POLITECNICO DI TORINO

Master's Degree in Physics of Complex Systems

Master's Degree Thesis

Learning capabilities of belief propagation based algorithms for sparse binary neural networks

Supervisors

Prof. Luca DALL'ASTA

Prof. Jean BARBIER

Candidate

Chen Yi ZHANG

October 2020

Abstract

With the growth in size and complexity of modern data sets, the computational and energetic costs of training large, fully connected neural networks, became an important issue. Therefore exploring the learning capabilities of sparse (possibly binarized) architectures, that possess many less degrees of freedom but empirically appear to generalize well, is an important research direction.
In this work, the learning performance of belief propagation (BP) based algorithms, applied to simple two layers sparse neural networks with discrete synapses, is analysed. Initially, the framework in which the work is carried on is the so called teacher-student scenario, in which the learning problem corresponds to the inference of the weight values of a teacher network whose architecture is given. In this first part BP provides encouraging results, allowing to perfectly reconstruct the weights of the 'teacher' network that generated the training data, using only a small number of data points.
Subsequently, the focus shifts to the mismatched setting with a discrepancy between the used architecture and the one of the 'teacher' network. The results are analysed in order to assess whether BP-based learning is relevant in this case as well, and if yes, to which extent of mismatch this is possible.

Acknowledgements

First of all I have to thank Jean and Manuel for making me always feel like a peer of theirs and for making the work throughout these months very enjoyable despite the difficult conditions. I also have to thank Federico Ricci Tersenghi for the interesting and instructive conversations that played a big part in this thesis work. Thanks to Alfredo Braunstein as well for the support and for tolerating my ravings on convolutions, even during the summer vacations.

Grazie a mamma e papà per i sacrifici estremi che hanno da sempre fatto e continuano a fare per me e per mio fratello Calvin. Ringrazio Calvin per essere la persona di cui posso sempre essere orgoglioso e infine ringrazio Fraus, che da un paio d'anni a questa parte mi accompagna in questa avventura costellata di difficoltà, che vissute in compagnia non sono poi risultate così insormontabili.

"2020 will be the sparse network year (it already has two zeros, that's a sign)"
François Lagunas

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 A little bit of motivation

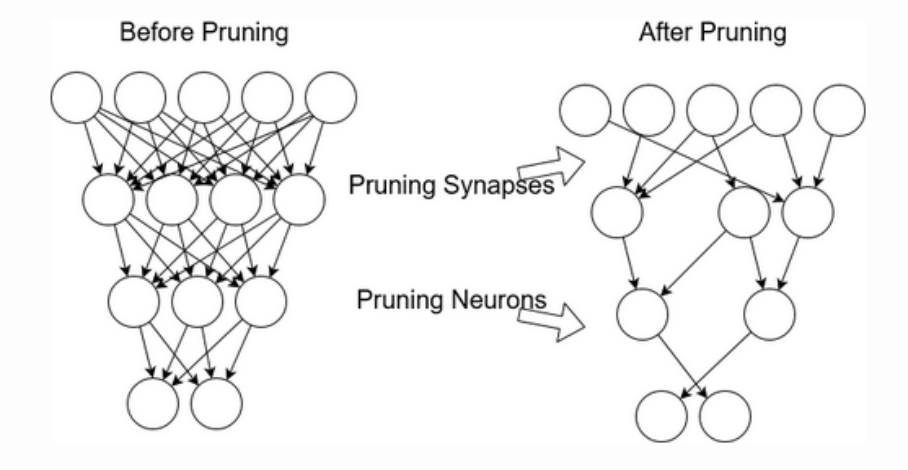


Figure 1.1: Example of pruning.
Image from: https://software.intel.com/content/www/us/en/develop/articles/compression-and-acceleration-of-high- dimensional-neural-networks.html

The successes of algorithms based on deep neural networks in finding patterns in increasingly large data sets, with growing accuracy levels, unfortunately comes along with also an increase in computational resources requirements. The number of parameters that need to be learnt in state-of-the-art neural networks has reached a point where training them, storing them and using them has become almost unpractical, especially in situations where the computational budget is low (e.g. mobile devices), therefore making it vital to find ways to reduce their size.

There are many so-called *model compression* techniques, that are used to reduce the size of already trained networks. The resulting models are often shown to perform in a comparable way to the original networks, while using a small fraction of the computational resources (in some instances the size of the systems can be shrunk by up to 90%), see [1] [2].

The reason behind why the compressed models perform well despite the enormous loss of information that they undergo, intuitively relies on the fact that the current optimization techniques work well on networks that are heavily overparametrised. This means that the networks that have to be trained have many more parameters that those that are actually needed to fit the data, therefore causing the resulting trained models to contain a lot of redundant information that in numerous cases can be removed without affecting significantly their performance.

One important example of model compression technique consists of the so called pruning algorithms (see Fig.1.1) that are typically implemented in three main steps:

1. Training a large densely connected neural network
2. Pruning: in this step redundant weights are removed (or *pruned*) and only the most relevant ones are kept; usually a large percentage of the trained weights are very small and are therefore set to zero, without affecting too much the accuracy of the model.
3. Fine-tuning of the survived weights to improve accuracy

Pruning efficiently solves the problem of the high cost of storing and using trained models (in terms of memory and speed), but the training part of the computational resources problem is still present. It is true that an active field of research is that of looking for new and more ingenious ways to do pruning and better optimization techniques, in order to reduce the cost of training (see for example [3] and [4]), however there is always at some point the need to train (at least partially) big and densely connected networks. Indeed it is still not very well understood why commonly used optimizers in state-of-the-art deep learning fail when one wants to train networks that are sparse, from scratch [5].

There is a vast literature about the successes of belief propagation algorithms (for an introduction see Appendix A) in solving complicated constrained optimization problems. One can find many examples by looking for instance at [6][7] and [8]. Furthermore there is a strong link between belief propagation and the statistical physics of disordered systems (it is often referred to as *cavity method* by physicists), and this yields the advantage that when some experimental results come up, one has already a strong theoretical framework to work in, while looking for analytical results to compare with the experimental ones.

These reasons triggered the attempt, reported in this brief work, of designing and implementing a BP algorithm to start tackling the problem of learning sparse neural networks from scratch.

1.2 Setting: the model under study

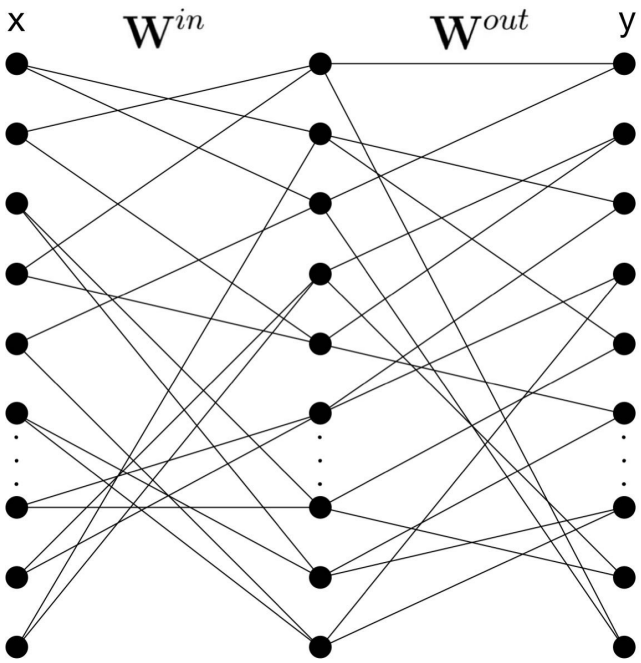


Figure 1.2: Example of two-layer sparse neural network

For the sake of simplicity and in order to work in a more controlled setting, the studied model is neither the most complex nor the most generic one, but despite the apparent simplicity of the system under study hopefully the results will be able to provide some insight into the learning capabilities of BP-based algorithms for SNN's.

Specifically let us consider two-layer networks, which means that there are three layers of *neurons*, respectively of sizes N_{in}, N_{hidden} and N_{out} (see Figure 1.2).

The N_{out}-dimensional output vector $\mathbf{y}$ is related to the N_{in}-dimensional input $\mathbf{x}$ by

$$\mathbf{y} = \phi\left(\mathbf{W}^{out} * \psi\left(\mathbf{W}^{in} * \mathbf{x}\right)\right) \tag{1.1}$$

where

- ϕ and ψ are the non-linearities in the system. They are functions acting element wise on their arguments (also called activation functions). The results reported in the following are obtained using as functions $tanh(x - b)$ with $b \in \mathbb{R}$ and $max\{x, ax\}$ with $a < 1$, also called *leaky ReLU*.
- $\mathbf{W}^{in*}$ and $\mathbf{W}^{out*}$ are the matrices of the weights in the two junctions, respectively between the input and hidden layer, and between the hidden and the output layer. The sparsity of the network is encoded in the fact that the number of non-zero elements in these matrices scales linearly with the sizes of the layers (and not quadratically as it would in a dense NN).
- The non-zero weights of $\mathbf{W}^{in*}$ and $\mathbf{W}^{out*}$ and the components of $\mathbf{x}$ take values in $\{-1, +1\}$

Always in the spirit of reducing the computational resources needed in deep learning, alongside techniques such as pruning, there is something called *quantized neural networks* (QNN's), which are networks where the precision of the weights is reduced, for instance by passing from real to integer values, with the extreme case being weights which allow a 1-bit representation i.e. binary weights. This quantization allows for enormous drops in the storage and usage costs of neural networks, and nonetheless in many instances the resulting models proved to perform quite well on popular data sets (e.g. MNIST, CIFAR-10 and SVHN) [9] [10].
The good performances of these quantized neural networks together with the fact that belief propagation is an algorithm that works with discrete variables, justify the choice to use binary valued weights.

Chapter 2

Bayes optimal case

In this setting there is a *teacher* that generates a random sparse neural network (i.e. $\mathbf{W}^{in\,*}$ and $\mathbf{W}^{out\,*}$), by randomly choosing the positions of the non-zero entries of the weight matrices and their values. Then it generates a certain number m of input vectors $\{\mathbf{x}_\eta\}_{\eta=1,\dots,m}$ and computes the associated outputs $\{\mathbf{y}_\eta\}_{\eta=1,\dots,m}$ by applying the model (1.1).

In particular, in the next experiments when the *teacher* generates the network, the connectivity of each node is fixed beforehand and all the nodes in the same layer have the same degree, which is to say that two consecutive layers of nodes in the network form a biregular graph (see Fig. 2.1).

Subsequently, the *student* is provided with the data, i.e. the input-output pairs $\{(\mathbf{x}_\eta;\, \mathbf{y}_\eta)\}_{\eta=1,\dots,m}$, and his objective is to learn from it. Being in the Bayes optimal case the *student* knows the model of the *teacher* exactly, that is he knows the positions of the non-zero weights. Therefore the objective reduces to the inference of the values only of the non-zero entries of the weight matrices.

More formally we can say that being in the Bayes optimal setting means that the teacher, i.e. the model that generates the data, hands over to the student the full and exact functional form of the following posterior distribution:

$$\begin{aligned} &P\left(\mathbf{W}^{in}, \mathbf{W}^{out} \,|\, \{\mathbf{y}_\eta\}_{\eta=1,\dots,m},\, \{\mathbf{x}_\eta\}_{\eta=1,\dots,m}\right) \propto \\ \propto\; &P\left(\{\mathbf{y}_\eta\}_{\eta=1,\dots,m},\, \{\mathbf{x}_\eta\}_{\eta=1,\dots,m} \,|\, \mathbf{W}^{in}, \mathbf{W}^{out}\right) \cdot P\left(\mathbf{W}^{in}, \mathbf{W}^{out}\right) \propto \\ \propto\; &P\left(\{\mathbf{y}_\eta\}_{\eta=1,\dots,m} \,|\, \mathbf{W}^{in}, \mathbf{W}^{out},\, \{\mathbf{x}_\eta\}_{\eta=1,\dots,m}\right) \cdot P\left(\mathbf{W}^{in}, \mathbf{W}^{out}\right) \end{aligned} \tag{2.1}$$

With the knowledge of the posterior distribution the student has to find the best estimates of the parameters of the model, but to define in a precise way what 'best estimate' means we first need to decide an appropriate error metric that quantifies

the quality of the inference. Since the weights in the problem are discrete valued, a meaningful error metric is the so-called *overlap*

$$O(\hat{\mathbf{w}},\, \mathbf{w}^*) \,=\, \frac{1}{N}\sum_i \delta_{w_i^*,\, \hat{w}_i} \tag{2.2}$$

where the estimators are designated by the hat, the ground-truth parameters with a * and N si the total number of parameters to be learned (i.e. the number of non-zero entries in the teacher weight matrices).
Since the student does not know the ground-truth (i.e. $\mathbf{W}^{in\,*}, \mathbf{W}^{out\,*}$), in the framework of Bayesian statistics what he can do is to use as error metric the average of the overlap with respect to the posterior distribution (2.1), also called *mean overlap (MO)*. Then by maximizing *MO* with respect to $\hat{\mathbf{w}}$, one obtains that the optimal estimator is the so called *Maximum Mean Overlap estimator* (*MMO*)

$$\hat{w}_i^{MMO} \,=\, \underset{w_i}{\operatorname{argmax}}\, \mu_i(w_i) \tag{2.3}$$

Where $\mu_i(w_i)$ are the posterior marginals.

2.1 Factor graph representation

As anticipated above the main quantities of interest are the marginals of the posterior distribution of Eq. (2.1). With the knowledge of the correct functional form of the latter the student can carry out the marginalization task using the so-called belief propagation algorithm (BP). For a brief introduction on the subject and for the notation that will be used throughout the next pages refer to Appendix A.

In order to implement the algorithm one needs first to define the factor graph that represents the aforementioned joint posterior distribution. However since the prior of the weights is uniform, its contribution is a constant that can be included in the normalization factor. Therefore the important quantity which is necessary to understand the shape of the factor graph is the likelihood

$$\begin{aligned} P(\{\mathbf{y}_\eta\}_{\eta=1,\dots,m} \,|\, \mathbf{W}^{in}, \mathbf{W}^{out},\, \{\mathbf{x}_\eta\}_{\eta=1,\dots,m}) \,= \\ = \prod_{a=1}^{N_{out}} \xi_a(\mathbf{w}_{\partial a}^{in};\, \mathbf{w}_{\partial a}^{out}) \end{aligned} \tag{2.4}$$

where the factors $\xi_a(\mathbf{w}_{\partial a}^{in};\, \mathbf{w}_{\partial a}^{out})$ are defined as

$$\xi_a(\mathbf{w}_{\partial a}^{in};\, \mathbf{w}_{\partial a}^{out}) \,=\, \prod_{\eta=1}^{m} \delta\left(y_{\eta,a} \,-\, \phi\left(\sum_{r=1}^{N_{hidden}} W_{ar}^{out}\, \psi\left(\sum_{s=1}^{N_{in}} W_{rs}^{in}\, x_{\eta,s}\right)\right)\right) \tag{2.5}$$

This expression suggests a factor graph representation with N_{out} factor nodes (one associated with each neuron in the output layer of the SNN) and one variable node for each non-zero weight. $\mathbf{w}_{\partial a}^{out}$ and $\mathbf{w}_{\partial a}^{in}$ correspond respectively to the non zero entries of the a-th row of the output matrix (with column indices $\{r\}$) and of the $\{r\}$-th rows of the input matrix. See Fig. 2.1.

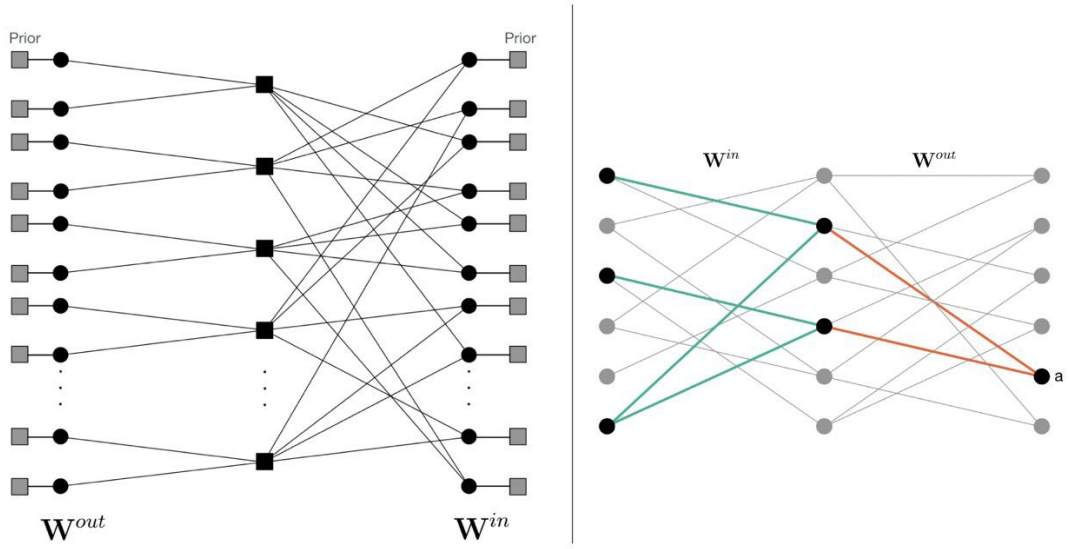


Figure 2.1: The first image is an example of factor graph representation of the problem. In the middle are the factor nodes, on the left the variable nodes corresponding to the output matrix weights, on the right the ones for the input matrix. The grey factors represent the prior terms in the context of mini-batch/online learning (see section 2.2).
The second image is a pictorial representation of the weights involved in a certain factor. The variables $\mathbf{w}_{\partial a}^{out}$ are highlighted in orange, while $\mathbf{w}_{\partial a}^{in}$ in green.

2.1.1 BP equations

Once the factor graph has been defined, the belief propagation algorithm (again see Appendix A) consists in updating some quantities called messages (which are associated to the edges of the factor graph) following the equations reported below.

For implementation convenience, among the variable nodes in the factor graph we distinguish between the ones associated to the input matrix weights and the ones associated to the output matrix weights. In particular the letters i, i', ... are used for the indexing of the former, j, j', ... for the latter, a, a', ... refer to the factor nodes and t counts the number of BP iterations.

Since the nodes associated with the weights in the output matrix are leaves in the factor graph, $\mu^{(t)}_{j\to a}(w^{out}_j) = \frac{1}{2}\ \forall t, a, j, w^{out}_j$, so there is no need to compute these messages. The update rules for the remaining messages are:

1. Message from the variable node w^{in}_i to the *a-th* factor node:

$$\mu^{(t+1)}_{i\to a}(w^{in}_i) \ \cong \prod_{a'\in\partial i\setminus a} \hat{\mu}^{(t)}_{a'\to i}(w^{in}_i) \tag{2.6}$$

2. Message from the *a-th* factor node to the variable node w^{in}_i:

$$\begin{aligned} &\hat{\mu}^{(t)}_{a\to i}(w^{in}_i) \ \cong \\ \cong &\sum_{\substack{\{w^{out}_j\},\, j\in\partial a \\ \{w^{in}_{i'}\},\, i'\in\partial a\setminus i}} \xi_a(\mathbf{w}^{in}_{\partial a\setminus i};\, w^{in}_i;\, \mathbf{w}^{out}_{\partial a}) \prod_{i'\in\partial a\setminus i} \mu^{(t)}_{i'\to a}(w^{in}_{i'}) \end{aligned} \tag{2.7}$$

3. Message from the *a-th* factor node to the variable node w^{out}_j:

$$\begin{aligned} &\hat{\mu}^{(t)}_{a\to j}(w^{out}_j) \ \cong \\ \cong &\sum_{\substack{\{w^{out}_{j'}\},\, j'\in\partial a\setminus j \\ \{w^{in}_i\},\, i\in\partial a}} \xi_a(\mathbf{w}^{in}_{\partial a};\, \mathbf{w}^{out}_{\partial a\setminus j};\, w^{out}_j) \prod_{i\in\partial a} \mu^{(t)}_{i\to a}(w^{in}_i) \end{aligned} \tag{2.8}$$

where the symbol $\cong$ is used to indicate that the left and right hand sides of the equations are equal up to normalization.

Notice that last messages do not need to be updated at each step t. In fact being directed towards leaves of the factor graph, their value does not affect any other message in the updates. Therefore one can run BP only for the messages coming from and going towards the nodes associated to the weights of the input matrix. Then after convergence has been reached one can first compute the messages going towards the nodes associated to the output weights and finally evaluate the so sought-after marginals as in Eq. (A.7)

2.2 Offline, mini-batch and online learning

In the context of machine learning, algorithms that process the available data all at once are often referred to as *offline* (such as the BP algorithm described above). In contrast to this there are the so called *mini-batch* algorithms, that carry out the learning task sequentially, as the name suggests, on small batches of data (the extreme case where the size of the batch is one is called *online*).[11]

Starting from the factor graph and the BP equations reported above, with a few minor adjustments, one can implement the mini-batch and the online algorithms and compare their performance with the offline version.

Let us denote by $\tilde{m}$ the number of data points in each mini-batch (the batch size), by $\mathbf{y}^{(k)}$, $\mathbf{x}^{(k)}$ the data points of the *k-th* batch and by $\mathcal{D}^{(k-1)} = \{\mathbf{y}^{(1)}, ..., \mathbf{y}^{(k-1)}, \mathbf{x}^{(1)}, ..., \mathbf{x}^{(k-1)}\}$ the data from the previously used batches. In this setting the posterior distribution when the *k-th* batch is made available reads:

$$\begin{aligned} P\left(\mathbf{W}^{in}, \mathbf{W}^{out} \,|\, \mathbf{y}^{(k)}, \mathbf{x}^{(k)}, \mathcal{D}^{(k-1)}\right) \propto \\ \propto P\left(\mathbf{y}^{(k)} \,|\, \mathbf{W}^{in}, \mathbf{W}^{out}, \mathbf{x}^{(k)}\right) \cdot P\left(\mathbf{W}^{in}, \mathbf{W}^{out} \,|\, \mathcal{D}^{(k-1)}\right) \end{aligned} \tag{2.9}$$

From this expression, one can see that the most significant difference with respect to what has been described in the previous section is that there is not an uniform prior that one can include in the normalization constant any more. Indeed now the 'prior' term $P\left(\mathbf{W}^{in}, \mathbf{W}^{out} \,|\, \mathcal{D}^{(k-1)}\right)$ is the posterior distribution that one obtains after the previous mini-batches have been used. Unfortunately the actual posterior is not available, as what BP provides are estimates of the posterior marginals. Therefore one can resort to the following approximation:

$$P\left(\mathbf{W}^{in}, \mathbf{W}^{out} \,|\, \mathcal{D}^{(k-1)}\right) \approx \prod_i \mu_i^{(k-1)}(w_i^{in}) \prod_j \mu_j^{(k-1)}(w_j^{out}) \tag{2.10}$$

where $\mu_i^{(k-1)}(w_i^{in})$ and $\mu_j^{(k-1)}(w_j^{out})$ denote the BP fixed point beliefs, obtained after using $\mathcal{D}^{(k-1)}$.

The implementation of this factorized prior term consists in connecting, in the factor graph described in the previous section, each variable node to an extra degree-1 factor node that contains the information about all the previous mini-batches. The associated factors read:

$$\begin{aligned} \xi_i^{(k)}(w_i^{in}) &= \mu_i^{(k-1)}(w_i^{in}) \\ \xi_j^{(k)}(w_j^{out}) &= \mu_j^{(k-1)}(w_j^{out}) \end{aligned} \tag{2.11}$$

It is interesting to notice that, in a statistical physics analogy, the weights to be learnt are the equivalent of an Ising spins system, the N_{out} factors indexed by a are couplings (of course there are not just pairwise interactions), and these additional factors introduced in the mini-batch setting are external fields, updated as new mini-batches are used.

2.3 The Max-Product algorithm

Belief propagation carries out the specific task of computing marginals of complex probability distributions, but it also belongs to a wide class of algorithms that

share the same structure, called message-passing algorithms, that allow to compute different quantities.
For instance, let us take the problem of finding a configuration $\mathbf{x}$ that maximizes $P(\mathbf{x})$. If $P(\mathbf{x})$ is a posterior distribution solving this problem would amount to do a MAP (maximum-a-posteriori) estimation of $\mathbf{x}$. One can accomplish this task by doing one slight change to the BP algorithm described in appendix A: in Eq. (A.6) for the update of the factor node to variable node messages, replace the summation over configurations with a maximization i.e.

$$\hat{\mu}^{(t)}_{a\to i}(x_i) \propto \max_{\underline{x}_{\partial a\setminus i}} \left\{ \xi_a(\underline{x}_{\partial a}) \prod_{j\in\partial a\setminus i} \mu^{(t)}_{j\to a}(x_j) \right\} \tag{2.12}$$

It is not hard to show that (see [6]) analogously to how the beliefs computed with the fixed point BP messages yield exact marginals for tree-graphical models, the beliefs computed with the fixed point max-product messages yield exactly the following quantities, called max-marginals

$$M_i(x_i^*) = \max_{\mathbf{x}} \{P(\mathbf{x}) : x_i = x_i^*\} \tag{2.13}$$

If $\forall i$ the maximum of $M_i(\cdot)$ is non degenerate, then by maximizing each max-marginal one would obtain for a tree graphical model the unique configuration $\mathbf{x}^*$ that maximizes $P(\mathbf{x})$.

The same experiments have been carried out both with the belief propagation (or sum-product) and the max-product algorithm, so that one can check that the optimal estimator that maximizes the overlap (see Eq. (2.2)) is indeed the one that maximizes the posterior marginals, and not the MAP estimator. This is also a further way to check that the algorithms have been properly implemented.

2.4 Experimental results

In this section we show some experimental results relative to the teacher student scenario in the Bayes optimal case. In the following some implementation details about the first experiments (see fig. 2.2 and 2.3) are described:

- The two activation functions that have been used are:
  1. leaky ReLU: $\phi(x) = max(x;\, ax)$
  2. hyperbolic tangent: $\phi(x) = tanh(x - b)$

  In all experiments the same non-linearity has been used for both layers.
  We chose $a = b = 0.1$

- The sizes of the layers are: $N_{in} = N_{hidden} = N_{out} = 1000$
- The teacher network is such that its weight matrices have two non-zero entries for each row and for each column.
- The simulations have been carried out using both the BP (or sum-product) and the max-product algorithms. Furthermore each experiment has been done in the offline, online and mini-batch variations, with mini-batch sizes $\tilde{m}_1 = 2$ and $\tilde{m}_2 = 4$
- Often times in algorithms that are designed to numerically solve fixed-point equations (such as belief propagation) one can introduce a so called *damping* in order to improve convergence properties. At each step of the algorithm, this consists in updating the messages with the weighted average between their old value and the value they would be assigned according to the message passing equations.

$$\mu_{(\cdot)}^{(t+1)} \leftarrow \alpha \mu_{(\cdot)}^{(t)} + (1-\alpha) f_{MP}(\mu_{(\cdot)}^{(t)}) \tag{2.14}$$

  where $f_{MP}(\cdot)$ refers to a generic message passing update rule and $0 \leq \alpha \leq 1$. The results shown in the following are obtained with $\alpha = 0{,}3$.

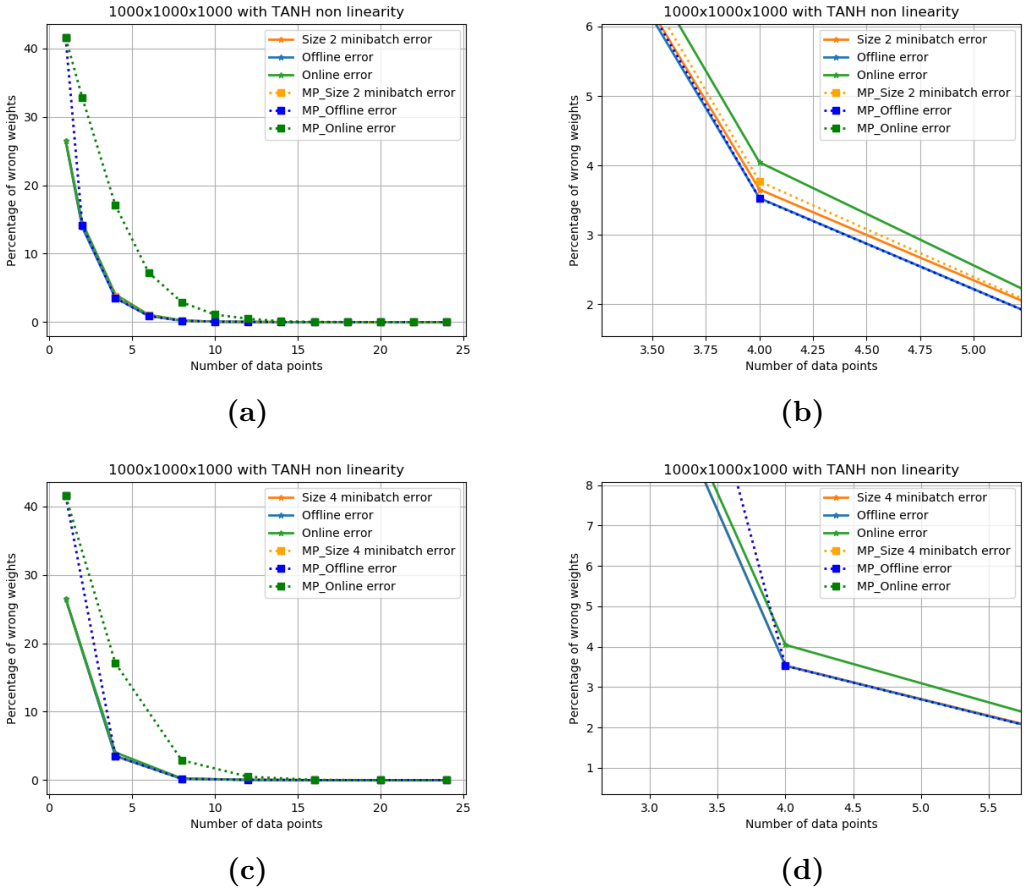


Figure 2.2: In this figure the results obtained with the $\phi(x) = tanh(x - b)$ non-linearity with $b = 0{,}1$ are shown. On the vertical axis there is the fraction of the student weight estimates that differ from the true teacher weights. In the sub-figures on the right there are the close ups of the ones shown on the left. Each plot is obtained by averaging over 10 simulations.

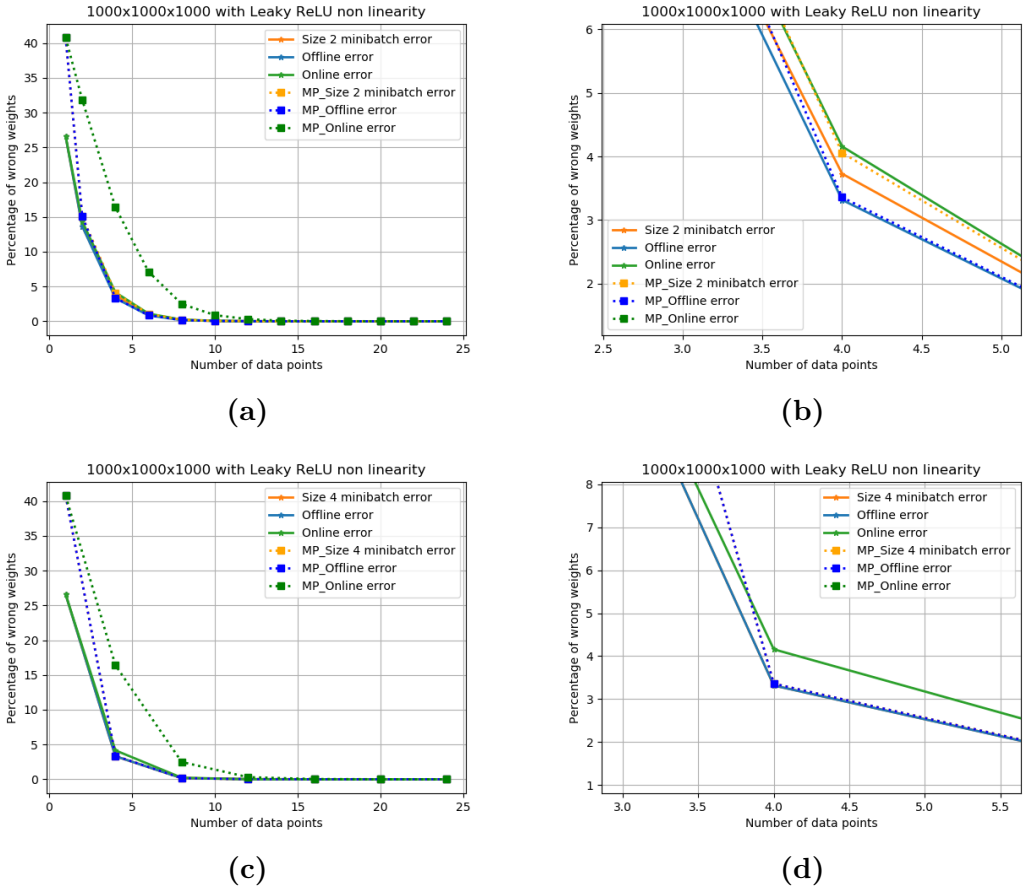


Figure 2.3: In this figure the results obtained with the $\phi(x) = max(x; ax)$ non-linearity with $a = 0{,}1$ are shown. On the vertical axis there is the fraction of the student weight estimates that differ from the true teacher weights. In the sub-figures on the right there are the close ups of the ones shown on the left. Each plot is obtained by averaging over 10 simulations.

One can see from the Fig. 2.2 and 2.3 in the last two pages that in the teacher student scenario, belief propagation allows to perfectly reconstruct the teacher weights using only a small number of data points. Furthermore, from the close-up sub-figures on the right of both images one may notice how the online and mini-batch versions of the algorithm perform very comparably with the offline variant: indeed for mini-batches of size $\tilde{m} = 4$ the difference practically disappears. This fact may potentially turn out to be useful in instances where an offline algorithm is unfeasible: for an example see Appendix B.
Furthermore it is interesting to observe that Fig. 2.2 and Fig. 2.3 contain plots that behave in a extremely similar way, despite coming from experiments with

networks containing different non-linearities.

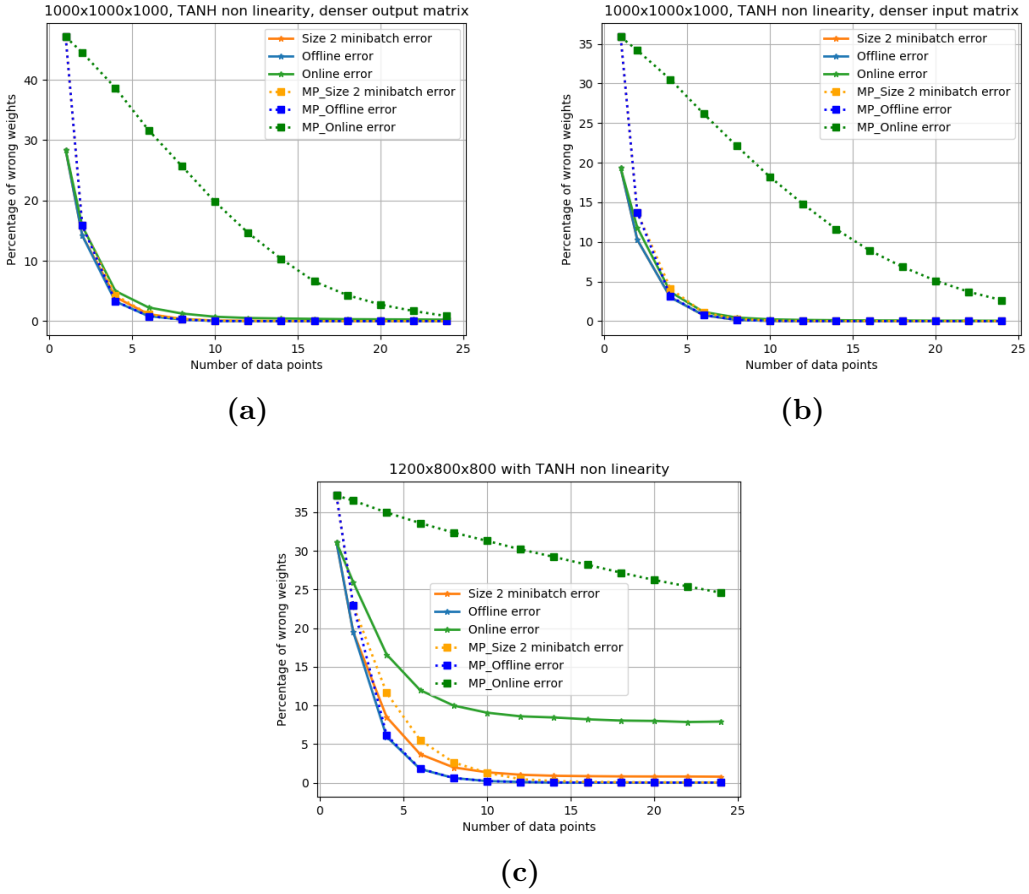


Figure 2.4: The plots shown in this figure refer to experiments carried out in a similar fashion to what is shown in Fig. 2.2 and Fig. 2.3 (same mini-batch sizes, same damping, etc.), but with a few differences in the implementation details. In subfig. **(a)** the output matrix has three non-zero elements in each row and each column instead of two as in the previous experiments. In subfig.**(b)** the denser matrix is the input one. In subfig.**(c)** the sizes of the layers are $N_{in} = 1200, N_{hidden} = 800, N_{out} = 800$, the input matrix has three non-zero elements per row and the output one has two. All these experiments have been carried out with the activation function $\phi(x) = tanh(x - b)$ with $b = 0{,}1$ and the plots have been obtained by averaging over 10 simulations each.

In addition to the simulations above, other experiments have been carried out with different network topologies obtained by varying the weight matrices densities

and the relative sizes of the layers in the SNN. In Fig. 2.4 one can see that introducing these differences does not change significantly the results in the offline case.
The largest degree of variability can be witnessed for the online max-product algorithm. However this may due to the fact that, while in the online version of the standard BP the expression in Eq. (2.10) amounts to approximating a certain joint distribution as the product of its marginals, by neglecting the correlations between the variables, if instead the message passing algorithm beliefs represent the max-marginals of Eq. (2.13) it is not clear what this approximation means.

Chapter 3

Mismatched case

3.1 Realizable rule setting

In light of the encouraging results obtained by belief propagation in the teacher-student scenario, the last part of this work aims to see whether BP-based learning can be relevant also in gradually more realistic settings, in which the architecture that one tries to train is different from the one that generated the data.

As a first step, we consider the so-called *realizable rule setting* in which the model of the student, although being different from the one of the teacher, contains the latter. In particular let us decompose the weight matrices as an element-wise product between a matrix $\mathbf{A}^{(\cdot)}$ that defines the topology of the network and a matrix $\mathbf{J}^{(\cdot)}$ that contains the values of the weights, i.e. $W_{i,j}^{(\cdot)} = A_{i,j}^{(\cdot)} J_{i,j}^{(\cdot)}$, with $A_{i,j}^{(\cdot)} \in \{0,1\}$. Working in the realizable rule setting means that the student network topology matrices $\mathbf{A}^{in}$ and $\mathbf{A}^{out}$ contain the ones of the teacher network $\mathbf{A}^{in\,*}$ and $\mathbf{A}^{out\,*}$ (i.e. $A_{ij}^{(\cdot)\,*} = 1 \implies A_{ij}^{(\cdot)} = 1$), and that $J_{i,j}^{in\,*}$, $J_{i,j}^{out\,*} \in \{-1, +1\}$ while $J_{i,j}^{in}$, $J_{i,j}^{out} \in \{-1,0, +1\}$.

The reason behind this choice is that in this setting one knows that the lowest reachable generalization error is zero, which provides an actual reference that can be used to assess how good the performance of the learning is.

3.2 Log-likelihoods and the max-sum algorithm

In this section we introduce a variation of the belief propagation algorithm in order to compare it with the standard BP. The reason for this is that once one moves away from the realizable rule setting and starts working with an arbitrary mismatch between the teacher and the student model, BP may not be the most suitable choice to tackle the problem. More precisely the criticality comes from the

update of the factor to variable nodes messages described in Eq. (2.7) and (2.8). Indeed if the mismatch between the student and the teacher is arbitrary it is not granted that the former would be able to reproduce the latter. Therefore it may be possible that the factors $\xi_a(\cdot)$ that appear in the aforementioned equations are zero for all the configurations of the neighbourhood of the corresponding factor node in the factor graph of Fig. 2.1. If this happens, then all the messages coming out of the factor nodes with the described issue will have the same value, which means that they do not provide any information for the estimation of the weights, which may in turn prevent the convergence of the algorithm.

The first step to make in order to obtain the aforementioned variation on BP is to approximate the factors of Eq. (2.5), with the following Boltzmann distributions:

$$\begin{aligned} \xi_a(\mathbf{w}_{\partial a}^{in};\, \mathbf{w}_{\partial a}^{out}) \;=\; \prod_{\eta=1}^{m} \delta\Bigg(y_{\eta,a} \;-\; \phi\Bigg(\sum_{r=1}^{N_{hidden}} W_{ar}^{out}\, \psi\Big(\sum_{s=1}^{N_{in}} W_{rs}^{in}\, x_{\eta,s} \Big) \Bigg) \Bigg) \;\simeq \\ \simeq\; \exp\Big[-\beta \mathcal{H}_a(\mathbf{w}_{\partial a}^{in};\, \mathbf{w}_{\partial a}^{out}) \Big] \end{aligned} \tag{3.1}$$

where β is the inverse temperature and the *Hamiltonian* reads

$$\mathcal{H}_a(\mathbf{w}_{\partial a}^{in};\, \mathbf{w}_{\partial a}^{out}) \;=\; \sum_{\eta=1}^{m} \left[y_{\eta,a} - \phi\left(\sum_{r=1}^{N_{hidden}} W_{ar}^{out}\, \psi\left(\sum_{s=1}^{N_{in}} W_{rs}^{in}\, x_{\eta,s} \right) \right) \right]^2 \tag{3.2}$$

With these 'softer' constraints (as in opposition with the 'hard' delta constraints), weight configurations that have different (non-zero) energies will give different contributions in the update of the factor to variable node messages. Furthermore one may notice that in the zero temperature limit (i.e. $\beta \to \infty$), if working in the realizable rule setting where the teacher is reproducible, the Boltzmann distribution concentrates around its minima allowing us to recover the original delta factors.

The following step is to re-parametrize the messages in terms of new quantities called log-likelihoods:

$$\begin{aligned} h_{i\to a}^{(t)}(w_i^{in/out}) &:= \frac{1}{\beta} \log\Big[\mu_{i\to a}^{(t)}(w_i^{in/out}) \Big] \\ u_{a\to i}^{(t)}(w_i^{in/out}) &:= \frac{1}{\beta} \log\Big[\hat{\mu}_{a\to i}^{(t)}(w_i^{in/out}) \Big] \end{aligned} \tag{3.3}$$

Using (3.3) and the sum-product BP equations (A.5) and (A.6), one obtains the new message passing update rules for the log-likelihoods:

1. Variable to factor node log-likelihoods

$$h_{i\to a}^{(t+1)}(w_i^{in/out}) \;=\; \sum_{b\in\partial i\setminus a} u_{b\to i}^{(t)}(w_i^{in/out}) \;+\; C_{i\to a}^{(t)} \tag{3.4}$$

2. Factor to variable node log-likelihoods

$$u_{a\to i}^{(t)}(w_i^{in/out}) \;= \\ = \frac{1}{\beta}\log\left[\sum_{(\mathbf{w}^{in};\mathbf{w}^{out})_{\partial a\setminus i}} e^{-\beta\mathcal{H}_a((\mathbf{w}^{in};\mathbf{w}^{out})_{\partial a\setminus i};w_i^{in/out})}\prod_{l\in\partial a\setminus i} e^{\beta h_{l\to a}^{(t)}(w_l^{in/out})}\right] + \hat{C}_{a\to i}^{(t)} \tag{3.5}$$

3. The 'log-beliefs': in the same way in which with the BP messages one can compute the beliefs $b_i(w_i)$ for each variable, using the log-likelihoods one can compute the following

$$\begin{aligned} h_i^{(t+1)}(w_i^{in/out}) \;&=\; \frac{1}{\beta}\log\left[b_i^{(t+1)}(w_i^{in/out})\right] \;= \\ &= \sum_{a\in\partial i} u_{a\to i}^{(t)}(w_i^{in/out}) \;+\; C_i^{(t)} \end{aligned} \tag{3.6}$$

   Analogously to what is done with the BP beliefs, by maximizing these log-beliefs one obtains the estimates for the weights.

Notice that these last equations are defined up to an overall additive constant. In particular we chose the constants $C_{i\to a}^{(t)}$, $\hat{C}_{a\to i}^{(t)}$ and $C_i^{(t)}$ in such a way that $\max_{w_i^{in/out}}\left[h_{i\to a}^{(t+1)}(w_i^{in/out})\right] = 0$, $\max_{w_i^{in/out}}\left[u_{a\to i}^{(t)}(w_i^{in/out})\right] = 0$ and $\max_{w_i^{in/out}}\left[h_i^{(t+1)}(w_i^{in/out})\right] = 0$.

Now by taking the zero temperature limit $\beta\to\infty$ of eq. (3.5) one gets, by virtue of the saddle-point approximation

$$u_{a\to i}^{(t)}(w_i^{in/out}) \;= \\ = \max_{(\mathbf{w}^{in};\mathbf{w}^{out})_{\partial a\setminus i}}\left[-\mathcal{H}_a\left((\mathbf{w}^{in};\mathbf{w}^{out})_{\partial a\setminus i};\, w_i^{in/out}\right) + \sum_{l\in\partial a\setminus i} h_{l\to a}^{(t)}(w_i^{in/out})\right] + \hat{C}_{a\to i}^{(t)} \tag{3.7}$$

Eq. (3.4) and Eq. (3.7) are the message passing rules of the so called max-sum algorithm. Another example of max-sum algorithm can be found in [12].

3.3 Soft decimation: reinforced message-passing algorithms

Both the 'regular' belief propagation and the max-sum algorithm, when applied to models with non-acyclic factor graphs, lack the guarantees of convergence to a fixed point that they would have in the case of a tree-graphical model. Furthermore in the case of the max-sum algorithm if the Hamiltonian written in Eq. (3.2) has multiple minima, even in cases where the factor graph is acyclic, these guarantees fall apart. There exist however some so-called *soft decimation* or *reinforcement* techniques that heuristically solve these problems. Examples of solutions analogous to the ones described below can be found for instance in [12][13][14].

For the regular sum-product BP algorithm, this reinforcement consists in changing the variable to factor node message update rule described in Eq. (A.5) with the following two equations:

$$b_i^{(t+1)}(x_i) \cong \left[\prod_{a\in\partial i} \hat{\mu}_{a\to i}^{(t)}(x_i)\right] \left(b_i^{(t)}(x_i)\right)^{rt} \tag{3.8}$$

$$\mu_{i\to a}^{(t+1)}(x_i) \cong \frac{b_i^{(t+1)}(x_i)}{\hat{\mu}_{a\to i}^{(t)}(x_i)} \tag{3.9}$$

where again the symbol $\cong$ indicates equality up to normalization.

Therefore the beliefs computed at a certain step of the algorithm are influenced by the beliefs computed at the previous step. The magnitude of this influence increases with t and with the parameter $r > 0$.

For the max-sum algorithm the variable to factor node update rule becomes:

$$h_i^{(t+1)}(x_i) = \sum_{a\in\partial i} u_{a\to i}^{(t)}(x_i) + \tilde{C}_i^{(t)} + rt \cdot h_i^{(t)}(x_i) \tag{3.10}$$

$$h_{i\to a}^{(t+1)}(x_i) = h_i^{(t+1)}(x_i) + \tilde{C}_{i\to a}^{(t)} - u_{a\to i}^{(t)}(x_i) \tag{3.11}$$

Where $\tilde{C}_{i\to a}^{(t)}$ and $\tilde{C}_i^{(t)}$ are not the same additive constants of Eqs. (3.4) and (3.6) respectively, but are chosen with the same criterion of setting to zero the maximum over x_i of $h_{i\to a}^{(t+1)}(x_i)$ and $h_i^{(t+1)}(x_i)$.

Intuitively what happens is that since the reinforcement effect increases with t, as 'time' passes at each iteration of BP or the max-sum (MS) algorithm, the updated beliefs or log-beliefs respectively, will have a gradually smaller variation

with respect to the the previous step, effectively forcing the convergence of the algorithm. Conceptually this is similar to what is described in section 2.2, where in the context of mini-batch or online learning the previously gained information on the system is encoded in the introduction of some external fields; the difference here is that the values of the fields are updated at each step of BP/MS and their intensity increases with t. The parameter r can be interpreted as a measure of the velocity of the reinforcement: the larger its value the faster the intensity of the aforementioned external fields will increase, but it will also likely come along with a reduction in the accuracy of the final weight estimation. Therefore fine tuning r to the appropriate value to use in the simulations is heuristically based on a trade off between speed of convergence to a fixed point and the accuracy of this fixed point.

In the case of the regular BP the soft-decimation forces the estimates of the marginal distributions to peak around one specific value, whereas in the case of MS the effect of the reinforcement is to increase the gap between the values of $h_i(x_i)$ for the different values of x_i.

3.4 Experimental results

In this section some experimental results for the realizable rule setting are presented. As previously mentioned, the performances of the standard BP, the max-sum algorithm and the corresponding reinforced versions are compared.

Here are some details about the implementation:

- The utilised activation function for the non-linearities is: $\phi(x) = tanh(x - b)$ with $b = 0{,}1$
- The sizes of the layers are: $N_{in} = N_{hidden} = N_{out} = 200$
- The teacher network is generated in such a way that its weight matrices have two non-zero entries for each row and for each column.
- The mismatch is encoded in the fact that for the construction of the student network, either the input matrix or the output matrix is generated in such a way that it contains the teacher *topology matrix* (i.e. $A_{ij}^{(\cdot)*} = 1 \implies A_{ij}^{(\cdot)} = 1$), but has three non-zero entries per row and column, instead of two.
- The value of the reinforcement velocity parameter is $r = 0{,}01$
- In the different sub-figures of Fig. 3.1 and 3.2 the following quantities are plotted:
  - In sub-figure **(a)** there is the fraction of the teacher non-zero weights that are wrongly estimated by the student

– In sub-figure **(b)** there is the error made in learning the teacher topology which is computed in the following way:

$$\epsilon_{topo} = \sum_{i}^{N_{hidden}} \sum_{j}^{N_{in}} \frac{A_{ij}^{in\,*}(1-|J_{ij}^{in}|) + A_{ij}^{in}(1-A_{ij}^{in\,*})|J_{ij}^{in}|}{N_{student}} + \\ + \sum_{l}^{N_{out}} \sum_{k}^{N_{hidden}} \frac{A_{lk}^{out\,*}(1-|J_{lk}^{out}|) + A_{lk}^{out}(1-A_{lk}^{out\,*})|J_{lk}^{out}|}{N_{student}} \quad (3.12)$$

where $N_{student}$ is the total number of (possibly) non-zero elements in the student network.

– In sub-figure **(c)** there is the average generalization error computed on the validation set $\{(\mathbf{x}_\eta^{test};\, \mathbf{y}_\eta^{test})\}_{\eta=1,\dots,N_{test}}$ which is provided by the teacher after the training has been carried out. The generalization error is computed as:

$$\epsilon_{gen} = \frac{1}{N_{test}} \sum_{\eta=1}^{N_{test}} \frac{\left\| \mathbf{y}_\eta^{test} - \mathbf{y}_\eta^{student} \right\|_2^2}{N_{out}} \quad (3.13)$$

where $\{\mathbf{y}_\eta^{student}\}_{\eta=1,\dots,N_{test}}$ are the output vectors computed from $\{\mathbf{x}_\eta^{test}\}_{\eta=1,\dots,N_{test}}$ and the trained student network. In all the following experiments $N_{test} = 50$.

In Fig. 3.1 are shown the results from the experiments where the mismatch is on the input matrix. In this case both BP, the max-sum algorithm and their reinforced counterparts perform similarly, all allowing to reconstruct both the topology and the weight values of the teacher, yielding a zero generalization error with a small number of data points just like in the Bayes optimal case.

If instead the mismatch is on the output matrix (see Fig. 3.2) what we observe is different. In this case we notice that belief propagation, although perfectly recovering the values of the weights that are actually present in the teacher network, fails to learn its topology, i.e. the trained student network has an excess of non-zero entries. Furthermore one can also see that the reinforcement does not affect significantly the performance of the standard BP.

In the case of the max-sum algorithm, both the reinforced and not reinforced versions allow to perfectly learn the teacher topology (see Fig. 3.2b), whereas as far as the reconstruction of the teacher weights is concerned, although they both perform more poorly with respect to the BP algorithms, the reinforcement does have a positive effect (see Fig. 3.2a). The end result is that in this case the max-sum algorithm with the reinforcement term is the one that performs better in terms of generalization error (see Fig. 3.2c).

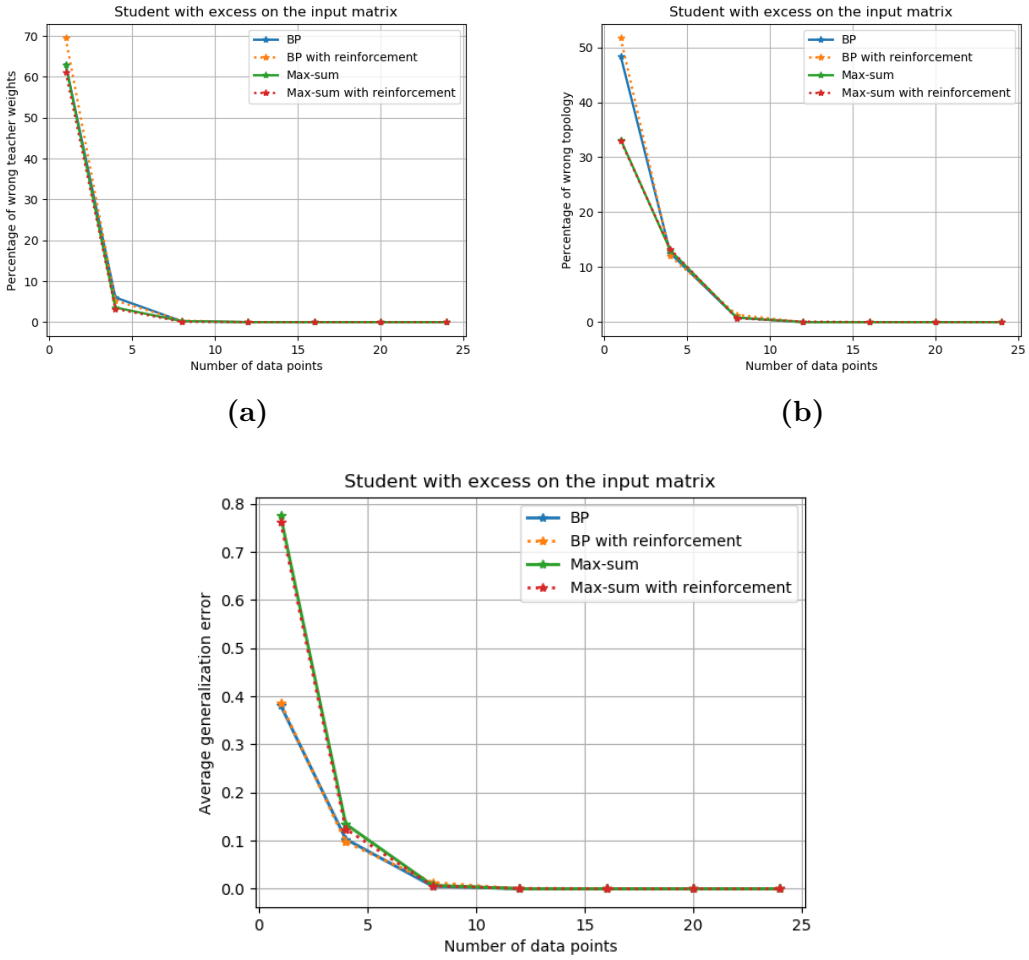
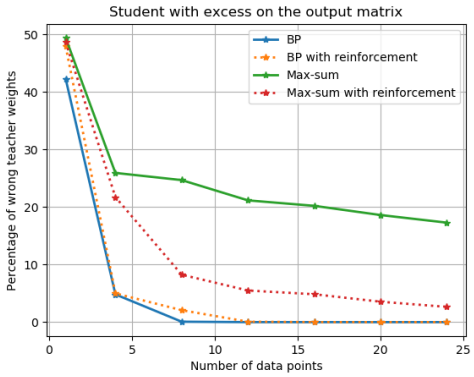
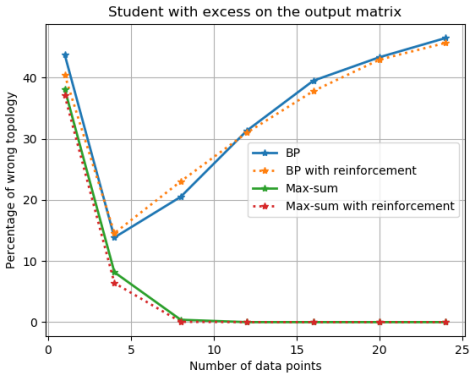


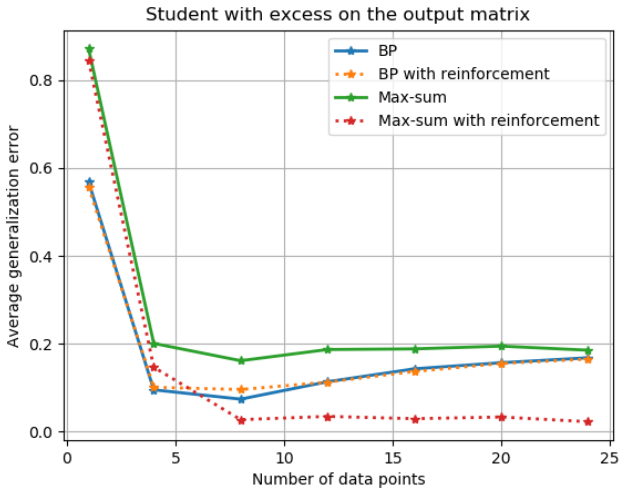
Figure 3.1: This figure refers to a setting where the student has the **input** matrix which is mismatched w.r.t. the teacher. Plots obtained by averaging over 10 experiments.



(a)



(b)



(c)

Figure 3.2: This figure refers to a setting where the student has the **output** matrix which is mismatched w.r.t. the teacher. Plots obtained by averaging over 10 experiments.

3.5 Conclusion and future directions

In this brief work different BP-based algorithms have been implemented to study the problem of training sparse neural networks from scratch.
In the teacher-student scenario we have obtained encouraging results, in which the teacher model is accurately reconstructed with a small number of data points, and these results do not seem to be majorly affected by changes in the non-linearities used in the network, in the size of the layers and by a different connectivity of the 'neurons'. In the realizable rule mismatched setting, however, there is still need for more extensive experimentation. Nonetheless the few results obtained until now seem to suggest that the max-sum algorithm with the reinforcement would be better suited for further simulations.

The following step would be that of going towards a setting with arbitrary mismatch and in this perspective, since the standard BP would probably not be very well suited for the training task because of reasons that we have mentioned earlier, the better performance of the reinforced max-sum algorithm is to some extent reassuring.

Another possible direction to explore is that of investigating what would happen in a multi-layer setup with SNN's with an arbitrary number of layers. An idea of implementation to tackle this problem can be seen in Appendix B.

Appendix A

A brief introduction to graphical models and belief propagation

Take the generic joint probability distribution of N random variables, taking values in a finite alphabet χ

$$P(\mathbf{x}) \; with \; \mathbf{x} = (x_1, \ldots, x_N), \; x_i \in \chi \tag{A.1}$$

Without any information on the mutual dependencies among the variables, the task of computing the marginals of the distribution is very costly ($\sim \mathcal{O}(|\chi|^N)$). However, if $P(\mathbf{x})$ factorizes in a non-trivial way, or so to speak the variables interact only 'locally', the complexity can be reduced significantly.

In particular let us suppose that the joint probability distribution can be written as the product of a certain number of factors, each depending only on a subset of the totality of the N variables. We call A and I the sets of indices, respectively identifying the different factors and variables.

To this factorization one can associate a graphical representation called *factor graph* (see Fig.A.1), which is the bipartite graph $G = (V = I \cup A;\, E \subset I \times A)$ where $(i, a) \in E$ if and only if the $a-th$ factor depends on x_i. We call ∂a the set of indices of the variables that appear in the $a-th$ factor, $\forall a \in A$, and $\partial i = \{a \in A : \, i \in \partial a\}$.

Then $P(\mathbf{x})$ reads

$$P(\mathbf{x}) \; = \; \frac{1}{\mathcal{Z}} \prod_{a \in A} \xi_a(\mathbf{x}_{\partial a}) \tag{A.2}$$

Where $\xi_a : \chi^{|\partial a|} \to \mathbb{R}_{\geq 0}$ with $a \in A$ are the factors, and $\mathbf{x}_{\partial a} = (x_{i_1}, \ldots, x_{i_{|\partial a|}})$ with $i_1, \ldots, i_{|\partial a|} \in \partial a$.

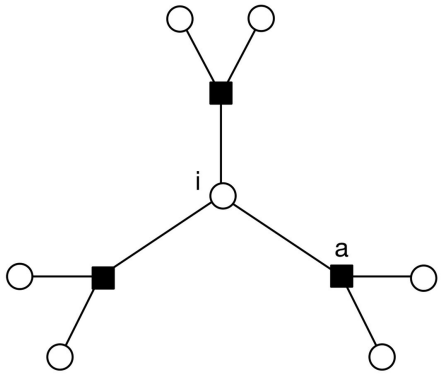


Figure A.1: Simple example of a factor graph

A.1 BP equations

For each edge (i, a) of the factor graph, one can define the following quantities, called messages

- $\hat{\mu}_{a\to i}(x_i)$: factor to variable node messages
- $\mu_{i\to a}(x_i)$: variable to factor node messages

and they take value in the space of probability distributions over the single variable space χ:

$$\sum_{x_i} \hat{\mu}_{a\to i}(x_i) = 1\,, \quad \sum_{x_i} \mu_{i\to a}(x_i) = 1 \qquad \forall\, i \in I,\, a \in A \tag{A.3}$$

$$\hat{\mu}_{a\to i}(x_i),\ \mu_{i\to a}(x_i) \geq 0 \qquad \forall\, i \in I,\, a \in A,\, x_i \in \chi \tag{A.4}$$

The belief propagation or message passing algorithm, consists in initializing all the messages in said probability space to some initial values $\hat{\mu}^{(0)}_{a\to i}(\cdot)$ and $\mu^{(0)}_{i\to a}(\cdot)$, and then updating them according to the following rules:

$$\mu^{(t+1)}_{i\to a}(x_i) \cong \prod_{b\in\partial i\setminus a} \hat{\mu}^{(t)}_{b\to i}(x_i) \tag{A.5}$$

$$\hat{\mu}^{(t)}_{a\to i}(x_i) \cong \sum_{\underline{x}_{\partial a\setminus i}} \xi_a(\underline{x}_{\partial a}) \prod_{j\in\partial a\setminus i} \mu^{(t)}_{j\to a}(x_j) \tag{A.6}$$

where the symbol $\cong$ stands for equality up to normalization.

The updates are carried on until convergence, i.e. when the maximum variation of the messages in a BP step is smaller than a certain desired precision value. With

the final values of the messages one can compute the so called beliefs, which are the BP estimates of the marginals of the original joint distribution

$$b_i(x_i) \cong \prod_{a \in \partial i} \hat{\mu}_{a \to i}(x_i) \tag{A.7}$$

In the special case in which the factor graph associated to the factorization of a probability distribution is a tree, belief propagation has some interesting properties:

- The algorithm always converges, and the fixed point is unique.
- The number of BP iterations to reach convergence grows linearly with the diameter of the factor graph.
- The beliefs computed at the fixed point, are the exact marginals.
- Other BP based algorithms allow also to carry out tasks such as sampling efficiently $P(\mathbf{x})$, computing its normalization constant (the partition function) and its modes.

However not always the factor graph associated to a problem is a tree, and in this case there are not such strong results on the properties of BP. Nonetheless, recently it has been shown that the algorithm can be surprisingly effective also on graphs with loops. The intuitive idea behind why this should be the case, is that BP is an algorithm that performs 'local' computations, and therefore should work as long as the graph is 'locally' a tree and the far apart variables can be considered approximately uncorrelated. The approximation where one assumes that the BP equations describe well a graphical model where the underlying factor graph is 'loopy', in the context of the study of spin systems, is called Bethe approximation. This can be seen as a more refined version of the so called *mean field* approximation, and in turn there have been studies on more sophisticated generalisations of BP (an example consists in the so called Cluster Variational Methods).
For more detailed and rigorous information on the subject refer to [15], [6] and [16].

Appendix B

Generalizing to a multi-layer setup

B.1 Hidden Variables

In this appendix we see a way to generalize the work carried out in this thesis (where the used architecture was always two-layer) to a multi-layer setup. Let us consider a neural network with L layers where we use the following notation:

- $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}$ are the weight matrices and the numeration increases starting from the layer closest to the output
- $\phi^{(1)}, \dots, \phi^{(L)}$ are the activation functions
- $n_0, \dots, n_L$ are the sizes of the layers, starting from the output layer

Then, the output vector reads

$$\mathbf{y} = \phi^{(1)}(\mathbf{W}^{(1)}\phi^{(2)}(\mathbf{W}^{(2)} \cdots \phi^{(L)}(\mathbf{W}^{(L)}\mathbf{x}))) \tag{B.1}$$

Analogously to the two layer setting, the aim is to compute the marginals of the posterior distribution of the weights given the data $\{(\mathbf{x}_\eta; \mathbf{y}_\eta)\}_{\eta=1,\dots,m}$

$$P\left(\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)} \,|\, \{\mathbf{y}_\eta\}_{\eta=1,\dots,m}, \{\mathbf{x}_\eta\}_{\eta=1,\dots,m}\right) \tag{B.2}$$

and once again the quantity of interest that allows to determine the structure of the underlying factor graph that represents the problem is the likelihood

$$P(\{\mathbf{y}_\eta\}_{\eta=1,\dots,m} \,|\, \mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}, \{\mathbf{x}_\eta\}_{\eta=1,\dots,m}) \tag{B.3}$$

In order to make the reasoning more easily generalizable for an arbitrary number of layers L, we introduce the auxiliary *hidden* variables $\mathbf{h}_\eta^{(l)} \in \mathbb{R}^{n_l}$ for $l = 1, \dots, L-1$

and $\eta = 1, \dots, m$. These variables represent the state of the neurons of the hidden layers and they have to be consistent with the state of the previous layers, in other words they have to satisfy the constraints:

$$\begin{aligned} \mathbf{h}_{\mu}^{(1)} &= \phi^{(2)}(\mathbf{W}^{(2)} \cdot \mathbf{h}_{\eta}^{(2)}) \\ &\vdots \\ \mathbf{h}_{\mu}^{(L-1)} &= \phi^{(L)}(\mathbf{W}^{(L)} \cdot \mathbf{x}_{\eta}) \end{aligned} \tag{B.4}$$

Successively one can write the following quantity that when marginalized with respect to the *hidden* variables returns the likelihood of Eq. (B.3)

$$P(\{\mathbf{y}_\eta;\, \mathbf{h}_\eta{}^{(1)};\, \mathbf{h}_\eta{}^{(L-1)}\}_{\eta=1,\dots,m} \,|\, \mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)},\ \{\mathbf{x}_\eta\}_{\eta=1,\dots,m}) =$$

$$= \prod_{a=1}^{n_0} \left[\prod_{\eta=1}^{m} \delta\left(y_{\eta,a};\, \phi^{(1)}(\mathbf{W}^{(1)} \cdot \mathbf{h}_{\eta}^{(1)})\right) \right] \prod_{l_1=1}^{n_1} \left[\prod_{\eta=1}^{m} \delta\left(h_{\eta,l_1}^{(1)};\, \phi^{(2)}(\mathbf{W}^{(2)} \cdot \mathbf{h}_{\eta}^{(2)})\right) \right] \cdots$$
$$\cdots\cdots \prod_{l_{L-1}=1}^{n_{L-1}} \left[\prod_{\eta=1}^{m} \delta\left(h_{\eta,l_{L-1}}^{(L-1)};\, \phi^{(L)}(\mathbf{W}^{(L)} \cdot \mathbf{x}_{\eta}),\right) \right] =$$

$$= \prod_{a=1}^{n_0} \xi_a(\mathbf{w}_{\partial a}^{(1)};\, \mathbf{h}_{\partial a}^{(1)}) \prod_{l_1=1}^{n_1} \xi_{l_1}(\mathbf{w}_{\partial l_1}^{(2)};\, \mathbf{h}_{\partial l_1}^{(1)};\, \mathbf{h}_{\partial l_1}^{(2)}) \cdots \prod_{l_{L-1}=1}^{n_{L-1}} \xi_{l_{L-1}}(\mathbf{w}_{\partial l_{L-1}}^{(L)};\, \mathbf{h}_{\partial l_{L-1}}^{(L-1)}) \tag{B.5}$$

From this expression one can build the factor graph representing the problem shown in Fig. B.1.

B.1.1 Problematic offline learning

Apart from the fact that this implementation is good for the sake of a better generalizability to a multi-layer setup, it also has the advantage that the number of neighbours of the factor nodes in the factor graph grows linearly with the number of non-zero elements in the rows of the weight matrices $\mathbf{W}^{(l)}$ with $l = 1, \dots, L$. Indeed if the size of these neighbourhoods grows too fast, the part of the BP algorithm where one updates the factor to variable node messages, becomes extremely costly in terms of computational time, as soon as one works with NN's that are

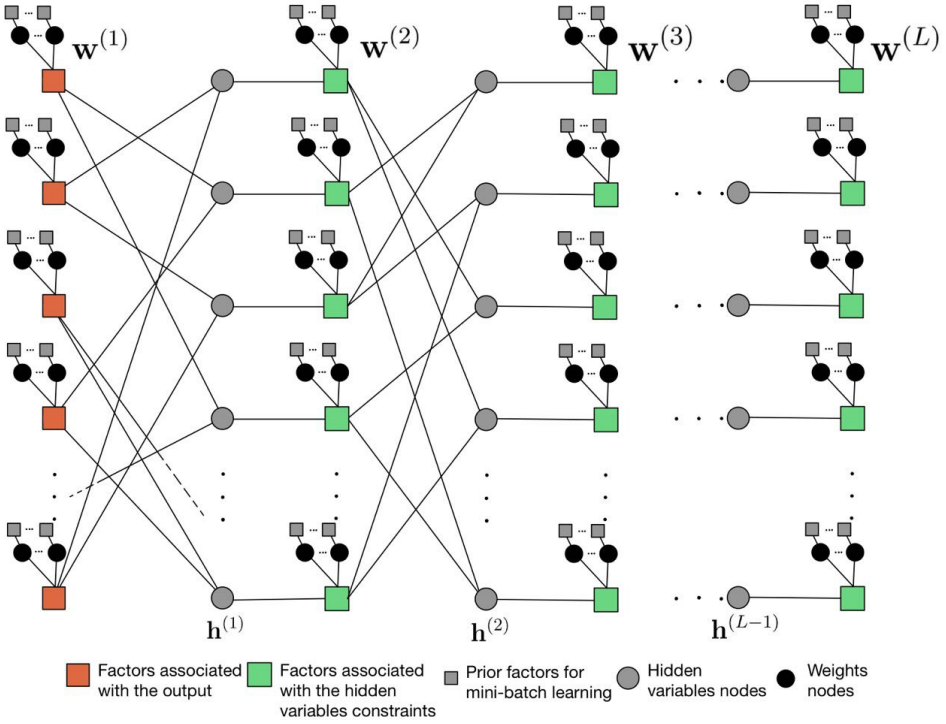


Figure B.1: Factor graph with the hidden variables nodes

slightly 'less sparse'.

The suspicious reader, may wonder at this point, the reason why this implementation with the hidden variables has not been used for the work carried out in this thesis in the two-layer case. The reason is that while apparently computationally more convenient, the BP algorithm on the factor graph of fig. B.1 has a significant problem: the variable nodes associated to the hidden variables, are not scalar quantities, whereas each grey circle represents a vector of size m, which is the number of data points. This represents an insurmountable computational impairment for the update of the factor to variable node messages in the offline learning case (see section 2.2).

A possible idea to deal with this problem would be to consider, in Eq. (B.5), the multiplications over the index η as the product of distinct factors. This however, despite solving the computational problem of having vector quantities as variables, generates another issue. Indeed the graphical representation of this factorization would be made of m distinct copies of the factor graph of Fig. B.1 and all the m copies of each one of the orange and green factors, would be connected to the same

variable nodes associated to the network weights. This causes the factor graph to be tremendously 'loopy', therefore making belief propagation a bad choice. This kind of 'excessive factorization' would cause the same problem to appear also in the implementation that has been actually used in this work. As in this case the described issue is more easily visualized, we show a pictorial representation in Fig. B.2.

Nonetheless the ideas presented in this appendix are not to be disregarded: indeed in chapter 2 we have seen that the BP algorithm implemented with mini-batches of very small size performs comparably to the offline case. Therefore one could use the factor graph described in this section, in order to implement an online algorithm to work in a multi-layer setup.

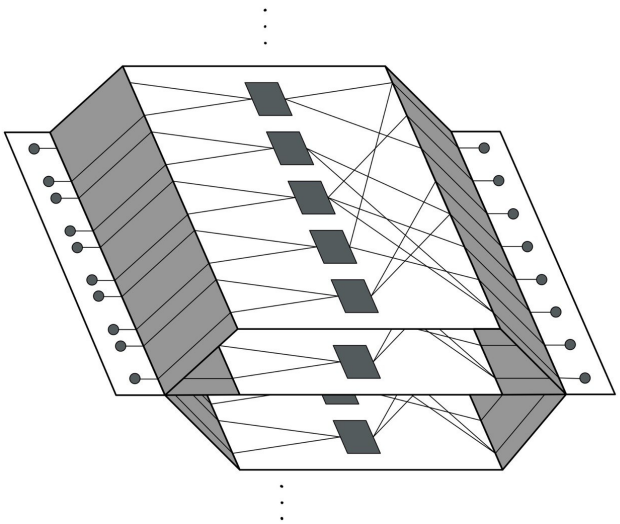


Figure B.2: Visual representation of the extremely 'loopy' factor graph coming from the factorization over η

Bibliography

[1] Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. *Exploring Sparsity in Recurrent Neural Networks*. 2017. arXiv: 1704.05119 [cs.LG] (cit. on p. 2).

[2] Song Han, Jeff Pool, John Tran, and William J. Dally. *Learning both Weights and Connections for Efficient Neural Networks*. 2015. arXiv: 1506.02626 [cs.NE] (cit. on p. 2).

[3] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. *Rethinking the Value of Network Pruning*. 2018. arXiv: 1810.05270 [cs.LG] (cit. on p. 2).

[4] Trevor Gale, Erich Elsen, and Sara Hooker. *The State of Sparsity in Deep Neural Networks*. 2019. arXiv: 1902.09574 [cs.LG] (cit. on p. 2).

[5] Utku Evci, Fabian Pedregosa, Aidan Gomez, and Erich Elsen. *The Difficulty of Training Sparse Neural Networks*. 2019. arXiv: 1906.10732 [cs.LG] (cit. on p. 2).

[6] Marc Mézard and Andrea Montanari. *Information, physics, and computation*. Oxford University Press, 2009 (cit. on pp. 2, 10, 27).

[7] Lenka Zdeborová and Florent Krzakala. «Statistical physics of inference: thresholds and algorithms». In: *Advances in Physics* 65.5 (Aug. 2016), pp. 453–552. ISSN: 1460-6976. DOI: 10.1080/00018732.2016.1211393. URL: http://dx.doi.org/10.1080/00018732.2016.1211393 (cit. on p. 2).

[8] Andrea Montanari and Rudiger Urbanke. *Modern Coding Theory: The Statistical Mechanics and Computer Science Point of View*. 2007. arXiv: 0704.2857 [cs.IT] (cit. on p. 2).

[9] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. *Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1*. 2016. arXiv: 1602.02830 [cs.LG] (cit. on p. 4).

[10] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. *Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations*. 2016. arXiv: 1609.07061 [cs.NE] (cit. on p. 4).

[11] Andre Manoel, Florent Krzakala, Eric W. Tramel, and Lenka Zdeborova. «Streaming Bayesian inference: Theoretical limits and mini-batch approximate message-passing». In: *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (Oct. 2017). DOI: 10.1109/allerton.2017.8262853. URL: http://dx.doi.org/10.1109/ALLERTON.2017.8262853 (cit. on p. 8).

[12] Carlo Baldassi and Alfredo Braunstein. «A Max-Sum algorithm for training discrete neural networks». In: *Journal of Statistical Mechanics: Theory and Experiment* 2015.8 (Aug. 2015), P08008. ISSN: 1742-5468. DOI: 10.1088/1742-5468/2015/08/p08008. URL: http://dx.doi.org/10.1088/1742-5468/2015/08/P08008 (cit. on pp. 18, 19).

[13] Alfredo Braunstein and Riccardo Zecchina. «Learning by Message Passing in Networks of Discrete Synapses». In: *Physical Review Letters* 96.3 (Jan. 2006). ISSN: 1079-7114. DOI: 10.1103/physrevlett.96.030201. URL: http://dx.doi.org/10.1103/PhysRevLett.96.030201 (cit. on p. 19).

[14] Tadaaki Hosaka and Yoshiyuki Kabashima. «Statistical mechanical approach to lossy data compression: Theory and practice». In: *Physica A: Statistical Mechanics and its Applications* 365.1 (June 2006), pp. 113–119. ISSN: 0378-4371. DOI: 10.1016/j.physa.2006.01.013. URL: http://dx.doi.org/10.1016/j.physa.2006.01.013 (cit. on p. 19).

[15] Jonathan Yedidia, William Freeman, and Yair Weiss. «Understanding belief propagation and its generalizations». In: vol. 8. Jan. 2003, pp. 239–269. ISBN: 1558608117 (cit. on p. 27).

[16] Alessandro Pelizzola. «Cluster variation method in statistical physics and probabilistic graphical models». In: *Journal of Physics A: Mathematical and General* 38.33 (Aug. 2005), R309–R339. ISSN: 1361-6447. DOI: 10.1088/0305-4470/38/33/r01. URL: http://dx.doi.org/10.1088/0305-4470/38/33/R01 (cit. on p. 27).

[17] Andre Manoel, Florent Krzakala, Marc Mezard, and Lenka Zdeborova. «Multi-layer generalized linear estimation». In: *2017 IEEE International Symposium on Information Theory (ISIT)* (June 2017). DOI: 10.1109/isit.2017.8006899. URL: http://dx.doi.org/10.1109/ISIT.2017.8006899.

[18] Marylou Gabrié. «Mean-field inference methods for neural networks». In: *Journal of Physics A: Mathematical and Theoretical* 53.22 (May 2020), p. 223002. ISSN: 1751-8121. DOI: 10.1088/1751-8121/ab7f65. URL: http://dx.doi.org/10.1088/1751-8121/ab7f65.

[19] K.Tamuly. *Compression and Acceleration of High-dimensional Neural Networks*. Nov. 2018. URL: https://software.intel.com/content/www/us/en/develop/articles/compression-and-acceleration-of-high-dimensional-neural-networks.html.

[20] F.Lagunas. *Is the future of Neural Networks Sparse? An Introduction (1/N)*. Feb. 2020. URL: https://medium.com/huggingface/is-the-future-of-neural-networks-sparse-an-introduction-1-n-d03923ecbd70.

[21] Mitchell A. Gordon. *Do We Really Need Model Compression?* Jan. 2020. URL: http://mitchgordon.me/machine/learning/2020/01/13/do-we-really-need-model-compression.html.