Simulink Implementation of Belief Propagation in Normal Factor Graphs

Amedeo Buonanno and Francesco A.N. Palmieri

Seconda Università di Napoli (SUN)
Dipartimento di Ingegneria Industriale e dell'Informazione,
via Roma 29, 81031 Aversa (CE) - Italy
{amedeo.buonanno,francesco.palmieri}@unina2.it

Abstract. A Simulink Library for rapid prototyping of belief network architectures using Forney-style Factor Graph is presented. Our approach allows to draw complex architectures in a fairly easy way giving to the user the high flexibility of Matlab-Simulink environment. In this framework the user can perform rapid prototyping because belief propagation is carried in a bi-directional data flow in the Simulink architecture. Results on learning a latent model for artificial characters recognition are presented.

Keywords: Belief Propagation, Factor Graph, Pattern Recognition, Machine Learning.

1 Introduction

Graphical models are a "marriage between probability theory and graph theory" [1] as they compactly encode complex distributions over a high-dimensional space. When a problem can be formulated in the form of a graph, it is very appealing to study the variables involved as part of an interconnected system where the reached equilibrium point is the solution. The similarities with the working of the nervous system makes this paradigm even more fascinating [2]. Bayesian inference on graphs, pioneered by Pearl [3], has become a very popular paradigm for approaching many problems in different fields such as communication, signal processing and artificial intelligence [4]. The Factor Graph is a particular type of Graphical model and represents an interesting way to model the interaction between stochastic variables. Following the formulation of Forney-style Factor Graphs (FFG) [5] (or normal graphs), Bayesian graphs can be drawn as block diagrams and probability distribution easily transformed and propagated. In this paper we report the results of our work in which we have designed and implemented a Simulink Library for quick prototyping of several network architectures using the FFG paradigm.

In Section 2 we briefly review the Factor Graph paradigm introducing the building blocks of our proposed Simulink Library. In Section 3 the two operating modes are introduced. In Section 4 we present the application of this tool to an artificial character recognition task.

S. Bassis et al. (eds.), *Recent Advances of Neural Networks Models and Applications*,
Smart Innovation, Systems and Technologies 37, DOI: 10.1007/978-3-319-18164-6_2

2 Simulink Factor Graph Library

Factor Graphs model the interaction among stochastic variables. In the FFG approach there are blocks, variables and directed edges [5]. Even if edges have a defined direction, probability flows in both directions (foward and backward) [4]. To associate to each stochastic variable two messages, we have used the built-in Two-Way Connection block that in Simulink allows bidirectional signal flow. In our Simulink implementation all the architectures can be built with just three main functional blocks: Variable, Factor and Diverter (Figure 1) that will be described in the folllowing. In our notation, we avoid the upper arrows [4] and use explicit letters: b for *backward* and f for *forward*.

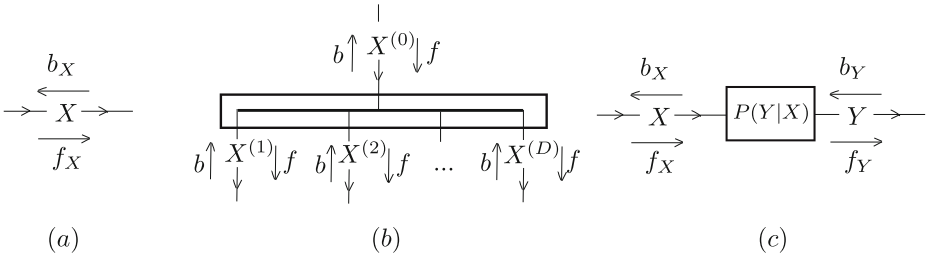

Fig. 1. Functional Blocks: (a) Variable, (b) Diverter, (c) Factor

2.1 Variable

For a *variable* X (Figure 1(a)) that takes values in the discrete alphabet $\mathcal{X} = \{x^1, x^2, ..., x^{M_X}\}$, forward and backward messages are in function form:

$$b_X(x^i), f_X(x^i), \quad i = 1 : M_X$$

and in vector form

$$\begin{aligned} \mathbf{b}_X &= (b_X(x^1), b_X(x^2), ..., b_X(x^{M_X}))^T \\ \mathbf{f}_X &= (f_X(x^1), f_X(x^2), ..., f_X(x^{M_X}))^T \end{aligned}$$

All messages are proportional ($\propto$) to discrete distributions and may be normalized to sum to one. Comprehensive knowledge about X is contained in the distribution p_X obtained through the product rule (in function form):

$$p_X(x^i) \propto f_X(x^i) b_X(x^i), \quad i = 1 : M_X$$

or $\mathbf{p}_X \propto \mathbf{f}_X \odot \mathbf{b}_X$, in vector form, where $\odot$ denotes the element-by-element product.

Each message b, f or p in the data flow is an $nT \times M$ matrix with nT the number of realizations and M the variable cardinality. Two-way connection blocks

allow the construction of a bi-directional data flow. The implementation for an Internal Variable block is shown in Figure 2 where the forward message on the port up (f_b_up) is transmitted on the port down (f_b_down) and conversely the backward message on the port down is transmitted on the port up. All distribution flow can be saved to workspace.

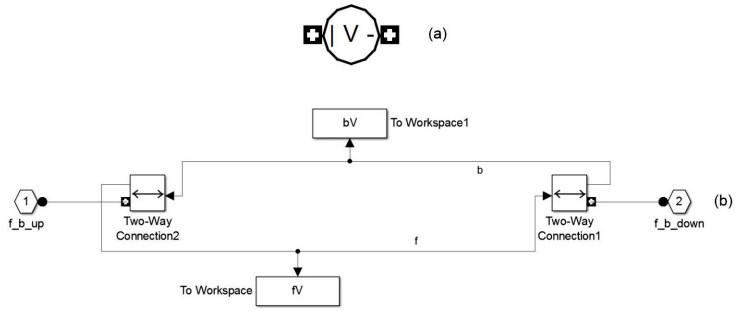

Fig. 2. The implementation of the Internal Variable block. The icon in the library (a) and its detailed scheme (b)

Similarly Figure 3 shows the detailed schemes of Source and Sink Variable blocks.

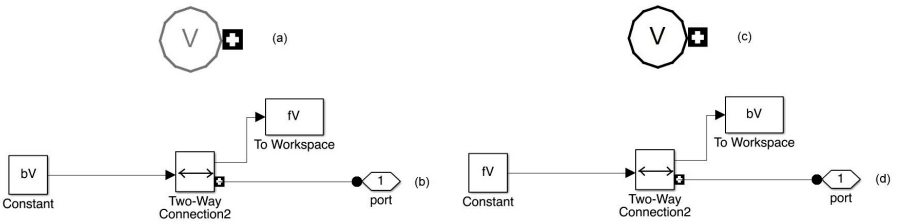

Fig. 3. The implementation of the Source Variable block and of the Sink Variable block. The icon in the library (a,c) and its detailed scheme (b,d) respectively for the Source and for the Sink

2.2 Diverter Block

The *diverter* block (Figure 1(b)) in the Bayesian model represents the equality constraint with the variable X replicated $D + 1$ times. Messages for incoming and outgoing branches carry different forward and backward information.

Messages that leave the block are obtained as the product of the incoming ones (in function form):

$$b_{X^{(0)}}(x^i) \propto \prod_{j=1}^{D} b_{X^{(j)}}(x^i)$$

$$f_{X^{(m)}} \quad \propto f_{X^{(0)}}(x^i) \prod_{j=1, j\neq m}^{D} b_{X^{(j)}}(x^i), \quad m = 1:D, i = 1:M_X$$

In vector form:

$$\mathbf{b}_{X^{(0)}} \propto \odot_{j=1}^{D} \mathbf{b}_{X^{(j)}},$$
$$\mathbf{f}_{X^{(m)}} \propto \mathbf{f}_{X^{(0)}} \odot_{j=1, j\neq m}^{D} \mathbf{b}_{X^{(j)}}, \quad m = 1:D$$

Figure 4 shows the detailed scheme of our implementation of the Diverter Block. Each port is connected to a variable in the network. After element-wise product among variables each variable is returned after normalization to one (each message is normalized to be a valid distribution).

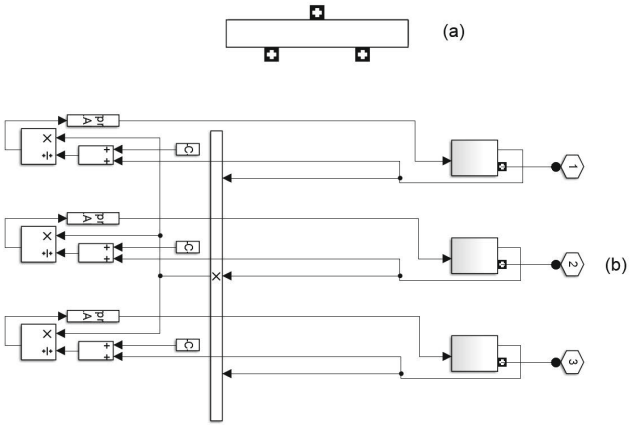

Fig. 4. Simulink implementation of a Diverter Block with three ports. The icon in the library (a) and its detailed scheme (b)

2.3 Factor Block

The *factor* block (Figure 1(c)) is the main block that represents the conditional probability matrix of Y given X. More specifically if X takes values in the discrete alphabet $\mathcal{X} = \{x^1, x^2, ..., x^{M_X}\}$ and Y in $\mathcal{Y} = \{y^1, y^2, ..., y^{M_Y}\}$, $P(Y|X)$ is the $M_X \times M_Y$ row-stochastic matrix:

$$P(Y|X) = [Pr\{Y = y^j | X = x^i\}]_{i=1:M_X}^{j=1:M_Y} = [\theta_{ij}]_{i=1:M_X}^{j=1:M_Y} = \theta$$

Outgoing messages are (in function form):

$$f_Y(y^j) \propto \sum_{i=1}^{M_X} \theta_{ij} f_X(x^i), \quad b_X(x^i) \propto \sum_{j=1}^{M_Y} \theta_{ij} b_Y(y^j)$$

In vector form:

$$\mathbf{f}_Y \propto P(Y|X)^T \mathbf{f}_X, \quad \mathbf{b}_X \propto P(Y|X)\mathbf{b}_Y$$

The above rules are rigorous translation of Bayes' theorem and marginalization (a complete review and proofs can be found in classical papers [4], [6]).

Figure 5 shows our implemention of the Factor Block with a Level2-MATLAB S-Function that wraps the Maximum Likelihood (ML) algorithm described in [7]. The system learns locally using nT realizations of the forward message of variable X, the nT realizations of backward message of variable Y and an initial value of matrix P. During learning, a new value of P is produced on each epoch and nT realizations of backward message for variable X and forward message for Y are sent to the adjacent blocks.

If the number of iteration is set to 0, the Block simply computes the nT realizations of backward of variable X and the nT realizations of forward message of variable Y (using the results in [8]).

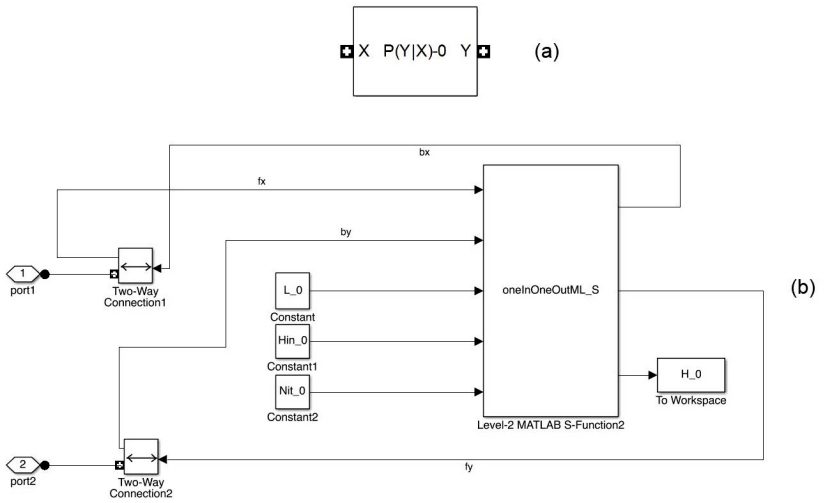

Fig. 5. Simulink implementation of the Factor Block. The icon in the library (a) and its detailed scheme (b) - During learning phase, given the initial value of Conditional Probability Matrix (Hin), the bacward messages for variable Y, the forward messages for variable X and the learning mask (L), a new value of H is computed applying Nit iterations of ML algorithm. If the Nit is set to 0, the block works in inference mode.

Using the implemented library, simply by dragging and connecting, the user can define a wide range of architectures that otherwise would have required the

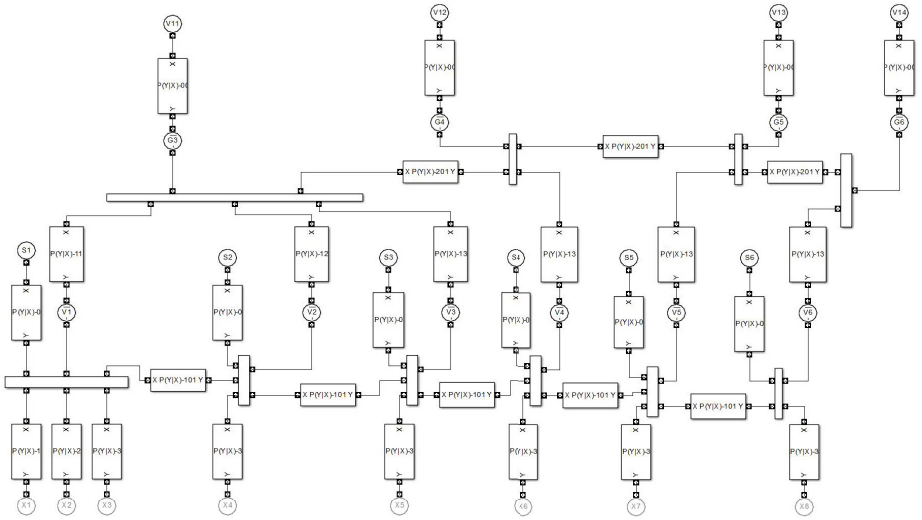

Fig. 6. A complex architecture designed using the proposed library

writing of a custom algorithm of belief propagation. Figure 6 shows a complex network drawn using the building blocks previously introduced.

3 Flow Control

During the simulation, each block uses messages coming from connected blocks and evolves producing new messages. The distributions exchanged among blocks are bi-directional and simultaneous, but the network flow is controlled from the top by a MATLAB script that sets parameters, triggers execution and collects results. The network can work in *Inference Mode*, when the block parameters are fixed, and in *Learning Mode*, when the block parameters are learned. In the Learning Phase (Figure 7(a)), based on epochs, after the Network Initialization (set to uniform all the variables, set the dimension of the messages), the model simulation is started defining purposely the Simulation Time and Model Parameters (values of Factors). At the end of simulation the new Model Parameters are used as initialization values for next epoch. This is done until the Maximum Number of Epochs is reached. In the Evolution Phase (Figure 7(b)), in the Parameter Initialization, the user has to adopt the correct values of parameters learned during Learning Phase.

The Model Simulation step is performed in the Simulink environment that has to be purposely configured using Fixed-Step Solver Type and with a Fixed Size Time Step. During the updating phase of simulation, Simulink determines the order in which the block methods must be triggered. The user cannot explicitly change this order, but he can assign priorities to non virtual blocks to indicate to Simulink their execution order relative to other blocks. Simulink tries to honor

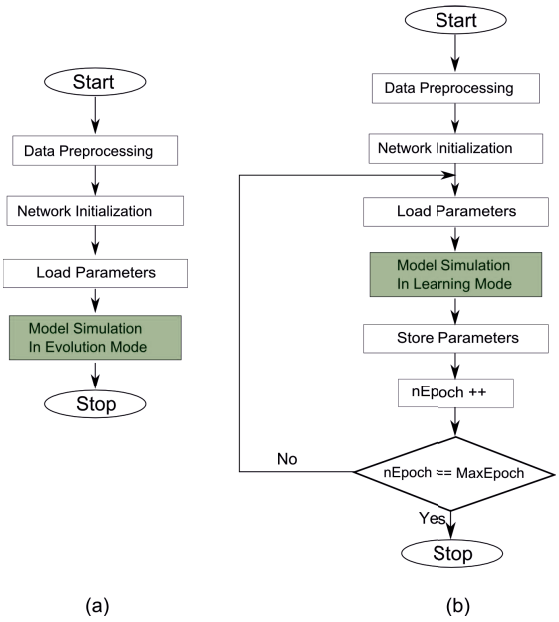

Fig. 7. Scheme for model simulation in the Inference mode (a) and in the Learning mode (b)

block priority settings, unless there is a conflict with data dependencies [9]. We have verified that Simulink automatically assigns the correct execution order, evaluating the From Workspace block (in the source blocks) and then the other blocks. To avoid wrongly assigned variables, each variable in each block is initialized with an uniform distribution. Each block automatically determines the dimension of the variable to which it is connected. During the simulation, each block uses the inputs coming from other blocks and evolves producing output to connecting blocks using the rules outlined in [8].

4 Characters Recognition Example

We have used the proposed Library in several applications. In this work we present the result obtained with a simple Latent Model applied to a recognition task on the Artificial Characters Dataset [10]. This dataset is formed by thousands of 12x8 black and white images representing the characters {'A', 'C', 'D', 'E', 'F', 'G', 'H', 'L', 'P', 'R'}. The network we have implemented is composed of 96 factors (a factor for each pixel) and only one hidden variable.

An image is a matrix of pixels, where each pixel can be considered as a stochastic variable that can assume value in a finite alphabet (2 symbols for black and white images). We have a set of random variables $\{X_1, X_2, ..., X_n\}$ that belong to

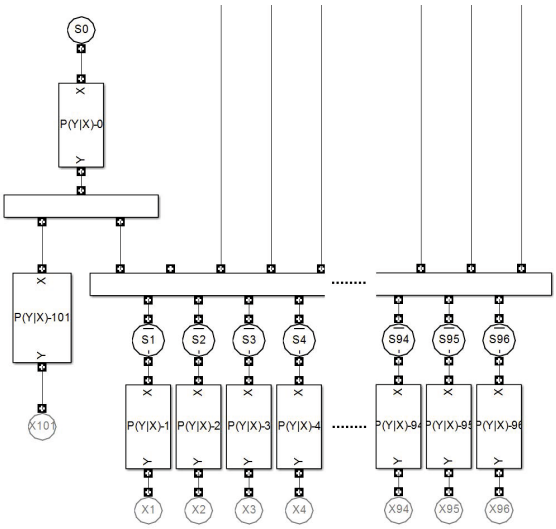

Fig. 8. The designed network for Artificial Characters recognition task using the implemented Library

a same finite alphabet $\mathcal{X}$. This set of variables is fully characterized by its joint probability mass function $p(X_1, X_2, ..., X_n)$. All the mutual interactions among the variables is contained in the structure of p. A variable can be: 1) known (instantiated): the backward message is the delta distribution; 2) completely unknown (erased): the backward message is a uniform distribution; 3) known softly: the backward message is a density. In all cases after message propagation the system responds with a forward message that is related to information stored in the system during the learning phase [11]. We use a simple Latent Model where each variable X_i (pixel) is connected to a Latent Variable (Figure 8) and there is also a Variable that contains the information of the presented character (X_{101}). In the Learning Phase the instantiated variables of training examples are injected in the network and using the ML algorithm in [7] the matrices $P(Y|X) - i$ are learned.

4.1 A Simulation

Using the Artificial Characters Dataset [10] we have trained our network with 800 training images of 12x8 black and white images representing the characters: {'A', 'C', 'D', 'E', 'F', 'G', 'H', 'L', 'P', 'R'} (Figure 9). The dimension of the embedding space is set to 150. The number of epochs for learning phase is set to 20 and each epoch is formed by 10 evolution steps.

To store all configurations the embedding space should have been set to 2^{96}, but the real configurations are much less. We limited the embedding space to 150 because computational issues. Even if we have used a small dimension of the embedding space, the system stores relevant structures of the presented images

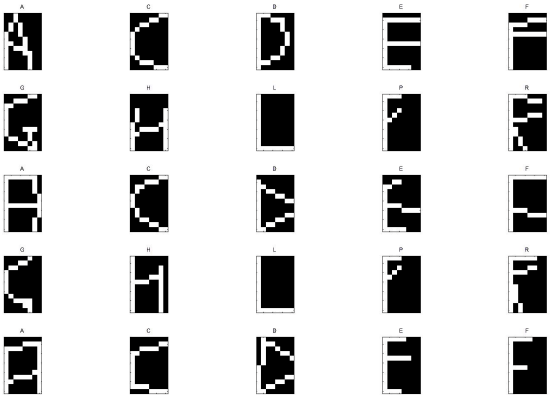

Fig. 9. 25 samples from the Training Set

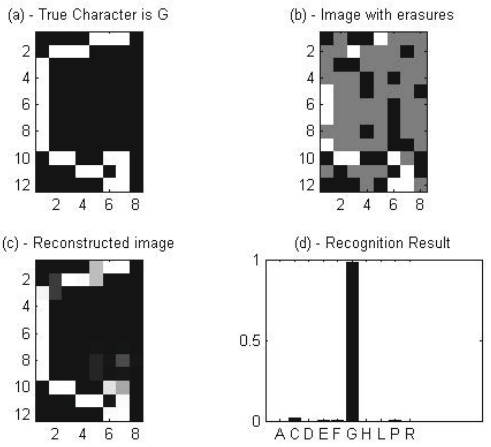

Fig. 10. Network answer - An image is retrieved from the Test Set (a), a big percentage of pixels are erased (gray pixels in (b)) and this information is injected in the network as backward messages. The network, after evolution, returns the Reconstructed image (c) and a probability distribution on the character set (d))

and presenting 800 test images, the system recognize the characters presented with an accuracy of 76%.

In Figure 10 the results of the recognition and completion task are presented. An image is retrieved from Test Set (Figure 10 (a)), a big percentage of pixels are erased (gray pixels in (Figure 10 (b))) and this information is injected in the network as backward messages of Source variables. The information about the presented character is set to uniform. The network, after the evolution (Inference Mode) returns the forward messages of Source variables that, combined

with the provided backward messages, give us the Reconstructed image (Figure 10 (c)). The network provides also the probability distribution on whole vocabulary (Figure 10 (d))

5 Conclusion

We have implemented a Library of Simulink blocks that permits to rapidly design a wide range of architectures using the Factor Graph paradigm. This approach allows to experiment on different architectures using Simulink bi-directional connections as probability pipelines. Current efforts are devoted to use this paradigm for various applications and to find more efficient implementations when the architectures grow in size and complexity.

References

1. Jordan, M. (ed.): Learning in Graphical Models. MIT Press (1998)
2. Hawkins, J.: On Intelligence (with Sandra Blakeslee). Times Books (2004)
3. Pearl, J.: Probabilistic reasoning in intelligent systems - networks of plausible inference. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann (1989)
4. Loeliger, H.A.: An introduction to factor graphs. IEEE Signal Processing Magazine 21(1), 28–41 (2004)
5. Forney, G.D.: Codes on graphs: Normal realizations. IEEE Transactions on Information Theory 47(2), 520–548 (2001)
6. Kschischang, F., Member, S., Frey, B.J., Loeliger, H.-A.: Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory 47, 498–519 (2001)
7. Palmieri, F.A.N.: A Comparison of Algorithms for Learning Hidden Variables in Normal Graphs. ArXiv e-prints (2013)
8. Palmieri, F.: Notes on factor graphs. In: Apolloni, B., Bassis, S., Marinaro, M. (eds.) WIRN. Frontiers in Artificial Intelligence and Applications, vol. 193, pp. 154–162. IOS Press (2008)
9. MATLAB Documentation Center - R2014A, ch. Control and Display the Sorted Order
10. Guvenir, H.A., Acar, B., Muderrisoglu, H.: Artificial characters data set. In: Bache, K., Lichman, M. (eds.) UCI Machine Learning Repository (2013), `https://archive.ics.uci.edu/ml/datasets/Artificial+Characters`
11. Palmieri, F., Ciuonzo, D., Mattera, D., Romano, G., Rossi, P.S.: From examples to bayesian inference. In: Apolloni, B., Bassis, S., Esposito, A., Morabito, F.C. (eds.) WIRN. Frontiers in Artificial Intelligence and Applications, vol. 234, pp. 97–104. IOS Press (2011)