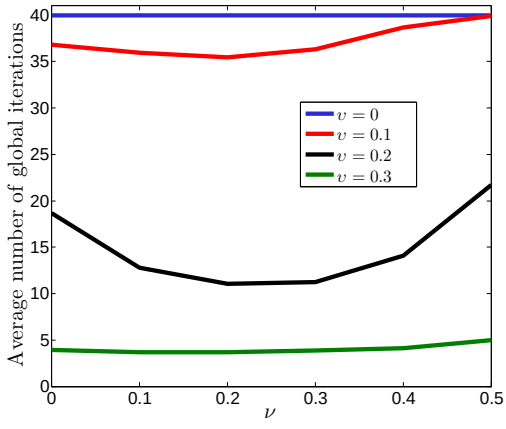


Figure 1: The effect of internal noise on the number of iterations performed by the neural algorithm to correct external noise, for different values of υ and ν with $\epsilon = 0.125$.

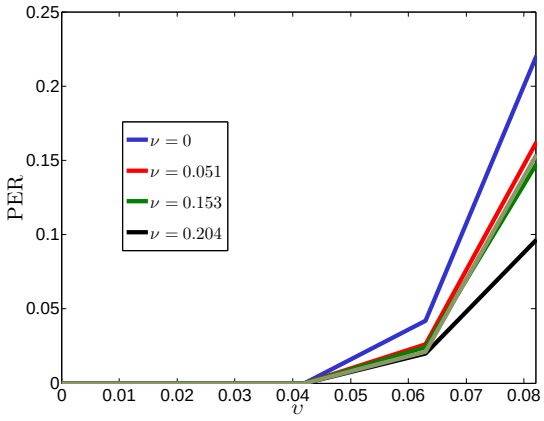


(a) Effect of internal noise at pattern neurons side.

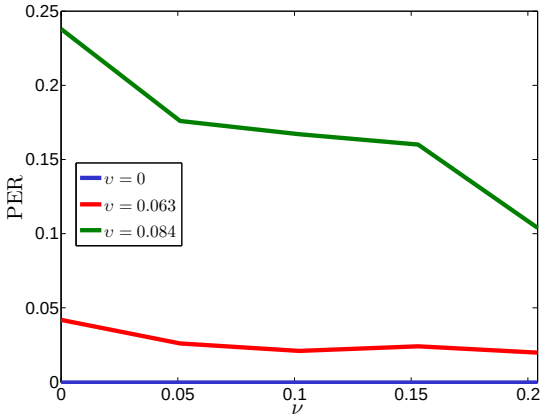


(b) Effect of internal noise at constraint neurons side.

Figure 2: The effect of internal noise on the number of iterations performed by the neural algrotihm to correct external noise, for different values of υ and ν with $\epsilon = 0.125$. The average iteration number of 40 indicate the failure of the neural algorithm.



(a) Effect of internal noise at pattern neurons side.



(b) Effect of internal noise at constraint neurons side.

Figure 3: The effect of the internal noise on final Pattern Error Rate (PER) as a function of υ and ν in absence of external noise.

2.1.2 Performance of the algorithm in absence of external noise

At this point, we are interested in investigating the effects of internal noise on a particular network without external noise. More specifically, we assume $\epsilon = 0$ but $\upsilon > 0$, $\nu > 0$. Furthermore, in contrast to what we have considered so far, the pattern neurons perform one update before starting the error correcting algorithm and a ± 1 noise is added to them with probability υ.[1] Therefore, even if there is no external noise, the pattern neurons could be corrupted due to internal noise parameter υ. The rest of the scenario is the same as before.

Figure 3a and 3b illustrate the results. As evident from the figures, the higher the noise is on the pattern side, the higher PER will be (i.e. the performance deteriorates as a function of internal noise).

Interestingly, this behavior is very similar to the effect of heat on the performance of wireless telegraphy operators observed in [5] (see Fig. 2 in the paper). We see virtually the same trend here is well. The two phenomenon might be related possibly because external heat will translate into neurons with more internal noise.

2.2 Theoeretical analysis

From our new results, we see that the overall BER reduces when

1. the noise on the constraint side is negligible and very close to zero,
2. and the noise at the side of pattern neurons increases (up to some value)

However, the results of our previous theoretical analysis (given in [2]) has a huge gap with that obtained in practice in these circumstances, as illustrated by Figure 4. In this section, we are interested in bridging this gap and obtain a tighter upper bound on the final BER. We will explain various attempts that we have made and their results in the following sections. Note that in all the

[1] Note that this is slightly different from our previous model in which the additive noise will only result in a $\pm$ noise value if the input becomes larger than a threshold φ.

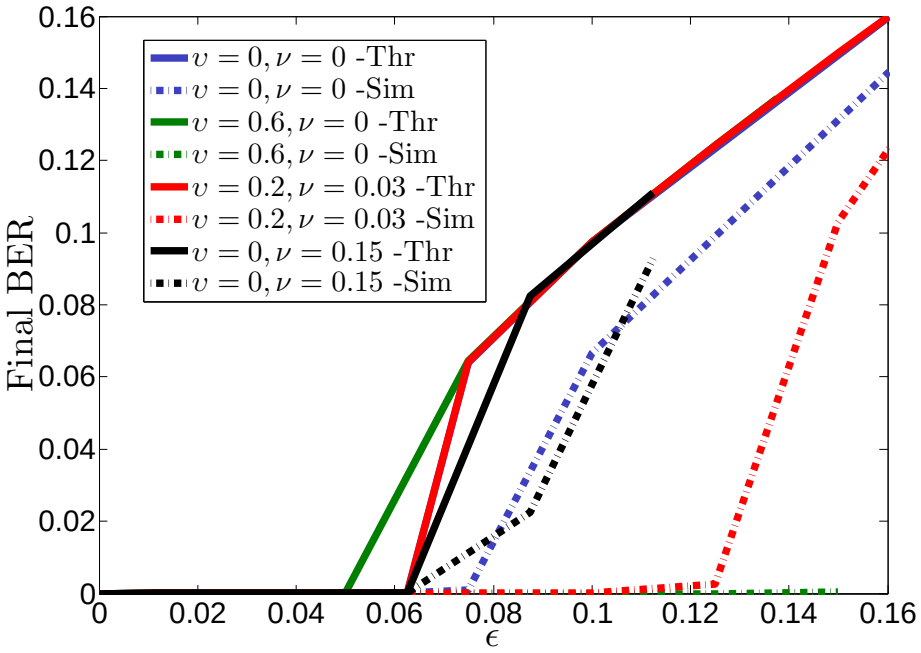


Figure 4: The final pattern error probability for the a network with $n = 400$, $L = 50$, and on average 40 and 20 pattern and constraint nodes per cluster cf. [6]. The blue curves correspond to the noiseless neural network.

upcoming attempts, we have set ν, the noise parameter at the constraint nodes, equal to zero as it seems to be the part of the graph where the bound is very loose.

2.2.1 Attempt 1: not the worst case and not the average case

When bounding the probability of correcting a single error in our analysis in [2], we considered the pessimistic scenario and set P_c, the average probability of correcting one error, to $\min_\ell P_c^{(\ell)}$. In other words, we chose the lowest probability among all the clusters. While this will certainly result in an upper bound on final BER, the bound will be rather loose. One solution for this case will be to set $P_c = \mathbb{E}(P_c^{(\ell)}$, i.e. considering the true average. However, while this makes the bound a bit tighter in some cases, it will not result in an "upper" bound in some other cases as the theoretical bound will become less than the simulation results. The reason might be the fact that the average-case analysis is usually accompanied by a concentration theorem, which is usually valid for trees and graphs without cycles. However, our neural computation graph is definitely not a tree and it is not surprising that we deviate from the average case in some situations.

Therefore, we tested another idea in which instead of picking the cluster with lowest probability of correcting one error, we first sort the probabilities in an ascending order. Then, at a pattern node with degree d, the probability of not being corrected in an iteration is equal to the probability that all of its neighbors (clusters) fail to correct that error. The probability of this event is given

by

$$\prod_{i=1}^{d}(1-\pi^{(i)}),$$

where $\pi^{(i)}$ is the probability that a cluster i can not correct the errors and is given by

$$\pi^{(i)} = 1 - P_c^{(i)}\rho(1-z), \tag{1}$$

with z being the probability of error at current iteration. Note that previously we had $P_c^{\min}$ but this time we have sorted the $P_c^{(i)}$'s and the above result should be tighter.

Result: So-So The bound improved but not to the extent that we hoped!

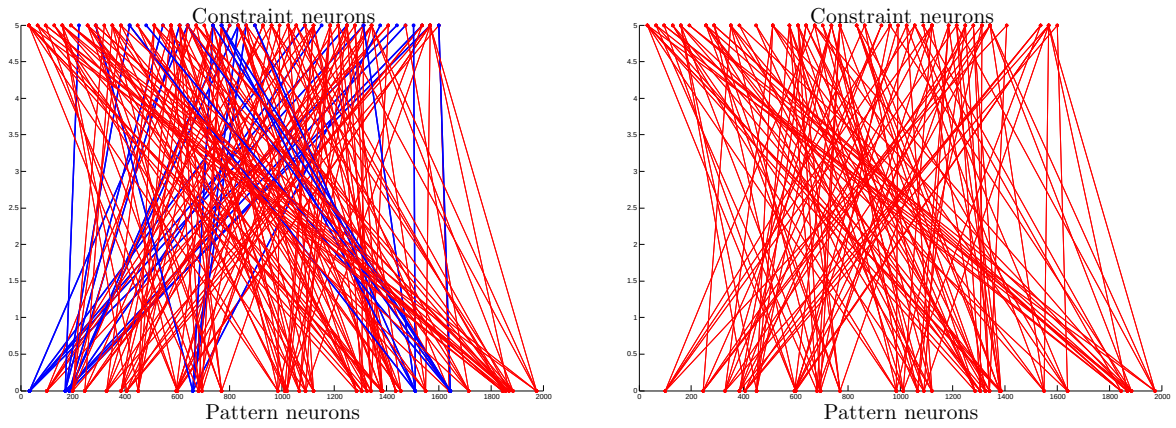
2.2.2 Attempt 2: stopping sets

In our approach, a *Stopping Set* (SS) is an external noise pattern for which the neural error correction algorithm get stuck in a *reliable* neural circuit, i.e. one without internal noise. Assuming that each neural cluster in a reliable network can only correct one external error (an assumtpion which is close to reality in many cases), then the stopping sets correspond to the cases in which each cluster contains either zero or two and more external errors. In other words, a stopping set does not contain any cluster with only a single error in it as, otherwise, the algorithm could have progressed further by correcting that single error.

Figure 5a illustrates a noise pattern which result in a stopping set. In the figure, only the nodes corrupted with external noise are shown for clarity. Pattern neurons that are connected to at least one cluster with a single error are colored red and other pattern neurons are colored red. Figure 5b illustrates the same network but after a sufficient number of decoding iterations which results in the algorithm to get stuck. Obviously, we have a stopping set in which no cluster has a single error and the algorithm could not proceed further, i.e. the external error can not get corrected.

As obvious from Figure 5, the stopping set is the result of clusters not being able to correct more than one external errors. And this is the place where internal noise might come to rescue! Interestingly, an "unreliable" neural circuit in which $\upsilon = .6$ could easily get out of the stopping set shown in Figure 5b and correct all of the external errors. The reason might be because of the fact that we try several times to correct errors in a cluster (and overall in the network) while making sure that the algorithm does not introduce new errors itself. Thus, the noise might act in our favor in one of these attempts and the algorithm might be able to avoid stopping set as depicted in Figure 5.

As a result, we reckon that we should focus on the probability of clusters correcting two external errors (or even more) and see if including this probability will tighten our bounds. To this end, we will first assume we have this probability and extend the analysis given by Luby et al. [4] to derive the new threshold for the fraction of external errors that can be corrected. Note that this is a bit different from finding the BER as this approach gives us only the threshold below which errors can be corrected with probability close to 1. To calculate a new upper bound on BER, we will propose a different approach based on the Density Evolution (DE) technique in Section 2.4.



(a) $t = 0$ (b) $t \to \infty$

Figure 5: An external noise pattern that contains a stopping set in a reliable neural circuit. Left figure shows tha original pattern and the right figure illustrates the result of the decoding algorithm after sufficient number of iterations where the algorithm gets stuck. Blue pattern nodes are those that are connected to at least one cluster with a single external error. Obviously, the stopping set on the right does not have any blue nodes.

2.3 Extending the analysis of the Peeling algorithm [4]

In [4], the authors propose a new decoding method, called the Peeling algorithm, and a novel way of analyzing the performance of the decoder. Our neural error correcting algorithm is very similar to the one proposed in [4] so we will adopt the method proposed in [4] to analyze our algorithm. The main trick is to define a decoding graph, which, in our case, contains only the pattern neurons corrupted with external noise and their corresponding neighbors. Then, in this graph, we should track cluster nodes with degree one and remove them as they can eliminate the only external error present in their domain. We repeat this procedure until either no error remains (i.e. all edges in the decoding graph are removed) or the algorithm get stuck in a stopping set.

Here, we extend the method proposed in [4] to the cases in which cluster neurons with degree two are removed with probability P_{c_2} in each iteration. This correspond to the event that a cluster with two external errors can correct the errors with probability P_{c_2}. For a "reliable" neural circuit, P_{c_2} is usually close to zero. However, for a network with internal noise we might have $P_{c_2} > 0$ which could be the reason behind the improved performance of the algorithm in presence of internal noise.

Following the notations and the approach proposed in [4], we define $\mathcal{R}^{(i)}(t)$ to be the number of edges connected to cluster nodes with degree i on the decoding graph, i.e. the clusters that have i corrupted pattern neurons among their members. Furthermore, letting E be the total number of edges in the original graph, we let $r^{(i)}(t) = \mathcal{R}^{(i)}(t)/E$ be the fraction of such edges with respect to the total number of edges in the original graph. Additionally, let e_t denote the *fraction* of edges at iteration t that are remained in the decoding graph and $a_t = \sum i\ell_t^{(i)}/e_t$ be the expected number of edges that will be deleted if a pattern neuron is removed.

At each iteration, we delete an edge that is connected to a cluster with degree 1 on the decoding graph with probability P_{c_1} and delete an edge that is connected to a cluster with degree 2 with probability P_{c_2}. Removing this edge will result in deleting the corresponding pattern neuron and all other edges connected to it. The probability that one of these edges were connected to a cluster

with degree i is $r_t^{(i+1)}/e_t$. This will result in $i-1$ edges with right-degree $i-1$ (instead of i edges with right degree i). Given that on average we remove $P_{c_1}+2P_{c_2}$ pattern neurons at each iteration (if we have clusters with degree one and two), we will get the following expression on the evolution of cluster nodes with various degrees:

$$\mathbb{E}\left(\mathcal{R}^{(i)}(t+1)-\mathcal{R}^{(i)}(t)\right)=\left(\frac{r^{(i+1)}(t)}{e_t}-\frac{r^{(i)}(t)}{e_t}\right)i(a_t-1)(P_{c_1}+2P_{c_2}), i>2, \tag{2}$$

$$\mathbb{E}\left(\mathcal{R}^{(2)}(t+1)-\mathcal{R}^{(2)}(t)\right)=\left(\frac{r^{(3)}(t)}{e_t}-\frac{r^{(2)}(t)}{e_t}\right)2(a_t-1)(P_{c_1}+2P_{c_2})-2P_{c_2}, \tag{3}$$

$$\mathbb{E}\left(\mathcal{R}^{(1)}(t+1)-\mathcal{R}^{(1)}(t)\right)=\left(\frac{r^{(2)}(t)}{e_t}-\frac{r^{(1)}(t)}{e_t}\right)(a_t-1)(P_{c_1}+2P_{c_2})-P_{c_1}. \tag{4}$$

Now we are interested in tracking $r^{(1)}(t)$ (and $r^{(2)}(t)$) to make sure that we have $r^{(1)}(t)>0$ for all t before the decoding stops. It is easy to show that the above recursions lead to the following differential equation (see also Appendxi B of [4]):

$$r_i'(x)=i\left(-r_{i+1}(x)+r_i(x)\right)\frac{\lambda'(x)}{\lambda(x)}(P_{c_1}+2P_{c_2}) \tag{5}$$

whose solutions is

$$r_i(x)=\left(\lambda(x)^i\right)^{P_{c_1}+2P_{c_2}}\left(c_i-i(P_{c_1}+2P_{c_2})\int_0^x r_i(y)\lambda(y)^{-i(P_{c_1}+2P_{c_2})}\frac{\lambda'(y)}{\lambda(y)}dy\right) \tag{6}$$

Now if we consider the highest non-zero i, denoted by μ, we have $r_{\mu+1}=0$. Thus,

$$r_\mu(x)=c_\mu\lambda(x)^{\mu(P_{c_1}+2P_{c_2})}$$

and recursively, we get [4]:

$$r_i(x)=\sum_{j\geq i}(-1)^{i+j}\binom{j-1}{i-1}c_j\lambda(x)^{j(P_{c_1}+2P_{c_2})}.$$

Since $\lambda(1)=1$, we get

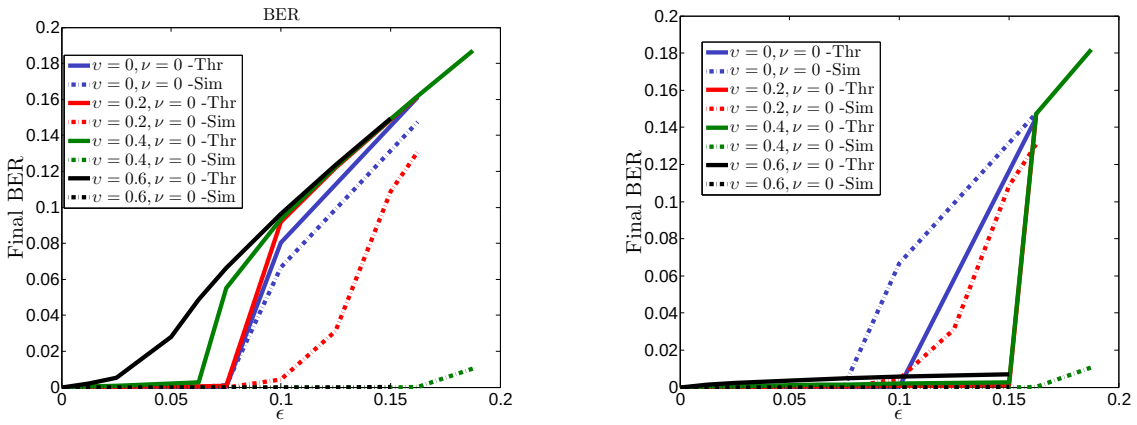
$$c_i=\sum_{j\geq i}\binom{j-1}{i-1}r_j(1).$$

Combining the above relationship with equation (5) of [4], we get

$$c_i=\sum_{m\geq i}\binom{m-1}{i-1}\rho_m\epsilon^i, \tag{7}$$

where ϵ is the probability of having an external error. Replacing equation (7) in (6) reuslts in

$$r_i(x)=\sum_{j\geq i}\sum_{m\geq j}(-1)^{i+j}\binom{j-1}{i-1}\binom{m-1}{j-1}\rho_m\left(\epsilon\lambda(x)^{P_{c_1}+2P_{c_2}}\right)^j \tag{8}$$



(a) $P_{c_1} \simeq 1$ and $P_{c_2} = 0$

(b) $P_{c_1} \simeq 1$ and $P_{c_2} = P_{c_1}$

Figure 6: Effect of including P_{c_2} in the theoretical bounds: the left figure illustrates the theoretical bounds as well as the simulations results for $P_{c_1} \simeq 1$ and $P_{c_2} = 0$. The right figure shows the same results for $P_{c_1} \simeq 1$ and $P_{c_2} = P_{c_1}$. The bounds in the right-hand figure are considerably lower.

Now all remains to do is to calculate $r_1(x)$. Using the relationship $r_1(x) = e(x) - \sum_{i>1} r_i(x)$ we obtain [4]:

$$r_1(x) = \epsilon\lambda(x)^{(P_{c_1}+2P_{c_2})}\left[x - 1 + \rho\left(1 - \epsilon\left(\lambda(x)\right)^{P_{c_1}+2P_{c_2}}\right)\right], \tag{9}$$

where $\lambda(x)$ and $\rho(x)$ are the degree distributions of the *original* graph, ϵ is the probability of external errors and x is found by solving the equation $dx/dt = -x/e_t$.

As a result, the decoding operation will be successful if we have $r_1(x) > 0$ for all $x \in (0, 1]$. This requires us to have

$$x > 1 - \rho\left(1 - \epsilon\left(\lambda(x)\right)^{P_{c_1}+2P_{c_2}}\right), \forall x \in (0, 1] \tag{10}$$

The appealing property of the above equation is that it reduces to the original form mentioned in [4] for $P_{c_1} = 1$ and $P_{c_2} = 0$.

We could generalize the above recursion by considering the probability of correcting 3 errors, 4 errors and so on. Letting $\bar{e} = P_{c_1} + 2P_{c_2} + 3P_{c_3} + \ldots$ be the average number of edges deleted in each round, we will get

$$x > 1 - \rho\left(1 - \epsilon\left(\lambda(x)\right)^{\bar{e}}\right), \forall x \in (0, 1] \tag{11}$$

To show that this approach will result in tighter bound, in Figure 6 we have compared our old bound with the new one for $P_{c_1} = P_{c_2} = 1$. The simulations results are shown as well. As evident from the figure, the bound is considerably lower. However, in many cases they are not upper bounds anymore and become less than simulation values. The reason is that we didn't calculate P_{c_2} exactly. So all remains now is to computed P_{c_2}, and in general P_{c_i} for $i \geq 2$.

2.4 New upper bound based on DE

We could alternatively use the DE approach to derive an upper bound on the final BER. To this end, we again assume that we have P_{c_1}, P_{c_2} and P_{c_3} but the approach can easily be extended if we have P_{c_i} for $i > 3$. Let $\Pi(t)$ be the average probability that a cluster node send a failure message, i.e. that it can not correct external errors laying in its domain. Then, the probability

that a pattern neuron with degree d_i can not be corrected will be the probability that none of its neighboring clusters can correct the error, i.e.

$$P_i = (\Pi(t))_i^d$$

Averaging over d_i we find that the average probability of error in iteration t will be

$$z(t+1) = \lambda(\Pi(t)) \tag{12}$$

Now consider a cluster ℓ that contains d_ℓ pattern neurons. This cluster will *not* send a failure message to a noisy pattern neuron in its domain with probability

1. P_{c_1}, if it is not connected to any other noisy neuron.
2. P_{c_2}, if it is connected to exaclty one other contraint neuron.
3. P_{c_3}, if it is connected to exaclty two other contraint neurons.
4. 0, if it is connected to more than two other contraint neuron.

Thus, we obtain

$$\Pi^{(\ell)}(t) = 1 - P_{c_1}(1-z)^{d_\ell - 1} - P_{c_2}\binom{d_\ell - 1}{1} z(1-z)^{d_\ell - 2} - P_{c_3}\binom{d_\ell - 1}{2} z^2(1-z)^{d_\ell - 3}.$$

Averaging over d_ℓ we obtain

$$\Pi(t) = \mathbb{E}_{d_\ell}\left(\Pi^{(\ell)}(t)\right) = 1 - P_{c_1}\rho(1-z(t)) - P_{c_2} z\rho'(1-z(t)) - 0.5P_{c_2}z^2\rho"(1-z(t)), \tag{13}$$

where $\rho'(x)$ and $\rho"(x)$ are derivatives of the function $\rho(x)$ with respect to x.

Equations (12) and (13) will give us the value of $z(t+1)$ as a function of $z(t)$. We can then calculate the final BER as $z(\infty)$.

2.5 Calculating P_{c_2}

2.5.1 Attempt 1

To calculate the probability of correcting two external errors, we could proceed as we did in [1]. To start, suppose in a given cluster ℓ, the pattern neurons b_1 and b_2 are corrupted with a $+1$ noise. Suppose b_1 and b_2 have degrees d_1 and d_2, respectively. Furthermore, they share $\bar{d}$ neighbors among the constrain neurons. Thus, the total number of violated constraints are $d_1 + d_2 - \bar{d}$. Let $P_2^{(\ell)}$ the probability that the cluster ℓ could not correct two errors. This can happen if

1. The two corrupted pattern neurons are not corrected.
2. At least one of other pattern neurons update their state by mistake.

Let $\bar{p}^{(\ell)}$ and $\bar{q}^{(\ell)}$ denote the probability of these two events, respectively. Then

$$P_2^{(\ell)} = 1 - \left(1 - \bar{p}^{(\ell)}\right)^2 \left(1 - \bar{q}^{(\ell)}\right)^{n_\ell - 2} \tag{14}$$

To proceed further, recall that we have assumed the noise parameters at constraint neurons are close to zero, i.e. $\nu = 0$.[2]

Now to calculate $\bar{q}^{(\ell)}$ first consider a non-corrupted pattern neuron b_i, with $i > 2$, whose degree is d_i and has $\bar{d}_i$ common neighbors among the $d_1 + d_2 - \bar{d}$ violated constraints. Thus, this node receives $\bar{d}_i$ non-zero feedback from its neighbors. Each of the non-zero messages will have the same sign as W_{ij}, the edge they are conveyed on, with probability $1/2$. Assuming that e of these messages have the same sign as the weight of the edge they are conveyed on, the net input sum node b_i receives is

$$h_i = e - (\bar{d}_i - e) = 2e - \bar{d}_i$$

Now node b_i updates its state if

$$|\frac{h_i}{d_i} + u_i| \geq \varphi, \tag{15}$$

where u_i is the uniformly distributed internal noise and φ is the update threshold. Thus, we have

$$q(h_i) = \frac{\min(\varphi - h_i/d_i, \upsilon) - \max(-\varphi - h_i/d_i, -\upsilon)}{2\upsilon} \tag{16}$$

By taking average over e and $\bar{d}_i$ we will get

$$q^{(\ell)} = \sum_{\bar{d}_i=0}^{d_i} \binom{d_i}{\bar{d}_i} \left(\frac{d_1 + d_2 - \bar{d}}{m_\ell}\right)^{\bar{d}_i} \left(1 - \frac{d_1 + d_2 - \bar{d}}{m_\ell}\right)^{d_i - \bar{d}_i} \sum_{e=0}^{\bar{d}_i} \binom{\bar{d}_i}{e} \left(\frac{1}{2}\right)^{\bar{d}_i} q(2e - \bar{d}_i) \tag{17}$$

Next, we take average over $\bar{d}$. To this end, the probability of having $\bar{d}$ common neighbors with b_1 for b_2 is $\binom{d_2}{\bar{d}} \left(\frac{d_1}{m_\ell}\right)^{\bar{d}} \left(1 - \frac{d_1}{m_\ell}\right)^{d_2 - \bar{d}}$. Thus, by averaging over d_1, d_2 and d_i we obtain

$$\bar{q}^{(\ell)} = \mathbb{E}_{d_1,d_2,d_i} \left[\sum_{\bar{d}=0}^{d_2} \binom{d_2}{\bar{d}} \left(\frac{d_1}{m_\ell}\right)^{\bar{d}} \left(1 - \frac{d_1}{m_\ell}\right)^{d_2 - \bar{d}} q^{(\ell)} \right] \tag{18}$$

We could use a similar approach to calculate $\bar{p}^{(\ell)}$. To this end, consider a corrupted node b_i, with $i = 1, 2$. It receives $d_i - \bar{d}$ messages that have the same sign as the weights they are conveyed on. For the remaining $\bar{d}$ messages, suppose e of them have the same sign as the weights connected to b_i. The probability of this event is $(1/2)^e (1/2)^{\bar{d}-e} = (1/2)^{\bar{d}}$. Thus, the input to node b_1 is $h_i = d_i - \bar{d} + e - (\bar{d} - e) = d_i + 2e - 2\bar{d}$. Now node b_i makes a mistake if

$$\frac{h_i}{d_i} + u_i < \varphi. \tag{19}$$

Thus, the probability of making mistake at b_i with the given conditions is

$$p(h_i) = \min(\frac{\max(\varphi - h_i/d_i, -\upsilon) + \upsilon}{2\upsilon}, 1) \tag{20}$$

By taking average over e and $\bar{d}$ we will get

$$\bar{p}^{(\ell)} = \mathbb{E}_{d_1,d_2} \left[\sum_{\bar{d}=0}^{d_1} \binom{d_1}{\bar{d}} \left(\frac{d_2}{m_\ell}\right)^{\bar{d}} \left(1 - \frac{d_2}{m_\ell}\right)^{d_1 - \bar{d}} \sum_{e=0}^{\bar{d}} \binom{\bar{d}}{e} \left(\frac{1}{2}\right)^{\bar{d}} p(d_1 + 2e - 2\bar{d}) \right] \tag{21}$$

[2]This assumption is made because this was the situation where our bound was less tight. Nevertheless, our upcoming analysis can easily be extended to cases where $\nu > 0$.

Result: So-so: We computed these values numerically. The plus side is that we observe the same trend as in practice, namely, $P_2^{(\ell)}$ is lower for networks that have a fair amount of internal noise, but only just. For instance, $P_2^{(\ell)} \simeq 1$ for $\upsilon = 0$ and $P_2^{(\ell)} \simeq 0.99$ for $\upsilon = 0.5$.

2.5.2 Attempt 2

We could etend the above approach further by splitting $p^{(\ell)}$ into two parts: the probability that a corrupted pattern neuron is not updated at all in an iteration, $p_1^{(\ell)}$, and the probability that the node is updated in the wrong direction, $p_2^{(\ell)}$. Obviously, $p^{(\ell)} = p_1^{(\ell)} + p_2^{(\ell)}$. Keeping in mind that we perform the error correction algorithm several times and the each cluster could correct one error with probability close to 1, we could think of different scenarios in which correcting two errors is possible in one or more iterations:

1. 2 errors get corrected in the first iteration and none of the non-corrupted neurons update its state
2. 1 of the errors get corrected in the first iteration and none of the non-corrupted neurons update its state. This way we will correct the other one in the next iteration with probability 1.
3. 2 errors get corrected in the first iteration and only one of the non-corrupted neurons update its state. We will correct this new error in the next round with probability close to 1.

Let $\tilde{p}_1$ denote the probability of the above events. However, we could also think of scenarios in which starting with two external errors, we end up with two (possibly different) errors in the pattern neurons after performing one iteration. Let $\tilde{p}_2$ denote the probability of this event. Therefore, we could repeat the same steps above again to hopefully eliminate the errors. Some of the scenarios that this situation occurs are

1. 2 erroneous nodes are not updated in the first iteration and none of the non-corrupted neurons update its state.
2. 1 of the errors get corrected in the first iteration and exactly one of the non-corrupted neuorns updates its state.
3. 2 errors get corrected in the first iteration and exactly two of the non-corrupted neurons update its state.

Havig the above definitions and notations we obtain

$$\begin{aligned} P_{c_2} &\geq \tilde{p}_1 + \tilde{p}_2\tilde{p}_1 + (\tilde{p}_2)^2\tilde{p}_1 + \ldots + (\tilde{p}_2)^(T-1)\tilde{p}_1 \\ &= \tilde{p}_1 \frac{1-(\tilde{p}_2)^T}{1-\tilde{p}_2}, \end{aligned} \tag{22}$$

where T is the number of attempts.

Result: Failure: Compared to the bound in the previous section, we did not gain much because $\tilde{p}_2$ is very small and even if $T \to \infty$, the gain we get is $1/(1-\tilde{p}_2)$. For $\tilde{p}_2$ very close to zero, this value is close 1 which means negligible gain.

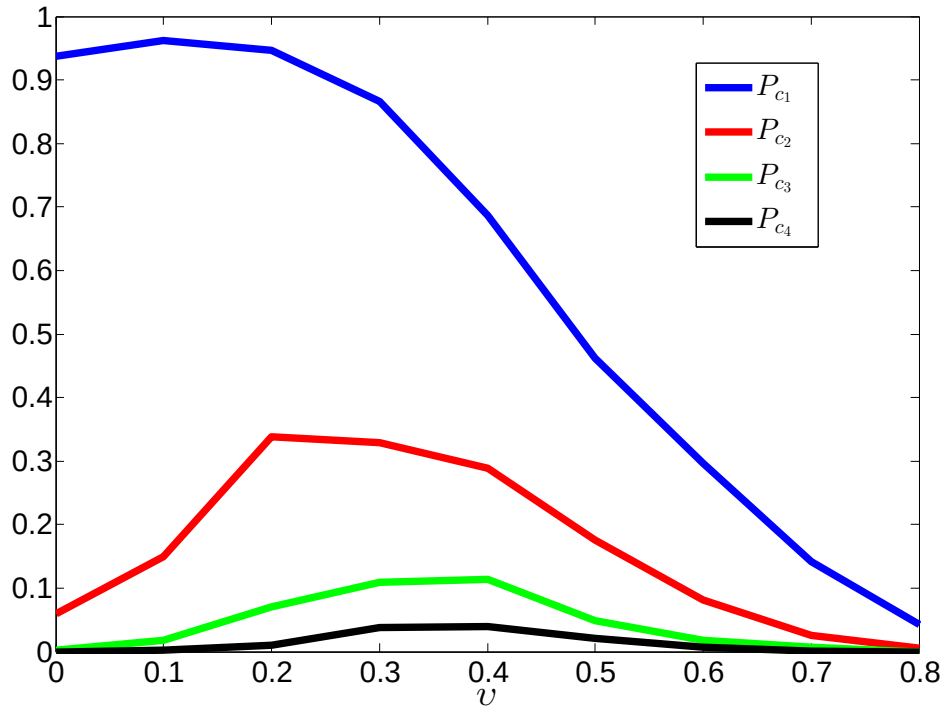


Figure 7: Probability that each cluster correct i errors for $i = 1, \ldots, 4$ as a function of υ, the noise parameter at the pattern neurons.

2.5.3 Attempt 3: using empirical methods

Given that theoretical bounds on P_{c_2} are too loose, we turned our attention to finding P_{c_2} empirically. More specifically, we let the intra-cluster recall algorithm run for a number of iterations when there are two external errors present in the cluster. We then compute the fraction of times each cluster was able to correct the errors and consider it as an approximation to $P_{c_2}^{(\ell)}$. We could then set $P_{c_2} = \min_\ell P_{c_2}^{(\ell)}$ to have an upper bound on the performance of the algorithm.

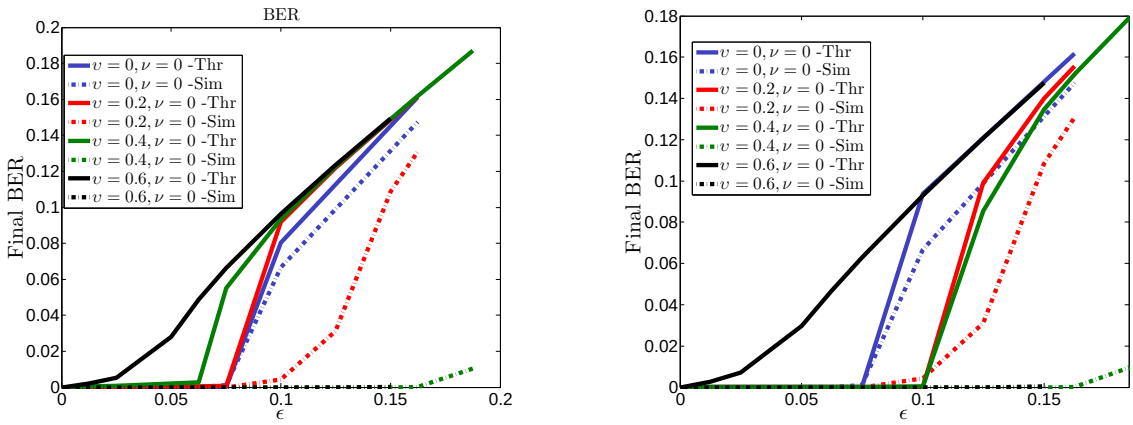
Figure 7 illustrates P_{c_i} for $i = 1, \ldots, 4$, i.e. the probability that each cluster correct i external errors, as a function of υ. As evident from the figure, having a fair amount of internal noise definitely helps the clusters to correct more external errors. However, as expected, the performance deteriorates again once the amount of internal noise exceeds a threshold.

2.6 Wrapping up: plugging P_{c_2} into new theoretical bounds

We now use the values obtained for P_{c_2} (and in general for P_{c_i}) into the bounds we derived in Section 2.4. Figure 8a illustrates the simulations results together with our old bound. Figure 8b shows the same result but for the new bound given by equations (12) and (13). We have used the empirical estimations for $P_{c_i},\ i = 1, \ldots, 4$.

From Figure 8 we see that:

1. The new bounds are tighter, which is nice.



(a) $P_{c_1} \simeq 1$ and $P_{c_i} = 0$ for $i = 2, 3, 4$.

(b) $P_{c_i},\ i = 1, \ldots, 4$ are derived empirically.

Figure 8: Old (left) and new (right) theoretical bounds as well as simulation results. For the new bound, $P_{c_i},\ i = 1, \ldots, 4$ are derived empirically and plugged into the density evolution equations (12) and (13). Obviously, the new bounds are much better but there is still room to progress.

2. However, a strange behavior is apparent in Figure 8b where for $\upsilon = 0.6$, we get a worse bound than $\upsilon = 0, 0.2, 0.4$ while the simulation results for $\upsilon = 0.6$ is the best among all scenarios. This phenomenon is caused by the fact that P_{c_i} is smaller for $\upsilon = 0.6$ than, say, $\upsilon = 0.4$. However, why this occurs is not known to us.

3 Multi-layer Nonlinear MCA

We have also spent some time to implement a multi-layer neural network to perform NLMCA (Non-Linear Minor Component Analysis). The idea is very similar to the idea of NLPCA mentioned in [7]. But here, instead of extracting the components with highest (non-linear) variance, we would like to find those with close-to-zero (non-linear) variance.

More specifically, our goal is to design a two-layer neural network, with n input pattern neurons, $x_1, \ldots, x_n$, connected via a $k \times n$ weight matrix W to $k > n$ intermediate neurons $z_1, \ldots, z_k$. We then have another layer in which m non-linear neurons $y_1, \ldots, y_m$ are connected via a weighted $m \times k$ matrix $\tilde{W}$ to the intermediate neurons. The neural activation function for the intermediate neuron z_i is

$$z_i = f((Wx)_i + \theta_i), \tag{23}$$

where f is usually a sigmoid function and θ_i is the tunable update threshold.[3] Likewise, the update rule for neurons in the last stage is

$$y_j = g((\tilde{W}z)_j + \tilde{\theta}_j), \tag{24}$$

in which g is a nonlinear function and $\tilde{\theta}_j$ is the tunable update threshold.

Our goal is to design a network such that for any input pattern x, we have $\|y\|_2 \simeq 0$. To this

[3]Note that this update threshold can also be learned by incorporating it as a column of the weight matrix W which is connected to a "dummy" pattern neuron whose state is always equal to 1. This is how we determine update thresholds in practice.

end, we define an optimization problem as follows

$$\min_{W,\tilde{W}} E = \sum_{x^\mu \in \mathcal{X}} \|y^\mu\|^2 + \text{sparsity penalty} \tag{25a}$$

subject to

$$\|W_i\|_2 = 1, \forall i = 1, \ldots, k \tag{25b}$$

and

$$\|\tilde{W}_i\|_2 = 1 \forall i = 1, \ldots, m \tag{25c}$$

In the above equations, W_i and $\tilde{W}_i$ are the i^{th} rows of the matrices W and $\tilde{W}$, respectively.

As usual, we take the derivative of the objective function with respect to W_{ij} and $\tilde{W}_{ij}$ to find the update rule for the weights. We also take care of the constraints by including them in the update rules using Lagrange multipliers. Thus, noting that all vectors are column vectors, in a straightforward manner we obtain the following matrix update rule

$$\tilde{W}(t+1) = \tilde{W}(t) - \tilde{\alpha}_t G'(u(t)) y(t) z(t)^\top, \tag{26}$$

where $u(t) = Wz(t)$ and $G'(.)$ is a diagonal matrix whose entries are the derivative of the function $g(.)$. Likewise, we find

$$W(t+1) = W(t) - \alpha_t F'(x(t)) \tilde{W}^\top G'(u(t)) y(t) x(t)^\top, \tag{27}$$

where $F'(.)$ is a diagonal matrix whose entries are the derivative of the function $f(.)$.

Note that the Lagrangian term and the sparsity penalty for ensuring the constraints (25b) and (25c) are not shown for brevity.

Results: So-so: The overall algorithm was implemented and worked over a cluster of pattern neurons for sparse filtered CIFAR-10 images. But then there were two problems:

1. Despite being slow, the algorithm was still capable of finding enough constraints. However, **the recall algorithm was not well-defined**. More specifically, we have not yet found out how should the recall algorithm work.

2. In some other cases the algorithm found a $\tilde{W}$ which was orthogonal to W and obviously this satisfies our goal to have $\|y\|_2 \simeq 0$. Thus, to avoid this situation we must add a suitable constraint.

The algorithm was also applied to the pixel-level images of the CIFAR-10 dataset but it failed completely.

4 Simulations over CIFAR-10 dataset

We also continued our simulations over the CIFAR-10 dataset. This time, we first performed the clustering algorithm for the already-learnt matrix we had last time hoping that it solves the problems we faced last time. But this approach failed.

5 New scheduling-based recall algorithm

Another idea that we tested was to redesign the scheduling-based recall algorithm (see the previous report). This time, we perform the scheduling not only in the inter-cluster algorithm, but also in the intra-cluster method. Furthermore, we consider each cluster to be only a single constraint. Now since we are interested in each cluster correct one error, we start from the first pattern neuron connected to the constraint neuron and update it in the direction given by the constraint. We then verify if the error is resolved and the constraint is satisfied. If not, we revert everything back to its condition and continue with other pattern neurons. Although not brilliant, this approach seems to be promising.

Results: Mostly success: The algorithm was able to correct some errors (rather small amount). But this inability was a result of bad learning algorithm in which the maximum allowed projection of the patterns was larger than what it should have been. We have to perform the learning algorithm more strictly and see what happens with the recall algorithm afterwards.

6 Future work

For the upcoming weeks, we have to address several issues. For the faulty neural networks, we must find out why P_{c_2} is lower for $\upsilon = 0.6$ while we see the seemingly opposite behavior in practice. Furthermore, our generalization of the Peeling Algorithm still lacks some details, specially since we have to track degree-two cluster nodes now.

For the NLMCA approach, we must first add a constraint to the learning algorithm to ensure that W will not be orthogonal to $\tilde{W}$ (i.e. the trivial solution). We should then define a proper recall algorithm.

Finally, the MATLAB codes need some serious organization as they are growing out of control!

References

[1] A. Karbasi, A. H. Salavati, A. Shokrollahi, L. R. Varshney, *Neural networks built from unreliable components*, Submitted to ISIT 2013.

[2] A. Karbasi, A. H. Salavati, A. Shokrollahi, and L. R. Varshney, Neural networks built from unreliable components, arXiv, Jan. 2013.

[3] K. Wiesenfeld, F. Moss, *Stochastic resonance and the benefits of noise: from ice ages to crayfish and SQUIDs*, Nature Vol. 373, 195, pp. 33–36

[4] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, *Efficient erasure correcting codes,* IEEE Trans. Inf. Theory, vol. 47, no. 2, pp. 569584, Feb. 2001.

[5] N. Mackworth, *Effects of heat on wireless operators,* British Journal of Industrial Medicine, vol. 3, no. 3, pp. 143158, 1946.

[6] A. Karbasi, A. H. Salavati, and A. Shokrollahi, *Iterative learning and denoising in convolutional neural associative memories,* in Proc. 30th Int. Conf. Mach. Learn. (ICML 2013), Jun. 2013, to appear.

[7] M. Scholz, M. Fraunholz, J. Selbig, *Nonlinear principal component analysis: neural network models and applications*, Principal manifolds for data visualization and dimension reduction, Lecture notes in computational science and enginee, Vol. 58, 2008, pp 44–67.