Progress Report
8-30 April 2013

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 10 days, we were mainly busy testing a new idea to learn non-linear constraints from the input data. Surprisingly, a simple trick does the job, as will be explained in this report.

We also implemented the recall phase for the Non-Linear Minor Component Analysis (NLMCA) idea. It seems that since we have a two-layer neural network, we should go for the "scheduling based" recall process.

I was also working with the semester project students on their projects. In one of the projects, we are working on applying the Maximization of Mutual Inofrmation (MMI) idea to increase the classification rate of an SVM classifier over the CIFAR-10 dataset.

2 Neural Network for Learning Polynomial Non-Linear Constraints

In our work up to this point, we have focused on learning linear (or close to linear) constraints within sample data. More specifically, let $x_1, \dots, x_n$ represent the elements of an n-dimensional input pattern x. We are interested in finding a weight vector w such that $w^\top x = w_i x_1 + \dots + w_n x_n = 0$ (or in general $w^\top x = c$, for some constant c). As we have seen, there are neural algorithms capable of achieving this goal and we could also make them find a w that is sparse.

Now it is a good point to ask ourselves if we could find a similar algorithm which identifies non-linear polynomial constraints, such as $w_1 x_1^2 + w_2 x_1 x_2 + w_3 \sqrt{x_3} = 0$. Interestingly, using a simple trick we could easily transform this problem to the linear case and apply the same learning algorithm we used before to identify non-linear constraints such as the above example.

To start, let us assume to have a two layer neural network, as shown in Figure 1. The connectivity matrices for the first and the second layers are given by W and $\tilde{W}$, respectively. Now, we give $\ln(x) = [\ln(x_1), \dots, \ln(x_n)]$ to the input part of the first layer. This is opposed to the usual method of feeding the first layer by $x = [x_1, \dots, x_n]$. Furthermore, let the activation function of the output neurons of the first layer be $f(.) = exp(.)$.

Therefore, the sate of the output neuron i in the first layer is determined by

$$z_i = f([Wx]_i) = \exp(\sum_{j=1}^{n} W_{ij} \ln(x_j)) = \prod_{j=1}^{n} x_j^{W_{ij}}$$

Thus, the output of the first layer, which is the input to the second layer, gives us the necessary polynomial terms. As a result, we can now apply the standard learning algorithm to identify "linear" constraints in the input of the second layer. Note that constraints are non-linear in x_j's but they are linear in z_i's!

The only necessary modification is that we now have to learn both W and $\tilde{W}$. To this end, we could apply the algorithm given in our previous reports to identify non-linear constraints in a neural network using sigmoid neurons. The algorithm is given below for the sake of completeness.

The objective function is given by

$$\min_{W,\tilde{W}} E = \sum_{x^\mu \in \mathcal{X}} \|y^\mu\|^2 + \text{sparsity penalty} \tag{1a}$$

subject to

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

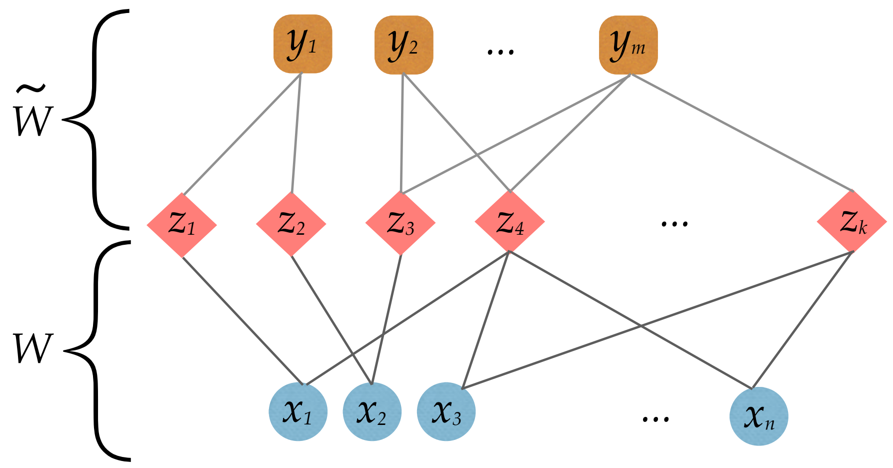

Figure 1: The neural network for learning non-linear polynomial constraints.

and

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

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

$$\tilde{W}(t+1) = \tilde{W}(t) - \tilde{\alpha}_t G'(u(t)) y(t) z(t)^\top, \tag{2}$$

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

$$W(t+1) = W(t) - \alpha_t F'(x(t)) \tilde{W}^\top G'(u(t)) y(t) x(t)^\top, \tag{3}$$

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 (1b) and (1c) are not shown for brevity.

Results: Mostly success: Preliminary results show that the network is capable of learning non-linear constraints.

3 NLMCA: The Recall Phase

In the past two weeks, we also implemented the recall phase for the NLMCA algorithm. Since the proposed architecture has a hidden layer, the recall phase is not as trivial as our previous work. In principle, if both layers are sparse, then one can perform the recall phase for the sub-space algorithm one layer at a time. However, there are two difficulties here:

1. Even if the layers are sparse, a single error in the input layer will reult in multiple inputs in the hidden layer, makes it difficult to correct.

2. The neural states in the second layer are not necessarily integer.

For this reason, we adopted the idea of *scheduling recall* process. It was implemented and tested.

Results: So so: We need further investigation as the performance depends on the quality of the learning phase as well. However, with the success of the NLPoly idea, NLMCA may not be our top prioriy.

References

[1] 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.

[2] 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.