Progress Report
1-15 December 2011

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)

January 19, 2012

1 Summary

In collaboration with Mr. Amin Karbasi, during the last two weeks, I was mainly working on two ideas: extending the neural error correcting algorithm in our ITW paper [2] to multi-level networks, giving it better error correction capabilities and making it more similar to the real world scenarios, as well as extending the approach message passing approach in compressive sensing [**?**] as a framework for the learning phase of our neural algorithms.

In what follows, I will explain how the first subject was rather successful and how our attempt for the second one failed.

2 Multi-level Constraint Enforcing Neural Network

In [2] we proposed a single-level neural network which was capable of using a m linear constraints to correct some errors in the input patterns of length n, assuming that the noiseless pattern satisfies those constraints. The set of constraints were captured in the connectivity matrix of a double-layer neural network, as shown in figure 1, and was assumed to be sparse and expander.

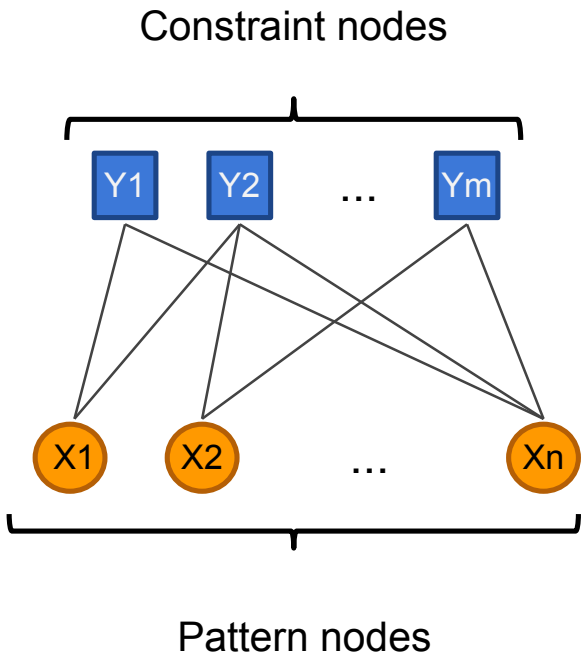

Figure 1: A single-level error correcting neural network [2].

Recently and in our previous report, we proposed an iterative algorithm that could learn the connectivity matrix from the set of training patterns by the means of simple operations in each iteration. The algorithm results in *rather* sparse graphs and although not expander, they will yield

acceptable error correction abilities. In summary, using a non-expander neural network and the error correction algorithm proposed in [2], we are guaranteed to correct one erroneous symbol and from that point on, the probability of correction is a decreasing function of the number of errors.

Therefore, in order to increase the number of correctable errors, we can either work on learning algorithm that yield expander graphs, or use the proposed simple algorithm but play with the architecture of the network. In this report, we focus on the latter. As a result, we have come up wih a neural network that consists of two level, as shown in figure 2.

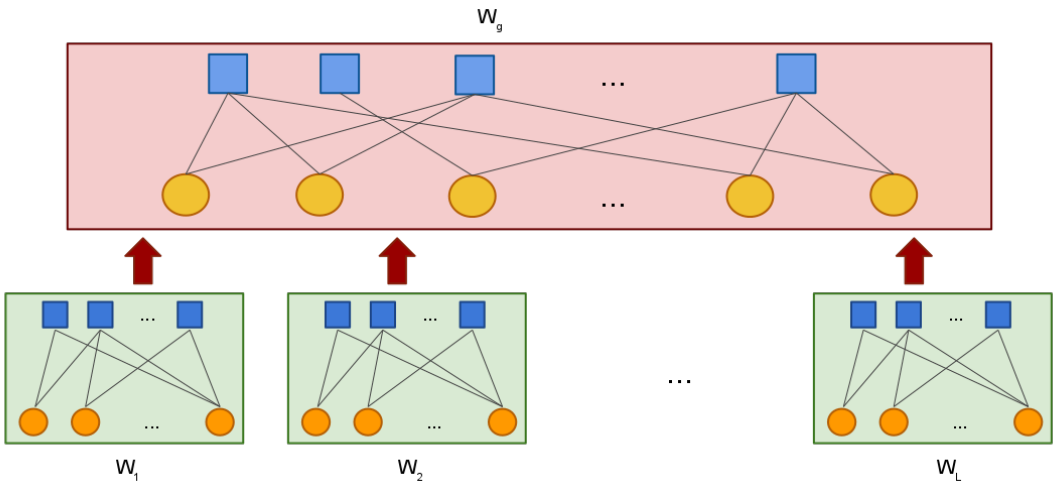

Figure 2: A melti-level error correcting neural network. Each neural box uses the same error correcting algorithm proposed in [2] but with possibly different connectivity matrices.

The idea behind the architecture shown in figure 2 is that we divide the input pattern of size nL into L sub-patterns of length n. Now we feed each sub-pattern to a neural network which enforces m constraints on the sub-pattern in order to correct the input errors. Now from [2] we know that if there is a single error in each sub-pattern, then we are guaranteed to correct them. Therefore, stacking L neural networks we will be able correct L error with 100% correction probability *if* they spread out over the sub-patterns, i.e. we have a single error in each sub-pattern. However, if there happens to be more than one error in a sub-pattern, if it is not corrected we feed the overall pattern of length nL to the second layer with the connectivity matrix W_g, which enforces m_g *global* constraints. And since the probability of correcting two erroneous nodes increases with the input size, we expect to have a better error correction probability in the second layer. Therefore, using this simple scheme we expect to gain a lot in correcting errors in the patterns.

2.1 Some remarks

First of all, one should note that the above method only works if there is some redundancy at the global level as well. If the set of weight matrices $W_1, \ldots, W_L$ define completely separate sub-spaces in the n-dimensional space, then for sure we gain nothing using this method.

However, if there are redundancies in the sub-spaces, we hope that the global constraints become helpful in the error correction process. A simple extreme is the case of repetition, where the subspaces look exactly the same. In this case, different sub-patterns should be equal to each other.

Another remark is that there is no need to have the length of the subspaces to be equal to each other. We can have different lengths for the sub-patterns belonging to each subspace and different number of constraints for that particular sub-space. This gives us more degree of freedom as well since we can spend some time to find the optimal length of each sub-pattern for a particular training data set. In this case, we are looking for constraints within a pattern as well as a set of global constraints that the overall pattern satisfies. This somewhat resembles the real world scenarios where, for instance, we have a word where different subsequences of letters also satisfy certain dependencies. Exploring this direction is one our future subjects of research.

Finally, one should note that the number of constraints for the second layer can be anywhere between $L(n-k) < m_g < nL-k$. The lower bound corresponds to the case that the global graph defined by the W_g is just made by concatenating all the sub-graphs. Therefore, the matrix W_g looks like below in this case:

$$\left(\begin{array}{c|c|c|c} W_1 & 0 & \dots & 0 \\ \hline 0 & W_2 & \dots & 0 \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline 0 & 0 & \dots & W_L \end{array}\right)_{L(n-k)\times nL}$$

The upper bound corresponds to the case that we put a lot more constraints on the patterns than just the ones imposed by the sub-patterns. In this way, although we are reducing the number of patterns that can be memorized, we will make the whole system more robust in the sense that more errors can be corrected.

2.2 Simulation results

We have simulated the proposed multi-level approach to have an estimate of the gain we get in terms of error rates by adding a second level. We constructed 4 sub-networks, each with n pattern and m constraint nodes. However, the subspaces that these patterns came from were different which resulted in different weight matrices $W_1, \dots, W_4$ for these networks. The patterns for each subspace were generated by multiplying a k dimensional random 0/1 vector by a random sparse 0/1 matrix $G_i \in \mathcal{R}^{k\times n}$, where $k = n - m$ and $i = 1, \dots, L$ for different networks in the first level. Then, these patterns were used to train the corresponding neural network, resulting in the weight matrices $W_1, \dots, W_L$. The learning algorithm is the same as the one we discussed in our previous report, which gives rather sparse matrix that are orthogonal to the set of patterns in the training set.

For the global constraint matrix W_g, we use the same algorithm to train over a number of patterns generated with length nL in the same manner as above but now with the generator matrix $G = [G_1|\dots|G_L]$. Therefore, for this layer we have $m_g = nL - k$ constraints, which is much larger than the other extreme $L(n-k)$. Therefore, we are enforcing lots of global constraints in addition to the local ones, which hopefully results in better error rates at the price of reducing the number of patterns that can be memorized.

We generated 10000 random patterns for the training set and trained all the networks over this set. For this report, we have disregarded the potential simultaneous learn-recall pgases because now

that we have four local networks and a big global one, training each of them over 10000 patterns out of possible 2^k will reduce the chances of having common patterns among the local and global networks, specially when k is large, like in our case.

For the recall phase, in each trial we pick a pattern randomly from the training set, corrupt a given number of its symbols with ± 1 noise and use the recall algorithm proposed in [2] to correct the errors. The errors are corrected at two stages: firstly, we divide the long pattern with length nL into smaller sub-patterns of length n and these sub-patterns are fed to the local networks in the first level for errors to be corrected. After convergence, the result is put back together as a pattern with length nL and is given to the global constraint network to correct the remaining errors in the pattern. When finished, we compare the output $\hat{\hat{x}}$ with the original (uncorrupted) pattern x. A pattern error is declared if $\hat{\hat{x}} \neq x$ and the bit error rate is also increased by the number of mismatching bits.

Figure 3 illustrates the pattern error rate at the output of the first and second layer respectively for $n = 90, 300$, $k = 30, 100$, $L = 4$ and $m_g = 330, 1100$, respectively. As one can see, the pattern error rate reduces substantially as we increase n. For 0 and 1 initial number of noisy nodes the final pattern error rate is usually zero and that is the reason there are no point on the plot for these coordinates. The same comment applies to the bit error rate as well.

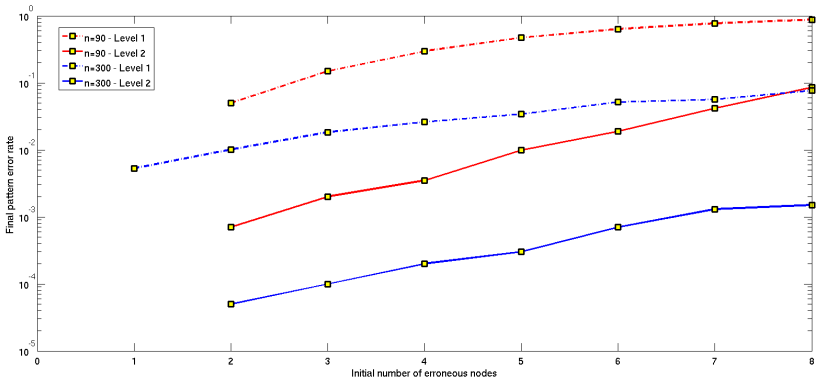

Figure 3: Pattern error rate agains the initial number of erroneous nodes

Figure 4 shows the bit error rates for the same scenario. It is obvious that adding a second layer, we will be able to improve the results to a great extent.

Figure 5 illustrates the gain we get by adding an additional second level to the network architecture. The gain is calculated as the ratio between the pattern (bit) error rate at the output of the first layer and the pattern (bit) error rate at the output of the second layer. Both outputs were compared to the original pattern fed to the network as the input of the first layer. Figure 5 shows that the performance has become many times better, specially for moderate number of initial errors. For small number of initial errors there is no point in using a second layer as even a single-level network is capable of dealing with noise. However, as patterns become more noisy, adding a second level with more global constraints improves the error rate a lot. Note that it is normal for the gain to drop at some point since the pattern or bit error rate of the first layer

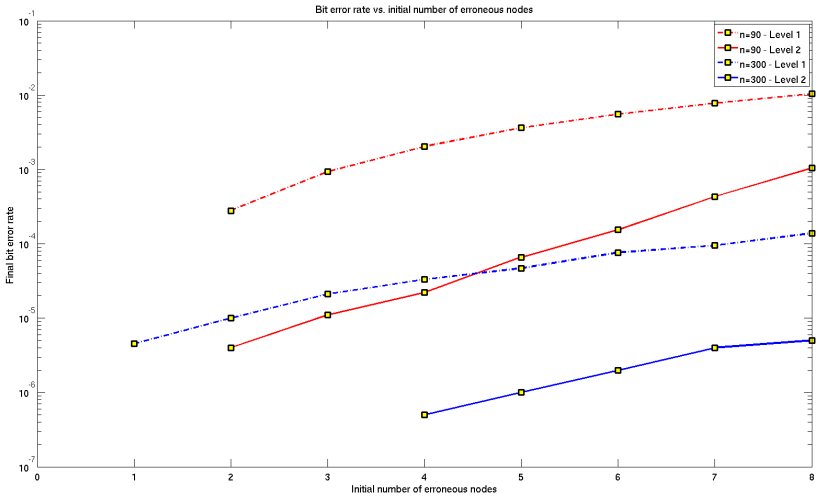

Figure 4: Bit error rate agains the initial number of erroneous nodes

reaches 1 eventually and those of the second layer only increases as a function of the initial number of errors. Please also note that for smaller number of initial noisy nodes, since the output of the second layer is noiseless, which means a PER equal to zero, it is pointless to calculate the gain as it would be infinity.

3 Leaning a Sparse Null Vector Using Iterative Message Passing Algorithms

Suppose we have a matrix $X_{\mathcal{S}\times n}$ composed of non-negative integer entries, where rows of X belong to a subspace with dimension $k < n$. We are interested in finding a vector $w \in \mathcal{R}^n$ such that $X.w = \mathbf{0}$, where $\mathbf{0}$ is the all-zero vector. There are a number of different approaches that can be used to solve this set of equations. However, we are interested in a simple iterative approach that can be implemented by a network of neurons. Note that we are not interested in the *optimal* solution. A sparse vector w (and not necessarily the sparsest such vector) satisfies our needs as we are going to use the given weight vector in the neural associative memory proposed in [2].

Hence, the problem we are interested to solve is

$$\min \|X.w\|_2 \tag{1a}$$

subject to

$$\|w\|_0 \leq q \tag{1b}$$

and

$$\|w\|_2^2 \geq \epsilon \tag{1c}$$

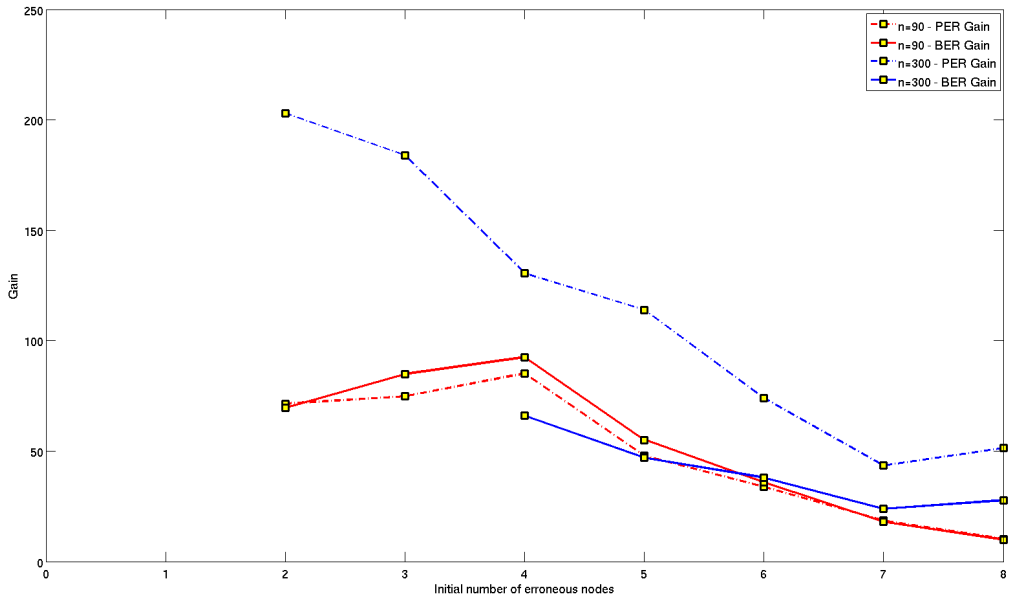

Figure 5: The gain in bit and pattern error rates due to addition of a second layer

where q determines the degree of sparsity and the constraint (1c) ensures that the algorithm will not converge to the all-zero solution.

3.1 The proposed algorithm

To solve the above problem, we first reformulate it as follows:

$$\min \|X.w\|_2 - \lambda(\|w\|_2^2 - \epsilon) + \gamma(g(w) - q') \tag{2}$$

In the above problem, we have approximated the constraint $\|w\|_0 \leq q$ with $g(w) \leq q'$ since $\|.\|_0$ is not a well-behaved function. The function $g(w)$ is chosen such that it favors sparsity. For instance one can pick $g(w)$ to be $\|.\|_1$, which leads to ℓ_1-norm minimizations.

Now in order to find the solution to problem (2), by calculating the derivative of the objective function and primal-dual optimization techniques we propose the following algorithm:

$$y(t) = \frac{X.w(t)}{\|X\|_2} \tag{3a}$$

$$w(t+1) = (1 + 2\lambda_t)w(t) - 2\alpha_t \frac{X^T y(t)}{\|X\|_2} - \gamma_t \nabla g(w) \tag{3b}$$

and

$$\lambda_{t+1} = \Big[\lambda_t + \delta(\epsilon - \|w\|_2^2)\Big] \tag{3c}$$

and

$$\gamma_{t+1} = [\gamma_t + \delta(g(w) - q')] \tag{3d}$$

where t denotes the iteration number, X^T is the transpose of matrix X, δ and α_t are small step sizes and $[.]_+$ denotes $\max(., 0)$.

In the next section, we derive the necessary conditions on α_t, λ_t and γ_t such that algorithm (3) converges to a sparse solution.

3.2 Proof of Convergence

Let $E(t) = \|y(t)\|_{\max}$. We would like to show that $E(t+1) < E(t)$ for all iterations t. To this end, let $f(w) = \nabla g(w)$ be a function from $\mathcal{R}^n$ to $\mathcal{R}^n$. Furthermore, let us denote $(1+2\lambda_t)w(t) - 2\alpha_t \frac{X^T y}{\|X\|_2}$ by $w'(t)$.

Now we have

$$\begin{aligned} E(t+1) &= \|y(t+1)\|_{\max} = \|\frac{X.w(t+1)}{\|X\|_2}\|_{\max} \\ &= \|\frac{X.w'(t)}{\|X\|} - \gamma_t \frac{X.f(w(t))}{\|X\|_2}\|_{\max} \\ &\leq \frac{\|X.w'(t)\|_{\max}}{\|X\|_2} + \gamma_t \frac{\|X.f(w(t))\|_{\max}}{\|X\|_2} \end{aligned} \tag{4}$$

ow expanding $\frac{X.w'(t)}{\|X\|}$ we will get

$$\frac{X.w'(t)}{\|X\|_2} = (1+2\lambda_t)y(t) - 2\alpha_t \frac{XX^T y}{\|X\|_2^2} = \left[(1+2\lambda_t)I_{\mathcal{S}\times\mathcal{S}} - 2\alpha_t \frac{XX^T}{\|X\|_2^2}\right] y(t) \tag{5}$$

Denoting the matrix $(1+2\lambda_t)I_{\mathcal{S}\times\mathcal{S}} - 2\alpha_t \frac{XX^T}{\|X\|_2^2}$ by C_t, we can further simplify inequality (4):

$$\begin{aligned} E(t+1) &\leq \|C_t y(t)\|_{\max} + \gamma_t \frac{\|X.f(w(t))\|_{\max}}{\|X\|_2} \\ &\leq \|C_t\|_{\max}\|y(t)\|_{\max} + \gamma_t \frac{\|X.f(w(t))\|_{\max}}{\|X\|_2} \\ &= \|C_t\|_{\max} E(t) + \gamma_t \frac{\|X.f(w(t))\|_{\max}}{\|X\|_2} \end{aligned} \tag{6}$$

Therefore, in order to show that $E(t+1) < E(t)$, we must write the term $\|X.f(w(t))\|_{\max}$ in terms of $\|X.w(t)\|_{\max}$ so to have an inequality in terms of $E(t)$. Unfortunately, for the functions that put penalty on the non-sparse solutions this can not be done easily. For instance if we let $g(w) = \|w\|_1$, then $f(w) = sign(w)$ (more or less). Then $\|X.f(w(t))\|_{\max}$ is not necessarily less than $\|X.w(t)\|_{\max}$. We have tried other choices such as $g(w) = \sum_{i=1}^n \tanh(\sigma w_i^2)$, which for large values of σ approximates the ℓ_0-norm. However, no success has been made.

From the simulation results we already know that using the above penalty terms (specially the second one which approximates the $\ell - 0$-norm), $E(t+1) < E(t)$ and the error term decreases constantly for moderate to large values of n. However, if n of $\mathcal{S}$ are very small, that is not the case. Hence, may be we have to rewrite the above inequalities in terms of probabilistic approaches.

4 Future Works

In order to extend the multi-level neural network, we must first find a way to generate patterns that belong to a subspace with dimensions $nL - m_g$, where m_g lies *within* the inside of bounds $L(n-k) < m_g < nL-k$. This will give us a way to investigate the trade off between the maximum number of mobilizable patterns and the degree of error correction possible.

Furthermore, so far we have assumed that the second layer enforces constraints in the same layer. However, it is possible that the second layer imposes a set of constraints in a totally different space. For this purpose, we need a mapping from one space into another. A good example is the written language. While they are local constraints on the spelling of the words, there are some constraints enforced by the grammar or the overall meaning of a sentence. The latter constraints are not on the space of letters but rather the space of grammar or meaning. Therefore, in order to for instance to correct an error in the word *_at*, we can replace _ with either *h*, to get *hat*, or *c* to get *cat*. Without any other clue, we can not find the correct answer. However, let's say say we have the sentence "*The _at ran away*". Then from the constraints in the space of meaning we know that the subject must be an animal or a person. Therefore, we can return *cat* as the correct answer. The overall system looks like figure 6. Finding a proper mapping is the subject of our future work.

On a similar direction, we might also think about decoders with side information. In the example above, the fact that we knew the answer is an animal helped us correct the *spelling*. In a neural network, we might think about this idea as having a Bidirectional Associative Memory (BAM) where on one side we have the patterns and on the other side we have a set of features associated with these patterns. Now when we are faced with a corrupted pattern, we use the information that we get from the correlation inside a pattern as well as external information to correct the pattern. In this case, side information will be in terms of fixing a set of features such as *being a living being*. Exploring this direction may result in more robust associative memories and/or practical fast error correcting codes.

Regarding the work on sparse learning, we already had some thoughts how to overcome the technical issues we face in the proof of convergence. However, extending this current proof to probabilistic arguments is another direction which we have to do some time in future.

References

[1] D. L. Donoho, A. Maleki, A. Montanari, *Message passing algorithms for compressed sensing*, Proc. Nat. Acad. Sci., Vol. 106, 2009, pp. 1891418919.

[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] 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.

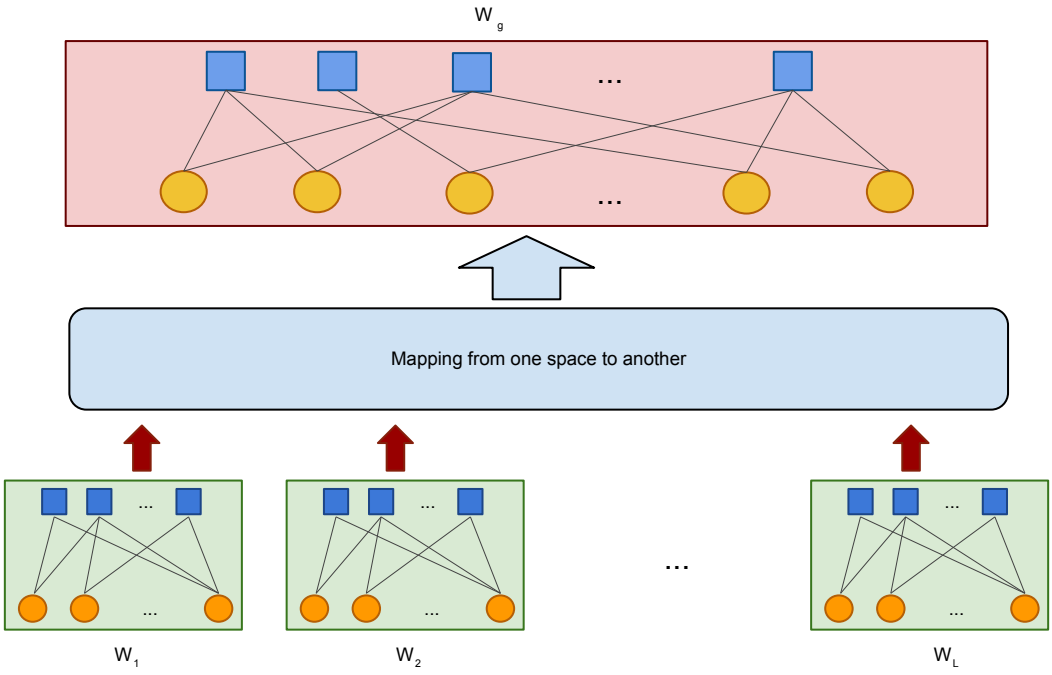

Figure 6: A melti-level error correcting neural network over two different spaces. Each neural box uses the same error correcting algorithm proposed in [2] but with possibly different connectivity matrices. The mapping then transforms the result to another space where it is treated with a different set of constraints.