Progress Report
1-15 October 2011

Amir Hesam Salavati
E-mail: raj.kumar@epfl.ch, hesam.salavati@epfl.ch

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

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

October 25, 2011

1 Summary

In the last two weeks, I was mainly busy with implementing an idea to solve the learning part of our neural constraint method in [2], i.e. to find a sparse matrix W such that for all vectors x in the training data set we have $Wx = 0$. The idea is based on the method to identify null basis vectors of the a subspace from the vectors that belong to that subspace [1]. The end result has been a partial success so far in the sense that the MATLAB code finds a relatively sparse basis for the null space from the patterns in the data set. The performance in the error correcting phase is also not that bad, however, since the end result is not an expander graph, it is still not comparable to what we expect. Fruthermore, since the code is slow to run, I have not yet managed to run extensive simultions over large graphs to see if the performance improves for larger network sizes. In what follows, you will find the details of the algorithm as well as some ideas for future works.

2 The main idea

The main idea of identifying the null basis is adopted from [1] with some modifications in order to make it realizible by our double layer network in a distributed fashion. Formally speaking, part of the problem that is addressed in [1] and concerns us is as follows: we have a subspace L that has k basis vectors in $\mathcal{R}^n$. Suppose instead of representing this class by its own basis vectors, we are interested in using the dual basis vectors to do the job, i.e $\bar{L} = span\{v_1, \ldots, v_m\}$, where $m = n - k$ and the orthogonal vectors $v_1, \ldots, v_m$ are in in the dual space of L, denoted by $\bar{L}$.

Now the goal is to find the set of dual basis vectors $v_1, \ldots, v_m$ from the data x. To this end, the Anti Hebbian version of the Stochastic Gradient Ascent (SGA) algorithm, proposed by Oja [4], is used. In general, we have a neural network with m layers of n neurons each. In each layer, the neurons are connected in a line and the weights between consecutive neurons, w_{ij} with $1 \leq i \leq n$ and $1 \leq j \leq m$,will represent the components of the dual basis vectors.

The learning phase for the weight vectors are given by equation (1a).

$$\Delta w_{ij} = -\alpha y_j (x_i^{(j-1)} - \frac{y_j w_{ij}}{\|w_{:j}\|^2}) \tag{1a}$$

$$y_j = \sum_{i=1}^{m} x_i^{(j-1)} w_j \tag{1b}$$

$$x_i^{(j)} = x_i^{(j-1)} - 2 y_j w_{ij} \tag{1c}$$

where x is the data vector, $x^{(1)} = x$ and α is a small constant.

In words, y_j is the projection of $x^{(j-1)}$ on the j^{th} basis vector. If for a given data vector x, y_j is equal to zero, namely, the data is orthogonal to the weight vector w_j, then according to equation (1a) the weight vector j will not be updated. However, if the data vector x has some projection over w_j then the weight vector is updated towards the direction to reduce this projection. Furthermore, since we are interested in finding m orthogonal vectors, for each such vectors, say w_j, we substract the effect of the previous layer and feed the new layer with $x^{(j)}$, according to equation (1c).

It can be shown that as a result of equation (1a), the first unit vector weight w_1 will converge to an orthogonal direction of the input data x. w_2 will converge to the orthogonal direction of the $x - y_1 w_1$, which is the *novel* part of x with respect to the first orthogonal component w_1 and so on.

In fact, it can be provedthat this method yields vectors that converge to the $m = n - k$ *smallest* eigenvectors of the matrix $E[xx^T | x \in L]$.

2.1 Some modifications

So far, equation (1a) will give us the necessary weight vectors. However, the way equation (1a) is implemented in [1] needs $n.m$ neurons. Nevertheless, with a small trick we can do the same job using our own neural network which is composed of $n+m$ neurons [2]. In [1], in order to imlement equation (1a) the authors have used m *line-layers*[1] of n neurons and the weights in each layer represent the null basis vectors. The training data is iven to first layer and once the weights for this layer are updated, the novelty part, i.e. the data minus its projection on the first layer, is fed to the next layer and the same procedure happens for the novel part of the data. This process continues for all layers and later for all patterns in the training data set.

However, if instead of using m line-layers, we employ our double layer neural network of $n+m$ neurons, we will be able to carry out a similar procedure to [1] as follows: After initialization of weights to some random value, we feed the data to the network and calculate the projection of the data on the weights connected to the first constraint node, y_1. In other words, we caclulate how much the first cosntraint is violated. The amount of violation is used to update all the edges connected to the first constraint node. Furthermore, the value of y_1 is sent back to the pattern nodes which they use to update their state according to equation (1c). Then the same process is performed with the updated pattern and the second constraint. After updating the pattern nodes again, we will do the same for all other constraints.

When all constraints are finished, we give the network another pattern from the training data set and repeat the whole process above once more. Finally, in order to make sure a gradual convergence and avoid any cancellations in the learning process, we go over the training data set a few times. This way, we will be able to learn the null space basis using $n+m$ neurons and in a gardual manner.

3 Making the matrix sparse

The above procedure will give us the null space basis for the given data set. However, the resulting matrix may not be sparse at all. In order to have a rather sparse connectivity matrix, we introduce a cost function similar to [3] in order to prioritize sparser solutions. Following the discussion we had before, the following cost function was used:

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\tanh(\sigma w_{ij})^2 \tag{2}$$

Equation (2) approximates $\sum_{i=1}^{m}\sum_{j=1}^{n}|sign(w_{ij})|$, since the power two term in $\tanh(\sigma w_{ij})^2$ acts like the absolute value function and the term $\tanh(\sigma(w_{ij}))$ looks like the sign function for large enough σ's.

We use the derivative of equations (2) with respect to w_{ij} in the update process (1a) in order to encourage the sparse solutions.Therefre, up to now the overall learning algorithm looks like the following equations:

[1] A line layer is a layer of neurons that are connected as a line to each other. So we have n neurons and n edges, the last of which is the output edge.

$$y_j = \sum_{i=1}^{m} x_i w_{ij} \tag{3a}$$

$$w_{ij} = w_{ij} - \alpha y_j \left(x_i - \frac{y_j w_{ij}}{\|w_{:j}\|^2} \right) - \lambda 2 \tanh(\sigma * w_{ij})(1 - (\tanh(\sigma w_{ij}))^2) \tag{3b}$$

$$x_i = x_i - 2\beta y_j w_{ij} \tag{3c}$$

where *lambda* is a small real number and *sigma* has a relatively large value.

4 The all-zero solution

The set of equations (3) has a trivial solution which is sparse and satisfies the set of constraints $Wx = y = 0$. In order to avoid this all-zero solution, I first introduced a cost for the all zero case as well so that the final result do not converge to the all-zero case. Two cost functions were tested with success, given by equations (4) and (5).

$$C_1 = \sum_{j=1}^{n} e^{-\gamma \sum_{i=1}^{m} w_{ij}^2} \tag{4}$$

where γ is a relatively large number.

$$C_2 = \sum_{j=1}^{n} \frac{1}{0.0001 + \sum_{i=1}^{m} \tanh(\sigma w_{ij})^2} \tag{5}$$

Howevere, the above cost functions are not local in the sense that in order to update the weights attached to the constraint node j, we need the value of the weights corresponding to the links which are connected to all the neighbors of this constrint node. Moreover, after modifying the initialization process for the weight matrix, even without the above cost functions the overall result will not converge to the all zero solution. In the new initialization process, instead of starting from a random weight matrix with real valued elements, I start with a random *sparse* weight matrix with 0/1 elements.

5 Faster algorithm: learn and recall at the same time!

So far, the algorithm yields a sparse matrix which satisfies the set of linear equations $Wx = 0$, for all vectors x in the training data set. However, since we have to go *several times* over 2^k such vectors, and for each vector update m vectors of length n, the whole learning process is extremely slow! Off course main part of this slowness is due to the large number of patterns that we want to memorize. Therefore, I have tried a different scheme in which the algorithm do the learning phase for a number of patterns, say 10000 of them. At the end of this learning phase, the algorithm has a rough estimation of the subspace that the training patterns belong to.

Then, the network starts doing the recall phase. During the recall phase, if the amount of constraint violation for each constraint node is less than a threshold, the algorithm assumes this is just a pattern that it had memorized before and do the normal error correction procedure explained

in our paper [2]. However, if the amount of constraint violation is too much, it assumes the input corresponds to a new pattern (possible noisy as well) and do a learning step to memorize this pattern. Now if this pattern is encountered later, it can be corrected.

Obviously, there is a tradeoff between accuracy and speed here, but this scenario makes sense biologically as well. Because we do not learn everything at once but rather do the process in a gradual manner.

6 Results

In order to test the algorithm, I picked a regular random 0/1 generator matrix with row degree and column degree equal to 1 and 3 respectively. The patterns were then composed of the product of all binary vectors of length k with this generator matrix G. In the recall phase, the maximum noise amplitude was assumed to be equal to 1, which is something that should be corrected later. For all cases, a total of 100000 patterns were used in the training set and the same amount of random patterns were used in the recall phase.

Table 1 summarizes the block and bit error rates for the algorithm given by equations (3) with the following parameters: $n = 90$, $k = 30$, $\alpha = 0.003$, $\beta = 0$, $\sigma = 100$,$\lambda = 0$ (see the next section to see why these two parameters are set to zero. I have tried other variations with success as well). The maximum column and row degrees of W were 12 and 18 respectively, which yields a rather sparse matrix.

Initial number of noisy nodes	0	1	2	3
Block error rate	0	0	0.0115	0.0372
Bit error rate	0	0	0.0004	0.0012

Table 1: Error performance with $n = 90$ and $k = 30$e

Table 2 illustrates the same results for the case of $n = 150$ and $k = 50$. The maximum column and row degrees of W were 12 and 18 again. Table 2 shows the results for the case of $n = 300$ and $k = 100$.

Initial number of noisy nodes	0	1	2	3
Block error rate	0	0	0.007	0.02
Bit error rate	0	0	0.00014	0.0004

Table 2: Error performance for $n = 150$ and $k = 50$.

Initial number of noisy nodes	0	1	2	3
Block error rate	0	0	0.0016	0.005
Bit error rate	0	0	0.000016	0.00005

Table 3: Error performance with $n = 300$ and $k = 100$e

Figure 1 compares the block error rates for the three cases mentioned above. From figure 1, it

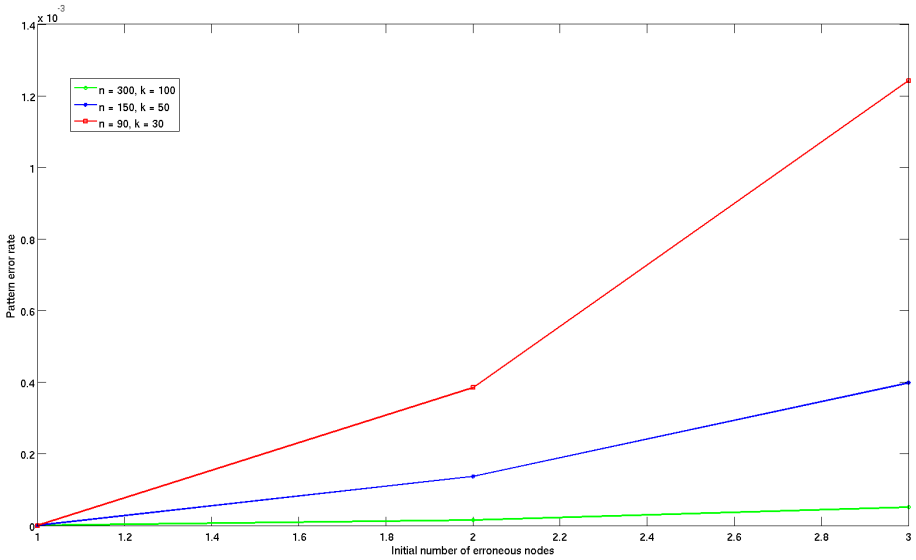

Figure 1: Pattern error rate for the error performance of the recall phase after the connectivity matrix was learned by the algorithm (3) for different values of n and k.

seems that error rate improves as the network size increases.

7 Future works

Just when I was writing this report and in order to bring the results, I witnessed a very interesting phenomena. Suppose we put $\beta = \lambda = 0$ in equation (3). In words, we shoud expect non-sparse solutions due to $\lambda = 0$ and redundant constraints due to $\beta = 0$, since we do not compute the novel part of the data for each constraint. However, what happens is that the solution is still sparse and half ot the constraint are unique and the other half are equal to the negative of the first half. So may be if we start from a sparse initialization (like what we do now), and pick β to a small value, we get sparse and good resluts at the same time. However, if β is chosen to be closer to 1 than to to 0, then we definitely need a λ which is rather large.

There are a couple of ideas similar to the ones introduced in [1] (and used in our algorithm) which I would like to test as well to see how they compare to the one that was tested in this report [5], [6], [7].

Making the code faster and parallel is another top priority since it enables us to simulate over larger network sizes. From figure 1, it seems that the error rate improves with network size growing so we might actually end up with a good error performance for large network sizes.

7.1 How about expansion property?

The algorithm explained in this report only gives us partially good results because it is sparse but not expander. In order to make the graph expander, we might consider introducing costs on non-expander solutions. However, this is rather tricky because we end up having non-local learning rules.

Following a discussion with one of my firends, I am currently working on the idea to make the first two singular values of the matrix W as far from each other as possible because it is known that for an expander graph, there is a good gap between the first two singular values. However, I have failed so far to incorporate this idea into a step-by-step learning alorithm, similar to the way we introduced the sparsity cost.

References

[1] L. Xu, A. Krzyzak, E. Oja, Neural nets for dual subspace pattern recognition method, Int. J. Neur. Syst., Vol. 2, No. 3, 1991, pp. 169-184.

[2] K.R. Kumar, A.H. Salavati and A. Shokrollahi, *Exponential pattern retrieval capacity with non-binary associative memory*, Proc. IEEE Information Theory Workshop, 2011.

[3] B. A. Olshausen, D. J. Field, *Emergence of simple-cell receptive field properties by learning a sparse code for natural images*, Nature Vol. 381, 1996, pp. 607-609.

[4] E. Oja, *Principal components, minor components, and linear neural networks*, Neur. Net., Vol. 5, No. 6, 1992, pp. 927-935.

[5] E. Oja, *Principal components, minor components, and linear neural networks*, Neural Net., Vol. 5, 1992, pp. 927-935.

[6] E. Oja, H. Ogawa, J. Wangviwattana, *Principal component analysis by homogeneous neural networks, part I: the weighted subspace criterion*, IEICE Tran. Inf. Sys., Vol. E75-D, No.3, 1992, pp.366-375.

[7] E. Oja, H. Ogawa, J. Wangviwattana, *Principal component analysis by homogeneous neural networks, part II: analysis and extensions of the learning algorithms*, IEICE Tran. Inf. Sys., Vol. E75-D, No. 3,pp. 366-375.