Reverse Engineering the Hamming Code with Automatic Graph Learning

Dr. Noah B. Jacobsen
Director of Data Science
Mission Solutions Group, Inc.
North Charleston, SC, USA
njacobsen@msg.us.com

***Abstract*—Probabilistic graphical models are used extensively across the information theory, artificial intelligence and machine learning disciplines. In this paper, we work towards realizing a generalized graph-based framework for automated learning in intelligent systems. The proposed *automatic graph learning* framework employs factor graphs, i.e. Tanner graphs from coding theory, to represent an arbitrary stochastic system of variables and factorized realizations of their joint probability density function. We develop algorithms that are capable of learning statistical relationships between system variables, which involves constructing an appropriate factor graph representation and generating estimates of its component probability density functions, from training data. In this paper, automatic graph learning is used to reverse engineer the Hamming code, based on training data comprised of input-output codeword pairs. We show that automatic graph learning is capable of replicating known decoder performance with an order of magnitude less training data than a multi-layer dense neural network.**

***Index Terms*—learning theory, probabilistic graph models, factor graphs, Bayesian and neural networks, probabilistic inference, sum-product algorithm**

I. Introduction

Probabilistic graphical models play a fundamental role in Artificial Intelligence (AI) research and systems engineering, for system modeling, knowledge representation and inference processing [1]–[4]. In this work, we build upon the framework of factor graphs, from probability and information theory, for modeling stochastic systems and implementing probabilistic inference algorithms. We approach the problem of supervised learning, *i.e.* training based learning, from first principles of probability theory. We assume that a factor graph model for the system in question is *a priori* unknown, and that training data is used to deduce an appropriate factor graph model and functional approximation of its component factors. The developed methodology is general in the sense that it can be applied to an arbitrary data system, wherein a suitable factor graph model is automatically learned from training data and inference computations are performed on the learned graph. Given a factor graph system model and characterization of is component factors, an instance of the sum-product algorithm is used to compute marginal densities, such as the *a posteriori* probabilities, on hidden variables.

Although graphical structure is a basic characteristic of neural networks (NNs), there is a disconnect between the NN model and probability law governing its variables (or features). Specific instances of neural networks such as convolutional neural networks [5] and recurrent neural networks [6] have demonstrated a high degree of success in several areas including object recognition and natural language processing. However, there is no generalized NN or NN-based framework that could be applied to an arbitrary learning system.

In this paper, we develop a generalized learning framework that captures arbitrary statistical structure between a system of variables. Our approach, termed Automatic Graph Learning (AGL), uses empirical estimates of low-order statistical moments to (i) select dependent subsets of variables to construct a system factor graph model and (ii) generate estimates of the functional form of the resulting factors via Fourier domain synthesis. Given the factor graph model and functional characterization of its component factors, the sum-product algorithm is then used to implement probabilistic inference computations.

The majority of research on automatic model learning focuses on selecting the best model from a predefined set of candidate models (*e.g.* AutoWEKA [7] or AutoML [8]). In contrast, our work builds a unified framework in which an appropriate model for representing and processing inference for an arbitrary system of data is automatically learned from training data.

Several authors have focused on the application of message passing algorithms, such as the sum-product algorithm, to neural networks. For example, [9] develops a framework for message passing neural networks (MPNNs) for predicting chemical properties of molecules, however they do not use the factor graph formalism nor attempt to address arbitrary data systems. In [10], the authors embed the neural action in the factor nodes of the factor graph. Factor nodes aggregate population activities of pools of neurons and then apply belief propagation (sum-product algorithm) to develop inference across the graph.

In [11], the authors develop variational algorithms for probabilistic graphical models which encompass a number of well-known inference algorithms while focusing on exponential families and their conjugate dual property. In [12], belief propagation algorithms are developed under the approximating assumption that all factors have the form of a rank-1 tensor. In our work, we impose no constraint on the family of densities used to model the posterior density, factor densities or other

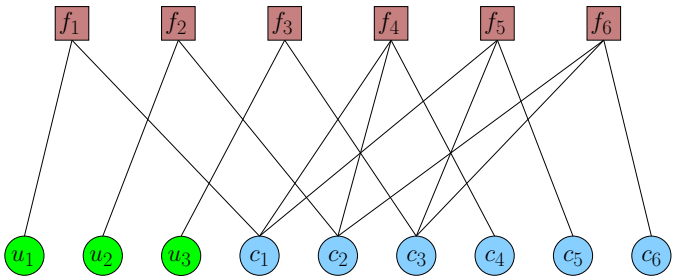

Fig. 1. Factor graph representing the joint density function of the $(6, 3)$ Hamming code. The input bits u_i, $i = 1, 2, 3$ are depicted as green circles and the output bits c_i, $i = 1, 2, \ldots, 6$ are light blue circles. Factors f_1, f_2, and f_3 represent equality constraints on the systematic code bits and factors f_4, f_5, and f_6 correspond to the parity check equations.

densities that arise in belief propagation.

Fourier series are truncated in [13] to develop approximations to multivariate probability density functions as well as variable marginalization in Fourier domain. In our work, instead of truncating the Fourier series of factor density functions, we use the relationship of Fourier coefficients to statistical moments to form estimates of the component factors based on truncated orders of moments. We further leverage direct Monte Carlo estimation of the joint density function in the Fourier domain. Our approach better relates to the probability law governing the system of variables and provides metrics to test for variable dependence.

Markov Chain Monte Carlo (MCMC) methods for Neural Networks and hybrid variants, *e.g.* [14], also represent a broad class of algorithms, and use sampling from the posterior density given training data to estimate expected values of system variables. Although we also use training data to estimate statistical moments via Monte Carlo integration, the methods developed in this work are more general because we do not assume that the output is in the form of statistical expectation, rather we estimate the density functions directly in the Fourier domain and use the relationship of the Fourier coefficients to statistical moments. To our knowledge, we are the main proponents of the statistical moments based approach for constructing probabilistic graphical models.

II. Factor Graphs and the Sum-Product Algorithm

In this section, we illustrate the application of factor graphs and the sum-product algorithm using a binary linear code, specifically the $(6, 3)$ Hamming code, and then later we apply the AGL framework to reverse engineer a decoder for the $(6, 3)$ Hamming code based on training data of input-output codeword pairs. In the following, we sometimes use an abbreviated notation for the joint *p.d.f.* for compactness, where $f_{XY}(x, y)$ is written as $f(X, Y)$.

The modeling and inference framework used in this work is based on factor graphs and application of the sum-product algorithm for estimating the state of hidden variables given the state of observed variables (see *e.g.* [2], [3]). Factor graphs provide a basic framework for representing an arbitrary physical system as the product of several lower-complexity component densities. Given a specification of the component

Generator Matrix:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Input: $\mathbf{u} = [u_1, u_2, u_3]$

Output: $\mathbf{c} = \mathbf{u}G = [c_1, c_2, c_3, c_4, c_5, c_6]$

Duality Condition: $GH^T = 0$

Dual (Parity-Check) Matrix:

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Valid codeword if and only if:

$$\mathbf{c}H^T = \mathbf{u}GH^T = 0.$$

Fig. 2. Generator and parity check matrices for the $(6, 3)$ Hamming code. Any valid codeword must satisfy the parity check equations defined by the dual matrix.

factors, the sum-product algorithm is implemented directly from the graph representation to compute marginal densities on hidden variables. The widely applicable sum-product algorithm is important due partly to its success in the development of high-performance Error Correcting Codes (ECCs) [4], [15]–[17].

A *factor graph* is a bipartite graph, in which (a) there are two-types of nodes and (b) edges of the graph connect only to nodes of different type (sometimes called a Tanner graph in coding theory). The *variable nodes* represent the variables of the system and the *factor nodes* represent the component factors (local statistical dependence) of the global system *p.d.f.* Edges of the factor graph connect factor nodes to variable nodes if and only if the corresponding factor is a function of the corresponding variable. Figure 1 depicts a factor graph relating the input and output bits of a $(6, 3)$ Hamming code.

We next describe the factor graph representation of the $(6, 3)$ Hamming code, which is a binary linear code with maximal codeword distance for its code rate. The notation $(6, 3)$ refers to the codeword length (6-bits) and number of input bits (3-bits), which results in a code rate of $1/2$ (ratio of input bits to code bits). A systematic binary linear code includes the input bits as part of the output codeword.

Let $U = (U_1, \ldots, U_k)$ represent uncoded information bits and $C = (C_1, \ldots, C_n)$ represent codewords of the $(n = 6, k = 3)$ Hamming code, where $C_1 = U_1$, $C_2 = U_2$ and, $C_3 = U_3$ are the systematic code bits and $C_4 = C_1 + C_2$, $C_5 = C_1 + C_3$, and $C_6 = C_2 + C_3$ are the parity bits (modulo-two arithmetic). The codewords of a binary linear code are given by matrix multiplication with the generator matrix, G, and are orthogonal to the parity check matrix, H, defined in Figure 2 for the $(6, 3)$ Hamming code. The rows of H span the null space of G and correspond to factor nodes of the factor graph representation of the code, shown in Figure 1.

Let $V = (U, C)$ denote the system variables. For the $(6, 3)$

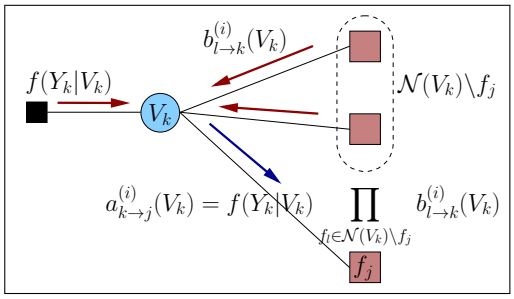

Fig. 3. Variable node update of belief propagation.

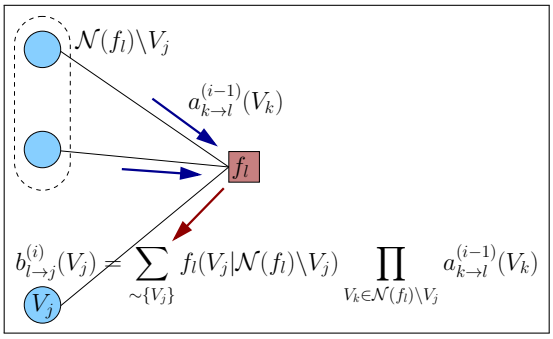

Fig. 4. Factor node update of belief propagation.

Hamming code, we can write a joint *p.d.f.* for the system of variables as [2]:

$$f(V) = f_1(U_1, C_1)f_2(U_2, C_2)f_3(U_3, C_3)\times \quad (1)$$
$$f_4(C_1, C_2, C_4)f_5(C_1, C_3, C_5)f_6(C_2, C_3, C_6),$$

where f_i, $i = 1, 2, 3$, represents the equality constraint on the systematic bits and the remaining factors correspond to parity check constraints. Equation (1) is expressed as a factor graph in Figure 1. Whereas the information bits are usually omitted from the code's factor graph, we have included these variable nodes because we must learn a model that enables us to compute inference on the hidden variables in a general system.

For numerical evaluations, we use Binary Phase Shift Keying (BPSK) modulation with real Additive White Gaussian Noise (AWGN) channel with for the received signal model:

$$Y_i = (-1)^{C_i} + W_i, \quad (2)$$

where $E[W_i^2] = N_0$ and the signal-to-noise ratio is $SNR = N_0^{-1}$. The *a posteriori* probability of the transmitted bits is given by:

$$f(V|Y) \propto f(V)\prod f(Y_i|C_i), \quad (3)$$

where $Y = (Y_1, \ldots, Y_6)$. The channel output likelihoods $\{f(Y_i|C_i)\}$ are often depicted as dongles hanging from the code bit variable nodes in Figure 1, reflecting the fact that they are a function only of the $\{C_i\}$ given the channel output $\{Y_i\}$.

There is a one-to-one correspondence between the edges of the factor graph in Figure 1 and the nonzero elements of the parity check matrix defined in Figure 2. Hence the binary linear code can be constructed as a factor graph (in the dual domain), chosen for its desirable characteristics for the decoder algorithm, such as density propagation and maximum size of the minimum loop (*i.e.* maximum girth). The belief propagation decoder is an instance of the sum-product algorithm [1]–[3]. Belief propagation comprises iterative message massing between nodes of the graph, illustrated in Figures 3 and 4. The algorithm assumes that an outgoing message on a graph edge is independent from the incoming message on the same edge. This assumption is violated after sufficiently many iterations for messages to traverse the length of the minimum loop. Nonetheless belief propagation is an important algorithm for inference processing on graphs and the result of so-called loopy belief propagation is approximate after the number of iterations exceeds one-half of the girth of the graph.

Given a factor graph model, we can estimate its component *p.d.f.*'s using empirical measurements of relevant system moments derived from training data. In practice, we may only use the low-order, low-degree moments to generate approximate estimates of factors of the system *p.d.f.* Given the graph model and estimates of its corresponding factors, the sum product algorithm can be directly applied to compute the conditional *p.d.f.*'s of hidden variables of the graph given the status of observed variables. Variable node processing and factor node processing steps of the sum-product algorithm are illustrated in Figure 3 and 4, for the case of binary parity check data.

III. Estimation of Factor Density Functions

In this work we use training data to learn a factor graph model of the data system and develop estimates of the factor *p.d.f.*s. We build the estimates of the factor density functions using estimates of the joint statistical moments between the system variables and Monte Carlo integration. Our methodology leverages the following theorem.

Theorem 1. *The joint* p.d.f. *of X and Y, denoted $f_{XY}(x, y)$, is uniquely determined by its joint statistical moments, $E[X^m Y^n]$, for all $m = 1, 2, 3...$ and $n = 1, 2, 3, ...$*

Proof. The theorem follows from substituting the Taylor expansion of the complex exponential in the multivariate Fourier transform of the joint *p.d.f.* $f_{XY}(x, y)$.

The *p.d.f.* of X and Y can be expressed as:

$$f_{XY}(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F_{XY}(\omega_1, \omega_2)e^{i2\pi(\omega_1 x+\omega_2 y)}d\omega_1 d\omega_2, \quad (4)$$

where $F_{XY}(\omega_1, \omega_2)$ denotes the two-dimensional Fourier transform of $f_{XY}(x, y)$:

$$F_{XY}(\omega_1,\omega_2) = \frac{1}{(2\pi)^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)e^{-i2\pi(\omega_1 x+\omega_2 y)}dxdy$$
$$= \frac{1}{(2\pi)^2}E\left[e^{-i2\pi(\omega_1 X+\omega_2 Y)}\right], \quad (5)$$
$$= \frac{1}{(2\pi)^2}\sum_{m=0}^{\infty}\sum_{n=0}^{\infty}(-i2\pi)^{m+n}\frac{\omega_1^m}{m!}\frac{\omega_2^n}{n!}E[X^m Y^n]. \quad (6)$$

Hence, assuming convergence of the integrals in Equations (4) and (6), the *p.d.f.* $f_{XY}(x,y)$ is expressed in terms of its joint moments $E[X^m Y^n]$ for all $m > 0$ and $n > 0$. □

Corollary 2. *The random variables X and Y are independent if and only if the following equation holds for all values of $m > 0$ and $n > 0$:*

$$E\left[X^m Y^n\right] = E\left[X^m\right] E\left[Y^n\right]. \tag{7}$$

Proof. For X and Y independent, we have $f_{XY}(x,y) = f_X(x) f_Y(y)$, which implies that $F_{XY}(\omega_1,\omega_2) = F_X(\omega_1) F_Y(\omega_2)$ and thus we can write the following:

$$\begin{aligned} E[X^m Y^n] &= \frac{(2\pi)^2}{(-i2\pi)^{m+n}} \frac{\partial^{m+n} F_{XY}(\omega_1,\omega_2)}{\partial^m \omega_1 \partial^n \omega_2}\bigg|_{\omega_1=0,\omega_2=0} \\ &= \frac{(2\pi)^2}{(-i2\pi)^{m+n}} \frac{d^m F_X(\omega_1)}{d^m \omega_1}\bigg|_{\omega_1=0} \frac{d^n F_Y(\omega_2)}{d^n \omega_2}\bigg|_{\omega_2=0} \\ &= E[X^m] E[Y^n], \end{aligned} \tag{8}$$

where we have assumed that the appropriate conditions for interchanging the order of differentiation and expectation hold true.

Conversely, substituting $E\left[X^m Y^n\right] = E\left[X^m\right] E\left[Y^n\right]$ in Equation (6) yields $F_{XY}(\omega_1,\omega_2) = F_X(\omega_1) F_Y(\omega_2)$ and thus $f_{XY}(x,y) = f_X(x) f_Y(y)$. □

The above results extend to larger multivariate *p.d.f.*s, wherein the joint statistical moments must be taken between all combinations of exponents and variables. A result of Corollary 2 is that if Equation (7) does not hold true for any m and n, then X and Y must be dependent random variables—a property that we will leverage when checking for dependence among subsets of system variables.

The central contribution of this work is the use of statistical moments estimated from training data to select dependent subsets of system variables (rows of the parity check matrix) and thus a factorization of the system *p.d.f.*, as in Equation (1). We then estimate the functional form of the component factors using (5) and (6). Assuming a stationary system (joint statistics are time-invariant), and *i.i.d.* training data (x_i, y_i), $i = 1, \ldots, T$, we have the following empirical estimates for the joint moments of X and Y:

$$E[X^m Y^n] = \lim_{T\to\infty} \frac{1}{T} \sum_{i=1}^{T} x_i^m y_i^n,\ 0 < m, n < \infty, \tag{9}$$

which follows from the law of large numbers. In practice, reduced complexity estimates are obtained by using only low order, low degree moments to select factors and to estimate Equation (6).

Since the component factors of the system posterior density depend only on subsets of system variables, the complexity of joint *p.d.f.* estimation is reduced. Moreover, inference processing using the sum-product algorithm benefits from small factor size in terms of complexity and performance.

IV. Reverse Engineering the Hamming Code

We use the proposed AGL framework to reverse engineer the $(6,3)$ Hamming code based on training data of T input-output codeword pairs. Our methodology comprises three phases: 1. factor selection, 2. factor density estimation, and 3. execution of the sum-product algorithm with learned model on noisy received codewords. These phases are described in detail below.

Phase 1: Factor Selection In Phase I, subsets of variables are tested for dependence by measuring empirical moments from training data and dependent subsets are selected to construct a factor graph for the system of variables. Graphically, this amounts to creating a set of factor nodes corresponding to the selected subsets of variables and adding edges from the factors to their corresponding variable nodes.

The main technique we use to select factors follows from Corollary 2. In particular, we identify dependent subsets of variables by checking if their joint moments, measured from training data, deviate from the product of their individual moments. Thus, if V_1 and V_2 represent dependent system variables, then there exists a pair of exponents (m,n) such that $0 < \Delta_{mn} \triangleq |E[V_1^m V_2^n] - E[V_1^m] E[V_2^n]|$. The larger the deviation Δ_{mn} from zero indicates a greater degree of confidence that the variables exhibit dependence. Thus the Δ_{mn} serve as a metrics for ranking dependent subsets. For binary data, we have $E[V_1^m V_2^n] = E[V_1 V_2]$ for all m and n greater than zero. Hence it suffices to measure only the first order moments.

Further, the training data is normalized to have zero mean and unit variance. Letting $v_i[1], \ldots, v_i[T]$ denote T realizations of the random variable V_i, we set: $\bar{v}_i[t] = \sigma_i^{-1}(v_i[t] - \mu_i)$, where $\mu_i = T^{-1} \sum v_i[t]$ and $\sigma_i^2 = T^{-1} \sum (v_i[t] - \mu_i)^2$. We then construct the metrics $\Delta_{ij} = |T^{-1} \sum \bar{v}_i[t] \bar{v}_j[t]|$ and look for those that exceed a threshold value. Similarly we add degree three factors whose joint empirical moments exceed another threshold value. Hence, we look for deviations of the joint moments from the product of their individual moments over the ensemble and determine thresholds $\{\tau_d\}$ for detecting variable dependence per degree of subset d.

For the reverse engineering problem, we chose the thresholds to yield full rank parity check matrices with high reliability over randomly selected training data and used only degree-2 and degree-3 factors. Future research involves generalizing threshold optimization, for example, by selecting factor degrees according to a prescribed degree distribution derived from mutual information transfer analysis of the training data [16].

We impose several other constraints on the factor selection candidate set. First, we enforce a linear independence condition such that a new factor will be added to the graph only if it is linearly independent from the set of existing factors. This condition prohibits over-determining the system factors arising from multiple ways of factoring coupled sets of dependent variables. Similarly limiting the rank of the graph to the dimension of the observed variables (total variables

less hidden variables) prevents over-determining the factors of the system. Additionally, we discard any new factor that overlaps with an existing factor by two or more variables. This condition avoids the creation of graph loops of size four which would degrade the performance of belief propagation (sum-product algorithm). Thus, the girth of the graph is at least six and we are guaranteed three iterations of belief propagation before the assumption of independence of factor messages is violated.

Due to the combinatorial complexity of factor detection for massive systems, it is necessary to use heuristic criteria to limit the search space for candidate variable subsets, such as those described above. The approach taken in this paper is to use low-order/low-degree statistical moments as the primary basis for selecting dependent subsets of variables. Factor selection is an open area for further research and additional avenues include sparse graphs, clustering methods, random sampling and physical models.

Phase 2: Factor Density Estimation Characterizing the functional form of the component factors of the global system *p.d.f.* is handled in Phase 2. We assume that the component factors represent the joint *p.d.f.* of their connected variables, *i.e.* posterior densities over the variables [3]. Thus, if $\{V_1, V_2\}$ and $\{V_2, V_3, V_4\}$ were identified as the dependent subsets of variables of a four variable system, then we assume that the global *p.d.f.* is given by $f(V_1, V_2, V_3, V_4) \propto f_1(V_1, V_2) f_2(V_2, V_3, V_4)$. To solve for the functional form of the factor *p.d.f.*, f_1 and f_2, we use Theorem 1, which enables us to construct estimates of the densities using empirical measurements of the joint statistical moments taken from training data as in Equation (6), or similarly, by directly computing the expected value of the complex exponential in Equation (5) via Monte Carlo integration. We mitigate the complexity of this phase by restricting attention to low-order moments or truncating the summations when building estimates of the component factors.

We specialize Equation (4) for the case of binary data:

$$f_{XY}(x, y) = \sum_{\omega_k, \omega_l \in \{0,1\}^2} F_{XY}(\omega_k, \omega_l)(-1)^{(\omega_k x + \omega_l y)} \quad (10)$$

where X and Y are binary random variables, and Equation (5) becomes:

$$F_{XY}(\omega_k, \omega_l) = \frac{1}{4} E\left[(-1)^{(\omega_k X + \omega_l Y)}\right]. \quad (11)$$

Three or more variables generalizes easily from the form above.

In the ECC reverse engineering example, we simply estimated Equation (11) from the training data with Monte Carlo integration and computed Equation (10) to provide estimates of the component factors of system posterior density. Hence, the same training data is used to separately select dependent variable subsets and then estimate their joint density functions. In future incarnations, it may be advantageous to jointly perform dependent subset selection and density estimation.

Phase 3: Sum-Product Algorithm Once the factor graph and its component densities are determined, it is straightforward to implement an instance of the sum-product algorithm

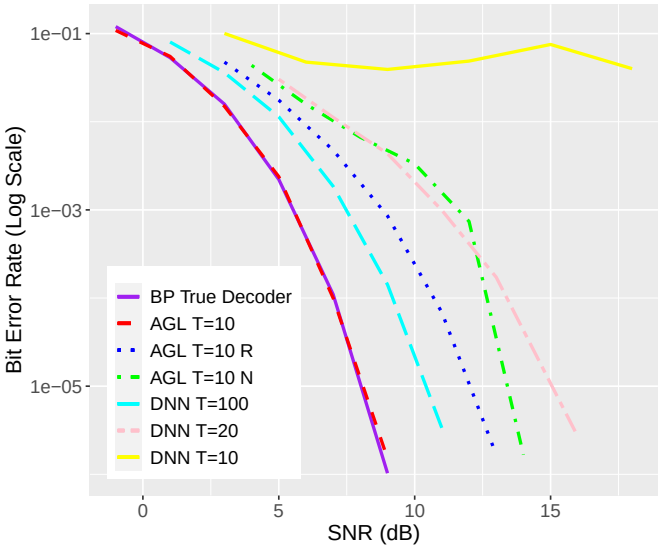

Fig. 5. Performance of Automatic Graph Learning (AGL) versus Dense Neural Network (DNN) with different number of training codeword pairs (T=10,20,100). AGL performs as well as belief propagation with known decoder, both with 8 iterations.

for computing inferences on hidden variables based on the observed variables. Since multiple factor graphs can be used to model the same data system, it is advantageous to choose graphs with desirable properties for the belief propagation algorithm. Sparse graphs with large girth are known to yield high performance ECCs [18]. Sparse models may not always be appropriate, however, it is of key interest to understand when sparse models can be used as an approximating class for the system posterior density without seriously degrading performance. In this work, we always select low-degree factors first before considering high-degree factors when constructing the factor graph model.

We use the *p.d.f.* computed with Equation (10) to implement the sum-product algorithm as follows. When computing the factor node update for variable V_l, the conditional *p.d.f.* $f(V_l|V_m, V_n)$ (derived from $f(V_l, V_m, V_n)$) is multiplied by the variable densities $a(V_m)$ and $a(V_n)$ received as messages from variable nodes V_m and V_n to obtain the joint density, which is then marginalized for the output variable V_l. This process is executed at each iteration of belief-propagation, according to the equations defined in Figures 3 and 4.

We implemented a belief propagation decoder using an AGL derived model as described above and compared the performance to the true decoder in Figure 5. The result shows that, with only 10 training codeword pairs, AGL has the potential to match performance of the true decoder. When the learned factor graph corresponds to a parity check matrix whose rows span the null space of the generator matrix, there is no difference in performance between the true decoder and the AGL model.

However, depending on the realization of the 10 training data samples, the factor selection algorithm may not always select a parity matrix that is rank-six or dual to the generator.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1507 (Dense)           (None, 128)               896
_________________________________________________________________
dense_1508 (Dense)           (None, 64)                8256
_________________________________________________________________
dense_1509 (Dense)           (None, 32)                2080
_________________________________________________________________
flatten_405 (Flatten)        (None, 32)                0
_________________________________________________________________
dense_1510 (Dense)           (None, 3)                 99
=================================================================
```

Fig. 6. Keras model summary of the layered fully-connected neural network.

The AGL curve labeled "T=10 R" depicts the performance of a rank-five parity matrix (whose rows lie in the null space of the generator matrix G) after three iterations of belief propagation decoding. The result shows a 3.8 dB loss compared to ideal curve at BER=10^{-5}. The AGL curve labeled "T=10 N" is rank-six but contains one factor that violates two of three orthogonality conditions (with G). Despite this impediment, the result demonstrates reasonable performance with a loss of roughly 5.5 dB compared to true decoder.

In 1000 trials of T=10 training codeword pairs, we found that 50.7% learned models were full-rank in the null space of G, 30.9% were rank-five in the null space of G, and 2.3% had one or two orthogonality constraint violations (corresponding to the three cases depicted in Figure 5). Our numerical results further suggest that variations on the sum-product algorithm may compensate for model mismatch. In particular the result labeled "T=10 N" was developed by a sum-product decoder that used the joint factor distribution when implementing Figure 4, instead of the conditional form as depicted.

Finally, we designed and trained a multi-layer Dense Neural Network (DNN) with the TensorFlow/Keras toolkit and used it to decode noisy channel outputs with the same AWGN channel model used above. The parameters of the DNN are listed in Figure 6, where the Adam optimizer was used with a mean squared error loss function. We found that the performance of the neural network was moderately improved by adding a small amount of AWGN to the modulated training data. The DNN is unable to match the performance of the AGL (T=10) when the parity matrix is dual to the generator, even with 100 training samples. AGL model mismatch cases yield comparable performance to DNN with T=100 and T=20 training samples. Hence, we conclude that AGL has the potential to outperform competitive neural networks with an order of magnitude less training data.

V. Conclusion

In this paper, we developed the AGL framework for applications in machine learning and artificial intelligence systems. We have shown that a suitable factor graph model can be learned by detecting dependent subsets of system variables, and component factor *p.d.f.*s can be estimated using Fourier domain representations. Numerical results are provided for the ECC reverse engineering problem, which show that AGL with limited training data has the potential to provide a competitive alternative to neural networks.

A challenging aspect of the approach is constructing the factor graph, due to the combinatorial complexity of selecting variable subsets and the difficulty of constructing an accurate model with limited training data. However, we have discussed several heuristics to reduce the candidate search space and, further, application specific prescriptions could be leveraged. Once the graph is determined, the results show that a relatively small amount of training is required to characterize the factor *p.d.f.*s, requiring only 10 training codewords to replicate the performance of the true belief propagation decoder. Finally, our results show that inference processing using belief propagation is still viable even when there is a small degree of model mismatch.

References

[1] J. Pearl, *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann Publishers, 1988.

[2] F. Kschischang, B. Frey, and H. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[3] D. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[4] N. Jacobsen and R. Soni, "Design of rate-compatible irregular LDPC codes based on edge growth and parity splitting," in *Proc. IEEE Veh. Tech. Conf. (VTC)*, Baltimore, MD, USA, Sept. 2007.

[5] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[6] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–8, 1982.

[7] L. Kotthoff, C. Thornton, H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA," vol. 17, pp. 1–5, 2016.

[8] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer learning with neural AutoML," in *32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[9] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *34th International Conference on Machine Learning*, Sydney, Australia, 2017.

[10] A. Steimer, W. Maass, and R. Douglas, "Belief propagation in networks of spiking neurons," *Neural Computation*, 2009.

[11] M. Wainwright and M. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1, pp. 1–305, 2008.

[12] A. Wrigley, W. S. Lee, and N. Ye, "Tensor belief propagation," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, Aug 2017, pp. 3771–3779.

[13] Y. Xue, S. Ermon, R. L. Bras, C. P. Gomes, and B. Selman, "Variable elimination in the fourier domain," in *Proceedings of the 33rd International Conference on Machine Learning*, New York, NY, 2016.

[14] R. M. Neal, *Bayesian learning for neural networks*, ser. Lecture Notes in Statistics. Springer-Verlag, 1996, no. 118.

[15] D. MacKay and R. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, pp. 1645–1646, Aug. 1996.

[16] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[17] N. Jacobsen, "Practical cooperative coding for half-duplex relay channels," in *Proc. Conf. on Inform. Sciences and Systems (CISS)*, Baltimore, MD, USA, Mar. 2009.

[18] R. Gallager, "Low density parity check codes," *IRE Trans. Information Theory*, vol. 8, pp. 21–28, Jan. 1962.